Libraries Guide

Xilinx Unified Libraries

Selection Guide

Design Elements (ACC1 to BYPOSC)

Design Elements (CAPTURE_SPARTAN2 to DECODE64)

Design Elements (F5MAP to FTSRLE)

Design Elements (GCLK to KEEPER)

Design Elements (LD to NOR16)

Design Elements (OAND2 to OXOR2)

Design Elements (PULLDOWN to ROM32X1)

Design Elements (SOP3 to XORCY_L)

Design Elements (X74_42 to X74_521)

Attributes, Constraints, and Carry Logic

About This Manual

This manual describes Xilinx's Unified Libraries and the attributes and constraints that can be used with the components.

Before using this manual, you should be familiar with the operations that are common to all Xilinx software tools: how to bring up the system, select a tool for use, specify operations, and manage design data. These topics are covered in the *Quick Start Guide*.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this page. You can also directly access some of these resources using the provided URLs.

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from The Programmable Logic Data Book, which describe device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Manual Contents

This manual contains the following chapters.

- Chapter 1, "Xilinx Unified Libraries"
- Chapter 2, "Selection Guide"
- Chapter 3, "Design Elements (ACC1 to BYPOSC)"
- Chapter 4, "Design Elements (CAPTURE_SPARTAN2 to DECODE64)"

- Chapter 5, “Design Elements (F5MAP to FTSRLE)”
- Chapter 6, “Design Elements (GCLK to KEEPER)”
- Chapter 7, “Design Elements (LD to NOR16)”
- Chapter 8, “Design Elements (OAND2 to OXOR2)”
- Chapter 9, “Design Elements (PULLDOWN to ROM32X1)”
- Chapter 10, “Design Elements (SOP3 to XORCY_L)”
- Chapter 11, “Design Elements (X74_42 to X74_521)”
- Chapter 12, “Attributes, Constraints, and Carry Logic”

Chapter 1, ”Xilinx Unified Libraries,” discusses the unified libraries, applicable device architectures for each library, contents of the other chapters, general naming conventions, and performance issues.

Chapter 2, “Selection Guide,” describes then lists design elements by function that are explained in detail in the “Design Elements” chapters.

Chapters 3 through 11, “Design Elements,” provide a graphic symbol, functional description, primitive versus macro table, truth table (when applicable), topology (when applicable), and schematics for macros of the design elements.

Chapter 12, “Attributes, Constraints, and Carry Logic,” provides information on all attributes, logical constraints, placement and timing constraints, relationally placed macros (RPMs), and carry logic.

Contents

Chapter 3 Design Elements (ACC1 to BYPOSC)

Chapter 5 Design Elements (F5MAP to FTSRLE)

Chapter 6 Design Elements (GCLK to KEEPER)

Chapter 7 Design Elements (LD to NOR16)

Chapter 8 Design Elements (OAND2 to OXOR2)

Chapter 9 Design Elements (PULLDOWN to ROM32X1)

Chapter 10 Design Elements (SOP3 to XORCY_L)

Chapter 11 Design Elements (X74_42 to X74_521)

Chapter 12 Attributes, Constraints, and Carry Logic

Conventions

This manual uses the following typographical and online document conventions. An example illustrates each typographical convention.

Typographical

The following conventions are used for all documents.

- `Courier font` indicates messages, prompts, and program files that the system displays.

  ```
  speed grade: -100
  ```

- **`Courier bold`** indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier bold are not literal and square brackets "[ ]" in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

  `rpt_del_net=`

  `Courier bold` also indicates commands that you select from a menu.

  `File` → **`Open`**

- *Italic font* denotes the following items.
  - Variables in a syntax statement for which you must supply values

    `edif2ngd` *design_name*

  - References to other manuals

    See the *Development System Reference Guide* for more information.

  - Emphasis in text

    If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets "[ ]" indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  `edif2ngd` [*option_name*] *design_name*

- Braces "{ }" enclose a list of items from which you must choose one or more.

  `lowpwr =`{**`on`** | **`off`**}

- A vertical bar " | " separates items in a list of choices.

  `lowpwr =`{**`on`** | **`off`**}

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT’
IOB #2: Name = CLKIN’
.
.
.
```

- A horizontal ellipsis “. . .” indicates that an item can be repeated one or more times.

  `allow block` *block_name loc1 loc2* `...` *locn;*

Online Document

The following conventions are used for online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click the red-underlined text to open the specified cross-reference.
- Blue-underlined text indicates an intrabook link, which is a cross-reference within a book. Click the blue-underlined text to open the specified cross-reference.

Note: The preceding conventions do not apply to the links in the Adobe Acrobat versions of the books.

Chapter 1

Xilinx Unified Libraries

This chapter describes the Unified Libraries and the applicable device architectures for each library. It also briefly discusses the contents of the other chapters, the general naming conventions, and performance issues.

This chapter consists of the following major sections.

- "Overview"
- "Applicable Architectures"
- "Selection Guide"
- "Design Elements"
- "Schematic Examples"
- "Naming Conventions"
- "Attributes, Constraints, and Carry Logic"
- "Flip-Flop, Counter, and Register Performance"
- "Unconnected Pins"

Overview

Xilinx maintains software libraries with thousands of functional design elements (primitives and macros) for different device architectures. New functional elements are assembled with each release of development system software. The catalog of design elements is known as the "Unified Libraries." Elements in these libraries are common to all Xilinx device architectures. This "unified" approach means that you can use your circuit design created with "unified" library elements across all current Xilinx device architectures that recognize the element you are using.

Elements that exist in multiple architectures look and function the same, but their implementations might differ to make them more efficient for a particular architecture. A separate library still exists for each architecture (or architectural group) and common symbols are duplicated in each one, which is necessary for simulation (especially board level) where timing depends on a particular architecture.

If you have active designs that were created with former Xilinx library primitives or macros, you may need to change references to the design elements that you were using to reflect the Unified Libraries' elements.

The *Libraries Guide* describes the primitive and macro logic elements available in the Unified Libraries for the Xilinx FPGA and CPLD devices. Common logic functions can be implemented with these elements and more complex functions can be built by combining macros and primitives. Several hundred design elements (primitives and macros) are available across multiple device architectures, providing a common base for programmable logic designs.

This libraries guide provides a functional selection guide, describes the design elements, and addresses attributes, constraints, and carry logic.

Applicable Architectures

Design elements for the XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, and Virtex libraries are included in the Xilinx Unified Libraries. Each library supports specific device architectures. For detailed information on the architectural families referenced below and the devices in each, refer to the current version of *The Programmable Logic Data Book* (an online version is available from the Xilinx web site, http://support.xilinx.com).

XC3000 Library

Information appearing under the title of XC3000 pertains to the following devices families:

- XC3000A
- XC3100A
- XC3000L
- XC3100L

The XC3000L and XC3100L are identical in architecture and features to the XC3000A and XC3100A, respectively, but operate at a nominal supply voltage of 3.3 V.

XC4000E Library

Information appearing under the title XC4000E pertains to the following device families:

- XC4000E
- XC4000L

The XC4000L is identical in architecture and features to the XC4000E but operates at a nominal supply voltage of 3.3 V.

XC4000X Library

Information appearing under the title XC4000X pertains to the following device families:

- XC4000EX
- XC4000XL
- XC4000XV
- XC4000XLA

The XC4000XL is identical in architecture and features to the XC4000EX but operates at a nominal supply voltage of 3.3 V. The XC4000XV has identical library symbols to the XC4000EX and XC4000XL but operates at a nominal supply voltage of 2.5 V and includes additional features.

XC4000

Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. This consists of the following device families:

- XC4000E
- XC4000L
- XC4000EX
- XC4000XL
- XC4000XV
- XC4000XLA

XC5200 Library

The information appearing under the title XC5200 pertains to the XC5200 family.

XC9000 Library

The information appearing under the title XC9000 pertains to the following CPLD device families:

- XC9500
- XC9500XL
- XC9500XV

Spartan Library

The information appearing under the title Spartan pertains to the Spartan family XCS* devices.

SpartanXL Library

The information appearing under the title SpartanXL pertains to the SpartanXL family XCS*XL devices.

Spartan2 Library

The information appearing under the title Spartan2 pertains to the Spartan2 family XC2S* devices.

Spartans

Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

Virtex Library

The information appearing under the title Virtex pertains to the Virtex family XCV* devices.

Selection Guide

The “Selection Guide” chapter briefly describes, then tabularly lists the logic elements that are explained in detail in the “Design Elements” sections. The tables included in this section are organized into functional categories. They list the available elements in each category along with a brief description of each element and an applicability table identifying which libraries (XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex) contain the element.

Design Elements

Design elements are organized in alphanumeric order, with all numeric suffixes in ascending order. For example, FDR precedes FDRS, and ADD4 precedes ADD8, which precedes ADD16.

The following information is provided for each library element.

- Graphic symbol
- Applicability table (with primitive versus macro identification)
- Functional description
- Truth table (when applicable)
- Topology (when applicable)
- Schematic for macros

Schematic Examples

Schematics are included for each library if the implementation differs.

Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

Naming Conventions

Examples of the general naming conventions for the unified library elements are shown in the following figures.

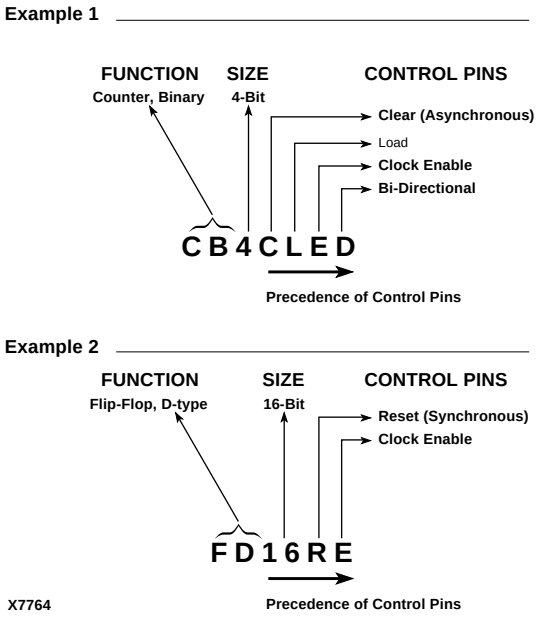


Figure 1-1 Naming Conventions

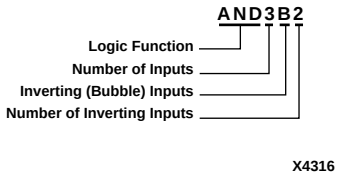


Figure 1-2 Combinatorial Naming Conventions

Refer to the “Selection Guide” chapter for examples of functional component naming conventions.

Attributes, Constraints, and Carry Logic

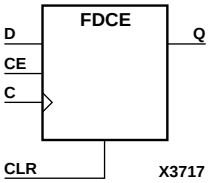
Attributes are instructions placed on symbols or nets in a schematic to indicate their placement, implementation, naming, directionality, and so forth. Constraints are a type of attribute used only to limit where an element should be placed. The “Attributes, Constraints, and Carry Logic” chapter provides information on all attributes and constraints.

Flip-Flop, Counter, and Register Performance

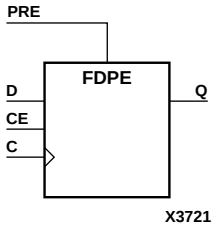
All counter, register, and storage functions are derived from the flip-flops (and latches in XC4000X and SpartanXL) available in the Configurable Logic Blocks (CLBs).

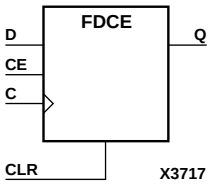
The D flip-flop is the basic building block for all architectures. Differences occur from the availability of asynchronous Clear (CLR) and Preset (PRE) inputs, and the source of the synchronous control signals, such as, Clock Enable (CE), Clock (C), Load enable (L), synchronous Reset (R), and synchronous Set (S). The basic flip-flop configuration for each architecture follows.

The XC3000 and XC5200 have a direct-connect Clock Enable input and a Clear input.

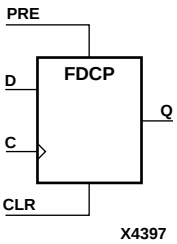


The XC4000s, XC9500XL, Spartan, and SpartanXL have a direct-connect Clock Enable input and a choice of either the Clear or the Preset inputs, but not both.

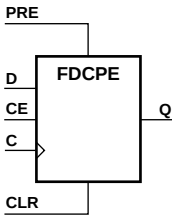


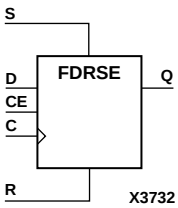


The basic XC9000 flip-flops have both Clear and Preset inputs.



Virtex and Spartan2 have two basic flip-flop types. One has both Clear and Preset inputs and one has both asynchronous and synchronous control functions.





The asynchronous and synchronous control functions, when used, have a priority that is consistent across all devices and architectures. These inputs can be either active-High or active-Low as defined by the macro. The priority, from highest to lowest is as follows.

- Asynchronous Clear (CLR)
- Asynchronous Preset (PRE)
- Synchronous Set (S)
- Synchronous Reset (R)
- Clock Enable (CE)

Note: The asynchronous CLR and PRE inputs, by definition, have priority over all the synchronous control and clock inputs.

For FPGA families, the Clock Enable (CE) function is implemented using two different methods in the Xilinx Unified Libraries; both are shown in the “Clock Enable Implementation Methods” figure.

- In method 1, CE is implemented by connecting the CE pin of the macro directly to the dedicated Enable Clock (EC) pin of the internal Configurable Logic Block (CLB) flip-flop. This allows one CE per CLB. CE takes precedence over the L, S, and R inputs. All flip-flops with asynchronous clear or preset use this method.
- In method 2, CE is implemented using function generator logic. This allows two CEs per CLB. CE has the same priority as the L, S, and R inputs. All flip-flops with synchronous set or reset use this method.

The method used in a particular macro is indicated by the inclusion of asynchronous clear, asynchronous preset, synchronous set, or synchronous reset in the macro’s description.

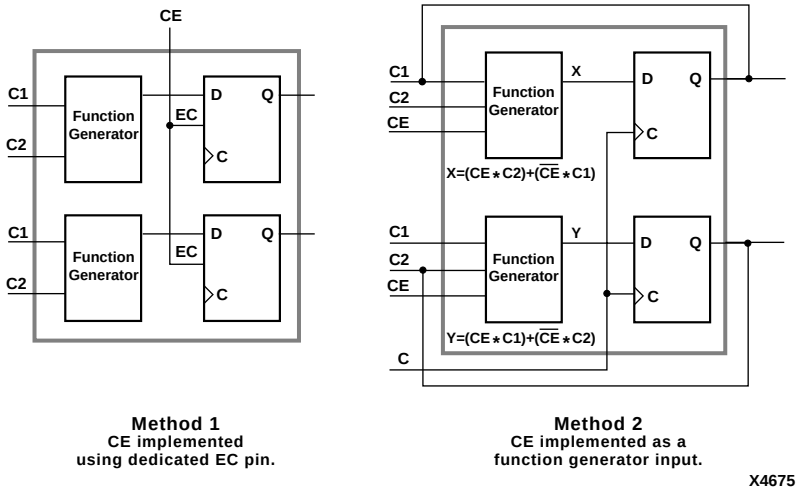


Figure 1-3 Clock Enable Implementation Methods

Unconnected Pins

Xilinx recommends that you *always* connect input pins in your schematics. This ensures that front end simulation functionally matches back end timing simulation. If an input pin is left unconnected, mapper errors may result.

If an output pin is left unconnected in your schematic, the corresponding function is trimmed. If the component has only one output, the entire component is trimmed. If the component has multiple outputs, the portion that drives the output is trimmed. As an example of the latter case, if the overflow pin (OFL) in an adder macro is unconnected, the logic that generates that term is trimmed, but the rest of the adder is retained (assuming all of the sum outputs are connected).

Chapter 2

Selection Guide

This chapter provides a CLB count for the design elements in each library plus a list of the Relationally Placed Modules (RPMs) by family. It also categorizes, by function, the logic elements that are described in detail in the "Design Elements" sections.

The chapter contains three major sections.

- "CLB Count"
- "Relationally Placed Macros"
- "Functional Categories"

CLB Count

Configurable Logic Blocks (CLBs) implement most of the logic in an FPGA. The following CLB Count table lists FPGA design elements in alphanumeric order with the number of CLBs needed for their implementation in each applicable library. Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific device architectures supported in each library.

Each XC5200 CLB contains four independent Logic Cells™ (LCs). In the following table, the numbers in the XC5200 column are the LC4 count.

Each Virtex and Spartan2 CLB contains two slices. In the following table, the numbers for Spartan2 and Virtex are the combined count for the two slices.

Note: This information is for reference only. The actual count could vary, depending upon the switch settings of the implementation tools; for example, the effort level in PAR (Place and Route).

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
ACC4	9	7	7	15	7	7	5	5
ACC8	17	11	11	27	11	11	9	9
ACC16	33	19	19	51	19	19	17	17
ACLK	1	-	-	-	-	-	-	-
ADD4	5	4	4	10	4	4	3	3
ADD8	9	6	6	18	6	6	5	5
ADD16	17	10	10	34	10	10	9	9
ADSU4	5	4	4	10	4	4	3	3
ADSU8	9	6	6	18	6	6	5	5
ADSU16	17	10	10	34	10	10	9	9

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
AND2	1	-	-	1	-	-	1	1
AND3	1	-	-	1	-	-	1	1
AND4	1	-	-	1	-	-	1	1
AND5	1	1	1	2	1	1	1	1
AND6	2	1	1	2	1	1	1	1
AND7	2	1	1	3	1	1	1	1
AND8	2	1	1	3	1	1	2	2
AND9	2	1	1	4	1	1	2	2
AND12	-	-	-	4	-	-	2	2
AND16	-	-	-	5	-	-	2	2
BRLSHFT4	4	4	4	4	4	4	8	8
BRLSHFT8	12	12	12	12	12	12	12	12
BSCAN	-	-	-	3	-	-	-	-
BUFE	1	-	-	-	-	-	-	-
BUFE4	1	-	-	-	-	-	-	-
BUFE8	1	-	-	-	-	-	-	-
BUFE16	1	-	-	-	-	-	-	-
BUFG	1	-	-	1	-	-	-	-
BUFGP	-	-	-	1	-	-	-	-
BUFGS	-	-	-	1	-	-	-	-
CB2CE	3	2	2	4	2	2	2	2
CB2CLE	4	3	3	5	3	3	3	3
CB2CLED	4	3	3	6	3	3	3	3
CB2RE	3	2	2	4	2	2	2	2
CB4CE	4	3	3	6	3	3	3	3
CB4CLE	7	5	5	9	5	5	5	5
CB4CLED	8	7	7	10	7	7	6	6
CB4RE	4	4	4	8	4	4	3	3
CB8CE	8	6	6	13	6	6	6	6
CB8CLE	13	10	10	18	10	10	9	9
CB8CLED	14	12	13	22	12	13	12	12
CB8RE	9	8	8	17	8	8	6	6
CB16CE	16	12	12	27	12	12	13	13
CB16CLE	26	18	18	36	18	18	18	18
CB16CLED	28	25	25	46	25	25	24	24
CB16RE	18	18	18	35	18	18	13	13
CC8CE	-	5	5	18	5	5	8	8
CC8CLE	-	6	6	19	6	6	9	9
CC8CLED	-	11	11	19	11	11	9	9

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
CC8RE	-	5	5	18	5	5	9	9
CC16CE	-	9	9	34	9	9	16	16
CC16CLE	-	10	10	35	10	10	17	17
CC16CLED	-	19	19	35	19	19	17	17
CC16RE	-	9	9	34	9	9	17	17
CD4CE	4	3	3	6	3	3	3	3
CD4CLE	7	5	5	10	5	5	5	5
CD4RE	5	6	5	9	6	5	3	3
CD4RLE	10	9	9	17	9	9	7	7
CJ4CE	2	2	2	4	2	2	2	2
CJ4RE	2	4	4	4	4	4	2	2
CJ5CE	3	3	3	5	3	3	3	3
CJ5RE	3	5	5	5	5	5	3	3
CJ8CE	4	4	4	8	4	4	4	4
CJ8RE	4	8	8	8	8	8	4	4
COMP2	1	1	1	1	1	1	1	1
COMP4	4	1	1	3	1	1	2	2
COMP8	9	4	4	5	4	4	3	3
COMP16	17	9	9	11	9	9	6	6
COMPM2	3	1	1	5	1	1	1	1
COMPM4	8	2	2	13	2	2	5	5
COMPM8	19	8	8	27	8	8	11	11
COMPM16	39	21	21	64	21	21	24	24
COMPMC8	-	7	7	18	7	7	8	8
COMPMC16	-	11	11	34	11	11	16	16
CR8CE	8	8	8	8	8	8	8	8
CR16CE	16	16	16	16	16	16	16	16
CY_INIT	-	-	-	1	-	-	-	-
CY_MUX	-	-	-	2	-	-	-	-
D2_4E	2	2	2	4	2	2	2	2
D3_8E	4	4	4	8	4	4	4	4
D4_16E	16	16	16	32	16	16	16	16
DEC_CC4	-	-	-	2	-	-	1	1
DEC_CC8	-	-	-	3	-	-	1	1
DEC_CC16	-	-	-	5	-	-	2	2
DECODE4	-	-	-	2	-	-	1	1
DECODE8	-	-	-	3	-	-	2	2
DECODE16	-	-	-	5	-	-	2	2
DECODE32	-	-	-	9	-	-	4	4

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
DECODE64	-	-	-	18	-	-	8	8
F5_MUX	-	-	-	1	-	-	-	-
F5MAP	-	-	-	1	-	-	-	-
FD	1	-	-	1	-	-	-	-
FD_1	1	-	-	1	-	-	-	-
FD4CE	4	2	2	4	2	2	2	2
FD4RE	2	4	4	4	4	4	2	2
FD8CE	4	4	4	8	4	4	4	4
FD8RE	4	8	8	8	8	8	4	4
FD16CE	8	8	8	16	8	8	8	8
FD16RE	8	16	16	16	16	16	8	8
FDC	1	1	1	1	1	1	-	-
FDC_1	1	1	1	1	1	1	-	-
FDCE	1	1	1	1	1	1	-	-
FDCE_1	1	1	1	1	1	1	-	-
FDP	-	1	1	1	1	1	-	-
FDP_1	-	1	1	1	1	1	-	-
FDPE	-	-	-	1	-	-	-	-
FDPE_1	-	1	1	1	1	1	-	-
FDR	1	1	1	1	1	1	-	-
FDRE	1	1	1	1	1	1	-	-
FDRS	1	1	1	1	1	1	-	-
FDRSE	1	2	2	3	2	2	-	-
FDS	1	1	1	1	1	1	-	-
FDSE	1	1	1	1	1	1	-	-
FDSR	1	1	1	1	1	1	-	-
FDSRE	1	2	2	3	2	2	-	-
FJKC	1	1	1	1	1	1	1	1
FJKCE	1	1	1	1	1	1	1	1
FJKP	-	1	1	1	1	1	1	1
FJKPE	-	1	1	1	1	1	1	1
FJKRSE	2	2	2	3	2	2	1	1
FJKSRE	2	2	2	3	2	2	1	1
FTC	1	1	1	1	1	1	1	1
FTCE	1	1	1	1	1	1	1	1
FTCLE	1	1	1	2	1	1	1	1
FTCLEX	-	-	-	-	-	-	1	1
FTP	-	1	1	1	1	1	1	1
FTPE	-	1	1	1	1	1	1	1

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
FTPLE	-	1	1	2	1	1	1	1
FTRSE	1	2	2	3	2	2	1	1
FTRSLE	3	2	2	4	2	2	2	2
FTSRE	1	2	2	3	2	2	1	1
FTSRLE	3	2	2	4	2	2	2	2
GCLK	1	-	-	-	-	-	-	-
IFD	-	-	-	1	-	-	-	-
IFD_1	-	-	-	1	-	-	-	-
IFD4	-	-	-	4	-	-	-	-
IFD8	-	-	-	8	-	-	-	-
IFD16	-	-	-	16	-	-	-	-
ILD	-	-	-	1	-	-	1	1
ILD_1	-	-	-	1	-	-	1	1
ILD4	-	-	-	4	-	-	2	2
ILD8	-	-	-	8	-	-	4	4
ILD16	-	-	-	16	-	-	8	8
IOPAD	-	-	-	1	-	-	-	-
LD	-	-	1	1	-	1	-	-
LD4	-	-	4	-	-	4	2	2
LD8	-	-	8	-	-	8	4	4
LD16	-	-	16	-	-	16	8	8
LD4CE	-	-	4	4	-	4	2	2
LD8CE	-	-	8	8	-	8	4	4
LD16CE	-	-	16	16	-	16	8	8
LD_1	-	-	1	1	-	1	-	-
LDC	-	-	1	1	-	1	-	-
LDC_1	-	-	1	1	-	1	-	-
LDCE	-	-	1	1	-	1	-	-
LDCE_1	-	-	-	1	-	-	-	-
LDPE	-	-	1	-	-	1	-	-
LDPE_1	-	-	1	-	-	1	-	-
M2_1	1	1	1	1	1	1	1	1
M2_1B1	1	1	1	1	1	1	1	1
M2_1B2	1	1	1	1	1	1	1	1
M2_1E	1	1	1	1	1	1	1	1
M4_1E	3	1	1	1	1	1	1	1
M8_1E	6	3	3	7	3	3	2	2
M16_1E	11	7	7	14	7	7	5	5
NAND2	1	-	-	1	-	-	1	1

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
NAND3	1	-	-	1	-	-	1	1
NAND4	1	-	-	1	-	-	1	1
NAND5	1	1	1	2	1	1	1	1
NAND6	2	1	1	2	1	1	1	1
NAND7	2	1	1	3	1	1	1	1
NAND8	2	1	1	3	1	1	2	2
NAND9	2	1	1	4	1	1	2	2
NAND12	-	-	-	4	-	-	2	2
NAND16	-	-	-	5	-	-	2	2
NOR2	1	-	-	1	-	-	1	1
NOR3	1	-	-	1	-	-	1	1
NOR4	1	-	-	1	-	-	1	1
NOR5	1	1	1	2	1	1	1	1
NOR6	2	1	1	2	1	1	1	1
NOR7	2	1	1	3	1	1	1	1
NOR8	2	1	1	3	1	1	2	2
NOR9	2	1	1	4	1	1	2	2
NOR12	-	-	-	4	-	-	2	2
NOR16	-	-	-	5	-	-	2	2
OFD	-	-	-	1	-	-	-	-
OFD_1	-	-	-	1	-	-	-	-
OFD4	-	-	-	4	-	-	-	-
OFD8	-	-	-	8	-	-	-	-
OFD16	-	-	-	16	-	-	-	-
OFDE	-	-	-	1	-	-	-	-
OFDE_1	-	-	-	1	-	-	-	-
OFDE4	-	-	-	4	-	-	-	-
OFDE8	-	-	-	8	-	-	-	-
OFDE16	-	-	-	16	-	-	-	-
OFDT	-	-	-	1	-	-	-	-
OFDT_1	-	-	-	1	-	-	-	-
OFDT4	-	-	-	4	-	-	-	-
OFDT8	-	-	-	8	-	-	-	-
OFDT16	-	-	-	16	-	-	-	-
OR2	1	-	-	1	-	-	1	1
OR3	1	-	-	1	-	-	1	1
OR4	1	-	-	1	-	-	1	1
OR5	1	1	1	2	1	1	1	1
OR6	2	1	1	2	1	1	1	1

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
OR7	2	1	1	3	1	1	1	1
OR8	2	1	1	3	1	1	2	2
OR9	2	1	1	3	1	1	2	2
OR12	-	-	-	4	-	-	2	2
OR16	-	-	-	5	-	-	2	2
RAM16X2	-	1	1	-	1	1	-	-
RAM16X2D	-	2	2	-	2	2	2	2
RAM16X2S	-	1	1	-	1	1	2	2
RAM16X4	-	2	2	-	2	2	-	-
RAM16X4D	-	4	4	-	4	4	4	4
RAM16X4S	-	2	2	-	2	2	4	4
RAM16X8	-	4	4	-	4	4	-	-
RAM16X8D	-	8	8	-	8	8	8	8
RAM16X8S	-	4	4	-	4	4	8	8
RAM32X2	-	2	2	-	2	2	-	-
RAM32X2S	-	2	-	-	2	-	2	2
RAM32X4	-	4	4	-	4	4	4	4
RAM32X4S	-	4	4	-	4	4	8	8
RAM32X8	-	8	8	-	8	8	-	-
RAM32X8S	-	8	8	-	8	8	-	-
SOP3	1	1	1	1	1	1	1	1
SOP4	1	1	1	1	1	1	1	1
SR4CE	2	2	2	4	2	2	2	2
SR4CLE	4	3	3	5	3	3	3	3
SR4CLED	5	5	5	10	5	5	5	5
SR4RE	2	4	4	4	4	4	2	2
SR4RLE	6	5	5	9	5	5	3	3
SR4RLED	7	8	8	14	8	8	5	5
SR8CE	4	4	4	8	4	4	4	4
SR8CLE	5	5	5	9	5	5	5	5
SR8CLED	9	9	9	18	9	9	9	9
SR8RE	4	8	8	8	8	8	4	4
SR8RLE	12	9	9	17	9	9	5	5
SR8RLED	13	9	9	26	9	9	9	9
SR16CE	8	8	8	16	8	8	8	8
SR16CLE	9	9	9	17	9	9	9	9
SR16CLED	17	17	17	34	17	17	17	17
SR16RE	8	16	16	16	16	16	8	8
SR16RLE	24	20	20	33	20	20	9	9

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
SR16RLED	25	19	19	50	19	19	17	17
UPAD	-	-	-	1	-	-	-	-
XNOR2	1	-	-	1	-	-	1	1
XNOR3	1	-	-	1	-	-	1	1
XNOR4	1	-	-	1	-	-	1	1
XNOR5	1	1	1	2	1	1	1	1
XNOR6	2	1	1	2	1	1	1	1
XNOR7	2	1	1	3	1	1	1	1
XNOR8	2	1	1	3	1	1	2	2
XNOR9	2	1	1	3	1	1	2	2
XOR2	1	-	-	1	-	-	1	1
XOR3	1	-	-	1	-	-	1	1
XOR4	1	-	-	1	-	-	1	1
XOR5	1	1	1	2	1	1	1	1
XOR6	2	1	1	2	1	1	1	1
XOR7	2	1	1	3	1	1	1	1
XOR8	2	1	1	3	1	1	2	2
XOR9	2	1	1	3	1	1	2	2
X74_42	5	5	5	10	5	5	-	-
X74_L85	14	9	9	20	9	9	-	-
X74_138	5	5	5	9	5	5	-	-
X74_139	2	2	2	4	2	2	-	-
X74_147	8	6	6	12	6	6	-	-
X74_148	10	6	6	14	6	6	-	-
X74_150	11	6	6	13	6	6	-	-
X74_151	6	3	3	7	3	3	-	-
X74_152	5	3	3	6	3	3	-	-
X74_153	6	3	3	6	3	3	-	-
X74_154	17	16	16	33	16	16	-	-
X74_157	4	2	2	4	2	2	-	-
X74_158	4	2	2	4	2	2	-	-
X74_160	8	6	6	11	6	6	-	-
X74_161	9	5	5	9	5	5	-	-
X74_162	8	6	6	13	6	6	-	-
X74_163	10	9	9	17	9	9	-	-
X74_164	5	4	4	8	4	4	-	-
X74_165S	8	5	5	9	5	5	-	-
X74_168	9	7	7	11	7	7	-	-
X74_174	7	4	4	6	4	4	-	-

Name	XC3000	XC4000E	XC4000X	XC5200*	Spartan	SpartanXL	Spartan2**	Virtex**
X74_194	7	5	5	12	5	5	-	-
X74_195	5	3	3	5	3	3	-	-
X74_273	9	5	5	8	5	5	-	-
X74_280	3	2	2	5	2	2	-	-
X74_283	4	6	6	8	6	6	-	-
X74_298	4	2	2	4	2	2	-	-
X74_352	6	3	3	6	3	3	-	-
X74_377	9	4	4	8	4	4	-	-
X74_390	3	3	3	4	3	3	-	-
X74_518	9	4	4	6	4	4	-	-
X74_521	9	4	4	6	4	4	-	-

*LC4 count
**Combined count for the two slices
- = zero (0) or the component is not applicable for that architecture

Relationally Placed Macros

This section lists the Relationally Placed Macros (RPMs). RPMs are "soft" macros that contain relative location constraint (RLOC) information. For more details, see the "Relationally Placed Macros (RPMs)" section of the "Attributes, Constraints, and Carry Logic" chapter.

The following table lists RPMs (except for CY4_* carry mode symbols) by library for easy identification. A check mark (√) in the column under the library name means the element is an RPM in that library. Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific device architectures supported in each library.

Note: The CY4_* RPMs are not listed here. To see a list of predefined carry mode names and their corresponding symbols (CY4_*), refer to the "Carry Logic Primitives and Symbols" section of the "Attributes, Constraints, and Carry Logic" chapter.

Element Name	XC4000E	XC4000X	XC5200	Spartan	SpartanXL	Spartan2	Virtex
ACC4	√	√	√	√	√	√	√
ACC8	√	√	√	√	√	√	√
ACC16	√	√	√	√	√	√	√
ADD4	√	√	√	√	√	√	√
ADD8	√	√	√	√	√	√	√
ADD16	√	√	√	√	√	√	√
ADSU4	√	√	√	√	√	√	√
ADSU8	√	√	√	√	√	√	√
ADSU16	√	√	√	√	√	√	√
AND6						√	√
AND7						√	√
AND8	√	√	√	√	√	√	√
AND9	√	√	√	√	√	√	√
AND12			√			√	√
AND16			√			√	√
CC8CE	√	√	√	√	√	√	√
CC8CLE	√	√	√	√	√	√	√
CC8CLED	√	√	√	√	√	√	√
CC8RE	√	√	√	√	√	√	√
CC16CE	√	√	√	√	√	√	√
CC16CLE	√	√	√	√	√	√	√
CC16CLED	√	√	√	√	√	√	√
CC16RE	√	√	√	√	√	√	√
COMPMC8	√	√	√	√	√	√	√
COMPMC16	√	√	√	√	√	√	√
CY_INIT			√				

Element Name	XC4000E	XC4000X	XC5200	Spartan	SpartanXL	Spartan2	Virtex
CY_MUX			√				
DECODE4	√	√	√	√	√	√	√
DECODE8	√	√	√	√	√	√	√
DECODE16	√	√	√	√	√	√	√
DECODE32			√			√	√
DECODE64			√			√	√
DEC_CC4			√				
DEC_CC8			√				
DEC_CC16			√				
NAND6						√	√
NAND7						√	√
NAND8	√	√	√	√	√	√	√
NAND9	√	√	√	√	√	√	√
NAND12			√			√	√
NAND16			√			√	√
NOR6						√	√
NOR7						√	√
NOR8	√	√	√	√	√	√	√
NOR9	√	√	√	√	√	√	√
NOR12			√			√	√
NOR16			√			√	√
OR6						√	√
OR7						√	√
OR8						√	√
OR9						√	√
OR12			√			√	√
OR16			√			√	√
XNOR6						√	√
XNOR7						√	√
XNOR8						√	√
XNOR9						√	√
XOR6						√	√
XOR7						√	√
XOR8						√	√
XOR9						√	√

Functional Categories

This section categorizes, by function, the logic elements that are described in detail in the "Design Elements" sections. Each category is briefly described. Tables under each category identify all the available elements for the function and indicate which libraries include the element.

Elements are listed in alphanumeric order under each category. There are a number of standard TTL 7400-type functions in the different libraries. All 7400-type functions start with a "X74" prefix and are listed after all other elements. The numeric sequence following the "X74" prefix uses ascending numbers, for example, X74_42 precedes X74_138.

A check mark (√) in the column under the library name means that the element applies to the devices that use that library. (Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific device families that use each library.) A blank column means that the element does not apply.

The categories are as follows.

- Arithmetic Functions
- Buffers
- Comparators
- Counters
- Data Registers
- Decoders
- Edge Decoders
- Encoders
- Flip-Flops
- General
- Input/Output Flip-Flops
- Input/Output Functions
- Input Latches
- Latches
- Logic Primitives
- Map Elements
- Memory Elements
- Multiplexers
- Shift Registers
- Shifters

Note: When converting your design between FPGA families, use elements that have equivalent functions in each of the architectural families (libraries) to minimize re-designing.

Arithmetic Functions

There are three types of arithmetic functions: accumulators (ACC), adders (ADD), and adder / subtracters (ADSU). With an ADSU, either unsigned binary or twos-complement operations cause an overflow. If the result crosses the overflow boundary, an overflow is generated. Similarly, when the result crosses the carry-out boundary, a carry-out is generated. The following figure shows the ADSU carry-out and overflow boundaries.

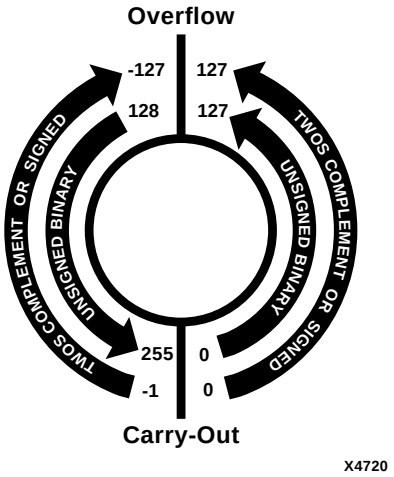


Figure 2-1 ADSU Carry-Out and Overflow Boundaries

ACC1		1-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

ACC4, 8, 16		4-, 8-, 16-Bit Loadable Cascadable Accumulators with Carry-In, Carry-Out, and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

ADD1		1-Bit Full Adder with Carry-In and Carry-Out						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

ADD4, 8, 16		4-, 8-, 16-Bit Cascadable Full Adders with Carry-In, Carry-Out, and Overflow						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

ADSU1		1-Bit Cascadable Adder / Subtracter with Carry-In, Carry-Out						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

ADSU4, 8, 16		4-, 8-, 16-Bit Cascadable Adders / Subtracters with Carry-In, Carry-Out and Overflow						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

X74_280		9-Bit Odd / Even Parity Generator / Checker						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√		

X74_283		4-Bit Full Adder with Carry-In and Carry-Out						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√		

Buffers

The buffers in this section route high fan-out signals, 3-state signals, and clocks inside a PLD device. The "Input / Output Functions" section later in this chapter covers off-chip interface buffers.

ACLK		Alternate Clock Buffer						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√								

BUF		General-Purpose Buffer						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

BUF4, 8, 16		General-Purpose Buffers						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

BUFCF		Fast Connect Buffer						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

BUFE, 4, 8, 16		Internal 3-State Buffers with Active High Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√*	√	√	√	√

* not supported for XC9500XL and XC9500XV devices

BUFFCLK		Global Fast Clock Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√						

BUFG		Global Clock Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

BUFGDLL		Clock Delay Locked Loop Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

BUFGE		Global Low Early Clock Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√						

BUFGLS		Global Low Skew Clock Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

BUFGP		Primary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√		√		√		√	√

BUFGS		Secondary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√		√		√			

BUFGSR		Global Set / Reset Input Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

BUFGTS		Global Three-State Input Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

BUFOD		Open-Drain Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

BUFT, 4, 8, 16		Internal 3-State Buffers with Active-Low Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√*	√	√	√	√

* not supported for XC9500XL and XC9500XV devices

GCLK		Global Clock Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

Comparators

There are two types of comparators, identity (COMP) and magnitude (COMPM).

COMP2, 4, 8, 16		2-, 4-, 8-, 16-Bit Identity Comparators						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

COMPM2, 4, 8, 16		2-, 4-, 8-, 16-Bit Magnitude Comparators						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

COMPMC8, 16		8-, 16-Bit Magnitude Comparators						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

X74_L85		4-Bit Expandable Magnitude Comparator						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_518		8-Bit Identity Comparator with Active-Low Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_521		8-Bit Identity Comparator with Active-Low Enable and Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

Counters

There are six types of counters with various synchronous and asynchronous inputs. The name of the counter defines the modulo or bit size, the counter type, and which control functions are included. The counter naming convention is shown in the following figure.

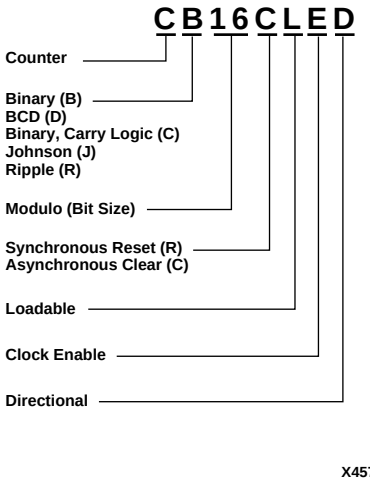


Figure 2-2 Counter Naming Convention

A carry-lookahead design accommodates large counters without extra gating. On TTL 7400-type counters with trickle clock enable (ENT), parallel clock enable (ENP), and ripple carry-out (RCO), both the ENT and ENP inputs must be High to count. ENT is propagated forward to enable RCO, which produces a High output with the approximate duration of the QA output. The following figure illustrates a carry-lookahead design.

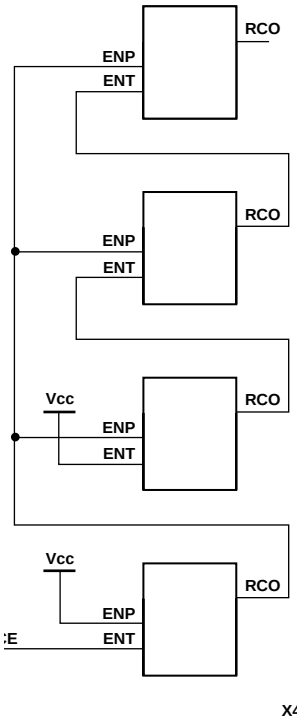


Figure 2-3 Carry-Lookahead Design

The RCO output of the first stage of the ripple carry is connected to the ENP input of the second stage and all subsequent stages. The RCO output of the second stage and all subsequent stages is connected to the ENT input of the next stage. The ENT of the second stage is always enabled / tied to VCC. CE is always connected to the ENT input of the first stage. This cascading method allows the first stage of the ripple carry to be built as a prescaler. In other words, the first stage is built to count very fast.

CB2CE, CB4CE, CB8CE, CB16CE		2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CB2CLE, CB4CLE, CB8CLE, CB16CLE		2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CB2CLED, CB4CLED, CB8CLED, CB16CLED		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CB2RE, CB4RE, CB8RE, CB16RE		2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CB2RLE, CB4RLE, CB8RLE, CB16RLE		2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

CB2X1, CB4X1, CB8X1, CB16X1		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

CB2X2, CB4X2, CB8X2, CB16X2		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

CC8CE, CC16CE		8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

CC8CLE, CC16CLE		8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

CC8CLED, CC16CLED		8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

CC8RE, CC16RE		8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

CD4CE		4-Bit Cascadable BCD Counter with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CD4CLE		4-Bit Loadable Cascadable BCD Counter with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CD4RE		4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CD4RLE		4-Bit Loadable Cascadable BCD Counter with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CJ4CE, CJ5CE, CJ8CE		4-, 5-, 8-Bit Johnson Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CJ4RE, CJ5RE, CJ8RE		4-, 5-, 8-Bit Johnson Counters with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CR8CE, CR16CE		8-, 16-Bit Negative-Edge Binary Ripple Counters with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

X74_160		4-Bit BCD Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_161		4-Bit Binary Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_162		4-Bit BCD Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_163		4-Bit Binary Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_168		4-Bit BCD Bidirectional Counter with Parallel and Trickle Clock Enables and Active-Low Load Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_390		4-Bit BCD / Bi-Quinary Ripple Counter with Negative-Edge Clocks and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

Data Registers

There are three TTL 7400-type data registers designed to function exactly as the TTL elements for which they are named.

X74_174		6-Bit Data Register with Active-Low Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_273		8-Bit Data Register with Active-Low Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_377		8-Bit Data Register with Active-Low Clock Enable						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√		

Decoders

Decoder names, shown in the following figure, indicate the number of inputs and outputs and if an enable is available. Decoders with an enable can be used as multiplexers. This group includes some standard TTL 7400-type decoders whose names have an "X74" prefix.

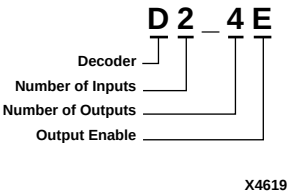


Figure 2-4 Decoder Naming Convention

D2_4E		2- to 4-Line Decoder / Demultiplexer with Enable						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

D3_8E		3- to 8-Line Decoder / Demultiplexer with Enable						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

D4_16E		4- to 16-Line Decoder / Demultiplexer with Enable						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

DEC_CC4, 8, 16		4-, 8-, 16-Bit Active Low Decoders						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
			√				√	√

X74_42		4- to 10-Line BCD-to-Decimal Decoder with Active-Low Outputs						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√		

X74_138		3- to 8-Line Decoder / Demultiplexer with Active-Low Outputs and Three Enables						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_139		2- to 4-Line Decoder / Demultiplexer with Active-Low Outputs and Active-Low Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_154		4- to 16-Line Decoder / Demultiplexer with Two Enables and Active-Low Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

Edge Decoders

Edge decoders are open-drain wired-AND gates that are available in different bit sizes.

DECODE4, 8, 16		4-, 8-, 16-Bit Active-Low Decoders						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√				√	√

DECODE32, 64		32- and 64-Bit Active-Low Decoders						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√				√	√

Encoders

There are two priority encoders (ENCPR) that function like the TTL 7400-type elements they are named after. There is a 10- to 4-line BCD encoder and an 8- to 3-line binary encoder.

X74_147		10- to 4-Line Priority Encoder with Active-Low Inputs and Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_148		8- to 3-Line Cascadable Priority Encoder with Active-Low Inputs and Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

Flip-Flops

There are three types of flip-flops (D, J-K, toggle) with various synchronous and asynchronous inputs. Some are available with inverted clock inputs and / or the ability to set in response to global set / reset rather than reset. The naming convention shown in the following figure provides a description for each flip-flop. D-type flip-flops are available in multiples of up to 16 in one macro.

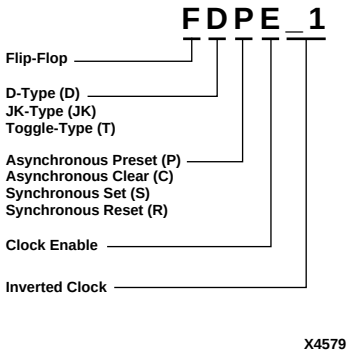


Figure 2-5 Flip-Flop Naming Convention

FD		D Flip-Flop						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FD_1		D Flip-Flop with Negative-Edge Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

FD4, 8, 16		Multiple D Flip-Flops						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

FD4CE, FD8CE, FD16CE		4-, 8-, 16-Bit Data Registers with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FD4RE, FD8RE, FD16RE		4-, 8-, 16-Bit Data Registers with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDC		D Flip-Flop with Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDC_1		D Flip-Flop with Negative-Edge Clock and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

FDCE		D Flip-Flop with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDCE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

FDCP		D Flip-Flop with Asynchronous Preset and Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√			√	√

FDCP_1		D Flip-Flop with Negative-Edge Clock and Asynchronous Preset and Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDCPE		D Flip-Flop with Clock Enable and Asynchronous Preset and Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√			√	√

FDCPE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset and Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDE		D Flip-Flop with Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDE_1		D Flip-Flop with Negative-Edge Clock and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDP		D Flip-Flop with Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FDP_1		D Flip-Flop with Negative-Edge Clock and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

FDPE		D Flip-Flop with Clock Enable and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FDPE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

FDR		D Flip-Flop with Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√			

FDR_1		D Flip-Flop with Negative-Edge Clock and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDRE		D Flip-Flop with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDRE_1		D Flip-Flop with Negative-Clock Edge, Clock Enable, and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDRS		D Flip-Flop with Synchronous Reset and Synchronous Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDRS_1		D Flip-Flop with Negative-Clock Edge and Synchronous Reset and Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDRSE		D Flip-Flop with Synchronous Reset and Set and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDRSE_1		D Flip-Flop with Negative-Clock Edge, Synchronous Reset and Set, and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDS		D Flip-Flop with Synchronous Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDS_1		D Flip-Flop with Negative-Edge Clock and Synchronous Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDSE		D Flip-Flop with Clock Enable and Synchronous Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FDSE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Synchronous Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

FDSR		D Flip-Flop with Synchronous Set and Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

FDSRE		D Flip-Flop with Synchronous Set and Reset and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

FJKC		J-K Flip-Flop with Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FJKCE		J-K Flip-Flop with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FJKCP		J-K Flip-Flop with Asynchronous Clear and Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

FJKCPE		J-K Flip-Flop with Asynchronous Clear and Preset and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

FJKP		J-K Flip-Flop with Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FJKPE		J-K Flip-Flop with Clock Enable and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FJKRSE		J-K Flip-Flop with Clock Enable and Synchronous Reset and Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FJKSRE		J-K Flip-Flop with Clock Enable and Synchronous Set and Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTC		Toggle Flip-Flop with Toggle Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTCE		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTCLE		Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTCLEX		Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

FTCP		Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

FTCPE		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

FTCPLE		Loadable Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

FTP		Toggle Flip-Flop with Toggle Enable and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FTPE		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FTPLE		Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√	√	√	√	√	√

FTRSE		Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTRSLE		Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTSRE		Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FTSRLE		Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

General

General elements include FPGA configuration functions, oscillators, boundary scan logic, and other functions not classified in other sections.

BSCAN		Boundary Scan Logic Control Circuit						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√		

BSCAN_SPARTAN2		Spartan2 Boundary Scan Logic Control Circuit						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	

BSCAN_VIRTEX		Virtex Boundary Scan Logic Control Circuit						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
								√

BYPOSC		Bypass Oscillator						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

CAPTURE_SPARTAN2		Spartan2 Register State Capture for Bitstream Readback						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	

CAPTURE_VIRTEX		Virtex Register State Capture for Bitstream Readback						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
								√

CK_DIV		Internal Multiple-Frequency Clock Divider						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

CLB		CLB Configuration Symbol						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

CLKDLL		Clock Delay Locked Loop						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

CLKDLLHF		High Frequency Clock Delay Locked Loop						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

CONFIG		Repository for Schematic-Level (Global) Attributes						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

CY_INIT		Initialization Stage for Carry Chain						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

GND		Ground-Connection Signal Tag						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

GXTL		Crystal Oscillator with ACLK Buffer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

IOB		IOB Configuration Symbol						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

KEEPER		KEEPER Symbol						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LUT1, 2, 3, 4		1-, 2-, 3-, 4-Bit Look-Up-Table with General Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LUT1_D, LUT2_D, LUT3_D, LUT4_D		1-, 2-, 3-, 4-Bit Look-Up-Table with Dual Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LUT1_L, LUT2_L, LUT3_L, LUT4_L		1-, 2-, 3-, 4-Bit Look-Up-Table with Local Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MD0		Mode 0, Input Pad Used for Readback Trigger Input						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√					

MD1		Mode 1, Output Pad Used for Readback Data Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√					

MD2		Mode 2, Input Pad						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√					

OSC		Crystal Oscillator Amplifier						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

OSC4		Internal 5-Frequency Clock-Signal Generator						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OSC5		Internal Multiple-Frequency Clock-Signal Generator						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

OSC52		Internal Multiple-Frequency Clock-Signal Generator						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

PULLDOWN		Resistor to GND for Input Pads						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

PULLUP		Resistor to VCC for Input PADs, Open-Drain, and 3-State Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

READBACK		FPGA Bitstream Readback Controller						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√		

STARTUP		User Interface to Global Clock, Reset, and 3-State Controls						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√		

STARTUP_SPARTAN2		Spartan2 User Interface to Global Clock, Reset, and 3-State Controls						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	

STARTUP_VIRTEX		Virtex User Interface to Global Clock, Reset, and 3-State Controls						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
								√

TCK		Boundary Scan Test Clock Input Pad						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√		

TDI		Boundary Scan Test Data Input Pad						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√		

TDO		Boundary Scan Data Output Pad						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√		

TIMEGRP		Schematic-Level Table of Basic Timing Specification Groups						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

TIMESPEC		Schematic-Level Timing Requirement Table						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

TMS		Boundary Scan Test Mode Select Input Pad						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√		

VCC		VCC-Connection Signal Tag						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Input/Output Flip-Flops

Input/Output flip-flops are configured in IOBs. They include flip-flops whose outputs are enabled by 3-state buffers, flip-flops that can be set upon global set/reset rather than reset, and flip-flops with inverted clock inputs. The naming convention specifies each flip-flop function and is illustrated in the following figure.

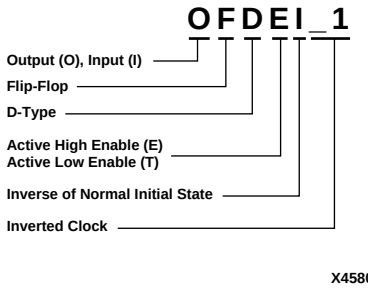


Figure 2-6 Input/Output Flip-Flop Naming Convention

IFD, 4, 8, 16		Single- and Multiple-Input D Flip-Flops						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

IFD_1		Input D Flip-Flop with Inverted Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

IFDI		Input D Flip-Flop (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

IFDI_1		Input D Flip-Flop with Inverted Clock (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

IFDX, 4, 8, 16		Single- and Multiple-Input D Flip-Flops with Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

IFDX_1		Input D Flip-Flop with Inverted Clock and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

IFDXI		Input D Flip-Flop with Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

IFDXI_1		Input D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

OFD, 4, 8, 16		Single- and Multiple-Output D Flip-Flops						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OFD_1		Output D Flip-Flop with Inverted Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

OFDE, 4, 8, 16		D Flip-Flops with Active-High Enable Output Buffers						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OFDE_1		D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

OFDEI		D Flip-Flop with Active-High Enable Output Buffer (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDEI_1		D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDEX, 4, 8, 16		D Flip-Flops with Active-High Enable Output Buffers and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDEX_1		D Flip-Flop with Active-High Enable Output Buffer, Inverted Clock, and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDEXI		D Flip-Flop with Active-High Enable Output Buffer and Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDEXI_1		D Flip-Flop with Active-High Enable Output Buffer, Inverted Clock, and Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDI		Output D Flip-Flop (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

OFDI_1		Output D Flip-Flop with Inverted Clock (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

OFDT, 4, 8, 16		Single and Multiple D Flip-Flops with Active-Low 3-State Output Buffers						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OFDT_1		D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

OFDTI		D Flip-Flop with Active-Low 3-State Output Buffer (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDTI_1		D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDTX, 4, 8, 16		Single and Multiple D Flip-Flops with Active-Low 3-State Output Buffers and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDTX_1		D Flip-Flop with Active-Low 3-State Output Buffer, Inverted Clock, and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDTXI		D Flip-Flop with Active-Low 3-State Output Buffer and Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDTXI_1		D Flip-Flop with Active-Low 3-State Output Buffer, Inverted Clock, and Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√		

OFDX, 4, 8, 16		Single- and Multiple-Output D Flip-Flops with Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

OFDX_1		Output D Flip-Flop with Inverted Clock and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

OFDXI		Output D Flip-Flop with Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

OFDXI_1		Output D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

Input/Output Functions

Input/Output Block (IOB) resources are configured into various I/O primitives and macros for convenience, such as, output buffers (OBUFs) and output buffers with an enable (OBUFEs). Pads used to connect the circuit to PLD device pins are also included.

Virtex and Spartan2 have multiple variants (primitives) to choose from for each selectI/O buffer. The I/O interface for each variant corresponds to a specific I/O standard.

IBUF, 4, 8, 16		Single- and Multiple-Input Buffers						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

IBUF_selectIO		Single Input Buffer with Selectable I/O Interface (16 primitives)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

IBUFG_selectIO		Dedicated Input Buffer with Selectable I/O Interface (16 primitives)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

IOBUF_selectIO		Bi-Directional Buffer with Selectable I/0 Interface (30 primitives)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

IOPAD, 4, 8, 16		Single- and Multiple-Input/Output Pads						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

IPAD, 4, 8, 16		Single- and Multiple-Input Pads						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OBUF, 4, 8, 16		Single- and Multiple-Output Buffers						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OBUF_selectIO		Single Output Buffer with Selectable I/O Interface (30 primitives)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

OBUFE, 4, 8, 16		3-State Output Buffers with Active-High Output Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OBUFT, 4, 8, 16		Single and Multiple 3-State Output Buffers with Active Low Output Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OBUFT_selectIO		Single 3-State Output Buffer with Active-Low Output Enable and Selectable I/O Interface (30 primitives)						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

OPAD, 4, 8, 16		Single- and Multiple-Output Pads						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

UPAD		Connects the I/O Node of an IOB to the Internal PLD Circuit						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Input Latches

Single and multiple input latches can hold transient data entering a chip. Input latches use the same naming convention as I/O flip-flops.

ILD, 4, 8, 16		Transparent Input Data Latches						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

ILD_1		Transparent Input Data Latch with Inverted Gate						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

ILDI		Transparent Input Data Latch (Asynchronous Preset)						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√			√	√	√	√

ILDI_1		Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√			√	√	√	√

ILDX, 4, 8, 16		Transparent Input Data Latches						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√			√	√	√	√

ILDX_1		Transparent Input Data Latch with Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

ILDXI		Transparent Input Data Latch (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

ILDXI_1		Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

ILFFX		Fast Capture Input Latch						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

ILFFXI		Fast Capture Input Latch (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

ILFLX		Fast Capture Transparent Input Latch						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

ILFLX_1		Fast Capture Input Latch with Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

ILFLXI_1		Fast Capture Input Latch with Inverted Gate (Asynchronous Preset)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

Latches

Latches (LD) are only available in the XC4000X, XC5200, XC9000, Spartan2, SpartanXL, and Virtex architectures. XC3000 and XC4000E latches that existed in previous macro libraries are not recommended for new designs.

LD		Transparent Data Latch						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√	√		√	√	√

LD_1		Transparent Data Latch with Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√			√	√	√

LD4, 8, 16		Multiple Transparent Data Latches						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√		√		√	√	√

LDC		Transparent Data Latch with Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√			√	√	√

LDC_1		Transparent Data Latch with Asynchronous Clear and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√			√	√	√

LDCE		Transparent Data Latch with Asynchronous Clear and Gate Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√			√	√	√

LDCE_1		Transparent Data Latch with Asynchronous Clear, Gate Enable, and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√			√	√	√

LD4CE, LD8CE, LD16CE		Transparent Data Latches with Asynchronous Clear and Gate Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√	√			√	√	√

LDCP		Transparent Data Latch with Asynchronous Clear and Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDCP_1		Transparent Data Latch with Asynchronous Clear and Preset and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDCPE		Transparent Data Latch with Asynchronous Clear and Preset and Gate Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDCPE_1		Transparent Data Latch with Asynchronous Clear and Preset, Gate Enable, and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDE		Transparent Data Latch with Gate Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDE_1		Transparent Data Latch with Gate Enable and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDP		Transparent Data Latch with Asynchronous Preset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDP_1		Transparent Data Latch with Asynchronous Preset and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LDPE		Transparent Data Latch with Asynchronous Preset and Gate Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√	√	√

LDPE_1		Transparent Data Latch with Asynchronous Preset, Gate Enable, and Inverted Gate						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√	√	√

Logic Primitives

Combinatorial logic gates that implement the basic Boolean functions are available in all architectures with up to five inputs in all combinations of inverted and non-inverted inputs, and with six to nine inputs non-inverted.

AND2-9		2- to 9-Input AND Gates with Inverted and Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

AND12, 16		12- and 16-Input AND Gates with Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√				√	√

INV, 4, 8, 16		Single and Multiple Inverters						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

MULT_AND		Fast Multiplier AND						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

NAND2-9		2- to 9-Input NAND Gates with Inverted and Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

NAND12, 16		12- and 16-Input NAND Gates with Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√				√	√

NOR2-9		2- to 9-Input NOR Gates with Inverted and Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

NOR12, 16		12 and 16-Input NOR Gates with Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√				√	√

OAND2		2-Input AND Gate with Invertible Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

ONAND2		2-Input NAND Gate with Invertible Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

ONOR2		2-Input NOR Gate with Invertible Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

OOR2		2-Input OR Gate with Invertible Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

OR2-9		2- to 9-Input OR Gates with Inverted and Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

OR12, 16		12- and 16-Input OR Gates with Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√				√	√

OXNOR2		2-Input Exclusive-NOR Gate with Invertible Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

OXOR2		2-Input Exclusive-OR Gate with Invertible Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

SOP3-4		Sum of Products						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

WAND1, 4, 8, 16		Open-Drain Buffers						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

WOR2AND		2-Input OR Gate with Wired-AND Open-Drain Buffer Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

XNOR2-9		2- to 9-Input XNOR Gates with Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

XOR2-9		2- to 9-Input XOR Gates with Non-Inverted Inputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

XORCY		XOR for Carry Logic with General Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

XORCY_D		XOR for Carry Logic with Dual Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

XORCY_L		XOR for Carry Logic with Local Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

Map Elements

Map elements are used in conjunction with logic symbols to constrain the logic to particular CLBs or particular F or H function generators.

CLBMAP		Logic-Partitioning Control Symbol						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√								

F5MAP		5-Input Function Partitioning Control Symbol						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
			√					

FMAP		F Function Generator Partitioning Control Symbol						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√	√	√

HMAP		H Function Generator Partitioning Control Symbol						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√			√	√		

Memory Elements

The XC4000, Spartan, and SpartanXL architectures have a number of static RAM configurations defined as macros. In the Virtex and Spartan2 architectures, they are defined as primitives. These 16- or 32-word RAMs are 1, 2, 4, and 8 bits wide. There are two ROMs in the XC4000, Spartan and SpartanXL architectures, 16X1 and 32X1.

The Virtex and Spartan2 series have dedicated blocks of on-chip 4096-bit single-port and dual-port synchronous RAM. Each port is configured to a specific data width. There are five single-port block RAM primitives and 30 dual-port block RAM primitives.

RAM16X1		16-Deep by 1-Wide Static RAM						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√						

RAM16X1D		16-Deep by 1-Wide Static Dual Port Synchronous RAM						
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√			√	√	√	√

RAM16X1D_1		16-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

RAM16X1S		16-Deep by 1-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM16X1S_1		16-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

RAM16X2		16-Deep by 2-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM16X2D		16-Deep by 2-Wide Static Dual Port Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM16X2S		16-Deep by 2-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM16X4		16-Deep by 4-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM16X4D		16-Deep by 4-Wide Static Dual Port Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM16X4S		16-Deep by 4-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM16X8		16-Deep by 8-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM16X8D		16-Deep by 8-Wide Static Dual Port Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM16X8S		16-Deep by 8-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM32X1		32-Deep by 1-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM32X1S		32-Deep by 1-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM32X1S_1		32-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

RAM32X2		32-Deep by 2-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM32X2S		32-Deep by 2-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM32X4		32-Deep by 4-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM32X4S		32-Deep by 4-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAM32X8		32-Deep by 8-Wide Static RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

RAM32X8S		32-Deep by 8-Wide Static Synchronous RAM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

RAMB4_Sn		4096-Bit Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits (5 primitives)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

RAMB4_Sn_Sn		4096-Bit Dual-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits (30 primitives)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

ROM16X1		16-Deep by 1-Wide ROM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

ROM32X1		32-Deep by 1-Wide ROM						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

Multiplexers

The multiplexer naming convention shown in the following figure indicates the number of inputs and outputs and if an enable is available. There are a number of TTL 7400-type multiplexers that have active-Low or inverted outputs.

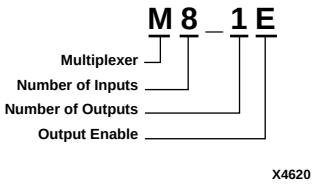


Figure 2-7 Multiplexer Naming Convention

CY_MUX		2-to-1 Multiplexer for Carry Logic						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

F5_MUX		2-to-1 Lookup Table Multiplexer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

M2_1		2-to-1 Multiplexer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

M2_1B1		2-to-1 Multiplexer with D0 Inverted						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

M2_1B2		2-to-1 Multiplexer with D0 and D1 Inverted						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

M2_1E		2-to-1 Multiplexer with Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

M4_1E		4-to-1 Multiplexer with Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

M8_1E		8-to-1 Multiplexer with Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

M16_1E		16-to-1 Multiplexer with Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

MUXCY		2-to-1 Multiplexer for Carry Logic with General Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXCY_D		2-to-1 Multiplexer for Carry Logic with Dual Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXCY_L		2-to-1 Multiplexer for Carry Logic with Local Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXF5		2-to-1 Lookup Table Multiplexer with General Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXF5_D		2-to-1 Lookup Table Multiplexer with Dual Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXF5_L		2-to-1 Lookup Table Multiplexer with Local Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXF6		2-to-1 Lookup Table Multiplexer with General Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXF6_D		2-to-1 Lookup Table Multiplexer with Dual Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

MUXF6_L		2-to-1 Lookup Table Multiplexer with Local Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

OMUX2		2-to-1 Multiplexer						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

X74_150		16-to-1 Multiplexer with Active-Low Enable and Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_151		8-to-1 Multiplexer with Active-Low Enable and Complementary Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_152		8-to-1 Multiplexer with Active-Low Output						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_153		Dual 4-to-1 Multiplexer with Active-Low Enables and Common Select Input						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_157		Quadruple 2-to-1 Multiplexer with Common Select and Active-Low Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_158		Quadruple 2-to-1 Multiplexer with Common Select, Active-Low Enable, and Active-Low Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_298		Quadruple 2-Input Multiplexer with Storage and Negative-Edge Clock						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_352		Dual 4-to-1 Multiplexer with Active-Low Enables and Outputs						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

Shift Registers

Shift registers are available in a variety of sizes and capabilities. The naming convention shown in the following figure illustrates available features.

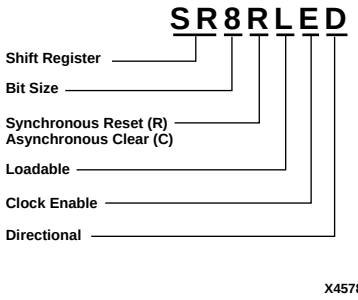


Figure 2-8 Shift Register Naming Convention

SR4CE, SR8CE, SR16CE		4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

SR4CLE, SR8CLE, SR16CLE		4-, 8-, 16-Bit Loadable Serial / Parallel-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

SR4CLED, SR8CLED, SR16CLED		4-, 8-, 16-Bit Shift Registers with Clock Enable and Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

SR4RE, SR8RE, SR16RE		4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

SR4RLE, SR8RLE, SR16RLE		4-, 8-, 16-Bit Loadable Serial / Parallel-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

SR4RLED, SR8RLED, SR16RLED		4-, 8-, 16-Bit Shift Registers with Clock Enable and Synchronous Reset						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

SRL16		16-Bit Shift Register Look-Up-Table (LUT)						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

SRL16_1		16-Bit Shift Register Look-Up-Table (LUT) with Negative-Clock Edge						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

SRL16E		16-Bit Shift Register Look-Up-Table (LUT) with Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

SRL16E_1		16-Bit Shift Register Look-Up-Table (LUT) with Negative-Edge Clock and Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

X74_164		8-Bit Serial-In Parallel-Out Shift Register with Active-Low Asynchronous Clear						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_165S		8-Bit Loadable Serial / Parallel-In Parallel-Out Shift Register with Clock Enable						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_194		4-Bit Loadable Bidirectional Serial / Parallel-In Parallel-Out Shift Register						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

X74_195		4-Bit Loadable Serial / Parallel-In Parallel-Out Shift Register						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√		

Shifters

Shifters are barrel shifters (BRLSHFT) of four and eight bits.

BRLSHFT4, 8		4-, 8-Bit Barrel Shifters						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

Chapter 3

Design Elements (ACC1 to BYPOSC)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

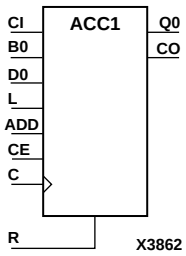
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

ACC1

1-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Macro	N/A	N/A	N/A	N/A



ACC1 can add or subtract a 1-bit unsigned-binary word to or from the contents of a 1-bit data register and store the results in the register. The register can be loaded with a 1-bit word. The synchronous reset (R) has priority over all other inputs and, when High, causes the output to go to logic level zero during the Low-to-High clock (C) transition. Clock (C) transitions are ignored when clock enable (CE) is Low.

The accumulator is asynchronously cleared, outputs Low, when power is applied. For CPLDs the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Load

When the load input (L) is High, CE is ignored and the data on the input D0 is loaded into the 1-bit register during the Low-to-High clock (C) transition.

Add

When control inputs ADD and CE are both High, the accumulator adds a 1-bit word (B0) and carry-in (CI) to the contents of the 1-bit register. The result is stored in the register and appears on output Q0 during the Low-to-High clock transition. The carry-out (CO) is not registered synchronously with the data output. CO always reflects the accumulation of input B0 and the contents of the register, which allows cascading of ACC1s by connecting CO of one stage to CI of the next stage. In add mode, CO acts as a carry-out, and CO and CI are active-High.

Subtract

When ADD is Low and CE is High, the 1-bit word B0 and CI are subtracted from the contents of the register. The result is stored in the register and appears on output Q0 during the Low-to-High clock transition. The carry-out (CO) is not registered synchronously with the data output. CO always reflects the accumulation of input B0 and the contents of the register, which allows cascading of ACC1s by connecting CO of one stage to CI of the next stage. In subtract mode, CO acts as a borrow, and CO and CI are active-Low.

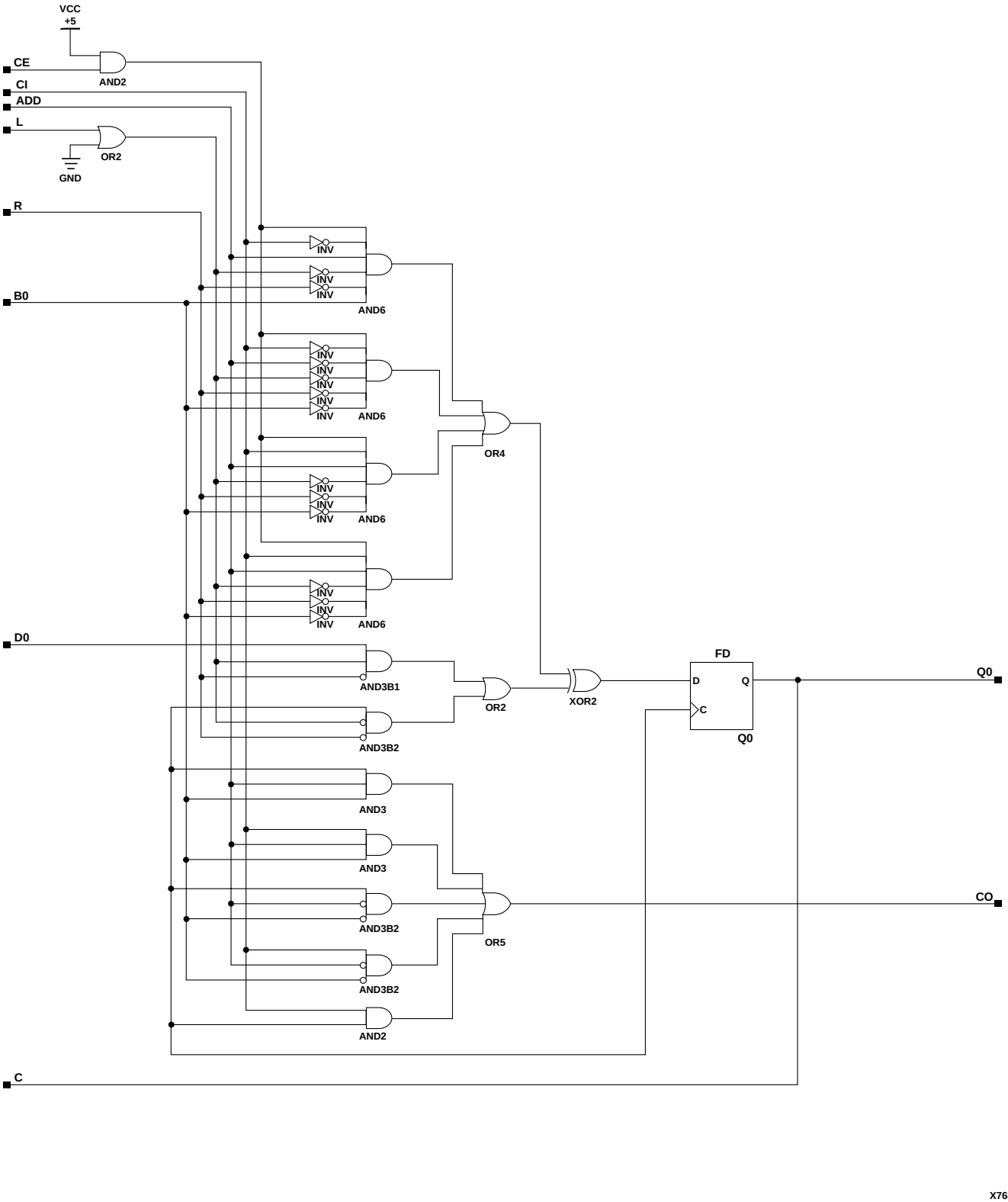
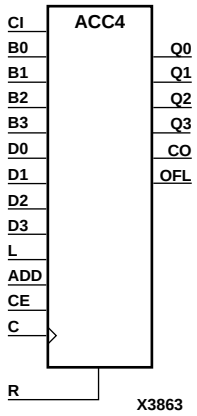


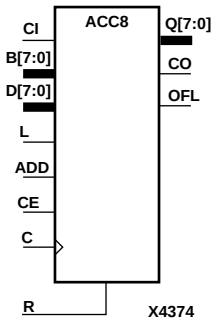
Figure 3-1 ACC1 Implementation XC9000

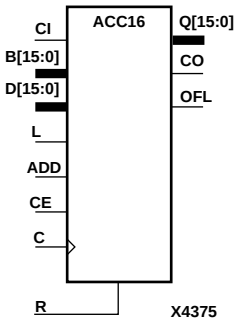
ACC4, 8, 16

4-, 8-, 16-Bit Loadable Cascadable Accumulators with Carry-In, Carry-Out, and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







ACC4, ACC8, ACC16 can add or subtract a 4-, 8-, 16-bit unsigned-binary, respectively or twos-complement word to or from the contents of a 4-, 8-, 16-bit data register and store the results in the register. The register can be loaded with the 4-, 8-, 16-bit word.

In the XC4000, Spartan, and SpartanXL, these accumulators are implemented using carry logic and relative location constraints, which assure most efficient logic placement.

The synchronous reset (R) has priority over all other inputs, and when High, causes all outputs to go to logic level zero during the Low-to-High clock (C) transition. Clock (C) transitions are ignored when clock enable (CE) is Low.

The accumulator is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Load

When the load input (L) is High, CE is ignored and the data on the D inputs is loaded into the register during the Low-to-High clock (C) transition. ACC4 loads the data on inputs D3 – D0 into the 4-bit register. ACC8 loads the data on D7 – D0 into the 8-bit register. ACC16 loads the data on inputs D15 – D0 into the 16-bit register.

Unsigned Binary Versus Twos Complement

ACC4, ACC8, ACC16 can operate, respectively, on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos complement uses OFL to determine when “overflow” occurs.

Unsigned Binary Operation

For unsigned binary operation, ACC4 can represent numbers between 0 and 15, inclusive; ACC8 between 0 and 255, inclusive; and ACC16 between 0 and 65535, inclusive. In add mode, CO is active (High) when the sum exceeds the bounds of the adder/subtracter. In subtract mode, CO is an active-Low borrow-out and goes Low when the difference exceeds the bounds. The carry-out (CO) is not registered synchronously

with the data outputs. CO always reflects the accumulation of the B inputs (B3 – B0 for ACC4, B7 – B0 for ACC8, B15 – B0 for ACC16) and the contents of the register. This allows cascading of ACC4s, ACC8s, or ACC16s by connecting CO of one stage to CI of the next stage. An unsigned binary "overflow" that is always active-High can be generated by gating the ADD signal and CO as follows.

```
unsigned overflow = CO XOR ADD
```

OFL should be ignored in unsigned binary operation.

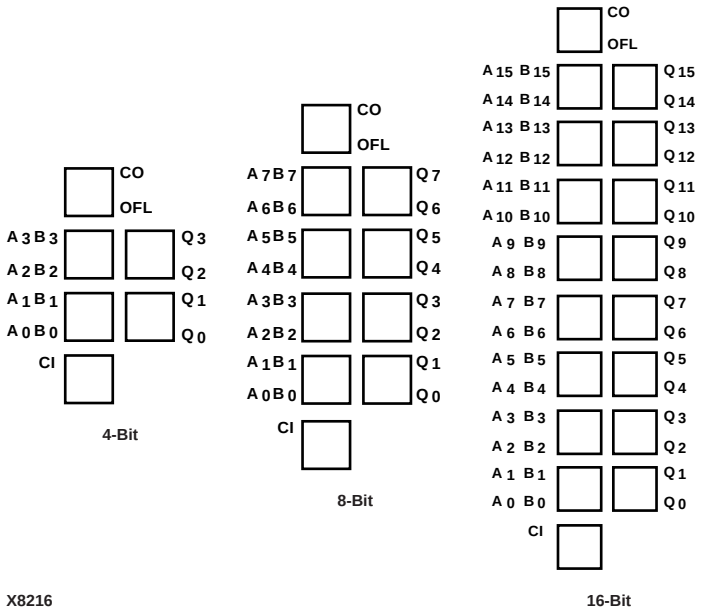
Twos-Complement Operation

For twos-complement operation, ACC4 can represent numbers between -8 and +7, inclusive; ACC8 between -128 and +127, inclusive; ACC16 between -32768 and +32767, inclusive. If an addition or subtraction operation result exceeds this range, the OFL output goes High. The overflow (OFL) is not registered synchronously with the data outputs. OFL always reflects the accumulation of the B inputs (B3 – B0 for ACC4, B7 – B0 for ACC8, B15 – B0 for ACC16) and the contents of the register, which allows cascading of ACC4s, ACC8s, or ACC16s by connecting OFL of one stage to CI of the next stage.

CO should be ignored in twos-complement operation.

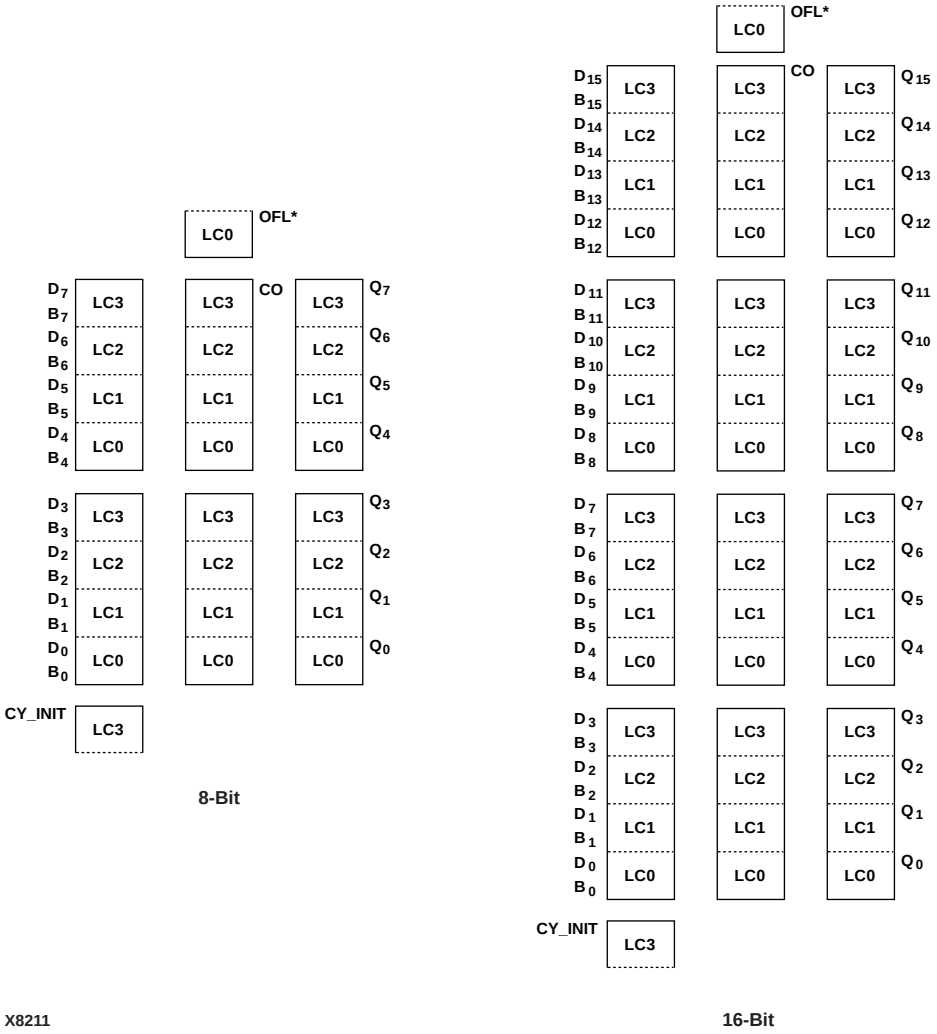
Topology for XC4000, Spartan, SpartanXL

This is the ACC4 (4-bit), ACC8 (8-bit), and ACC16 (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



Topology for XC5200

This is the ACC8 (8-bit) and ACC16 (16-bit) topology for XC5200 devices.



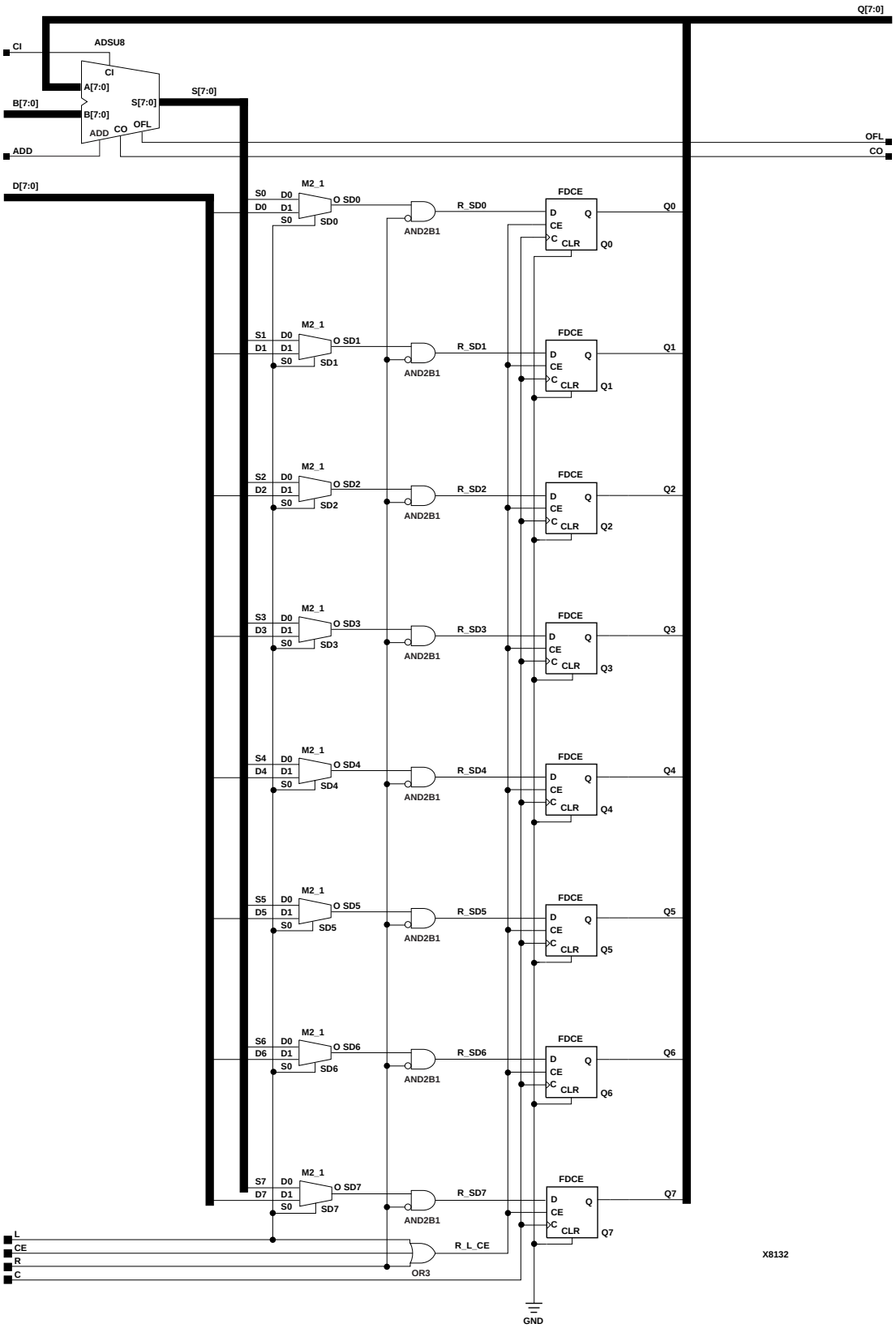


Figure 3-2 ACC8 Implementation XC3000

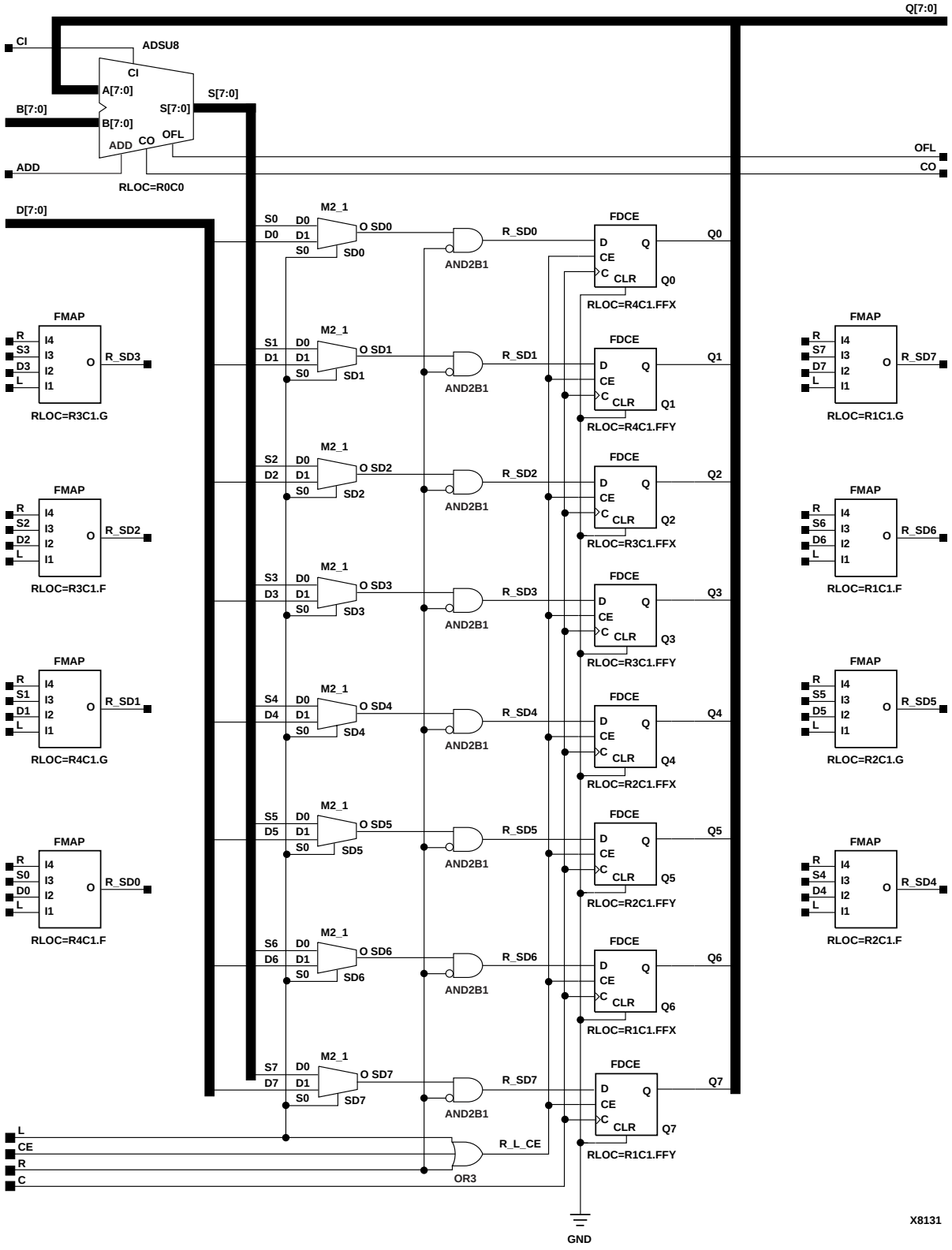


Figure 3-3 ACC8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

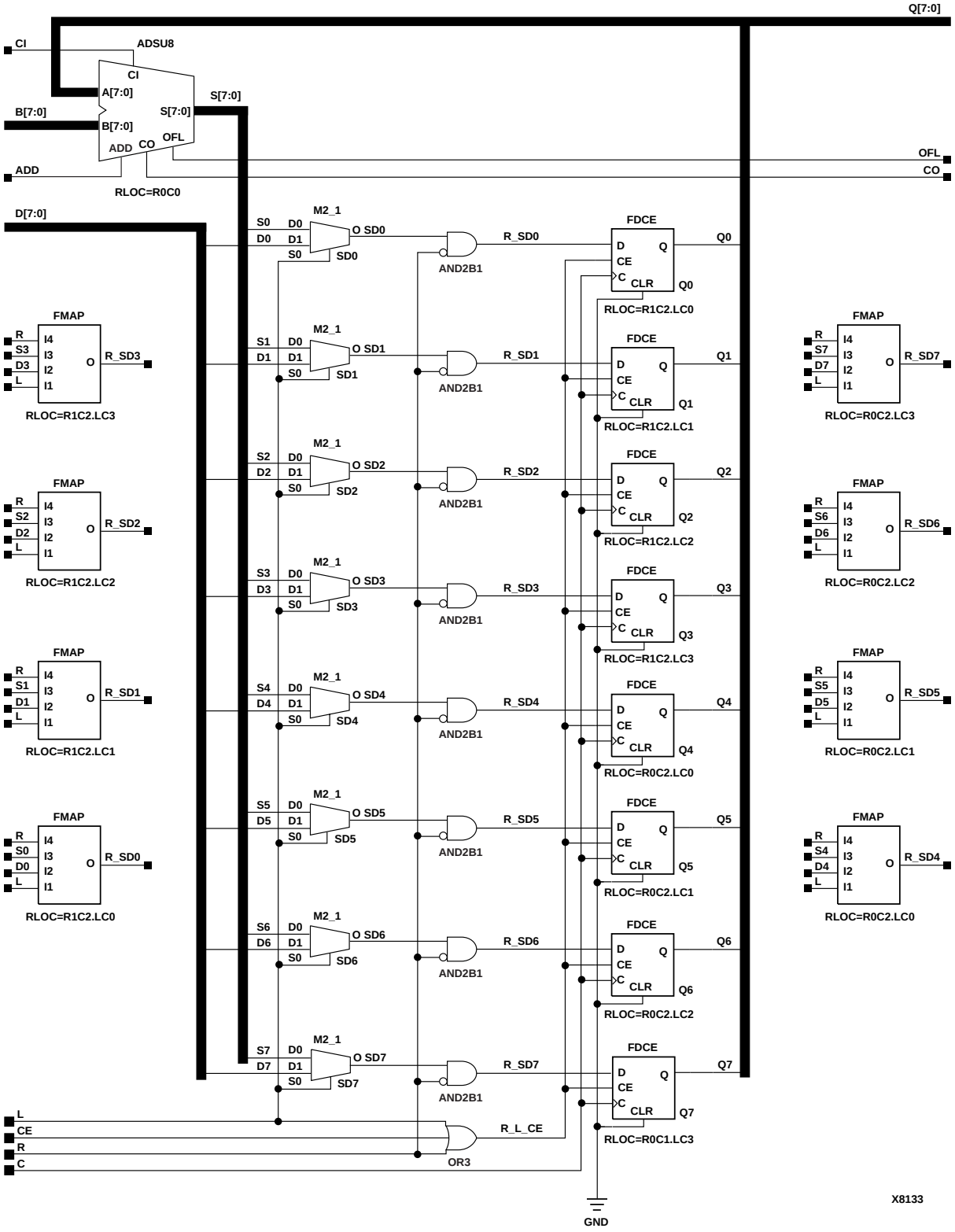


Figure 3-4 ACC8 Implementation XC5200

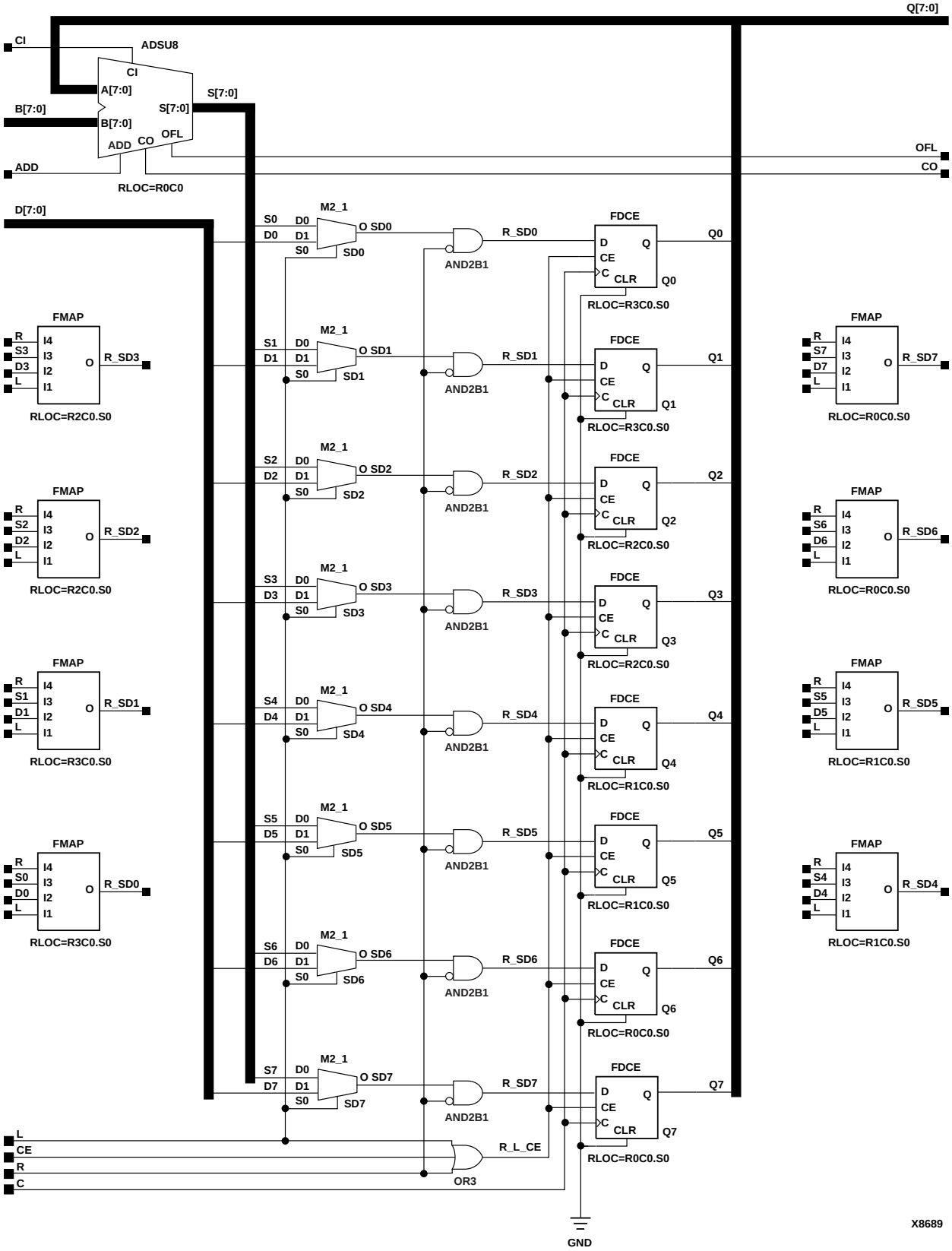


Figure 3-5 ACC8 Implementation Spartan2, Virtex

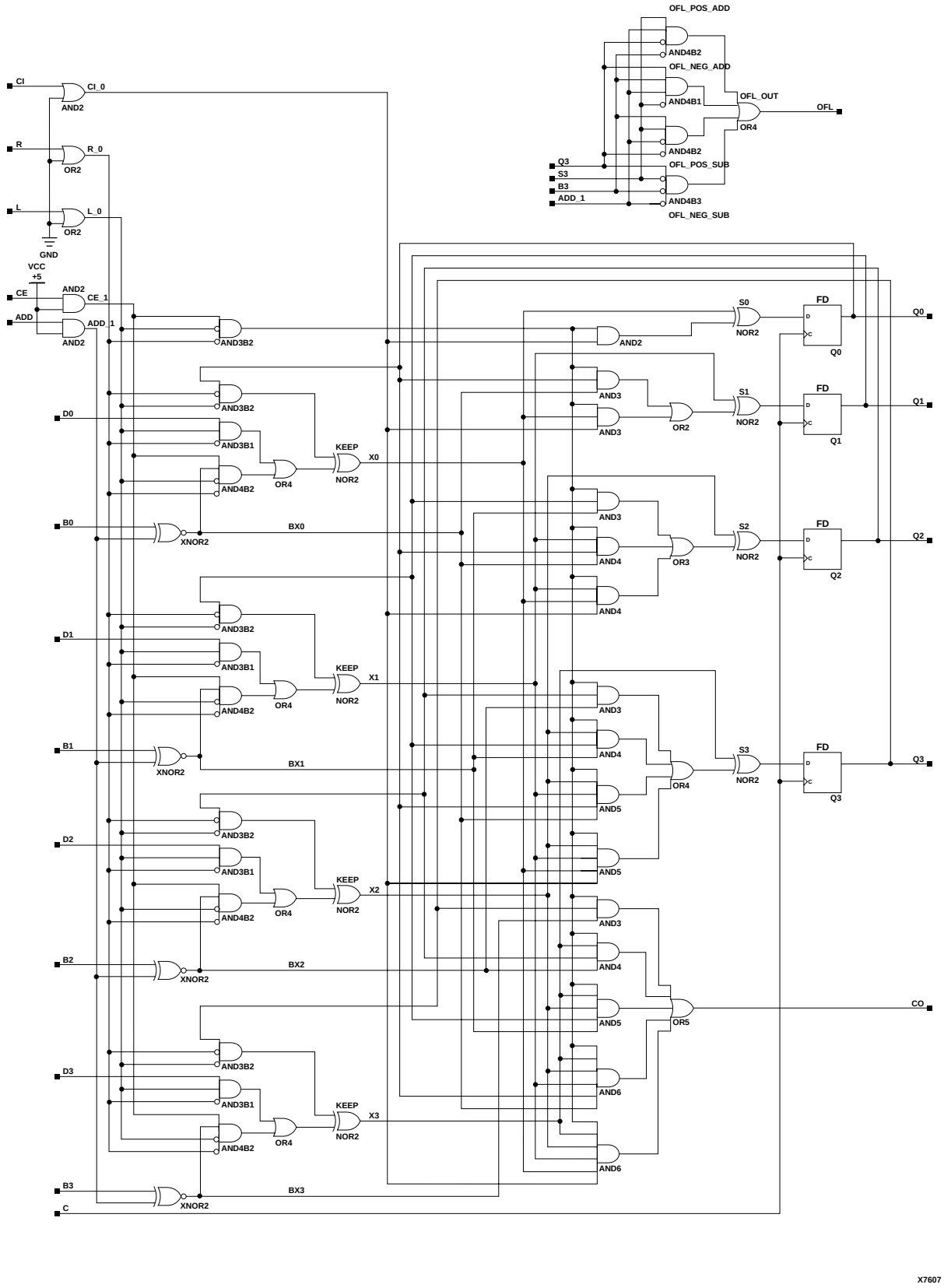


Figure 3-6 ACC4 Implementation XC9000

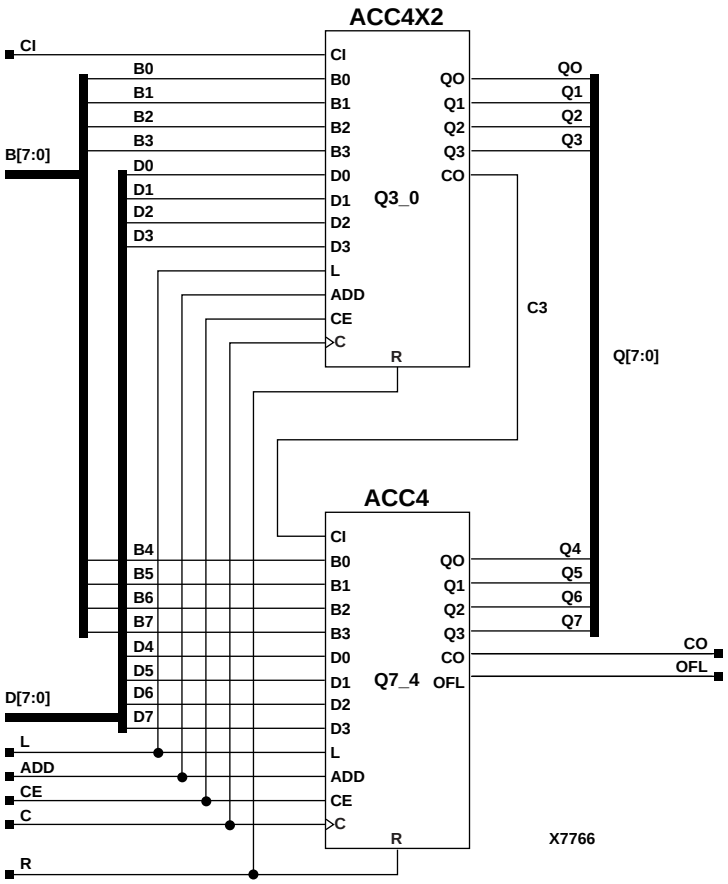


Figure 3-7 ACC8 Implementation XC9000

ACLK

Alternate Clock Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

X3883

ACLK, the alternate clock buffer, is used to distribute high fan-out clock signals throughout a PLD device. One ACLK buffer on each device provides direct access to every Configurable Logic Block (CLB) and Input Output Block (IOB) clock pin. The ACLK buffer is slightly slower than the global clock buffer (GCLK) but otherwise similar. Unlike GCLK, the routing resources used for the ACLK network can be used to route other signals if it is not used. For this reason, if only one of the GCLK and ACLK buffers is used, GCLK is preferred. The ACLK input (I) can come from one of the following sources.

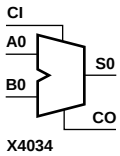
- A CMOS-level signal on the dedicated BCLKIN pin. BCLKIN is a direct CMOS-only input to the ACLK buffer. To use the BCLKIN pin, connect the input of the ACLK element to IBUF and IPAD elements.
- A CMOS- or TTL-level external signal. To connect an external input to the ACLK buffer, connect the input of the ACLK element to the output of the IBUF for that signal. Unless the corresponding IPAD element is constrained otherwise, PAR typically places that IOB directly adjacent to the ACLK buffer.
- The on-chip crystal oscillator. The output of the XTAL oscillator on XC3000 devices is directly adjacent to the ACLK buffer input. If the GXTL element is used, the output of the XTAL oscillator is automatically connected to the ACLK buffer; do not use the ACLK element for anything else.
- An internal signal. To drive the ACLK buffer with an internal signal, connect that signal directly to the input of the ACLK element.

For a negative-edge clock, insert an INV (inverter) element between the ACLK output and the clock input. Inversion is performed inside the CLB, or in the case of IOB clock pins, on the IOB clock line (that controls the clock sense for the IOBs on an entire edge of the chip).

ADD1

1-Bit Full Adder with Carry-In and Carry-Out

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A



ADD1 is a cascadable 1-bit full adder with carry-in and carry-out. It adds two 1-bit words (A and B) and a carry-in (CI), producing a binary sum (S0) output and a carry-out (CO).

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

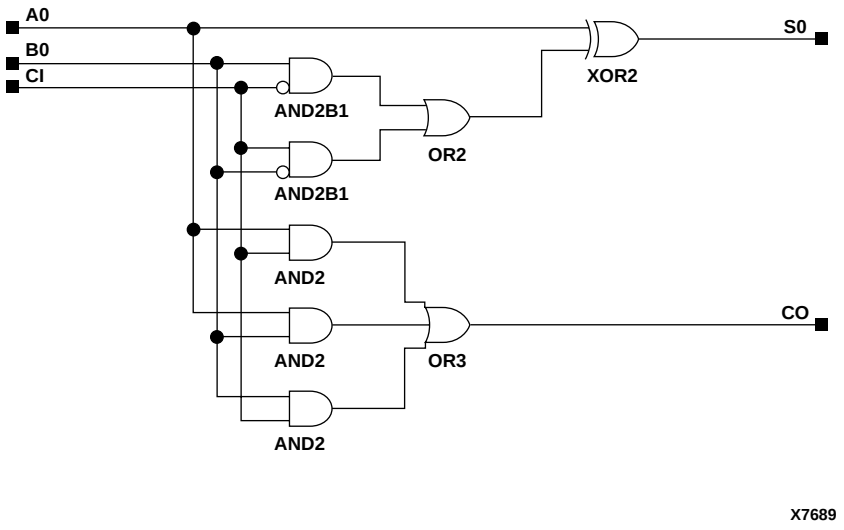
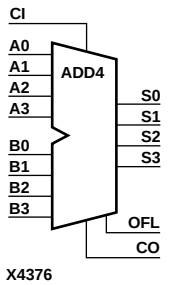


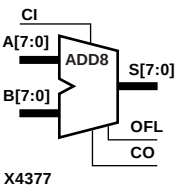
Figure 3-8 ADD1 Implementation XC9000

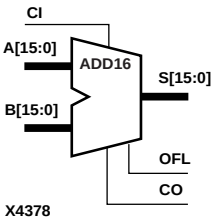
ADD4, 8, 16

4-, 8-, 16-Bit Cascadable Full Adders with Carry-In, Carry-Out, and Overflow

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







ADD4, ADD8, and ADD16 add two words and a carry-in (CI), producing a sum output and carry-out (CO) or overflow (OFL). ADD4 adds A3 – A0, B3 – B0, and CI producing the sum output S3 – S0 and CO (or OFL). ADD8 adds A7 – A0, B7 – B0, and CI, producing the sum output S7 – S0 and CO (or OFL). ADD16 adds A15 – A0, B15 – B0 and CI, producing the sum output S15 – S0 and CO (or OFL).

ADD4, ADD8, and ADD16 are implemented in the XC4000, Spartan, and SpartanXL using carry logic and relative location constraints, which assure most efficient logic placement.

Unsigned Binary Versus Twos Complement

ADD4, ADD8, ADD16 can operate on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers, respectively. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos-complement uses OFL to determine when “overflow” occurs. Therefore, if you want to interpret the inputs as unsigned binary, you should follow the CO output. If you want to interpret the inputs as twos complement, you should follow the OFL output.

Unsigned Binary Operation

For unsigned binary operation, ADD4 can represent numbers between 0 and 15, inclusive; ADD8 between 0 and 255, inclusive; ADD16 between 0 and 65535, inclusive. CO is active (High) when the sum exceeds the bounds of the adder.

OFL is ignored in unsigned binary operation.

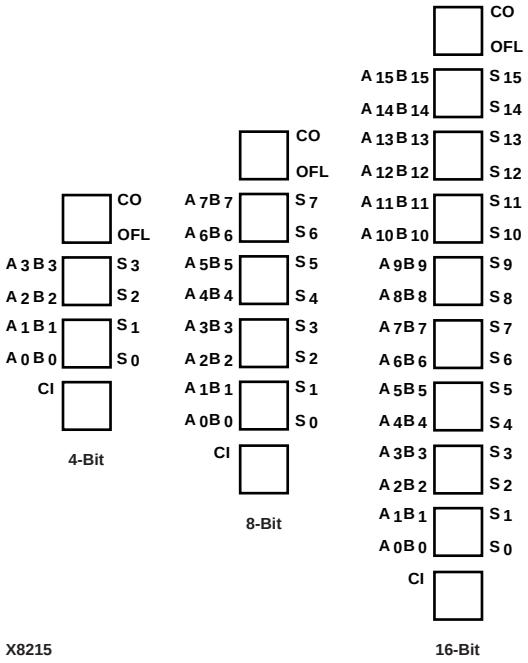
Twos-Complement Operation

For twos-complement operation, ADD4 can represent numbers between -8 and +7, inclusive; ADD8 between -128 and +127, inclusive; ADD16 between -32768 and +32767, inclusive. OFL is active (High) when the sum exceeds the bounds of the adder.

CO is ignored in twos-complement operation.

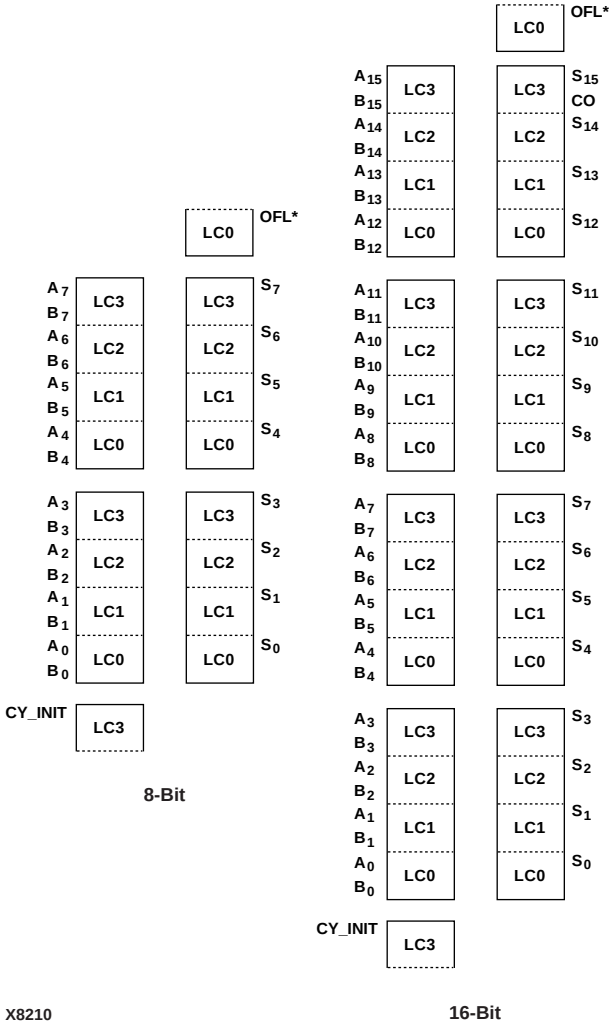
Topology for XC4000, Spartan, SpartanXL

This is the ADD4 (4-bit), ADD8 (8-bit), and ADD16 (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



Topology for XC5200

This is the ADD8 (8-bit) and ADD16 (16-bit) topology for XC5200 devices.



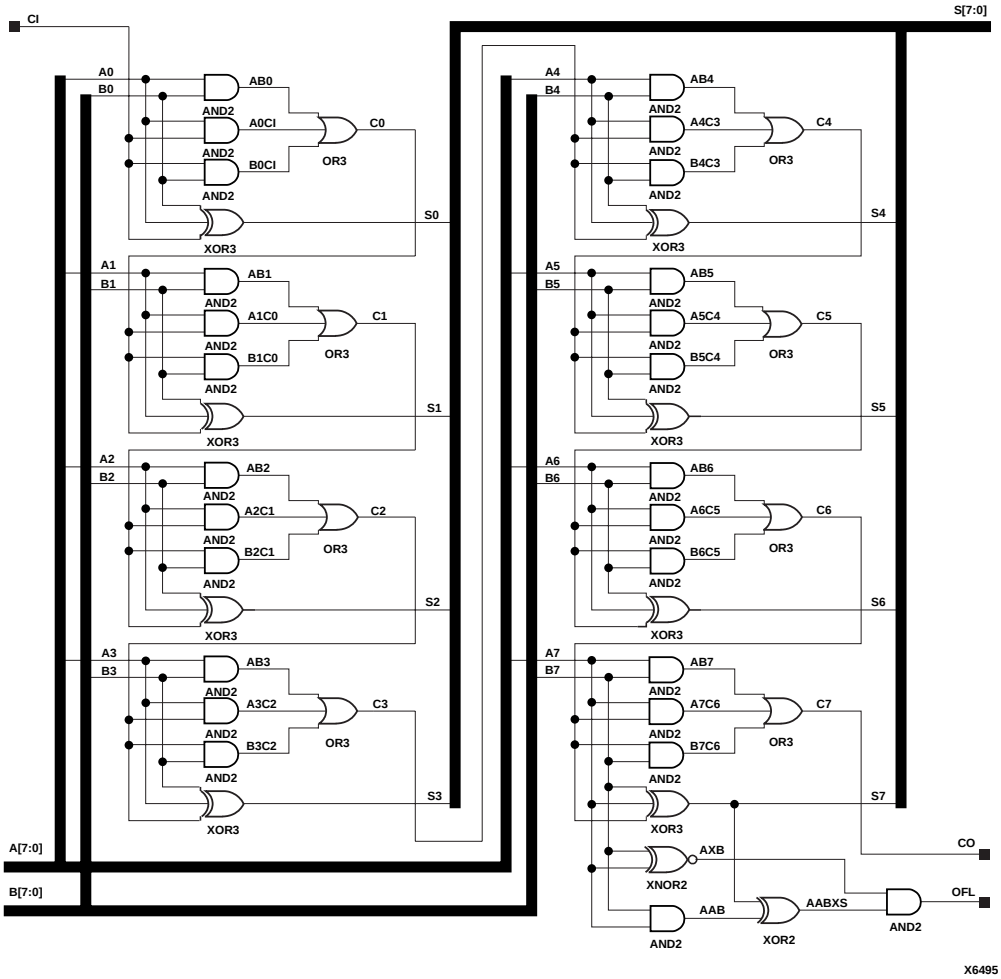


Figure 3-9 ADD8 Implementation XC3000

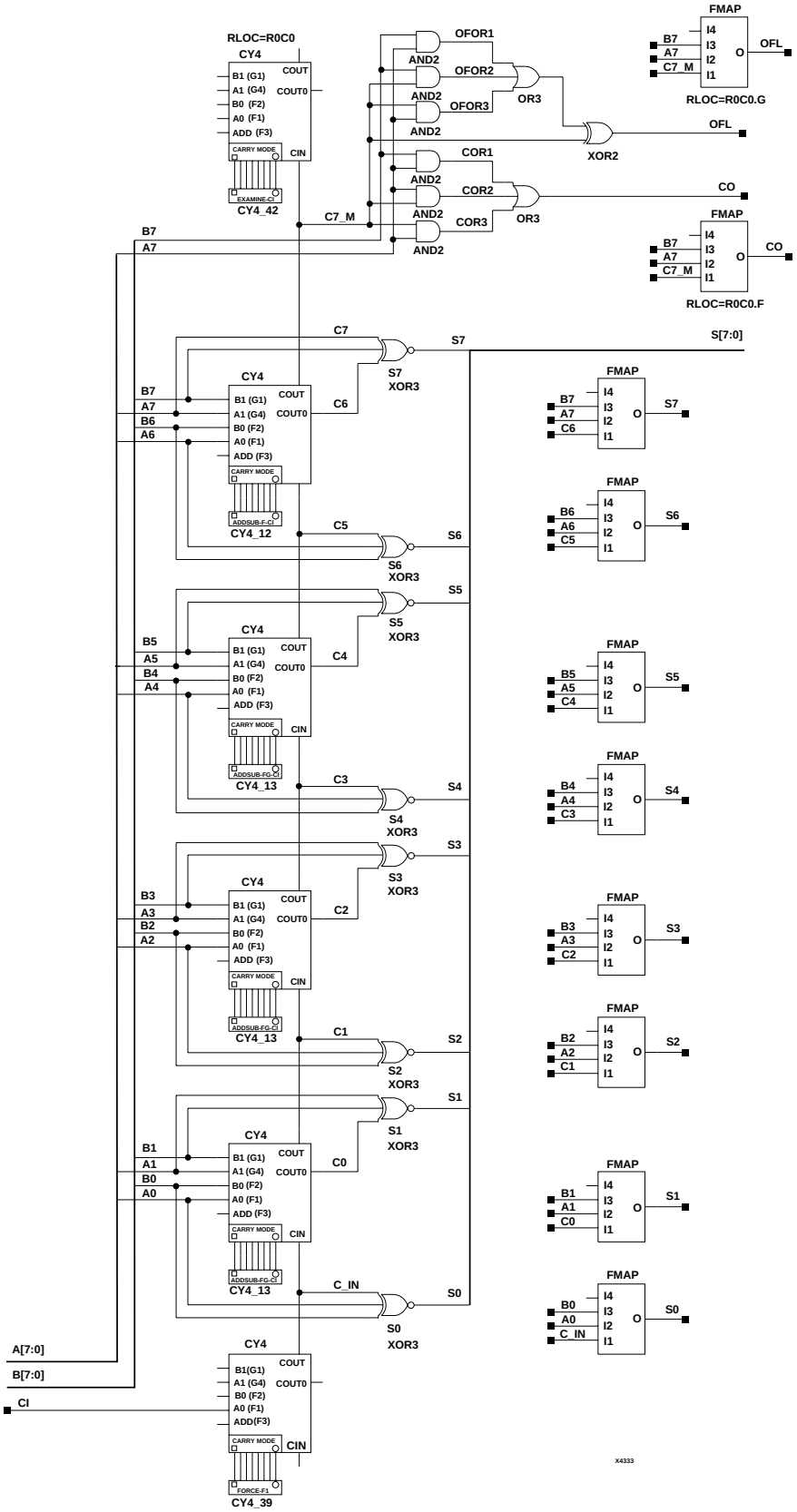


Figure 3-10 ADD8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

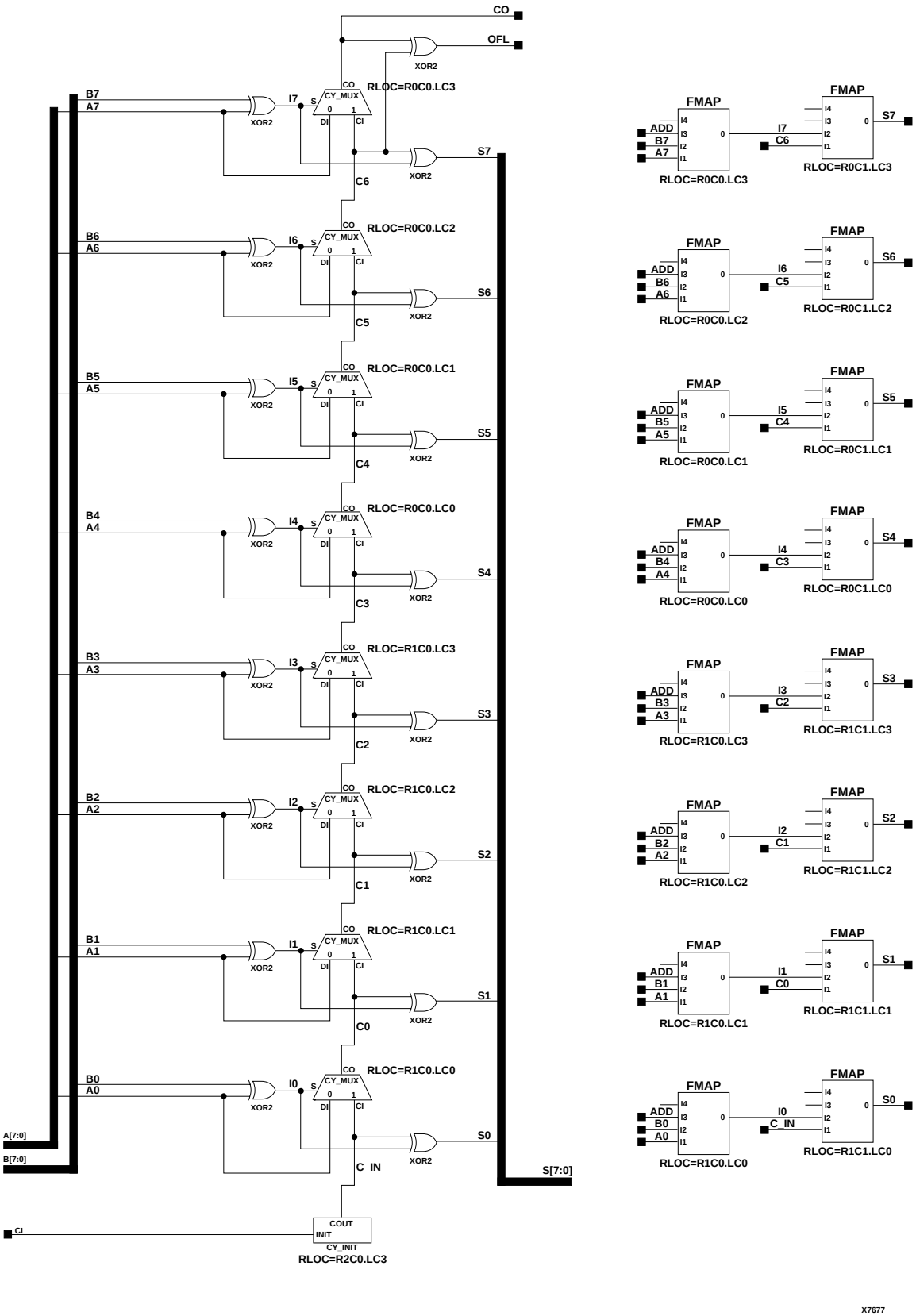


Figure 3-11 ADD8 Implementation XC5200

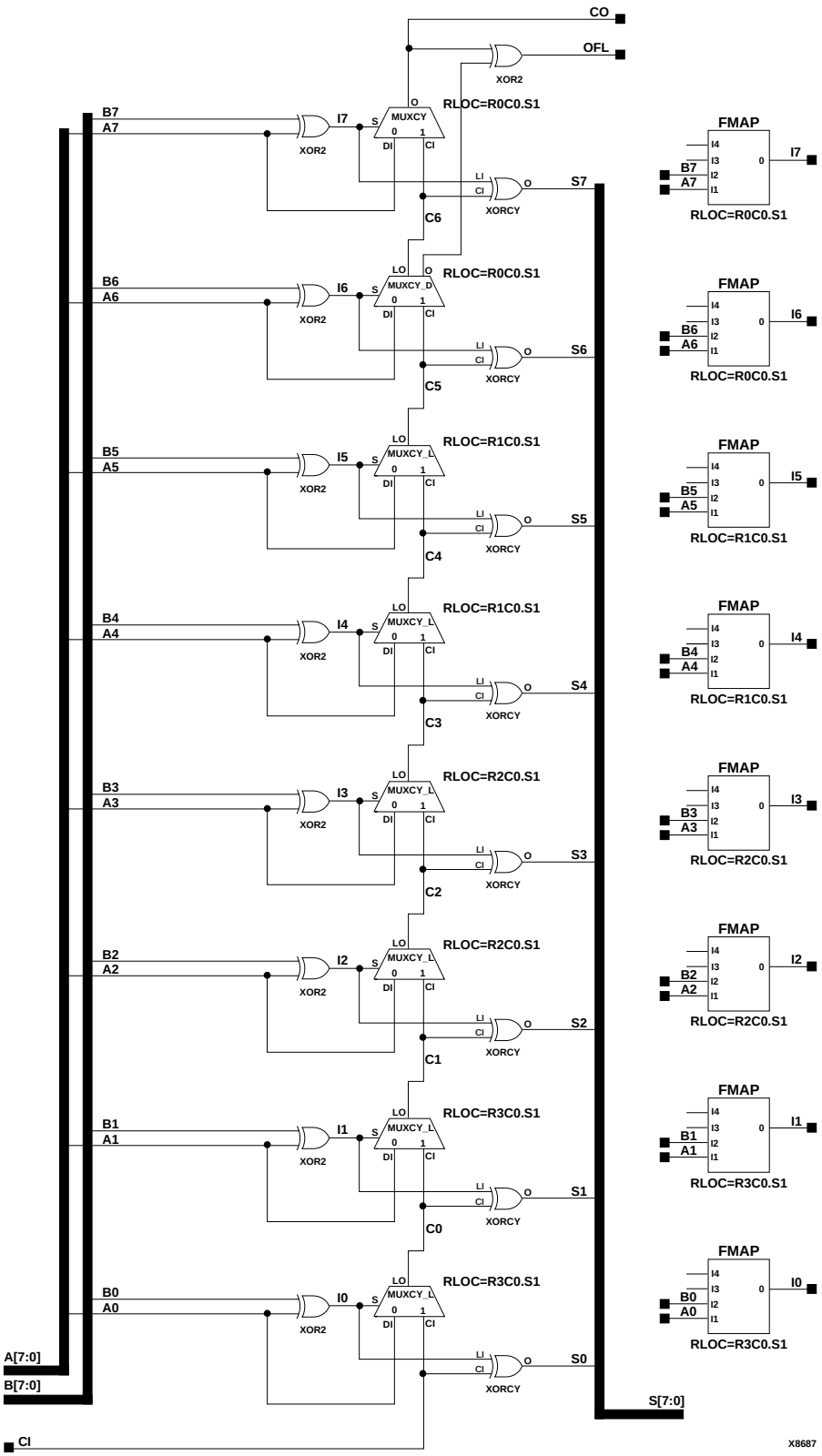


Figure 3-12 ADD8 Implementation Spartan2, Virtex

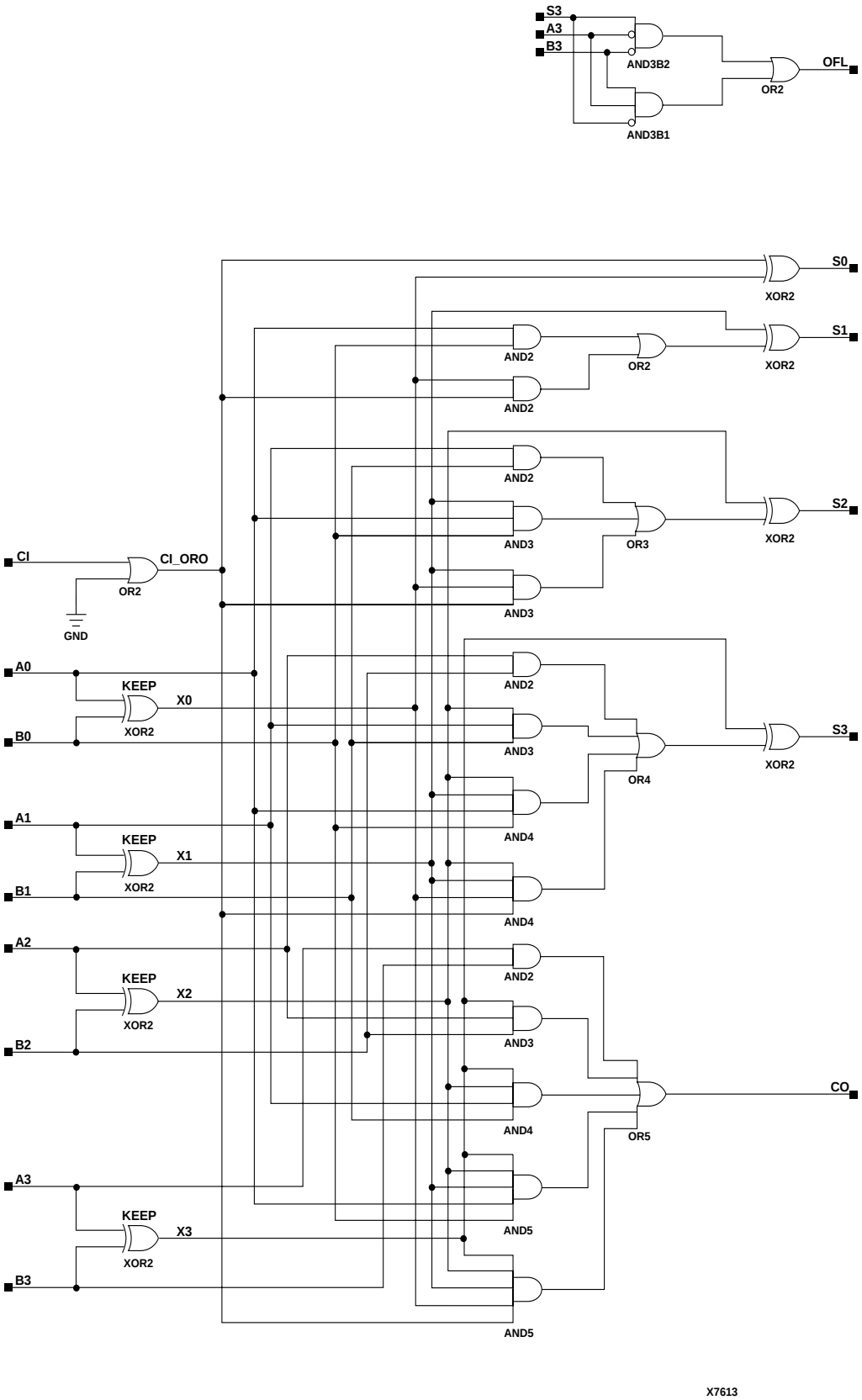


Figure 3-13 ADD4 Implementation XC9000

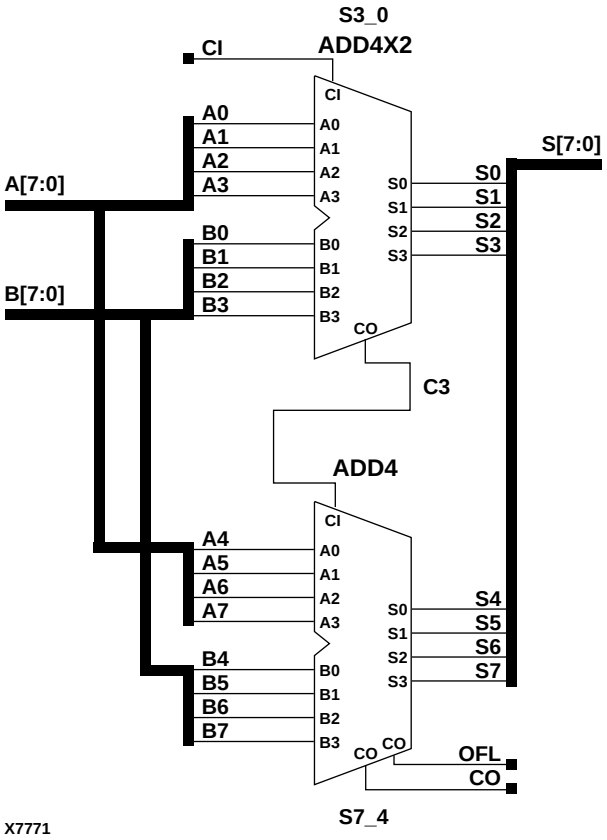
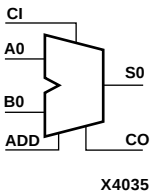


Figure 3-14 ADD8 Implementation XC9000

ADSU1

1-Bit Cascadable Adder/Subtracter with Carry-In and Carry-Out

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A



When the ADD input is High, two 1-bit words (A0 and B0) are added with a carry-in (CI), producing a 1-bit output (S0) and a carry-out (CO). When the ADD input is Low, B0 is subtracted from A0, producing a result (S0) and borrow (CO). In add mode, CO represents a carry-out, and CO and CI are active-High. In subtract mode, CO represents a borrow, and CO and CI are active-Low.

Table 3-1 Add Function, ADD=1

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 3-2 Subtract Function, ADD=0

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	1	0
0	1	0	0	0
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	1
1	1	1	0	1

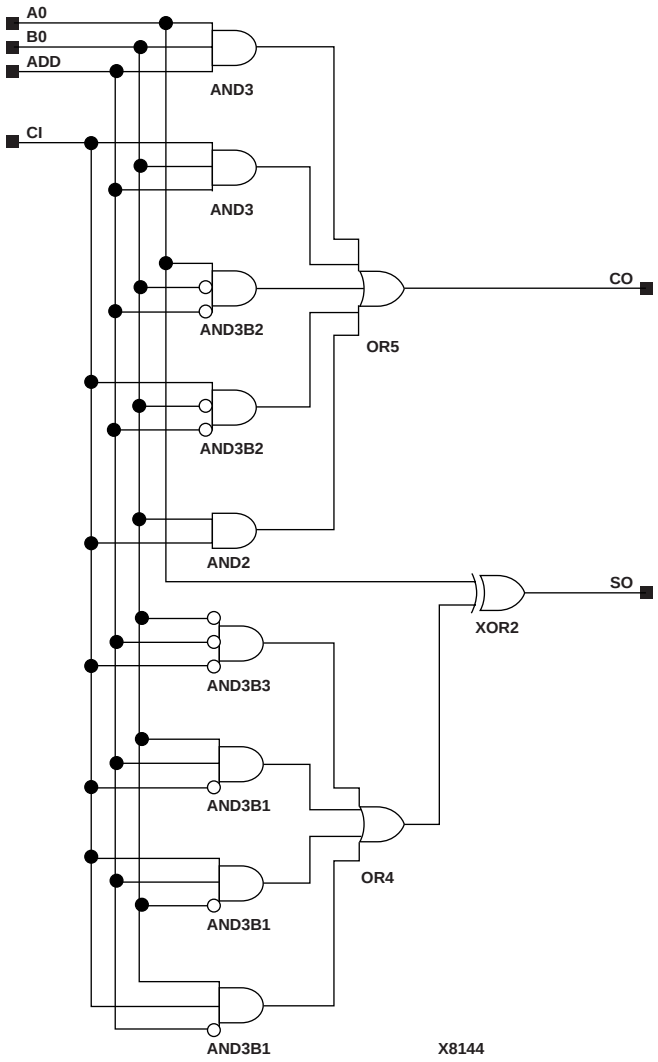
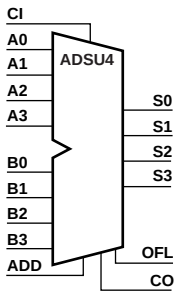


Figure 3-15 ADSU1 Implementation XC9000

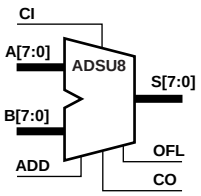
ADSU4, 8, 16

4-, 8-, 16-Bit Cascadable Adders/Subtracters with Carry-In, Carry-Out, and Overflow

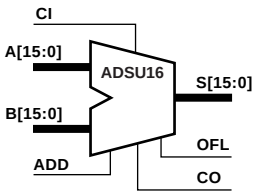
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



X4379



X4380



X4381

When the ADD input is High, ADSU4, ADSU8, and ADSU16 add two words and a carry-in (CI), producing a sum output and carry-out (CO) or overflow (OFL). ADSU4 adds two 4-bit words (A3 – A0 and B3 – B0) and a CI, producing a 4-bit sum output (S3 – S0) and CO or OFL. ADSU8 adds two 8-bit words (A7 – A0 and B7 – B0) and a CI producing, an 8-bit sum output (S7 – S0) and CO or OFL. ADSU16 adds two 16-bit words (A15 – A0 and B15 – B0) and a CI, producing a 16-bit sum output (S15 – S0) and CO or OFL.

When the ADD input is Low, ADSU4, ADSU8, and ADSU16 subtract Bz – B0 from Az– A0, producing a difference output and CO or OFL. ADSU4 subtracts B3 – B0 from A3 – A0, producing a 4-bit difference (S3 – S0) and CO or OFL. ADSU8 subtracts B7 – B0 from A7 – A0, producing an 8-bit difference (S7 – S0) and CO or OFL. ADSU16 subtracts B15 – B0 from A15 – A0, producing a 16-bit difference (S15 – S0) and CO or OFL.

In add mode, CO and CI are active-High. In subtract mode, CO and CI are active-Low. OFL is active-High in add and subtract modes.

ADSU4, ADSU8, and ADU16 are implemented in the XC4000, Spartan, and SpartanXL using carry logic and relative location constraints, which assure most efficient logic placement.

ADSU4, ADSU8, and ADSU16 CI and CO pins do not use the CPLD carry chain.

Unsigned Binary Versus Twos Complement

ADSU4, ADSU8, ADSU16 can operate, respectively, on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos complement uses OFL to determine when “overflow” occurs.

With adder/subtracters, either unsigned binary or twos-complement operations cause an overflow. If the result crosses the overflow boundary, an overflow is generated. Similarly, when the result crosses the carry-out boundary, a carry-out is generated. The following figure shows the ADSU carry-out and overflow boundaries.

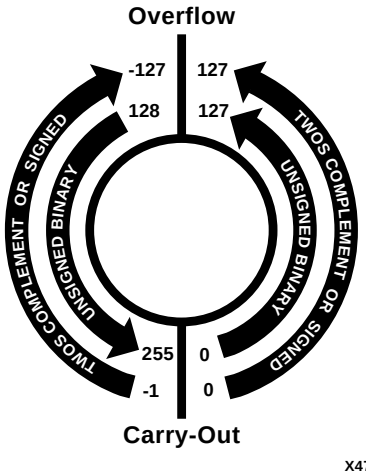


Figure 3-16 ADSU Carry-Out and Overflow Boundaries

Unsigned Binary Operation

For unsigned binary operation, ADSU4 can represent numbers between 0 and 15, inclusive; ADSU8 between 0 and 255, inclusive; ADSU16 between 0 and 65535, inclusive. In add mode, CO is active (High) when the sum exceeds the bounds of the adder/subtracter. In subtract mode, CO is an active-Low borrow-out and goes Low when the difference exceeds the bounds.

An unsigned binary "overflow" that is always active-High can be generated by gating the ADD signal and CO as follows.

```
unsigned overflow = CO XNOR ADD
```

OFL is ignored in unsigned binary operation.

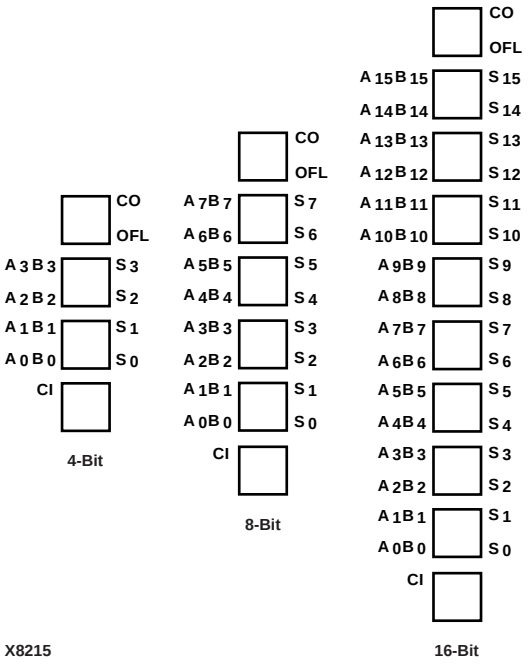
Twos-Complement Operation

For twos-complement operation, ADSU4 can represent numbers between -8 and +7, inclusive; ADSU8 between -128 and +127, inclusive; ADSU16 between -32768 and +32767, inclusive. If an addition or subtraction operation result exceeds this range, the OFL output goes High.

CO is ignored in twos-complement operation.

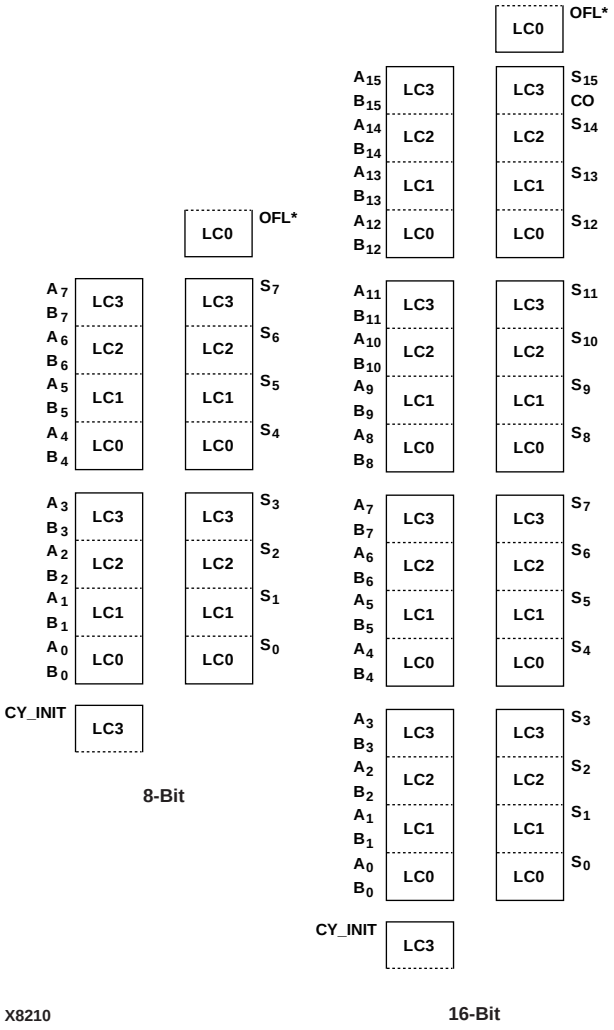
Topology for XC4000, Spartan, SpartanXL

This is the ADSU4 (4-bit), ADSU8 (8-bit), and ADSU16 (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



XC5200 Topology

This is the ADSU8 (8-bit) and ADSU16 (16-bit) topology for XC5200 devices.



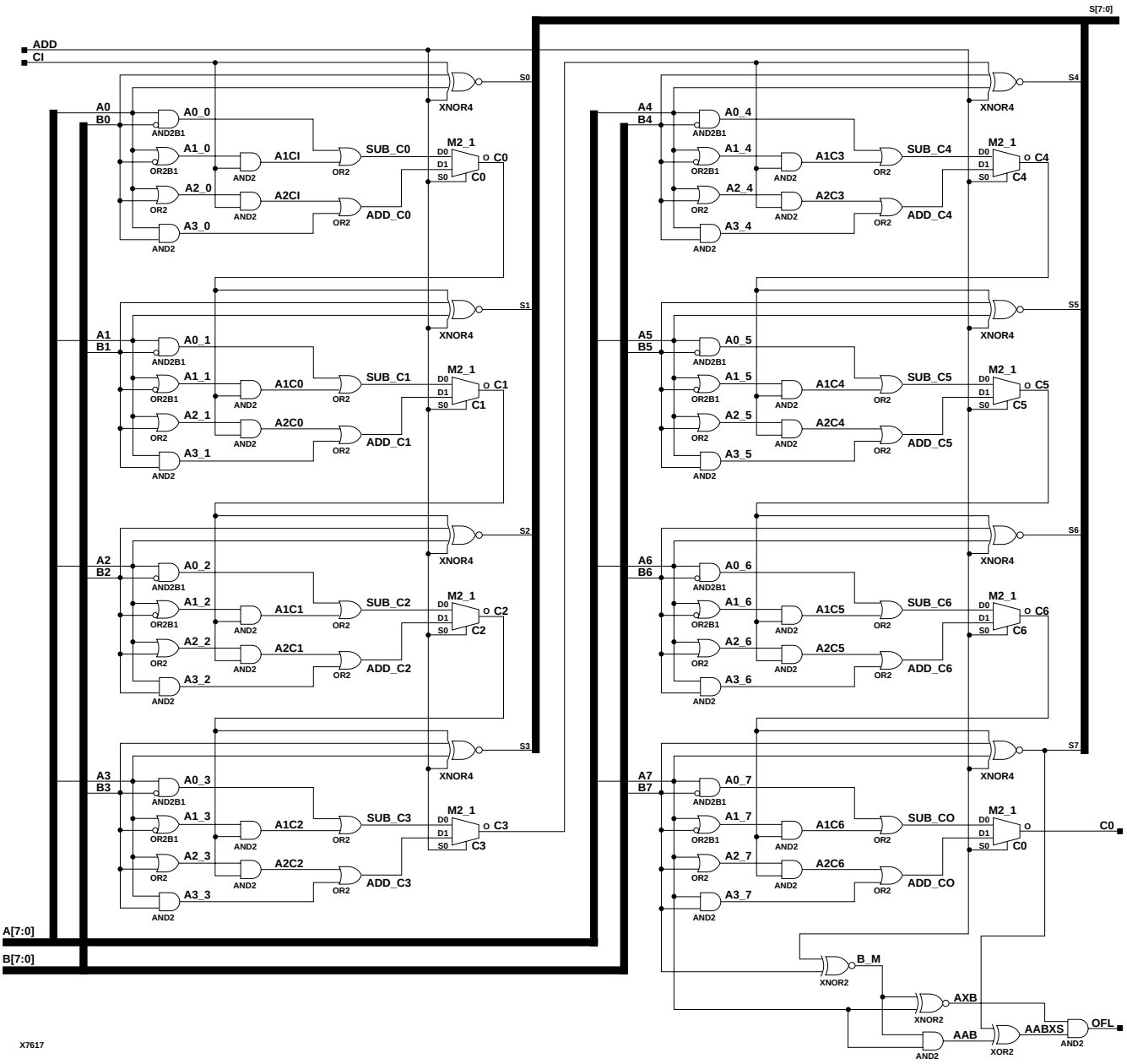


Figure 3-17 ADSU8 Implementation XC3000

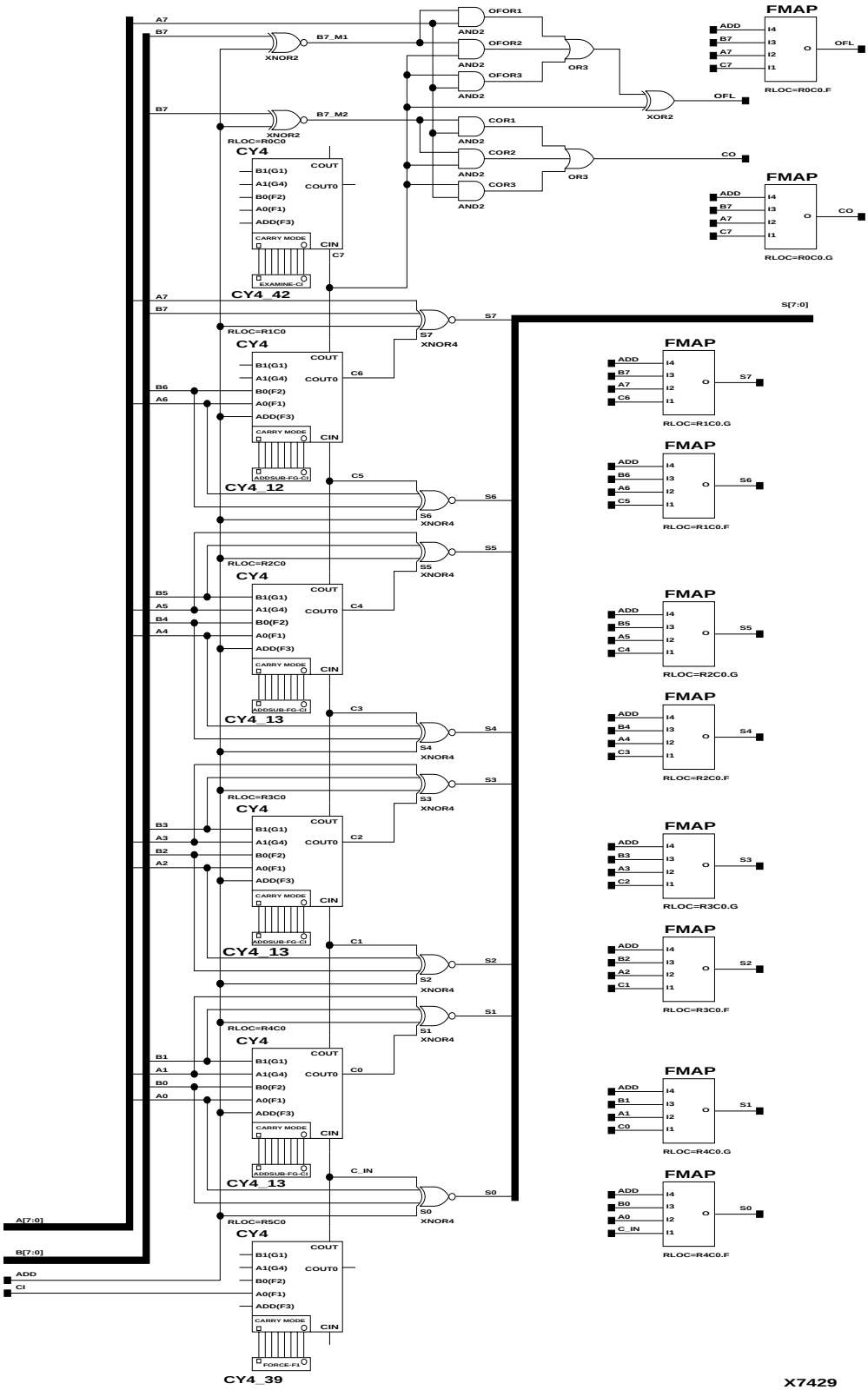


Figure 3-18 ADSU8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

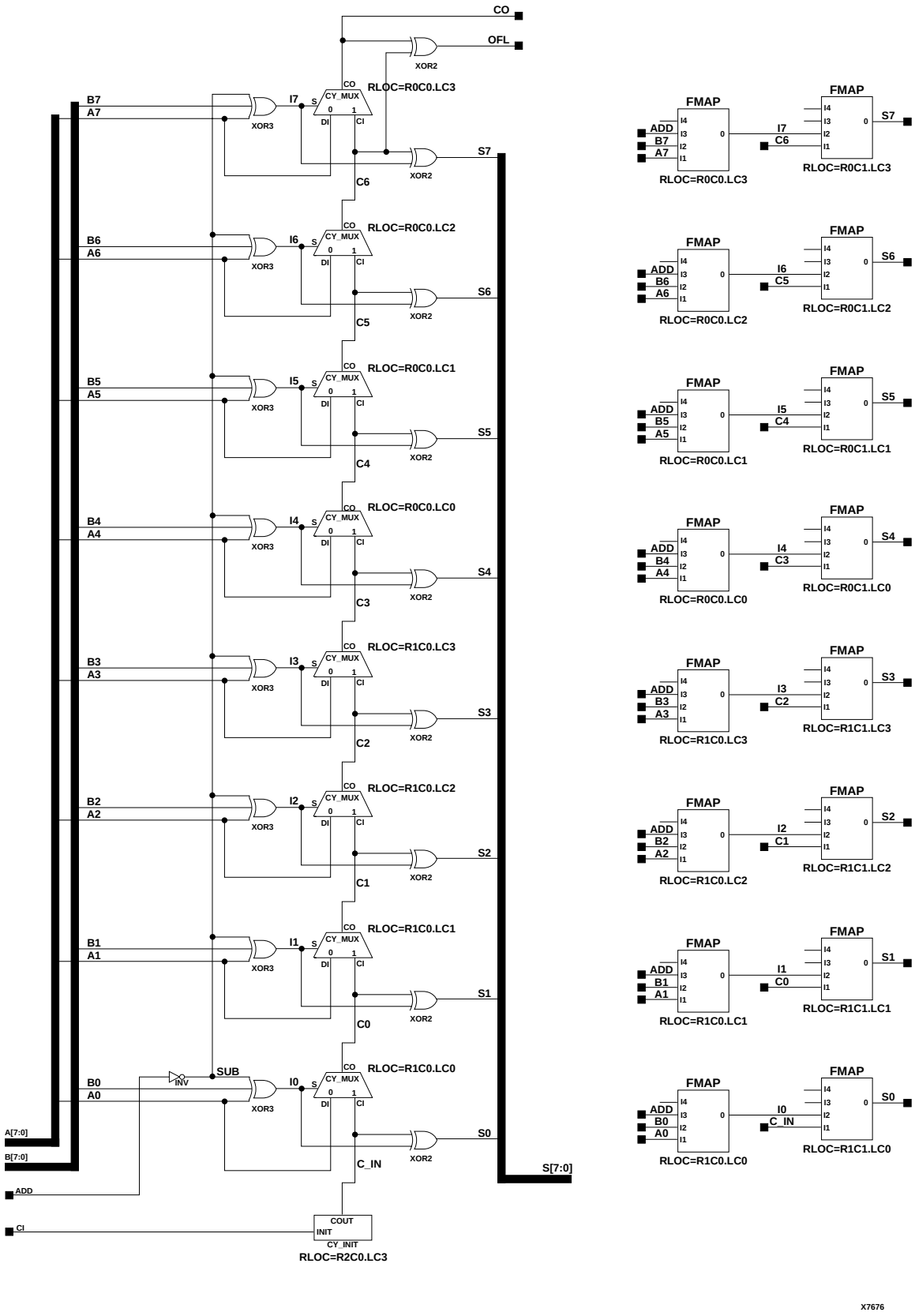


Figure 3-19 ADSU8 Implementation XC5200

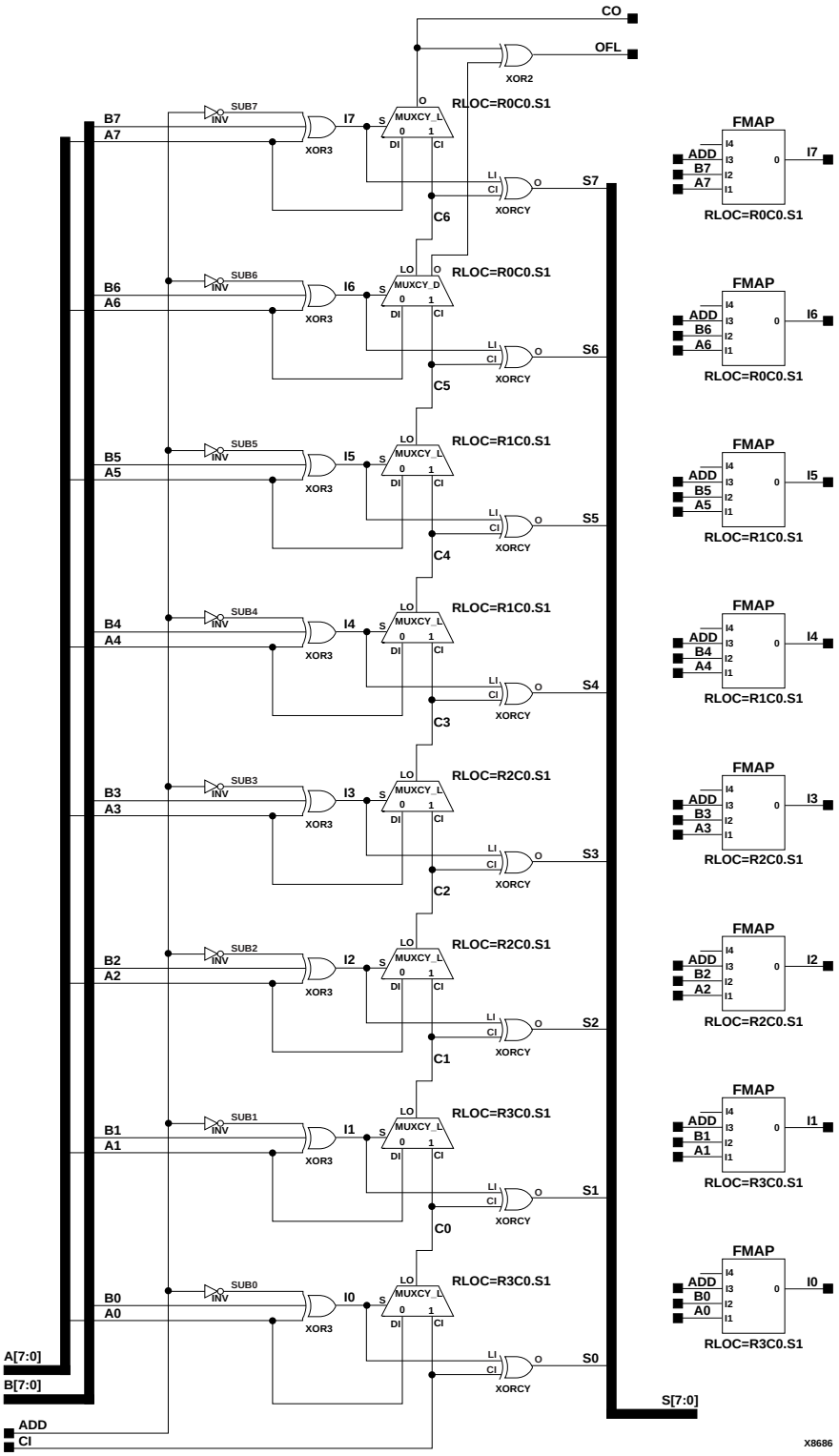


Figure 3-20 ADSU8 Implementation Spartan2, Virtex

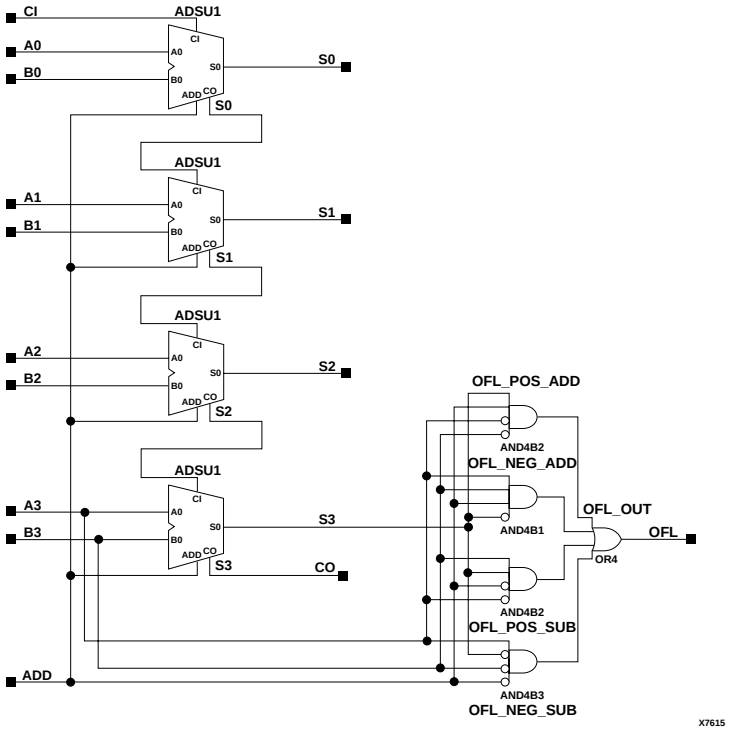


Figure 3-21 ADSU4 Implementation XC9000

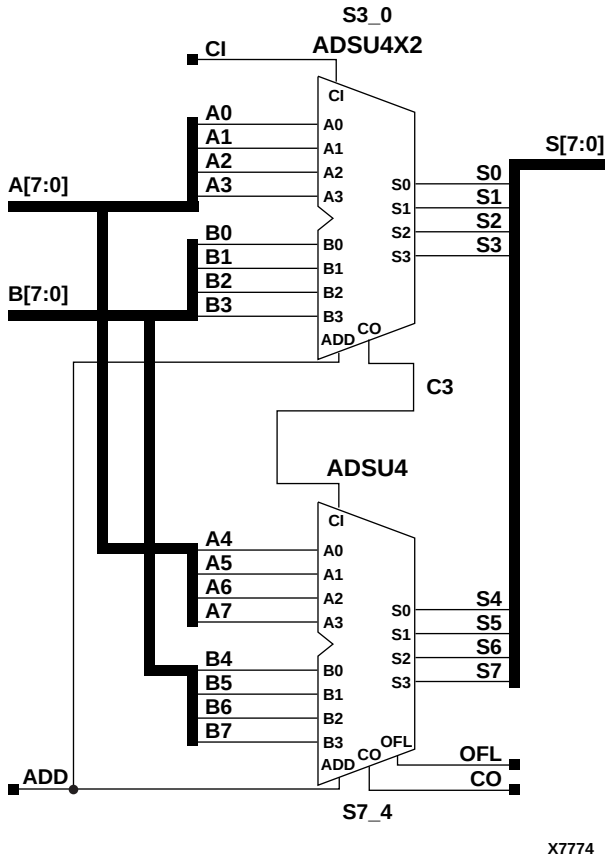


Figure 3-22 ADSU8 Implementation XC9000

AND2-9

2- to 9-Input AND Gates with Inverted and Non-Inverted Inputs

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
AND2, AND2B1, AND2B2, AND3, AND3B1, AND3B2, AND3B3, AND4, AND4B1, AND4B2, AND4B3, AND4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5, AND5B1, AND5B2, AND5B3, AND5B4, AND5B5	Primitive	Primitive	Primitive	Macro	Primitive	Primitive	Primitive	Primitive	Primitive
AND6, AND7, AND8, AND9	Macro	Macro	Macro	Macro	Primitive	Macro	Macro	Macro	Macro

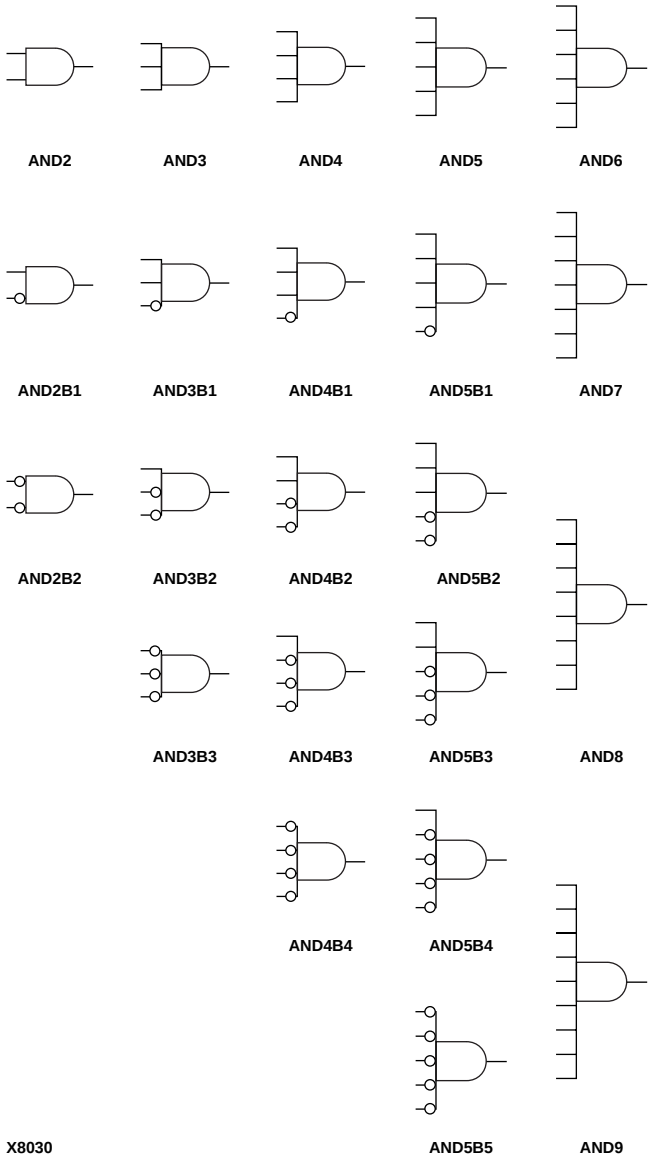


Figure 3-23 AND Gate Representations

The AND function is performed in the Configurable Logic Block (CLB) function generators for XC3000, XC4000E, XC4000X, XC5200, Spartan, and SpartanXL.

AND functions of up to five inputs are available in any combination of inverting and non-inverting inputs. AND functions of six to nine inputs are available with only non-inverting inputs. To make some or all inputs inverting, use external inverters. Because each input uses a CLB resource in FPGAs, replace functions with unused inputs with functions having the appropriate number of inputs.

Refer to “AND12, 16” for information on additional AND functions for the XC5200, Spartan2, and Virtex.

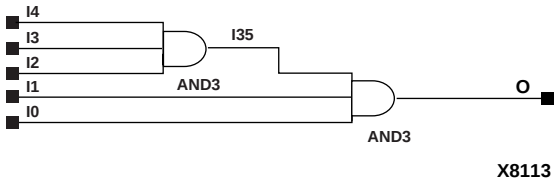


Figure 3-24 AND5 Implementation XC5200

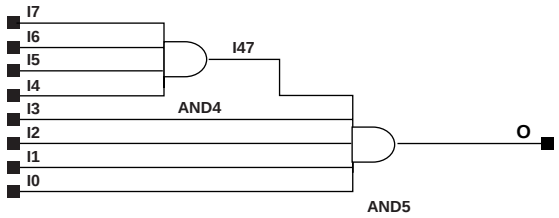


Figure 3-25 AND8 Implementation XC3000

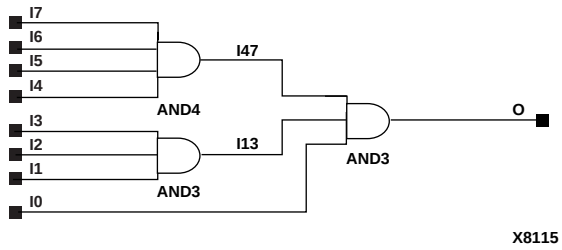


Figure 3-26 AND8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

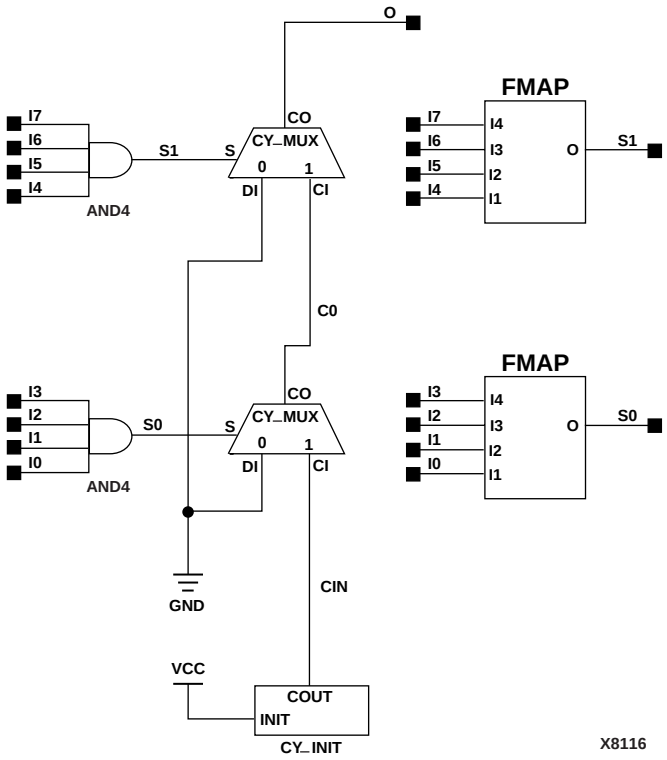


Figure 3-27 AND8 Implementation XC5200

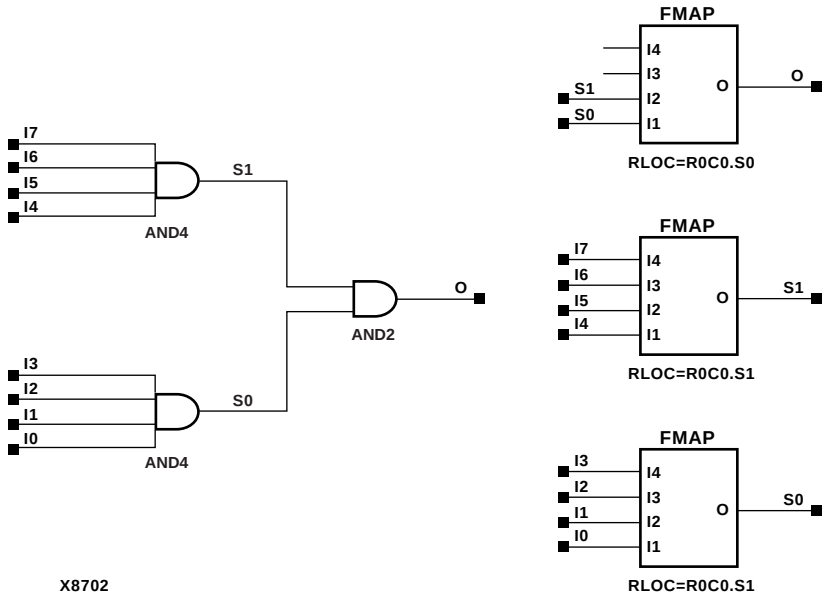
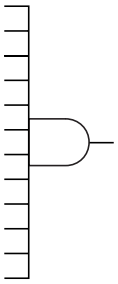


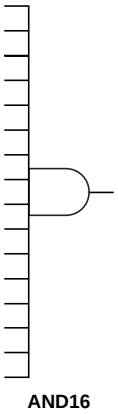
Figure 3-28 AND8 Implementation Spartan2, Virtex

AND12, 16

12- and 16-Input AND Gates with Non-Inverted Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	Macro	N / A	N / A	N / A	Macro	Macro





AND12 and AND16 functions are performed in the Configurable Logic Block (CLB) function generator.

The 12- and 16-input AND functions are available only with non-inverting inputs. To invert all of some inputs, use external inverters.

Refer to “AND2-9” for information on more AND functions.

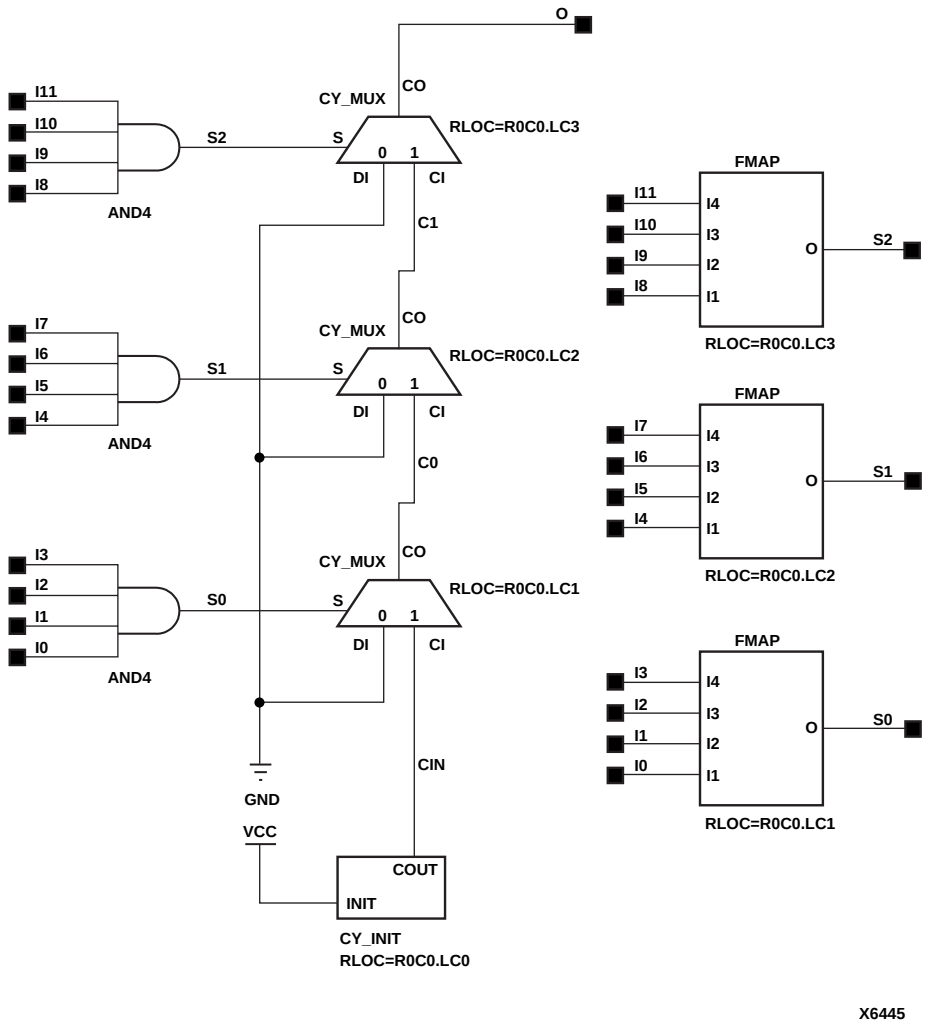


Figure 3-29 AND12 Implementation XC5200

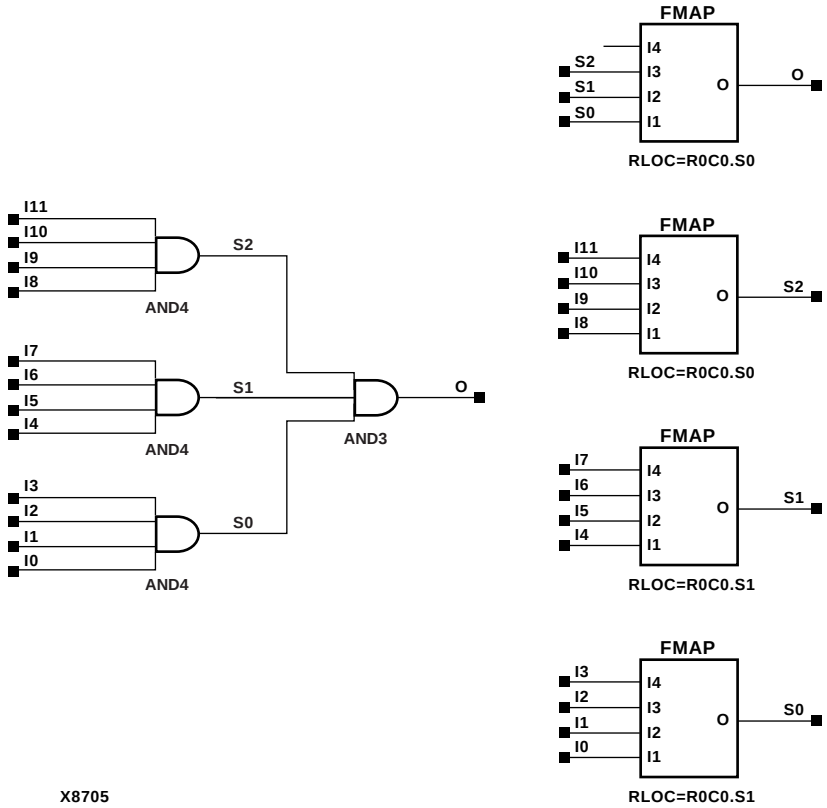


Figure 3-30 AND12 Implementation Spartan2, Virtex

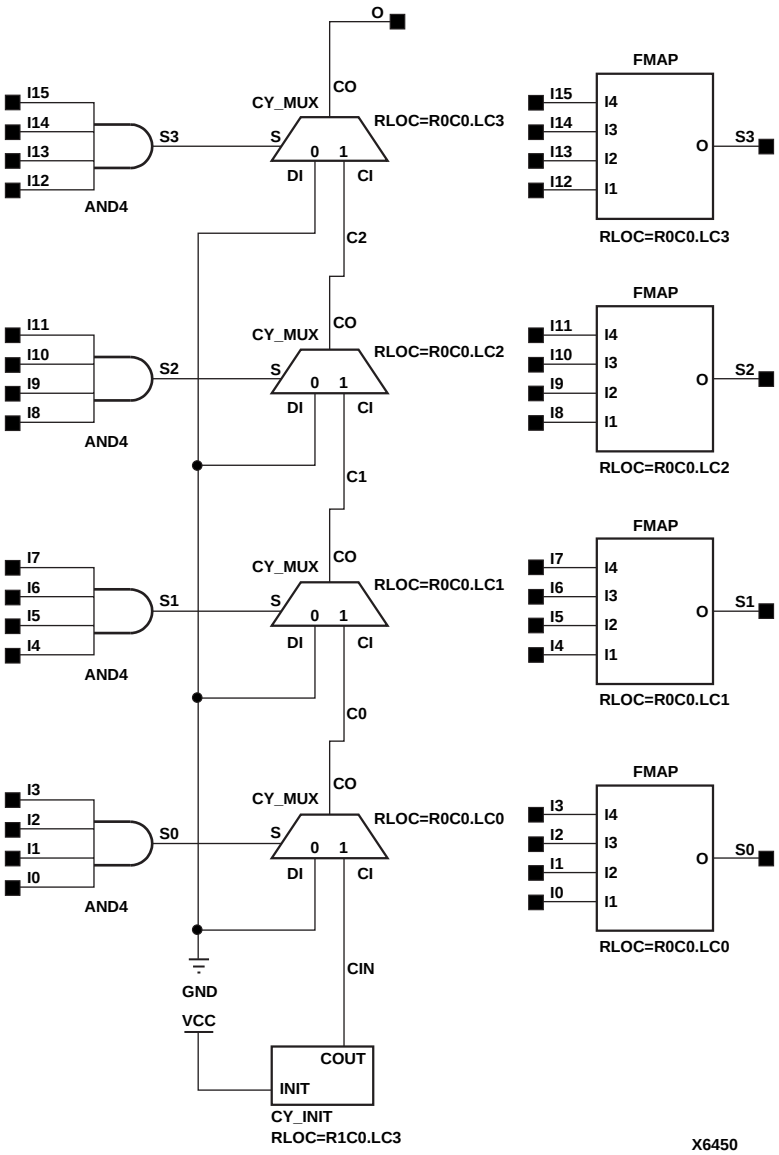


Figure 3-31 AND16 Implementation XC5200

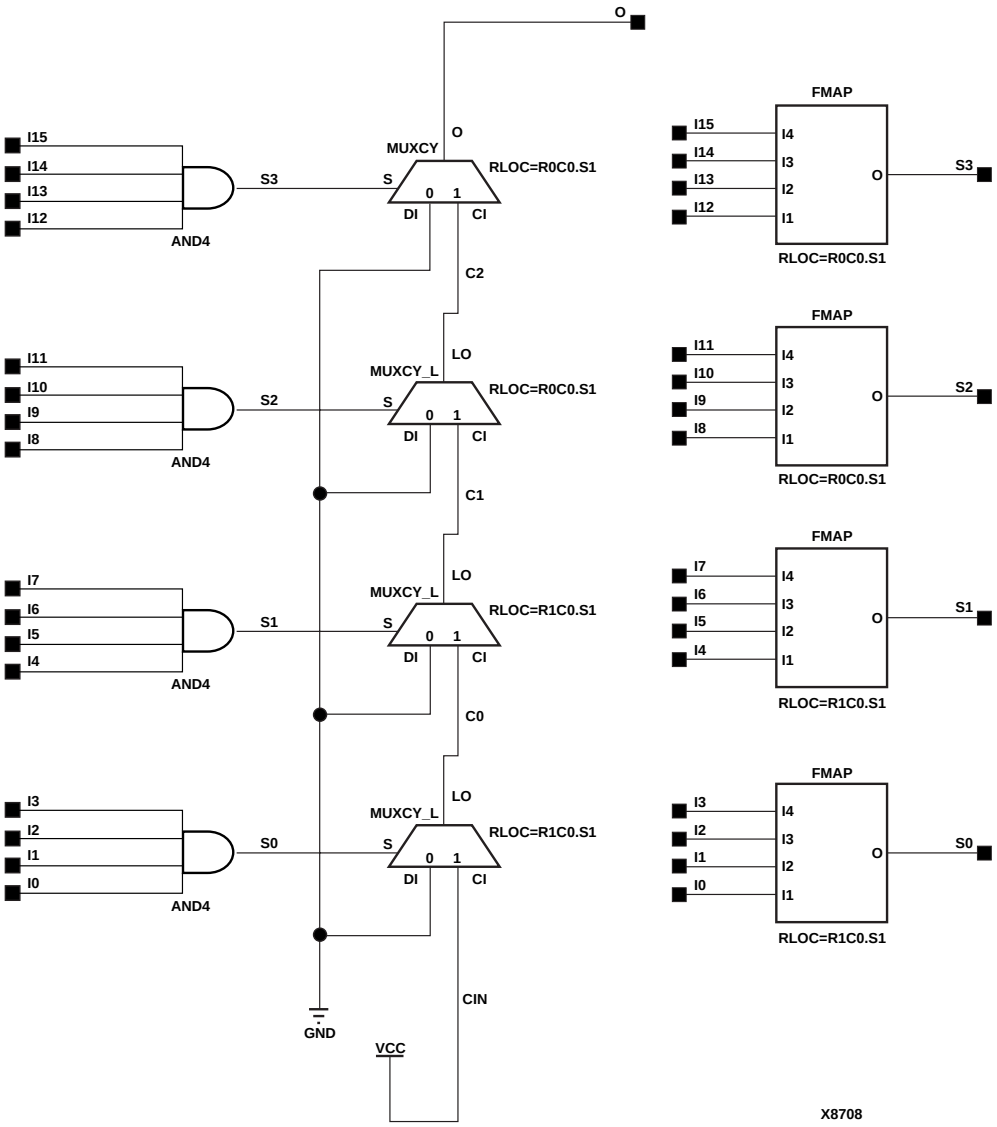
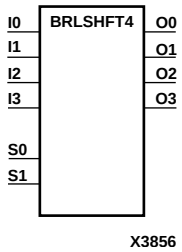


Figure 3-32 AND16 Implementation Spartan2, Virtex

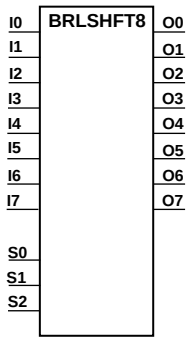
BRLSHFT4, 8

4-, 8-Bit Barrel Shifters

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



X3856



X3857

BRLSHFT4, a 4-bit barrel shifter, can rotate four inputs (I3 – I0) up to four places. The control inputs (S1 and S0) determine the number of positions, from one to four, that the data is rotated. The four outputs (O3 – O0) reflect the shifted data inputs.

BRLSHFT8, an 8-bit barrel shifter, can rotate the eight inputs (I7 – I0) up to eight places. The control inputs (S2 – S0) determine the number of positions, from one to eight, that the data is rotated. The eight outputs (O7 – O0) reflect the shifted data inputs.

Table 3-3 BRLSHFT4 Truth Table

Inputs						Outputs			
S1	S0	I0	I1	I2	I3	O0	O1	O2	O3
0	0	a	b	c	d	a	b	c	d
0	1	a	b	c	d	b	c	d	a
1	0	a	b	c	d	c	d	a	b
1	1	a	b	c	d	d	a	b	c

Table 3-4 BRLSHFT8 Truth Table

Inputs											Outputs							
S2	S1	S0	I0	I1	I2	I3	I4	I5	I6	I7	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h
0	0	1	a	b	c	d	e	f	g	h	b	c	d	e	f	g	h	a
0	1	0	a	b	c	d	e	f	g	h	c	d	e	f	g	h	a	b
0	1	1	a	b	c	d	e	f	g	h	d	e	f	g	h	a	b	c
1	0	0	a	b	c	d	e	f	g	h	e	f	g	h	a	b	c	d
1	0	1	a	b	c	d	e	f	g	h	f	g	h	a	b	c	d	e
1	1	0	a	b	c	d	e	f	g	h	g	h	a	b	c	d	e	f
1	1	1	a	b	c	d	e	f	g	h	h	a	b	c	d	e	f	g

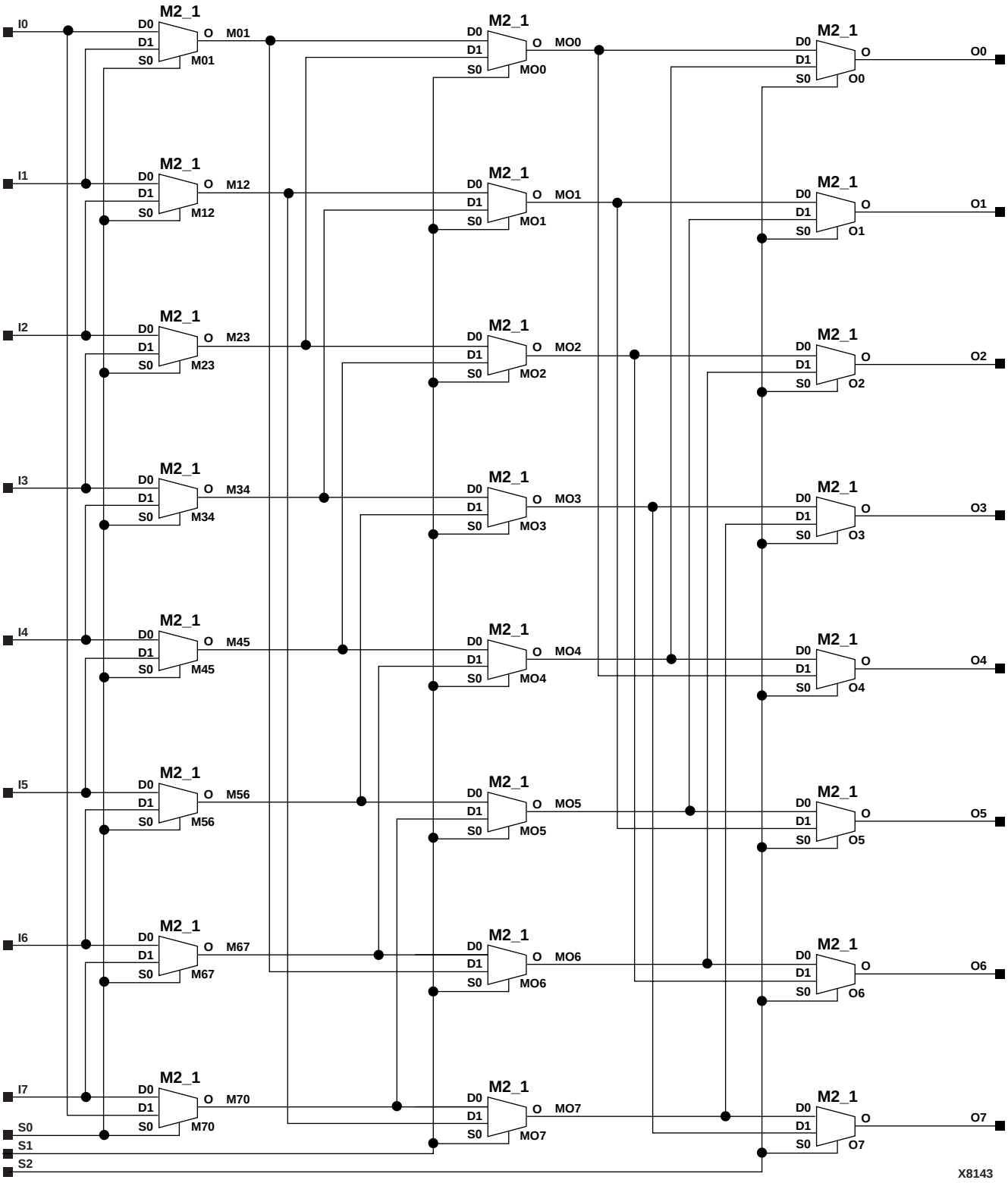
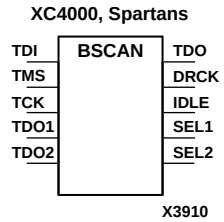


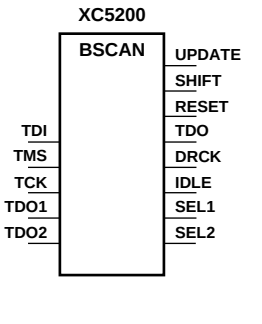
Figure 3-33 BRLSHFT8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

BSCAN

Boundary Scan Logic Control Circuit

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	N/A	N/A



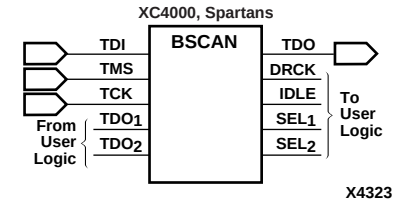


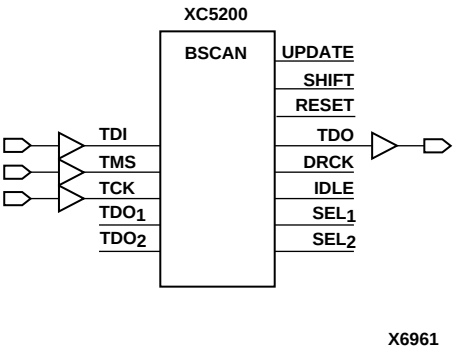
The BSCAN symbol indicates that boundary scan logic should be enabled after the programmable logic device (PLD) configuration is complete. It also provides optional access to some special features of the XC5200 boundary scan logic.

Note: For specific information on boundary scan for each architecture, refer to *The Programmable Logic Data Book*.

To indicate that boundary scan remains enabled after configuration, connect the BSCAN symbol to the TDI, TMS, TCK, and TDO pads. The other pins on BSCAN do not need to be connected, unless those special functions are needed. A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active. The TDO2 and SEL2 pins perform a similar function for the USER2 instruction. The DRCK output provides access to the data register clock (generated by the TAP controller). The IDLE output provides access to a version of the TCK input, which is only active while the TAP controller is in the Run-Test-Idle state. The RESET, UPDATE, and SHIFT pins of the XC5200 BSCAN symbol represent the decoding of the corresponding state of the boundary scan internal state machine. These functions are not available in the XC4000E, XC4000X, Spartan, and SpartanXL.

If boundary scan is used only before configuration is complete, do not include the BSCAN symbol in the design. The TDI, TMS, TCK, and TDO pins can be reserved for user functions.

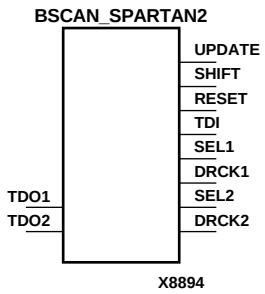




BSCAN_SPARTAN2

Spartan2 Boundary Scan Logic Control Circuit

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	N/A



The BSCAN_SPARTAN2 symbol is used to create internal boundary scan chains in a Spartan2 device. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Spartan2. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_SPARTAN2 symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

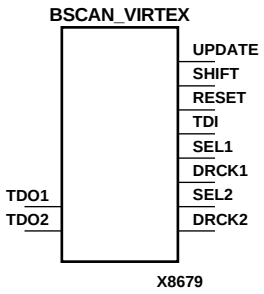
A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active.The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, and SHIFT pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note: For specific information on boundary scan for an architecture, refer to *The Programmable Logic Data Book*.

BSCAN_VIRTEX

Virtex Boundary Scan Logic Control Circuit

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive



The BSCAN_VIRTEX symbol is used to create internal boundary scan chains in a Virtex device. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Virtex. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_VIRTEX symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

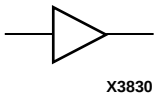
A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active.The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, and SHIFT pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note: For specific information on boundary scan for an architecture, refer to *The Programmable Logic Data Book*.

BUF

General-Purpose Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



BUF is a general purpose, non-inverting buffer.

In FPGA architectures, BUF is usually not necessary and is removed by the partitioning software (MAP). The BUF element can be preserved for reducing the delay on a high fan-out net, for example, by splitting the net and reducing capacitive loading. In this case, the buffer is preserved by attaching an X (explicit) attribute to both the input and output nets of the BUF.

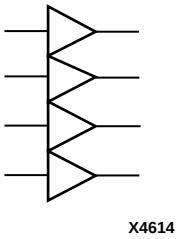
In CPLD architecture, BUF is usually removed, unless you inhibit optimization by applying the OPT=OFF attribute to the BUF symbol or by using the LOGIC_OPT=OFF global attribute.

BUF4, 8, 16

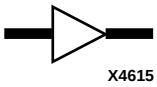
General-Purpose Buffers

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A

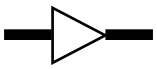
BUF4



BUF8



BUF16



BUF4, 8, 16 are general purpose, non-inverting buffers.

In CPLD architecture, BUF4, BUF8, and BUF16 are usually removed, unless you inhibit optimization by applying the OPT=OFF attribute to the BUF4, BUF8, or BUF16 symbol or by using the LOGIC_OPT=OFF global attribute.

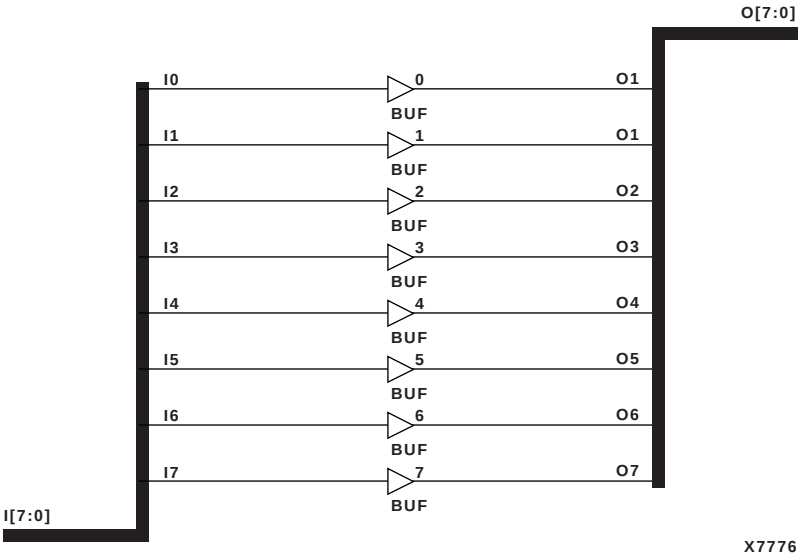
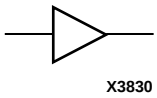


Figure 3-34 BUF8 Implementation XC9000

BUFCF

Fast Connect Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



BUFCF is a single fast connect buffer used to connect the outputs of the LUTs and some dedicated logic directly to the input of another LUT. Using this buffer implies CLB packing. No more than four LUTs may be connected together as a group.

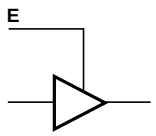
BUFE, 4, 8, 16

Internal 3-State Buffers with Active High Enable

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
BUFE	Macro	Macro	Macro	Macro	Primitive*	Macro	Macro	Primitive	Primitive
BUFE4, BUFE8, BUFE16	Macro	Macro	Macro	Macro	Macro*	Macro	Macro	Macro	Macro

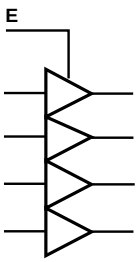
* not supported for XC9500XL and XC9500XV devices

BUFE



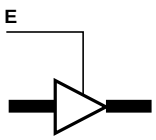
X3790

BUFE4



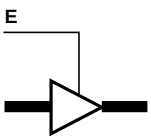
X3797

BUFE8



X3809

BUFE16



X3821

BUFE, BUFE4, BUFE8, and BUFE16 are single or multiple tristate buffers with inputs I, I3 – I0, I7 – I0, and I15 – I0, respectively; outputs O, O3 – O0, O7 – O0, and O15 – O0, respectively; and active-High output enable (E). When E is High, data on the inputs of the buffers is transferred to the corresponding outputs. When E is Low, the output is high impedance (Z state or Off). The outputs of the buffers are connected to horizontal longlines in FPGA architectures.

The outputs of separate BUFE symbols can be tied together to form a bus or a multiplexer. Make sure that only one E is High at any one time. If none of the E inputs is active-High, a "weak-keeper" circuit (FPGAs) keeps the output bus from floating but does not guarantee that the bus remains at the last value driven onto it.

In XC3000, XC4000E, XC4000X, Spartan, and SpartanXL, the E signal in BUFE macros is implemented by using a BUFT with an inverter on the active-Low enable (T) pin. This inverter can add an extra level of logic to the data path. Pull-up resistors can be used to establish a High logic level if all BUFE elements are Off.

In the XC5200 architecture, pull-ups cannot be used in conjunction with BUFT or BUFE macros because there are no pull-ups available at the ends of the horizontal longlines.

For XC9500 devices, BUFE output nets assume the High logic level when all connected BUFE/BUFT buffers are disabled. On-chip 3-state multiplexing is not available in XC9500XL devices.

For Virtex and Spartan2, BUFE elements need a PULLUP element connected to their output. NGDBuild inserts a PULLUP element if one is not connected.

Inputs		Outputs
E	I	O
0	X	Z
1	1	1
1	0	0

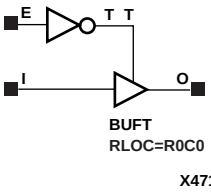


Figure 3-35 BUFE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

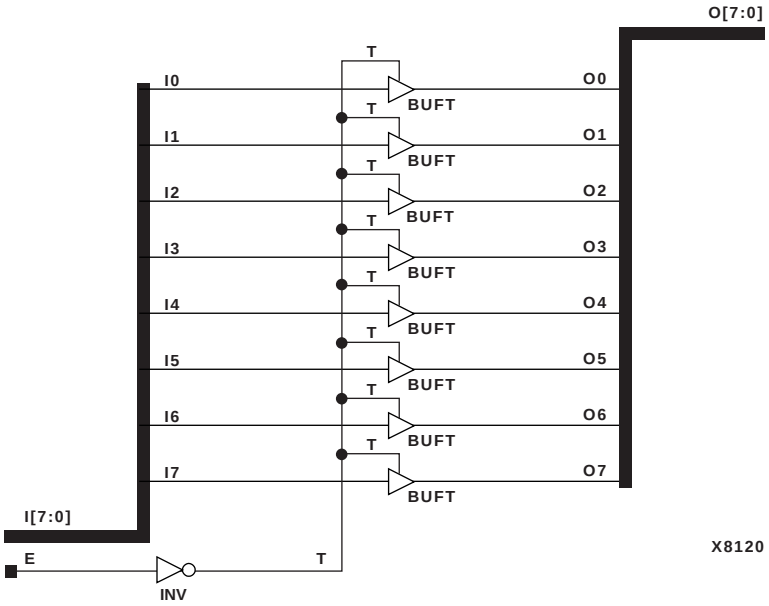


Figure 3-36 BUFE8 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

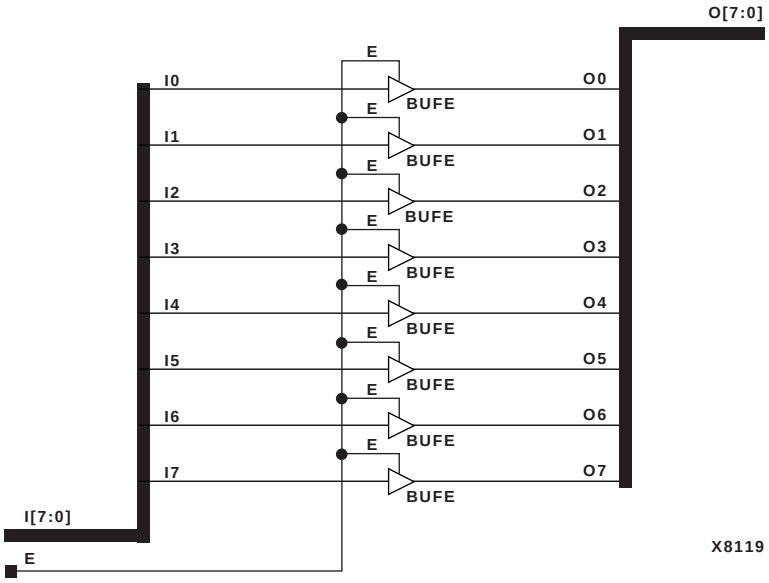
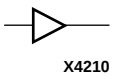


Figure 3-37 BUFE8 Implementation XC9000, Spartan2, Virtex

BUFFCLK

Global Fast Clock Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A	N/A



BUFFCLK (FastCLK buffer) provides the fastest way to bring a clock into the XC4000X device. Four of these buffers are present on those devices — two on the left edge of the die and two on the right edge.

Using BUFFCLK, you can create a very fast pin-to-pin path by driving the F input of the CLB function generator with BUFFCLK output.

You can use BUFFCLK to minimize the setup time of input devices if positive hold time is acceptable. Use BUFFCLK to clock the Fast Capture latch and a slower clock buffer to capture the standard IOB flip-flop or latch. Either the Global Early buffer (BUFGE) or the Global Low-Skew buffer (BUFGLS) can be used for the second storage element (the one used should be the same clock as the internal related logic).

You can also use BUFFCLK to provide a fast Clock-to-Out on device output pins.

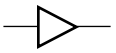
These buffers can access IOBs on one half of the die edge only. They can each drive two of the four vertical lines accessing the IOBs on the left edge of the device or two of the eight vertical lines accessing the IOBs on the right edge of the device. They can only access the CLB array through single and double-length lines.

BUFFCLKs must be driven by the semi-dedicated IOBs. They are not accessible from internal nets.

BUFG

Global Clock Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



BUFG, an architecture-independent global buffer, distributes high fan-out clock signals throughout a PLD device. The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target PLD device. If you want to use a specific type of buffer, instantiate it manually.

To use a BUFG in a schematic, connect the input of the BUFG symbol to the clock source. Depending on the target PLD family, the clock source can be an external PAD symbol, an IBUF symbol, or internal logic. For a negative-edge clock input, insert an INV (inverter) symbol between the BUFG output and the clock input. The inversion is implemented at the Configurable Logic Block (CLB) or Input Output Block (IOB) clock pin.

For an XC3000 design, you can use a maximum of two BUFG symbols (assuming that no specific GCLK or ACLK buffer is specified). For XC3000 designs, MAP always selects an ACLK, then a GCLK.

For XC4000, Spartan, or SpartanXL designs, you can use a maximum of eight BUFG symbols (assuming that no specific BUFGP or BUFGS buffers are specified). For XC4000, Spartan, or SpartanXL designs, MAP always selects a BUFGS before a BUFGP.

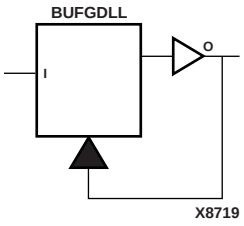
For XC9000 designs, consult the device data sheet for the number of available global pins. For XC9000 designs, BUFG is always implemented using an IOB. Connect the input of BUFG to an IPAD or an IOPAD that represents an external signal source. Each BUFG can drive any number of register clocks in a design. For XC9000 designs, the output of a BUFG may also be used as an ordinary input signal to other logic elsewhere in the design.

In Virtex and Spartan2, the BUFG cannot be driven directly from a pad. It can be driven from an IBUFG to indicate to use the dedicated pin (GCLKIOB pin) or from an internal driver to create an internal clock. BUFG can also be driven with an IBUF to represent an externally driven clock that does not use the dedicated pin.

BUFGDLL

Clock Delay Locked Loop Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



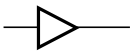
BUFGDLL is a special purpose clock delay locked loop buffer for clock skew management. It is provided as a user convenience for the most frequently used configuration of elements for clock skew management. Internally, it consists of an IBUFG driving the CLKIN pin of a CLKDLL followed by a BUFG that is driven by the CLK0 pin of the CLKDLL. Because BUFGDLL already contains an input buffer (IBUFG), it can only be driven by a top-level port (IPAD).

Any DUTY_CYCLE_CORRECTION attribute on a BUFGDLL applies to the underlying CLKDLL symbol.

BUFGE

Global Low Early Clock Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A	N/A



X4210

Eight Global Early buffers (BUFGE), two on each corner of the device, provide an earlier clock access than the potentially heavily loaded Global Low-Skew buffers (BUFGLS).

BUFGE can facilitate the fast capture of device inputs using the Fast Capture latches ILFFX and ILFLX. For fast capture, take a single clock signal and route it through both a BUFGE and a BUFGLS. Use the BUFGE to clock the fast capture latch and the BUFGLS to clock the normal input flip-flop or latch.

You can also use BUFGE to provide a fast Clock-to-Out on device output pins.

BUFGLS

Global Low Skew Clock Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A

X4210

Each corner of the XC4000X or SpartanXL device has two Global Low-Skew buffers (BUFGLS). Any of the eight buffers can drive any of the eight vertical global lines in a column of CLBs. In addition, any of the buffers can drive any of the four vertical lines accessing the IOBs on the left edge of the device and any of the eight vertical lines accessing the IOBs on the right edge of the device.

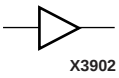
IOBs at the top and bottom edges of the device are accessed through the vertical global lines in the CLB array. Any global low-skew buffer can, therefore, access every IOB and CLB in the device.

The global low-skew buffers can be driven by either semi-dedicated pads or internal logic.

BUFGP

Primary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Primitive	N / A	Macro	N / A	Primitive	N / A	Primitive	Primitive



BUFGP, a primary global buffer, is used to distribute high fan-out clock or control signals throughout PLD devices. In Virtex and Spartan2, BUFGP is equivalent to an IBUFG driving a BUFG. In CPLD designs, BUFGP is treated like BUFG. A BUFGP provides direct access to Configurable Logic Block (CLB) and Input Output Block (IOB) clock pins and limited access to other CLB inputs. Four BUFGPs are available on each XC4000E and Spartan device, one in each corner. The input to a BUFGP comes only from a dedicated IOB.

Alongside each column of CLBs in an XC4000E or Spartan device are four global vertical lines, which are in addition to the standard vertical longlines. Each one of the four global vertical lines can drive the CLB clock (K) pin directly. In addition, one of the four lines can drive the F3 pin, a second line can drive the G1 pin, a third can drive the C3 pin, and a fourth can drive the C1 pin. Each of the four BUFGPs drives one of these global vertical lines. These same vertical lines are also used for the secondary global buffers (refer to the “BUFGS” section for more information).

Because of its structure, a BUFGP can always access a clock pin directly. However, it can access only one of the F3, G1, C3, or C1 pins, depending on the corner in which the BUFGP is placed. When the required pin cannot be accessed directly from the vertical line, PAR feeds the signal through another CLB and uses general purpose routing to access the load pin.

To use a BUFGP in a schematic, connect the input of the BUFGP element directly to the PAD symbol. Do not use any IBUFs, because the signal comes directly from a dedicated IOB. The output of the BUFGP is then used throughout the schematic. For a negative-edge clock, insert an INV (inverter) element between the output of the BUFGP and the clock input. This inversion is performed inside each CLB or IOB.

A Virtex or Spartan2 BUFGP must be sourced by an external signal. Other BUFGPs can be sourced by an internal signal, but PAR must use the dedicated IOB to drive the BUFGP, which means that the IOB is not available for use by other signals. If possible, use a BUFGS instead, because it can be sourced internally without using an IOB.

The dedicated inputs for BUFGPs are identified by the names PGCK1 through PGCK4 in pinouts in XC4000E and Spartan. The package pin that drives the BUFGP depends on which corner the BUFGP is placed by PAR.

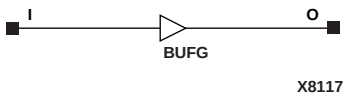
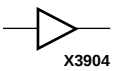


Figure 3-38 BUFGP Implementation XC5200

BUFGS

Secondary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Primitive	N / A	Macro	N / A	Primitive	N / A	N / A	N / A



BUFGS, a secondary global buffer, distributes high fan-out clock or control signals throughout a PLD device. In CPLD designs, BUFGS is treated like BUFG. BUFGS provides direct access to Configurable Logic Block (CLB) clock pins and limited access to other CLB inputs. Four BUFGSs are available on each XC4000E and Spartan device, one in each corner. The input to a BUFGS comes either from a dedicated Input Output Block (IOB) or from an internal signal.

Alongside each column of CLBs in an XC4000E or Spartan device are four global vertical lines, which are in addition to the standard vertical longlines. Each one of the four global vertical lines can drive the CLB clock (K) pin directly. In addition, one of the four lines can drive the F3 pin, a second line can drive the G1 pin, a third can drive the C3 pin, and a fourth can drive the C1 pin. Each of the four BUFGSs can drive any of these global vertical lines and are also used as the primary global buffers (refer also to the “BUFGP” section for more information).

Because of its structure, a BUFGS can always access a clock pin directly. Because the BUFGS is more flexible than the BUFGP, it can use additional global vertical lines to access the F3, G1, C3, and C1 pins but requires multiple vertical lines in the same column. If the vertical lines in a given column are already used for BUFGPs or another BUFGS, PAR might have to feed signals through other CLBs to reach the load pins.

To use a BUFGS in a schematic, connect the input of the BUFGS element either directly to the PAD symbol (for an external input) or to an internally sourced net. For an external signal, do not use any IBUFs, because the signal comes directly from the dedicated IOB. The output of the BUFGS is then used throughout the schematic. For a negative-edge clock, insert an INV (inverter) element between the output of the BUFGS and the clock input. This inversion is performed inside each CLB or IOB.

The dedicated inputs for BUFGSs are identified by the names SGCK1 through SGCK4 in pinouts in XC4000E and Spartan. The package pin that drives the BUFGS depends on which corner the BUFGS is placed by PAR.

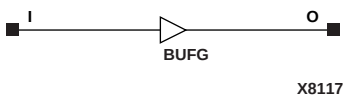


Figure 3-39 BUFGS Implementation XC5200

BUFGSR

Global Set/Reset Input Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A


X3831

BUFGSR, an XC9000-specific global buffer, distributes global set/reset signals throughout selected flip-flops of an XC9000 device. Global Set/Reset (GSR) control pins are available on XC9000 devices; consult device data sheets for availability.

BUFGSR always acts as an input buffer. To use it in a schematic, connect the input of the BUFGSR symbol to an IPAD or an IOPAD representing the GSR signal source. GSR signals generated on-chip must be passed through an OBUF-type buffer before they are connected to BUFGSR.

For global set/reset control, the output of BUFGSR normally connects to the CLR or PRE input of a flip-flop symbol, like FDCP, or any registered symbol with asynchronous clear or preset. The global set/reset control signal may pass through an inverter to perform an active-low set/reset. The output of BUFGSR may also be used as an ordinary input signal to other logic elsewhere in the design. Each BUFGSR can control any number of flip-flops in a design.

BUFGTS

Global Three-State Input Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A

BUFGTS, an XC9000-specific global buffer, distributes global output-enable signals throughout the output pad drivers of an XC9000 device. Global Three-State (GTS) control pins are available on XC9000 devices; consult device data sheets for availability.

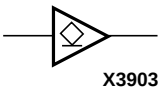
BUFGTS always acts as an input buffer. To use it in a schematic, connect the input of the BUFGTS symbol to an IPAD or an IOPAD representing the GTS signal source. GTS signals generated on-chip must be passed through an OBUF-type buffer before they are connected to BUFGTS.

For global 3-state control, the output of BUFGTS normally connects to the E input of a 3-state output buffer symbol, OBUFE. The global 3-state control signal may pass through an inverter or control an OBUFT symbol to perform an active-low output-enable. The same 3-state control signal may even be used both inverted and non-inverted to enable alternate groups of device outputs. The output of BUFGTS may also be used as an ordinary input signal to other logic elsewhere in the design. Each BUFGTS can control any number of output buffers in a design.

BUFOD

Open-Drain Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



BUFOD is a buffer with input (I) and open-drain output (O). When the input is Low, the output is Low. When the input is High, the output is off. To establish an output High level, a pull-up resistor is tied to output O. One pull-up resistor uses the least power; two pull-up resistors achieve the fastest Low-to-High speed.

To indicate two pull-up resistors, append a DOUBLE parameter to the pull-up symbol attached to the output (O) node. Refer to the appropriate CAE tool interface user guide for details.

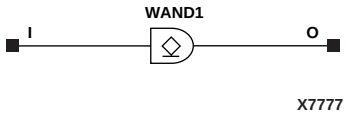


Figure 3-40 BUFOD Implementation XC4000E, XC4000X

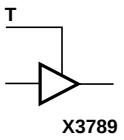
BUFT, 4, 8, 16

Internal 3-State Buffers with Active-Low Enable

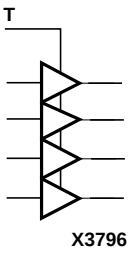
Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
BUFT	Primitive	Primitive	Primitive	Primitive	Primitive*	Primitive	Primitive	Primitive	Primitive
BUFT4, BUFT8, BUFT16	Macro	Macro	Macro	Macro	Macro*	Macro	Macro	Macro	Macro

* not supported for XC9500XL and XC9500XV devices

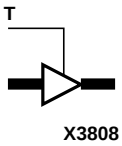
BUFT



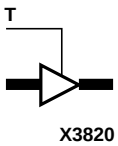
BUFT4



BUFT8



BUFT16



BUFT, BUFT4, BUFT8, and BUFT16 are single or multiple 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15 – 10, respectively; outputs O, O3 – O0, O7 – O0, and O15 – O0, respectively; and active-Low output enable (T). When T is Low, data on the inputs of the buffers is transferred to the corresponding outputs. When T is High, the output is high impedance (Z state or off). The outputs of the buffers are connected to horizontal longlines in FPGA architectures.

The outputs of separate BUFT symbols can be tied together to form a bus or a multiplexer. Make sure that only one T is Low at one time. If none of the T inputs is active (Low), a "weak-keeper" circuit (FPGAs) prevents the output bus from floating but does not guarantee that the bus remains at the last value driven onto it.

Pull-up resistors can be used to establish a High logic level if all BUFT elements are off in XC3000, XC4000, Spartan, and SpartanXL.

In the XC5200 architecture, pull-ups cannot be used in conjunction with BUFT or BUFE macros because there are no pull-ups available at the ends of the horizontal longlines.

For XC9500 devices, BUFT output nets assume the High logic level when all connected BUFE/BUFT buffers are disabled. On-chip 3-state multiplexing is not available in XC9500XL devices.

For Virtex and Spartan2, when all BUFTs on a net are disabled, the net is High. For correct simulation of this effect, a PULLUP element must be connected to the net. NGDBuild inserts a PULLUP element if one is not connected so that back-annotation simulation reflects the true state of the Virtex or Spartan2 chip.

Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0

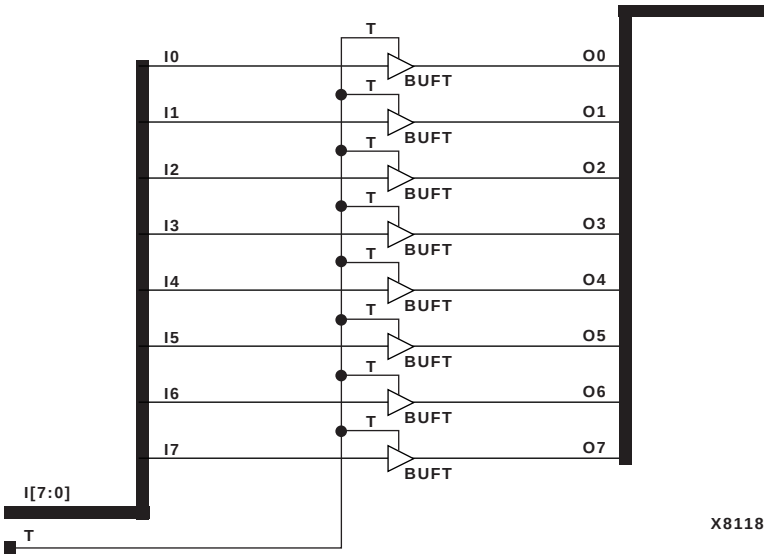
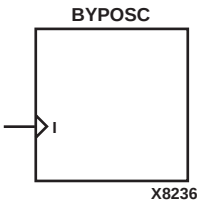


Figure 3-41 BUFT8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

BYPOSC

Bypass Oscillator

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



BYPOSC provides for definition of a user clock for the charge pump via its I pin. When the BYPOSC symbol is not used or its I pin is not connected, the charge pump uses an internal clock.

Chapter 4

Design Elements (CAPTURE_SPARTAN2 to DECODE64)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

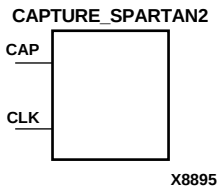
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

CAPTURE_SPARTAN2

Spartan2 Register State Capture for Bitstream Readback

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	N/A



CAPTURE_SPARTAN2 provides user control over when to capture register (flip-flop and latch) information for readback. Spartan2 devices provide the readback function through dedicated configuration port instructions, instead of with a READBACK component as in other FPGA architectures. The CAPTURE_SPARTAN2 symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Note: Spartan2 only allows for capturing register (flip-flop and latch) states. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted High CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. The Low-to-High clock transition triggers the capture clock (CLK) which clocks out the readback data.

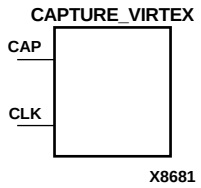
By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, you can add the ONESHOT attribute to CAPTURE_SPARTAN2. Refer to the “ONESHOT” section of the “Attributes, Constraints, and Carry Logic” chapter for information on the ONESHOT attribute.

For details on the Spartan2 readback functions, refer to the Xilinx web site, http://support.xilinx.com.

CAPTURE_VIRTEX

Virtex Register State Capture for Bitstream Readback

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive



CAPTURE_VIRTEX provides user control over when to capture register (flip-flop and latch) information for readback. Virtex devices provide the readback function through dedicated configuration port instructions, instead of with a READBACK component as in other FPGA architectures. The CAPTURE_VIRTEX symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Note: Virtex only allows for capturing register (flip-flop and latch) states. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted High CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. The Low-to-High clock transition triggers the capture clock (CLK) which clocks out the readback data.

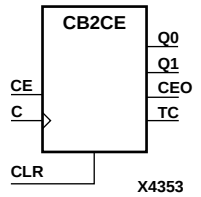
By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, you can add the ONESHOT attribute to CAPTURE_VIRTEX. Refer to the “ONESHOT” section of the “Attributes, Constraints, and Carry Logic” chapter for information on the ONESHOT attribute.

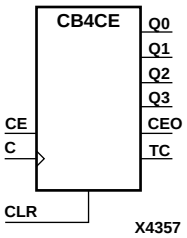
For details on the Virtex readback functions, refer to the Virtex datasheets on the Xilinx web site, http://support.xilinx.com.

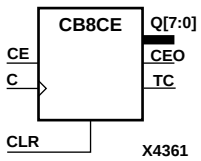
CB2CE, CB4CE, CB8CE, CB16CE

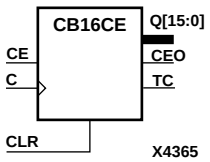
2-, 4-, 8-,16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









CB2CE, CB4CE, CB8CE, and CB16CE are, respectively, 2-, 4-, 8-, and 16-bit (stage), asynchronous, clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs		
CLR	CE	C	Qz – Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

z= 1 for CB2CE; z = 3 for CB4CE; z = 7 for CB8CE; z = 15 for CB16CE
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0
CEO = TC • CE

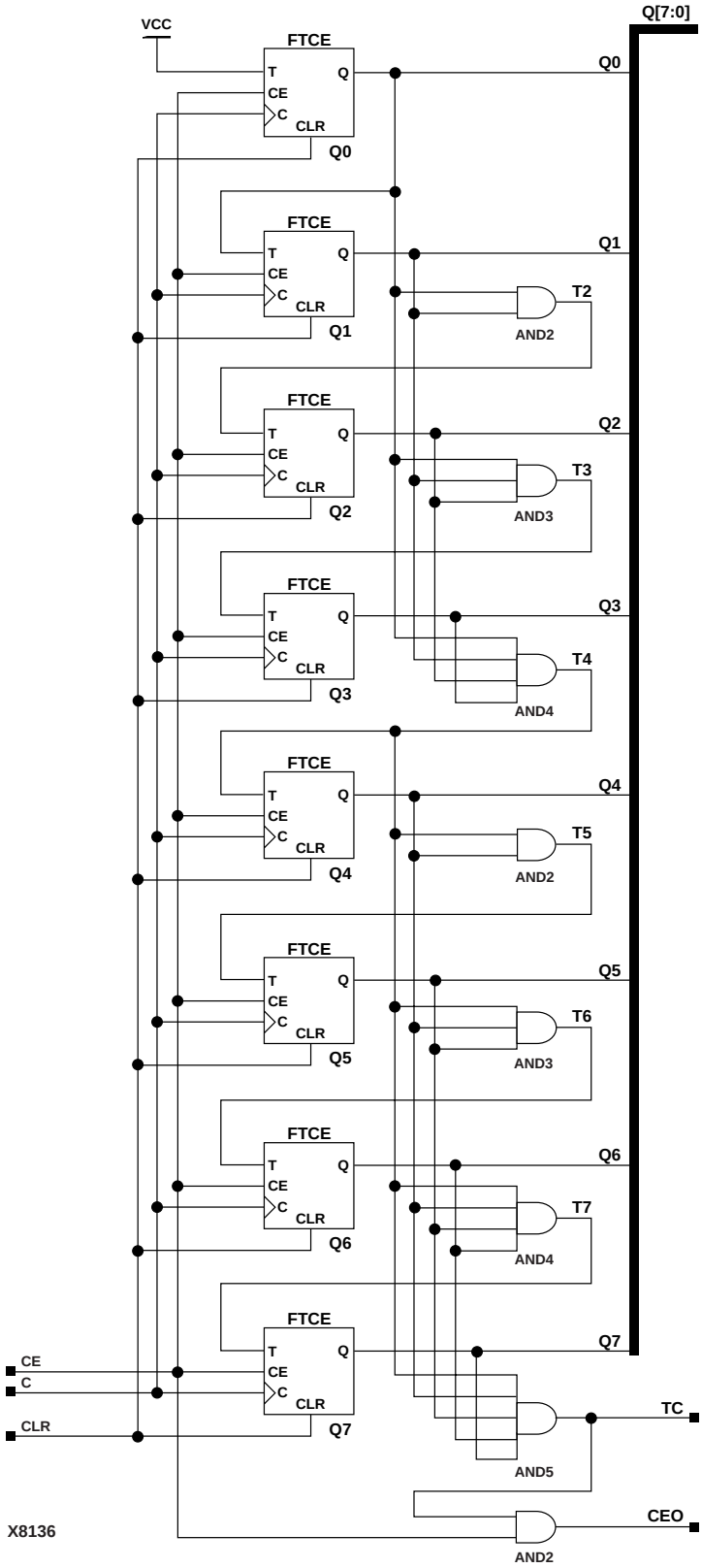


Figure 4-1 CB8CE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

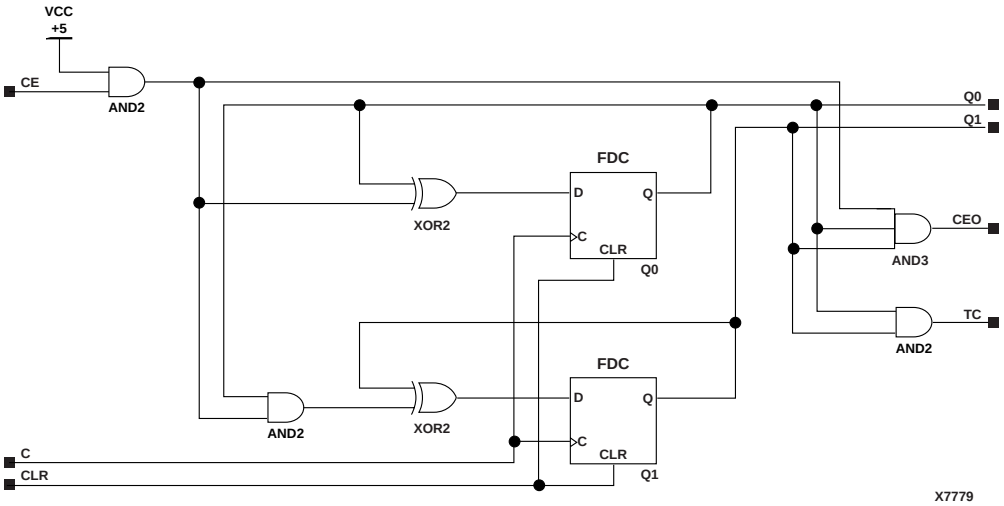


Figure 4-2 CB2CE Implementation XC9000

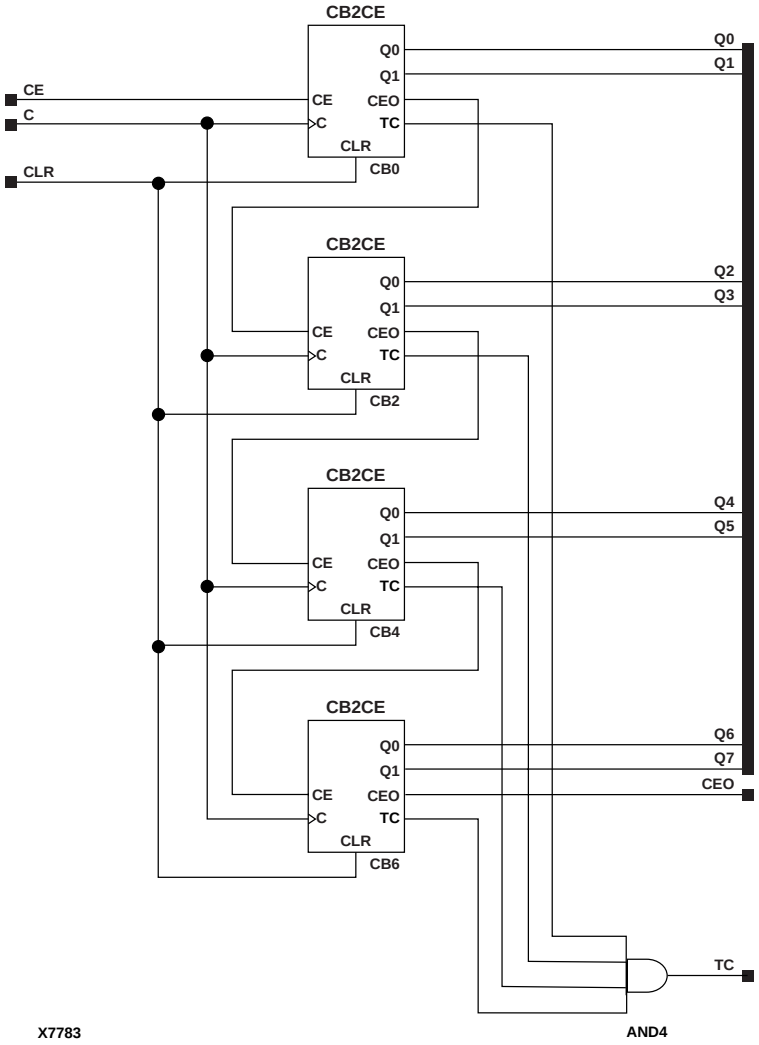
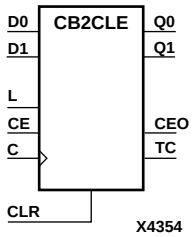


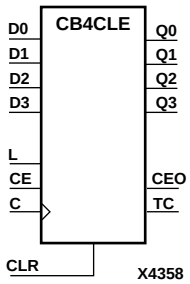
Figure 4-3 CB8CE Implementation XC9000

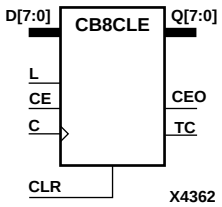
CB2CLE, CB4CLE, CB8CLE, CB16CLE

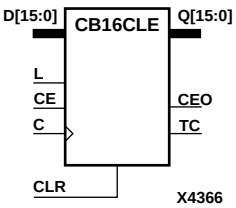
2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









CB2CLE, CB4CLE, CB8CLE, and CB16CLE are, respectively, 2-, 4-, 8-, and 16-bit (stage) synchronously loadable, asynchronously clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs		
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

z= 1 for CB2CLE; z = 3 for CB4CLE; z = 7 for CB8CLE; z = 15 for CB16CLE
dn = state of referenced input (Dn) one setup time prior to active clock transition.
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0
CEO = TC • CE

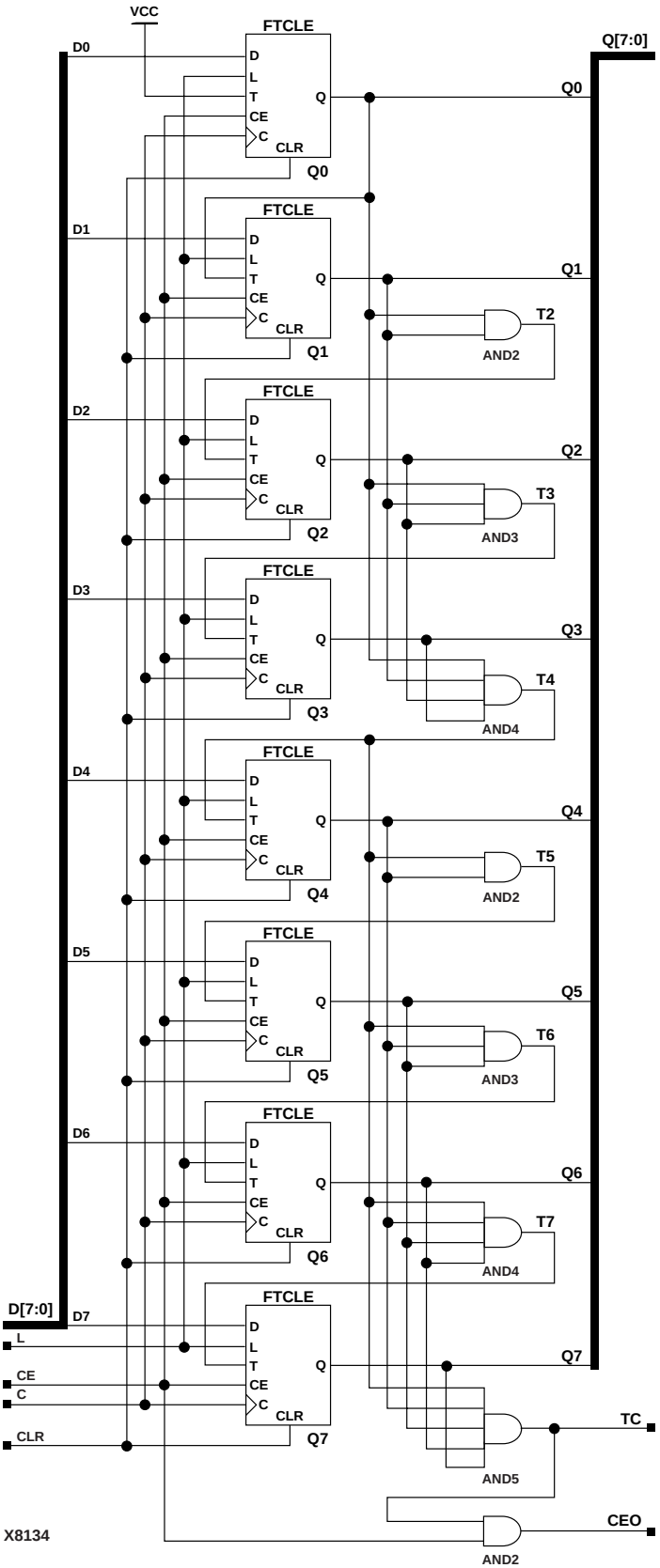


Figure 4-4 CB8CLE Implementation XC3000

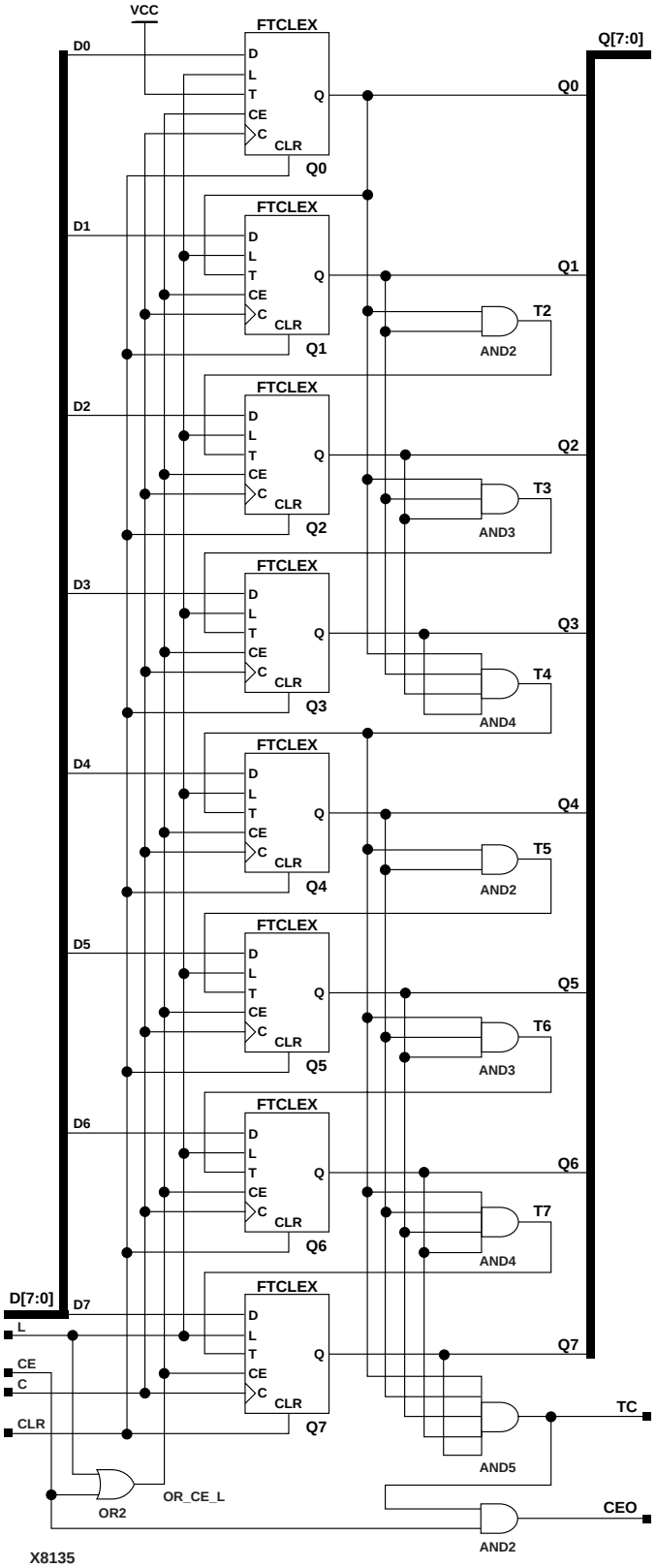


Figure 4-5 CB8CLE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

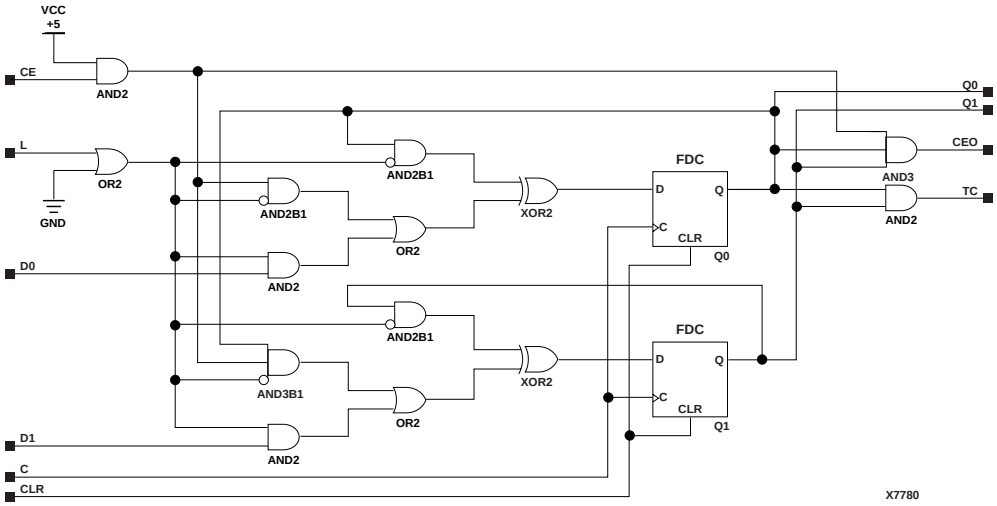


Figure 4-6 CB2CLE Implementation XC9000

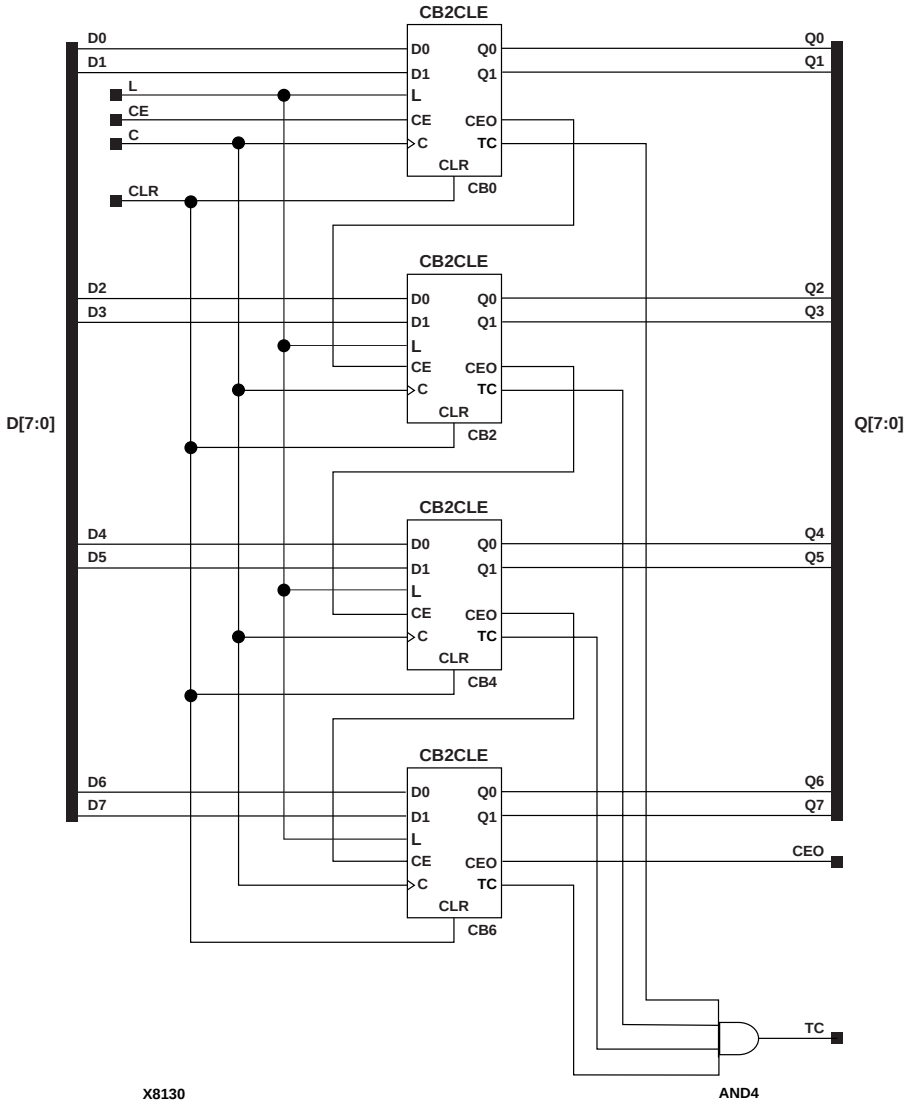
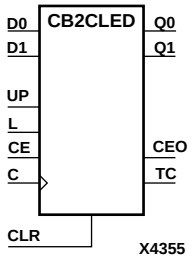


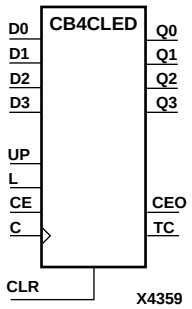
Figure 4-7 CB8CLE Implementation XC9000

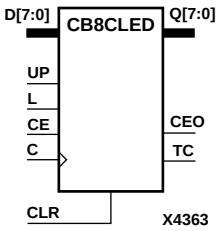
CB2CLED, CB4CLED, CB8CLED, CB16CLED

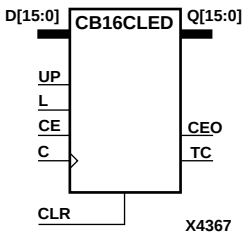
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









CB2CLED, CB4CLED, CB8CLED, and CB16CLED are, respectively, 2-, 4-, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the CEO output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage.

When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not. For CPLD designs, refer to the "CB2X1, CB4X1, CB8X1, CB16X1" section for high-performance cascadable, bidirectional counters.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted with an inverter in front of the GR/GSR input of STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX.

Inputs						Outputs		
CLR	L	CE	C	UP	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

z = 1 for CB2CLED; z = 3 for CB4CLED; z = 7 for CB8CLED; z = 15 for CB16CLED
dn = state of referenced input (Dn), one setup time prior to active clock transition
TC = (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP) + ($\overline{Qz}$ • $\overline{Q(z\text{-}1)}$ • $\overline{Q(z\text{-}2)}$ • ... • $\overline{Q0}$ • $\overline{UP}$)
CEO = TC • CE

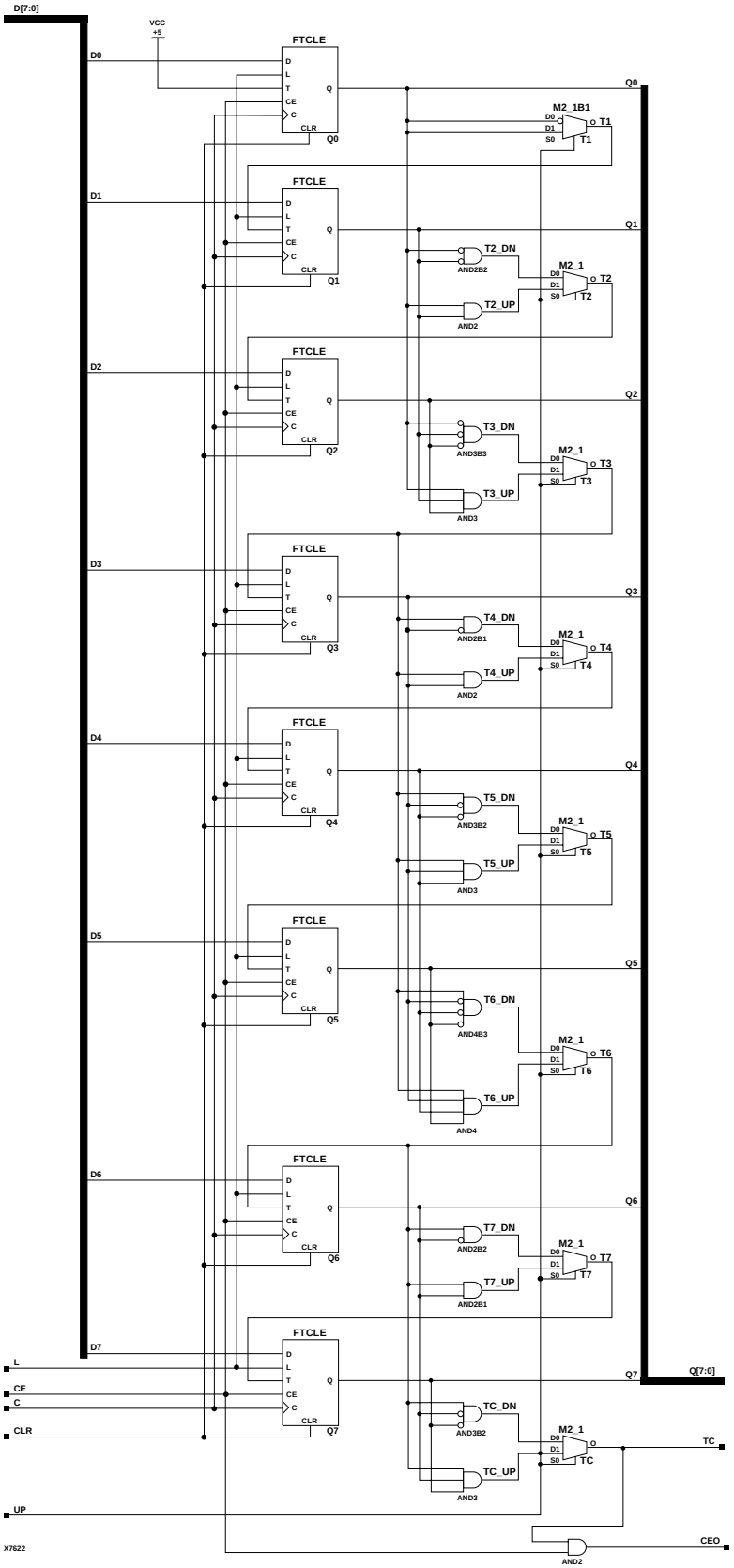


Figure 4-8 CB8CLED Implementation XC3000

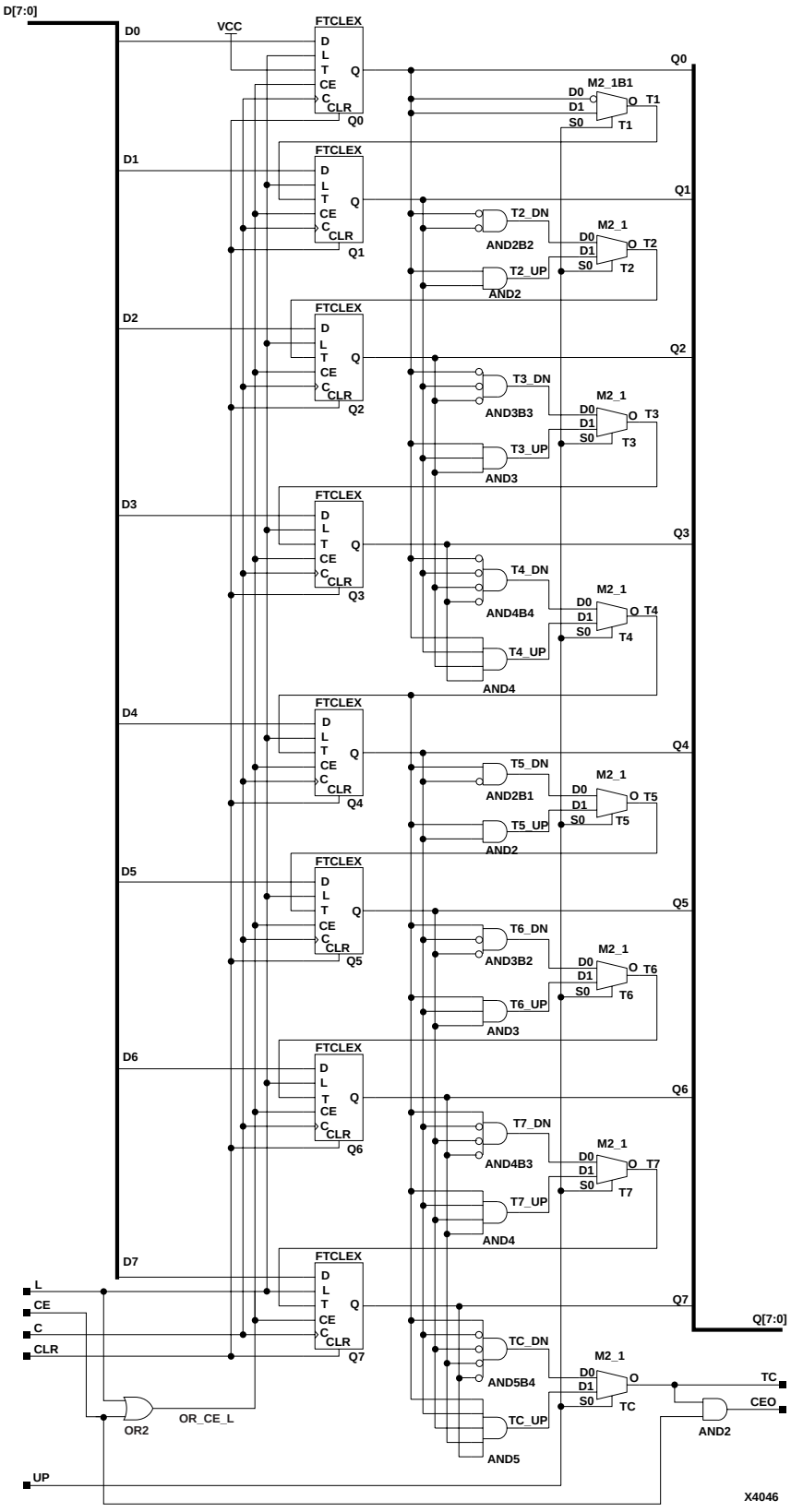


Figure 4-9 CB8CLED Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

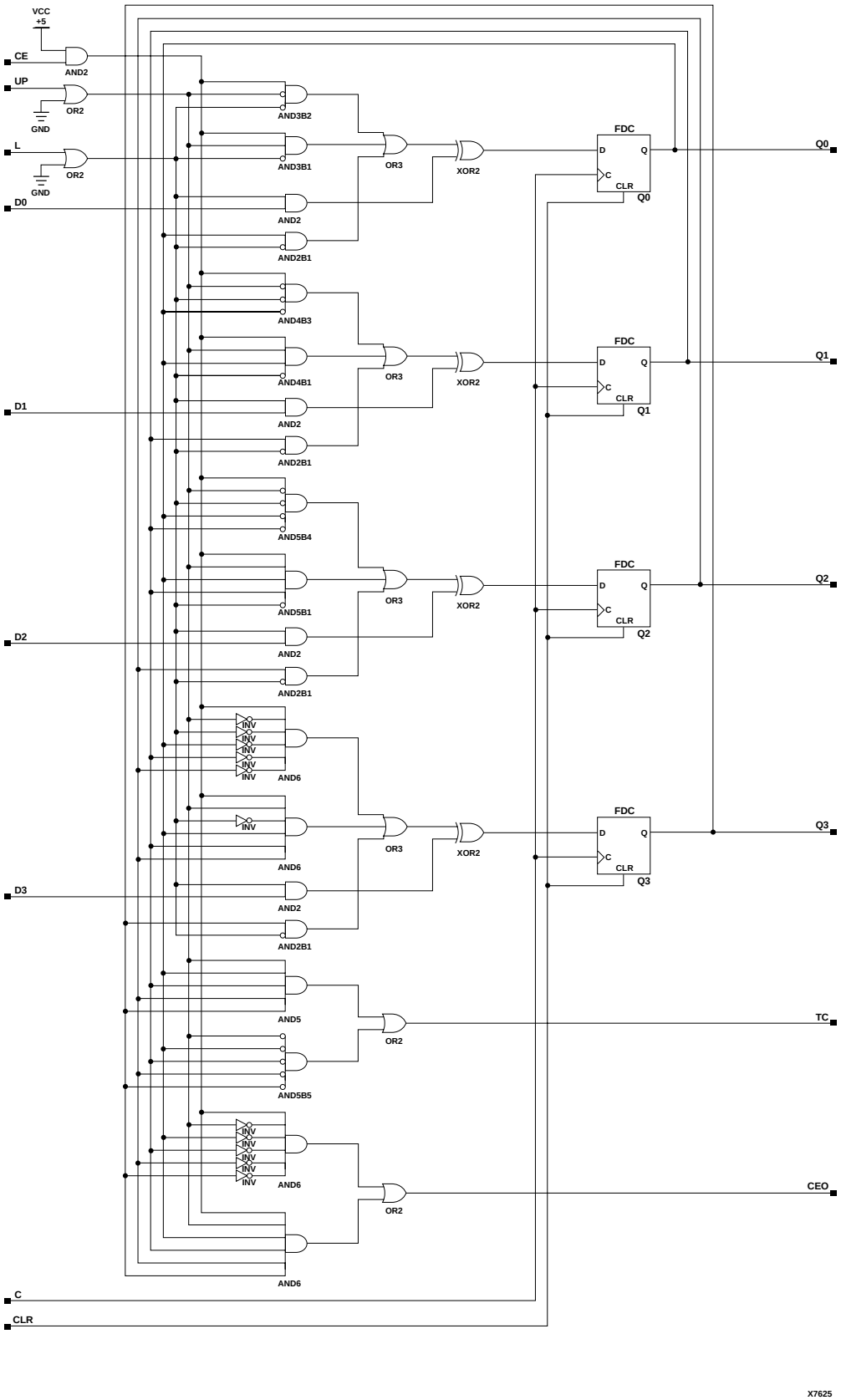
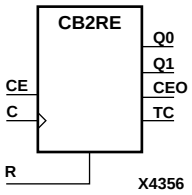


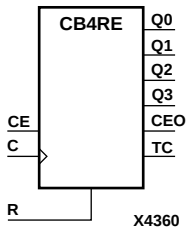
Figure 4-10 CB4CLED Implementation XC9000

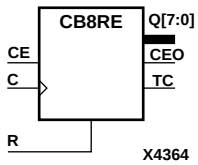
CB2RE, CB4RE, CB8RE, CB16RE

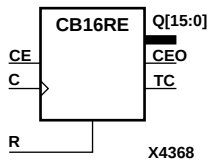
2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









CB2RE, CB4RE, CB8RE, and CB16RE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, resettable, cascadable binary counters. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero during the Low-to-High clock transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when both Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs	
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

z = 1 for CB2RE; z = 3 for CB4RE; z = 7 for CB8RE; z = 15 for CB16RE
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0)
CEO = TC • CE

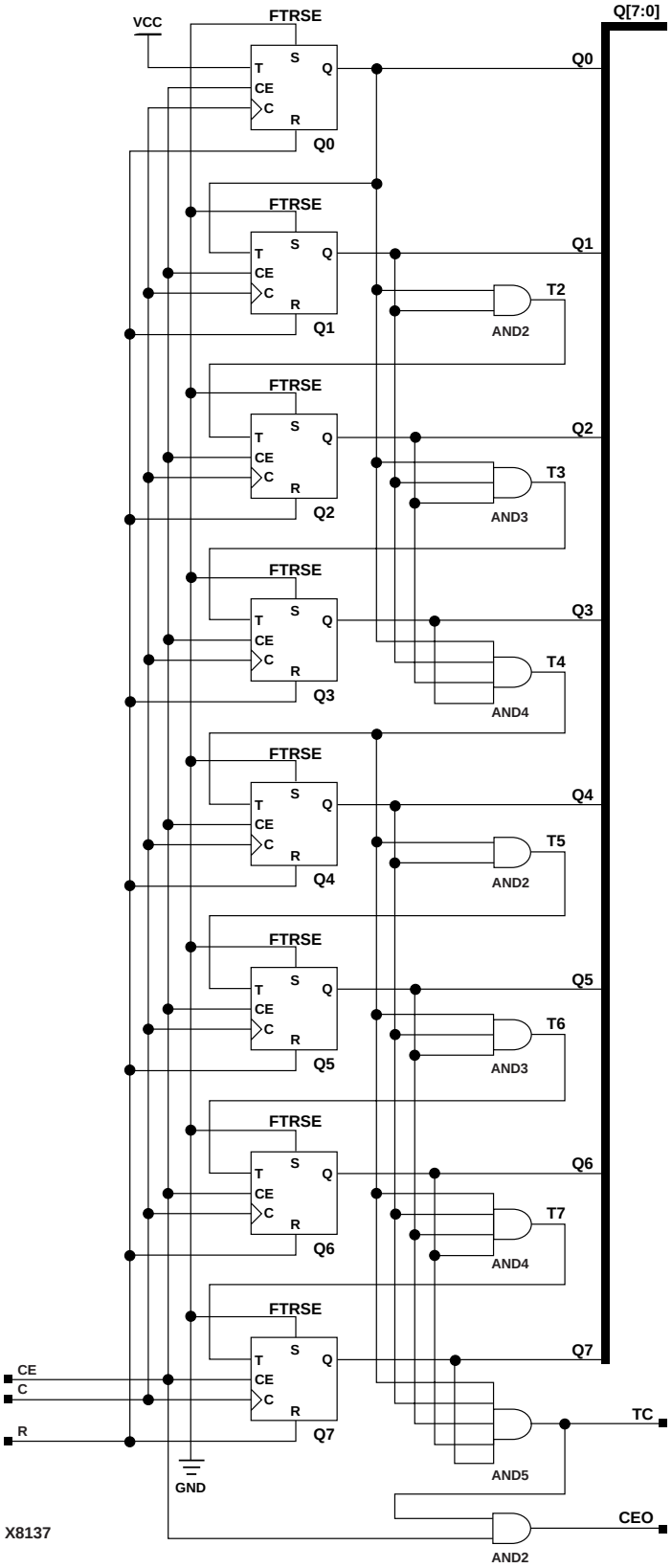


Figure 4-11 CB8RE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

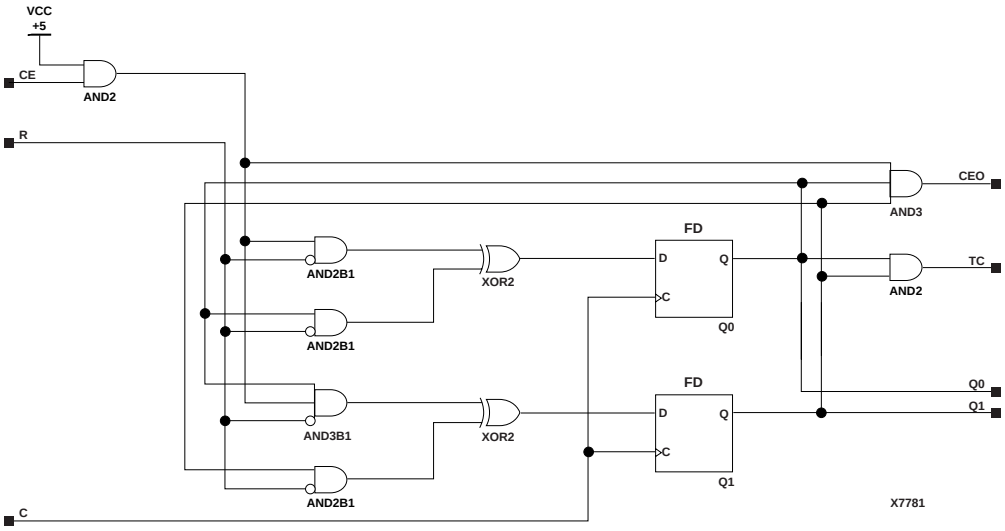


Figure 4-12 CB2RE Implementation XC9000

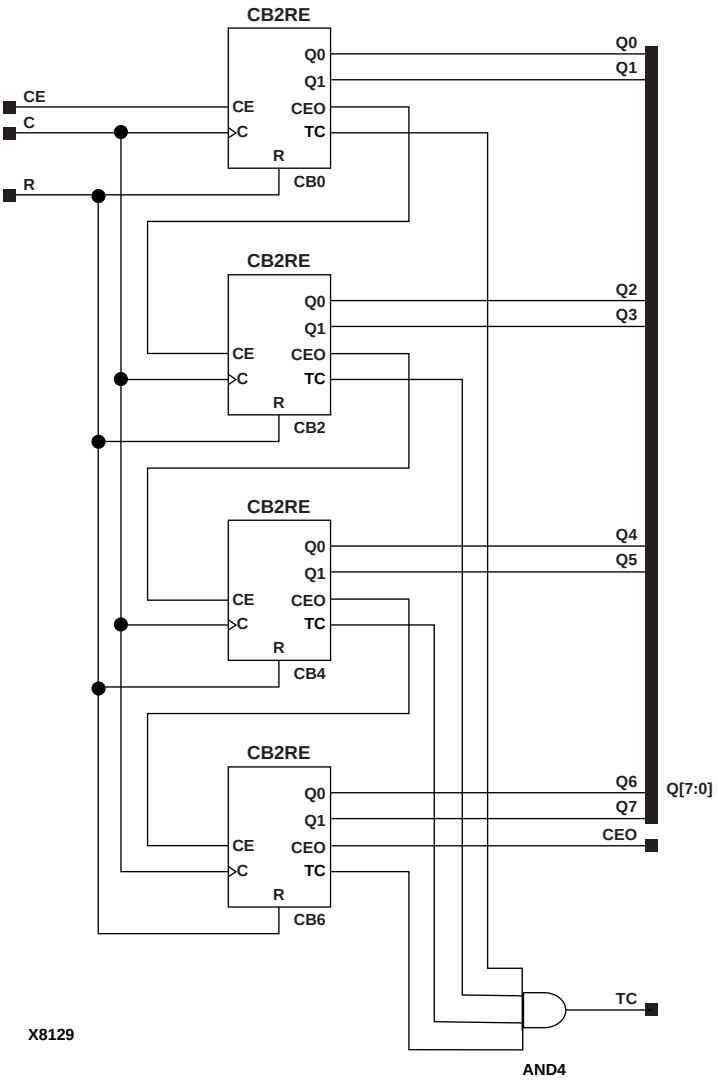
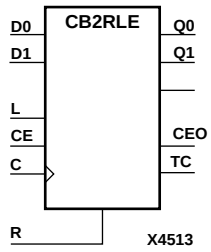


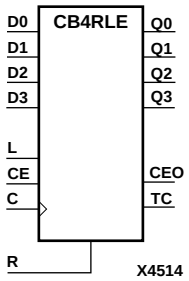
Figure 4-13 CB8RE Implementation XC9000

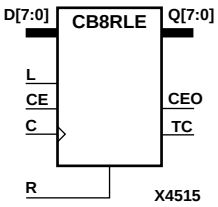
CB2RLE, CB4RLE, CB8RLE, CB16RLE

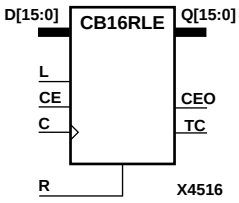
2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A









CB2RLE, CB4RLE, CB8RLE, and CB16RLE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, cascadable binary counter. The synchronous reset (R) is the highest priority input. The synchronous R, when High, overrides all other inputs and resets the Q outputs, terminal count (TC), and clock enable out (CEO) outputs to Low on the Low-to-High clock (C) transition.

The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of CE. The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High. The CEO output is High when all Q outputs and CE are High to allow direct cascading of counters.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and by connecting the C, L, and R inputs in parallel. The maximum length of the counter is determined by the accumulated CE-to-CEO propagation delays versus the clock period. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs		
R	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	↑	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

z = 1 for CB2RLE; z = 3 for CB4RLE; z = 7 for CB8RLE; z = 15 for CB16RLE
dn = state of referenced input (Dn) one setup time prior to active clock transition
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0
CEO = TC • CE

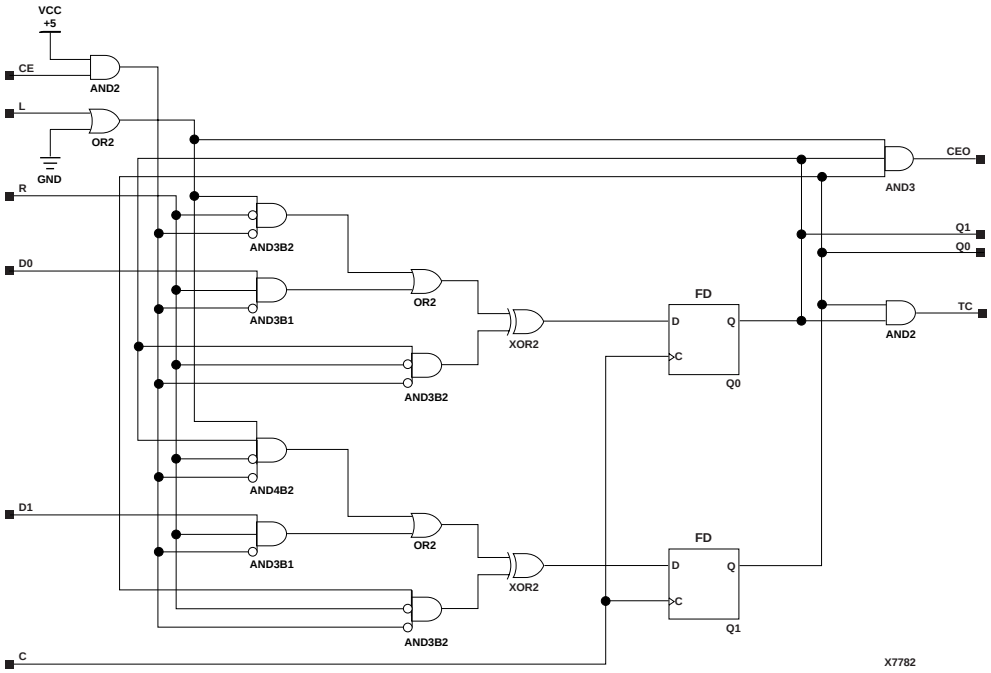


Figure 4-14 CB2RLE Implementation XC9000

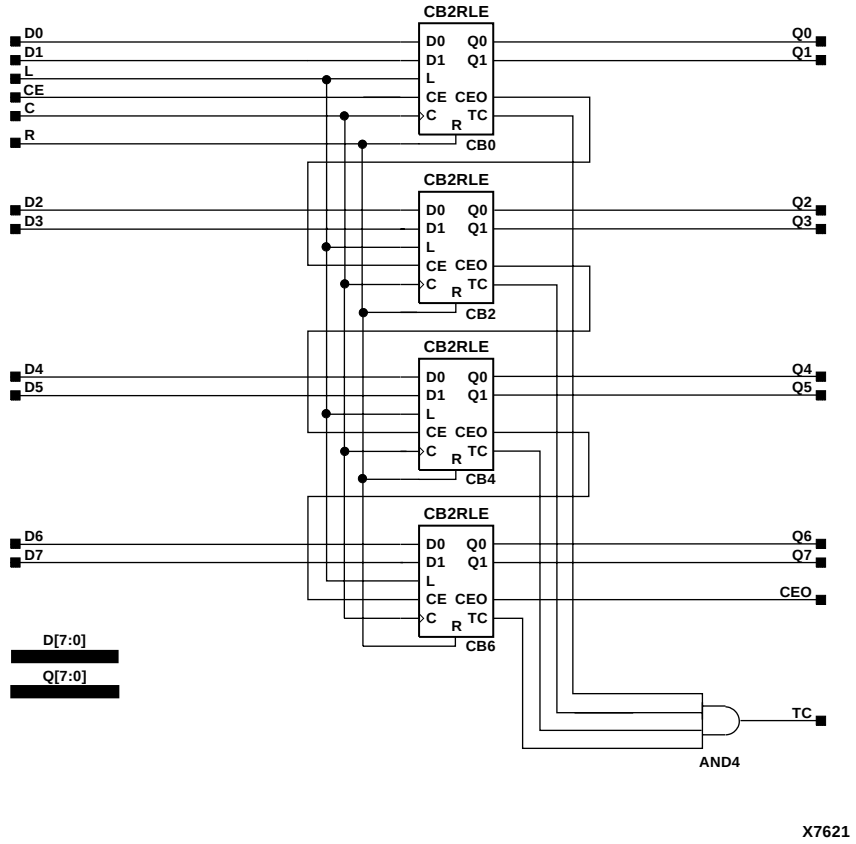
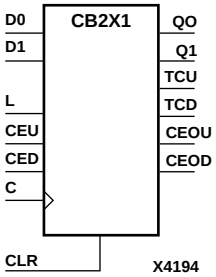


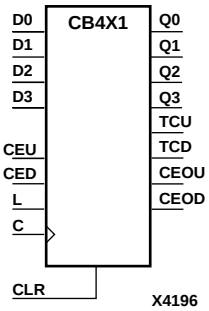
Figure 4-15 CB8RLE Implementation XC9000

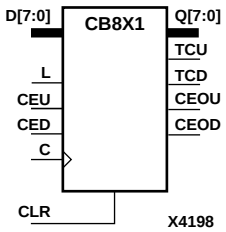
CB2X1, CB4X1, CB8X1, CB16X1

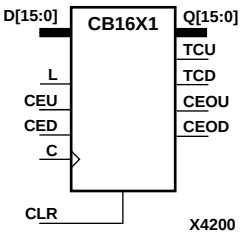
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A









CB2X1, CB4X1, CB8X1, and CB16X1 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronously loadable, asynchronously clearable, bidirectional binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in the CPLD architecture.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, clock enable outputs CEOU and CEOD go to Low and High, respectively, independent of clock transitions. The data on the D inputs loads into the counter on the Low-to-High clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

The Q outputs increment when CEU is High, provided CLR and L are Low, during the Low-to-High clock transition. The Q outputs decrement when CED is High, provided CLR and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are connected directly to the CEU and CED inputs, respectively, of the next stage. The clock, L, and CLR inputs are connected in parallel.

In Xilinx CPLD devices, the maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
CLR	**L**	**CEU**	**CED**	**C**	**Dz–D0**	**Qz–Q0**	**TCU**	**TCD**	**CEOU**	**CEOD**
1	X	X	X	X	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CB2X1; z = 3 for CB4X1; z = 7 for CB8X1; z = 15 for CB16X1
dn = state of referenced input (Dn) one setup time prior to active clock transition
TCU = $Qz \cdot \overline{Q(z-1)} \cdot \overline{Q(z-2)} \cdot \ldots \cdot \overline{Q0}$
TCD = $\overline{Qz} \cdot \overline{Q(z-1)} \cdot \overline{Q(z-2)} \cdot \ldots \cdot \overline{Q0}$
CEOU = TCU • CEU
CEOD = TCD • CED

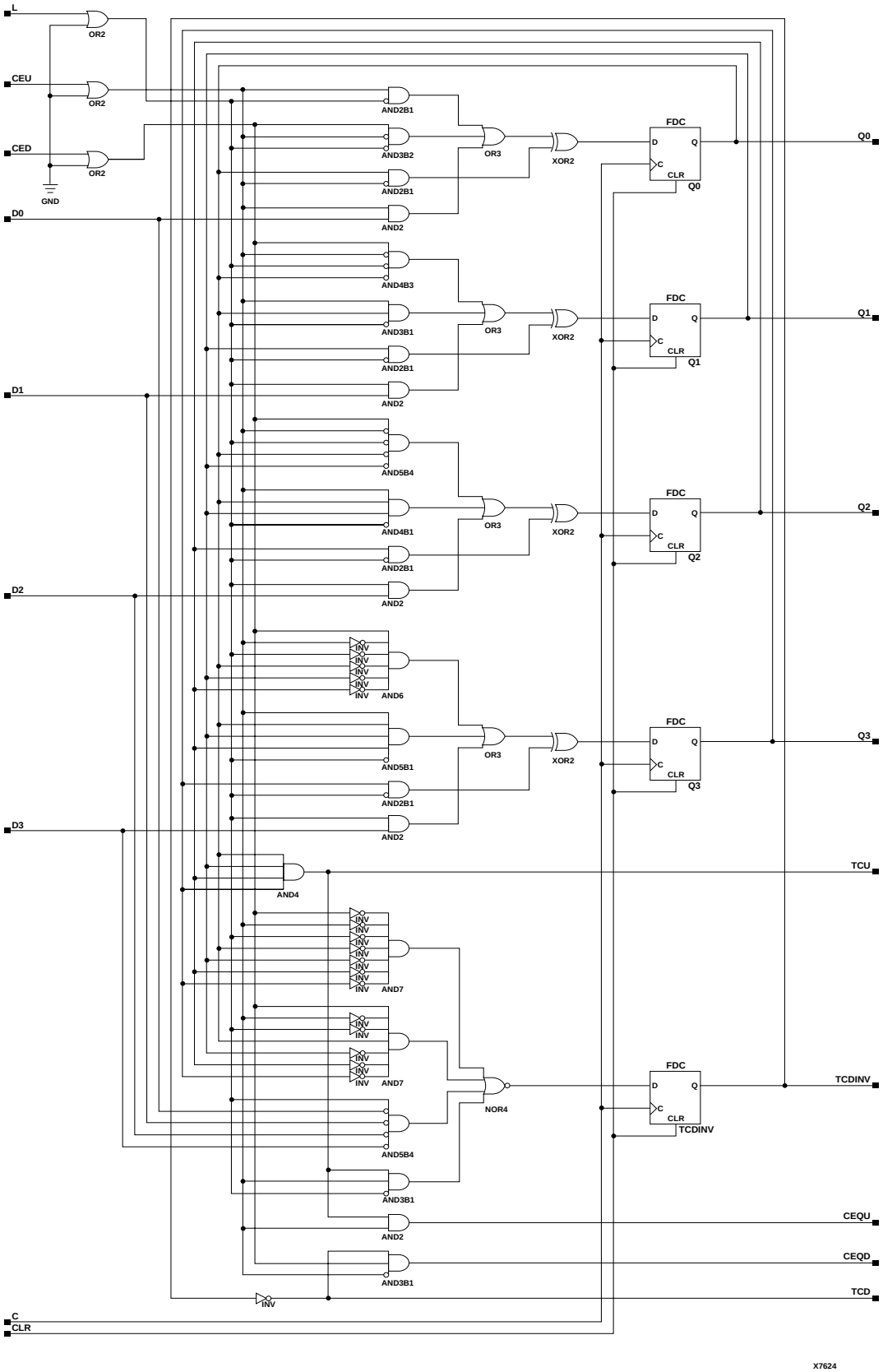
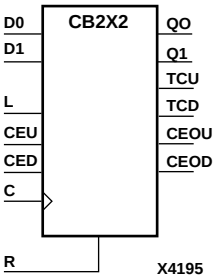


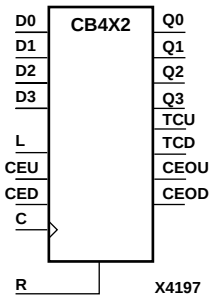
Figure 4-16 CB4X1 Implementation XC9000

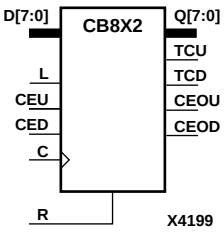
CB2X2, CB4X2, CB8X2, CB16X2

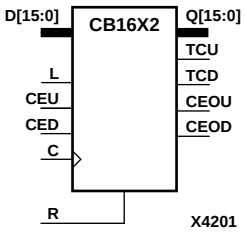
2-, 4-, 8-, and 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Macro	N/A	N/A	N/A	N/A









CB2X2, CB4X2, CB8X2, and CB16X2 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, bidirectional binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in the CPLD architecture.

The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, and clock enable outputs CEOU and CEOD go to Low and High, respectively, on the Low-to-High clock (C) transition. The data on the D inputs loads into the counter on the Low-to-High clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

All Q outputs increment when CEU is High, provided R and L are Low during the Low-to-High clock transition. All Q outputs decrement when CED is High, provided R and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are, respectively, connected directly to the CEU and CED inputs of the next stage. The C, L, and R inputs are connected in parallel.

In Xilinx CPLD devices, the maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
R	**L**	**CEU**	**CED**	**C**	**Dz – D0**	**Qz – Q0**	**TCU**	**TCD**	**CEOU**	**CEOD**
1	X	X	X	↑	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CB2X2; z = 3 for CB4X2; z = 7 for CB8X2; z = 15 for CB16X2
d = state of referenced input (Dn) one setup time prior to active clock transition
TCU = $Qz \cdot \overline{Q(z-1)} \cdot \overline{Q(z-2)} \cdot \ldots \cdot \overline{Q0}$
TCD = $\overline{Qz} \cdot \overline{Q(z-1)} \cdot \overline{Q(z-2)} \cdot \ldots \cdot \overline{Q0}$
CEOU = TCU • CEU
CEOD = TCD • CED

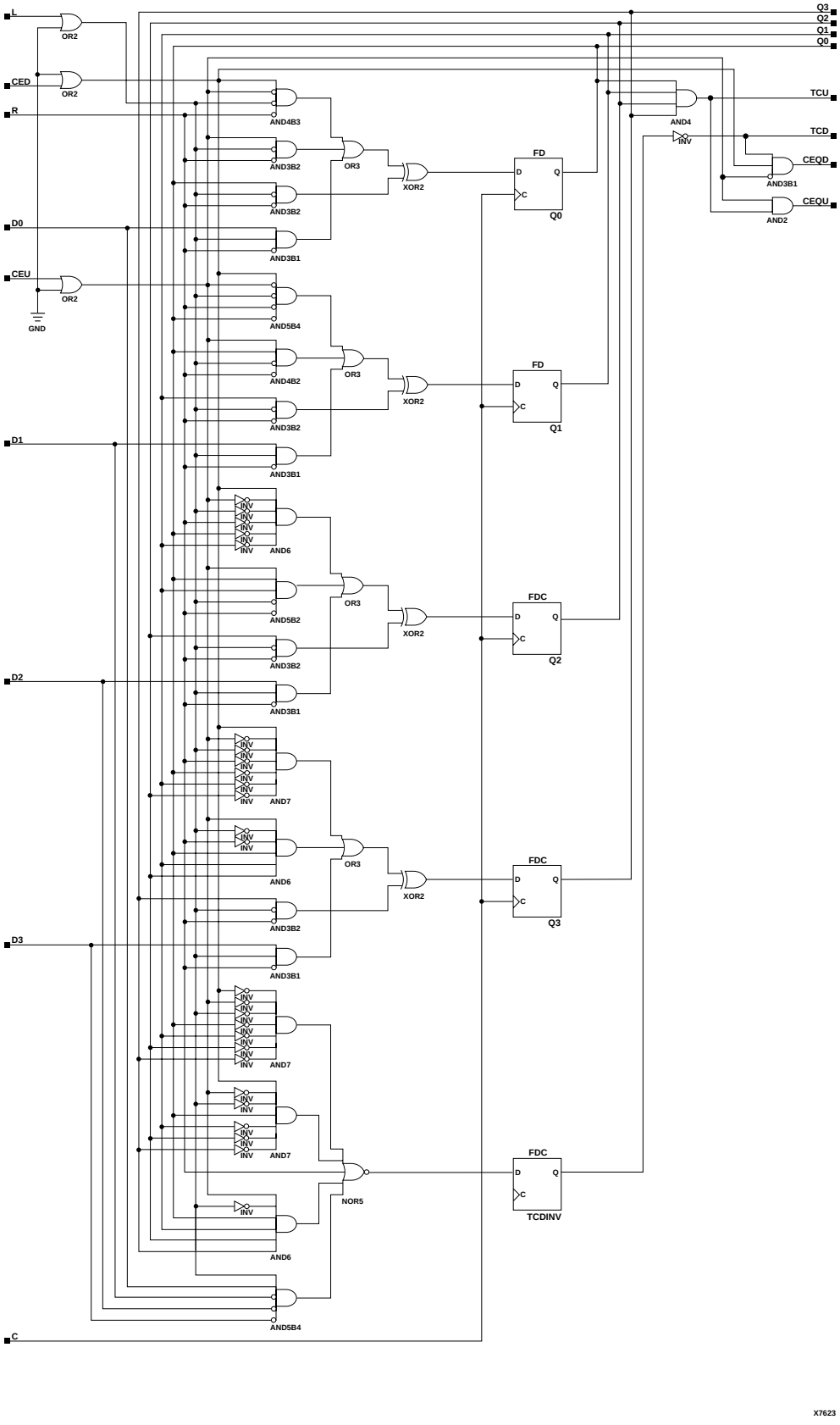
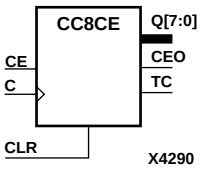


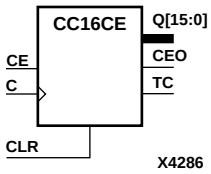
Figure 4-17 CB4X2 Implementation XC9000

CC8CE, CC16CE

8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	N / A	Macro	Macro	Macro	Macro





CC8CE and CC16CE are, respectively, 8- and 16-bit (stage), asynchronous, clearable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

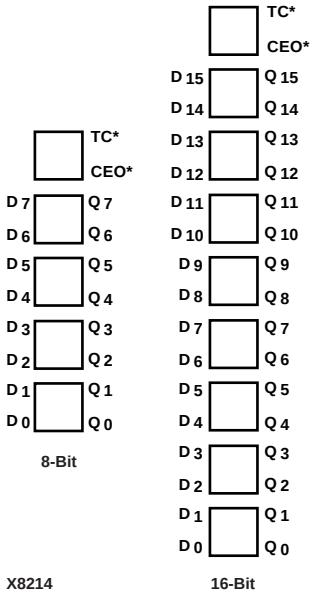
The counter is asynchronously cleared, with Low outputs, when power is applied. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs		
CLR	CE	C	Qz – Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

z = 7 for CC8CE; z = 15 for CC16CE
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0
CEO = TC • CE

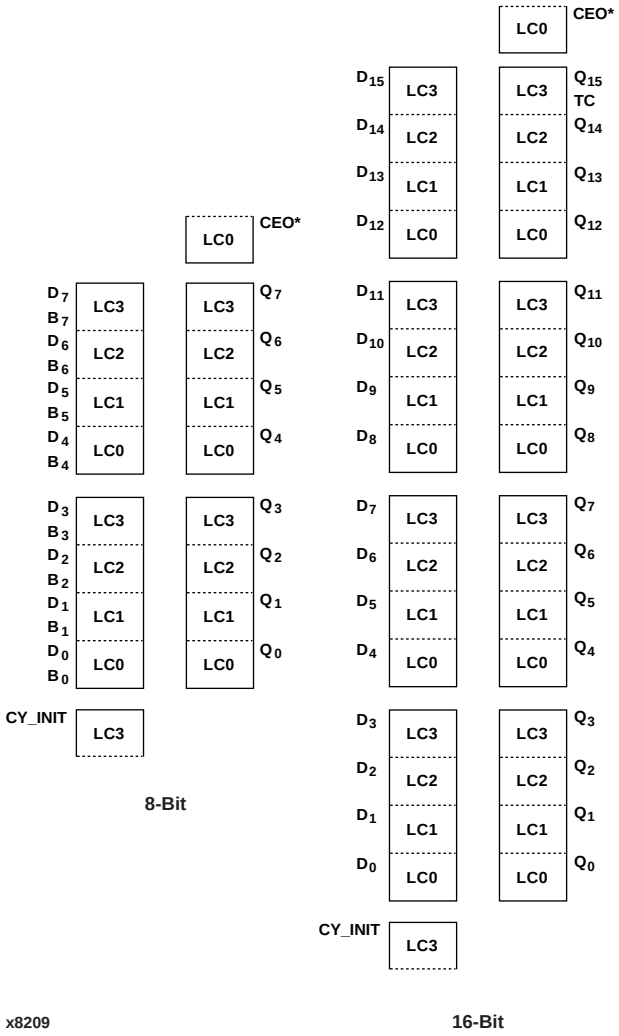
Topology for XC4000, Spartan, SpartanXL

This is the CC8CE (8-bit) and CC16CE (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



Topology for XC5200

This is the CC8CE (8-bit) and CC16CE (16-bit) topology for XC5200 devices.



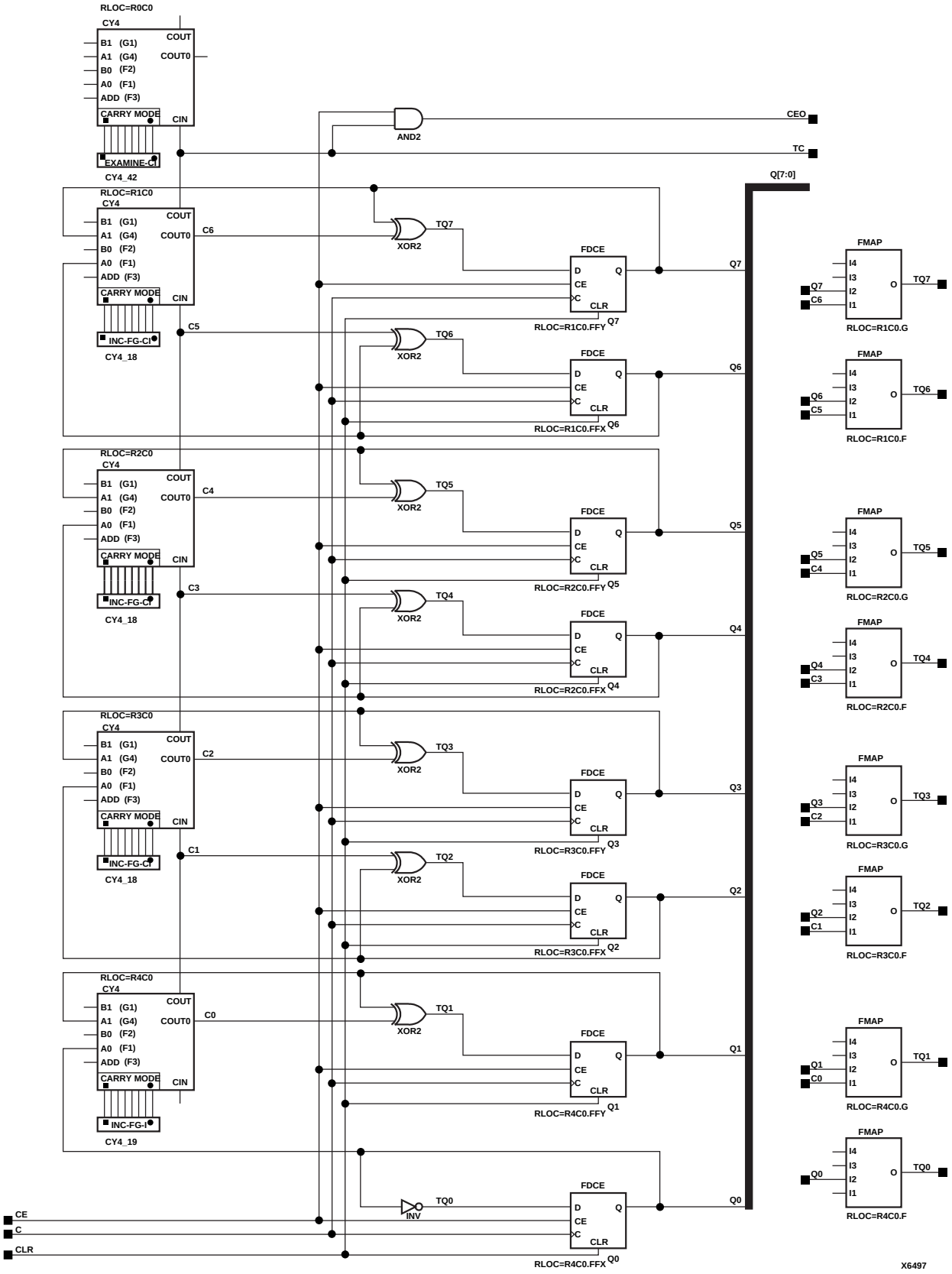


Figure 4-18 CC8CE Implementation XC4000E, XC4000X, Spartan, SpartanXL

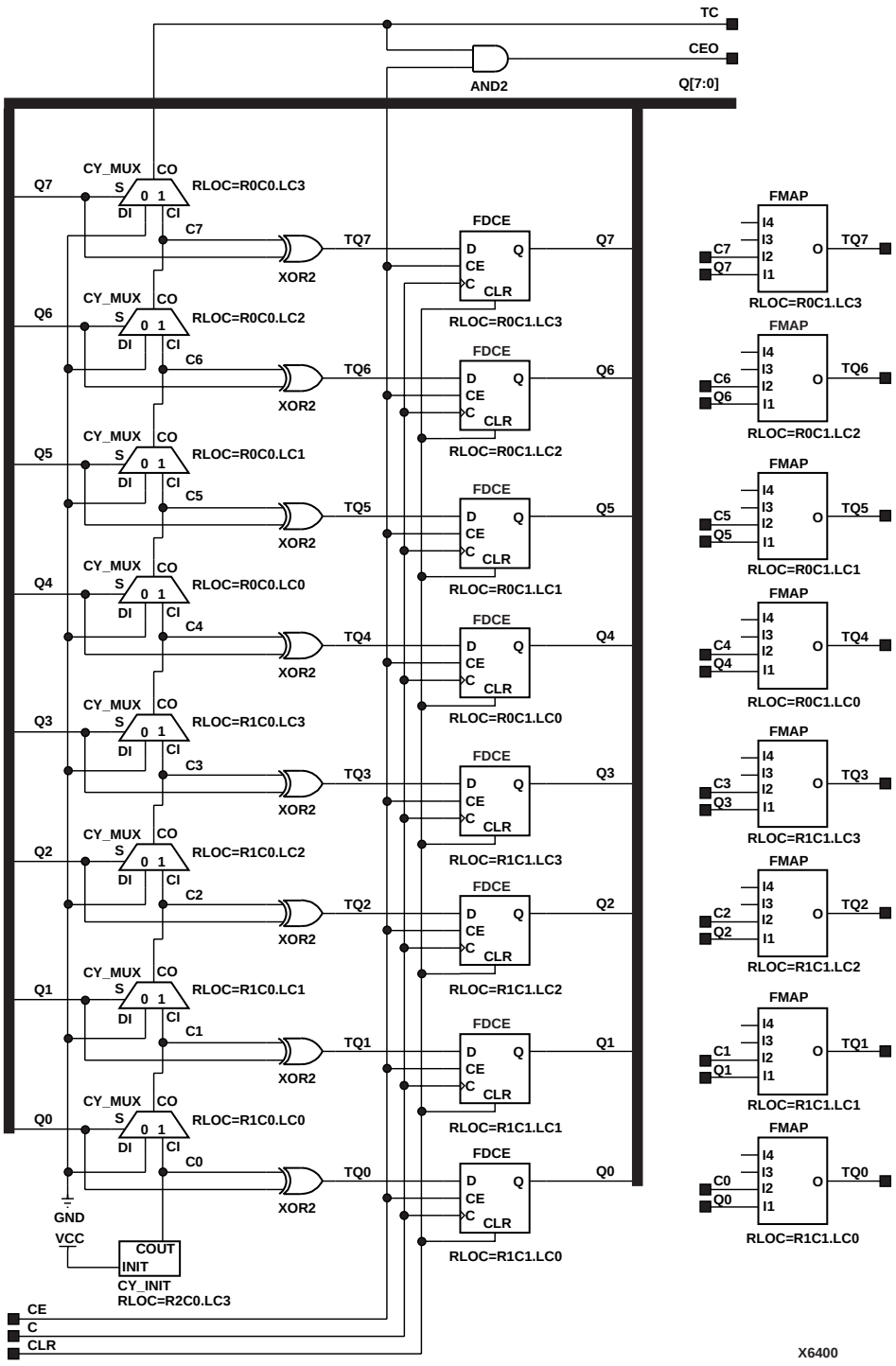


Figure 4-19 CC8CE Implementation XC5200

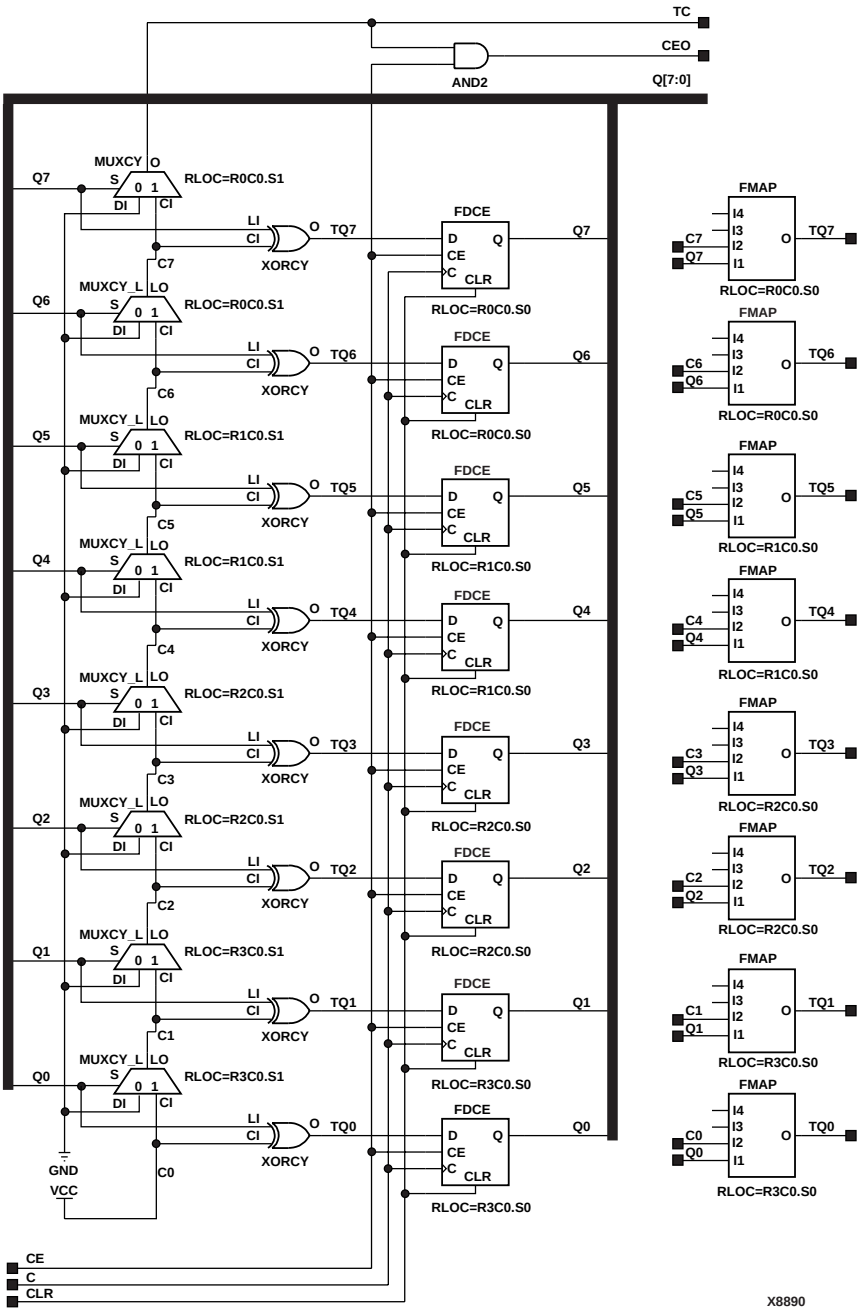
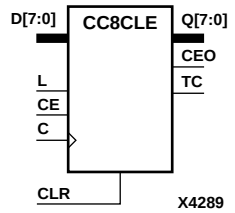


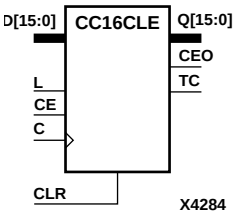
Figure 4-20 CC8CE Implementation Spartan2, Virtex

CC8CLE, CC16CLE

8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	N / A	Macro	Macro	Macro	Macro





CC8CLE and CC16CLE are, respectively, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable binary counter. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

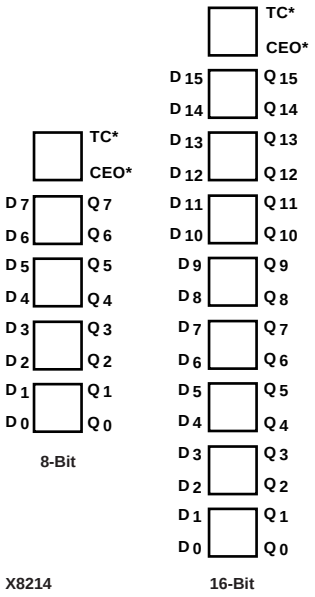
The counter is asynchronously cleared, with Low output, when power is applied. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs		
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

z = 7 for CC8CLE; z = 15 for CC16CLE
dn = state of referenced input (Dn) one setup time prior to active clock transition
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0
CEO = TC • CE

Topology for XC4000, Spartan, SpartanXL

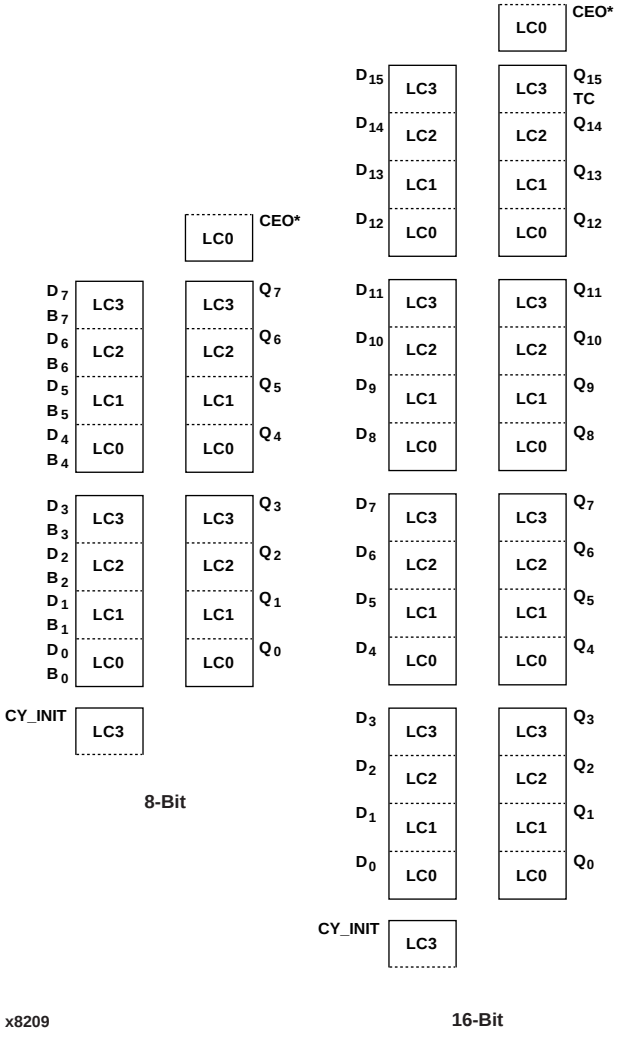
This is the CC8CLE (8-bit) and CC16CLE (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



In the process of combining the logic that loads CEO and TC, the place and route software might map the logic that generates CEO and TC to different function generators. If this mapping occurs, the CEO and TC logic cannot be placed in the uppermost CLB as indicated in the illustration.

Topology for XC5200

This is the CC8CLE (8-bit) and CC16CLE (16-bit) topology for XC5200 devices.



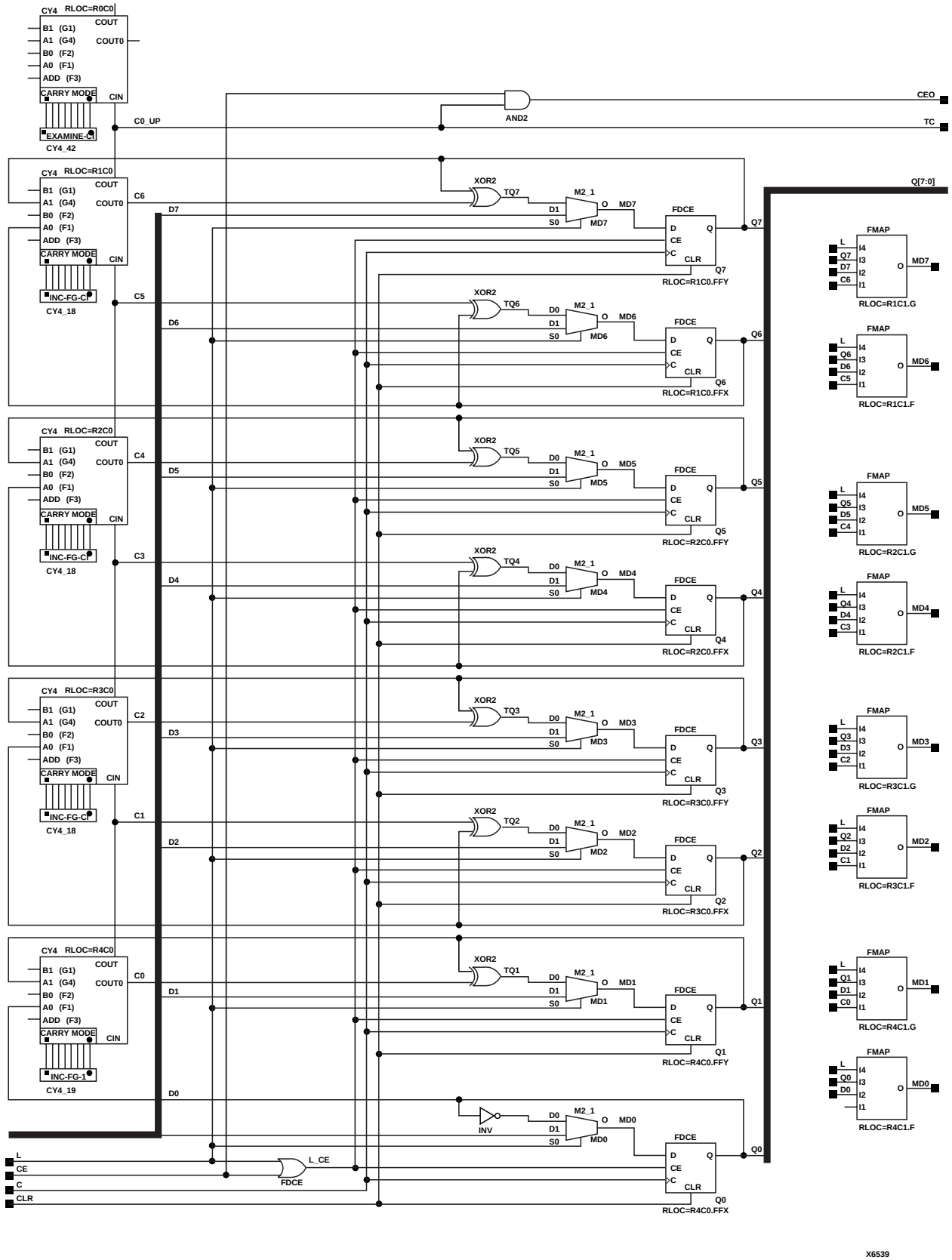


Figure 4-21 CC8CLE Implementation XC4000E, XC4000X, Spartan, SpartanXL

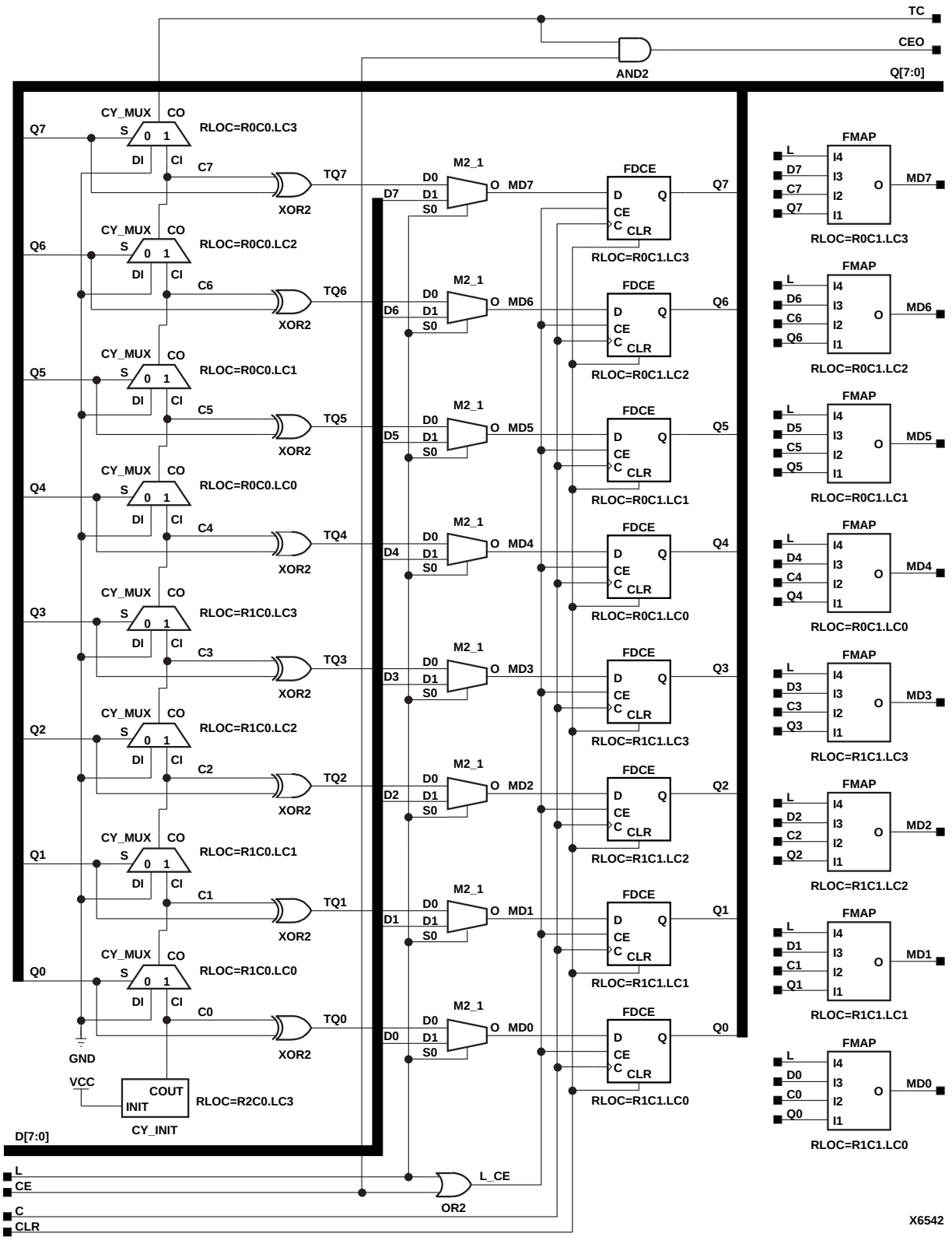


Figure 4-22 CC8CLE Implementation XC5200

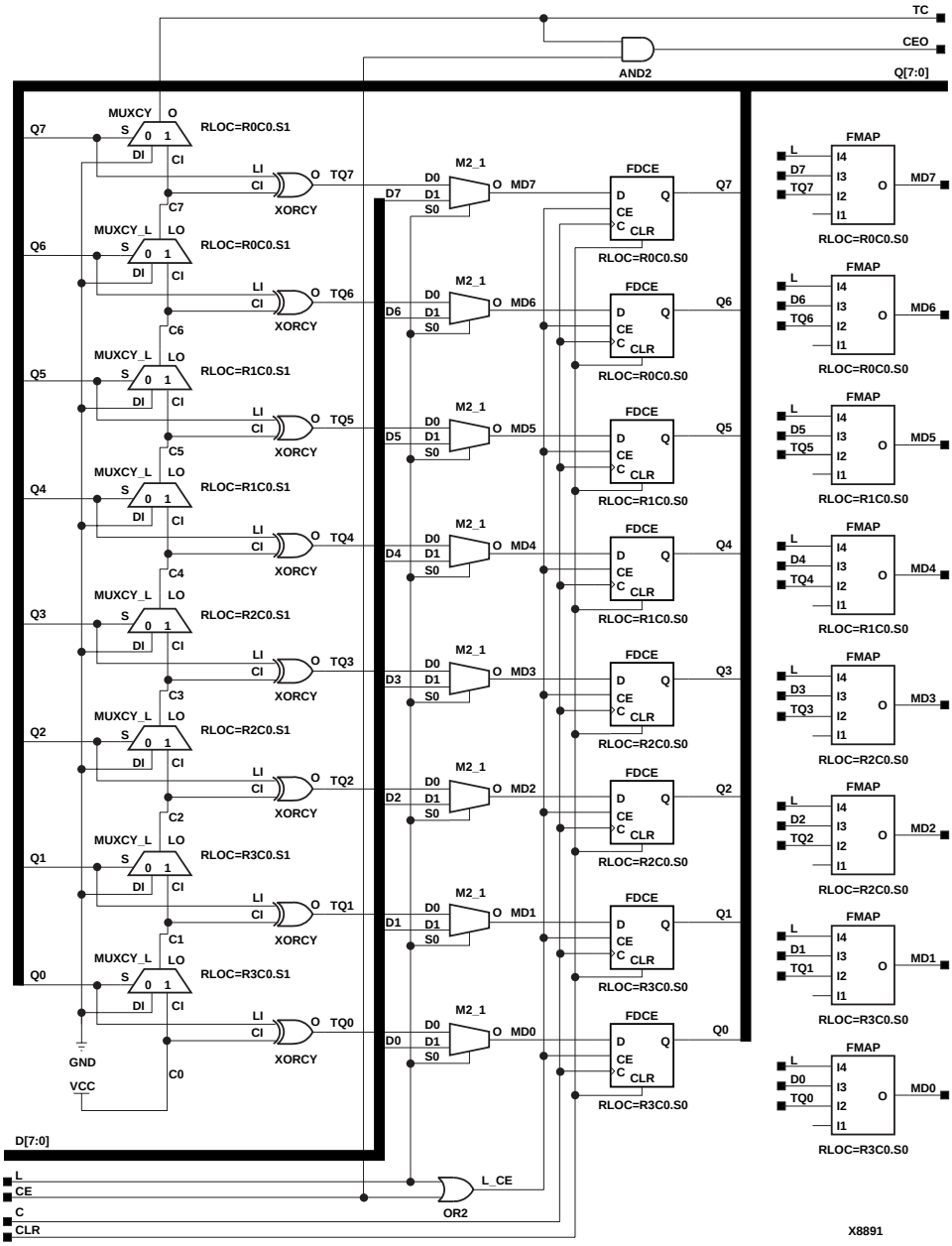
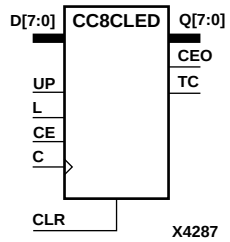


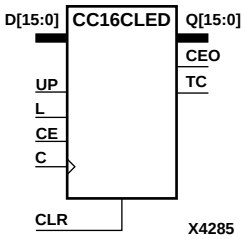
Figure 4-23 CC8CLE Implementation Spartan2, Virtex

CC8CLED, CC16CLED

8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	N / A	Macro	Macro	Macro	Macro





CC8CLED and CC16CLED are, respectively, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. These counters are implemented using carry logic with relative location constraints, which assures most efficient logic placement.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the count enable out (CEO) output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

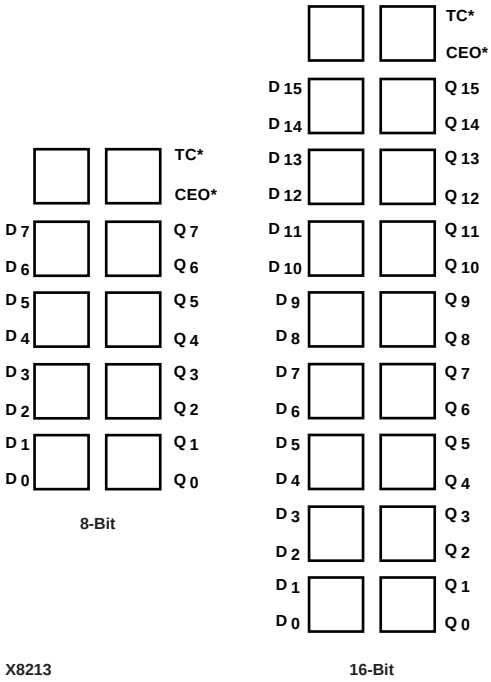
The counter is asynchronously cleared, outputs Low, when power is applied. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs		
CLR	L	CE	C	UP	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

z = 7 for CC8CLED; z = 15 for CC16CLED
dn = state of referenced input (Dn) one setup time prior to active clock transition
TC = (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP) + ($\overline{Qz}$ • $\overline{Q(z-1)}$ • $\overline{Q(z-2)}$ • ... • $\overline{Q0}$ • $\overline{UP}$)
CEO = TC • CE

Topology for XC4000, Spartan, SpartanXL

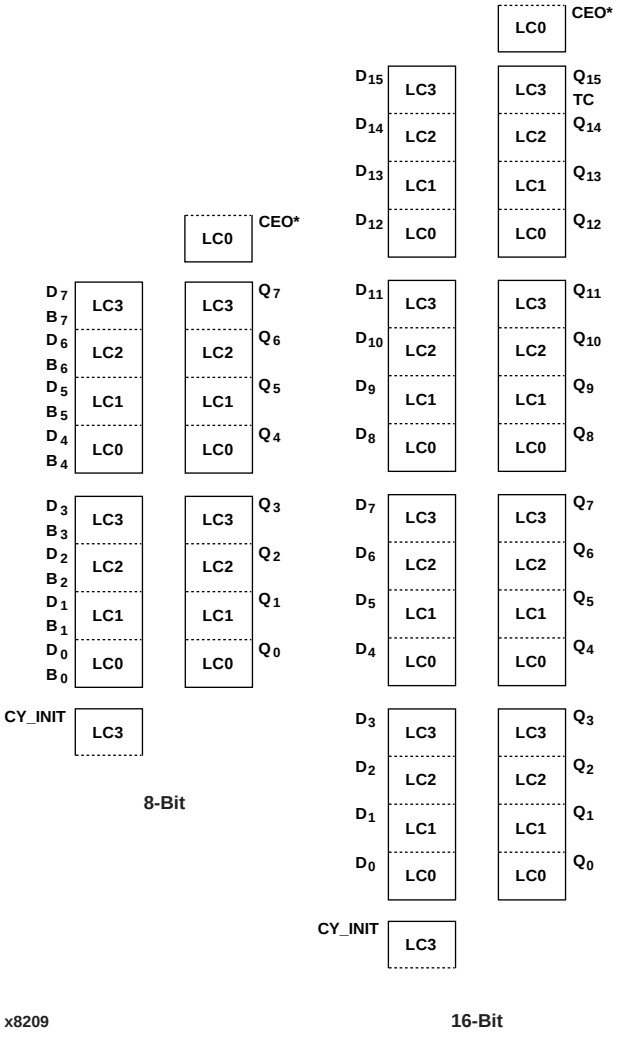
This is the CC8CLED (8-bit) and CC16CLED (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



In the process of combining the logic that loads CEO and TC, the place and route software might map the logic that generates CEO and TC to different function generators. If this mapping occurs, the CEO and TC logic cannot be placed in the uppermost CLB as indicated in the illustration.

Topology for XC5200

This is the CC8CLED (8-bit) and CC16CLED (16-bit) topology for XC5200 devices.



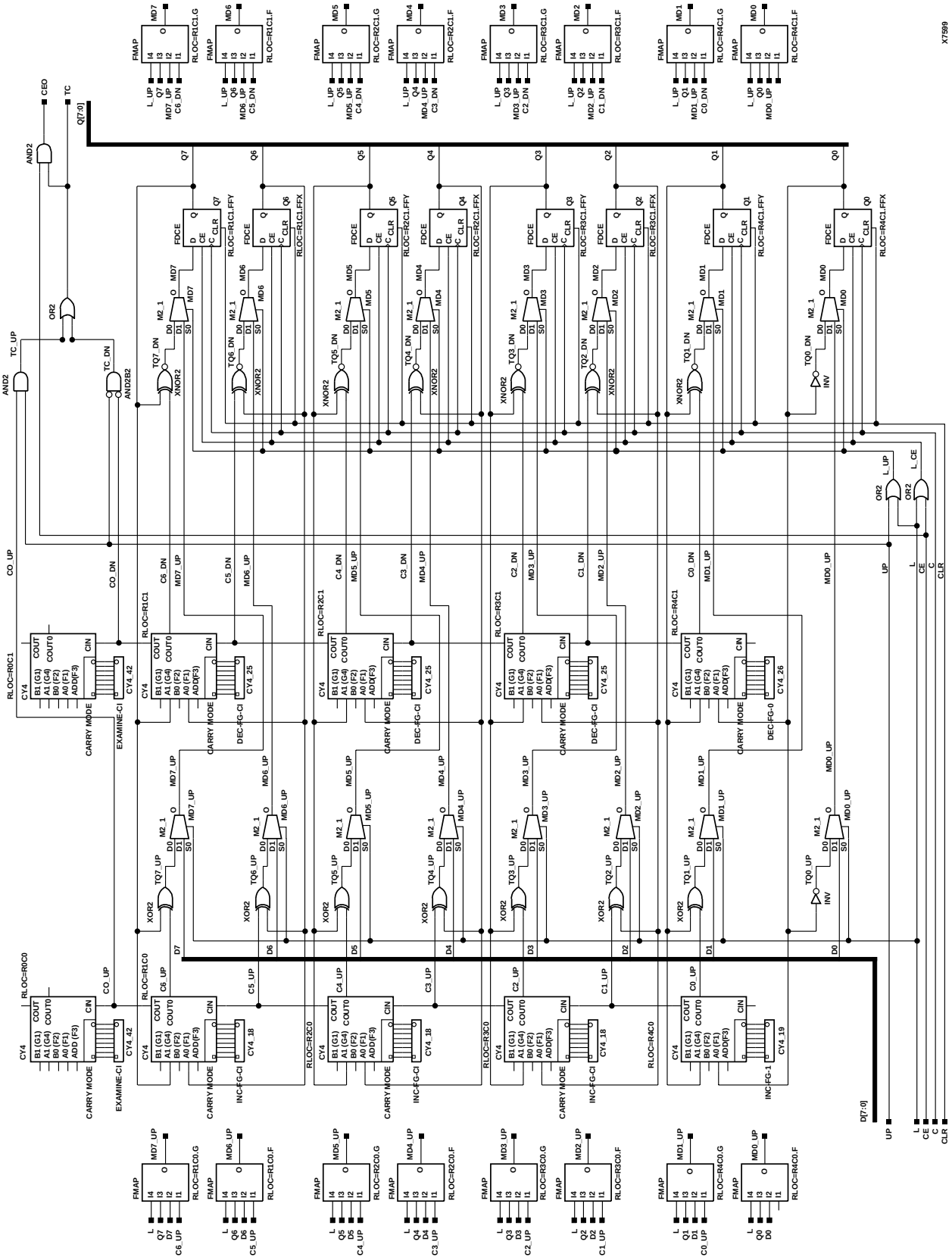


Figure 4-24 CC8CLED Implementation XC4000E, XC4000X, Spartan, SpartanXL

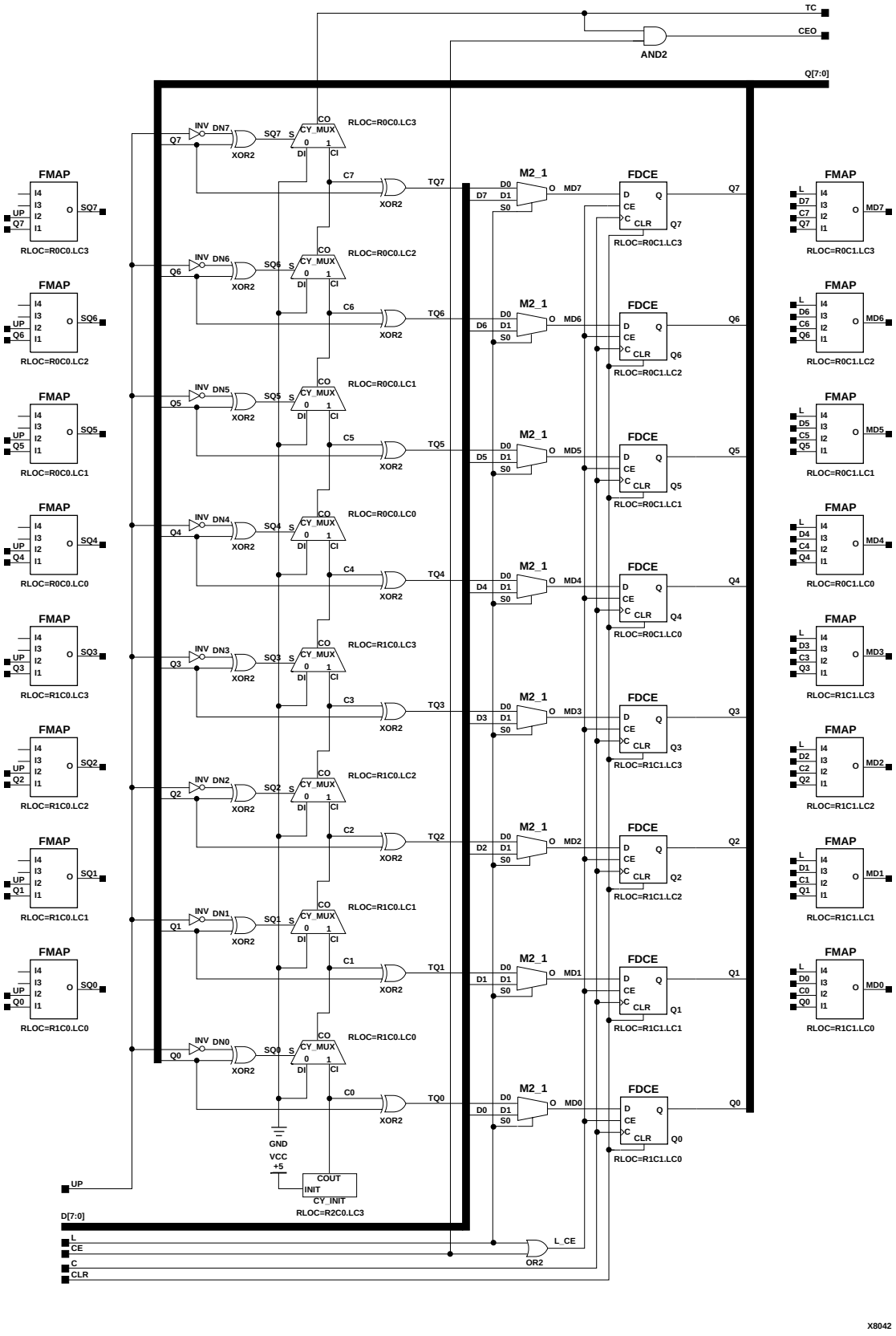


Figure 4-25 CC8CLED Implementation XC5200

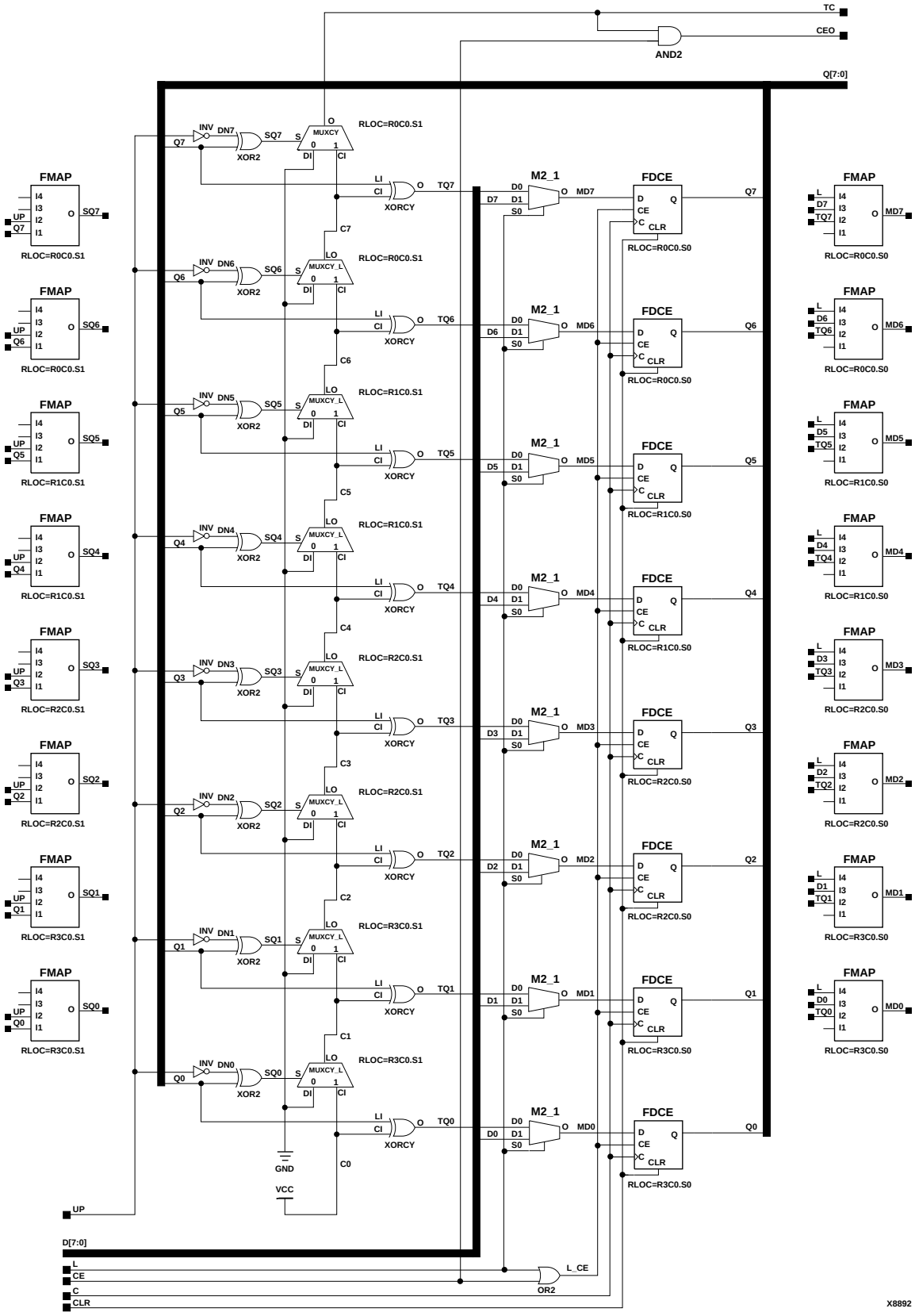
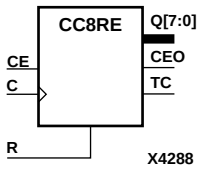


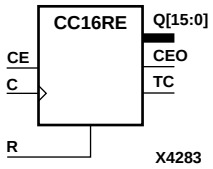
Figure 4-26 CC8CLED Implementation Spartan2, Virtex

CC8RE, CC16RE

8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	N / A	Macro	Macro	Macro	Macro





CC8RE and CC16RE are, respectively, 8- and 16-bit (stage), synchronous, resettable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs and CE are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

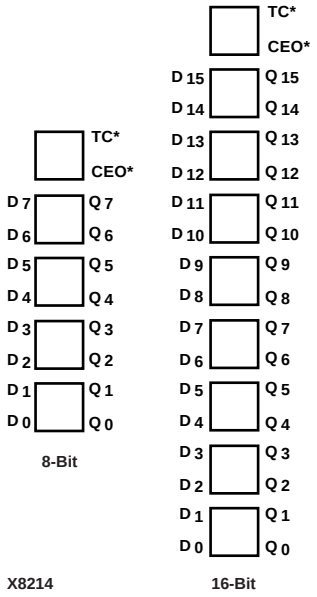
The counter is asynchronously cleared, with Low outputs, when power is applied. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

z = 7 for CC8RE; z = 15 for CC16RE
TC = Qz • Q(z-1) • Q(z-2) • ... • Q0 • CE
CEO = TC • CE

Topology for XC4000, Spartan, SpartanXL

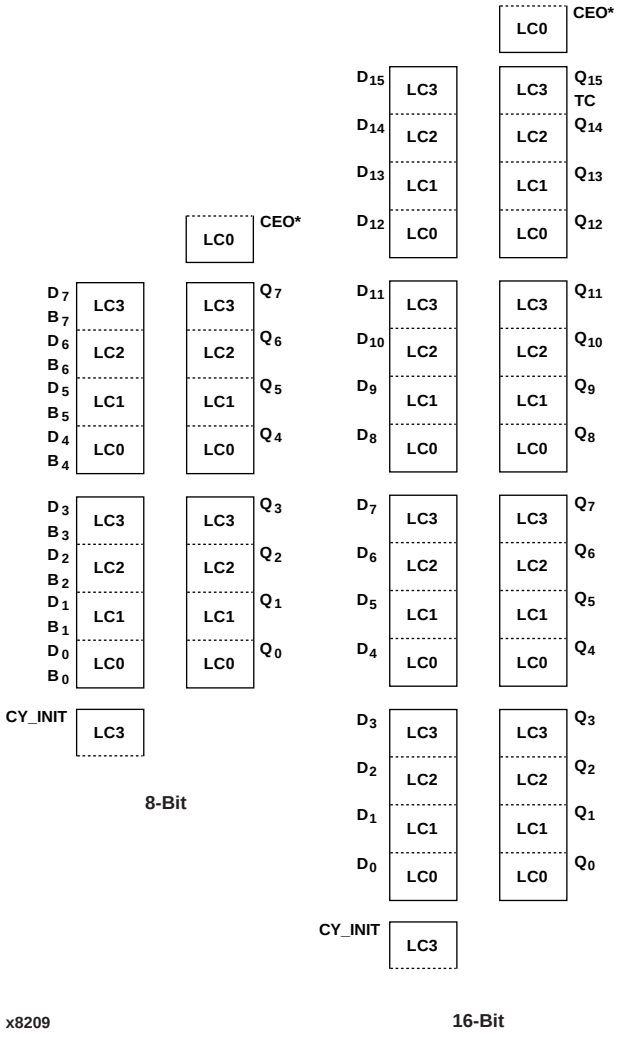
This is the CC8RE (8-bit) and CC16RE (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



In the process of combining the logic that loads CEO and TC, the place and route software might map the logic that generates CEO and TC to different function generators. If this mapping occurs, the CEO and TC logic cannot be placed in the uppermost CLB as indicated in the illustration.

Topology for XC5200

This is the CC8RE (8-bit) and CC16RE (16-bit) topology for XC5200 devices.



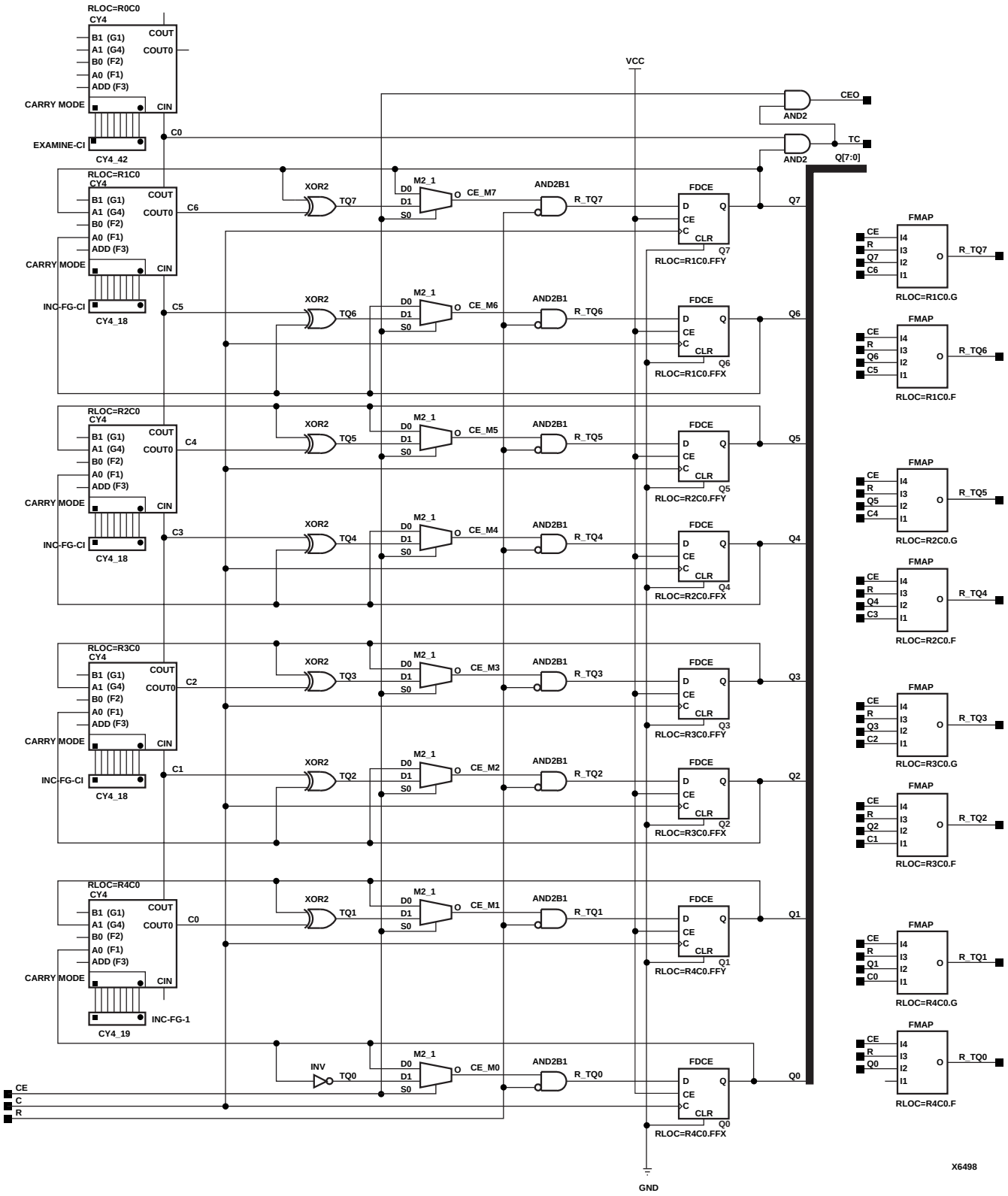


Figure 4-27 CC8RE Implementation XC4000E, XC4000X, Spartan, SpartanXL

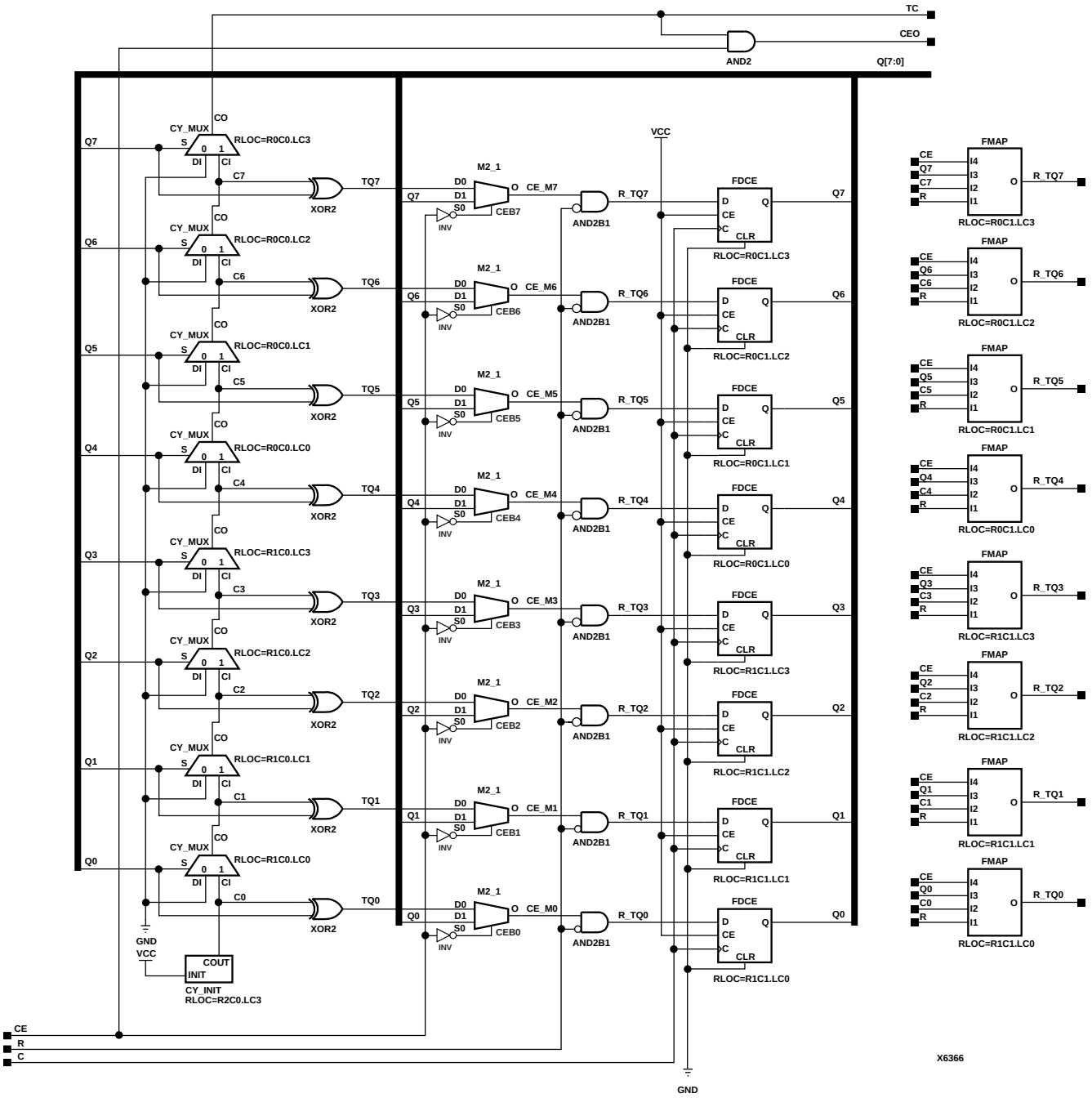


Figure 4-28 CC8RE Implementation XC5200

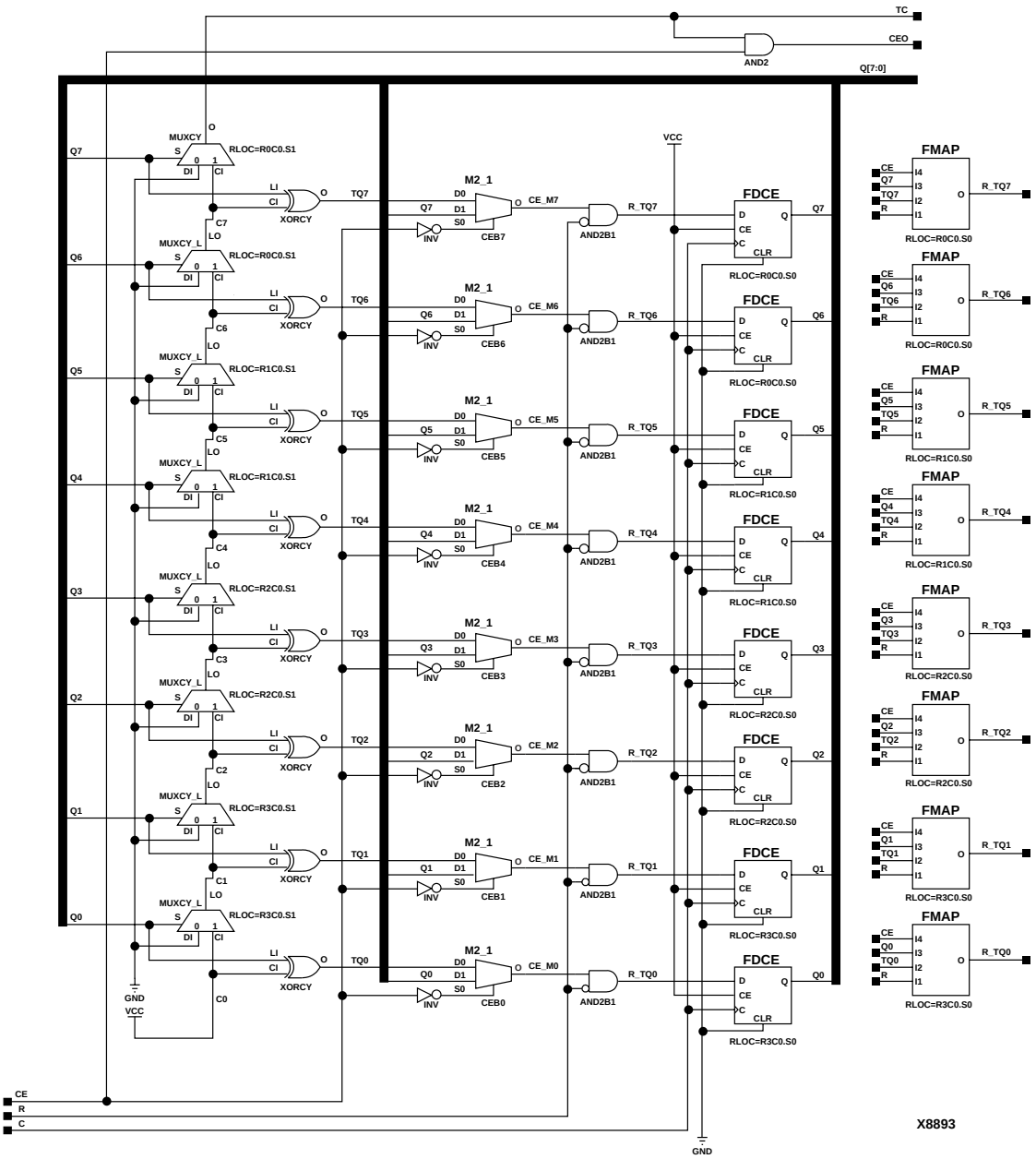
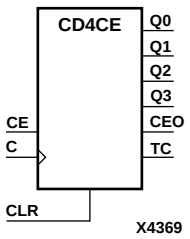


Figure 4-29 CC8RE Implementation Spartan2, Virtex

CD4CE

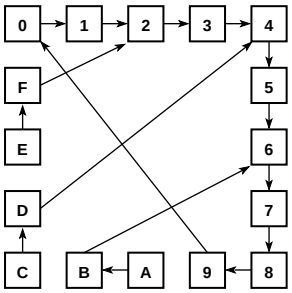
4-Bit Cascadable BCD Counter with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



CD4CE is a 4-bit (stage), asynchronous, clearable, cascadable binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when clock enable (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for FPGA architectures, as shown in the following state diagram. For XC9000, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse to the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs					
CLR	**CE**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	X	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

TC = Q3 • $\overline{Q2}$ • $\overline{Q1}$ • Q0
CEO = TC • CE

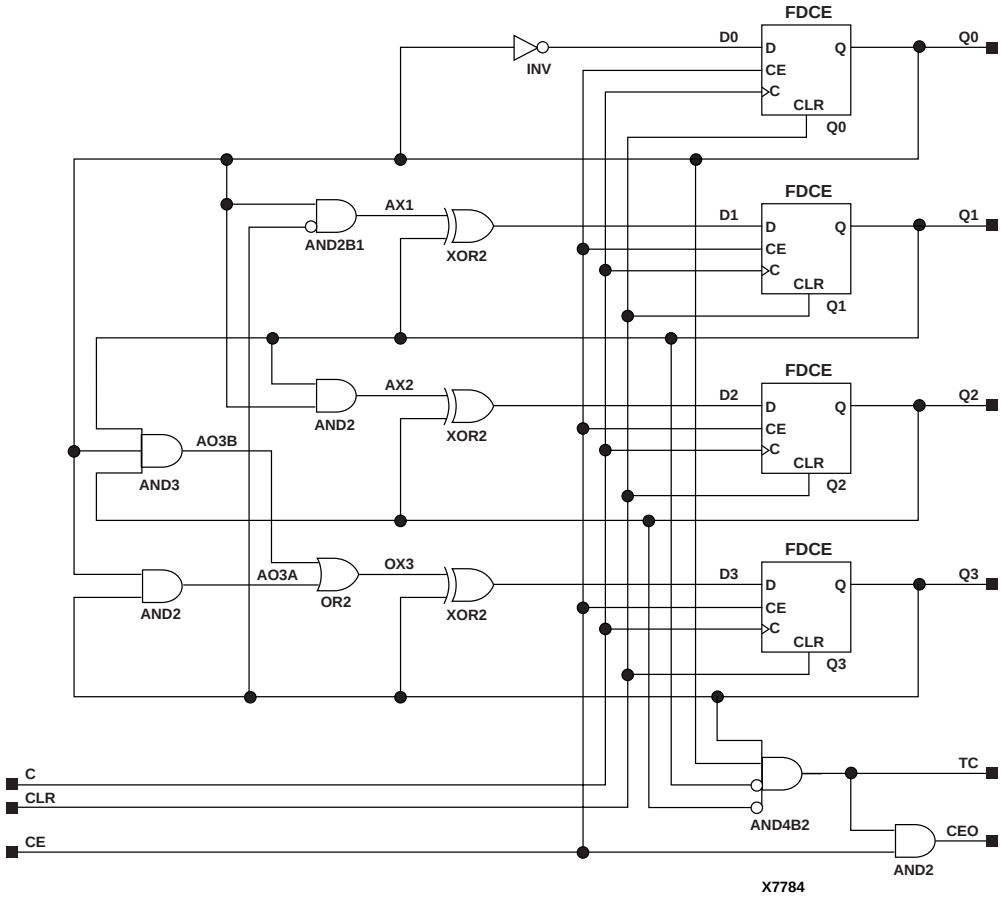


Figure 4-30 CD4CE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

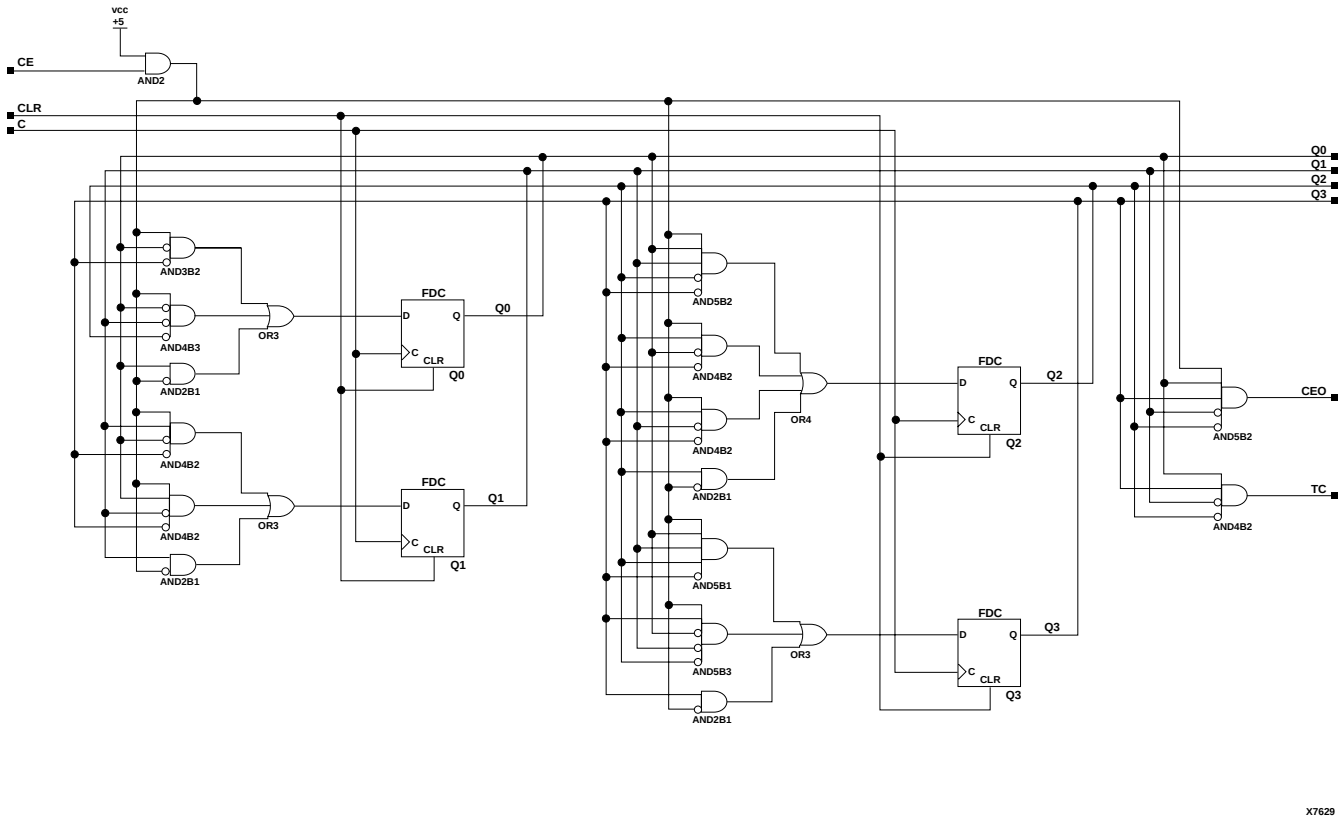
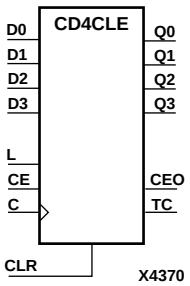


Figure 4-31 CD4CE Implementation XC9000

CD4CLE

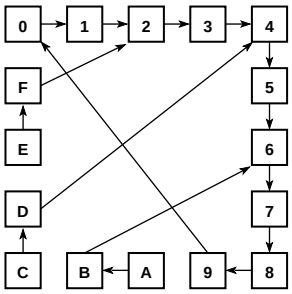
4-Bit Loadable Cascadable BCD Counter with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



CD4CLE is a 4-bit (stage), synchronously loadable, asynchronously clearable, binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition. The Q outputs increment when clock enable input (CE) is High during the Low- to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for FPGAs, as shown in the following state diagram. For XC9000, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs					
CLR	**L**	**CE**	**D3 – D0**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	X	X	X	0	0	0	0	0	0
0	1	X	D3 – D0	↑	d3	d2	d1	d0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

d = state of referenced input one setup time prior to active clock transition
TC = Q3 • $\overline{Q2}$ • $\overline{Q1}$ • Q0
CEO = TC • CE

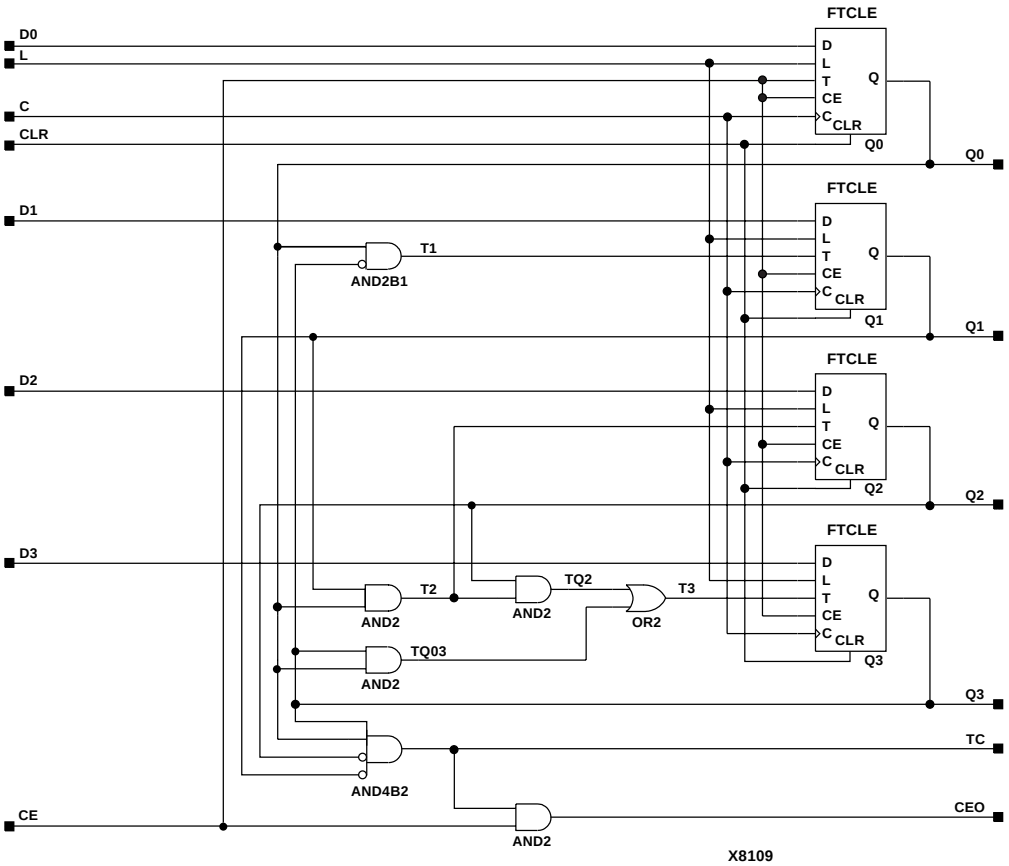


Figure 4-32 CD4CLE Implementation XC3000

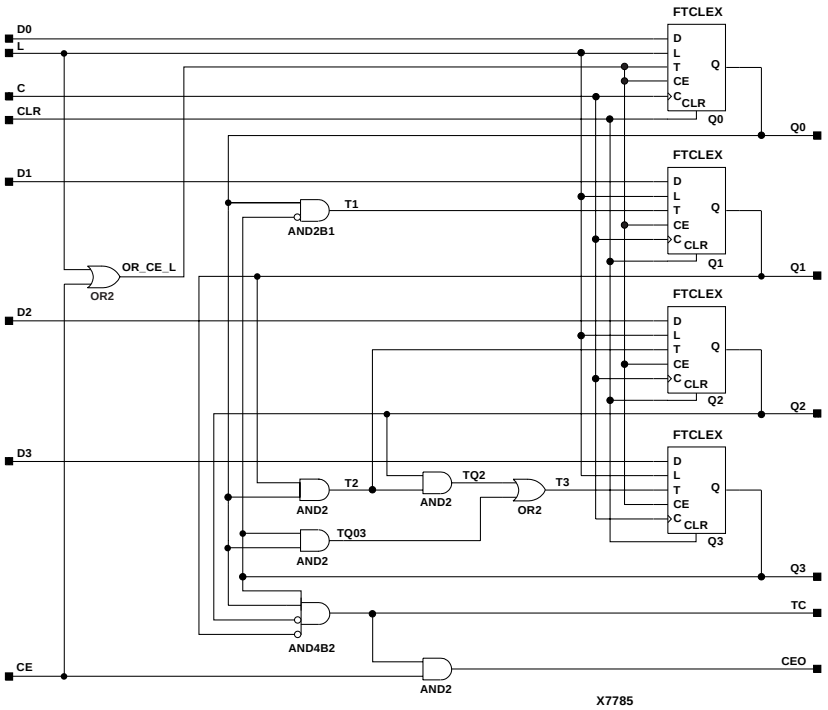


Figure 4-33 CD4CLE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

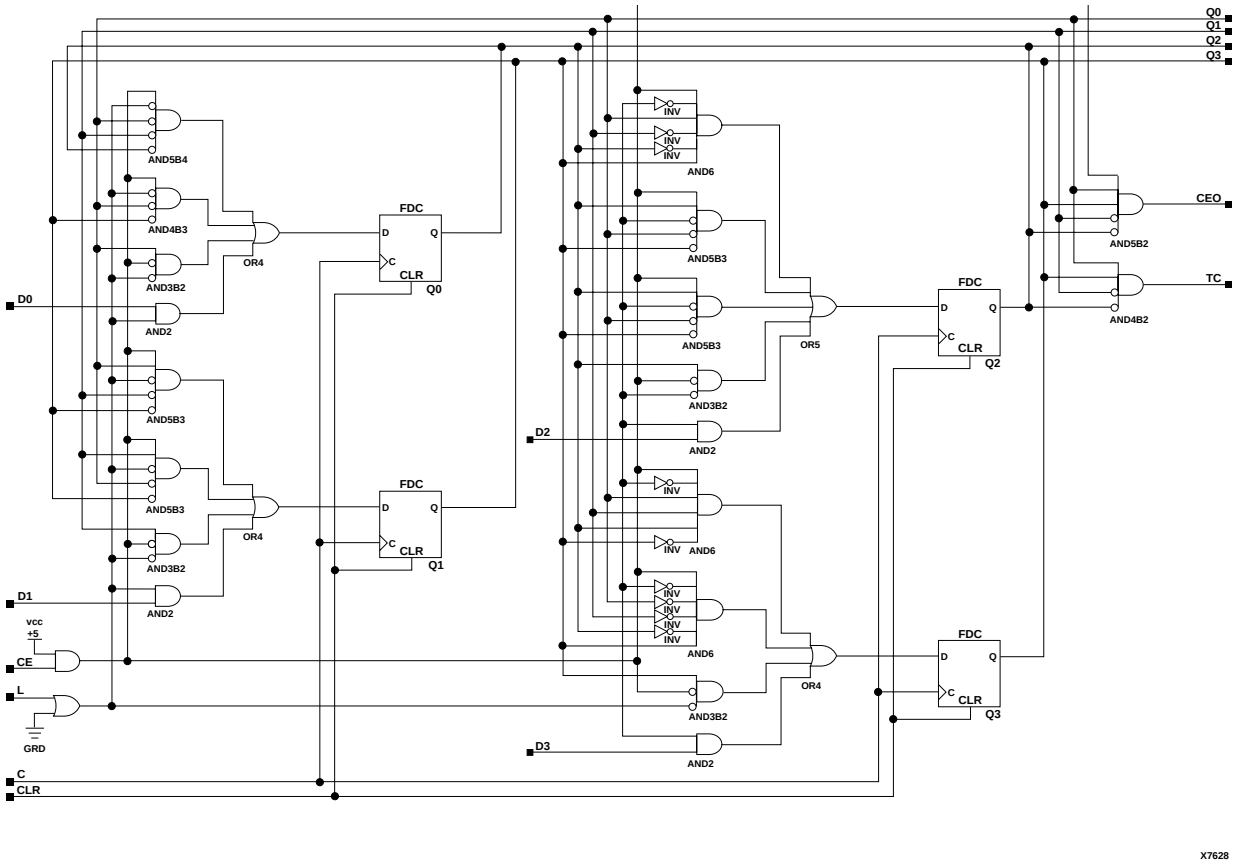
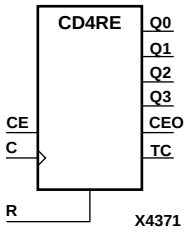


Figure 4-34 CD4CLE Implementation XC9000

CD4RE

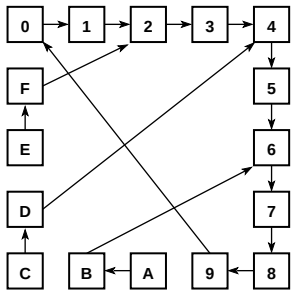
4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



CD4RE is a 4-bit (stage), synchronous, resettable, cascadable binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for FPGAs, as shown in the following state diagram. For XC9000, the counter resets to zero or recovers within the first clock cycle.

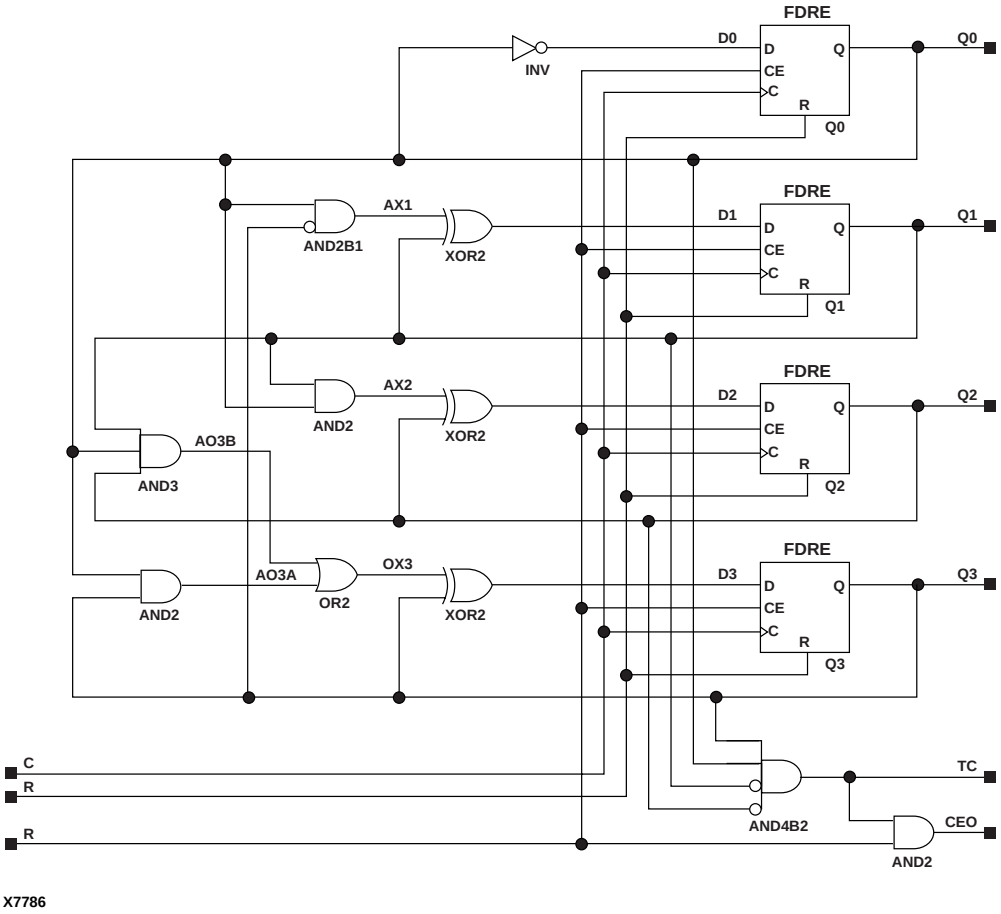


Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs					
R	**CE**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	↑	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

TC = Q3 • $\overline{Q2}$ • $\overline{Q1}$ • Q0
CEO = TC • CE



X7786

Figure 4-35 CD4RE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

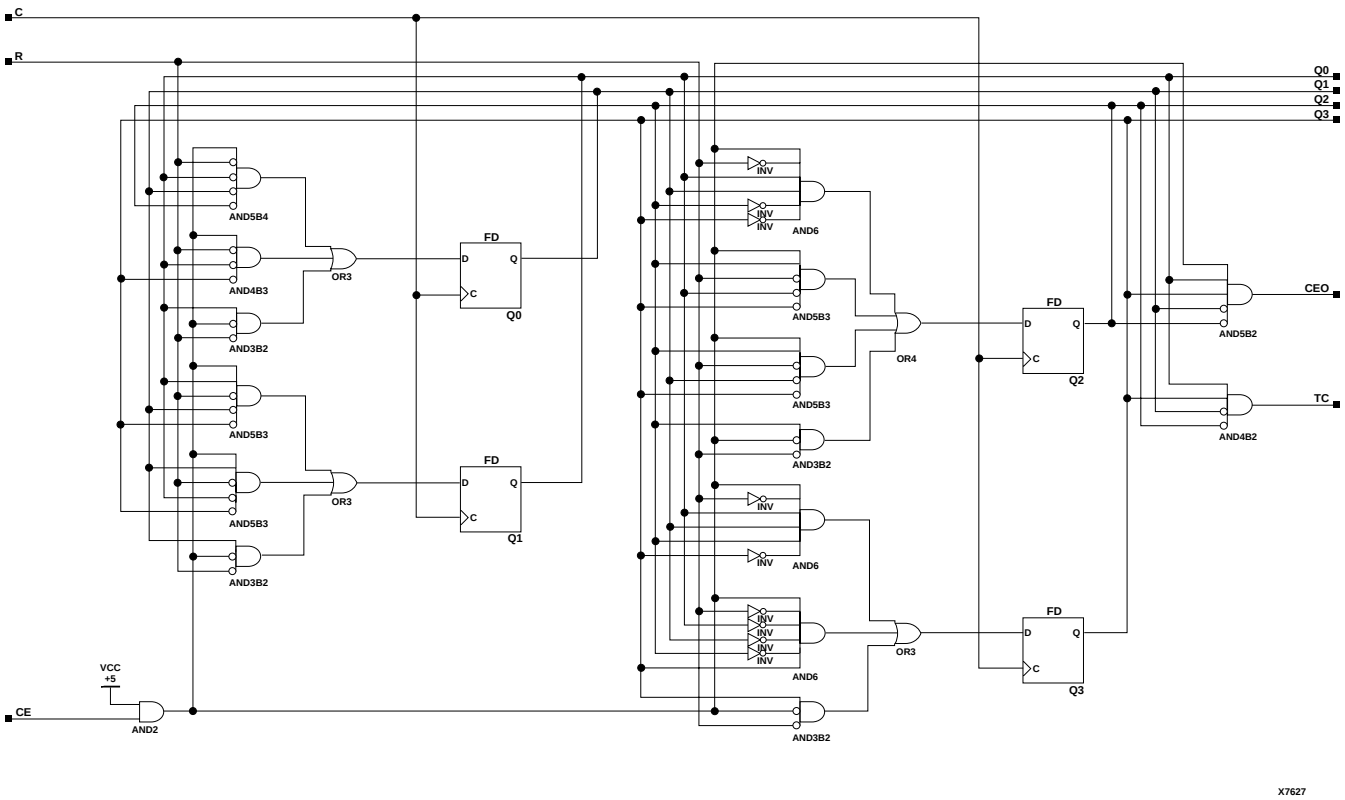
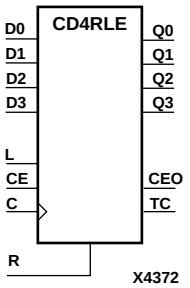


Figure 4-36 CD4RE Implementation XC9000

CD4RLE

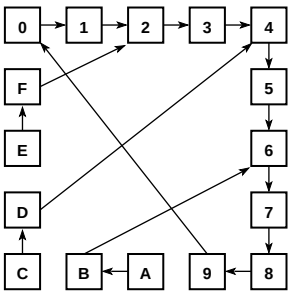
4-Bit Loadable Cascadable BCD Counter with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



CD4RLE is a 4-bit (stage), synchronous, loadable, resettable, binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for FPGAs, as shown in the following state diagram. For XC9000, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs					
R	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	↑	0	0	0	0	0	0
0	1	X	D3 – D0	↑	d3	d2	d1	d0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

d = state of referenced input one setup time prior to active clock transition
TC = Q3 • $\overline{Q2}$ • $\overline{Q1}$ • Q0
CEO = TC • CE

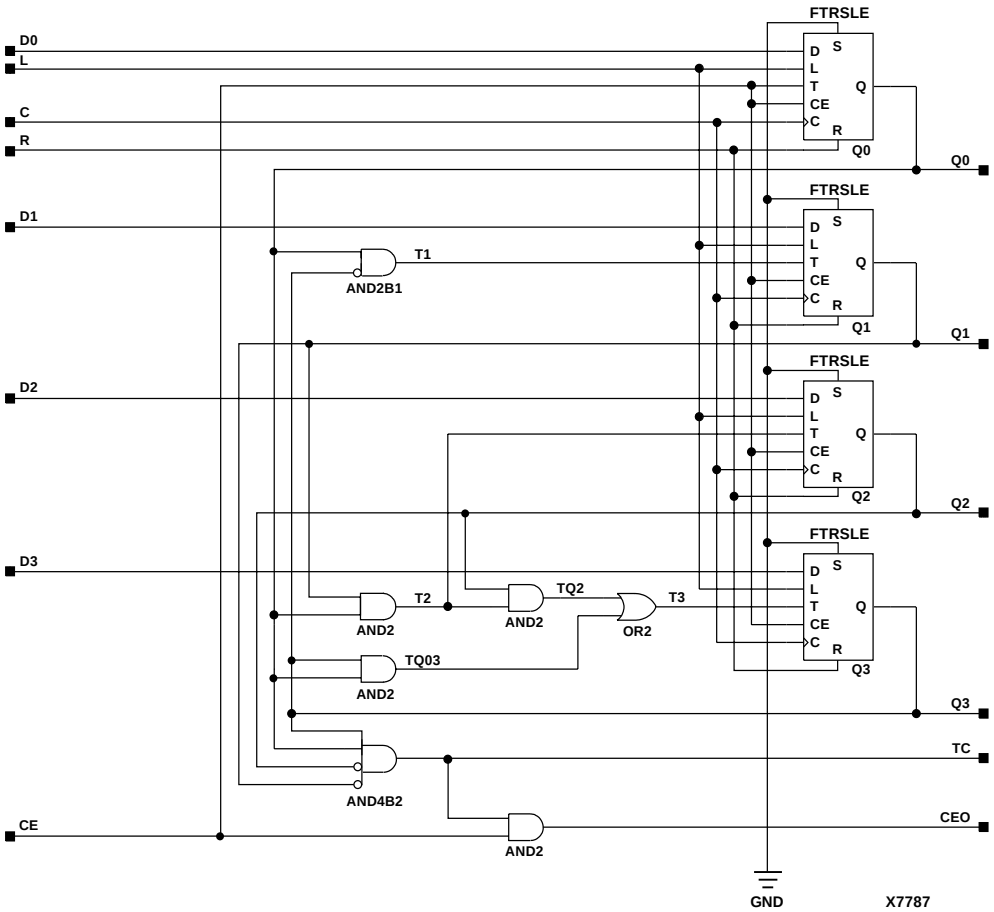


Figure 4-37 CD4RLE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

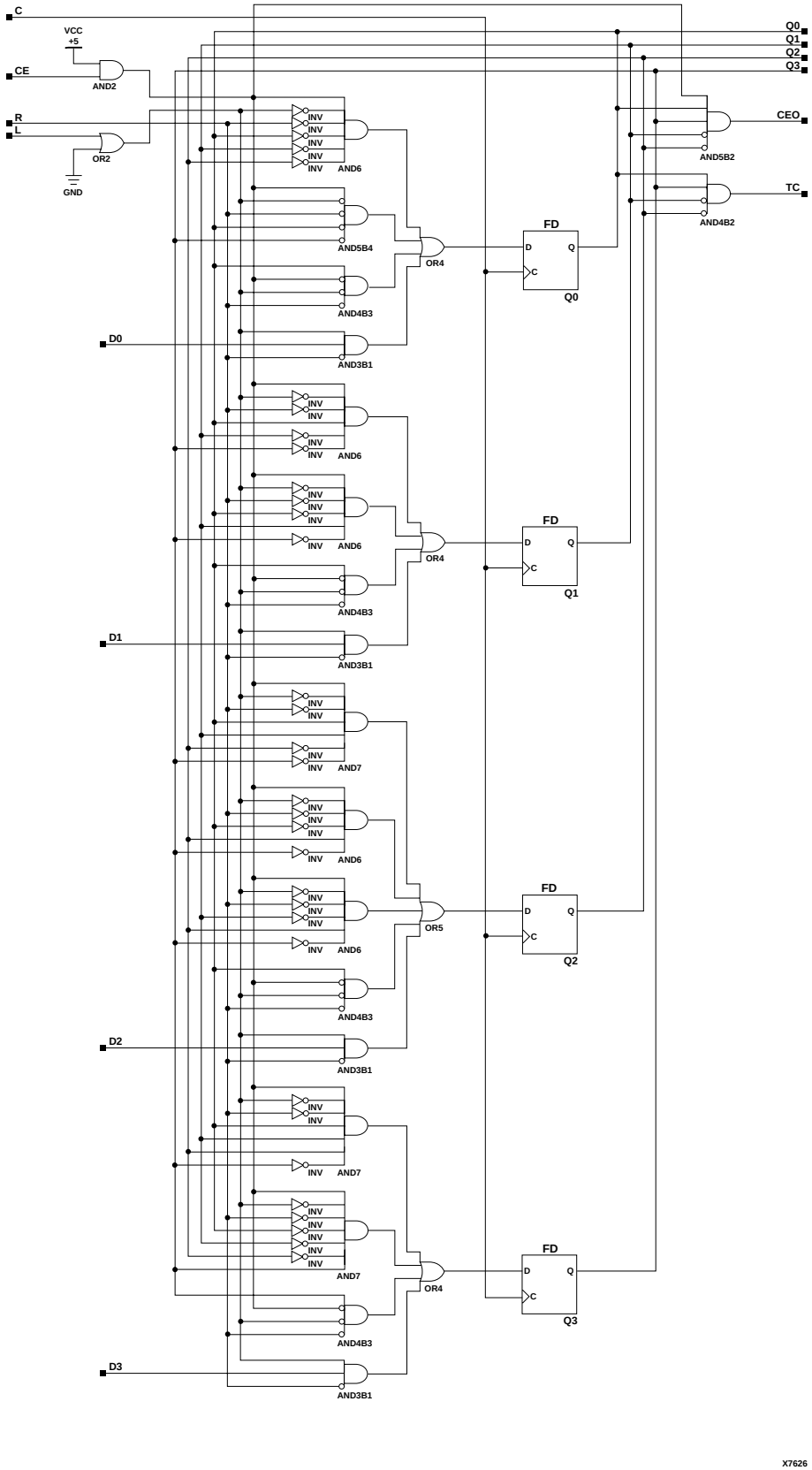
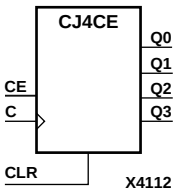


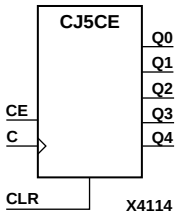
Figure 4-38 CD4RLE Implementation XC9000

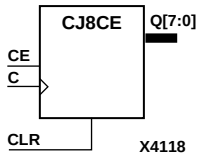
CJ4CE, CJ5CE, CJ8CE

4-, 5-, 8-Bit Johnson Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







CJ4CE, CJ5CE, and CJ8CE are clearable Johnson / shift counters. The asynchronous clear (CLR) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero, independent of clock (C) transitions. The counter increments (shifts Q0 to Q1, Q1 to Q2,and so forth) when the clock enable input (CE) is High during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

For CJ4CE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operation. For CJ5CE, the Q4 output is inverted and fed back to input Q0. For CJ8CE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Table 4-1 CJ4CE Truth Table

Inputs			Outputs			
CLR	CE	C	Q0	Q1	Q2	Q3
1	X	X	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

Table 4-2 CJ5CE Truth Table

Inputs			Outputs				
CLR	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	X	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

Table 4-3 CJ8CE Truth Table

Inputs			Outputs	
CLR	**CE**	**C**	**Q0**	**Q1 – Q7**
1	X	X	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition

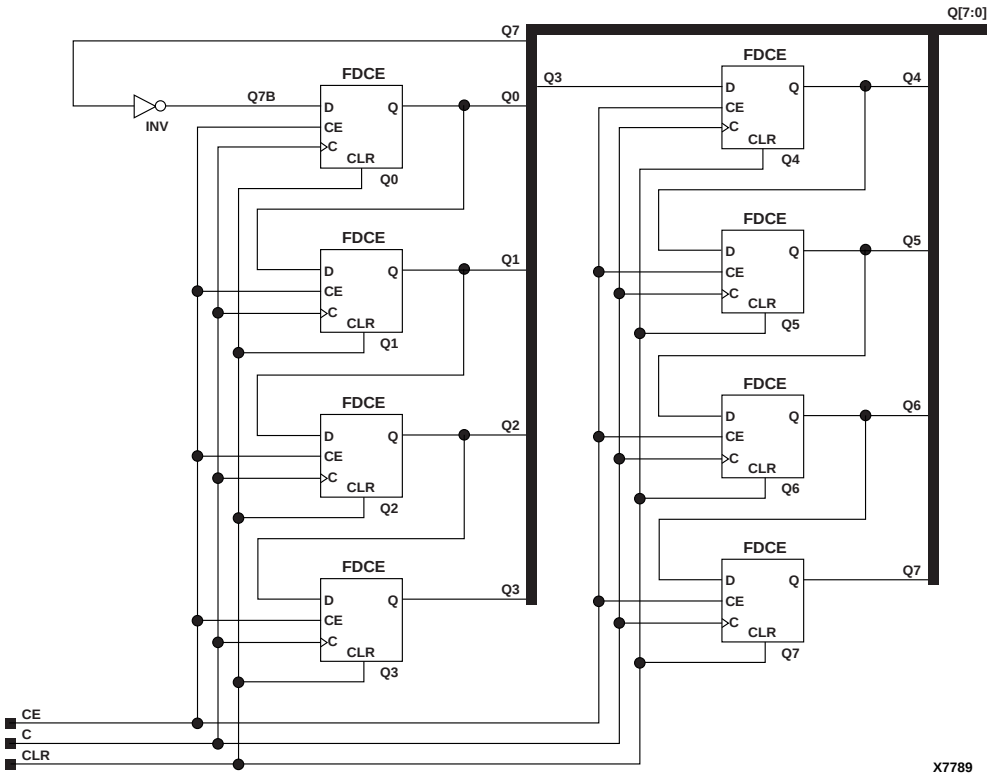
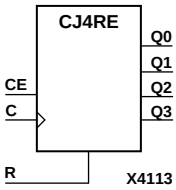


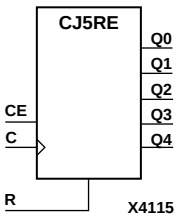
Figure 4-39 CJ8CE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

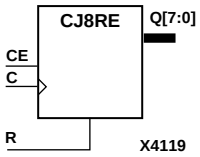
CJ4RE, CJ5RE, CJ8RE

4-, 5-, 8-Bit Johnson Counters with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







CJ4RE, CJ5RE, and CJ8RE are resettable Johnson / shift counters. The synchronous reset (R) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero during the Low-to-High clock (C) transition. The counter increments (shifts Q0 to Q1, Q1 to Q2, and so forth) when the clock enable input (CE) is High during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

For CJ4RE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operation. For CJ5RE, the Q4 output is inverted and fed back to input Q0. For CJ8RE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Table 4-4 CJ4RE Truth Table

Inputs			Outputs			
R	CE	C	Q0	Q1	Q2	Q3
1	X	↑	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

Table 4-5 CJ5RE Truth Table

Inputs			Outputs				
R	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	↑	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

Table 4-6 CJ8RE Truth Table

Inputs			Outputs	
R	**CE**	**C**	**Q0**	**Q1 – Q7**
1	X	↑	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition

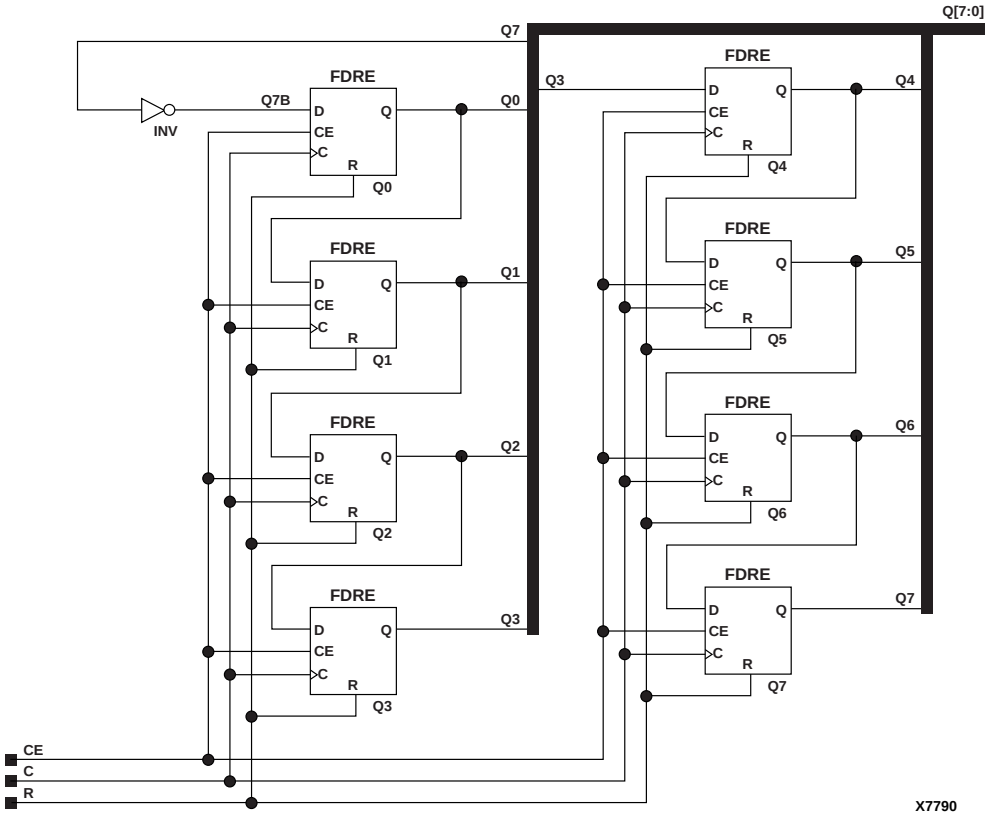
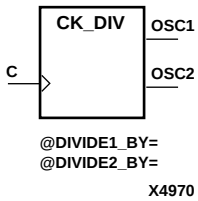


Figure 4-40 CJ8RE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

CK_DIV

Internal Multiple-Frequency Clock Divider

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



CK_DIV divides a user-provided external clock signal with different divide factors on either or both of the outputs. Only one CK_DIV may be used per design. The CK_DIV is not available if the OSC5 element is used.

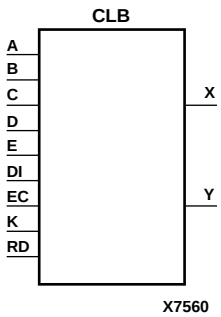
The clock frequencies of the OSC1 and OSC2 outputs are determined by specifying the DIVIDE1_BY=n_1 attribute for the OSC1 output and the DIVIDE2_BY=n_2 attribute for the OSC2 output. n_1 and n_2 are integer numbers by which the clock input (C) is divided to produce the desired output clock frequency. The available values of n_1 and n_2 are shown in the following table.

n_1	n_2
4	2
16	8
64	32
256	128
	1,024
	4,096
	16,384
	65,536

CLB

CLB Configuration Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A



The CLB symbol enables you to manually specify a CLB configuration. It allows you to enter portions of a logic design directly in terms of the physical CLB, rather than schematically. Using the CLB symbol provides precise partitioning control and requires knowledge of the CLB architecture. Use it in place of the equivalent captured logic and not in conjunction with it.

A blank XC3000 CLB primitive symbol and its corresponding configured CLB primitive and circuit are shown in the following figure.

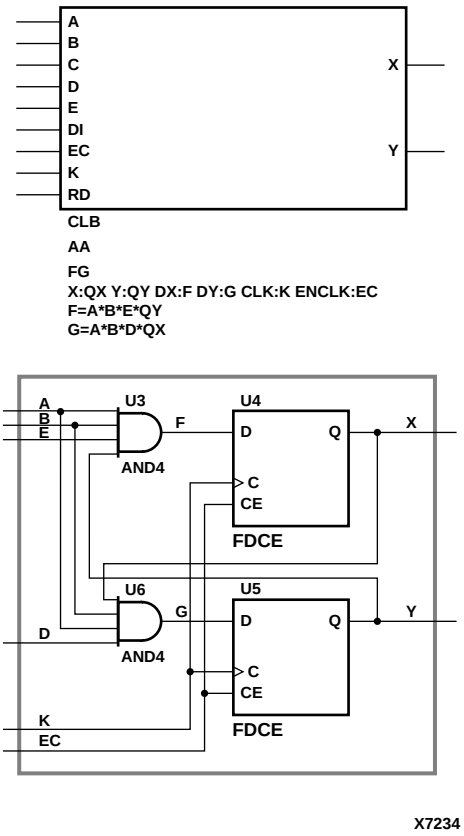


Figure 4-41 XC3000 CLB Primitive Example and Equivalent Circuit

CLB symbol pins correspond to actual CLB pins. Signals connected to these pins in a schematic are connected to the corresponding CLB pins in the design. You must specify the BASE, CONFIG, EQUATE_F, and EQUATE_G commands for the CLB. It is not necessary for the translator program to parse the commands specifying the CLB configuration. The mapping program from the LCA Xilinx netlist to the LCA design checks these commands for errors.

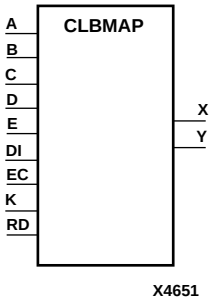
The configuration commands must be consistent with the connections. For example, if you use the A input in an equation, connect a signal to the A pin. Refer to the applicable CAE tool interface user guide for more information on specifying the CLB configuration commands in the schematic.

You can specify the location of a CLB on the device using the LOC attribute. When specifying the LOC attribute, a valid CLB name (AA, AB, and so forth) must be used. Refer to the "LOC" section of the "Attributes, Constraints, and Carry Logic" chapter for more information.

CLBMAP

Logic-Partitioning Control Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A



The CLBMAP symbol is used to control logic partitioning into XC3000 family CLBs. The CLBMAP symbol is not a substitute for logic. It is used in addition to combinational gates, latches, and flip-flops for mapping control.

At the schematic level, you can implement a portion of logic using gates, latches, and flip-flops and specify that the logic be grouped into a single CLB by using the CLBMAP symbol. You must name the signals that are the inputs and outputs of the CLB, then draw the signals to appropriate pins of the CLBMAP symbol, or name the CLBMAP signals and logic signals correspondingly. The symbol can have unconnected pins, but all signals on the logic group to be mapped must be specified on a symbol pin.

CLBMAP primitives and equivalent circuits are shown for XC3000 families in the following figure.

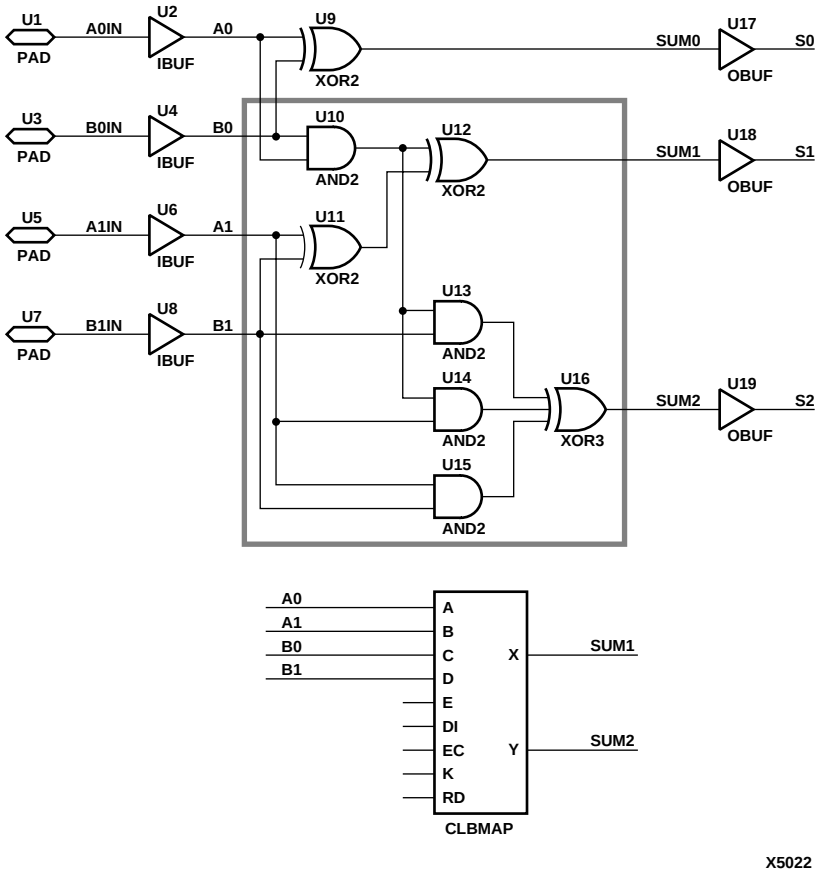


Figure 4-42 XC3000 CLBMAP Primitive Example and Equivalent

Use the MAP=*type* parameter with the CLBMAP symbol to further define how much latitude you want to give the mapping program. The following table shows MAP option characters and their meanings.

MAP Option Character	Function
P	Pins.
C	Closed — Adding logic to or removing logic from the CLB is not allowed.
L	Locked — Locking CLB pins.
O	Open — Adding logic to or removing logic from the CLB is allowed.
U	Unlocked — No locking on CLB pins.

Possible types of MAP parameters for FMAP are: MAP=PUC, MAP=PLC, MAP=PLO, and MAP=PUO. The default parameter is PUO. If one of the "open" parameters is used (PLO or PUO), only the output signals must be specified.

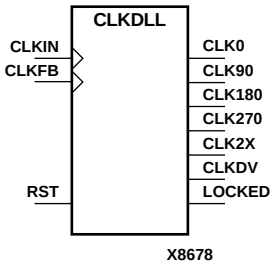
Note: Currently, only PUC and PUO are observed. PLC and PLO are translated into PUC and PUO, respectively.

You can lock individual pins using the "P" (Pin lock) parameter on the CLBMAP pin in conjunction with the PUC parameter. Refer to the appropriate CAE tool interface user guide for information on changing symbol parameters for your schematic editor.

CLKDLL

Clock Delay Locked Loop

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



CLKDLL is a clock delay locked loop used to minimize clock skew. CLKDLL synchronizes the clock signal at the feed back clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within 250 ps of each other.

The frequency of the clock signal at the CLKIN input must be in the range 25 - 90 MHz. The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG input can only be connected to the CLK0 or CLK2X output of CLKDLL. The BUFG connected to the CLKFB input of the CLKDLL must be sourced from either the CLK0 or CLK2X outputs of the same CLKDLL. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X and CLKDV outputs is always 50-50. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLL to its initial (power-on) state. The signal at the RST input is synchronized to the clock signal at the CLKIN input. The reset becomes effective at the second Low-to-High transition of the clock signal at the CLKIN input after assertion of the RST signal.

Table 4-7 CLKDLL Outputs

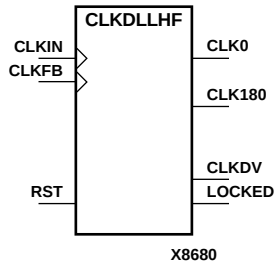
Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK90	Clock at 1x CLKIN frequency, shifted 90^{o} with regards to CLK0
CLK180	Clock at 1x CLKIN frequency, shifted 180^{o} with regards to CLK0
CLK270	Clock at 1x CLKIN frequency, shifted 270^{o} with regards to CLK0
CLK2X	Clock at 2x CLKIN frequency
CLKDV	Clock at (1/n)x CLKIN frequency, n=CLKDV_DIVIDE value
LOCKED	CLKDLL locked

Note: Refer to the "PERIOD Specifications on CLKDLLs" section of the "Using Timing Constraints" chapter in the *Development System Reference Guide* for additional information on using the TNM, TNM_NET, and PERIOD attributes with CLKDLL components.

CLKDLLHF

High Frequency Clock Delay Locked Loop

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



CLKDLLHF is a high frequency clock delay locked loop used to minimize clock skew. CLKDLLHF synchronizes the clock signal at the feed back clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within 250 ps of each other.

The frequency of the clock signal at the CLKIN input must be in the range 60 - 180 MHz. The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG input can only be connected to the CLK0 output of CLKDLLHF. The BUFG connected to the CLKFB input of the CLKDLLHF must be sourced from the CLK0 output of the same CLKDLLHF. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Only the CLK0 output can be used. CLK0 must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted output (CLK180) is the same as that of the CLK0 output. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLL to its initial (power-on) state. The signal at the RST input is synchronized to the clock signal at the CLKIN input. The reset becomes effective at the second Low-to-High transition of the clock signal at the CLKIN input after assertion of the RST signal.

Table 4-8 CLKDLLHF Outputs

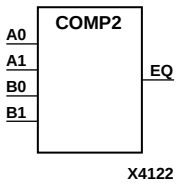
Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLKIN frequency, shifted 180^{o} with regards to CLK0
CLKDV	Clock at (1/n)x CLKIN frequency, n=CLKDV_DIVIDE value
LOCKED	CLKDLL locked

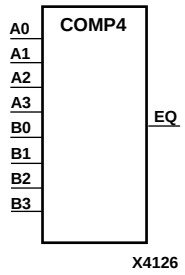
Note: Refer to the "PERIOD Specifications on CLKDLLs" section of the "Using Timing Constraints" chapter in the *Development System Reference Guide* for additional information on using the TNM, TNM_NET, and PERIOD attributes with CLKDLLHF components.

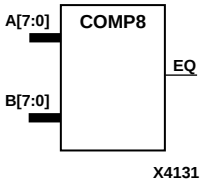
COMP2, 4, 8, 16

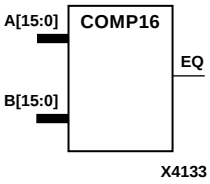
2-, 4-, 8-, 16-Bit Identity Comparators

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









COMP2, COMP4, COMP8, and COMP16 are, respectively, 2-, 4-, 8-, and 16-bit identity comparators. The equal output (EQ) of the COMP2 2-bit, identity comparator is High when the two words A1 – A0 and B1 – B0 are equal. EQ is high for COMP4 when A3 – A0 and B3 – B0 are equal; for COMP8, when A7 – A0 and B7 – B0 are equal; and for COMP16, when A15 – A0 and B15 – B0 are equal.

Equality is determined by a bit comparison of the two words. When any two of the corresponding bits from each word are not the same, the EQ output is Low.

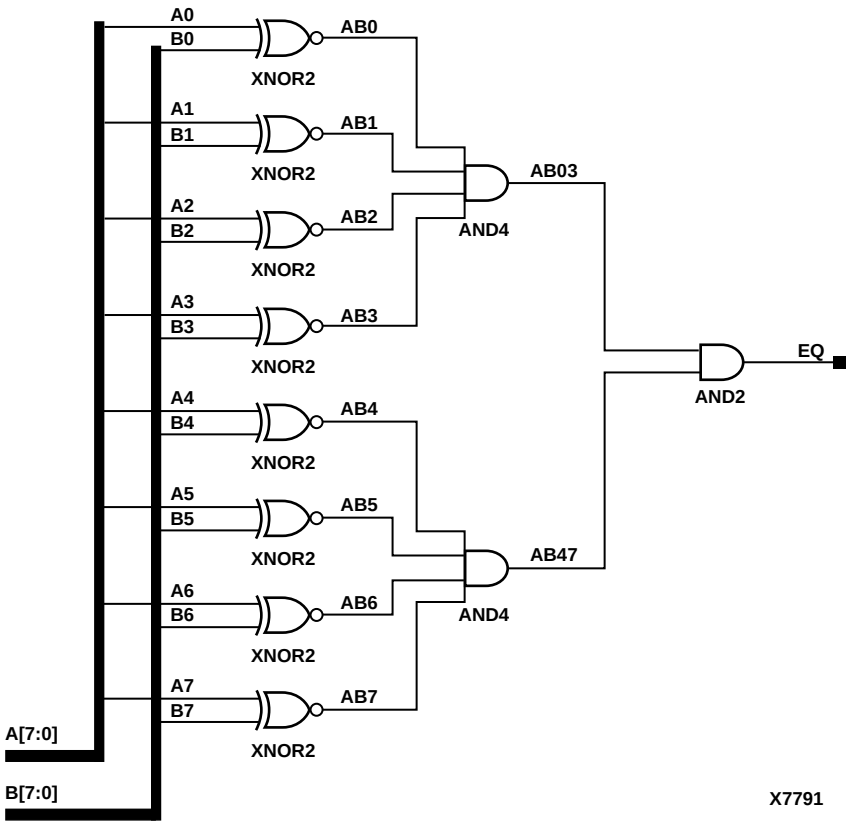
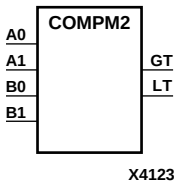


Figure 4-43 COMP8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

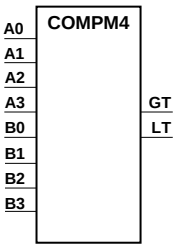
COMPM2, 4, 8, 16

2-, 4-, 8-, 16-Bit Magnitude Comparators

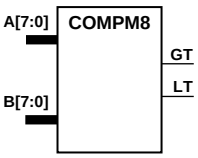
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



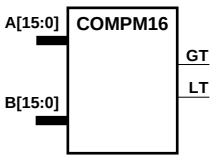
X4123



X4127



X4132



X4134

COMPM2, COMPM4, COMPM8, and COMPM16 are, respectively, 2-, 4-, 8-, and 16-bit magnitude comparators that compare two positive binary-weighted words.

COMPM2 compares A1 – A0 and B1 – B0, where A1 and B1 are the most significant bits. COMPM4 compares A3 – A0 and B3 – B0, where A3 and B3 are the most significant bits. COMPM8 compares A7 – A0 and B7 – B0, where A7 and B7 are the most significant bits. COMPM16 compares A15 – A0 and B15 – B0, where A15 and B15 are the most significant bits.

The greater-than output (GT) is High when A>B, and the less-than output (LT) is High when A<B. When the two words are equal, both GT and LT are Low. Equality can be measured with this macro by comparing both outputs with a NOR gate.

Table 4-9 COMPM2 Truth Table

Inputs				Outputs	
A1	B1	A0	B0	GT	LT
0	0	0	0	0	0
0	0	1	0	1	0
0	0	0	1	0	1
0	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	0
1	1	0	1	0	1
1	1	1	1	0	0
1	0	X	X	1	0
0	1	X	X	0	1

Table 4-10 COMPM4 Truth Table

Inputs				Outputs	
A3, B3	**A2, B2**	**A1, B1**	**A0, B0**	**GT**	**LT**
A3>B3	X	X	X	1	0
A3<B3	X	X	X	0	1
A3=B3	A2>B2	X	X	1	0
A3=B3	A2<B2	X	X	0	1
A3=B3	A2=B2	A1>B1	X	1	0
A3=B3	A2=B2	A1<B1	X	0	1
A3=B3	A2=A2	A1=B1	A0>B0	1	0
A3=B3	A2=B2	A1=B1	A0<B0	0	1
A3=B3	A2=B2	A1=B1	A0=B0	0	0

Table 4-11 COMPM8 Truth Table (also representative of COMPM16)

Inputs								Outputs	
A7, B7	**A6, B6**	**A5, B5**	**A4, B4**	**A3, B3**	**A2, B2**	**A1, B1**	**A0, B0**	**GT**	**LT**
A7>B7	X	X	X	X	X	X	X	1	0
A7<B7	X	X	X	X	X	X	X	0	1
A7=B7	A6>B6	X	X	X	X	X	X	1	0
A7=B7	A6<B6	X	X	X	X	X	X	0	1
A7=B7	A6=B6	A5>B5	X	X	X	X	X	1	0
A7=B7	A6=B6	A5<B5	X	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4>B4	X	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4<B4	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3>B3	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3<B3	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2>B2	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2<B2	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1>B1	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1<B1	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0>B0	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0<B0	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0=B0	0	0

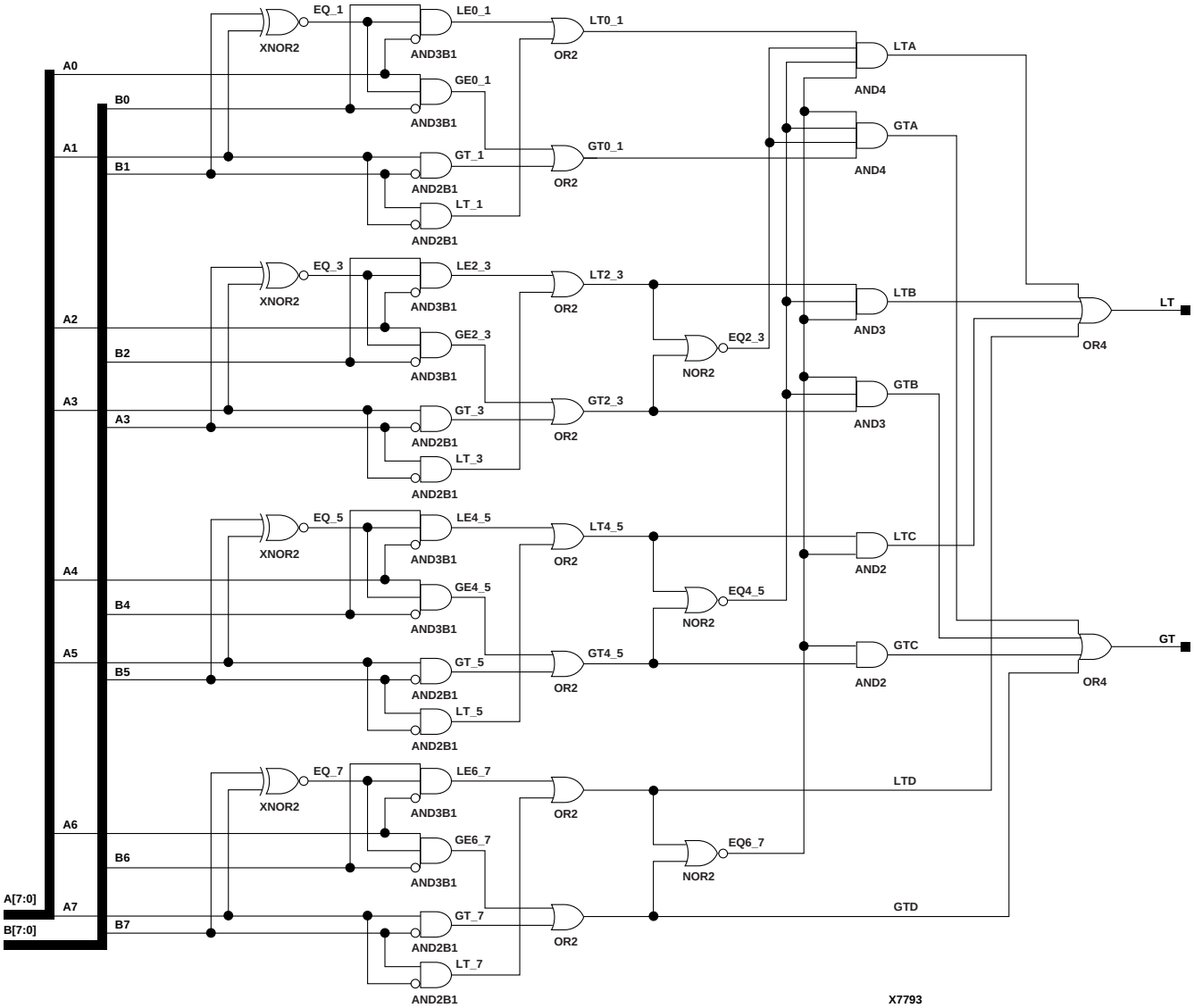


Figure 4-44 COMPM8 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

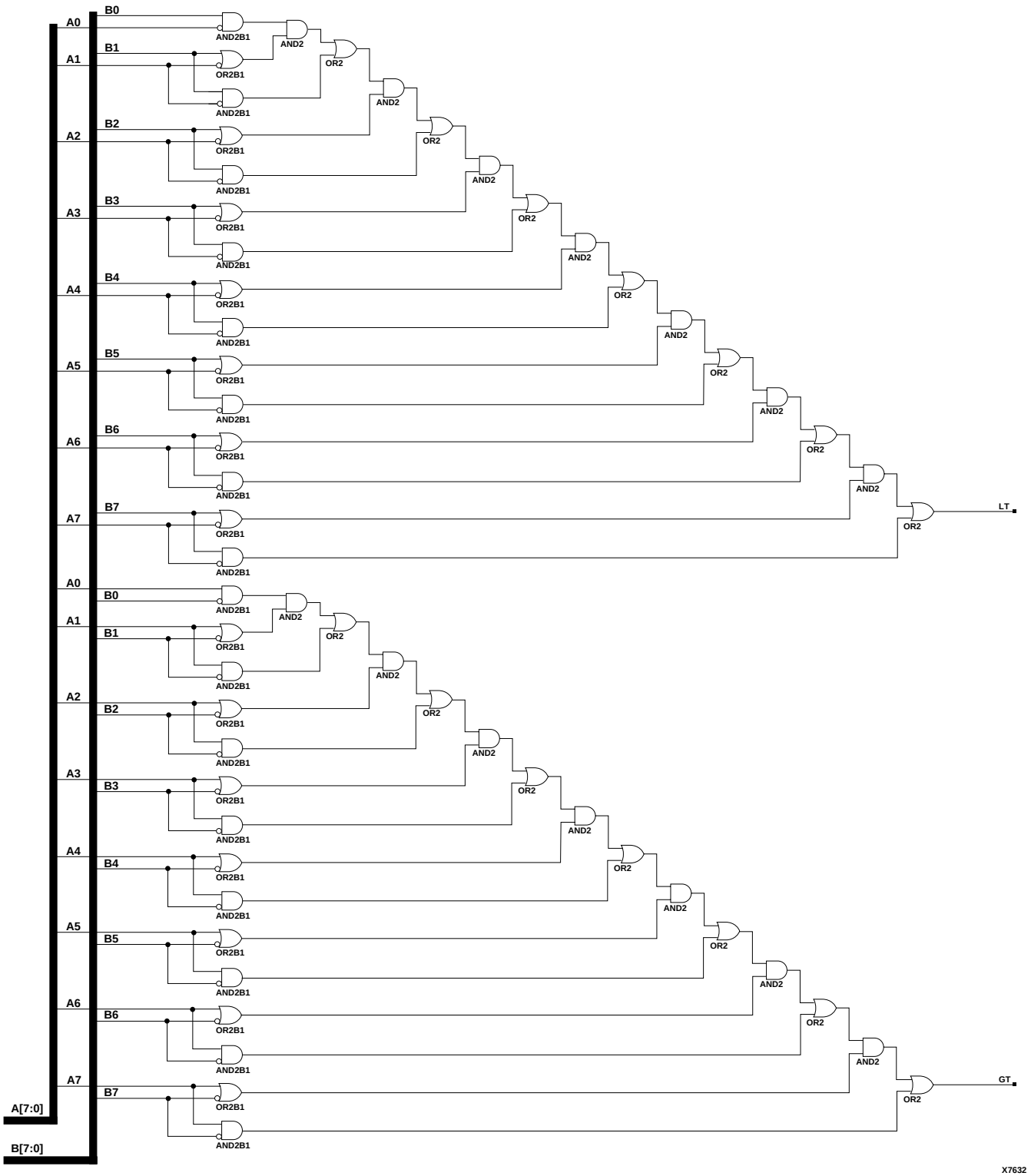
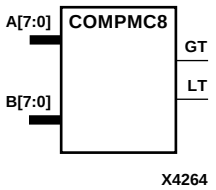


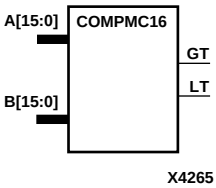
Figure 4-45 COMPM8 Implementation XC9000

COMPMC8, 16

8-, 16-Bit Magnitude Comparators

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	N / A	Macro	Macro	Macro	Macro





COMPMC8 is an 8-bit, magnitude comparator that compares two positive binary-weighted words A7 – A0 and B7 – B0, where A7 and B7 are the most significant bits. COMPMC16 is a 16-bit, magnitude comparator that compares two positive binary-weighted words A15 – A0 and B15 – B0, where A15 and B15 are the most significant bits.

These comparators are implemented using carry logic with relative location constraints to ensure efficient logic placement.

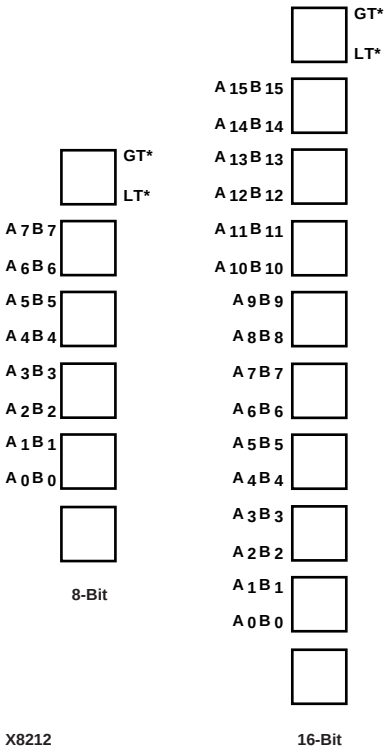
The greater-than output (GT) is High when A>B, and the less-than output (LT) is High when A<B. When the two words are equal, both GT and LT are Low. Equality can be flagged with this macro by connecting both outputs to a NOR gate.

Table 4-12 COMPMC8 Truth Table (also representative of COMPMC16)

Inputs								Outputs	
A7, B7	A6, B6	A5, B5	A4, B4	A3, B3	A2, B2	A1, B1	A0, B0	GT	LT
A7>B7	X	X	X	X	X	X	X	1	0
A7<B7	X	X	X	X	X	X	X	0	1
A7=B7	A6>B6	X	X	X	X	X	X	1	0
A7=B7	A6<B6	X	X	X	X	X	X	0	1
A7=B7	A6=B6	A5>B5	X	X	X	X	X	1	0
A7=B7	A6=B6	A5<B5	X	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4>B4	X	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4<B4	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3>B3	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3<B3	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2>B2	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2<B2	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1>B1	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1<B1	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0>B0	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0<B0	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0=B0	0	0

Topology for XC4000, Spartan, SpartanXL

This is the COMPMC8 (8-bit) and COMPMC16 (16-bit) topology for XC4000E, XC4000X, Spartan, and SpartanXL devices.



In the process of combining the logic that loads GT and LT, the place and route software might map the logic that generates GT and LT to different function generators. If this mapping occurs, the GT and LT logic cannot be placed in the uppermost CLB, as indicated in the illustration.

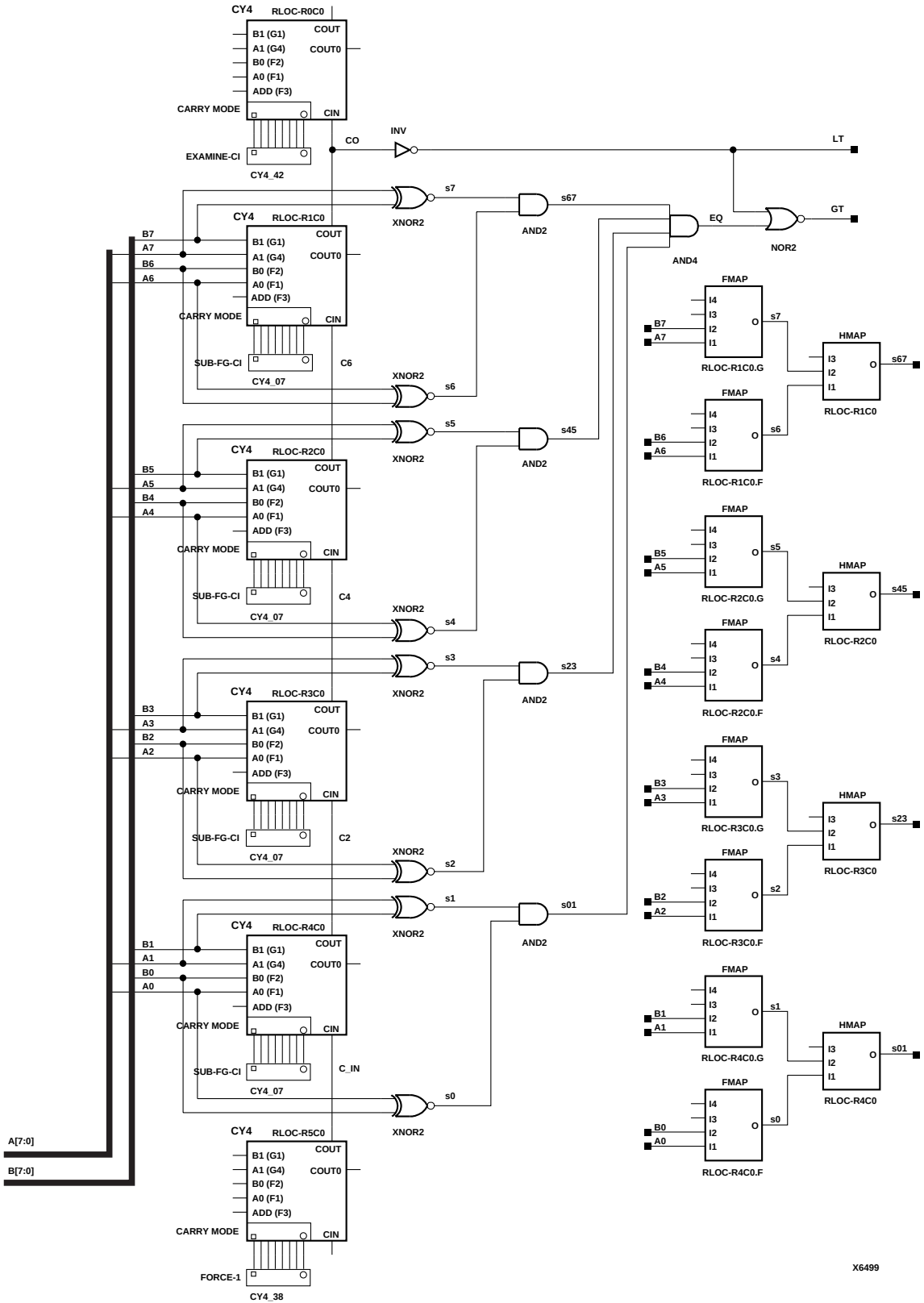


Figure 4-46 COMPMC8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

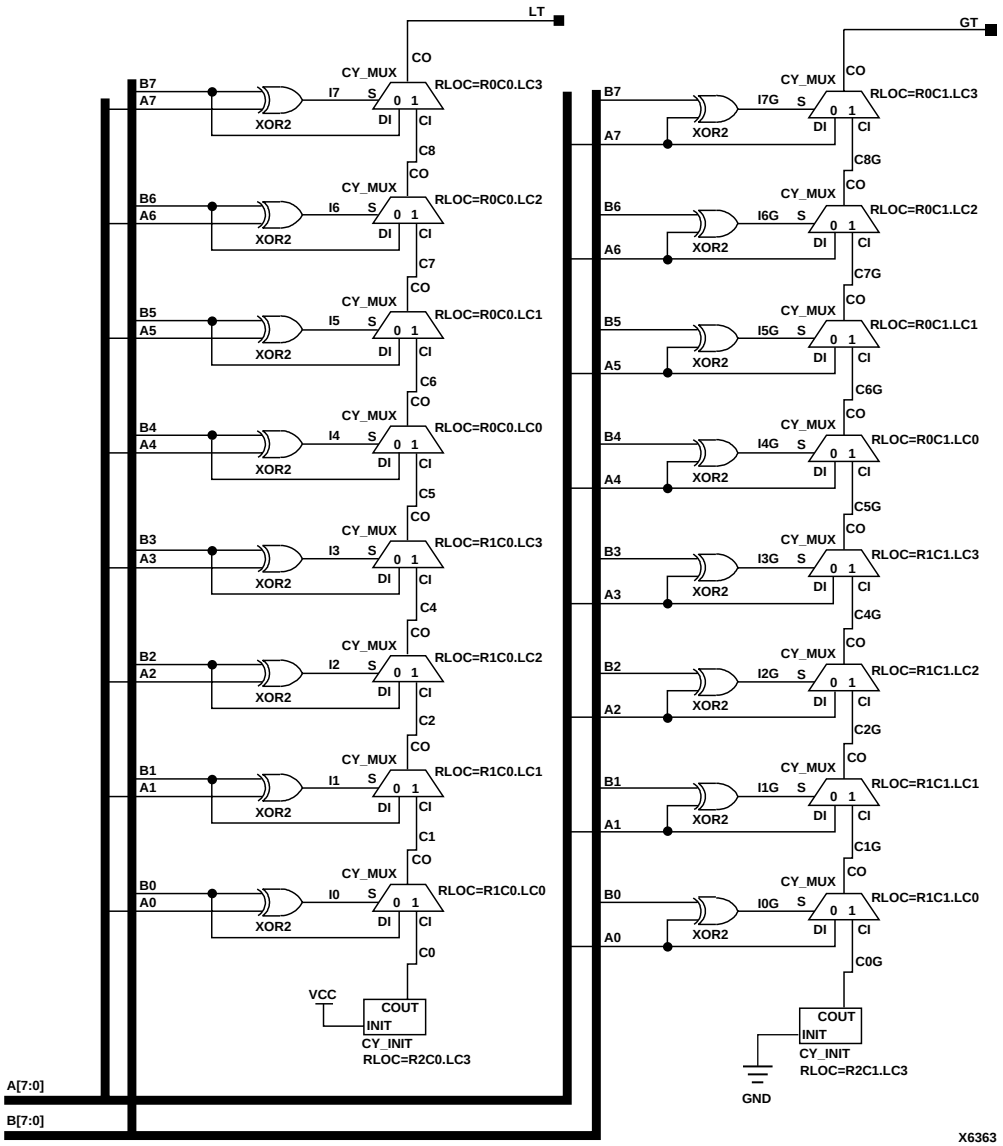


Figure 4-47 COMPMC8 Implementation XC5200

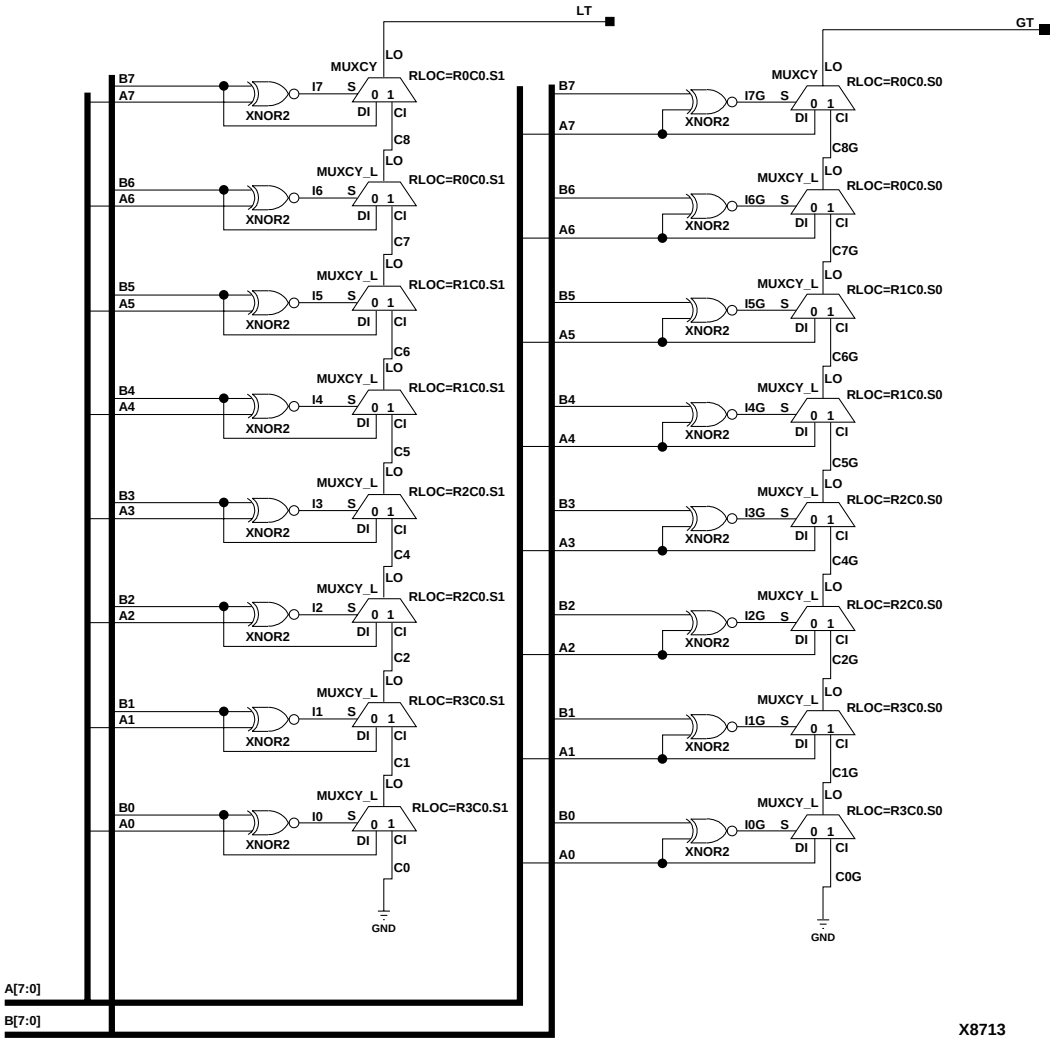


Figure 4-48 COMPMC8 Implementation Spartan2, Virtex

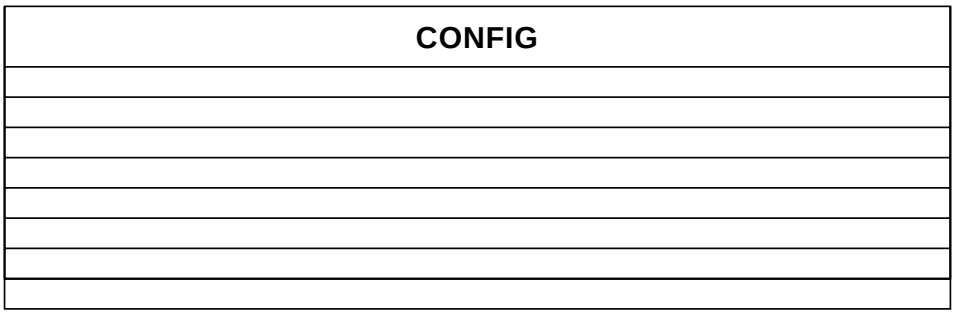
CONFIG

Repository for Schematic-Level (Global) Attributes

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

The CONFIG primitive is a table that you can use to specify up to eight attributes that affect the entire design (global attributes such as PART or PROHIBIT).

When using certain CAE software packages, global properties cannot be attached to the “Schematic” or “Sheet.” Instead, they must be attached to the CONFIG symbol. Enter attributes using the same syntax that you would use in a UCF file. The global attributes can be any length, but only 30 characters are displayed in the CONFIG window. The CONFIG table is shown in the following figure.

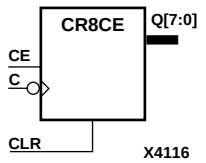


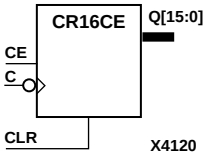
X7763

CR8CE, CR16CE

8-, 16-Bit Negative-Edge Binary Ripple Counters with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro





CR8CE and CR16CE are 8-bit and 16-bit, cascadable, clearable, binary, ripple counters. The asynchronous clear (CLR), when High, overrides all other inputs and causes the Q outputs to go to logic level zero. The counter increments when the clock enable input (CE) is High during the High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low.

Larger counters can be created by connecting the last Q output (Q7 for CR8CE, Q15 for CR16CE) of the first stage to the clock input of the next stage. CLR and CE inputs are connected in parallel. The clock period is not affected by the overall length of a ripple counter. The overall clock-to-output propagation is $n(t_{C-Q})$, where n is the number of stages and the time t_{C-Q} is the C-to-Qz propagation delay of each stage.

The counter is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CLR	CE	C	Qz – Q0
1	X	X	0
0	0	X	No Chg
0	1	↓	Inc

z = 7 for CR8CE; z = 15 for CR16CE.

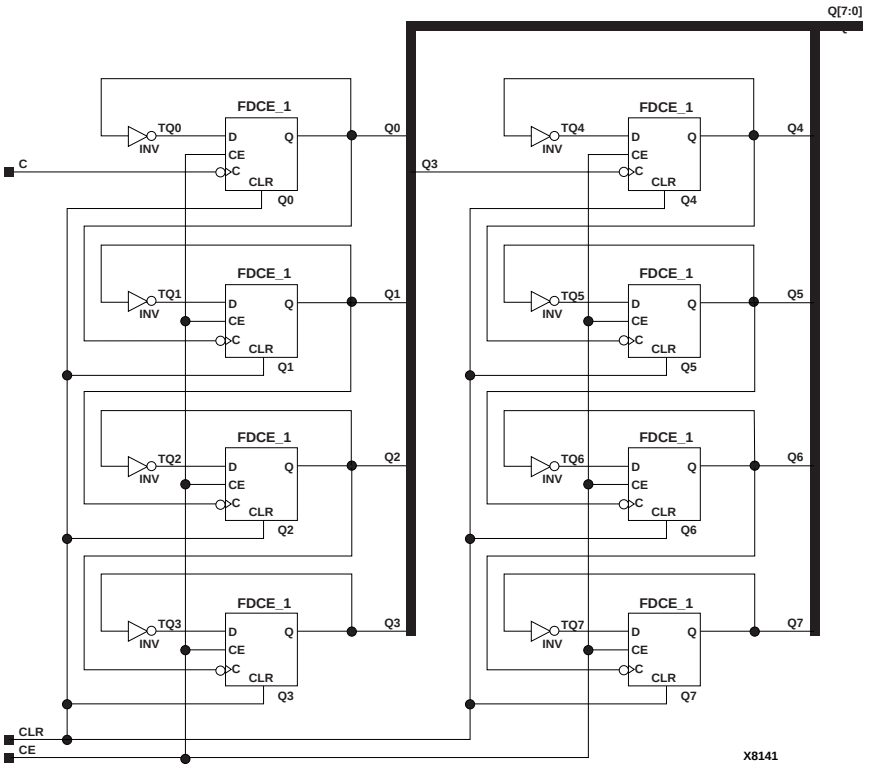


Figure 4-49 CR8CE Implementation XC3000

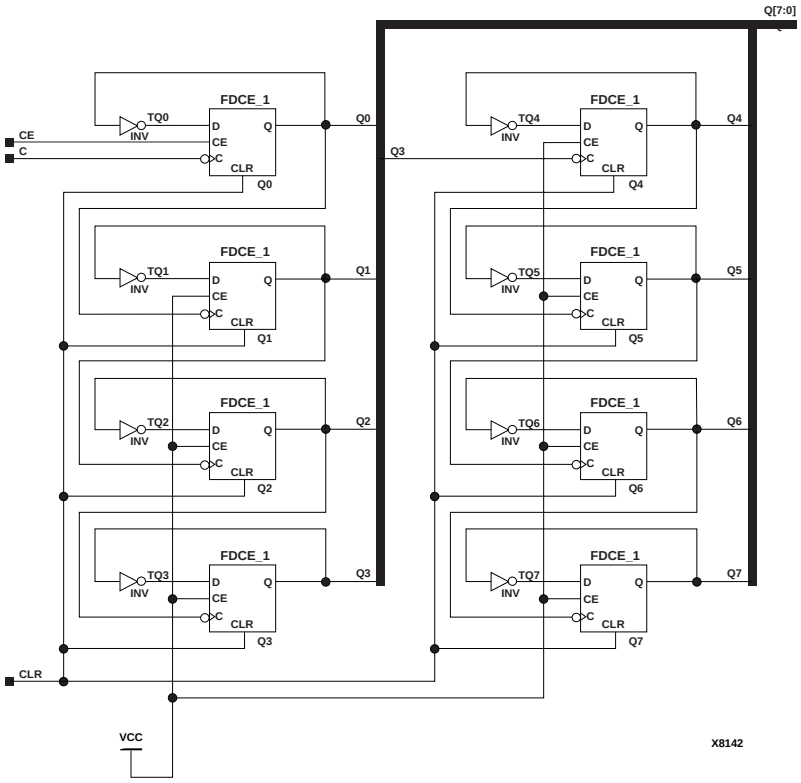


Figure 4-50 CR8CE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

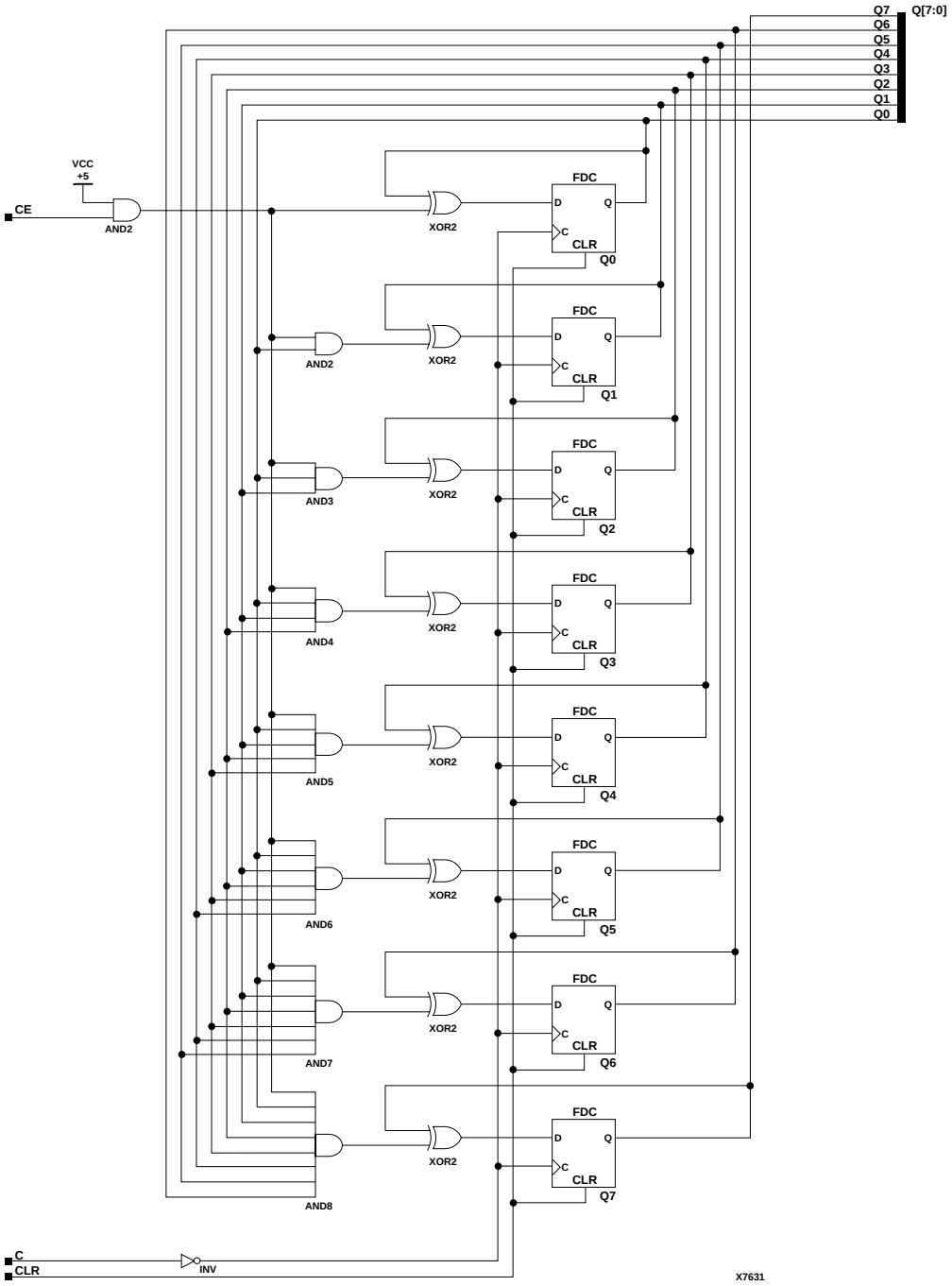
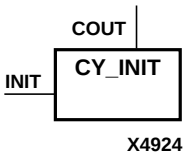


Figure 4-51 CR8CE Implementation XC9000

CY_INIT

Initialization Stage for Carry Chain

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Macro	N/A	N/A	N/A	N/A	N/A



CY_INIT is used to initialize the carry chain in the XC5200 architecture. It is used in conjunction with multiple CY_MUX elements to implement high speed carry-propagate or high speed cascade logic. CY_INIT must be placed in the logic cell (LC) immediately below the least-significant carry element (CY_MUX) in the carry/cascade chain. The INIT input is driven from the direct input (DI) to LC. The CY_INIT carry-out (COUT) drives the C_{in} input of the first LC in the carry chain. The COUT output reflects the state of the DI input. This figure represents the schematic implementation of CY_INIT.

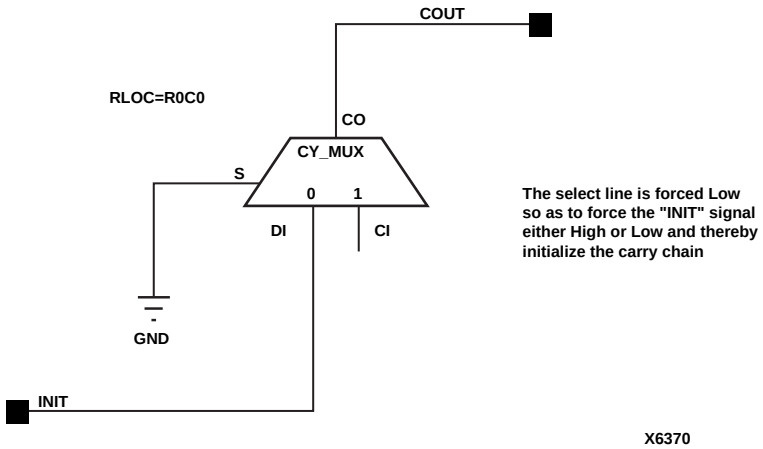
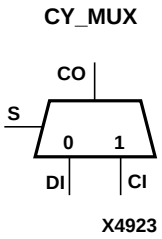


Figure 4-52 CY _INIT 4-Bit Adder Implementation XC5200

CY_MUX

2-to-1 Multiplexer for Carry Logic

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



CY_MUX is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the CY_MUX. The carry in (CI) input of an LC is connected to the CI input of the CY_MUX. The select input (S) of the CY_MUX is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (CO) of the CY_MUX reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

Inputs			Outputs
S	DI	CI	CO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

The following figure depicts the application of the CY_MUX for a 4-bit adder. Also shown are the associated FMAP symbols and the CY_INIT function.

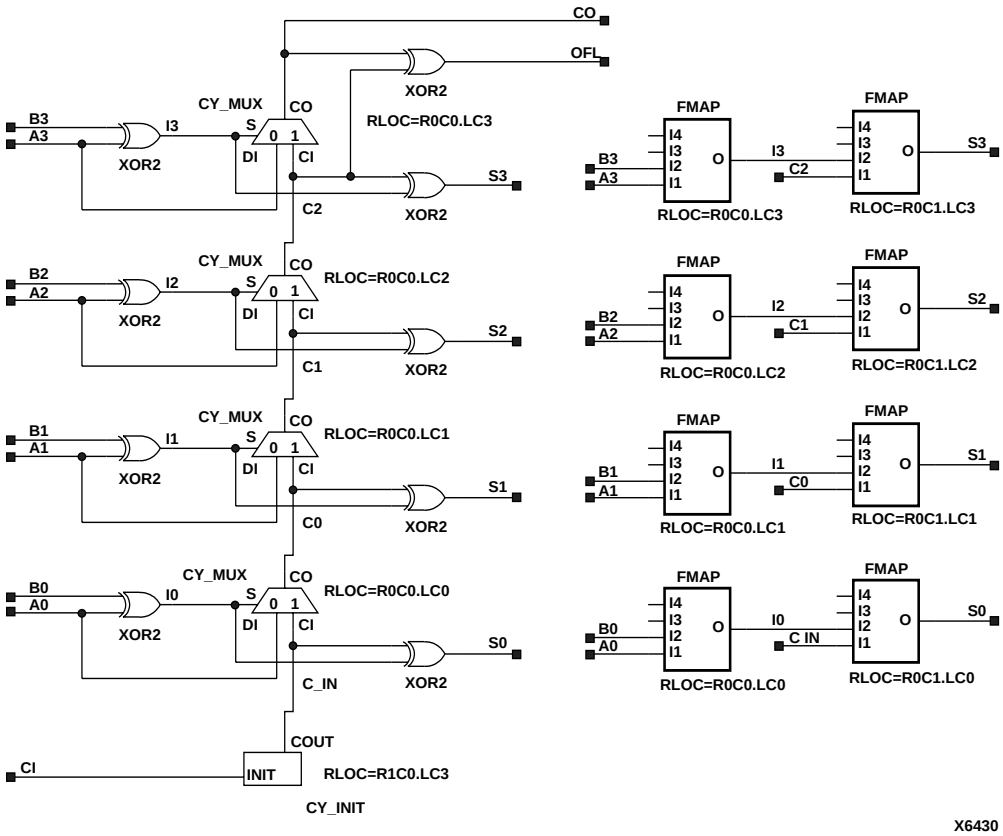
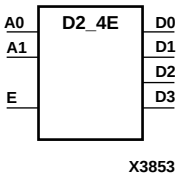


Figure 4-53 CY_MUX 4-Bit Adder Schematic XC5200

D2_4E

2- to 4-Line Decoder/Demultiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



When the enable (EN) input of the D2_4E decoder / demultiplexer is High, one of four active-High outputs (D3 – D0) is selected with a 2-bit binary address (A1 – A0) input. The non-selected outputs are Low. Also, when the EN input is Low, all outputs are Low. In demultiplexer applications, the EN input is the data input.

Inputs			Outputs			
A1	A0	E	D3	D2	D1	D0
X	X	0	0	0	0	0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0

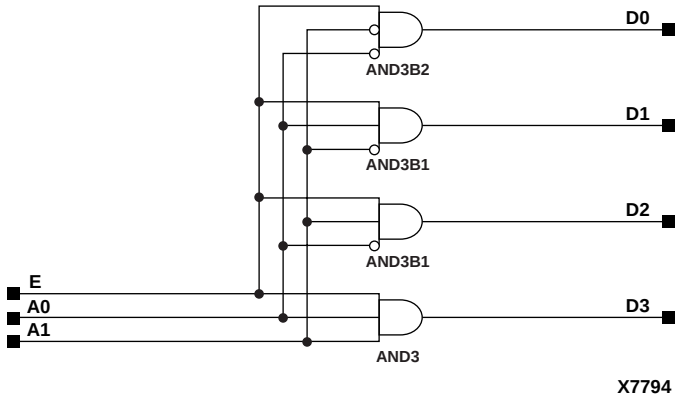
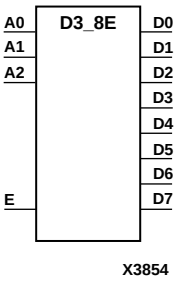


Figure 4-54 D2_4E Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

D3_8E

3- to 8-Line Decoder/Demultiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



When the enable (EN) input of the D3_8E decoder / demultiplexer is High, one of eight active-High outputs (D7 – D0) is selected with a 3-bit binary address (A2 – A0) input. The non-selected outputs are Low. Also, when the EN input is Low, all outputs are Low. In demultiplexer applications, the EN input is the data input.

Inputs				Outputs							
A2	A1	A0	E	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

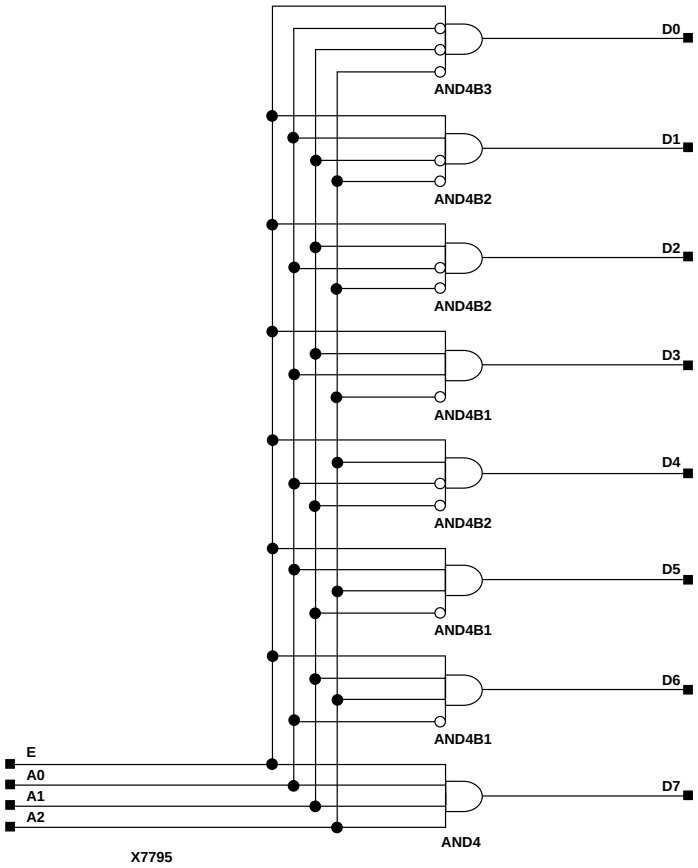
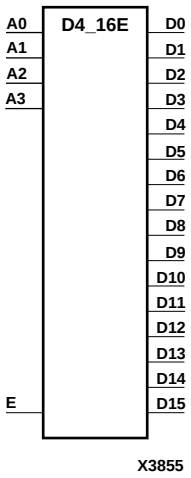


Figure 4-55 D3_8E Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

D4_16E

4- to 16-Line Decoder/Demultiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



When the enable (EN) input of the D4_16E decoder / demultiplexer is High, one of 16 active-High outputs (D15 – D0) is selected with a 4-bit binary address (A3 – A0) input. The non-selected outputs are Low. Also, when the EN input is Low, all outputs are Low. In demultiplexer applications, the EN input is the data input.

Refer to the “D3_8E” section for a representative truth table derivation.

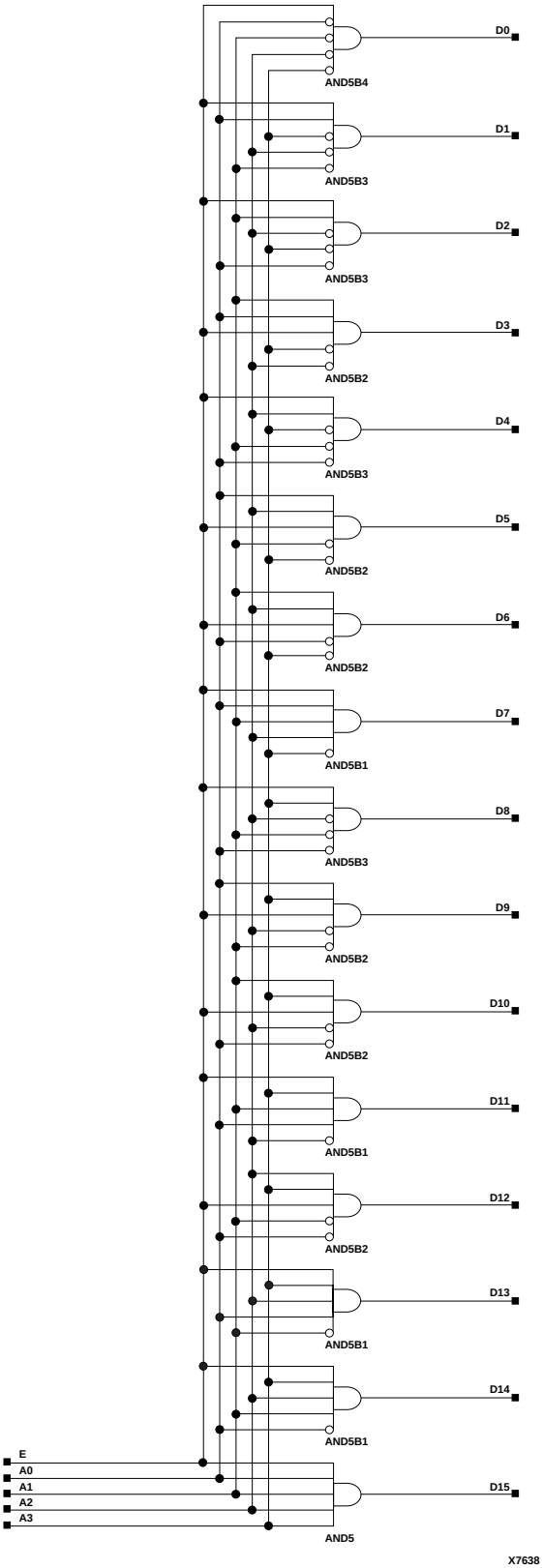
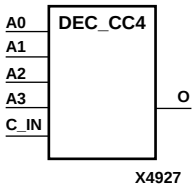


Figure 4-56 D4_16E Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

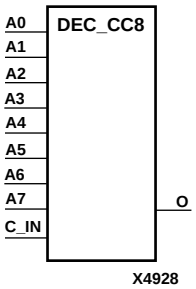
DEC_CC4, 8, 16

4-, 8-, 16-Bit Active Low Decoders

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	Macro	N / A	N / A	N / A	Macro	Macro

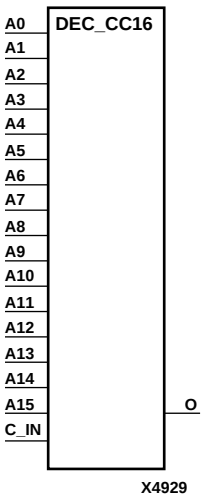


These decoders are used to build wide-decoder functions. They are implemented by cascading CY_MUX elements driven by lookup tables (LUTs). The C_IN pin can only be driven by a CY_INIT or by the output (O) of a previous decode stage. When one or more of the inputs (A) are Low, the output is Low. When all the inputs are High and the C_IN input is High, the output is High. You can decode patterns by adding inverters to inputs.



Inputs					Outputs
A0	A1	…	Az	C_IN	O
1	1	1	1	1	1
X	X	X	X	0	0
0	X	X	X	X	0
X	0	X	X	X	0
X	X	X	0	X	0

z = 3 for DEC_CC4; z = 7 for DECC_CC8; z = 15 for DECC_CC16



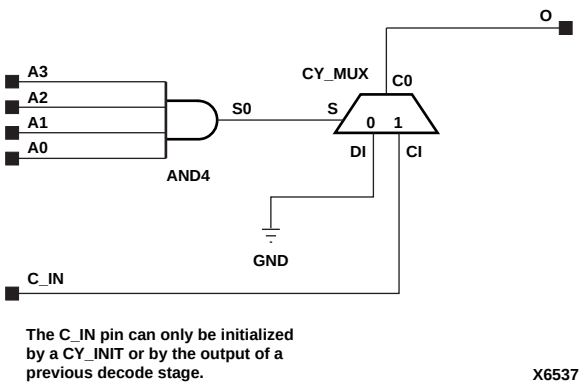


Figure 4-57 DEC_CC4 Implementation XC5200

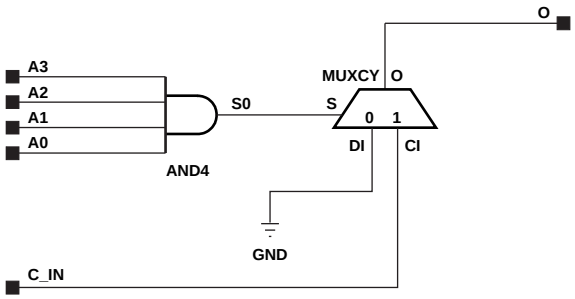


Figure 4-58 DEC_CC4 Implementation Spartan2, Virtex

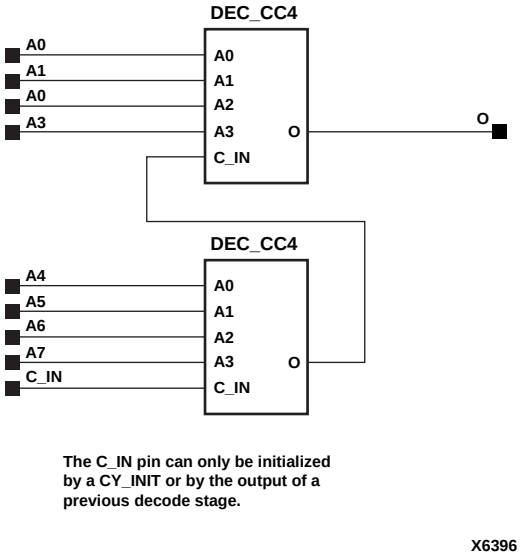


Figure 4-59 DEC_CC8 Implementation XC5200, Spartan2, Virtex

DECODE4, 8, 16

4-, 8-, 16-Bit Active-Low Decoders

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	N / A	N / A	N / A	Macro	Macro

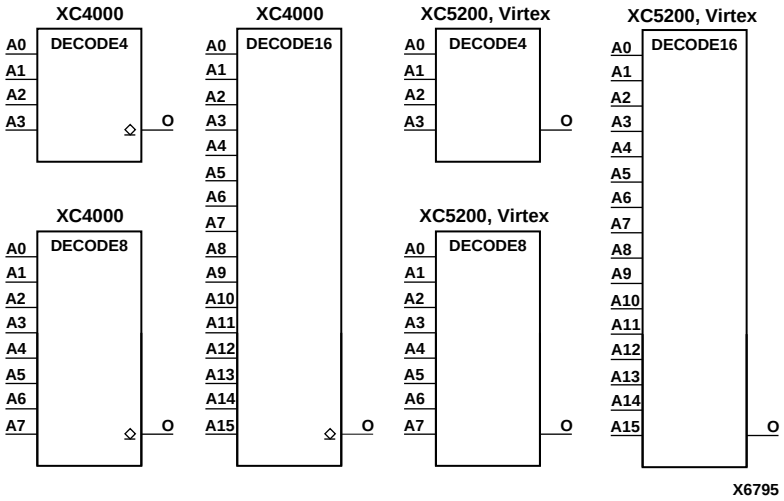


Figure 4-60 DECODE Representations

In the XC4000 architectures, decoders are open-drain, wired-AND gates. When one or more of the inputs (A) are Low, output (O) is Low. When all the inputs are High, the output is High or Off. A pull-up resistor must be connected to the output node to achieve a true logic High. A double pull-up resistor can be used to achieve faster performance; however, it uses more power. The software implements these macros using the open-drain AND gates around the periphery of the devices. (Diamonds in library symbols indicate an open-drain output.)

In XC5200, decoders are implemented by cascading CY_MUX elements driven by lookup tables (LUTs). When one or more of the inputs are Low, the output is Low. When all the inputs are High, the output is High. You can decode patterns by adding inverters to inputs. Pull-ups cannot be used on XC5200 longlines.

In Virtex and Spartan2, decoders are implemented using combinations of LUTs and MUXCYs.

Inputs				Outputs*
A0	A1	…	Az	O
1	1	1	1	1
0	X	X	X	0
X	0	X	X	0
X	X	X	0	0

z = 3 for DECODE4, z = 7 for DECODE8; z = 15 for DECODE16
*A pull-up resistor must be connected to the output to establish High-level drive current.

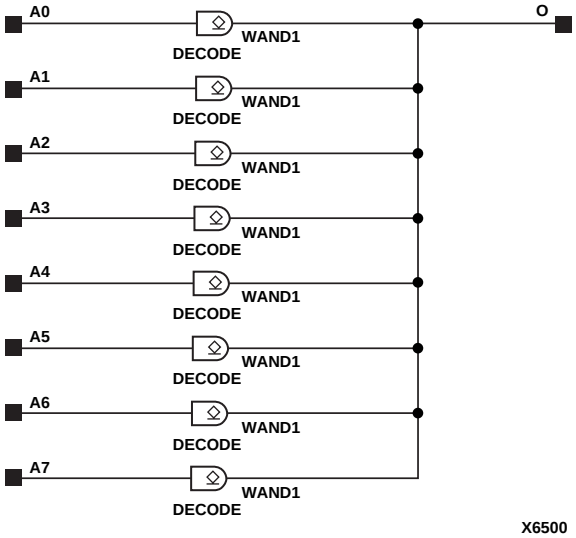


Figure 4-61 DECODE8 Implementation XC4000E, XC4000X

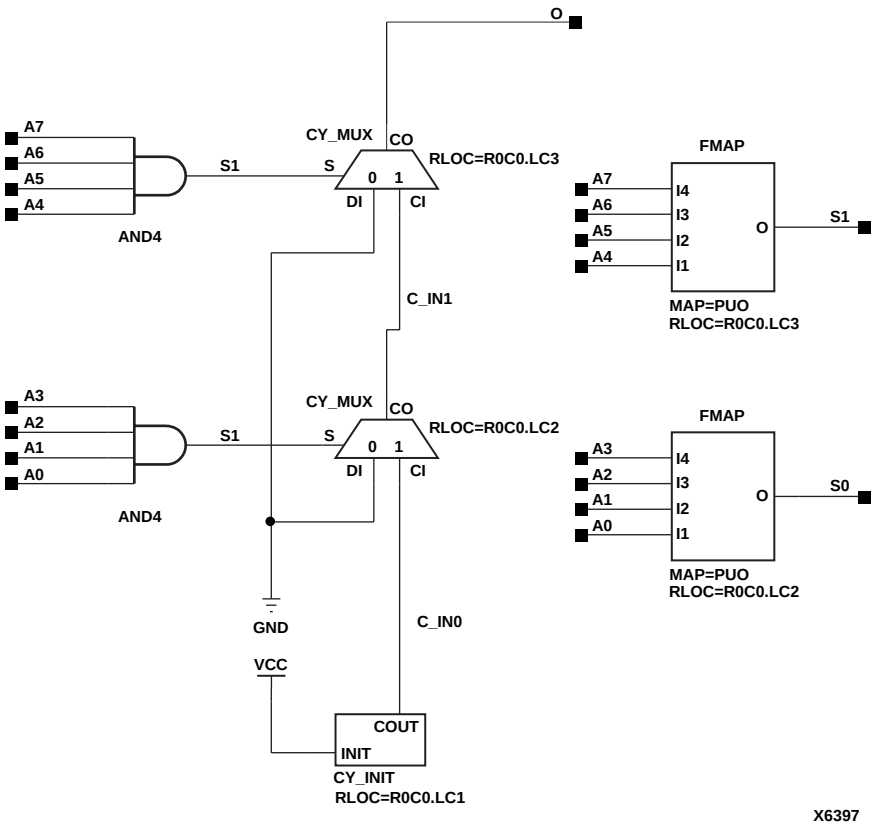


Figure 4-62 DECODE8 Implementation XC5200

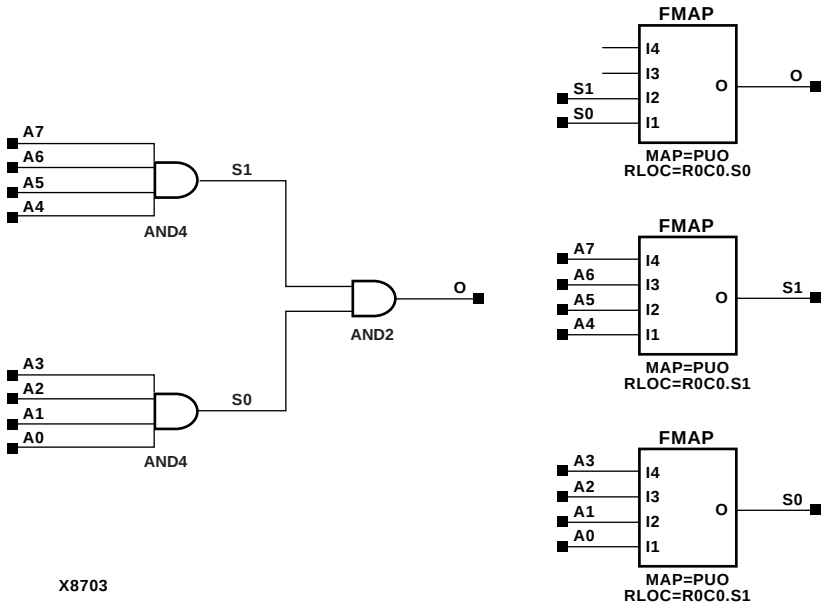
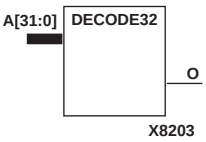


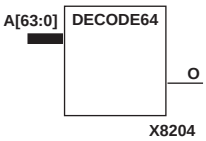
Figure 4-63 DECODE8 Implementation Spartan2, Virtex

DECODE32, 64

32- and 64-Bit Active-Low Decoders

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	Macro	N / A	N / A	N / A	Macro	Macro





DECODE32 and DECODE64 are 32- and 64-bit active-low decoders. In XC5200, decoders are implemented by cascading CY_MUX elements driven by lookup tables (LUTs). When one or more of the inputs are Low, the output is Low. When all the inputs are High, the output is High. You can decode patterns by adding inverters to inputs. Pull-ups cannot be used on XC5200 longlines.

In Virtex and Spartan2, decoders are implemented using combinations of LUTs and MUXCYs.

Refer to the “DECODE4, 8, 16” section for a representative schematic.

Inputs				Outputs
A0	A1	…	Az	O
1	1	1	1	1
0	X	X	X	0
X	0	X	X	0
X	X	X	0	0

z = 31 for DECODE32; z = 63 for DECODE64

Chapter 5

Design Elements (F5MAP to FTSRLE)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

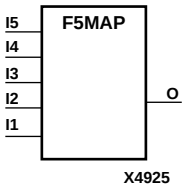
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

F5MAP

5-Input Function Partitioning Control Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



The F5MAP symbol is used to control the logic partitioning of 5-input functions into the top or bottom half of a CLB. The F5MAP symbol is not a substitute for logic. It is used in addition to combinatorial gates for mapping control.

At the schematic level, any 5-input logic function can be implemented using gates and mapped into half of a single CLB by using the F5MAP symbol. The signals that are the inputs and outputs of the 5-input function must be labelled and connected to appropriate pins of the F5MAP symbol, or the F5MAP signals and logic signals must have identical labels. The symbol can have unconnected pins, but all signals on the logic group to be mapped must be specified on a symbol pin.

Using F5MAP forces any 5-input function to be implemented by two lookup tables (LUTs), the direct input (DI), and the F5_MUX primitive, which are contained within adjacent CLB logic cells LC0 and LC1 or LC2 and LC3.

The connections within a CLB are shown in the "Two LUTs in Parallel Combined to Create a 5-Input Function" figure. An F5MAP primitive example is shown in the "F5MAP Primitive Example" figure.

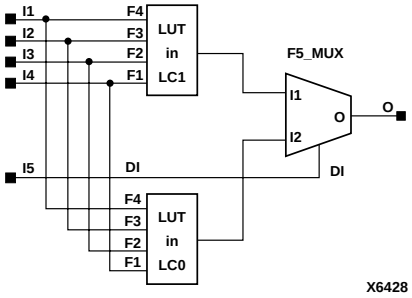


Figure 5-1 Two LUTs in Parallel Combined to Create a 5-Input Function

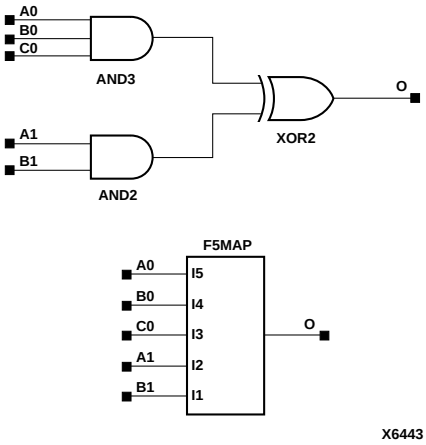
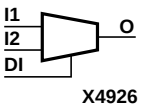


Figure 5-2 F5MAP Primitive Example

F5_MUX

2-to-1 Lookup Table Multiplexer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



F5_MUX provides a multiplexer function in one half of a CLB. The output from the lookup table (LUT) in LC1 is connected to the I1 input of the F5_MUX. The output from the LUT in LC0 is connected to the I2 input. The direct input (DI) of LC0 is connected to the DI input of the F5_MUX. The output (O) reflects the state of the selected input. When Low, DI selects I1; when High, DI selects I2. Similarly, the F5_MUX can connect to the LUTs in LC2 and LC3. The F5_MUX can also implement any 5-input function in the top or bottom half of a CLB when the mapping of the function is controlled by F5MAP.

Inputs			Outputs
DI	I1	I2	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

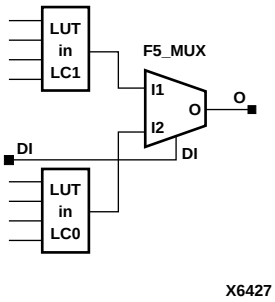
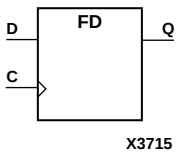


Figure 5-3 F5_MUX Representation

FD

D Flip-Flop

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FD is a single D-type flip-flop with data input (D) and data output (Q). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Refer to the “FD4, 8, 16” section for information on multiple D flip-flops for the XC9000.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1

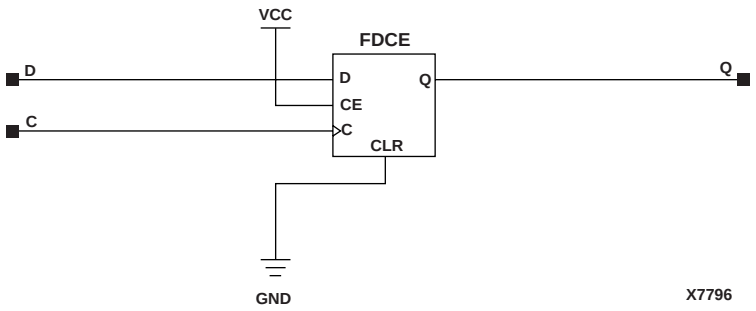


Figure 5-4 FD Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

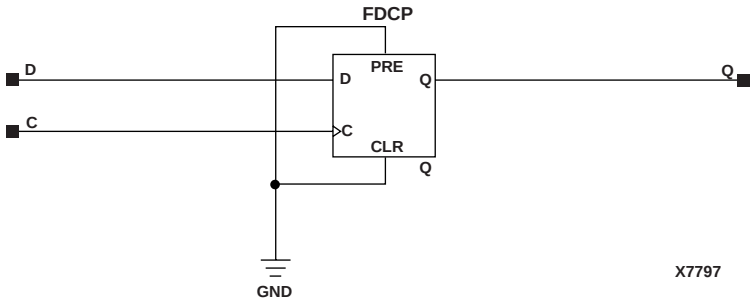
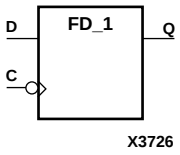


Figure 5-5 FD Implementation XC9000

FD_1

D Flip-Flop with Negative-Edge Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Primitive	Primitive



FD_1 is a single D-type flip-flop with data input (D) and data output (Q). The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1

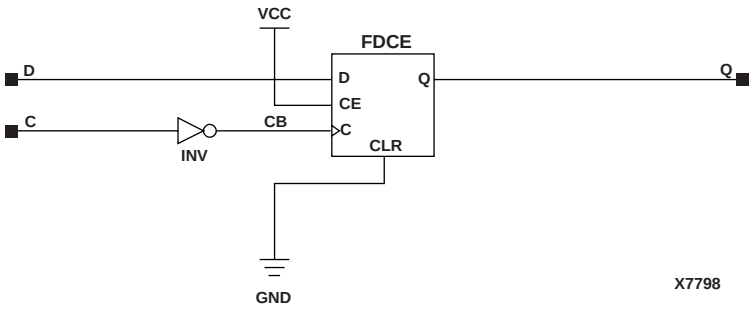
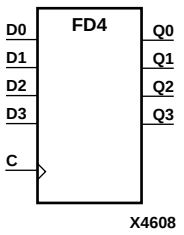


Figure 5-6 FD_1 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

FD4, 8, 16

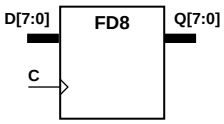
Multiple D Flip-Flops

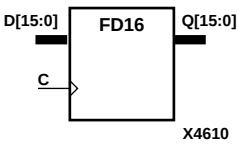
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A



FD4, FD8, FD16 are multiple D-type flip-flops with data inputs (D) and data outputs (Q). FD4, FD8, and FD16 are, respectively, 4-bit, 8-bit, and 16-bit registers, each with a common clock (C). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.





Inputs		Outputs
Dz – D0	C	Qz – Q0
0	↑	0
1	↑	1

z = 3 for FD4; z = 7 for FD8; z = 15 for FD16

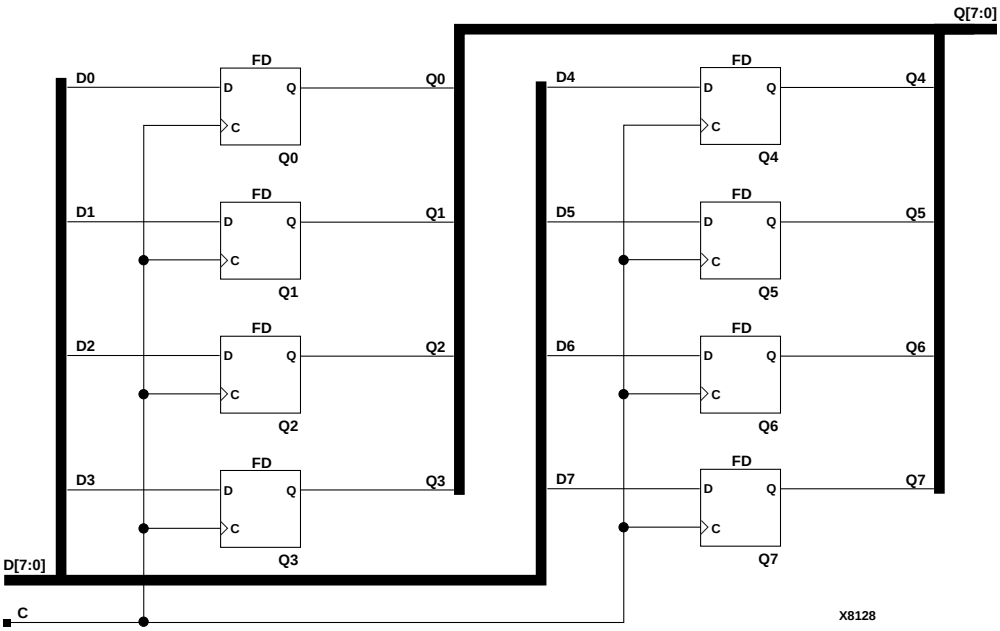
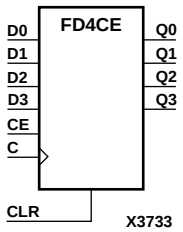


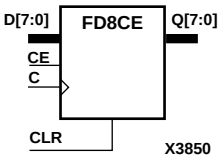
Figure 5-7 FD8 Implementation XC9000

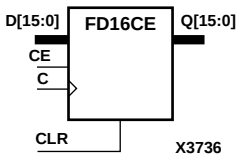
FD4CE, FD8CE, FD16CE

4-, 8-, 16-Bit Data Registers with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







FD4CE, FD8CE, and FD16CE are, respectively, 4-, 8-, and 16-bit data registers with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q) during the Low-to-High clock (C) transition. When CLR is High, it overrides all other inputs and resets the data outputs (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	CE	Dz – D0	C	Qz – Q0
1	X	X	X	0
0	0	X	X	No Chg
0	1	Dn	↑	dn

z = 3 for FD4CE; z = 7 for FD8CE; z = 15 for FD16CE.
dn = state of corresponding input (Dn) one setup time prior to active clock transition

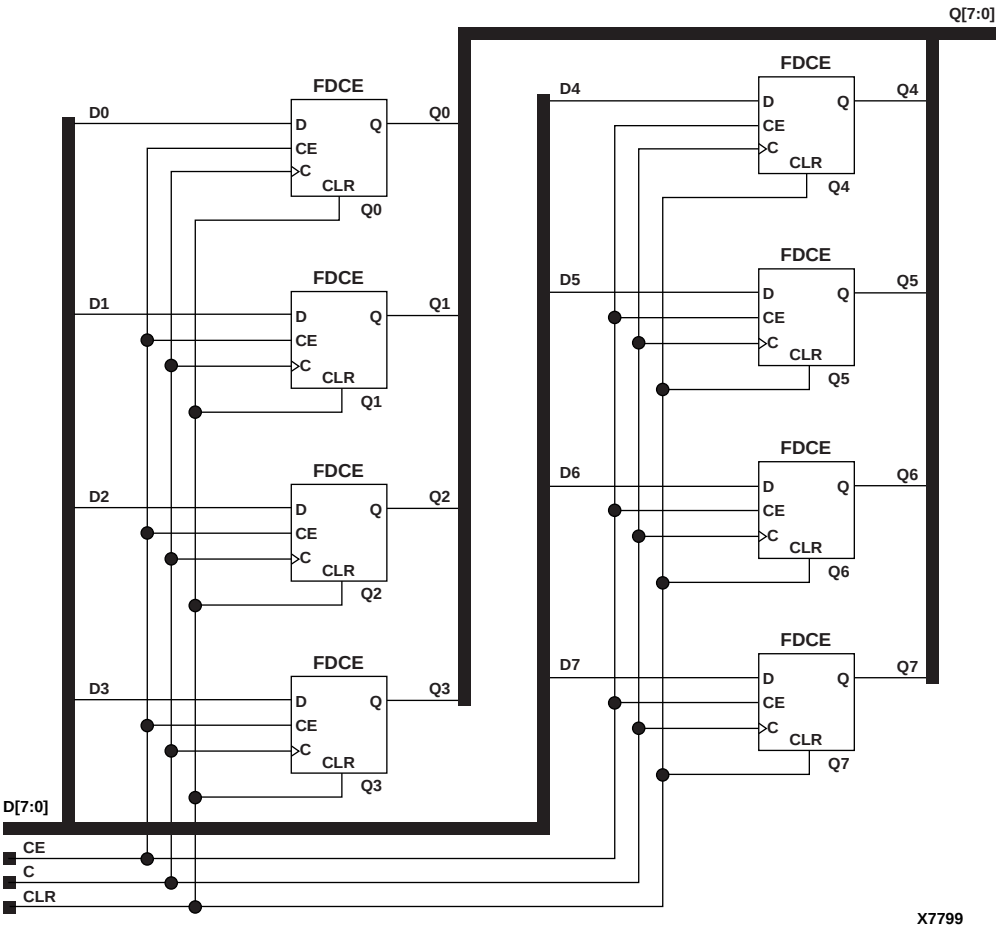
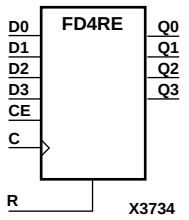


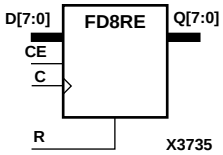
Figure 5-8 FD8CE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

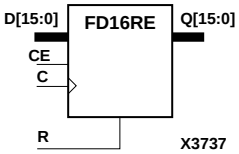
FD4RE, FD8RE, FD16RE

4-, 8-, 16-Bit Data Registers with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







FD4RE, FD8RE, and FD16RE are, respectively, 4-, 8-, and 16-bit data registers. When the clock enable (CE) input is High, and the synchronous reset (R) input is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q0) during the Low-to-High clock (C) transition. When R is High, it overrides all other inputs and resets the data outputs (Q) Low on the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
R	CE	Dz – D0	C	Qz – Q0
1	X	X	↑	0
0	0	X	X	No Chg
0	1	Dn	↑	dn

z = 3 for FD4RE; z = 7 for FD8RE; z = 15 for FD16RE
dn = state of referenced input (Dn) one setup time prior to active clock transition

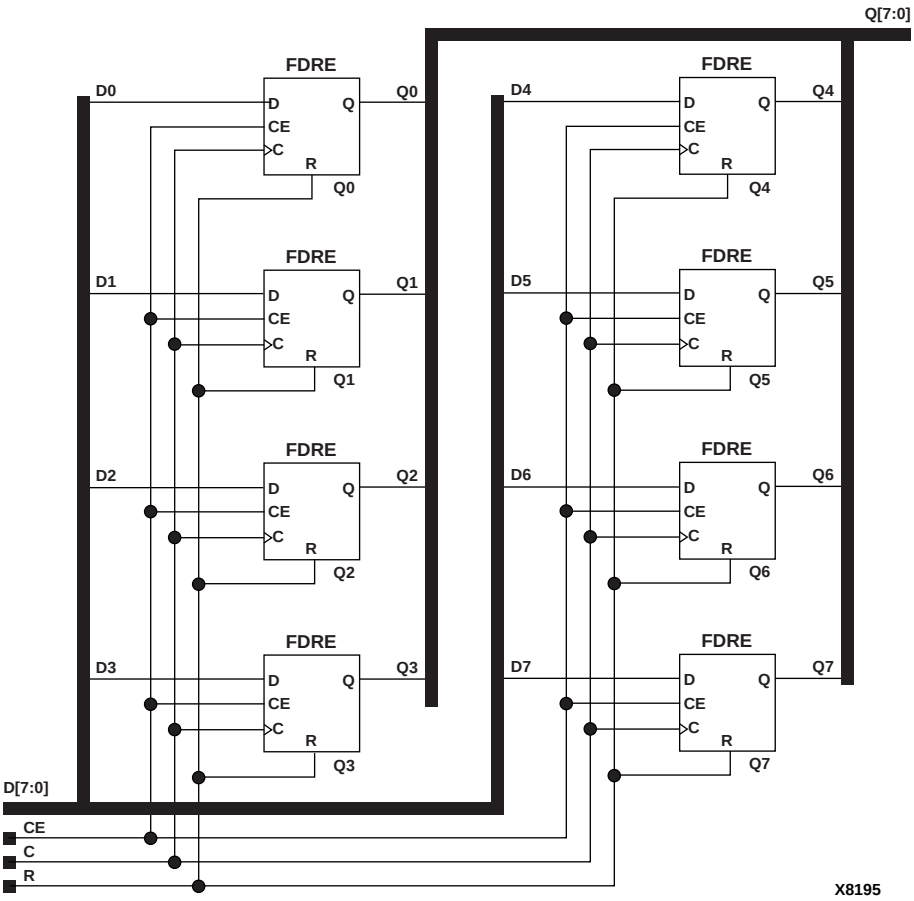
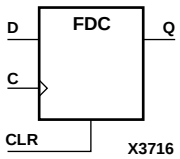


Figure 5-9 FD8RE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

FDC

D Flip-Flop with Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDC is a single D-type flip-flop with data (D) and asynchronous clear (CLR) inputs and data output (Q). The asynchronous CLR, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low on the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↑	1
0	0	↑	0

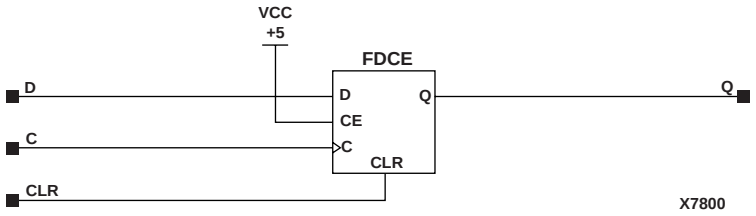


Figure 5-10 FDC Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

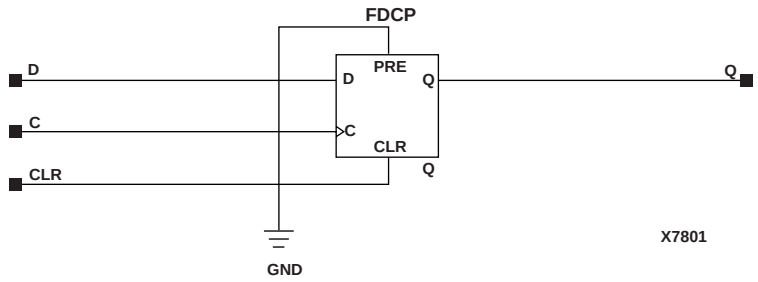
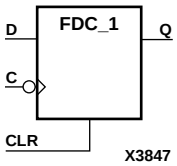


Figure 5-11 FDC Implementation XC9000

FDC_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Primitive	Primitive



FDC_1 is a single D-type flip-flop with data input (D), asynchronous clear input (CLR), and data output (Q). The asynchronous CLR, when active, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↓	1
0	0	↓	0

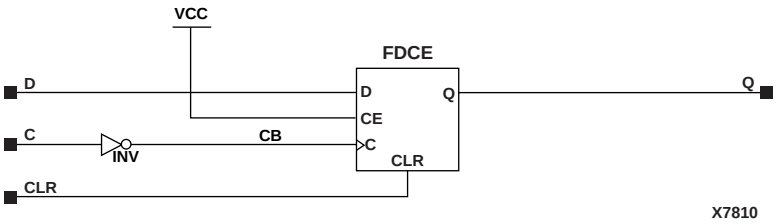
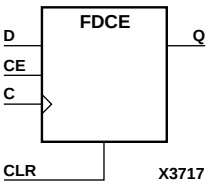


Figure 5-12 FDC_1 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

FDCE

D Flip-Flop with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



FDCE is a single D-type flip-flop with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data input (D) of FDCE is transferred to the corresponding data output (Q) during the Low-to-High clock (C) transition. When CLR is High, it overrides all other inputs and resets the data output (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

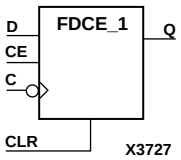
For XC9500XL and XC9500XV devices, logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDCE and FDPE flip-flops primitives may take advantage of the clock-enable p-term.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0

FDCE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Primitive	Primitive



FDCE_1 is a single D-type flip-flop with data (D), clock enable (CE), asynchronous clear (CLR) inputs, and data output (Q). The asynchronous CLR input, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	↓	No Chg
0	1	1	↓	1
0	1	0	↓	0

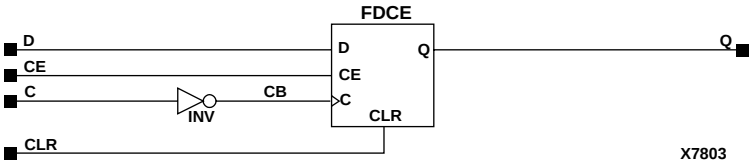
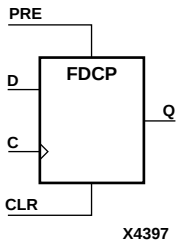


Figure 5-13 FDCE_1 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

FDCP

D Flip-Flop Asynchronous Preset and Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Primitive	N/A	N/A	Primitive	Primitive



FDCP is a single D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the Low-to-High clock (C) transition.

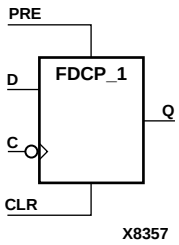
The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↑	0
0	0	1	↑	1

FDCP_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Preset and Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDCP_1 is a single D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the High-to-Low clock (C) transition.

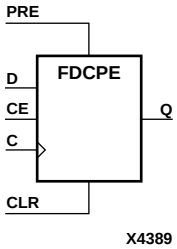
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↓	0
0	0	1	↓	1

FDCPE

D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Macro	N/A	N/A	Primitive	Primitive



FDCPE is a single D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↑	0
0	0	1	1	↑	1

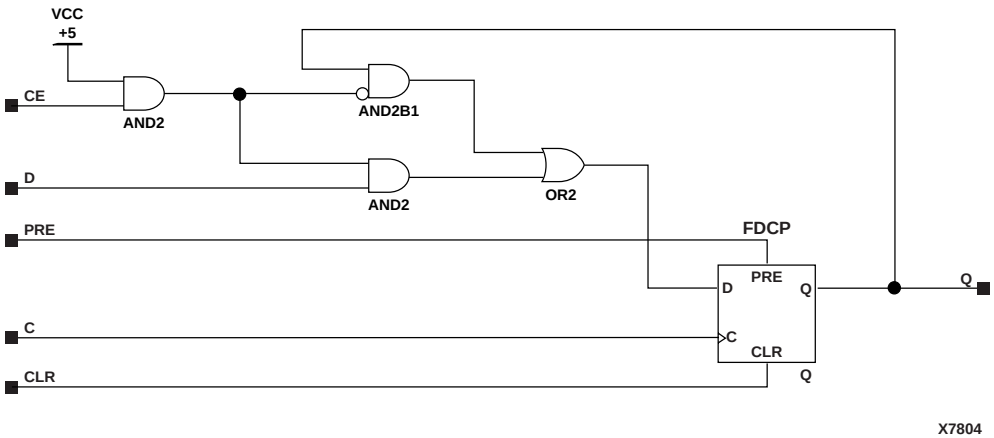
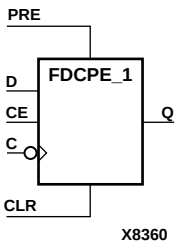


Figure 5-14 FDCPE Implementation XC9000

FDCPE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset and Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDCPE_1 is a single D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

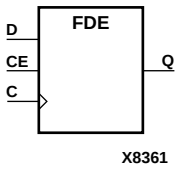
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↓	0
0	0	1	1	↓	1

FDE

D Flip-Flop with Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDE is a single D-type flip-flop with data input (D), clock enable (CE), and data output (Q). When clock enable is High, the data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition.

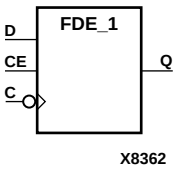
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs			Outputs
CE	D	C	Q
0	X	X	No Chg
1	0	↑	0
1	1	↑	1

FDE_1

D Flip-Flop with Negative-Edge Clock and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDE_1 is a single D-type flip-flop with data input (D), clock enable (CE), and data output (Q). When clock enable is High, the data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

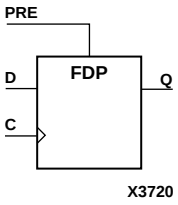
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs			Outputs
CE	D	C	Q
0	X	X	No Chg
1	0	↓	0
1	1	↓	1

FDP

D Flip-Flop with Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDP is a single D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the Low-to-High clock (C) transition.

For FPGAs, the flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The active level of the GR/GSR defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↑	1	1
0	↑	0	0

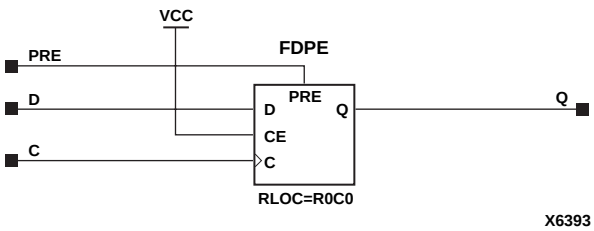


Figure 5-15 FDP Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

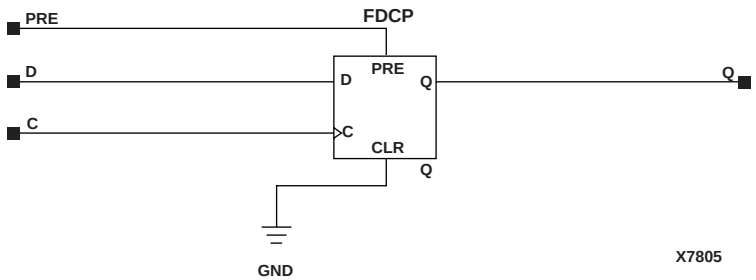
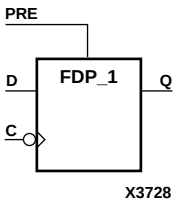


Figure 5-16 FDP Implementation XC9000

FDP_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	N/A	Macro	Macro	Primitive	Primitive



FDP_1 is a single D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the High-to-Low clock (C) transition.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The active level of the GR/GSR defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↓	1	1
0	↓	0	0

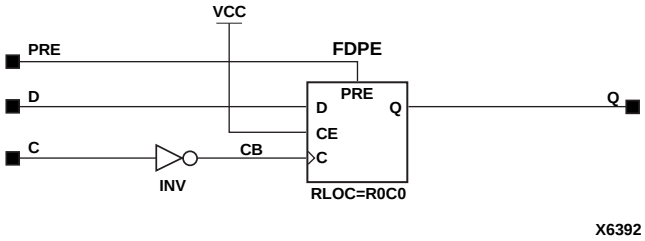
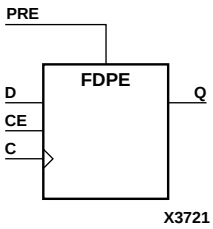


Figure 5-17 FDP_1 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

FDPE

D Flip-Flop with Clock Enable and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Macro	Primitive	Primitive	Primitive	Primitive	Primitive



FDPE is a single D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the Low-to-High clock (C) transition. When CE is Low, the clock transitions are ignored.

For FPGAs, the flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The active level of the GR/GSR defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For XC9500XL and XC9500XV devices, logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDCE and FDPE flip-flops primitives may take advantage of the clock-enable p-term.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	↑	0
0	1	1	↑	1

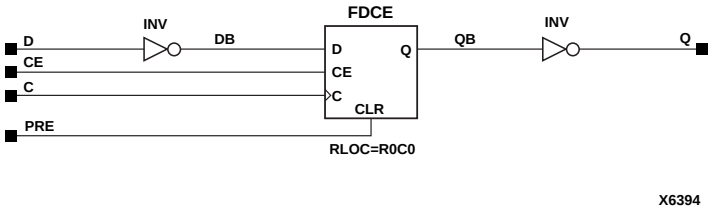
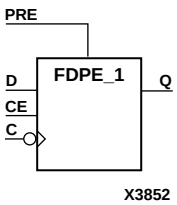


Figure 5-18 FDPE Implementation XC5200

FDPE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	N/A	Macro	Macro	Primitive	Primitive



FDPE_1 is a single D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The active level of the GR/GSR defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

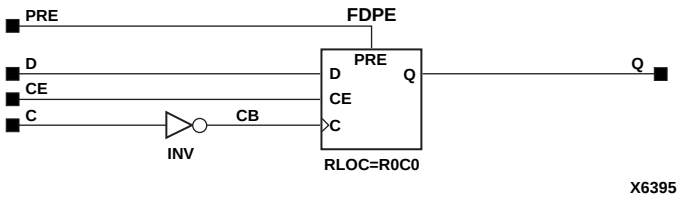
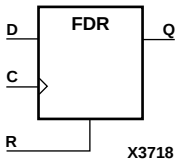


Figure 5-19 FDPE_1 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

FDR

D Flip-Flop with Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDR is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
R	D	C	Q
1	X	↑	0
0	1	↑	1
0	0	↑	0

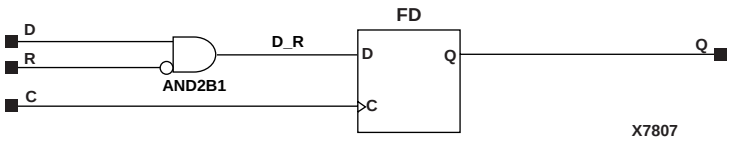
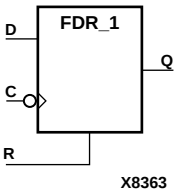


Figure 5-20 FDR Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

FDR_1

D Flip-Flop with Negative-Edge Clock and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDR_1 is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the High-to-Low clock transition.

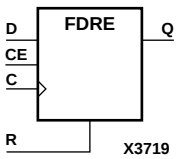
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs			Outputs
R	D	C	Q
1	X	↓	0
0	1	↓	1
0	0	↓	0

FDRE

D Flip-Flop with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDRE is a single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↑	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0

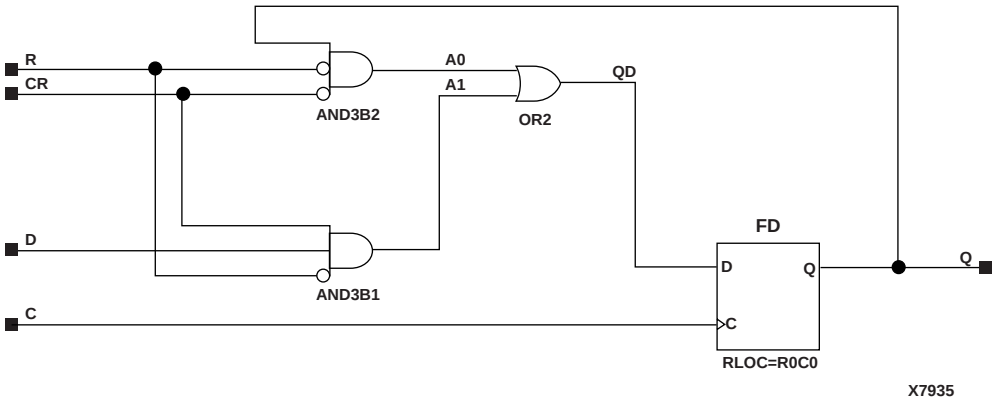


Figure 5-21 FDRE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

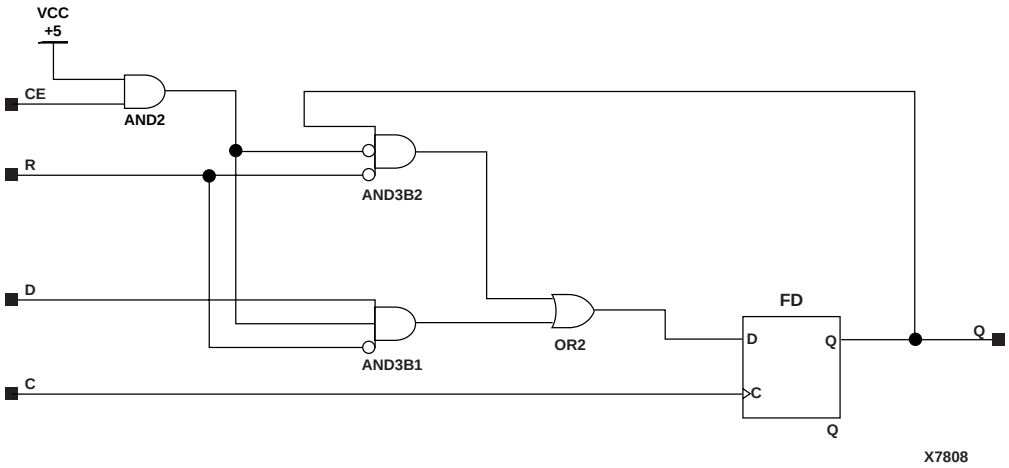
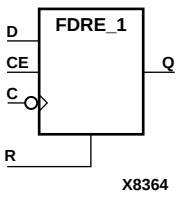


Figure 5-22 FDRE Implementation XC9000

FDRE_1

D Flip-Flop with Negative-Clock Edge, Clock Enable, and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDRE_1 is a single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the High-to-Low clock transition.

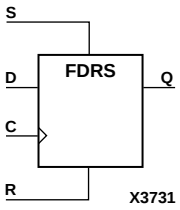
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↓	0
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

FDRS

D Flip-Flop with Synchronous Reset and Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDRS is a single D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High clock (C) transition. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↑	0
0	1	X	↑	1
0	0	1	↑	1
0	0	0	↑	0

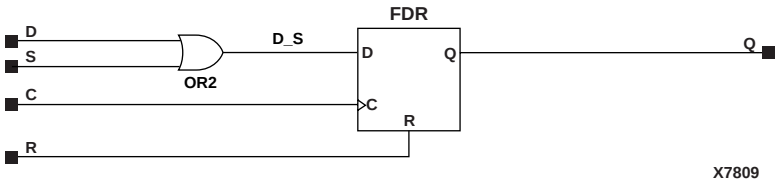


Figure 5-23 FDRS Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

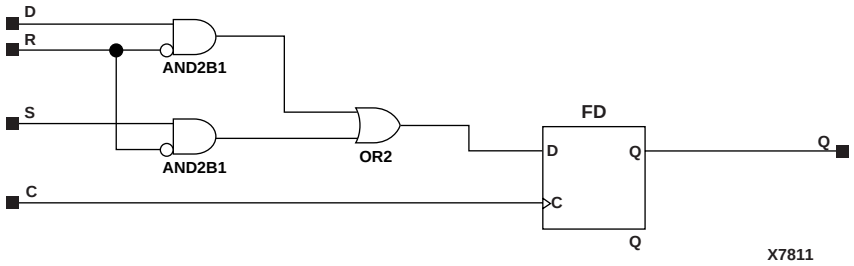
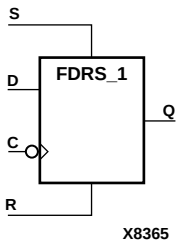


Figure 5-24 FDRS Implementation XC9000

FDRS_1

D Flip-Flop with Negative-Clock Edge and Synchronous Reset and Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDRS_1 is a single D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the High-to-Low clock (C) transition. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the High-to-Low clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the High-to-Low clock transition.

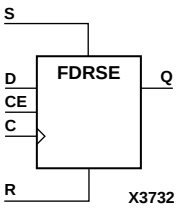
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↓	0
0	1	X	↓	1
0	0	1	↓	1
0	0	0	↓	0

FDRSE

D Flip-Flop with Synchronous Reset and Set and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDRSE is a single D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High clock transition. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0

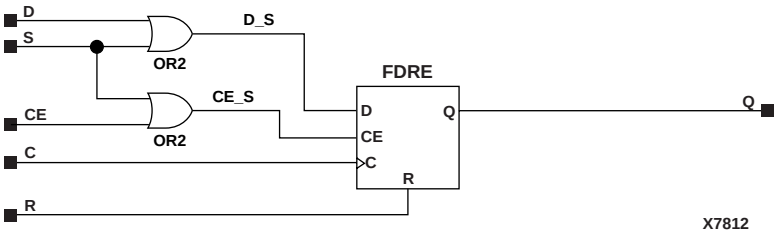


Figure 5-25 FDRSE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

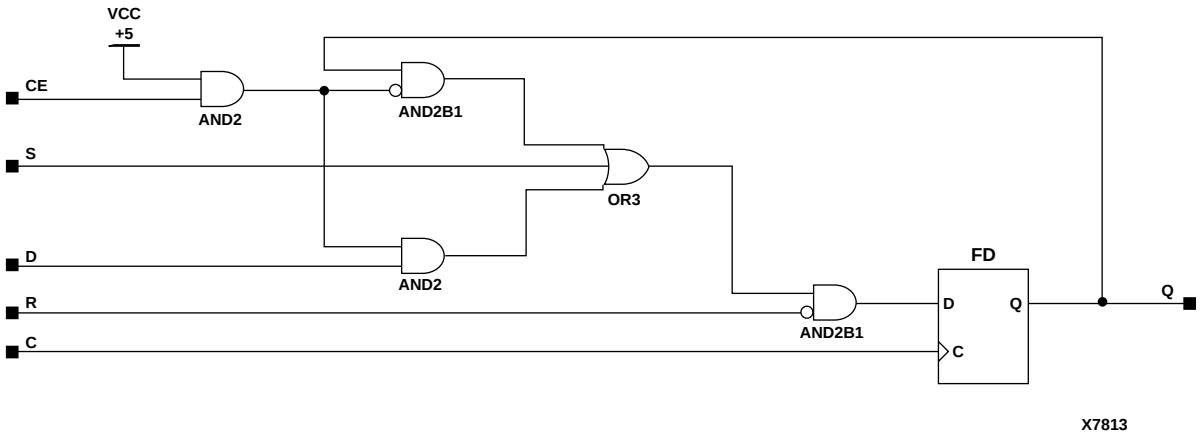
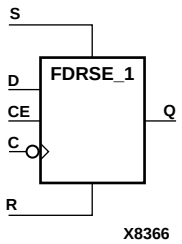


Figure 5-26 FDRSE Implementation XC9000

FDRSE_1

D Flip-Flop with Negative-Clock Edge, Synchronous Reset and Set, and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDRSE_1 is a single D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the High-to-Low clock transition. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the High-to-Low clock transition.

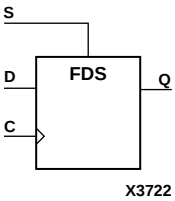
The flip-flop is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↓	0
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	1	↓	1
0	0	1	0	↓	0

FDS

D Flip-Flop with Synchronous Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDS is a single D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the Low-to-High clock (C) transition.

For Virtex and Spartan2, the flip-flop is asynchronously preset, output High, when power is applied. For all other devices, the flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
S	D	C	Q
1	X	↑	1
0	1	↑	1
0	0	↑	0

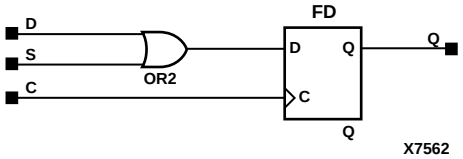
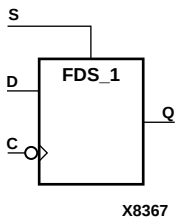


Figure 5-27 FDS Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

FDS_1

D Flip-Flop with Negative-Edge Clock and Synchronous Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDS_1 is a single D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the High-to-Low clock (C) transition.

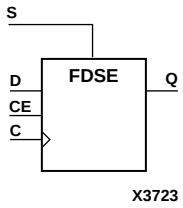
For Virtex and Spartan2, the flip-flop is asynchronously preset, output High, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs			Outputs
S	D	C	Q
1	X	↓	1
0	1	↓	1
0	0	↓	0

FDSE

D Flip-Flop with Clock Enable and Synchronous Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Primitive	Primitive



FDSE is a single D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the Low-to-High clock (C) transition.

For Virtex and Spartan2, the flip-flop is asynchronously preset, output High, when power is applied. For all other devices, the flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↑	1
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0

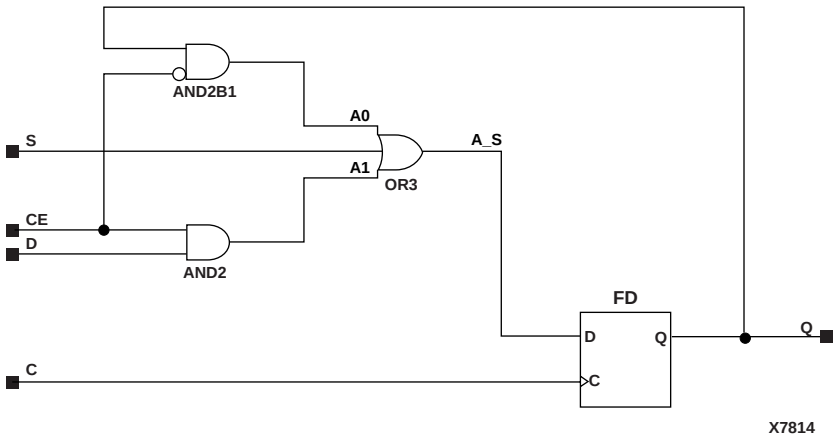


Figure 5-28 FDSE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

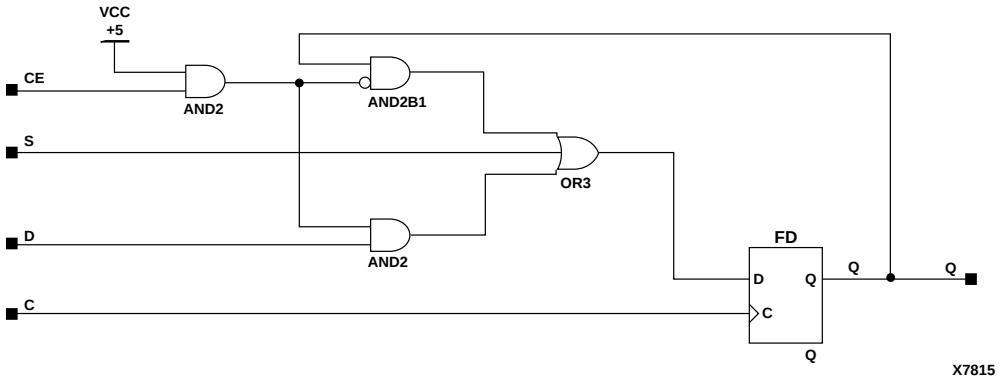
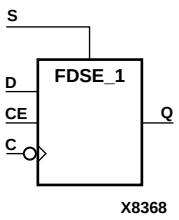


Figure 5-29 FDSE Implementation XC9000

FDSE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Synchronous Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



FDSE_1 is a single D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the High-to-Low clock (C) transition.

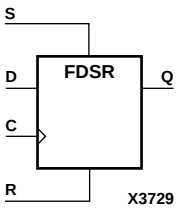
For Virtex and Spartan2, the flip-flop is asynchronously preset, output High, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↓	1
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

FDSR

D Flip-Flop with Synchronous Set and Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N/A	N/A



FDSR is a single D-type flip-flop with data (D), synchronous reset (R) and synchronous set (S) inputs and data output (Q). When the set (S) input is High, it overrides all other inputs and sets the Q output High during the Low-to-High clock transition. (Set has precedence over Reset.) When reset (R) is High and S is Low, the flip-flop is reset, output Low, on the Low-to-High clock transition. Data on the D input is loaded into the flip-flop when S and R are Low on the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP symbol.

Inputs				Outputs
S	R	D	C	Q
1	X	X	↑	1
0	1	X	↑	0
0	0	1	↑	1
0	0	0	↑	0

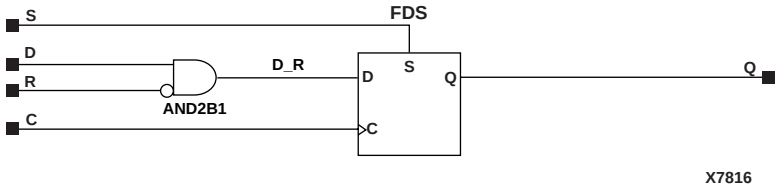


Figure 5-30 FDSR Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

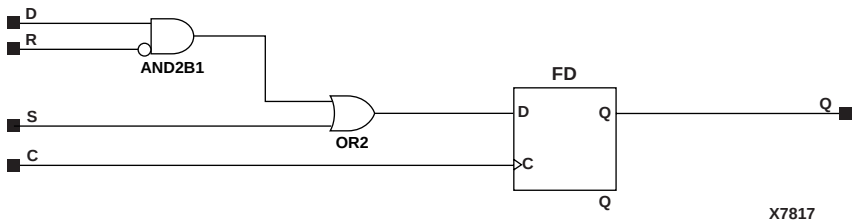
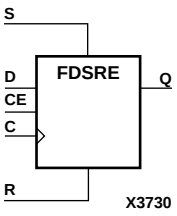


Figure 5-31 FDSR Implementation XC9000

FDSRE

D Flip-Flop with Synchronous Set and Reset and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N/A	N/A



FDSRE is a single D-type flip-flop with synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, it overrides all other inputs and sets the Q output High during the Low-to-High clock transition. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low during the Low-to-High clock transition. Data is loaded into the flip-flop when S and R are Low and CE is High during the Low-to-high clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP symbol.

Inputs					Outputs
S	R	CE	D	C	Q
1	X	X	X	↑	1
0	1	X	X	↑	0
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0

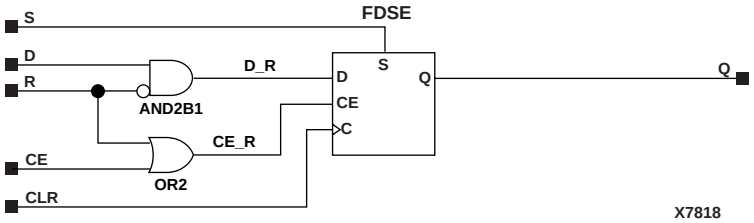


Figure 5-32 FDSRE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

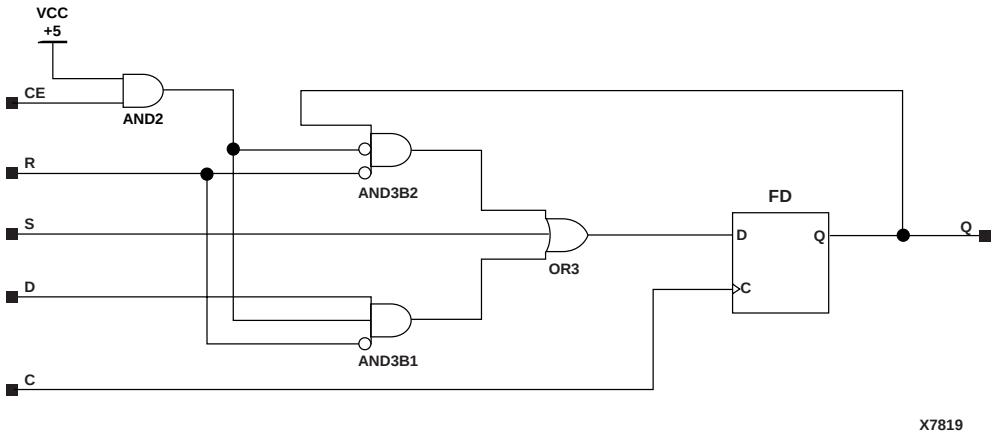
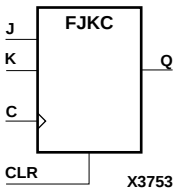


Figure 5-33 FDSRE Implementation XC9000

FJKC

J-K Flip-Flop with Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FJKC is a single J-K-type flip-flop with J, K, and asynchronous clear (CLR) inputs and data output (Q). The asynchronous clear (CLR) input, when High, overrides all other inputs and resets the Q output Low. When CLR is Low, the output responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	J	K	C	Q
1	X	X	X	0
0	0	0	↑	No Chg
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle

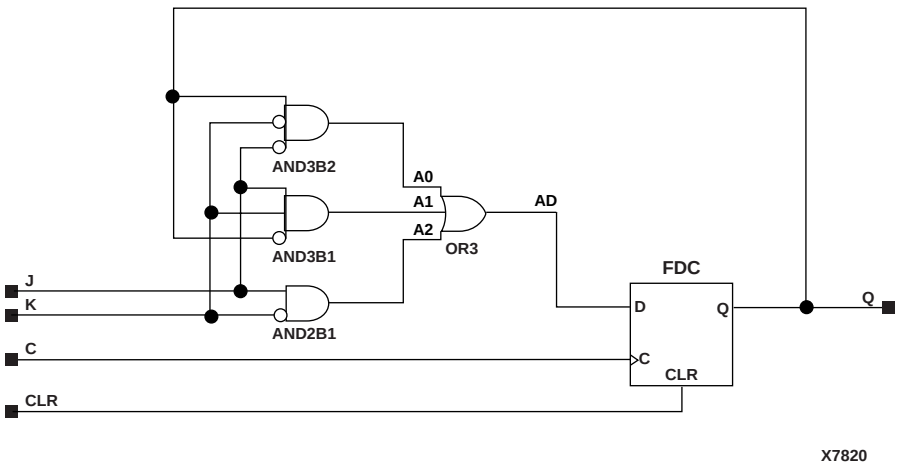


Figure 5-34 FJKC Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

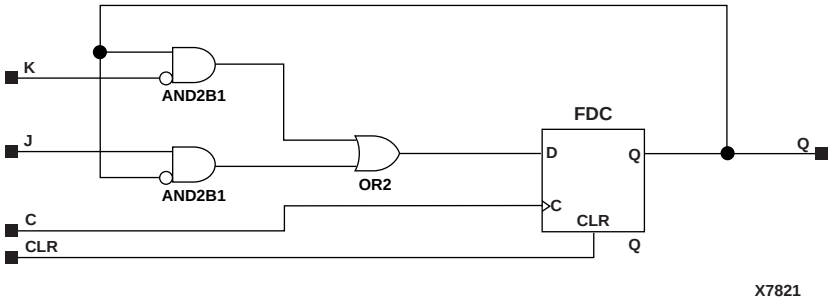
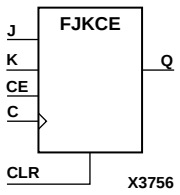


Figure 5-35 FJKC Implementation XC9000

FJKCE

J-K Flip-Flop with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FJKCE is a single J-K-type flip-flop with J, K, clock enable (CE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous clear (CLR), when High, overrides all other inputs and resets the Q output Low. When CLR is Low and CE is High, Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs
CLR	CE	J	K	C	Q
1	X	X	X	X	0
0	0	X	X	X	No Chg
0	1	0	0	X	No Chg
0	1	0	1	↑	0
0	1	1	0	↑	1
0	1	1	1	↑	Toggle

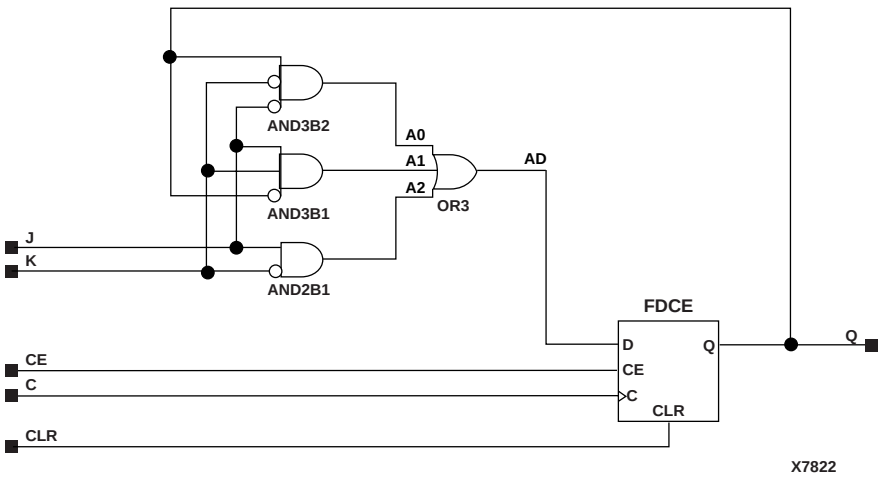


Figure 5-36 FJKCE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

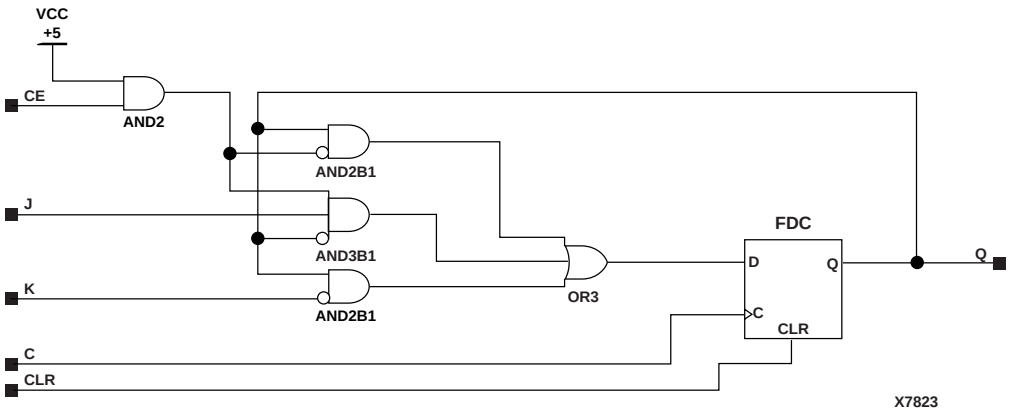
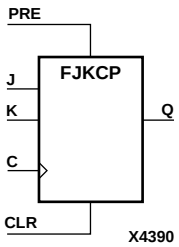


Figure 5-37 FJKCE Implementation XC9000

FJKCP

J-K Flip-Flop with Asynchronous Clear and Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A



FJKCP is a single J-K-type flip-flop with J, K, asynchronous clear (CLR), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous clear input (CLR), when High, overrides all other inputs and resets the Q output Low. The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When CLR and PRE are Low, Q responds to the state of the J and K inputs during the Low-to-High clock transition, as shown in the following truth table.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	J	K	C	Q
1	0	X	X	X	0
0	1	X	X	X	1
0	0	0	0	X	No Chg
0	0	0	1	↑	0
0	0	1	0	↑	1
0	0	1	1	↑	Toggle

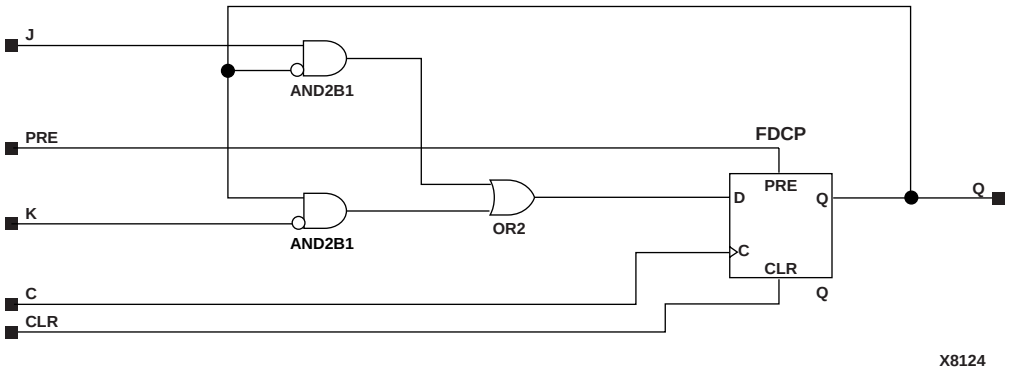
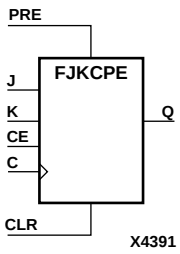


Figure 5-38 FJKCP Implementation XC9000

FJKCPE

J-K Flip-Flop with Asynchronous Clear and Preset and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A



FJKCPE is a single J-K-type flip-flop with J, K, asynchronous clear (CLR), asynchronous preset (PRE), and clock enable (CE) inputs and data output (Q). The asynchronous clear input (CLR), when High, overrides all other inputs and resets the Q output Low. The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When CLR and PRE are Low and CE is High, Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs
CLR	PRE	CE	J	K	C	Q
1	X	X	X	X	X	0
0	1	X	X	X	X	1
0	0	0	0	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle

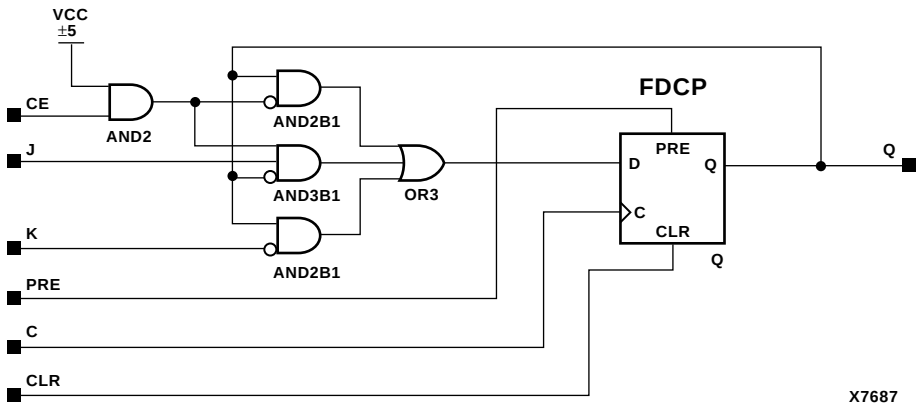
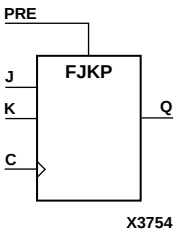


Figure 5-39 FJKCPE Implementation XC9000

FJKP

J-K Flip-Flop with Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FJKP is a single J-K-type flip-flop with J, K, and asynchronous preset (PRE) inputs and data output (Q). The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When PRE is Low, the Q output responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition.

For FPGAs, the flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The GR/GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or the STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
PRE	J	K	C	Q
1	X	X	X	1
0	0	0	X	No Chg
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle

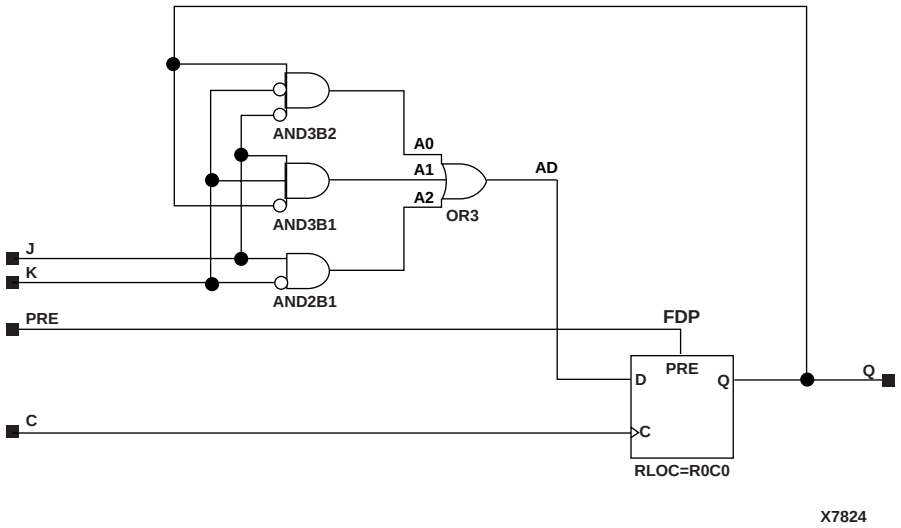


Figure 5-40 FJKP Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

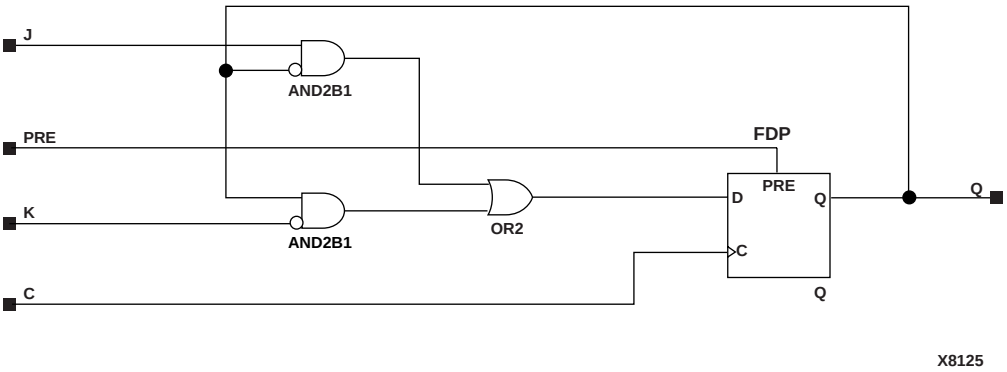
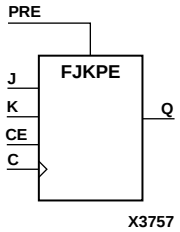


Figure 5-41 FJKP Implementation XC9000

FJKPE

J-K Flip-Flop with Clock Enable and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FJKPE is a single J-K-type flip-flop with J, K, clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous preset (PRE), when High, overrides all other inputs and sets the Q output High. When PRE is Low and CE is High, the Q output responds to the state of the J and K inputs, as shown in the truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

For FPGAs, the flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The GR/GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or the STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
PRE	CE	J	K	C	Q
1	X	X	X	X	1
0	0	X	X	X	No Chg
0	1	0	0	X	No Chg
0	1	0	1	↑	0
0	1	1	0	↑	1
0	1	1	1	↑	Toggle

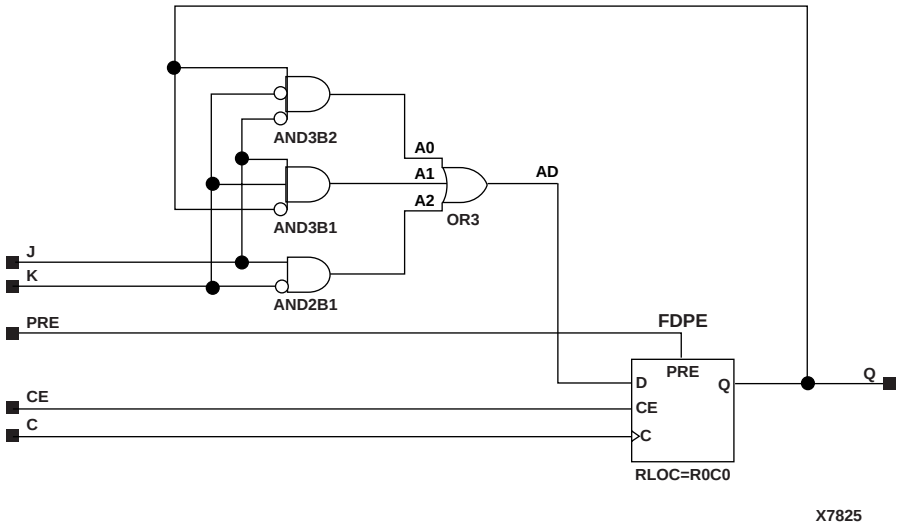


Figure 5-42 FJKPE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

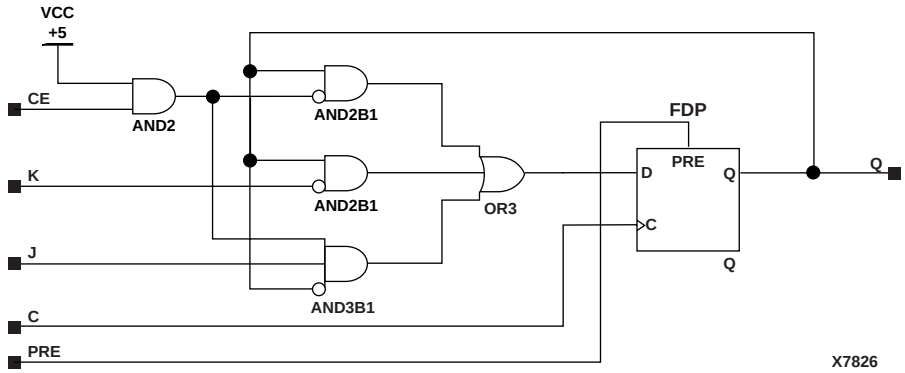
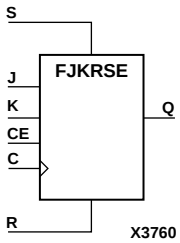


Figure 5-43 FJKPE Implementation XC9000

FJKRSE

J-K Flip-Flop with Clock Enable and Synchronous Reset and Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FJKRSE is a single J-K-type flip-flop with J, K, synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). When synchronous reset (R) is High, all other inputs are ignored and output Q is reset Low. (Reset has precedence over Set.) When synchronous set (S) is High and R is Low, output Q is set High. When R and S are Low and CE is High, output Q responds to the state of the J and K inputs, according to the following truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs
R	S	CE	J	K	C	Q
1	X	X	X	X	↑	0
0	1	X	X	X	↑	1
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle

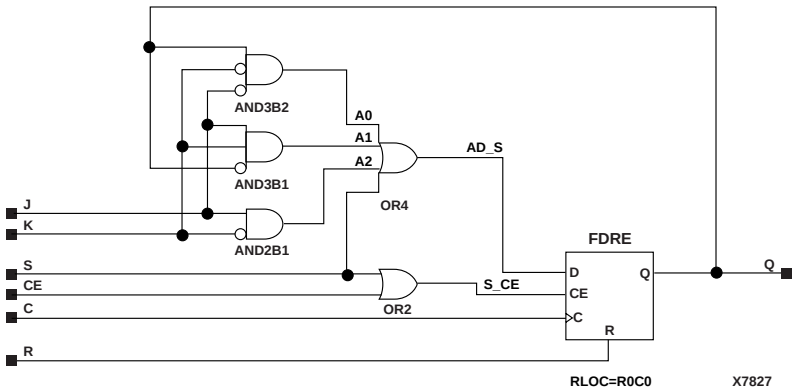


Figure 5-44 FJKRSE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

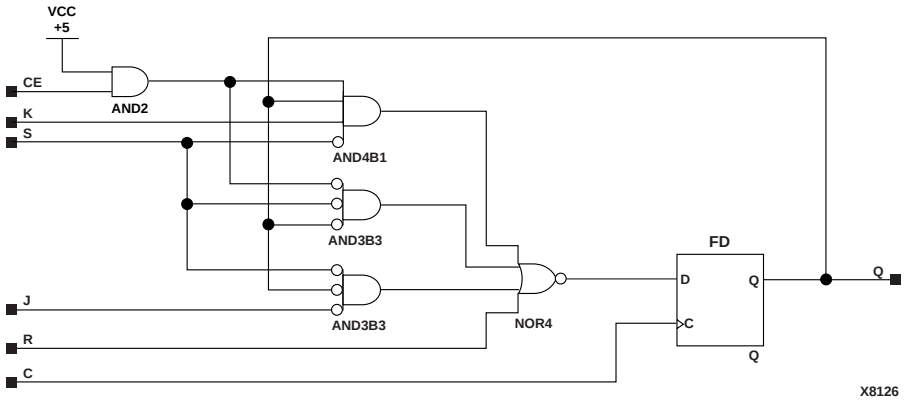
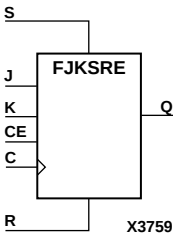


Figure 5-45 FJKRSE Implementation XC9000

FJKSRE

J-K Flip-Flop with Clock Enable and Synchronous Set and Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FJKSRE is a single J-K-type flip-flop with J, K, synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, all other inputs are ignored and output Q is set High. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low. When S and R are Low and CE is High, output Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs
S	R	CE	J	K	C	Q
1	X	X	X	X	↑	1
0	1	X	X	X	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle

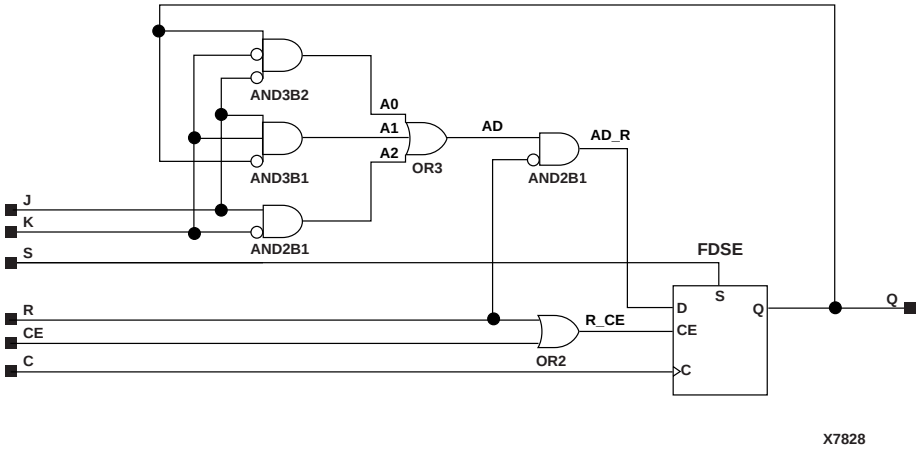


Figure 5-46 FJKSRE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

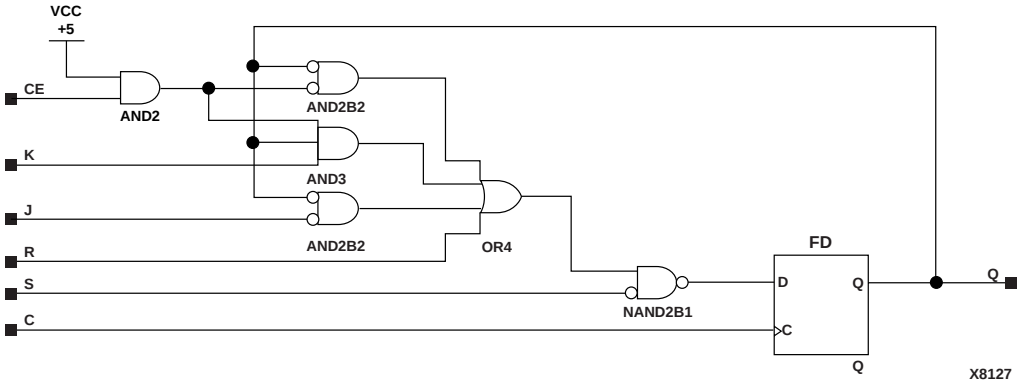
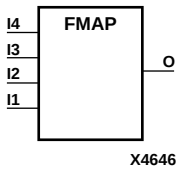


Figure 5-47 FJKSRE Implementation XC9000

FMAP

F Function Generator Partitioning Control Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Primitive	Primitive	Primitive	N / A	Primitive	Primitive	Primitive	Primitive



The FMAP symbol is used to control logic partitioning into XC4000 4-input function generators. For XC4000, Spartan, and SpartanXL, the place and route software chooses an F or a G function generator as a default, unless you specify an F or G. The FMAP symbol is used in an XC5200, a Virtex, or a Spartan2 device to map logic to the function generator of a slice. Refer to the appropriate CAE tool interface user guide for information about specifying this attribute in your schematic design editor.

For the XC4000, Spartan, and SpartanXL devices, the FMAP symbol is usually used with the HMAP symbol, which partitions logic into the 3-input generator of the Configurable Logic Block (CLB). You can implement a portion of logic using gates, latches, and flip-flops and specify the logic to be grouped into F, G, and H function generators by naming logic signals and FMAP / HMAP signals correspondingly. These symbols are used for mapping control in addition to the actual gates, latches, and flip-flops, not as a substitute for them.

The following figure gives an example of how logic can be placed using FMAP and HMAP symbols.

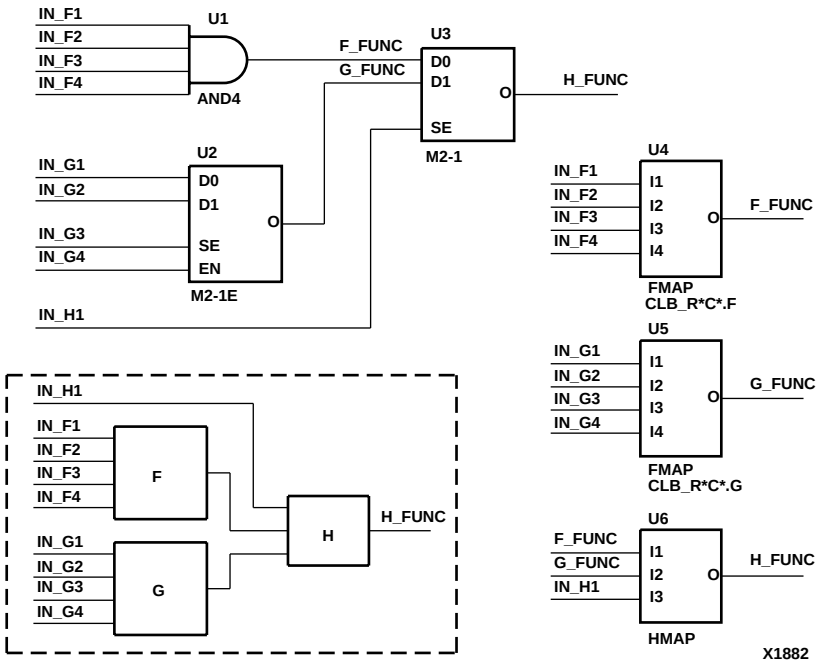


Figure 5-48 Partitioning Logic Using FMAP and HMAP Symbols

The MAP=*type* parameter can be used with the FMAP symbol to further define how much latitude you want to give the mapping program. The following table shows MAP option characters and their meanings.

MAP Option Character	Function
P	Pins.
C	Closed — Adding logic to or removing logic from the CLB is not allowed.
L	Locked — Locking CLB pins.
O	Open — Adding logic to or removing logic from the CLB is allowed.
U	Unlocked — No locking on CLB pins.

Possible types of MAP parameters for FMAP are MAP=PUC, MAP=PLC, MAP=PLO, and MAP=PUO. The default parameter is PUO. If one of the "open" parameters is used (PLO or PUO), only the output signals must be specified.

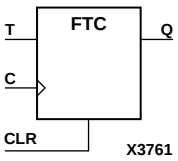
Note: Currently, only PUC and PUO are observed. PLC and PLO are translated into PUC and PUO, respectively.

The FMAP symbol can be assigned to specific CLB locations using LOC attributes. Refer to the "Mapping Constraint Examples" section of the "Attributes, Constraints, and Carry Logic" chapter and to the appropriate CAE tool interface user guide for more information on assigning LOC attributes.

FTC

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTC is a synchronous, resettable toggle flip-flop. The asynchronous clear (CLR) input, when High, overrides all other inputs and resets the data output (Q) Low. The Q output toggles, or changes state, when the toggle enable (T) input is High and CLR is Low during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CLR	T	C	Q
1	X	X	0
0	0	X	No Chg
0	1	↑	Toggle

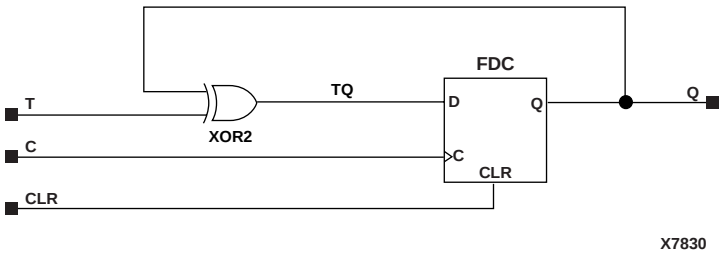


Figure 5-49 FTC Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

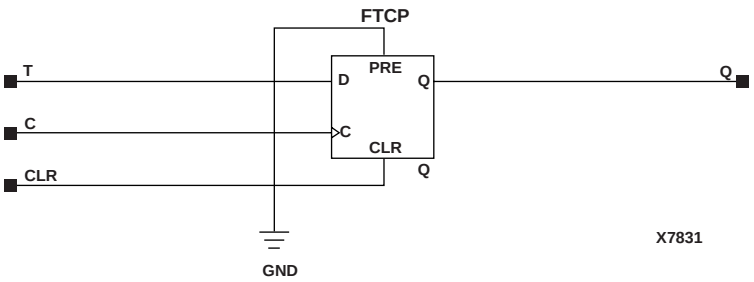
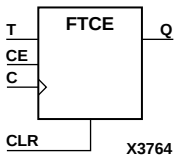


Figure 5-50 FTC Implementation XC9000

FTCE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTCE is a toggle flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the data output (Q) is reset Low. When CLR is Low and toggle enable (T) and clock enable (CE) are High, Q output toggles, or changes state, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	CE	T	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle

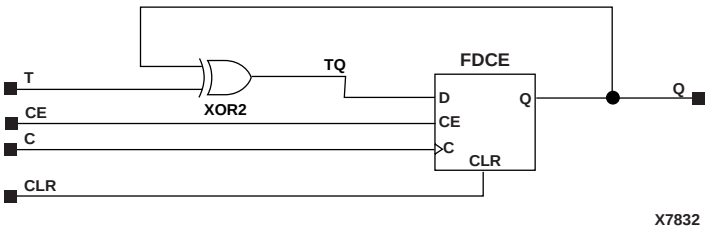


Figure 5-51 FTCE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

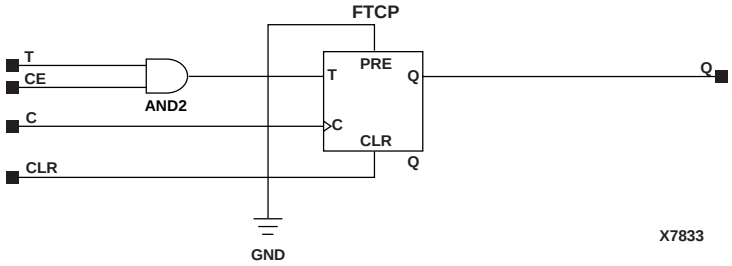
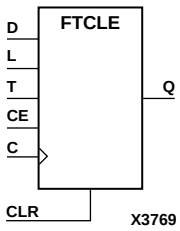


Figure 5-52 FTCE Implementation XC9000

FTCLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTCLE is a toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High and CLR is Low, clock enable (CE) is overridden and the data on data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	X	X	1	↑	1
0	1	X	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle

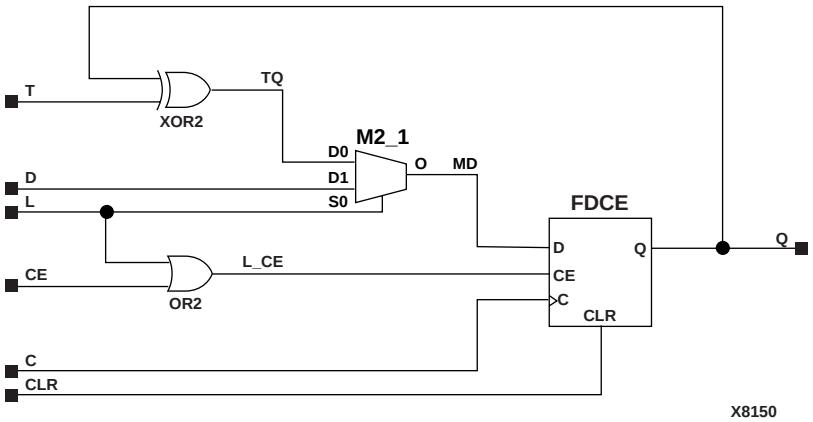


Figure 5-53 FTCLE Implementation XC3000

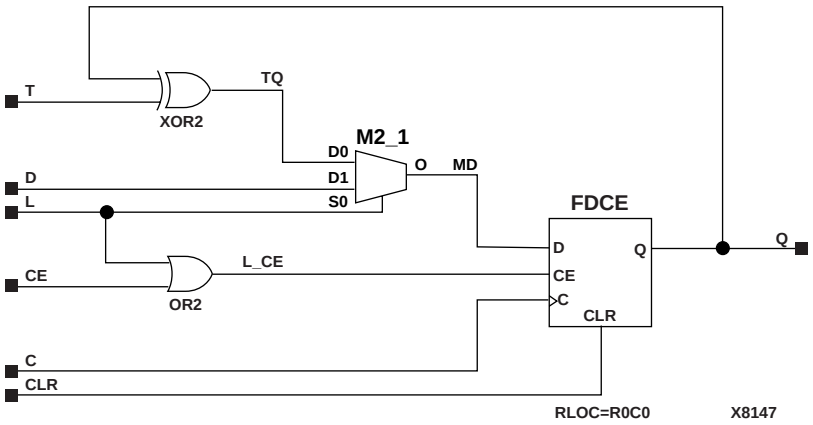


Figure 5-54 FTCLE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

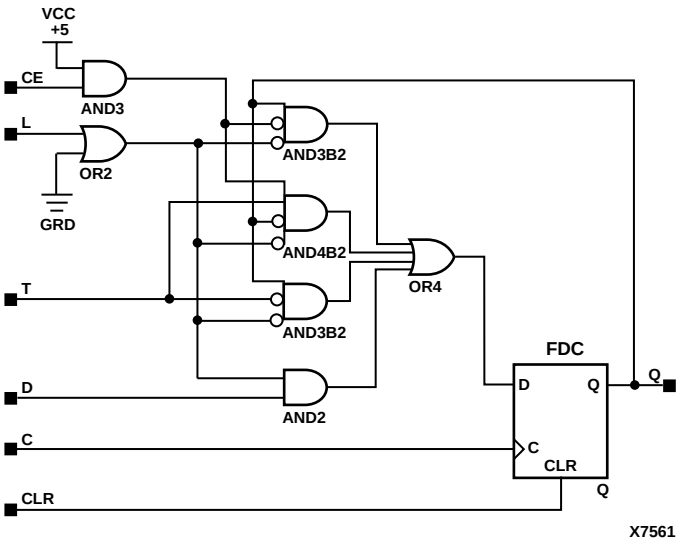
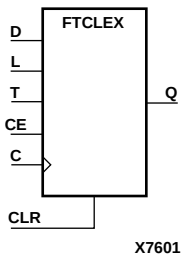


Figure 5-55 FTCLE Implementation XC9000

FTCLEX

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	N/A	Macro	Macro	Macro	Macro



FTCLEX is a toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High, CLR is Low, and CE is High, the data on data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	1	X	1	↑	1
0	1	1	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle

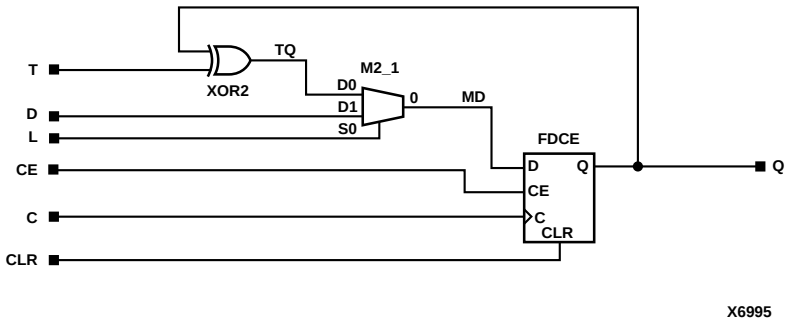
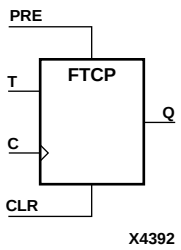


Figure 5-56 FTCLEX Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

FTCP

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A



FTCP is a toggle flip-flop with toggle enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. When the toggle enable input (T) is High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition.

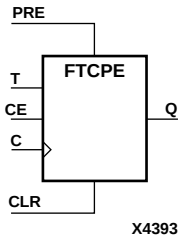
The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	T	C	Q
1	0	X	X	0
0	1	X	X	1
0	0	0	X	No Chg
0	0	1	↑	Toggle

FTCPE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	Macro	N / A	N / A	N / A	N / A



FTCPE is a toggle flip-flop with toggle and clock enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. When the toggle enable input (T) and the clock enable input (CE) are High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	CE	T	C	Q
1	0	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle

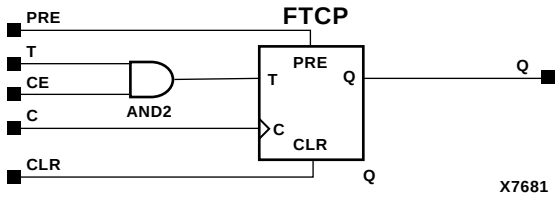
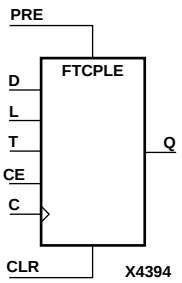


Figure 5-57 FTCPE Implementation XC9000

FTCPLE

Loadable Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	Macro	N/A	N/A	N/A	N/A



FTCPLE is a loadable toggle flip-flop with toggle and clock enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. The load input (L) loads the data on input D into the flip-flop on the Low-to-High clock transition, regardless of the state of the clock enable (CE). When the toggle enable input (T) and the clock enable input (CE) are High and CLR, PRE, and L are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs							Outputs
CLR	PRE	L	CE	T	C	D	Q
1	X	X	X	X	X	X	0
0	1	X	X	X	X	X	1
0	0	1	X	X	↑	0	0
0	0	1	X	X	↑	1	1
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	↑	X	Toggle

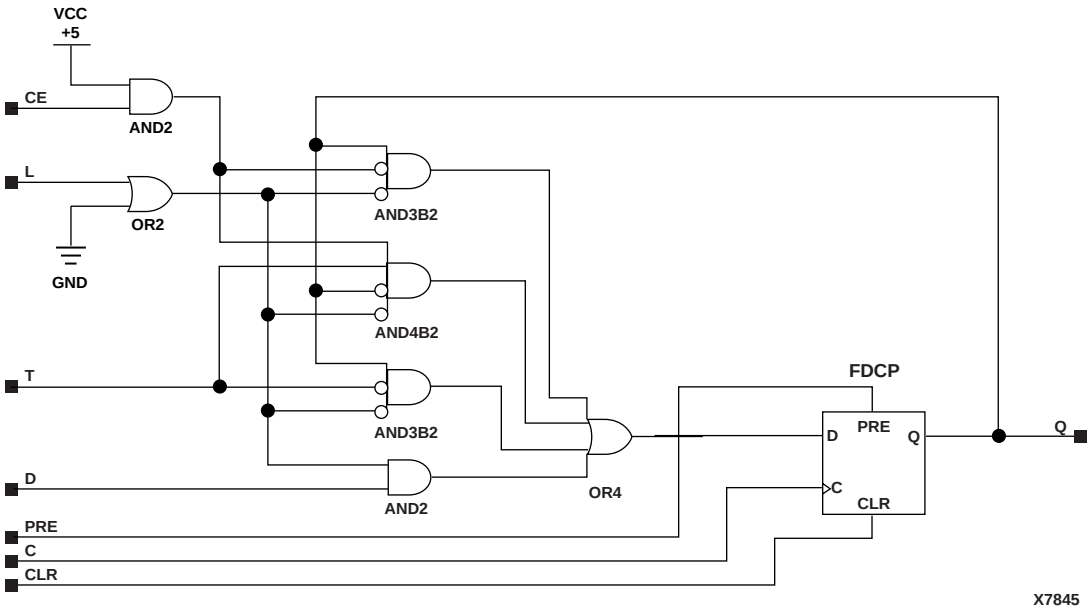
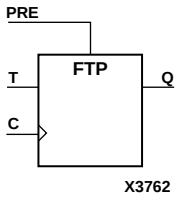


Figure 5-58 FTCPLE Implementation XC9000

FTP

Toggle Flip-Flop with Toggle Enable and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTP is a toggle flip-flop with toggle enable and asynchronous preset. When the asynchronous preset (PRE) input is High, all other inputs are ignored and output Q is set High. When toggle-enable input (T) is High and PRE is Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition.

For FPGAs, the flip-flop is asynchronously preset to output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The GR/GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or the STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
PRE	T	C	Q
1	X	X	1
0	0	X	No Chg
0	1	↑	Toggle

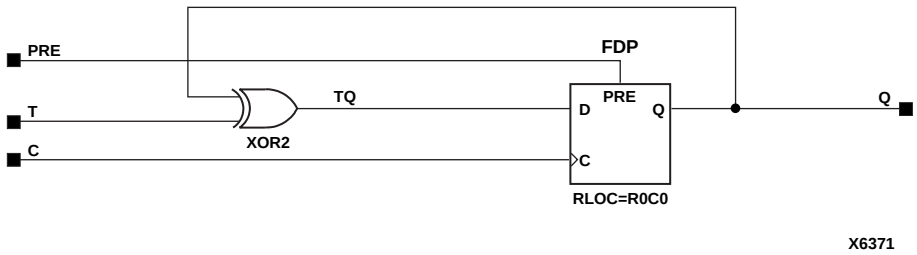


Figure 5-59 FTP Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

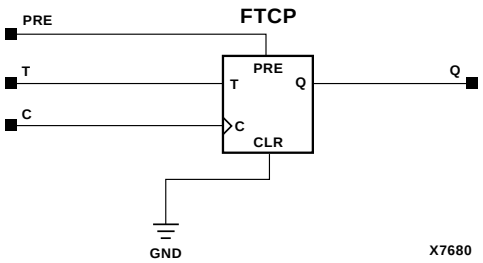
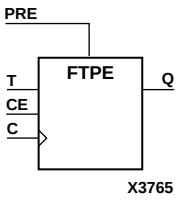


Figure 5-60 FTP Implementation XC9000

FTPE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTPE is a toggle flip-flop with toggle and clock enable and asynchronous preset. When the asynchronous preset (PRE) input is High, all other inputs are ignored and output Q is set High. When the toggle enable input (T) is High, clock enable (CE) is High, and PRE is Low, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

For FPGAs, the flip-flop is asynchronously preset to output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set / reset (GSR for XC4000, Spartans, Virtex) is active. The GR / GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
PRE	CE	T	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle

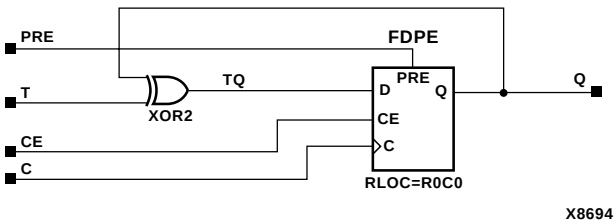


Figure 5-61 FTPE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

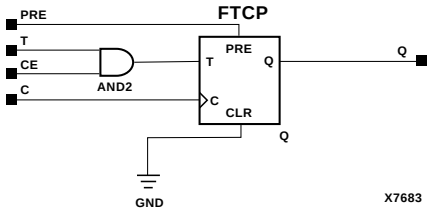
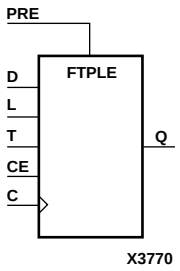


Figure 5-62 FTPE Implementation XC9000

FTPLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTPLE is a toggle/loadable flip-flop with toggle and clock enable and asynchronous preset. When the asynchronous preset input (PRE) is High, all other inputs are ignored and output Q is set High. When the load enable input (L) is High and PRE is Low, the clock enable (CE) is overridden and the data (D) is loaded into the flip-flop during the Low-to-High clock transition. When L and PRE are Low and toggle-enable input (T) and CE are High, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

For FPGAs, the flip-flop is asynchronously preset to output High, when power is applied. FPGAs simulate power-on when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The GR/GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs
PRE	L	CE	T	D	C	Q
1	X	X	X	X	X	1
0	1	X	X	1	↑	1
0	1	X	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle

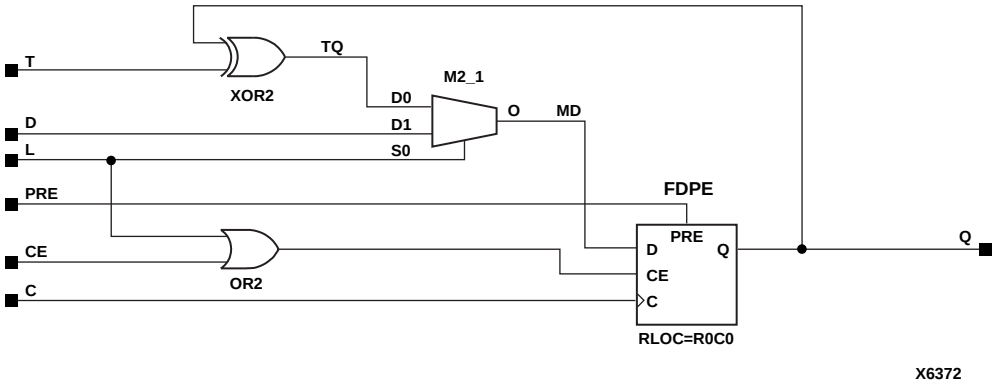


Figure 5-63 FTPLE Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

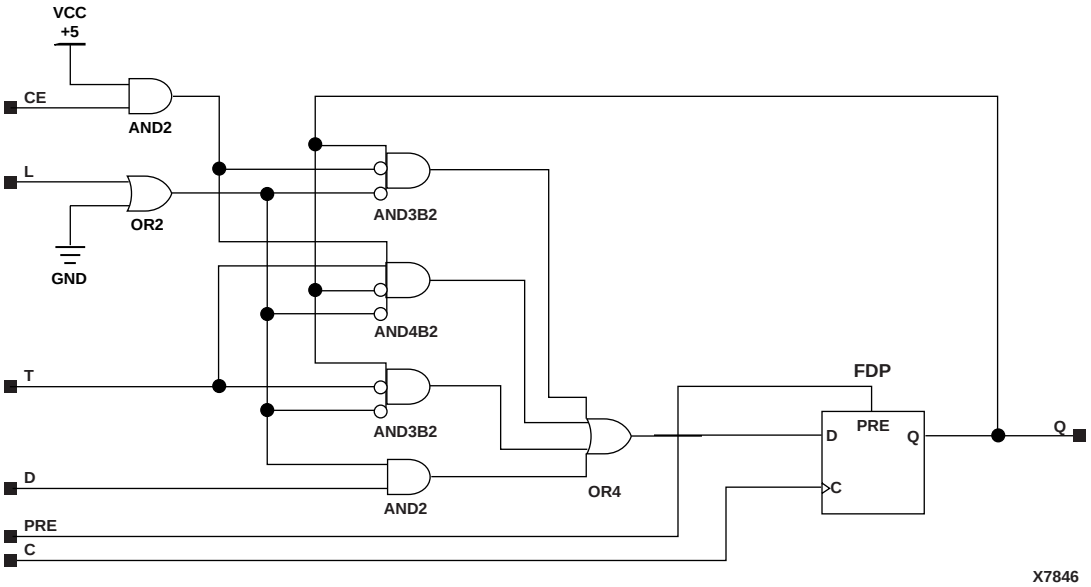
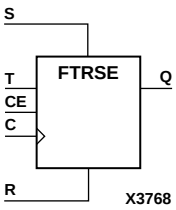


Figure 5-64 FTPLE Implementation XC9000

FTRSE

Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTRSE is a toggle flip-flop with toggle and clock enable and synchronous reset and set. When the synchronous reset input (R) is High, it overrides all other inputs and the data output (Q) is reset Low. When the synchronous set input (S) is High and R is Low, clock enable input (CE) is overridden and output Q is set High. (Reset has precedence over Set.) When toggle enable input (T) and CE are High and R and S are Low, output Q toggles, or changes state, during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs
R	S	CE	T	C	Q
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle

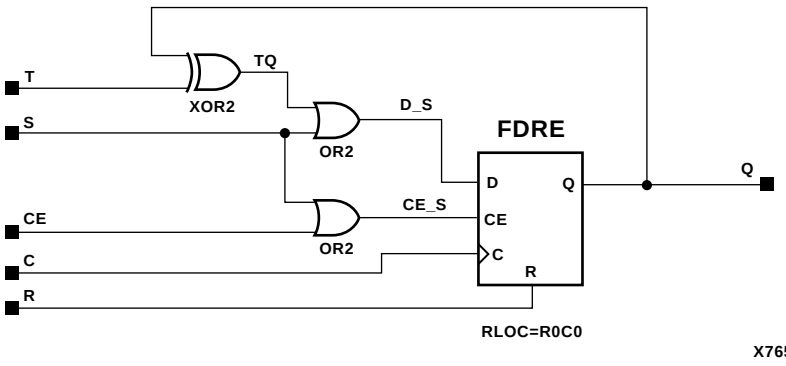


Figure 5-65 FTRSE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

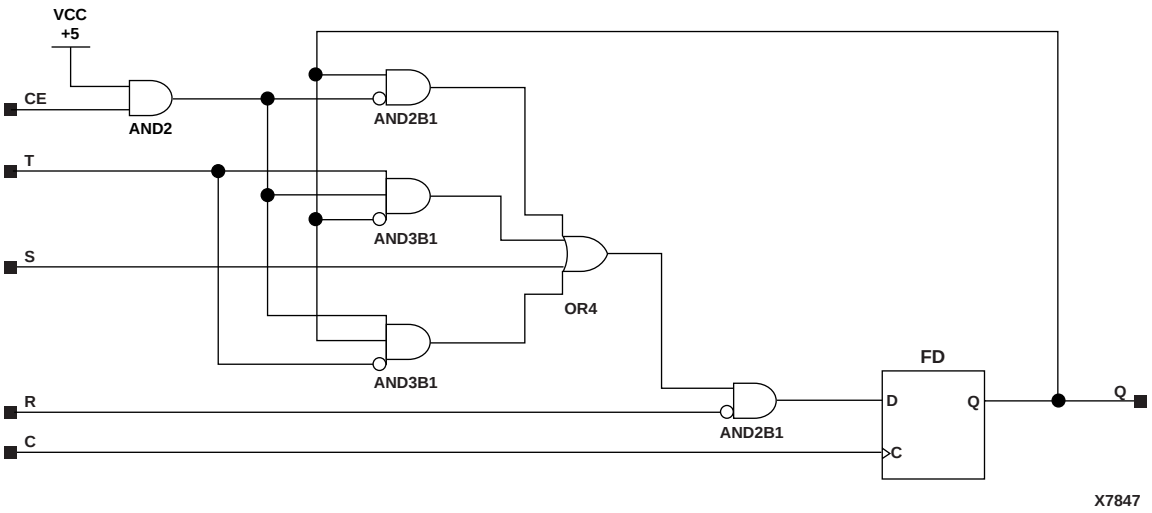
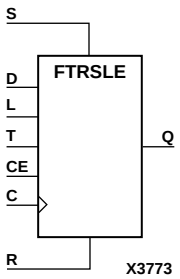


Figure 5-66 FTRSE Implementation XC9000

FTRSLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTRSLE is a toggle/loadable flip-flop with toggle and clock enable and synchronous reset and set. The synchronous reset input (R), when High, overrides all other inputs and resets the data output (Q) Low. (Reset has precedence over Set.) When R is Low and synchronous set input (S) is High, the clock enable input (CE) is overridden and output Q is set High. When R and S are Low and load enable input (L) is High, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When R, S, and L are Low and CE is High, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs							Outputs
R	S	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	0
0	1	X	X	X	X	↑	1
0	0	1	X	X	1	↑	1
0	0	1	X	X	0	↑	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle

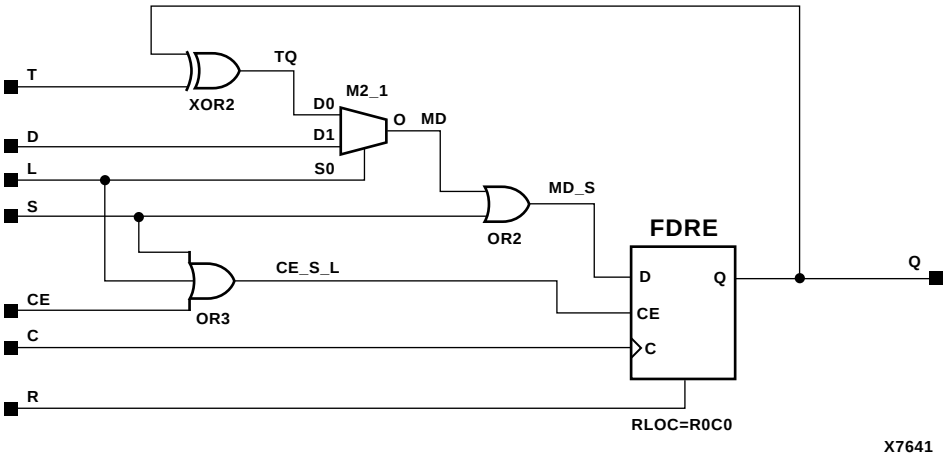


Figure 5-67 FTRSLE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

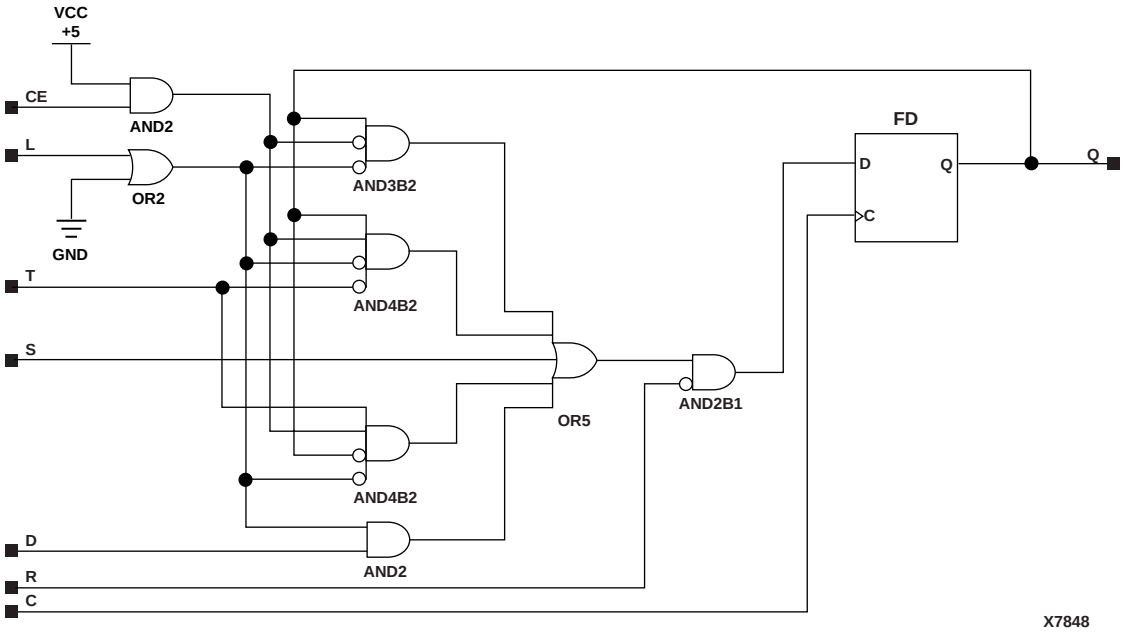
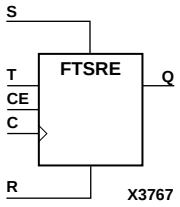


Figure 5-68 FTRSLE Implementation XC9000

FTSRE

Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTSRE is a toggle flip-flop with toggle and clock enable and synchronous set and reset. The synchronous set input, when High, overrides all other inputs and sets data output (Q) High. (Set has precedence over Reset.) When synchronous reset input (R) is High and S is Low, clock enable input (CE) is overridden and output Q is reset Low. When toggle enable input (T) and CE are High and S and R are Low, output Q toggles, or changes state, during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs					Outputs
S	R	CE	T	C	Q
1	X	X	X	↑	1
0	1	X	X	↑	0
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle

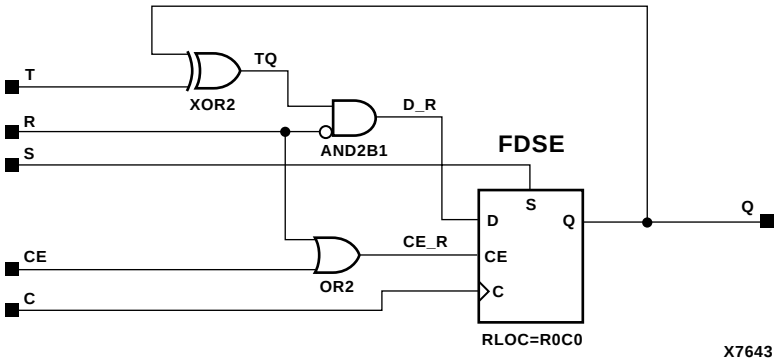


Figure 5-69 FTSRE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

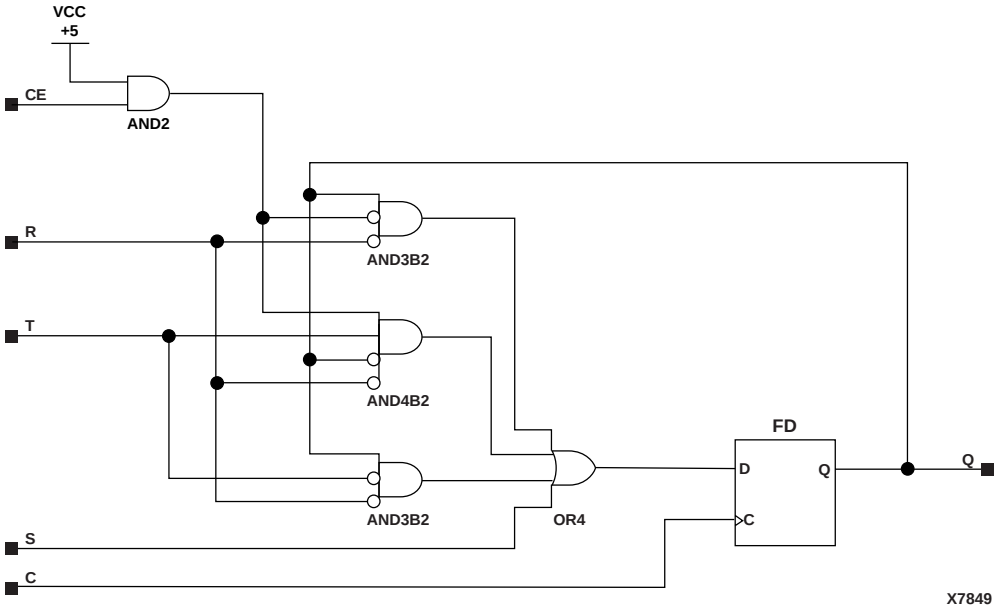
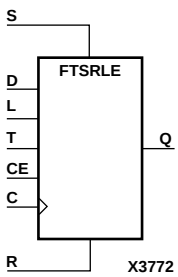


Figure 5-70 FTSRE Implementation XC9000

FTSRLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



FTSRLE is a toggle/loadable flip-flop with toggle and clock enable and synchronous set and reset. The synchronous set input (S), when High, overrides all other inputs and sets data output (Q) High. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, clock enable input (CE) is overridden and output Q is reset Low. When load enable input (L) is High and S and R are Low, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When the toggle enable input (T) and CE are High and S, R, and L are Low, output Q toggles, or changes state, during the Low-to- High clock transition. When CE is Low, clock transitions are ignored.

For FPGAs, the flip-flop is asynchronously cleared, output Low, when global reset (GR for XC5200) or global set/reset (GSR for XC4000, Spartans, Virtex) is active. The GR/GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For CPLDs, the flip-flop is asynchronously preset when a High-level pulse is applied on the PRLD global net.

Inputs							Outputs
S	R	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	1
0	1	X	X	X	X	↑	0
0	0	1	X	X	1	↑	1
0	0	1	X	X	0	↑	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle

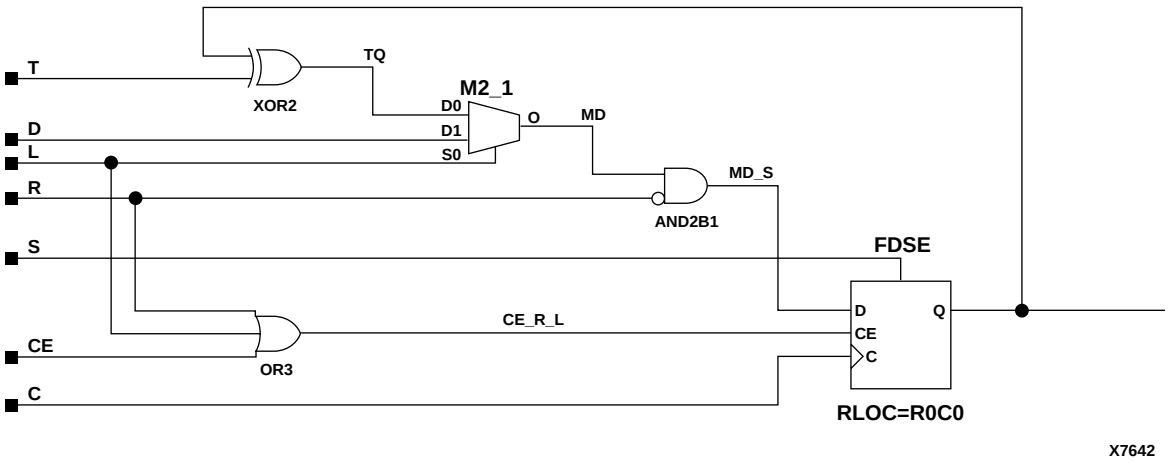


Figure 5-71 FTSRLE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

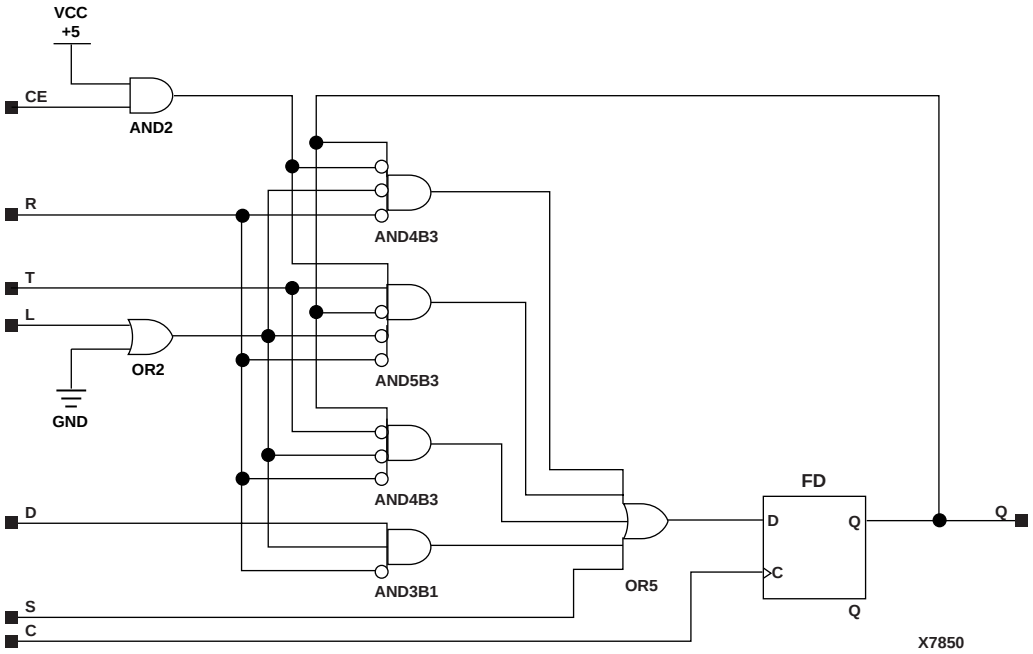


Figure 5-72 FTSRLE Implementation XC9000

Chapter 6

Design Elements (GCLK to KEEPER)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

GCLK

Global Clock Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

X3884

GCLK, the global clock buffer, distributes high fan-out clock signals. One GCLK buffer on each device provides direct access to every Configurable Logic Block (CLB) and Input Output Block (IOB) clock pin. If it is not used in a design, its routing resources are not used for any signals. Therefore, the GCLK should always be used for the highest fan-out clock net in the design. The GCLK input (I) can come from one of the following sources.

- From a CMOS-level signal on the dedicated TCLKIN pin (XC3000 only). TCLKIN is a direct CMOS-only input to the GCLK buffer. To use the TCLKIN pin, connect the input of the GCLK element to the IBUF and IPAD elements.
- From a CMOS or TTL-level external signal. To connect an external input to the GCLK buffer, connect the input of the GCLK element to the output of the IBUF for that signal. Unless the corresponding IPAD element is constrained otherwise, PAR typically places the IOB directly adjacent to the GCLK buffer.
- From an internal signal. To drive the GCLK buffer with an internal signal, connect that signal directly to the input of the GCLK element.

The output of the GCLK buffer can drive all the clock inputs on the chip, but it cannot drive non-clock inputs. For a negative-edge clock, insert an INV (inverter) element between the GCLK output and the clock input. This inversion is performed inside the CLB, or in the case of IOB clock pins, on the IOB clock line (which controls the clock sense for the IOBs on an entire edge of the chip).

GND

Ground-Connection Signal Tag

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

X3858

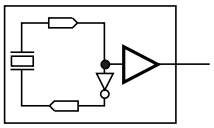
The GND signal tag, or parameter, forces a net or input function to a Low logic level. A net tied to GND cannot have any other source.

When the logic-trimming software or fitter encounters a net or input function tied to GND, it removes any logic that is disabled by the GND signal. The GND signal is only implemented when the disabled logic cannot be removed.

GXTL

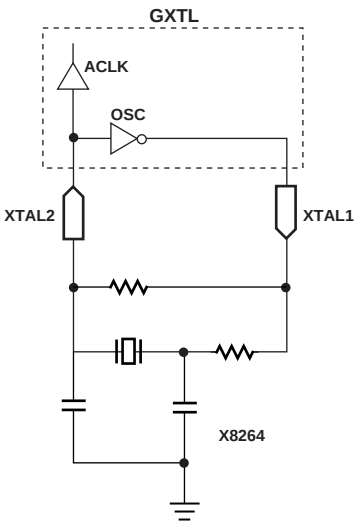
Crystal Oscillator with ACLK Buffer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A



The GXTL element drives an internal ACLK buffer with a frequency derived from an external crystal-controlled oscillator. The GXTL (or ACLK) output is connected to an internal clock net.

There are two dedicated input pins (XTAL 1 and XTAL 2) on each FPGA device that are internally connected to pads and input/output blocks that are in turn connected to the GXTL amplifier. The external components are connected as shown in the following figure.



Refer to *The Programmable Logic Data Book* for details on component selection and tolerances.

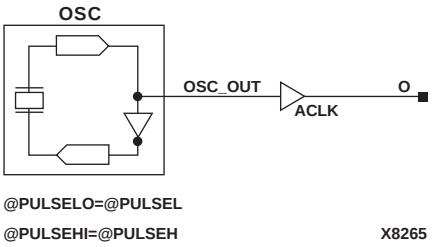
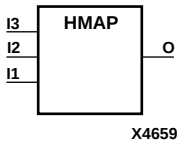


Figure 6-1 GXTL Implementation XC3000

HMAP

H Function Generator Partitioning Control Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	N/A	N/A



The HMAP symbol is used to control logic partitioning into 3-input H function generators for XC4000 and Spartans. It is usually used with FMAP, which partitions logic into F and G function generators. You can implement a portion of logic using gates, latches, and flip-flops and specify the logic to be grouped into F, G, and H function generators by naming logic signals and HMAP/FMAP signals correspondingly. These symbols are used for mapping control in addition to the actual gates, latches, and flip-flops and not as a substitute for them. The following figure gives an example of how logic can be placed using HMAP and FMAP symbols.

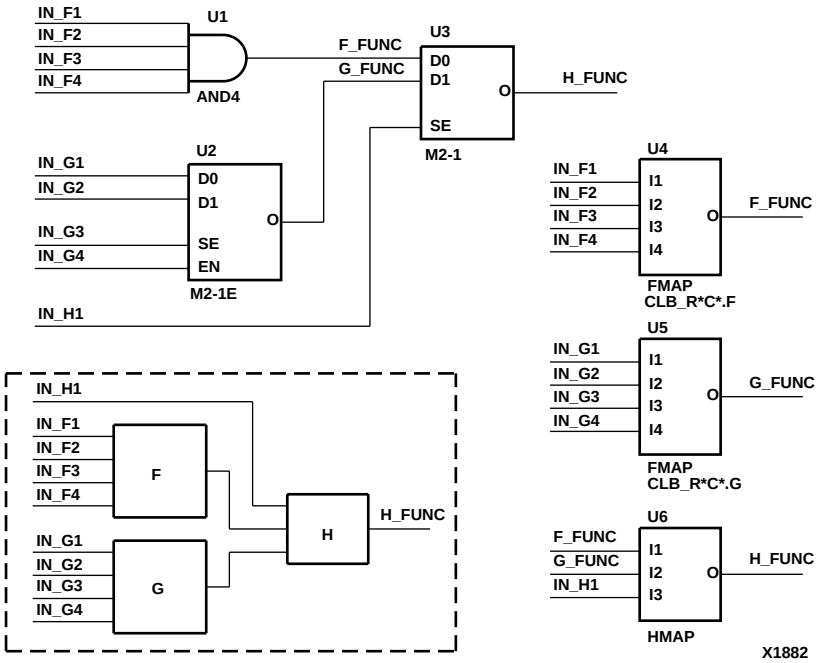


Figure 6-2 Partitioning Logic Using FMAP and HMAP Symbols

The MAP=*type* parameter can only be set to the default value, PUC, for the HMAP symbol. PUC means pins are not locked to the signals but the CLB is closed to addition or removal of logic.

The HMAP symbol can be assigned to specific CLB locations using LOC attributes. Refer to the "LOC" section of the "Attributes, Constraints, and Carry Logic" chapter for more information on assigning LOC attributes.

IBUF, 4, 8, 16

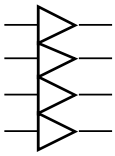
Single- and Multiple-Input Buffers

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
IBUF	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IBUF4, IBUF8, IBUF16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro

IBUF

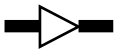
X3784

IBUF4



X3791

IBUF8



X3803

IBUF16

X3815

IBUF, IBUF4, IBUF8, and IBUF16 are single- and multiple-input buffers. An IBUF isolates the internal circuit from the signals coming into a chip. IBUFs are contained in input/output blocks (IOBs). IBUF inputs (I) are connected to an IPAD or an IOPAD. IBUF outputs (O) are connected to the internal circuit.

For Virtex and Spartan2, refer to the “IBUF_selectIO” section for information on IBUF variants with selectable I/O interfaces. IBUF, 4, 8, and 16 use the LVTTL standard.

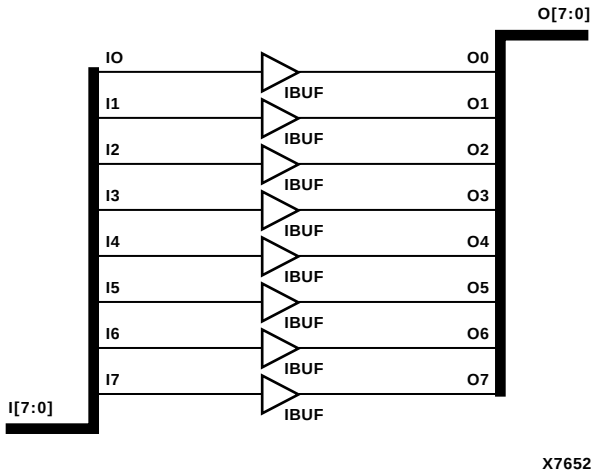
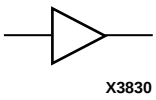


Figure 6-3 IBUF8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

IBUF_*selectIO*

Single Input Buffer with Selectable I/O Interface

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



For Virtex and Spartan2, IBUF and its variants (listed below) are single input buffers whose I/O interface corresponds to a specific I/O standard. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. For example, IBUF_SSTL3_II is a single input buffer that uses the SSTL3_II I/O-signaling standard.

An IBUF isolates the internal circuit from the signals coming into a chip. For Virtex and Spartan2, the dedicated GCLKIOB pad is input only. IBUF inputs (I) are connected to an IPAD or IOPAD. IBUF outputs (O) are connected to the internal circuit.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffer components. Refer to the "SelectI/O Usage Rules" section below for information on using these components.

Component	I/O Standard	VREF
IBUF	LVTTL	N/A
IBUF_LVCMOS2	LVCMOS2	N/A
IBUF_PCI33_3	PCI33_3	N/A
IBUF_PCI33_5	PCI33_5	N/A
IBUF_PCI66_3	PCI66_3	N/A
IBUF_GTL	GTL	0.80
IBUF_GTLP	GTL+	1.00
IBUF_HSTL_I	HSTL_I	0.75
IBUF_HSTL_III	HSTL_III	0.90
IBUF_HSTL_IV	HSTL_IV	0.75
IBUF_SSTL2_I	SSTL2_I	1.10
IBUF_SSTL2_II	SSTL2_II	1.10
IBUF_SSTL3_I	SSTL3_I	0.90
IBUF_SSTL3_II	SSTL3_II	1.50
IBUF_CTT	CTT	1.50
IBUF_AGP	AGP	1.32

SelectI/O Usage Rules

The Virtex and Spartan2 architectures include a versatile SelectI/O interface to multiple voltage and drive standards. To select an I/O standard, you must choose the appropriate component from the Virtex or Spartan2 library. Each standard has a full set of I/O buffer components (input, in/out, output, 3-state output). For example, for an input buffer of the GTL standard, you would choose IBUF_GTL. Refer to the "IBUF_selectIO", "IBUFG_selectIO", "IOBUF_selectIO", "OBUF_selectIO", and "OBUFT_selectIO" sections for information on the various input/output buffer components available to implement the desired standard.

The hardware implementation of the various I/O standards requires that certain usage rules be followed. As shown in the following table, each I/O standard has voltage source requirements for input reference (VREF), output drive (VCCO), or both. Each Virtex and Spartan2 device has eight banks (two on each edge). Each bank has voltage sources shared by all I/O in the bank. Therefore, in a particular bank, the voltage source (for either input or output) must be of the same type. The Input Banking (VREF) Rules section and the Output Banking (VCCO) Rules section below summarize the SelectI/O component usage rules based on the hardware implementation.

I/O Standard	VCCO	VREF
LVTTL	3.3	N/A
LVCMOS2	2.5	N/A
PCI33_3 (PCI 33MHz 3.3V)	3.3	N/A
PCI33_5 (PCI 33MHZ 5.0V)	3.3	N/A
PCI66_3 (PCI 66MHz 3.3V)	3.3	N/A
GTL	N/A	0.80
GTL+	N/A	1.00
HSTL_I	1.5	0.75
HSTL_III	1.5	0.90
HSTL_IV	1.5	0.75
SSTL2_I	2.5	1.10
SSTL2_II	2.5	1.10
SSTL3_I	3.3	0.90
SSTL3_II	3.3	1.50
CTT	3.3	1.50
AGP	3.3	1.32

Input Banking (VREF) Rules

The low-voltage I/O standards that have a differential amplifier input require a voltage reference input (VREF). The VREF voltage source is provided as an external signal to the chip that is banked internal to the chip.

- Any input buffer component that does not require a VREF source (LVTTL, LVCMOS2, PCI*) can be placed in any bank.
- All input buffer components that require a VREF source (GTL*, HSTL*, SSTL*, CTT, AGP) must be of the same I/O standard in a particular bank. For example,

IBUF_SSTL2_I and IBUFG_SSTL2_I are compatible since they are the same I/O standard (SSTL2_I).

- If the bank contains any input buffer component that requires a VREF source, the following conditions apply.
  - One or more VREF sources must be connected to the bank via an IOB.
  - The number of VREF sources is dependent on the device and package.
  - The locations of the VREF sources are fixed for each device/package.
  - All VREF sources must be used in that bank.
- If the bank contains no input buffer component that requires a VREF source, the IOBs for VREF sources can be used for general I/O.
- Output buffer components of any type can be placed in the bank.

Output Banking (VCCO) Rules

Because Virtex and Spartan2 have multiple low-voltage standards and also needs to be 5V tolerant, some control is required over the distribution of VCCO, the drive source voltage for output pins. To provide for maximum flexibility, the output pins are banked. In comparison to the VREF sources described above, the VCCO voltage sources are dedicated pins on the device and do not consume valuable IOBs.

- Any output buffer component that does not require a VCCO source (GTL, GTL+) can be placed in any bank.
- To be placed in a particular bank, all output buffer components that require VCCO must have the same supply voltage (VCCO). For example, OBUF_SSTL3_I and OBUF_PCI33_3 are compatible in the same output bank since VCCO=3.3 for both.
- Input buffer components of any type can be placed in the bank.
- The configuration pins on a Virtex and Spartan2 device are on the right side of the chip. When configuring the device through a serial prom, the user is required to use a VCCO of 3.3V in the two banks on the right hand side of the chip. If the user is not configuring the device through a serial prom, the VCCO requirement is dependent upon the configuration source.

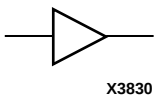
Banking Rules for OBUFT_*selectIO* with KEEPER

If a KEEPER symbol is attached to an OBUFT_*selectIO* component (3-state output buffer) for an I/O standard that requires a VREF (for example, OBUFT_GTL, OBUFT_SSTL3_I), then the OBUFT_*selectIO* component follows the same rules as an IOBUF_*selectIO* component for the same standard. It must follow both the input banking and output banking rules. The KEEPER element requires that the VREF be properly driven.

IBUFG_*selectIO*

Dedicated Input Buffer with Selectable I/O Interface

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



IBUFG and its variants (listed below) are dedicated input buffers for connecting to the clock buffer (BUFG) or CLKDLL. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the I/O interface standard used by the component. For example, IBUFG_CTT is an input buffer that uses the CTT I/O- signaling standard.

The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target PLD device. The IBUFG output can only be connected to the CLKIN input of a CLKDLL or to the input of a BUFG. The IBUFG can only be driven by an IPAD.

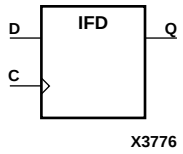
The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffer components. Refer to the "SelectI/O Usage Rules" section under the IBUF_*selectIO* section for information on using these components.

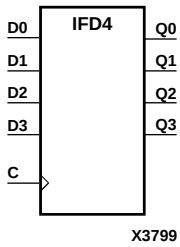
Component	I/O Standard	VREF
IBUFG	LVTTL	N/A
IBUFG_LVCMOS2	LVCMOS2	N/A
IBUFG_PCI33_3	PCI33_3	N/A
IBUFG_PCI33_5	PCI33_5	N/A
IBUFG_PCI66_3	PCI66_3	N/A
IBUFG_GTL	GTL	0.80
IBUFG_GTLP	GTL+	1.00
IBUFG_HSTL_I	HSTL_I	0.75
IBUFG_HSTL_III	HSTL_III	0.90
IBUFG_HSTL_IV	HSTL_IV	0.75
IBUFG_SSTL2_I	SSTL2_I	1.10
IBUFG_SSTL2_II	SSTL2_II	1.10
IBUFG_SSTL3_I	SSTL3_I	0.90
IBUFG_SSTL3_II	SSTL3_II	1.50
IBUFG_CTT	CTT	1.50
IBUFG_AGP	AGP	1.32

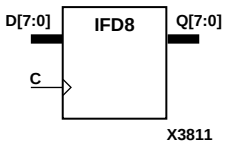
IFD, 4, 8, 16

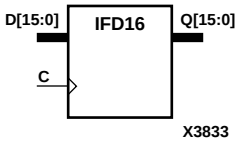
Single- and Multiple-Input D Flip-Flops

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
IFD	Primitive	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro
IFD4, IFD8, IFD16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









The IFD D-type flip-flop is contained in an input / output block (IOB), except for XC5200 and XC9000. The input (D) of the flip-flop is connected to an IPAD or an IOPAD (without using an IBUF). The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flops are asynchronously cleared with Low outputs when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFD, IFD_1, ILD, and ILD_1 combinations, refer to the “ILD, 4, 8, 16” section.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1

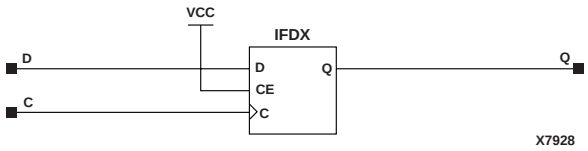


Figure 6-4 IFD Implementation XC4000E, XC4000X, Spartan, SpartanXL

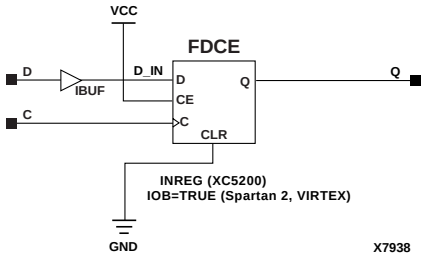


Figure 6-5 IFD Implementation XC5200, Spartan2, Virtex

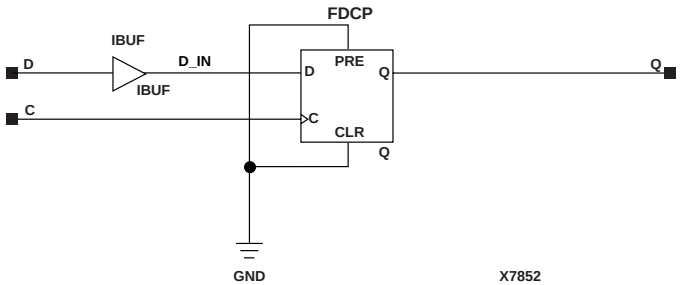


Figure 6-6 IFD Implementation XC9000

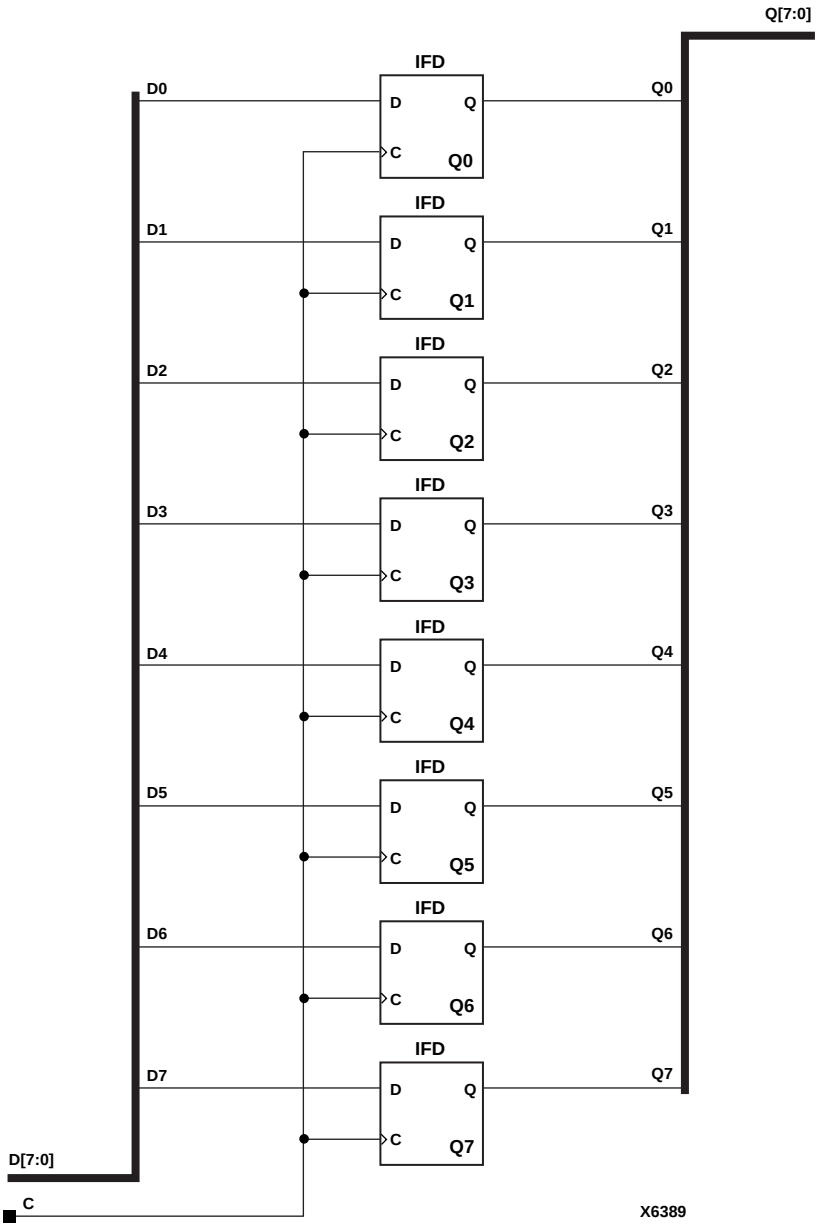
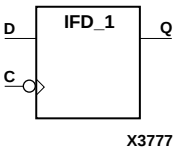


Figure 6-7 IFD8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

IFD_1

Input D Flip-Flop with Inverted Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Macro	Macro



The IFD_1 D-type flip-flop is contained in an input/output block (IOB) except for XC5200. The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input also provides data input for the flip-flop, which synchronizes data entering the chip. The D input data is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFD, IFD_1, ILD, and ILD_1 combinations, refer to the “ILD, 4, 8, 16” section.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1

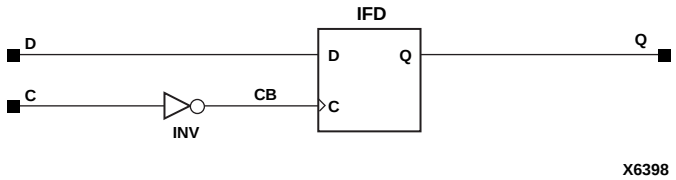


Figure 6-8 IFD_1 Implementation XC3000, XC4000E, XC4000X, Spartan, SpartanXL

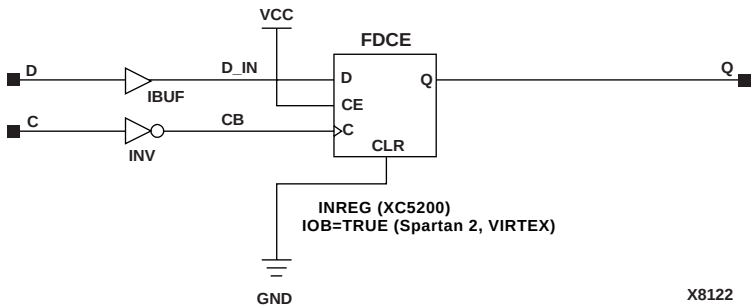
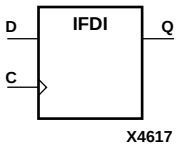


Figure 6-9 IFD_1 Implementation XC5200, Spartan2, Virtex

IFDI

Input D Flip-Flop (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



The IFDI D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDI, IFDI_1, ILDI, and ILDI_1 combinations, refer to the "ILDI" section.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1

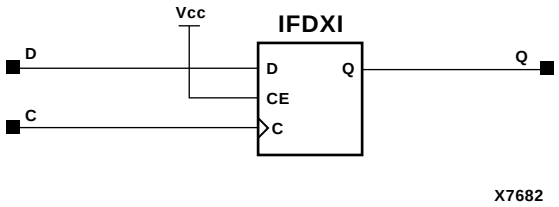


Figure 6-10 IFDI Implementation XC4000E, XC4000X, Spartan, SpartanXL

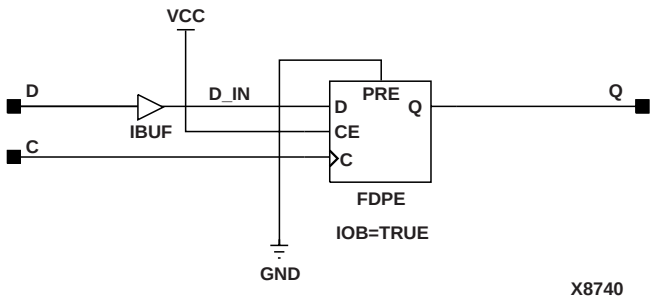
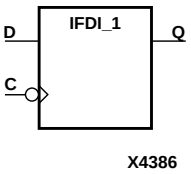


Figure 6-11 IFDI Implementation Spartan2, Virtex

IFDI_1

Input D Flip-Flop with Inverted Clock (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



The IFDI_1 D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDI, IFDI_1, ILDI, and ILDI_1 combinations, refer to the "ILDI" section.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1

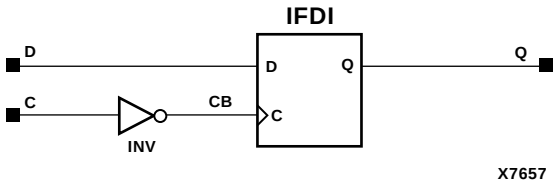


Figure 6-12 IFDI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

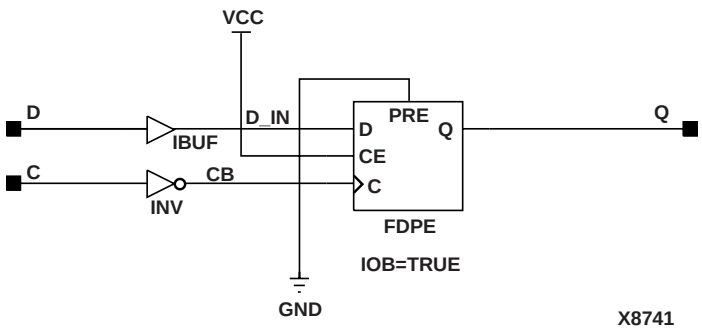
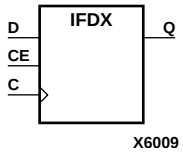


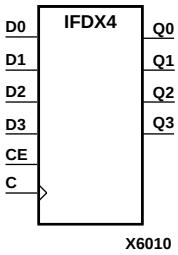
Figure 6-13 IFDI_1 Implementation Spartan2, Virtex

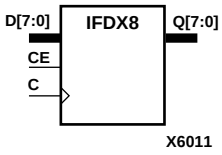
IFDX, 4, 8, 16

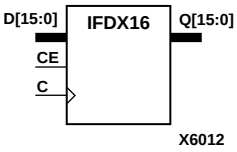
Single- and Multiple-Input D Flip-Flops with Clock Enable

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
IFDX	N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Macro	Macro
IFDX4, IFDX8, IFDX16	N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro









The IFDX D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD (without using an IBUF). The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When CE is Low, flip-flop outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDX, IFDX_1, ILDX, and ILDX_1 combinations, refer to the "ILDX, 4, 8, 16" section.

Inputs			Outputs
CE	Dn	C	Qn
1	Dn	↑	dn
0	X	X	No Chg

dn = state of referenced input (Dn) one setup time prior to active clock transition

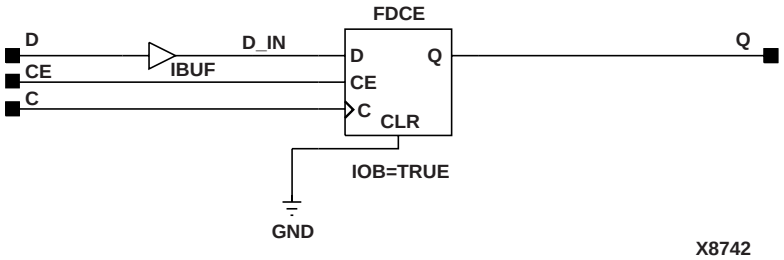


Figure 6-14 IFDX Implementation Spartan2, Virtex

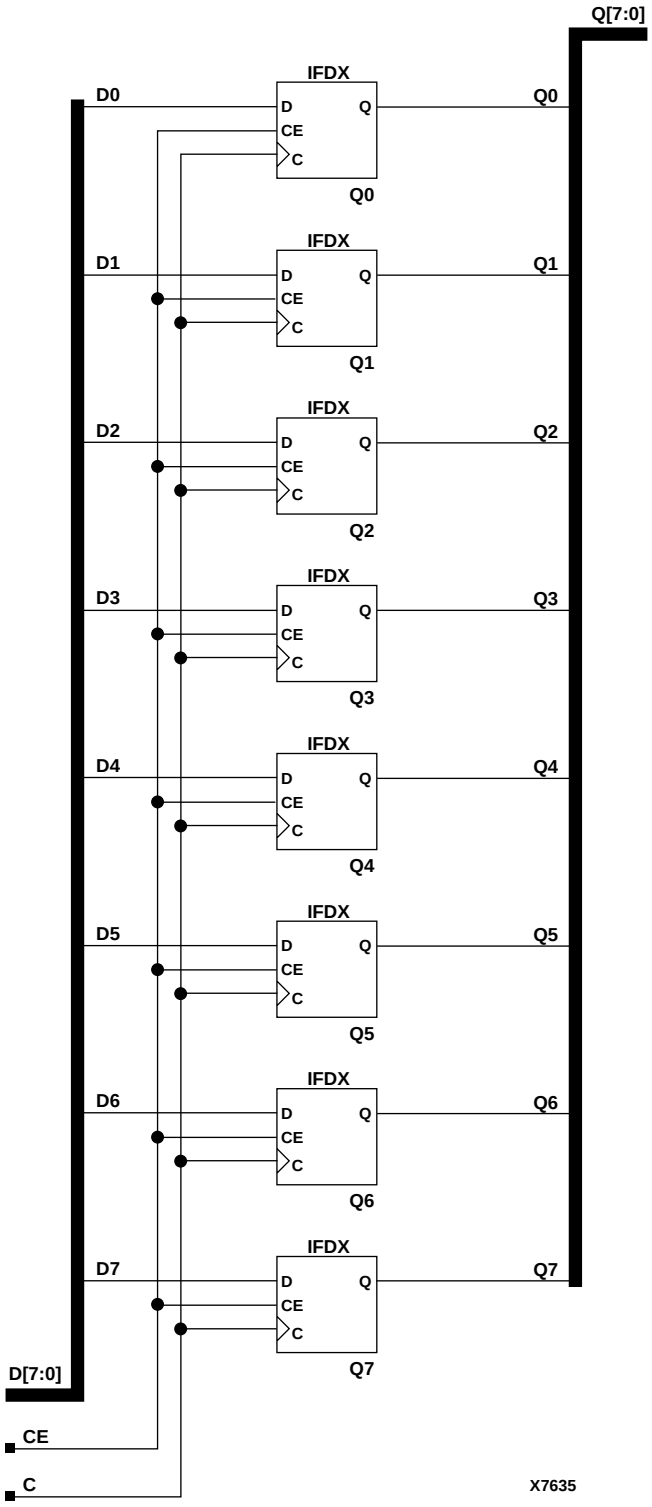
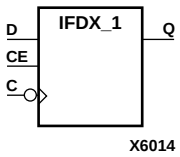


Figure 6-15 IFDX8 Implementation XC4000E, XC4000X, Spartan, SpartanXL, Spartan2, Virtex

IFDX_1

Input D Flip-Flop with Inverted Clock and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro



The IFDX_1 D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input also provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously cleared with Low output, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDX, IFDX_1, ILDX, and ILDX_1 combinations, refer to the "ILDX, 4, 8, 16" section.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of D input one setup time prior to active clock transition

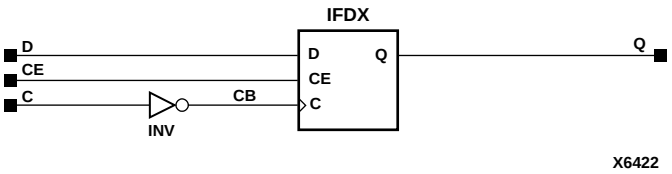


Figure 6-16 IFDX_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

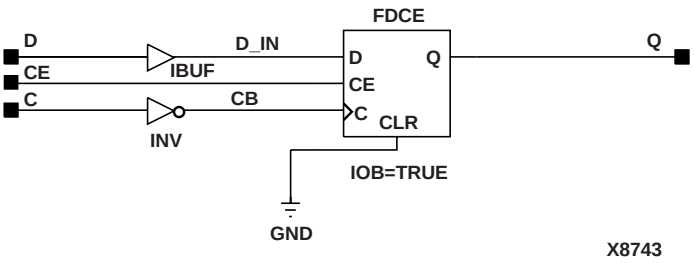
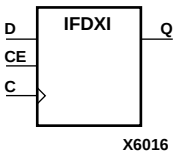


Figure 6-17 IFDX_1 Implementation Spartan2, Virtex

IFDXI

Input D Flip-Flop with Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Macro	Macro



The IFDXI D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously preset with High output, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, refer to the “ILDXI” section.

Inputs			Outputs
CE	D	C	Q
1	D	↑	d
0	X	X	No Chg

d = state of D input one setup time prior to active clock transition

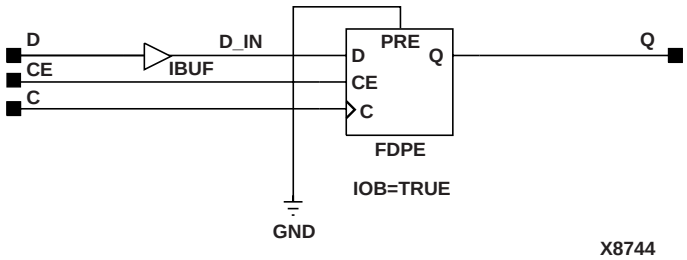
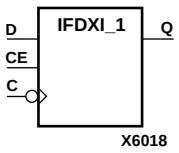


Figure 6-18 IFDXI Implementation Spartan2, Virtex

IFDXI_1

Input D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro



The IFDXI_1 D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously preset with High output when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, refer to the “ILDXI” section.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of D input one setup time prior to active clock transition

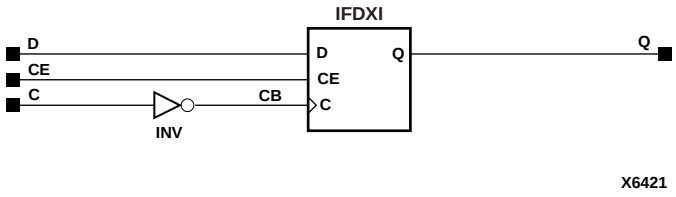


Figure 6-19 IFDXI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

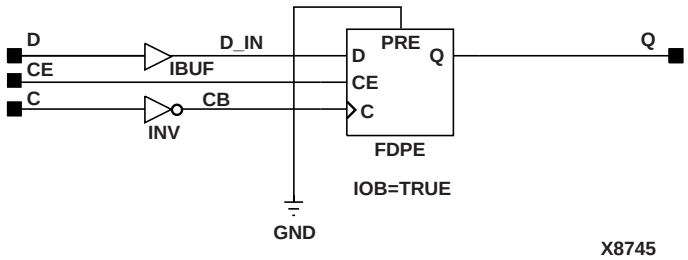
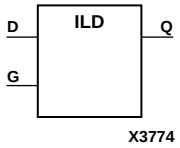


Figure 6-20 IFDXI_1 Implementation Spartan2, Virtex

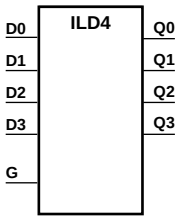
ILD, 4, 8, 16

Transparent Input Data Latches

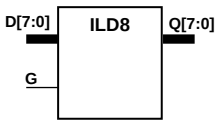
Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
ILD	Primitive	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro
ILD4, ILD8, ILD16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



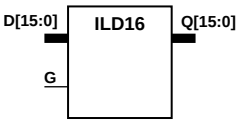
X3774



X3798



X3810



X3832

ILD, ILD4, ILD8, and ILD16 are single or multiple transparent data latches, which can be used to hold transient data entering a chip. The ILD latch is contained in an input / output block (IOB), except for XC5200 and XC9000. The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (G) is High, data on the inputs (D) appears on the outputs (Q). Data on the D inputs during the High-to-Low G transition is stored in the latch.

The latch is asynchronously cleared with Low output when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

ILDs and IFDs for XC3000

The XC3000 ILD is actually the input flip-flop master latch. If both ILD and IFD elements are controlled by the same clock signal, the relationship between the transparent sense of the latch and the active edge of the flip-flop is fixed as follows: a transparent High latch (ILD) corresponds to a falling edge-triggered flip-flop (IFD_1), and a transparent Low latch (ILD_1) corresponds to a rising edge-triggered flip-flop (IFD). Because the place and route software does not support using both phases of a clock for IOBs on a single edge of the device, certain combinations of ILD and IFD elements are not allowed.

Refer to the following figure for legal IFD, IFD_1, ILD, and ILD_1 combinations for the XC3000.

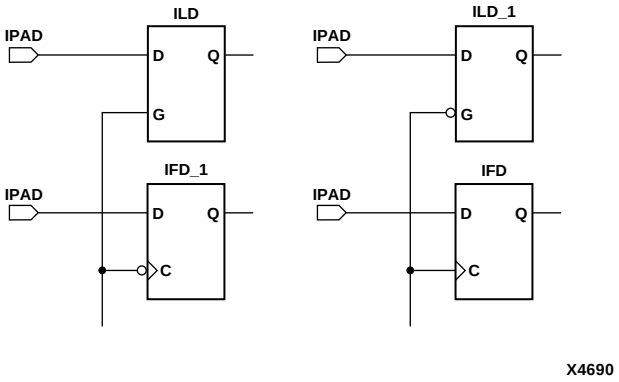


Figure 6-21 Legal Combinations of IFD and ILD for a Single Device Edge of an XC3000 IOB

ILDs and IFDs for XC4000E, XC4000X, Spartan, SpartanXL

In XC4000, Spartan, and SpartanXL, the ILD is actually the input flip-flop master latch. It is possible to access two different outputs from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILD) corresponds to a falling edge-triggered flip-flop (IFD_1). Similarly, a transparent Low latch (ILD_1) corresponds to a rising edge-triggered flip-flop (IFD).

Refer to the following figure for legal IFD, IFD_1, ILD, and ILD_1 combinations for the XC4000, Spartan, and SpartanXL.

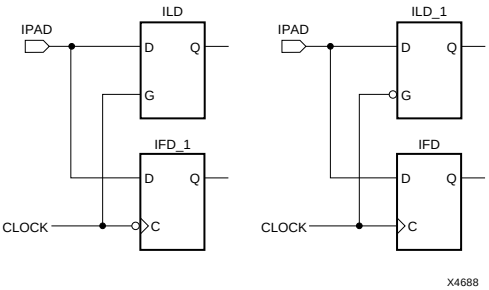


Figure 6-22 Legal Combinations of IFD and ILD for a Single IOB in XC4000E, XC4000X, Spartan, or SpartanXL

Inputs		Outputs
G	**D**	**Q**
1	1	1
1	0	0
0	X	d

d = state of referenced input one setup time prior to active G transition

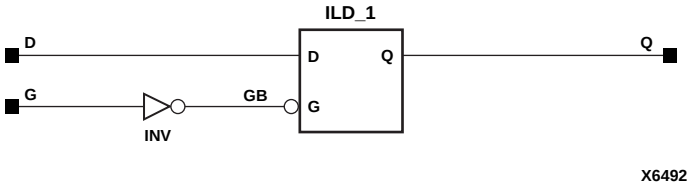


Figure 6-23 ILD Implementation XC4000E, XC4000X, Spartan, SpartanXL

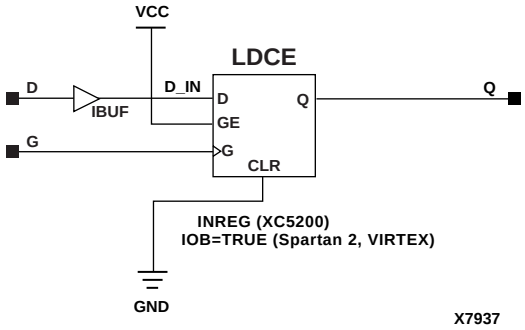


Figure 6-24 ILD Implementation XC5200, Spartan2, Virtex

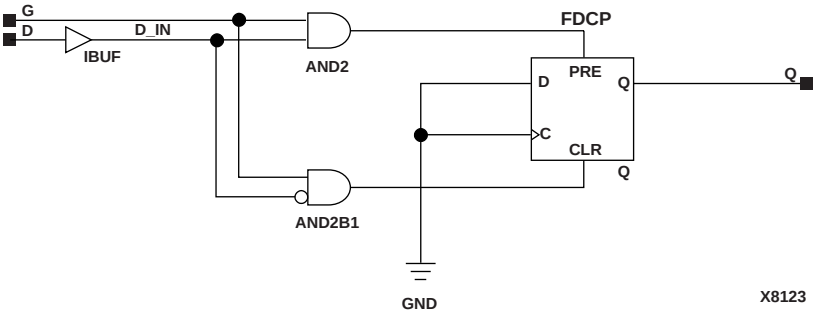


Figure 6-25 ILD Implementation XC9000

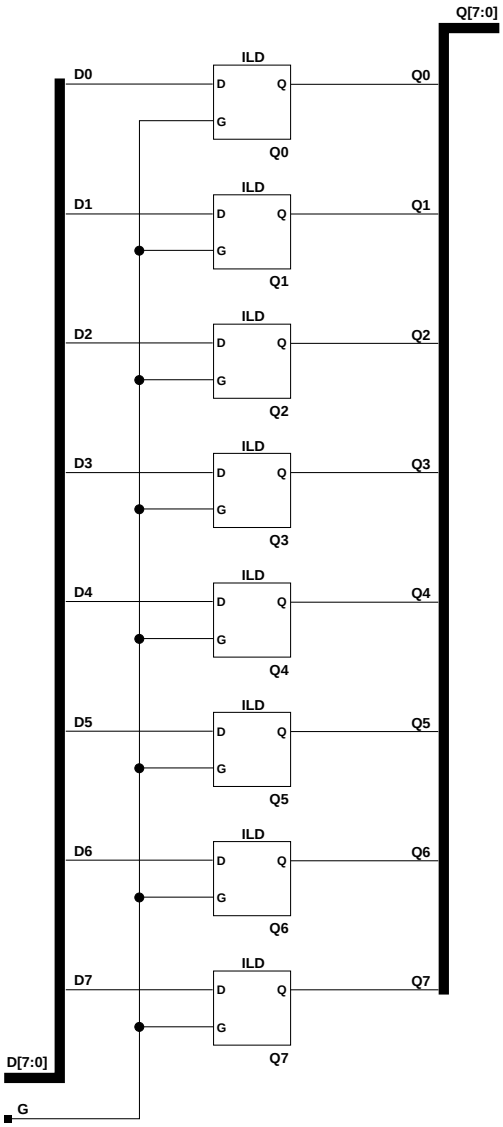
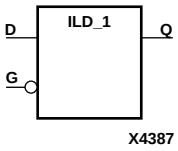


Figure 6-26 ILD8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

ILD_1

Transparent Input Data Latch with Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Macro	Macro



ILD_1 is a transparent data latch, which can be used to hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFD, IFD_1, ILD, and ILD_1 combinations, refer to the “ILD, 4, 8, 16” section.

Inputs		Outputs
G	D	Q
0	1	1
0	0	0
1	X	d

d = state of referenced input one setup time prior to Low-to-High gate transition

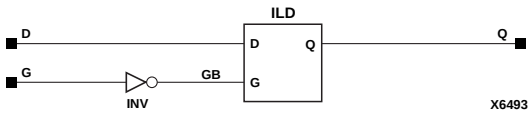


Figure 6-27 ILD_1 Implementation XC3000

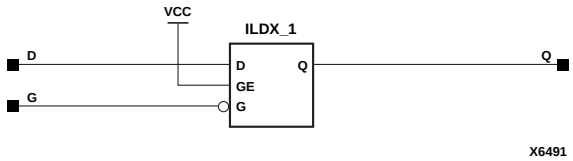


Figure 6-28 ILD_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

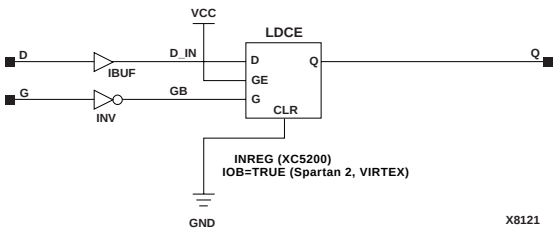
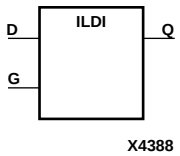


Figure 6-29 ILD_1 Implementation XC5200, Spartan2, Virtex

ILDI

Transparent Input Data Latch (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



ILDI is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is High, data on the input (D) appears on the output (Q). Data on the D input during the High-to-Low G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

ILDIs and IFDIs

The ILDI is actually the input flip-flop master latch. It is possible to access two different outputs from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDI) corresponds to a falling edge-triggered flip-flop (IFDI_1). Similarly, a transparent Low latch (ILDI_1) corresponds to a rising edge-triggered flip-flop (IFDI).

Refer to the following figure for legal IFDI, IFDI_1, ILDI, and ILDI_1 combinations.

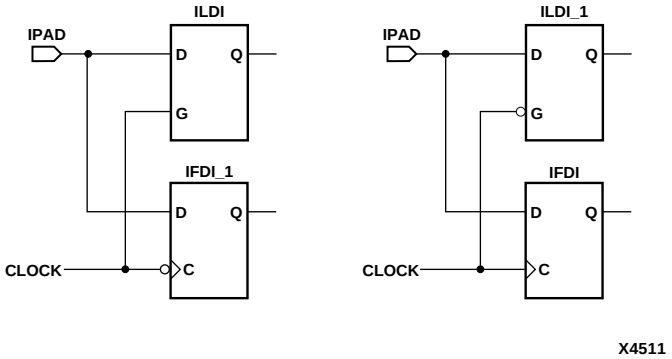


Figure 6-30 Legal Combinations of IFDI and ILDI for a Single IOB in XC4000E, XC4000X, Spartan, or SpartanXL

Inputs		Outputs
G	D	Q
1	1	1
1	0	0
0	X	d

d = state of referenced input one setup time prior to High-to-Low gate transition

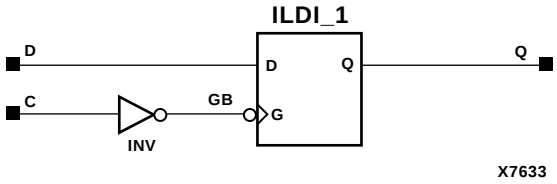


Figure 6-31 ILDI Implementation XC4000E, XC4000X, Spartan, SpartanXL

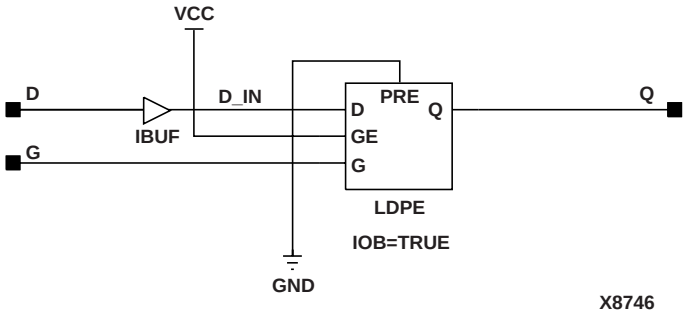
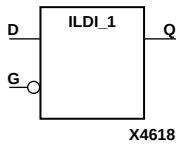


Figure 6-32 ILDI Implementation Spartan2, Virtex

ILDI_1

Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



ILDI_1 is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDI, IFDI_1, ILDI, and ILDI_1 combinations, refer to the "ILDI" section.

Inputs		Outputs
G	D	Q
0	1	1
0	0	0
1	X	d

d = state of input one setup time prior to High-to-Low gate transition

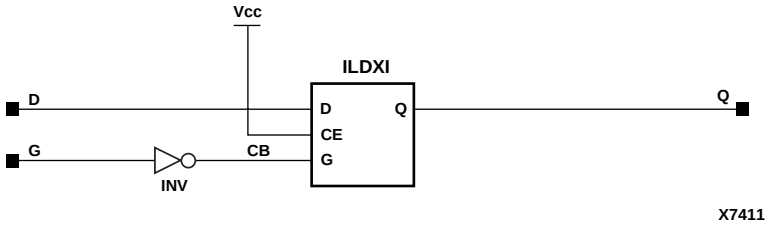


Figure 6-33 ILDI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

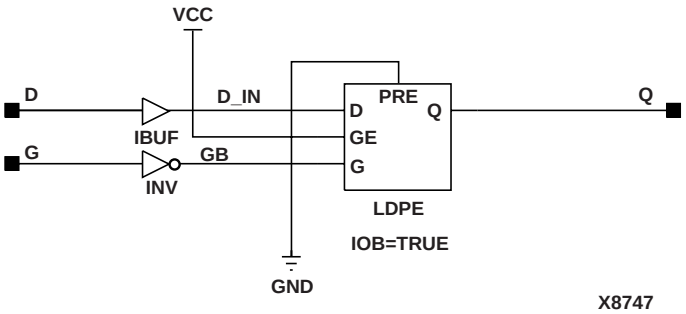
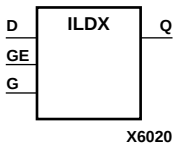


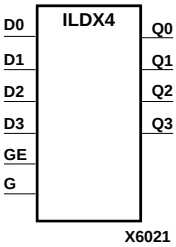
Figure 6-34 ILDI_1 Implementation Spartan2, Virtex

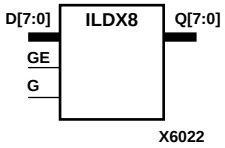
ILDX, 4, 8, 16

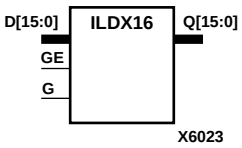
Transparent Input Data Latches

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
ILDX	N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro
ILDX4, ILDX8, ILDX16	N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro









ILDX, ILDX4, ILDX8, and ILDX16 are single or multiple transparent data latches, which can be used to hold transient data entering a chip. The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (G) is High, data on the inputs (D) appears on the outputs (Q). Data on the D inputs during the High-to-Low G transition is stored in the latch.

The latch is asynchronously cleared, output Low, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

ILDXs and IFDXs

The ILDX is actually the input flip-flop master latch. Two different outputs can be accessed from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDX) corresponds to a falling edge-triggered flip-flop (IFDX_1). Similarly, a transparent Low latch (ILDX_1) corresponds to a rising edge-triggered flip-flop (IFDX).

Refer to the following figure for legal IFDX, IFDX_1, ILDX, and ILDX_1 combinations.

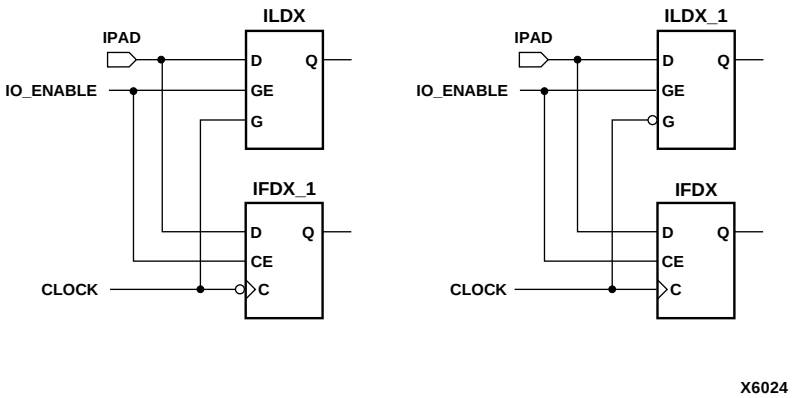


Figure 6-35 Legal Combinations of IFDX and ILDX for a Single IOB in XC4000E, XC4000X, Spartan, or SpartanXL

Inputs			Outputs
GE	**G**	**D**	**Q**
0	X	X	No Chg
1	0	X	No Chg
1	1	1	1
1	1	0	0
1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

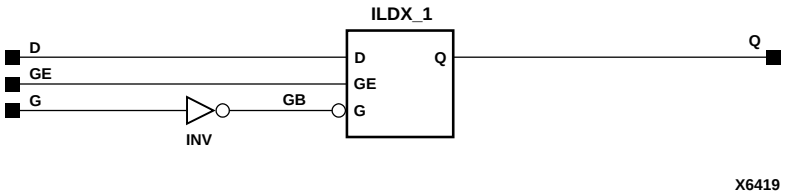


Figure 6-36 ILDX Implementation XC4000E, XC4000X, Spartan, SpartanXL

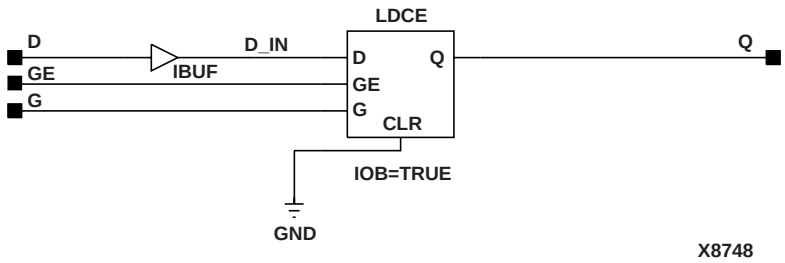


Figure 6-37 ILDX Implementation Spartan2, Virtex

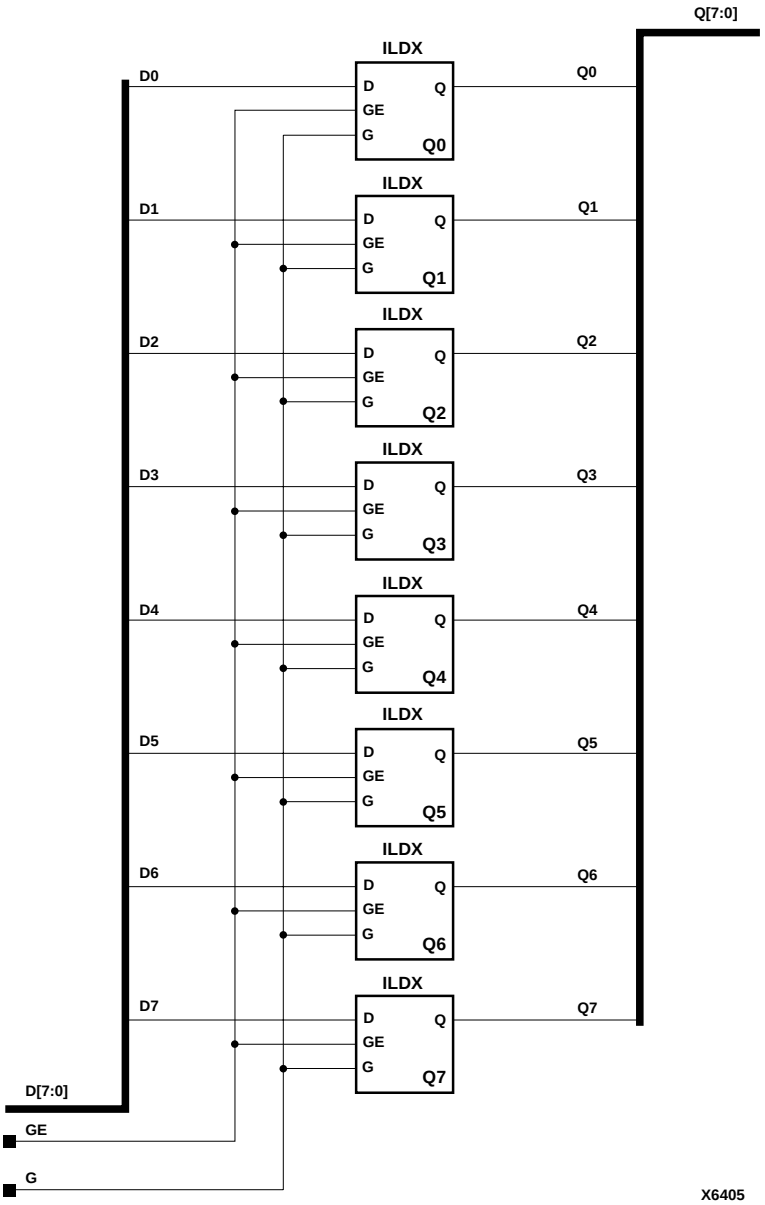
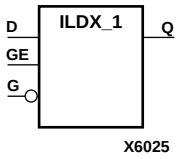


Figure 6-38 ILDX8 Implementation XC4000E, XC4000X, Spartan, SpartanXL, Spartan2, Virtex

ILDX_1

Transparent Input Data Latch with Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Macro	Macro



ILDX_1 is a transparent data latch, which can be used to hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously cleared with Low output, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDX, IFDX_1, ILDX, and ILDX_1 combinations, refer to the "ILDX, 4, 8, 16" section.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	X	No Chg
1	0	1	1
1	0	0	0
1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

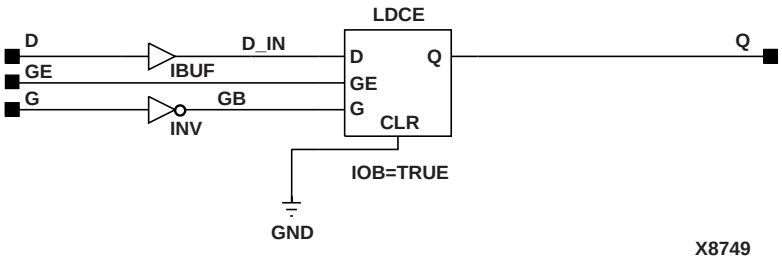
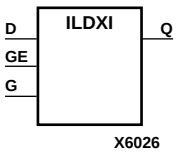


Figure 6-39 ILDX_1 Implementation Spartan2, Virtex

ILDXI

Transparent Input Data Latch (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



ILDXI is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is High, data on the input (D) appears on the output (Q). Data on the D input during the High-to-Low G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

ILDXIs and IFDXIs

The ILDXI is actually the input flip-flop master latch. Two different outputs can be accessed from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDXI) corresponds to a falling edge-triggered flip-flop (IFDXI_1). Similarly, a transparent Low latch (ILDXI_1) corresponds to a rising edge-triggered flip-flop (IFDXI). Refer to the following figure for legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations.

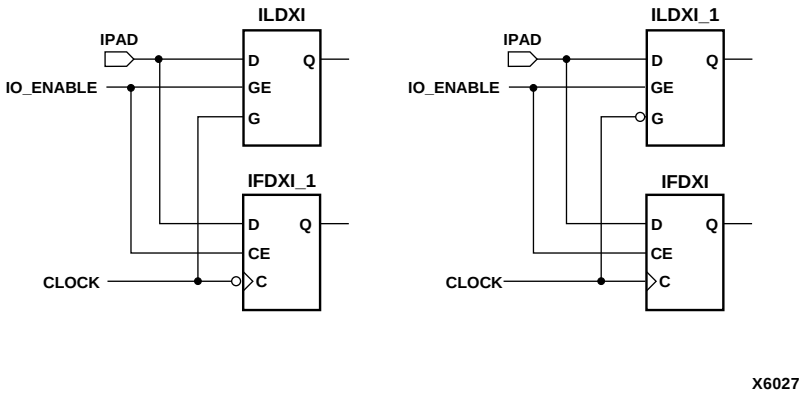


Figure 6-40 Legal Combinations of IFDXI and ILDXI for a Single IOB

Inputs			Outputs
GE	**G**	**D**	**Q**
0	X	X	No Chg
1	0	X	No Chg
1	1	1	1
1	1	0	0
1	↓	D	d

d = state of referenced input one setup time prior to High-to-Low gate transition

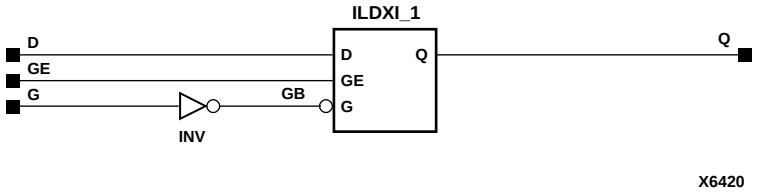


Figure 6-41 ILDXI Implementation XC4000E, XC4000X, Spartan, SpartanXL

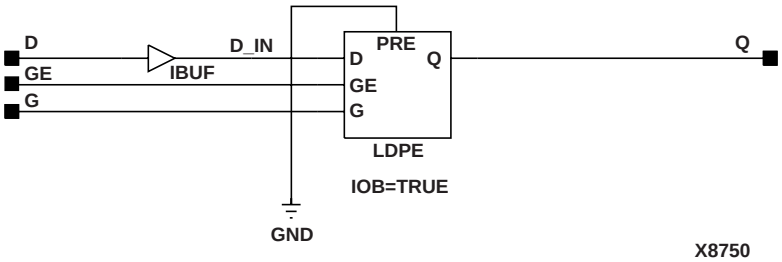
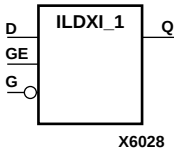


Figure 6-42 ILDXI Implementation Spartan2, Virtex

ILDXI_1

Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Macro	Macro



ILDXI_1 is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, refer to the “ILDXI” section.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	X	No Chg
1	0	1	1
1	0	0	0
1	↑	D	d

d = state of referenced input one setup time prior to Low-to-High gate transition

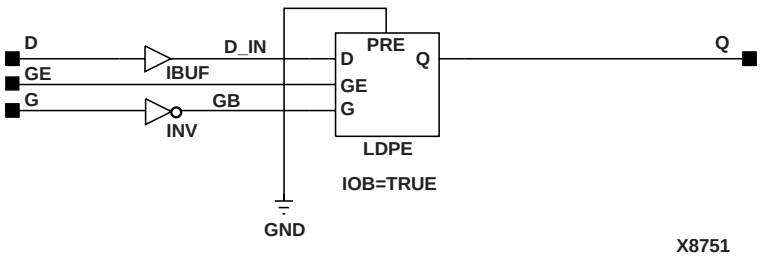
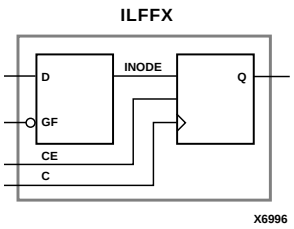


Figure 6-43 ILDXI_1 Implementation Spartan2, Virtex

ILFFX

Fast Capture Input Latch

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



ILFFX, an optional latch that drives the input flip-flop, allows the very fast capture of input data. Located on the input side of an IOB, the latch is latched by the output clock — the clock used for the output flip-flop — rather than the input clock. Thus, two different clocks can be used to clock the two input storage elements. The following figure shows an example IOB block diagram of the XC4000X IOB. After the data is captured, it is then synchronized to the internal clock (C) by the IOB flip-flop.

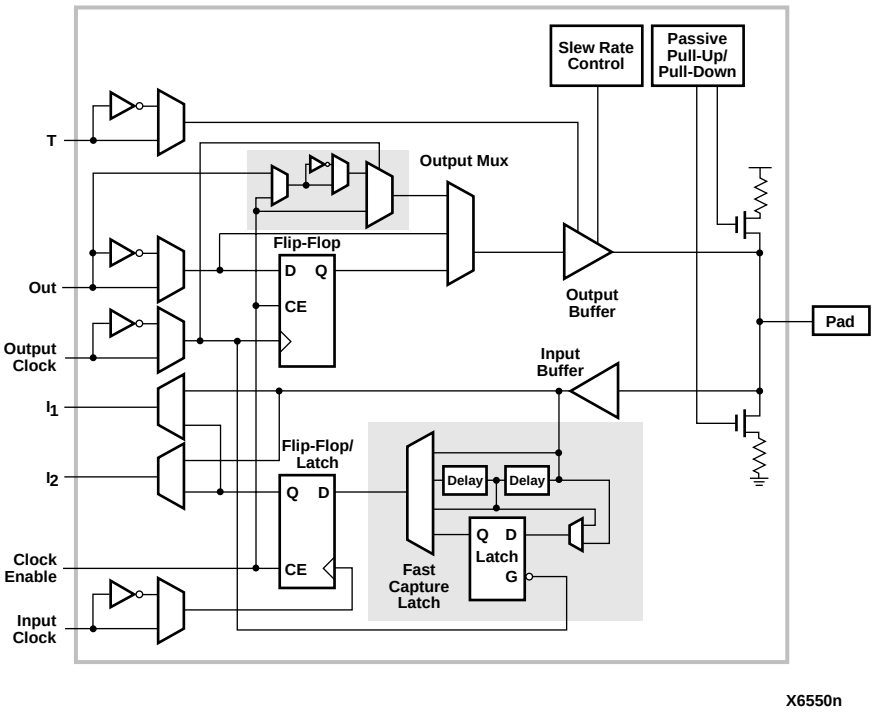


Figure 6-44 Block Diagram of XC4000X IOB

The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (GF) is Low, the data at the input (D) appears at INODE and is stored during the Low-to-High GF transition. The captured INODE data appears at output (Q) during a Low-to-High clock (C) transition.

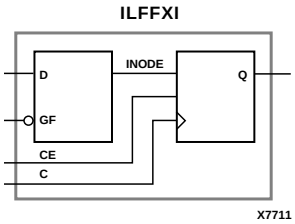
The fast latch is asynchronously cleared when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	**D**	**GF**	**C**	**Q**
0	X	X	X	No Chg
1	X	1	↑	INODE
1	0	0	↑	0
1	1	0	↑	1

ILFFXI

Fast Capture Input Latch (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



ILFFXI, an optional latch that drives the input flip-flop, allows the very fast capture of input data. Located on the input side of an IOB, the latch is latched by the output clock — the clock used for the output flip-flop — rather than the input clock. Thus, two different clocks can be used to clock the two input storage elements. See “Block Diagram of XC4000X IOB” figure in the ILFFX section. After the data is captured, it is then synchronized to the internal clock by the IOB flip-flop.

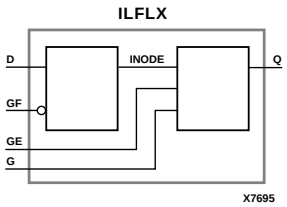
The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (GF) is Low, the data at the input (D) appears at INODE and is stored during the Low-to-High GF transition. The captured INODE data appears at output (Q) during a Low-to-High clock (C) transition.

This component is identical to ILFFX except that on active GSR it is preset instead of cleared. The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

ILFLX

Fast Capture Transparent Input Latch

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	Macro	N / A	N / A	N / A	Macro	N / A	N / A



ILFLX, an optional latch that drives the input latch, allows the very fast capture of input data. Located on the input side of an IOB, the latch is latched by the output clock — the clock used for the output flip-flop — rather than the input clock. Thus, two different clocks can be used to clock the two input storage elements. See the "Block Diagram of XC4000X IOB" figure in the ILFFX section. After the data is captured, it is then synchronized to the internal clock by the IOB latch.

The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (GF) is Low, the data at the input (D) appears at INODE and is stored during the Low-to-High GF transition. The captured INODE data appears at output (Q) when gate (G) is high.

Inputs				Outputs
GE	D	GF	G	Q
0	X	X	X	No Chg
1	X	X	0	No Chg
1	X	1	1	INODE
1	0	0	1	0
1	1	0	1	1

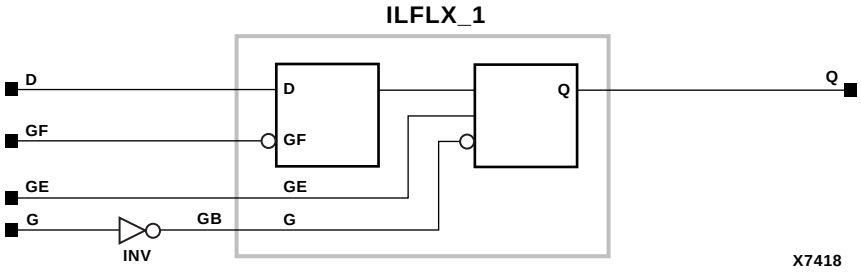
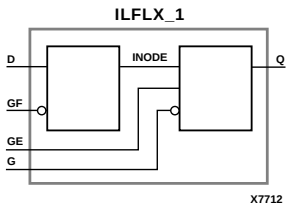


Figure 6-45 ILFLX Implementation XC4000X, SpartanXL

ILFLX_1

Fast Capture Input Latch with Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



ILFLX_1, an optional latch that drives the input latch, allows the very fast capture of input data. Located on the input side of an IOB, the latch is latched by the output clock — the clock used for the output flip-flop — rather than the input clock. Thus, two different clocks can be used to clock the two input storage elements. See the "Block Diagram of XC4000X IOB" figure in the ILFFX section. After the data is captured, it is then synchronized to the internal clock by the IOB latch.

The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (GF) is Low, the data at the input (D) appears at INODE and is stored during the Low-to-High GF transition. The captured INODE data appears on the output (Q) when the gate (G) is Low.

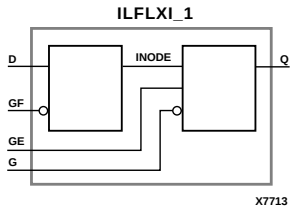
The fast latch is asynchronously cleared when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
GE	D	GF	G	Q
0	X	X	X	No Chg
1	X	X	1	No Chg
1	X	1	0	INODE
1	0	0	0	0
1	1	0	0	1

ILFLXI_1

Fast Capture Input Latch with Inverted Gate (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



ILFLXI_1, an optional latch that drives the input latch, allows the very fast capture of input data. Located on the input side of an IOB, the latch is latched by the output clock — the clock used for the output flip-flop — rather than the input clock. Thus, two different clocks can be used to clock the two input storage elements. See the "Block Diagram of XC4000X IOB" figure in the ILFFX section. After the data is captured, it is then synchronized to the internal clock by the IOB latch.

The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (GF) is Low, the data at the input (D) appears at INODE and is stored during the Low-to-High GF transition. The captured INODE data appears on the output (Q) when the gate (G) is Low.

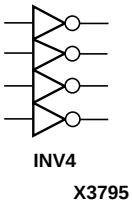
The fast latch is asynchronously preset when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

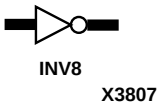
Inputs				Outputs
GE	D	GF	G	Q
0	X	X	X	No Chg
1	X	X	1	No Chg
1	X	1	0	INODE
1	0	0	0	0
1	1	0	0	1

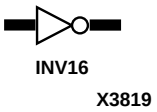
INV, 4, 8, 16

Single and Multiple Inverters

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
INV	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
INV4, INV8, INV16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







INV, INV4, INV8, and INV16 are single and multiple inverters that identify signal inversions in a schematic.

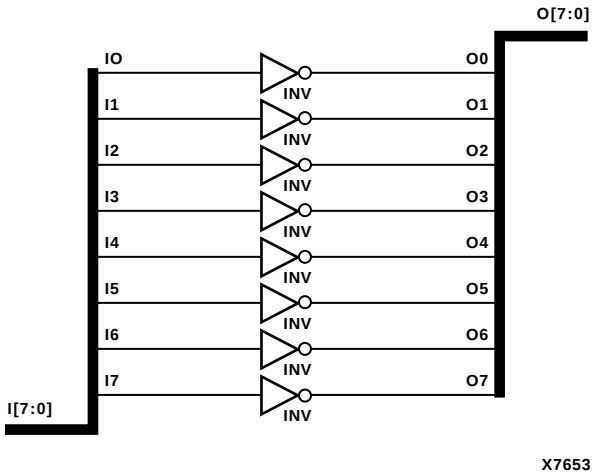
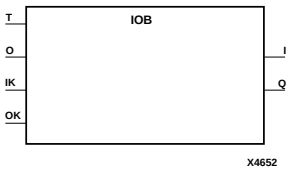


Figure 6-46 INV8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

IOB

IOB Configuration Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A



The IOB symbol is used to manually specify an IOB configuration. Use it in place of, not in conjunction with, other I/O primitives. The configuration of the IOB is specified using the BASE and CONFIG commands. Enter these commands on the schematic; the translator puts them into the CFG records in the LCA Xilinx netlist file. It is not necessary for the translator program to parse the commands specifying the IOB configuration. The mapping program from the LCA Xilinx netlist to the FPGA design checks these commands for errors.

Refer to the appropriate CAE tool interface user guide for more information on specifying the IOB configuration commands in a schematic.

The XC3000 blank IOB primitive symbol and its corresponding configured IOB primitive and circuit are shown in the "XC3000 IOB Primitive Example and Equivalent Circuit" figure.

The configuration commands must be consistent with the connections to the pins on the symbol. For example, if the configuration commands specify the IOB as a 3-state buffer, the T and O pins must be connected to signals.

You can specify the location of the IOB on the device. When specifying the LOC attribute, a valid IOB location name must be used. Refer to the "LOC" section of the "Attributes, Constraints, and Carry Logic" chapter for more information on the LOC attribute.

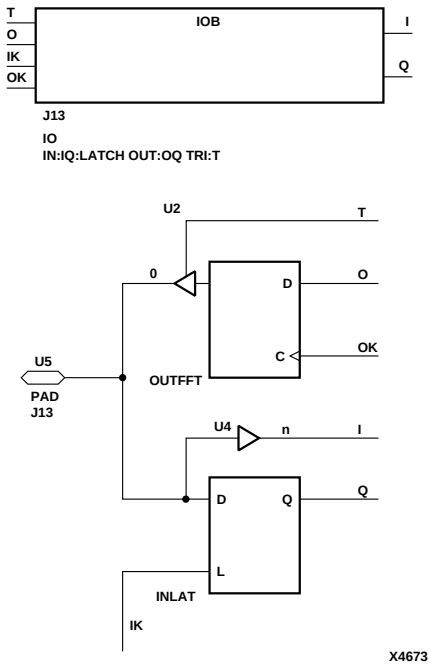
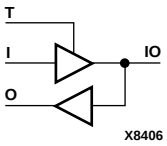


Figure 6-47 XC3000 IOB Primitive Example and Equivalent Circuit

IOBUF_*selectIO*

Bi-Directional Buffer with Selectable I/0 Interface

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



IOBUF and its variants (listed below) are bi-directional buffers whose I/O interface corresponds to a specific I/O standard.The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. The S, F, and 2, 4, 6, 8, 12, 16, 24 extensions specify the slew rate (SLOW or FAST) and the drive power (2, 4, 6, 8, 12, 16, 24 mA) for the LVTTL standard variants. For example, IOBUF_F_2 is a bi-directional buffer that uses the LVTTL I/O-signaling standard with a FAST slew and 2mA of drive power.

IOBUF (LVTTL) has selectable drive and slew rates using the DRIVE and FAST or SLOW constraints. The defaults are DRIVE=12 mA and SLOW slew.

IOBUFs are composites of IBUF and OBUFT elements. The O output is X (unknown) when IO (input/output) is Z. IOBUFs can be implemented as interconnections of their component elements.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffer components. Refer to the "SelectI/O Usage Rules" section under the IBUF_*selectIO* section for information on using these components.

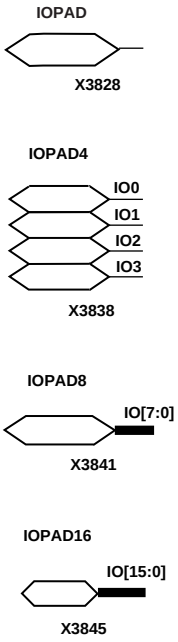
Component	I/O Standard	VCCO	VREF
IOBUF	LVTTL	3.3	N/A
IOBUF_S_2	LVTTL	3.3	N/A
IOBUF_S_4	LVTTL	3.3	N/A
IOBUF_S_6	LVTTL	3.3	N/A
IOBUF_S_8	LVTTL	3.3	N/A
IOBUF_S_12	LVTTL	3.3	N/A
IOBUF_S_16	LVTTL	3.3	N/A
IOBUF_S_24	LVTTL	3.3	N/A
IOBUF_F_2	LVTTL	3.3	N/A
IOBUF_F_4	LVTTL	3.3	N/A
IOBUF_F_6	LVTTL	3.3	N/A
IOBUF_F_8	LVTTL	3.3	N/A
IOBUF_F_12	LVTTL	3.3	N/A
IOBUF_F_16	LVTTL	3.3	N/A
IOBUF_F_24	LVTTL	3.3	N/A
IOBUF_LVCMOS2	LVCMOS2	2.5	N/A
IOBUF_PCI33_3	PCI33_3	3.3	N/A
IOBUF_PCI33_5	PCI33_5	3.3	N/A

Component	I/O Standard	VCCO	VREF
IOBUF_PCI66_3	PCI66_3	3.3	N / A
IOBUF_GTL	GTL	N / A	0.80
IOBUF_GTLP	GTL+	N / A	1.00
IOBUF_HSTL_I	HSTL_I	1.5	0.75
IOBUF_HSTL_III	HSTL_III	1.5	0.90
IOBUF_HSTL_IV	HSTL_IV	1.5	0.75
IOBUF_SSTL2_I	SSTL2_I	2.5	1.10
IOBUF_SSTL2_II	SSTL2_II	2.5	1.10
IOBUF_SSTL3_I	SSTL3_I	3.3	0.90
IOBUF_SSTL3_II	SSTL3_II	3.3	1.50
IOBUF_CTT	CTT	3.3	1.50
IOBUF_AGP	AGP	3.3	1.32

IOPAD, 4, 8, 16

Single- and Multiple-Input/Output Pads

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
IOPAD	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IOPAD4, IOPAD8, IOPAD16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



IOPAD, IOPAD4, IOPAD8, and IOPAD16 are single and multiple input/output pads. The IOPAD is a connection point from a device pin, used as a bidirectional signal, to a PLD device. The IOPAD is connected internally to an input/output block (IOB), which is configured by the software as a bidirectional block. Bidirectional blocks can consist of any combinations of a 3-state output buffer (such as OBUFT or OFDE) and any available input buffer (such as IBUF or IFD). Refer to the appropriate CAE tool interface user guide for details on assigning pin location and identification.

Note: The LOC attribute cannot be used on IOPAD multiples.

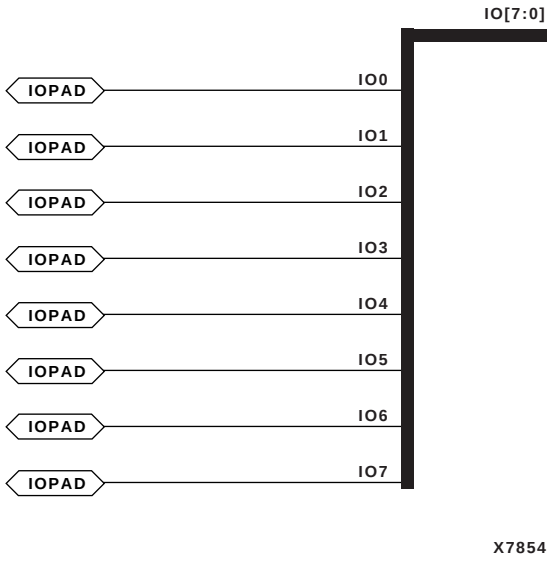
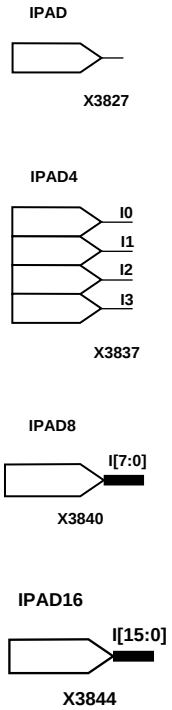


Figure 6-48 IOPAD8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

IPAD, 4, 8, 16

Single- and Multiple-Input Pads

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
IPAD	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IPAD4, IPAD8, IPAD16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



IPAD, IPAD4, IPAD8, and IPAD16 are single and multiple input pads. The IPAD is a connection point from a device pin used for an input signal to the PLD device. It is connected internally to an input/output block (IOB), which is configured by the software as an IBUF, IFD, or ILD. Refer to the appropriate CAE tool interface user guide for details on assigning pin location and identification.

For Virtex and Spartan2, IPADs must be used to drive IBUF and IBUFG inputs. An IPAD can be inferred by NGDBUILD if one is missing on an IBUF or IBUFG input.

Note: The LOC attribute cannot be used on IPAD multiples.

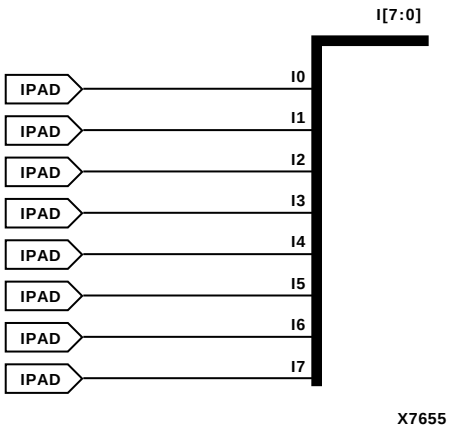
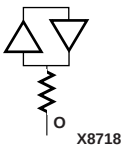


Figure 6-49 IPAD8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

KEEPER

KEEPER Symbol

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



KEEPER is a weak keeper element used to retain the value of the net connected to its bidirectional O pin. For example, if a logic 1 is being driven onto the net, KEEPER drives a weak/resistive 1 onto the net. If the net driver is then tri-stated, KEEPER continues to drive a weak/resistive 1 onto the net.

For additional information on using a KEEPER element with SelectI/O components, refer to the “SelectI/O Usage Rules” in the "IBUF_*selectIO*" section.

Chapter 7

Design Elements (LD to NOR16)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

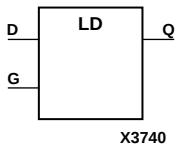
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: "XC3000 Library," "XC4000E Library," "XC4000X Library," "XC5200 Library," "XC9000 Library," "Spartan Library," "SpartanXL Library," "Spartan2 Library," and "Virtex Library."

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

LD

Transparent Data Latch

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Macro	Macro	Macro	N/A	Macro	Primitive	Primitive



LD is a transparent data latch. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Refer to the "LD4, 8, 16" section for information on multiple transparent data latches for the XC4000X, XC9000, and SpartanXL.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

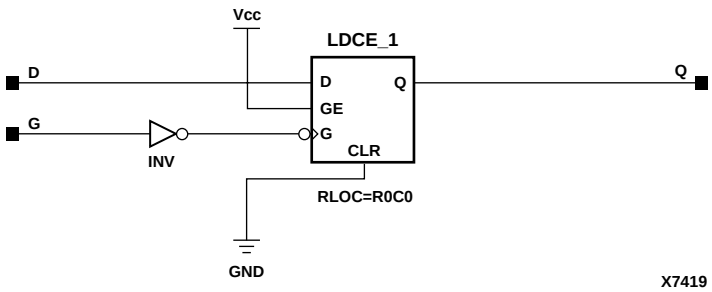


Figure 7-1 LD Implementation XC4000X, SpartanXL

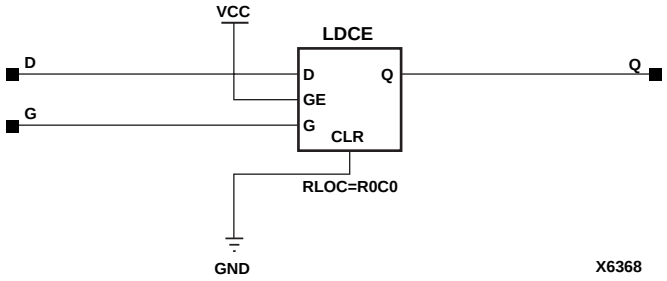


Figure 7-2 LD Implementation XC5200

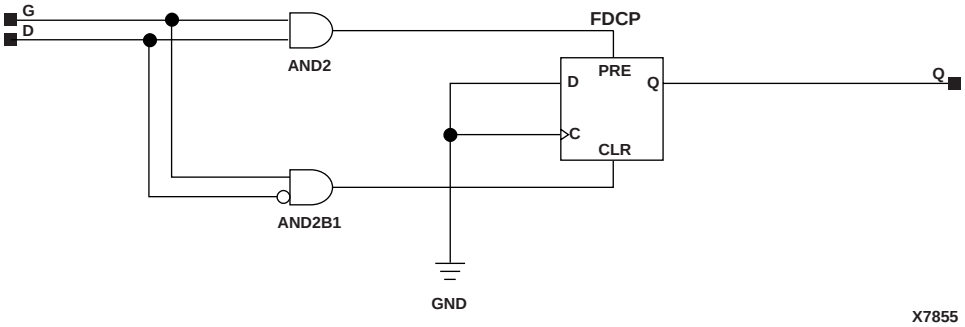
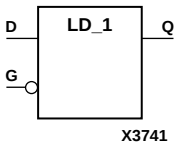


Figure 7-3 LD Implementation XC9000

LD_1

Transparent Data Latch with Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Macro	Macro	N/A	N/A	Macro	Primitive	Primitive



LD_1 is a transparent data latch with an inverted gate. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

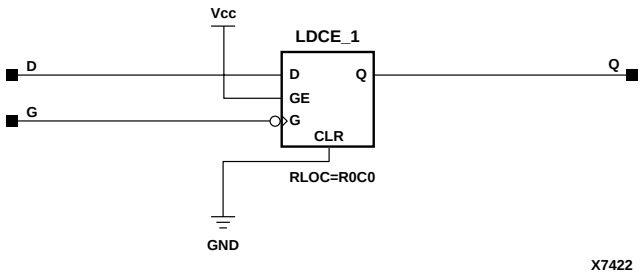


Figure 7-4 LD_1 Implementation XC4000X, SpartanXL

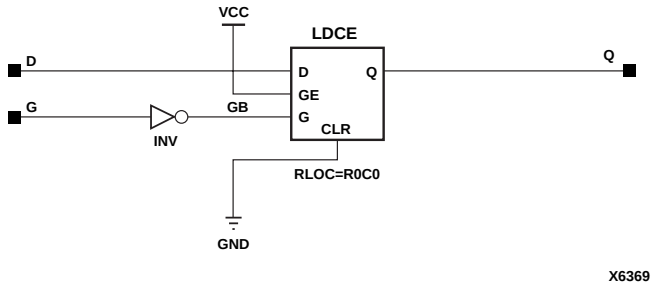
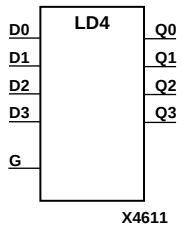


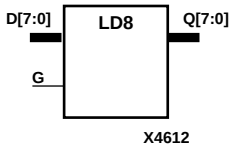
Figure 7-5 LD_1 Implementation XC5200

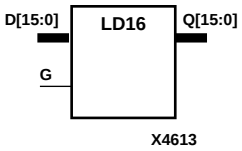
LD4, 8, 16

Multiple Transparent Data Latches

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	Macro	N / A	Macro	N / A	Macro	Macro	Macro







LD4, LD8, and LD16 have, respectively, 4, 8, and 16 transparent data latches with a common gate enable (G). The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Refer to the “LD” section for information on single transparent data latches.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

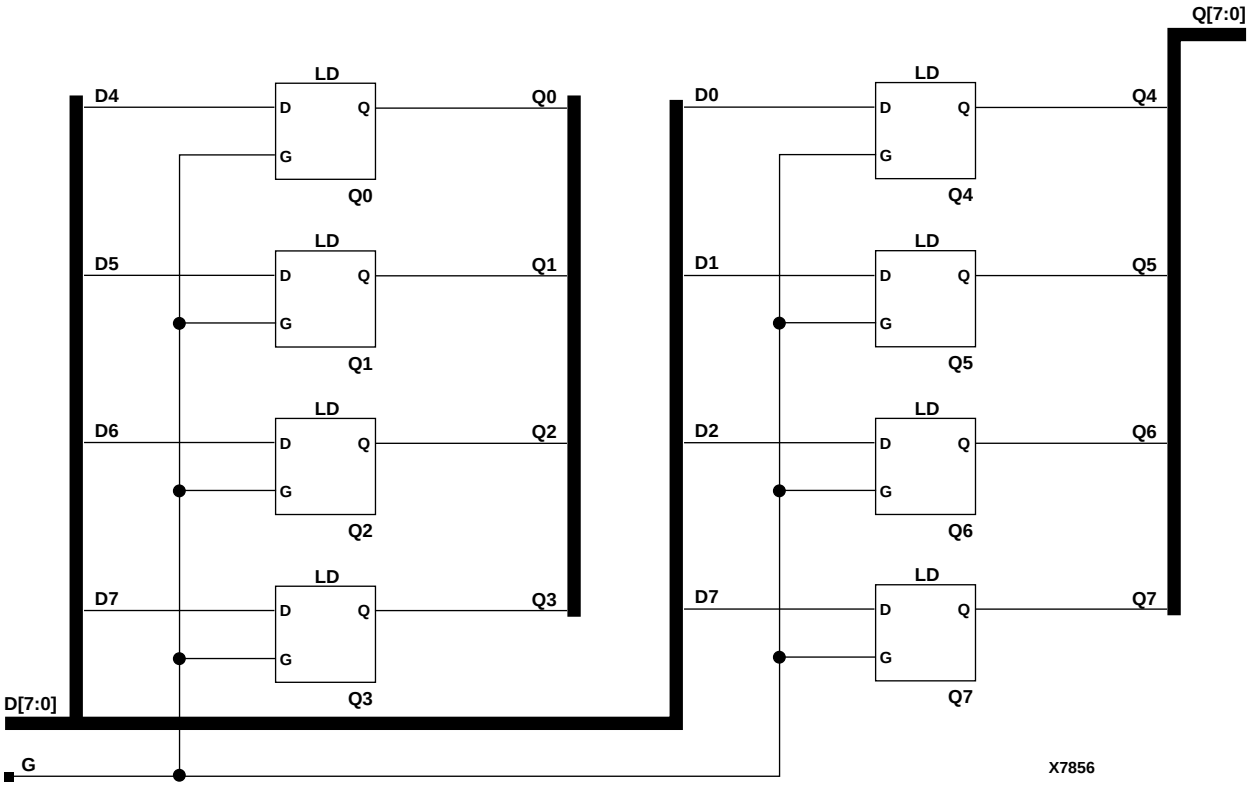
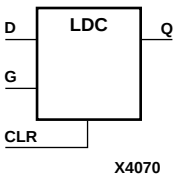


Figure 7-6 LD8 Implementation XC4000X, XC9000, SpartanXL, Spartan2, Virtex

LDC

Transparent Data Latch with Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Macro	Macro	N/A	N/A	Macro	Primitive	Primitive



LDC is a transparent data latch with asynchronous clear. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate enable (G) input is High and CLR is Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains low.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CLR	G	D	Q
1	X	X	0
0	1	0	0
0	1	1	1
0	0	X	No Chg
0	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

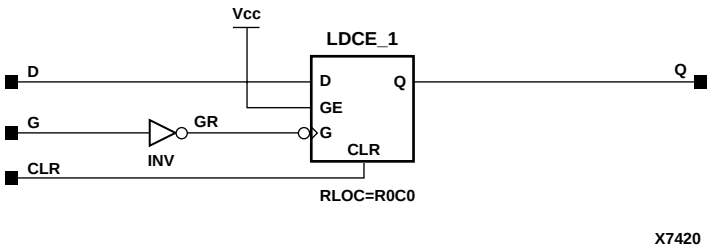


Figure 7-7 LDC Implementation XC4000X, SpartanXL

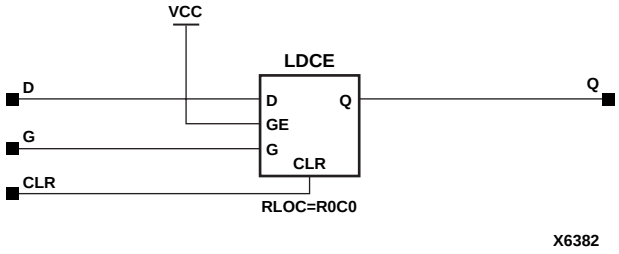
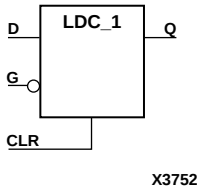


Figure 7-8 LDC Implementation XC5200

LDC_1

Transparent Data Latch with Asynchronous Clear and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Macro	Macro	N/A	N/A	Macro	Primitive	Primitive



LDC_1 is a transparent data latch with asynchronous clear and inverted gate. When the asynchronous clear input (CLR) is High, it overrides the other inputs (D and G) and resets the data (Q) output Low. Q reflects the data (D) input while the gate enable (G) input and CLR are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CLR	G	D	Q
1	X	X	0
0	0	0	0
0	0	1	1
0	1	X	No Chg
0	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

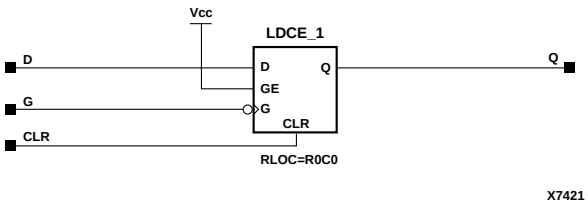


Figure 7-9 LDC_1 Implementation XC4000X, SpartanXL

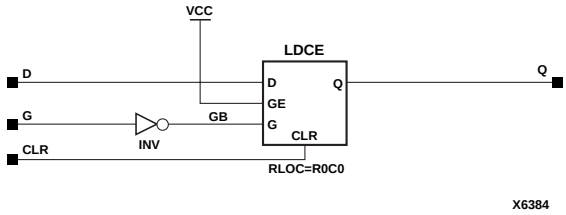
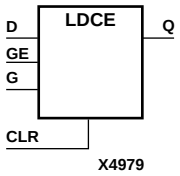


Figure 7-10 LDC_1 Implementation XC5200

LDCE

Transparent Data Latch with Asynchronous Clear and Gate Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Macro	Primitive	N/A	N/A	Macro	Primitive	Primitive



LDCE is a transparent data latch with asynchronous clear and gate enable. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High and CLR is Low. If GE is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains low.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	GE	G	D	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	0	0
0	1	1	1	1
0	1	0	X	No Chg
0	1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

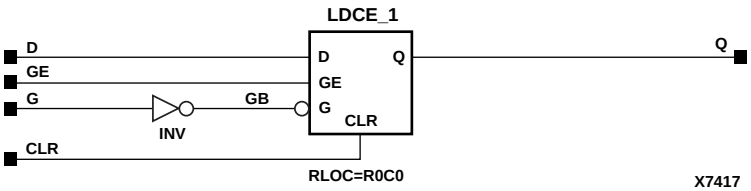
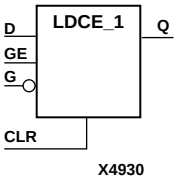


Figure 7-11 LDCE Implementation XC4000X, SpartanXL

LDCE_1

Transparent Data Latch with Asynchronous Clear, Gate Enable, and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	Macro	N/A	N/A	Primitive	Primitive	Primitive



LDCE_1 is a transparent data latch with asynchronous clear, gate enable, and inverted gate. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate (G) input and CLR are Low and gate enable (GE) is High. If GE is Low, the data on D cannot be latched. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High or GE remains Low.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	GE	G	D	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	0	0
0	1	0	1	1
0	1	1	X	No Chg
0	1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

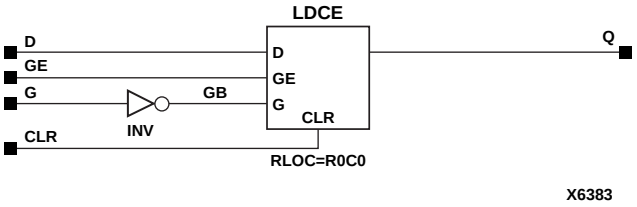
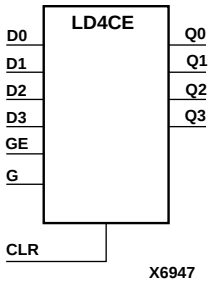


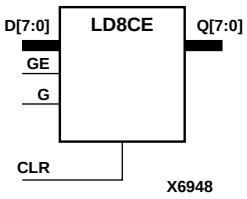
Figure 7-12 LDCE_1 Implementation XC5200

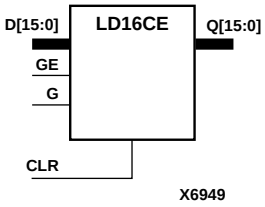
LD4CE, LD8CE, LD16CE

Transparent Data Latches with Asynchronous Clear and Gate Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro







LD4CE, LD8CE, and LD16CE have, respectively, 4, 8, and 16 transparent data latches with asynchronous clear and gate enable. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) outputs Low. Q reflects the data (D) inputs while the gate (G) input is High, gate enable (GE) is High, and CLR is Low. If GE for is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as GE remains Low.

The latch is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR (XC5200) and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
CLR	GE	G	Dn	Qn
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	1	1
0	1	1	0	0
0	1	0	X	No Chg
0	1	↓	Dn	dn

Dn = referenced input, for example, D0, D1, D2
Qn = referenced output, for example, Q0, Q1, Q2
dn = referenced input state, one setup time prior to High-to-Low gate transition

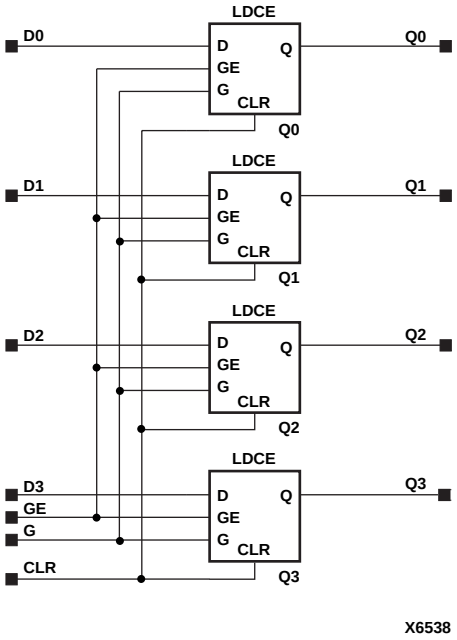


Figure 7-13 LD4CE Implementation XC4000X, XC5200, SpartanXL, Spartan2, Virtex

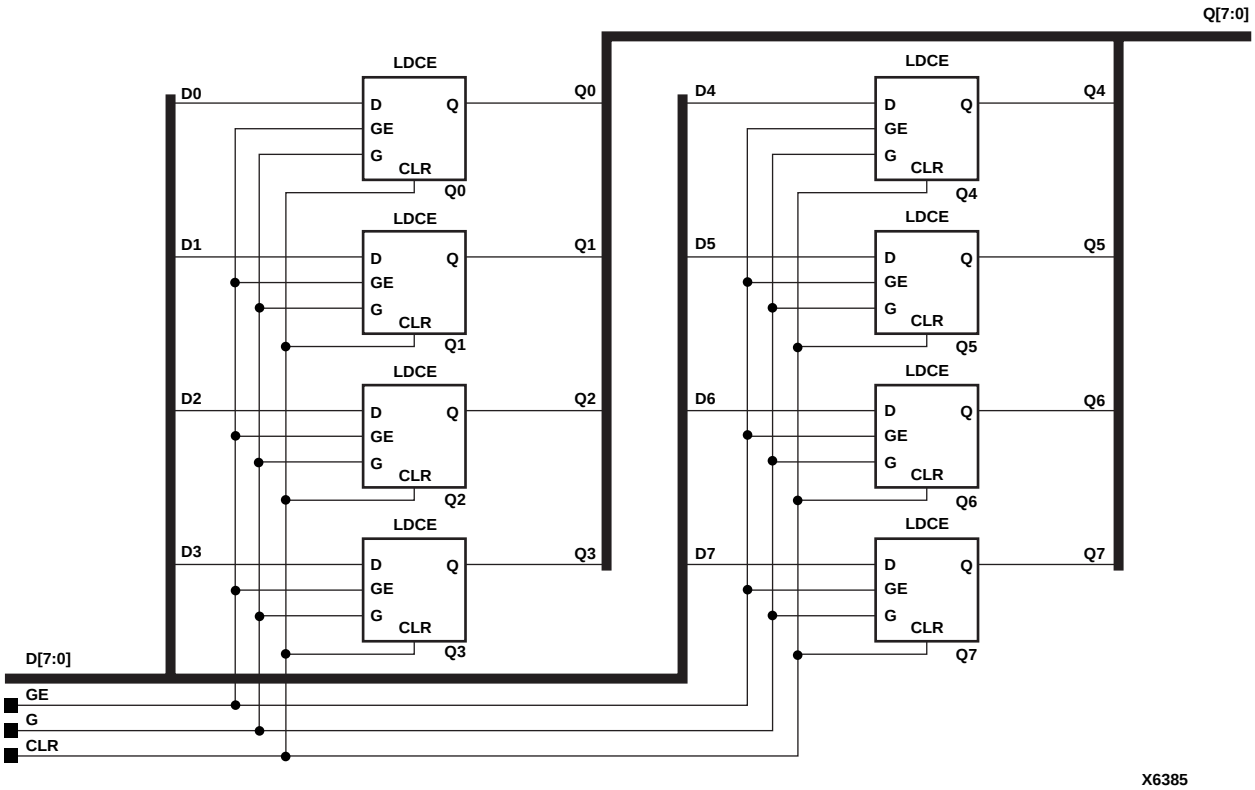
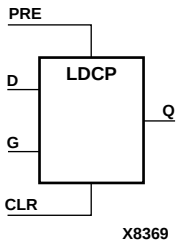


Figure 7-14 LD8CE Implementation XC4000X, XC5200, SpartanXL, Spartan2, Virtex

LDCP

Transparent Data Latch with Asynchronous Clear and Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDCP is a transparent data latch with data (D), asynchronous clear (CLR) and preset (PRE) inputs. When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is low, it presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input is High and CLR and PRE are Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

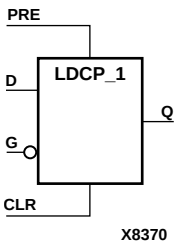
Inputs				Outputs
CLR	PRE	G	D	Q
1	X	X	X	0
0	1	X	X	1
0	0	1	1	1
0	0	1	0	0
0	0	0	X	No Chg
0	0	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

LDCP_1

Transparent Data Latch with Asynchronous Clear and Preset and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDCP_1 is a transparent data latch with data (D), asynchronous clear (CLR) and preset (PRE) inputs. When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while gate (G) input, CLR, and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

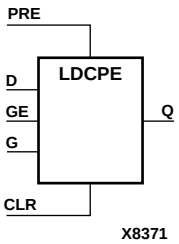
Inputs				Outputs
CLR	PRE	G	D	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	1	1
0	0	0	0	0
0	0	1	X	No Chg
0	0	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

LDCPE

Transparent Data Latch with Asynchronous Clear and Preset and Gate Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDCPE is a transparent data latch with data (D), asynchronous clear (CLR), asynchronous preset (PRE), and gate enable (GE). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High and CLR and PRE are Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remain Low.

The latch is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

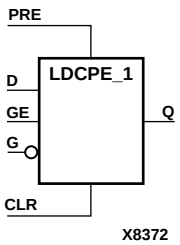
Inputs					Outputs
CLR	PRE	GE	G	D	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	1	0	0
0	0	1	1	1	1
0	0	1	0	X	No Chg
0	0	1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

LDCPE_1

Transparent Data Latch with Asynchronous Clear and Preset, Gate Enable, and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDCPE_1 is a transparent data latch with data (D), asynchronous clear (CLR), asynchronous preset (PRE), and gate enable (GE). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while gate enable (GE) is High and gate (G), CLR, and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G is High or GE is Low.

The latch is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

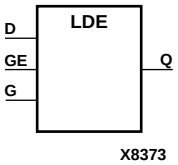
Inputs					Outputs
CLR	PRE	GE	G	D	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	X	No Chg
0	0	1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

LDE

Transparent Data Latch with Gate Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDE is a transparent data latch with data (D) and gate enable (GE) inputs. Output Q reflects the data (D) while the gate (G) input and gate enable (GE) are High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remain Low.

The latch is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

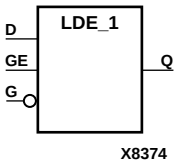
Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	0	0
1	1	1	1
1	0	X	No Chg
1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

LDE_1

Transparent Data Latch with Gate Enable and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDE_1 is a transparent data latch with data (D) and gate enable (GE) inputs. Output Q reflects the data (D) while the gate (G) input is Low and gate enable (GE) is High. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G is High or GE is Low.

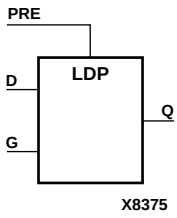
The latch is asynchronously cleared, output Low, when power is applied. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	0	0
1	0	1	1
1	1	X	No Chg
1	↑	D	d
d = state of input one setup time prior to Low-to-High gate transition			

LDP

Transparent Data Latch with Asynchronous Preset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDP is a transparent data latch with asynchronous preset (PRE). When the PRE input is High, it overrides the other inputs and resets the data (Q) output High. Q reflects the data (D) input while gate (G) input is High and PRE is Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. For Virtex and Spartan2, GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

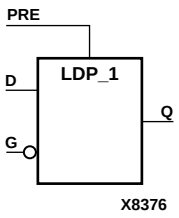
Inputs			Outputs
PRE	G	D	Q
1	X	X	1
0	1	0	0
0	1	1	1
0	0	X	No Chg
0	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

LDP_1

Transparent Data Latch with Asynchronous Preset and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



LDP_1 is a transparent data latch with asynchronous preset (PRE). When the PRE input is High, it overrides the other inputs and resets the data (Q) output High. Q reflects the data (D) input while gate (G) input and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. For Virtex and Spartan2, GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

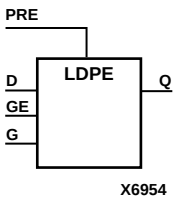
Inputs			Outputs
PRE	G	D	Q
1	X	X	1
0	0	0	0
0	0	1	1
0	1	X	No Chg
0	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

LDPE

Transparent Data Latch with Asynchronous Preset and Gate Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Macro	N/A	N/A	N/A	Macro	Primitive	Primitive



LDPE is a transparent data latch with asynchronous preset and gate enable. When the asynchronous preset (PRE) is High, it overrides the other input and presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High. If GE is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs
PRE	GE	G	D	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	1	0	0
0	1	1	1	1
0	1	0	X	No Chg
0	1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

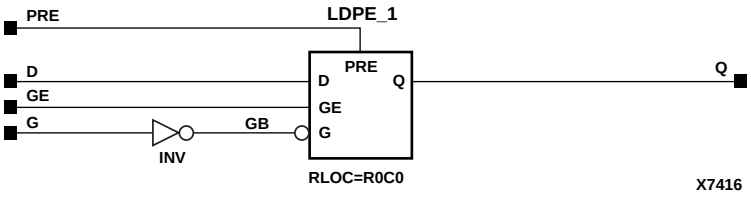
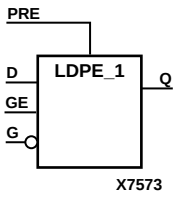


Figure 7-15 LDPE Implementation XC4000X, SpartanXL

LDPE_1

Transparent Data Latch with Asynchronous Preset, Gate Enable, and Inverted Gate

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	Primitive	Primitive



LDPE_1 is a transparent data latch with asynchronous preset, gate enable, and inverted gated. When the asynchronous preset (PRE) is High, it overrides the other input and presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input is Low and gate enable (GE) is High.

If GE is low, data on D cannot be latched. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High or GE remains Low.

The latch is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

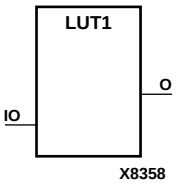
Inputs				Outputs
PRE	GE	G	D	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	0	0
0	1	0	1	1
0	1	1	X	No Chg
0	1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

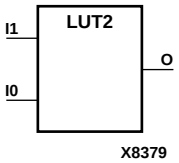
LUT1, 2, 3, 4

1-, 2-, 3-, 4-Bit Look-Up-Table with General Output

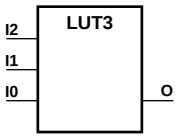
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



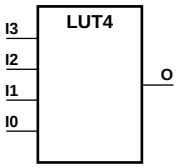
X8358



X8379



X8382



X8385

LUT1, LUT2, LUT3, and LUT4 are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with general output (O).

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1 provides a look-up-table version of a buffer or inverter.

LUTs are the basic Virtex and Spartan2 building blocks. Two LUTs are available in each CLB slice; four LUTs are available in each CLB. The variants, “LUT1_D, LUT2_D, LUT3_D, LUT4_D” and “LUT1_L, LUT2_L, LUT3_L, LUT4_L,” provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Table 7-1 LUT3 Function Table

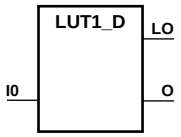
Inputs			Outputs
I2	I1	I0	O
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

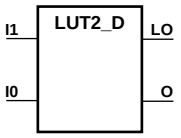
LUT1_D, LUT2_D, LUT3_D, LUT4_D

1-, 2-, 3-, 4-Bit Look-Up-Table with Dual Output

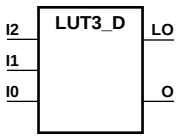
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



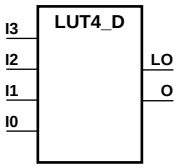
X8377



X8380



X8383



X8386

LUT1_D, LUT2_D, LUT3_D, and LUT4_D are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with two functionally identical outputs, O and LO. The O output is a general interconnect. The LO output is used to connect to another output within the same CLB slice and to the fast connect buffer.

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1_D provides a look-up-table version of a buffer or inverter.

See also “LUT1, 2, 3, 4” and “LUT1_L, LUT2_L, LUT3_L, LUT4_L.”

Table 7-2 LUT3_D Function Table

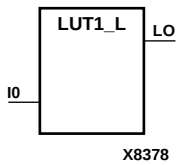
Inputs			Outputs	
I2	I1	I0	O	LO
0	0	0	INIT[0]	INIT[0]
0	0	1	INIT[1]	INIT[1]
0	1	0	INIT[2]	INIT[2]
0	1	1	INIT[3]	INIT[3]
1	0	0	INIT[4]	INIT[4]
1	0	1	INIT[5]	INIT[5]
1	1	0	INIT[6]	INIT[6]
1	1	1	INIT[7]	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

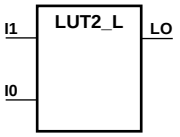
LUT1_L, LUT2_L, LUT3_L, LUT4_L

1-, 2-, 3-, 4-Bit Look-Up-Table with Local Output

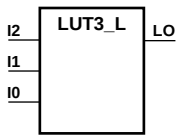
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



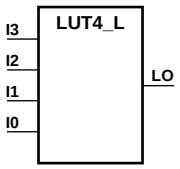
X8378



X8381



X8384



X8387

LUT1_L, LUT2_L, LUT3_L, and LUT4_L are, respectively, 1-, 2-, 3-, and 4- bit look-up-tables (LUTs) with a local output (LO) that is used to connect to another output within the same CLB slice and to the fast connect buffer.

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1_L provides a look-up-table version of a buffer or inverter.

See also “LUT1, 2, 3, 4” and “LUT1_D, LUT2_D, LUT3_D, LUT4_D.”

Table 7-3 LUT3_L Function Table

Inputs			Outputs
I2	I1	I0	LO
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

MD0

Mode 0, Input Pad Used for Readback Trigger Input

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A

The MD0 input pad is connected to the Mode 0 (M0) input pin, which is used to determine the configuration mode on XC4000 and XC5200 devices. Following configuration, MD0 can be used as an input pad, but it must be connected through an IBUF to the user circuit. However, the user input signal must not interfere with the device configuration. XC5200 devices allow an MD0 pad to be used as an output pad; XC4000 devices do not. The IOB associated with the MD0 pad has no flip-flop or latch. This pad is usually connected (automatically) to the RTRIG input of the READBACK function.

MD1

Mode 1, Output Pad Used for Readback Data Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A

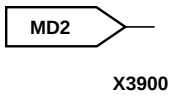
X3898

The MD1 output pad is connected to the Mode 1 (M1) output pin, which is used to determine the configuration mode on XC4000 and XC5200 devices. Following configuration, MD1 can be used as a 3-state or simple output pad, but it must be connected through an OBUF or an OBUFT to the user circuit. However, the user output signal must not interfere with the device configuration. XC5200 devices allow an MD1 pad to be used as an input pad; XC4000 devices do not. The IOB associated with an MD1 pad has no flip-flop or latch. This pad is usually connected to the DATA output of the READBACK function, and the output-enable input of the 3-state OBUFT is connected to the RIP output of the READBACK function.

MD2

Mode 2, Input Pad

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A



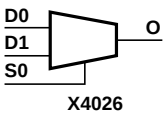
X3900

The MD2 input pad is connected to the Mode 2 (M2) input pin, which is used to determine the configuration mode on XC4000 and XC5200 devices. Following configuration, MD2 can be used as an input pad, but it must be connected through an IBUF to the user circuit. However, the user input signal must not interfere with the device configuration. XC5200 devices allow an MD2 pad to be used as an output pad; XC4000 devices do not. The IOB associated with it has no flip-flop or latch.

M2_1

2-to-1 Multiplexer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



The M2_1 multiplexer chooses one data bit from two sources (D1 or D0) under the control of the select input (S0). The output (O) reflects the state of the selected data input. When Low, S0 selects D0 and when High, S0 selects D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	1
0	X	0	0

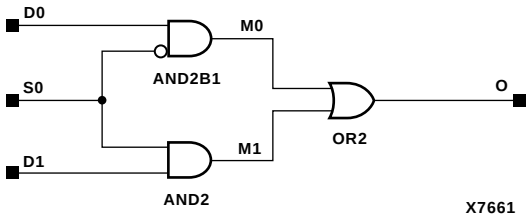
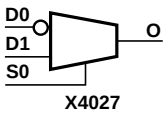


Figure 7-16 M2_1 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

M2_1B1

2-to-1 Multiplexer with D0 Inverted

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



The M2_1B1 multiplexer chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When S0 is Low, the output (O) reflects the state of $\overline{D0}$. When S0 is High, O reflects the state of D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	0
0	X	0	1

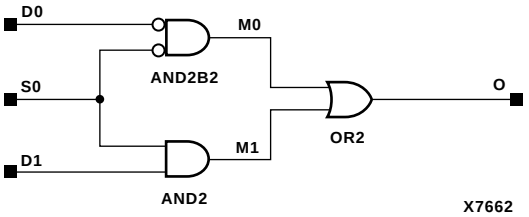
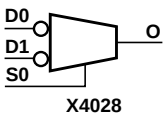


Figure 7-17 M2_1B1 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

M2_1B2

2-to-1 Multiplexer with D0 and D1 Inverted

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



The M2_1B2 multiplexer chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When S0 is Low, the output (O) reflects the state of $\overline{D0}$. When S0 is High, O reflects the state of $\overline{D1}$.

Inputs			Outputs
S0	D1	D0	O
1	1	X	0
1	0	X	1
0	X	1	0
0	X	0	1

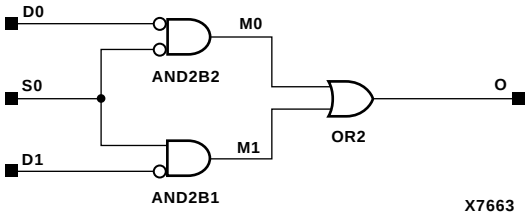
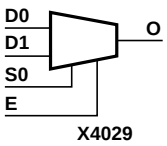


Figure 7-18 M2_1B2 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

M2_1E

2-to-1 Multiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



M2_1E is a 2-to-1 multiplexer with enable. When the enable input (E) is High, the M2_1E chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When E is High, the output (O) reflects the state of the selected input. When Low, S0 selects D0 and when High, S0 selects D1. When E is Low, the output is Low.

Inputs				Outputs
E	S0	D1	D0	O
0	X	X	X	0
1	0	X	1	1
1	0	X	0	0
1	1	1	X	1
1	1	0	X	0

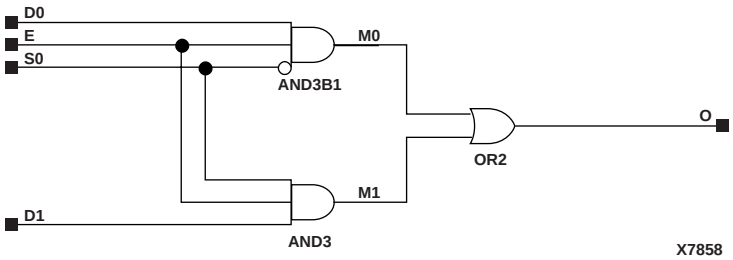
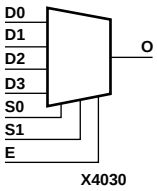


Figure 7-19 M2_1E Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

M4_1E

4-to-1 Multiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



M4_1E is an 4-to-1 multiplexer with enable. When the enable input (E) is High, the M4_1E multiplexer chooses one data bit from four sources (D3, D2, D1, or D0) under the control of the select inputs (S1 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs							Outputs
E	S1	S0	D0	D1	D2	D3	O
0	X	X	X	X	X	X	0
1	0	0	D0	X	X	X	D0
1	0	1	X	D1	X	X	D1
1	1	0	X	X	D2	X	D2
1	1	1	X	X	X	D3	D3

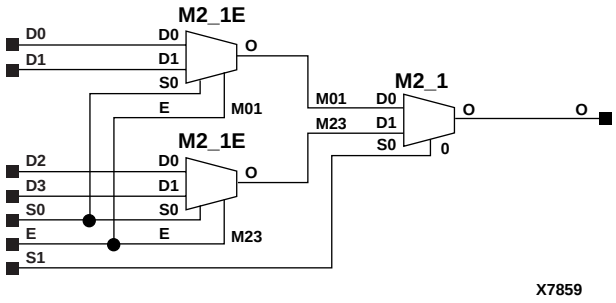


Figure 7-20 M4_1E Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

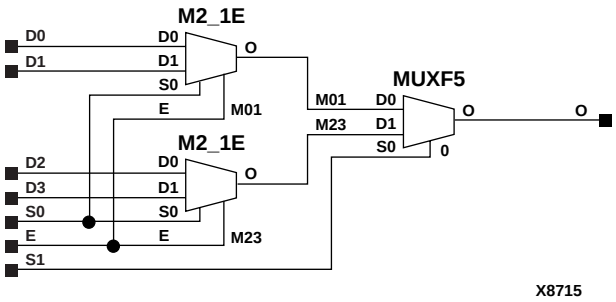
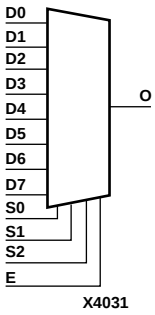


Figure 7-21 M4_1E Implementation Spartan2, Virtex

M8_1E

8-to-1 Multiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



M8_1E is an 8-to-1 multiplexer with enable. When the enable input (E) is High, the M8_1E multiplexer chooses one data bit from eight sources (D7 – D0) under the control of the select inputs (S2 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs					Outputs
E	S2	S1	S0	D7 – D0	O
0	X	X	X	X	0
1	0	0	0	D0	D0
1	0	0	1	D1	D1
1	0	1	0	D2	D2
1	0	1	1	D3	D3
1	1	0	0	D4	D4
1	1	0	1	D5	D5
1	1	1	0	D6	D6
1	1	1	1	D7	D7

Dn represents signal on the Dn input; all other data inputs are don't-cares (X).

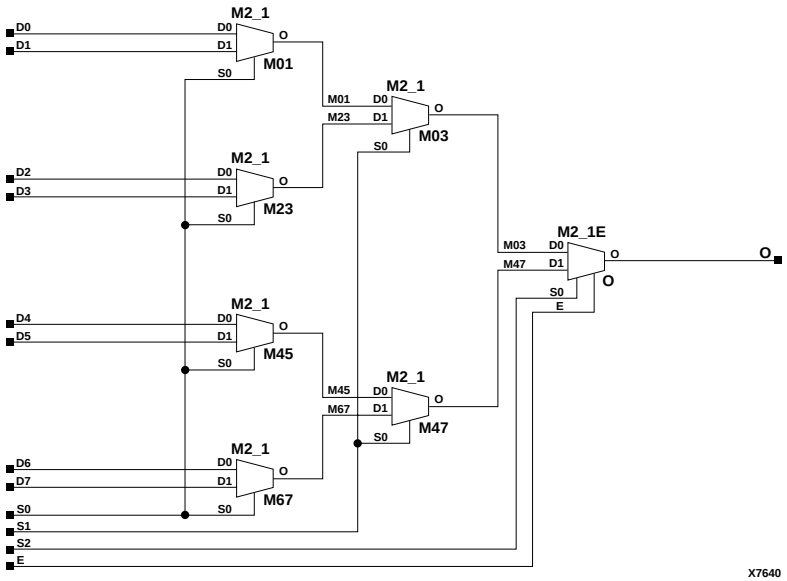


Figure 7-22 M8_1E Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

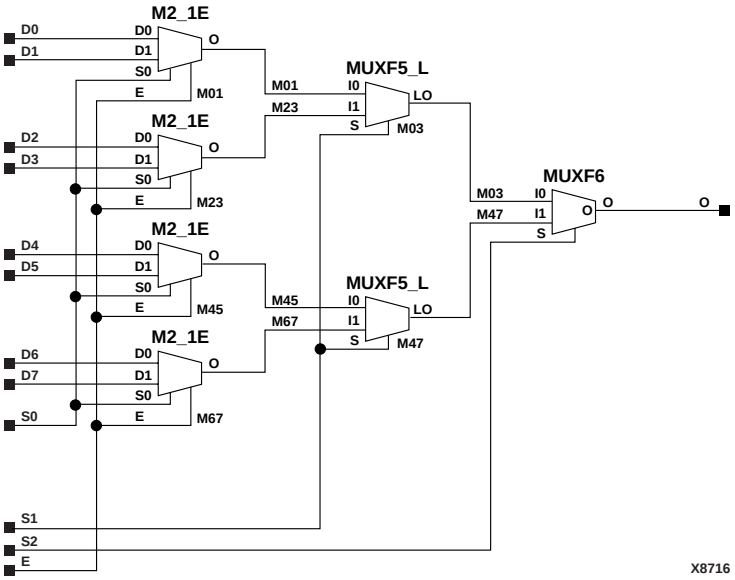
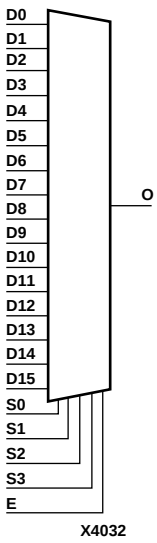


Figure 7-23 M8_1E Implementation Spartan2, Virtex

M16_1E

16-to-1 Multiplexer with Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



M16_1E is a 16-to-1 multiplexer with enable. When the enable input (E) is High, the M16_1E multiplexer chooses one data bit from 16 sources (D15 – D0) under the control of the select inputs (S3 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

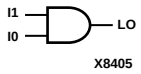
Inputs						Outputs
E	**S3**	**S2**	**S1**	**S0**	**D15 – D0**	**O**
0	X	X	X	X	X	0
1	0	0	0	0	D0	D0
1	0	0	0	1	D1	D1
1	0	0	1	0	D2	D2
1	0	0	1	1	D3	D3
. . .	. . .	. . .	. . .	. . .	. . .	. . .
1	1	1	0	0	D12	D12
1	1	1	0	1	D13	D13
1	1	1	1	0	D14	D14
1	1	1	1	1	D15	D15

Dn represents signal on the Dn input; all other data inputs are don't-cares (X).

MULT_AND

Fast Multiplier AND

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MULT_AND is an AND component used *exclusively* for building fast and smaller multipliers. The I1 and I0 inputs *must* be connected to the I1 and I0 inputs of the associated LUT. The LO output *must* be connected to the DI input of the associated MUXCY, MUXCY_D, or MUXCY_L. See the "Example Multiplier Using MULT_AND" figure.

Inputs		Output
I1	I0	LO
0	0	0
0	1	0
1	0	0
1	1	1

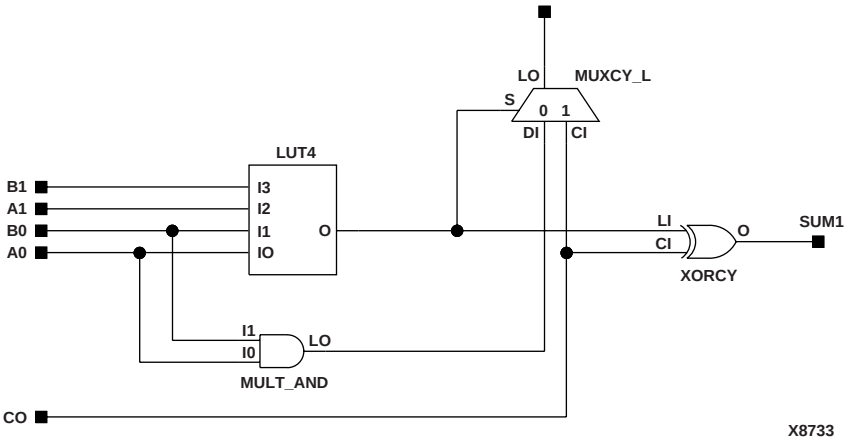
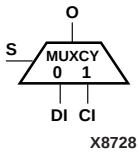


Figure 7-24 Example Multiplier Using MULT_AND

MUXCY

2-to-1 Multiplexer for Carry Logic with General Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXCY is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY. The carry in (CI) input of an LC is connected to the CI input of the MUXCY. The select input (S) of the MUX is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (O) of the MUXCY reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

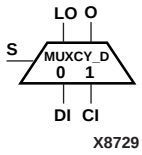
The variants, “MUXCY_D” and “MUXCY_L,” provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	DI	CI	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

MUXCY_D

2-to-1 Multiplexer for Carry Logic with Dual Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXCY_D is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY_D. The carry in (CI) input of an LC is connected to the CI input of the MUXCY_D. The select input (S) of the MUX is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (O and LO) of the MUXCY_D reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

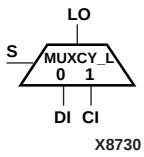
See also “MUXCY” and “MUXCY_L.”

Inputs			Outputs	
S	DI	CI	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

MUXCY_L

2-to-1 Multiplexer for Carry Logic with Local Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXCY_L is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY_L. The carry in (CI) input of an LC is connected to the CI input of the MUXCY_L. The select input (S) of the MUX is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (LO) of the MUXCY_L reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

The LO output can only connect to other inputs within the same CLB slice.

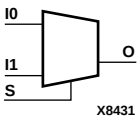
See also “MUXCY” and “MUXCY_D.”

Inputs			Outputs
S	**DI**	**CI**	**LO**
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

MUXF5

2-to-1 Lookup Table Multiplexer with General Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXF5 provides a multiplexer function in one half of a Virtex or Spartan2 CLB for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

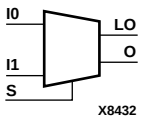
The variants, “MUXF5_D” and “MUXF5_L,” provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	**I0**	**I1**	**O**
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

MUXF5_D

2-to-1 Lookup Table Multiplexer with Dual Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXF5_D provides a multiplexer function in one half of a Virtex or Spartan2 CLB for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

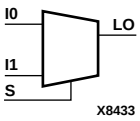
See also “MUXF5” and “MUXF5_L.”

Inputs			Outputs	
S	I0	I1	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

MUXF5_L

2-to-1 Lookup Table Multiplexer with Local Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXF5_L provides a multiplexer function in one half of a Virtex or Spartan2 CLB for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output is used to connect to other inputs within the same CLB slice.

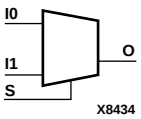
See also “MUXF5” and “MUXF5_L.”

Inputs			Output
S	I0	I1	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

MUXF6

2-to-1 Lookup Table Multiplexer with General Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXF6 provides a multiplexer function in a full Virtex or Spartan2 CLB for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

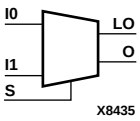
The variants, “MUXF6_D” and “MUXF6_L,” provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

MUXF6_D

2-to-1 Lookup Table Multiplexer with Dual Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXF6_D provides a multiplexer function in a full Virtex or Spartan2 CLB for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

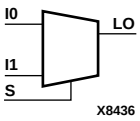
See also “MUXF6” and “MUXF6_L.”

Inputs			Outputs	
S	**I0**	**I1**	**O**	**LO**
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

MUXF6_L

2-to-1 Lookup Table Multiplexer with Local Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



MUXF6_L provides a multiplexer function in a full Virtex or Spartan2 CLB for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output is used to connect to other inputs within the same CLB slice.

See also “MUXF6” and “MUXF6_D.”

Inputs			Output
S	I0	I1	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

NAND2-9

2- to 9-Input NAND Gates with Inverted and Non-Inverted Inputs

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
NAND2, NAND2B1, NAND2B2, NAND3, NAND3B1, NAND3B2, NAND3B3, NAND4, NAND4B1, NAND4B2, NAND4B3, NAND4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5, NAND5B1, NAND5B2, NAND5B3, NAND5B4, NAND5B5	Primitive	Primitive	Primitive	Macro	Primitive	Primitive	Primitive	Primitive	Primitive
NAND6, NAND7, NAND8, NAND9	Macro	Macro	Macro	Macro	Primitive	Macro	Macro	Macro	Macro

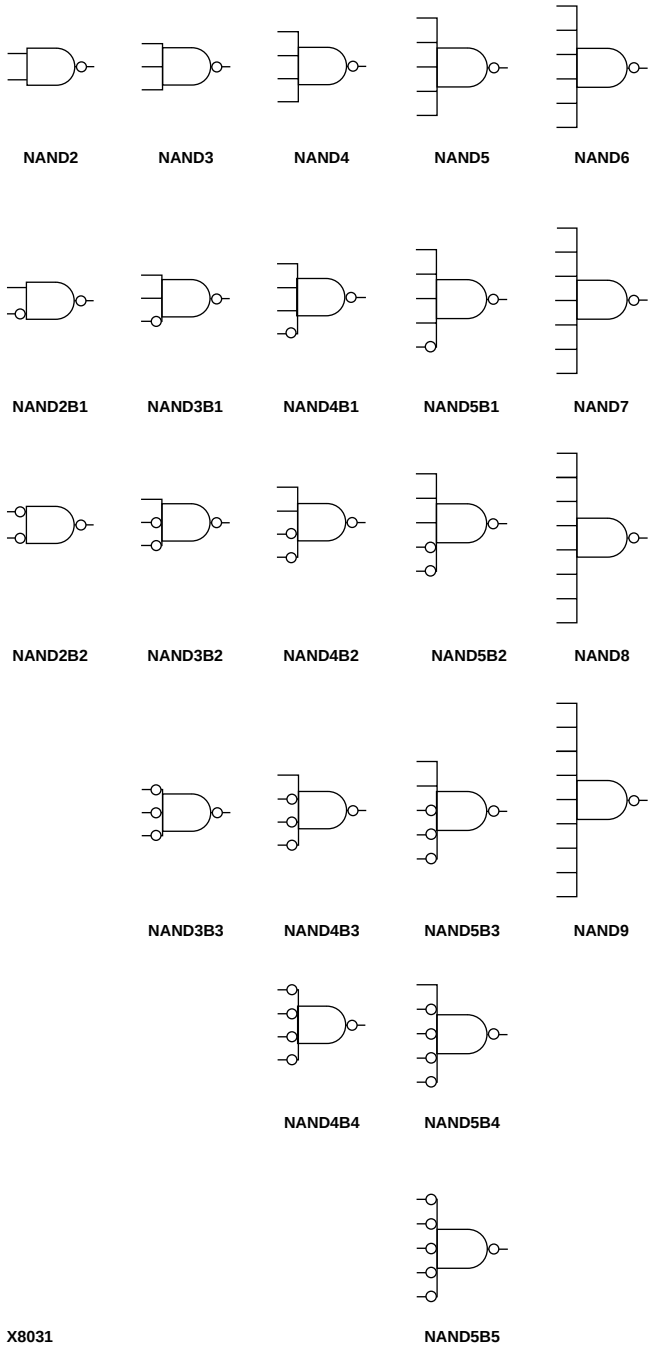


Figure 7-25 NAND Gate Representations

The NAND function is performed in the Configurable Logic Block (CLB) function generators for XC3000, XC4000, XC5200, Spartan, and SpartanXL. NAND gates of up to five inputs are available in any combination of inverting and non-inverting inputs. NAND gates of six to nine inputs are available with only non-inverting inputs. To invert inputs, use external inverters. Since each input uses a CLB resource, replace gates with unused inputs with gates having the necessary number of inputs.

Refer to the "NAND12, 16" section for information on additional NAND functions for the XC5200, Spartan2, and Virtex.

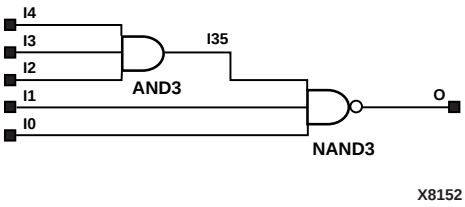


Figure 7-26 NAND5 Implementation XC5200

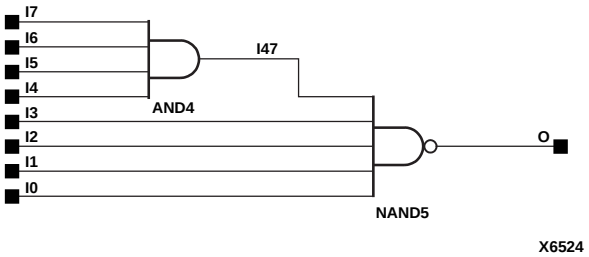


Figure 7-27 NAND8 Implementation XC3000

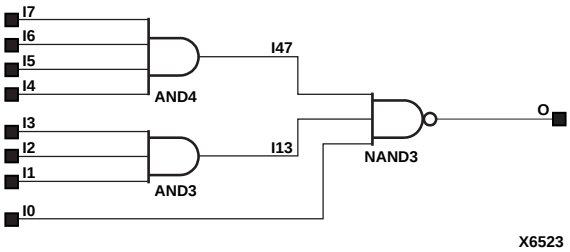


Figure 7-28 NAND8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

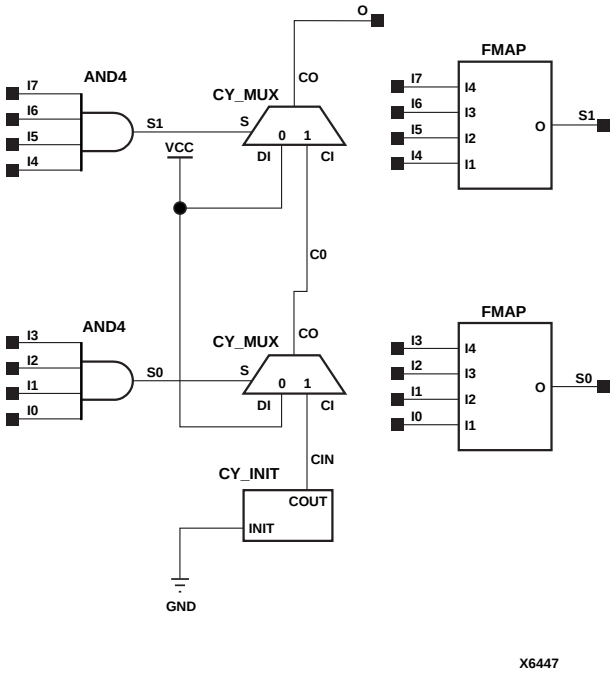


Figure 7-29 NAND8 Implementation XC5200

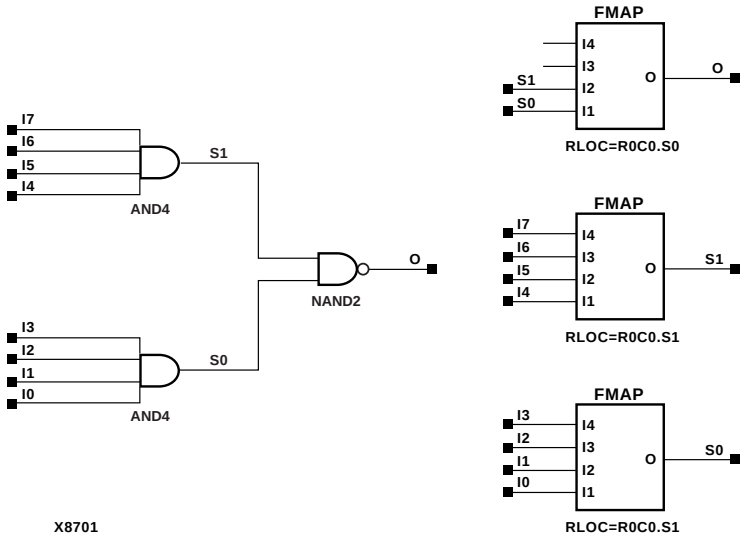
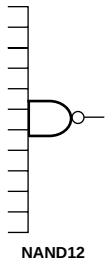


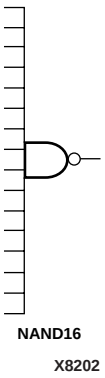
Figure 7-30 NAND8 Implementation Spartan2, Virtex

NAND12, 16

12- and 16-Input NAND Gates with Non-Inverted Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	Macro	N / A	N / A	N / A	Macro	Macro





The NAND function is performed in the Configurable Logic Block (CLB) function generators for XC5200, Spartan2, and Virtex. The 12- and 16-input NAND functions are available only with non-inverting inputs. To invert some or all inputs, use external inverters.

Refer to the “NAND2-9” section for more information on NAND functions.

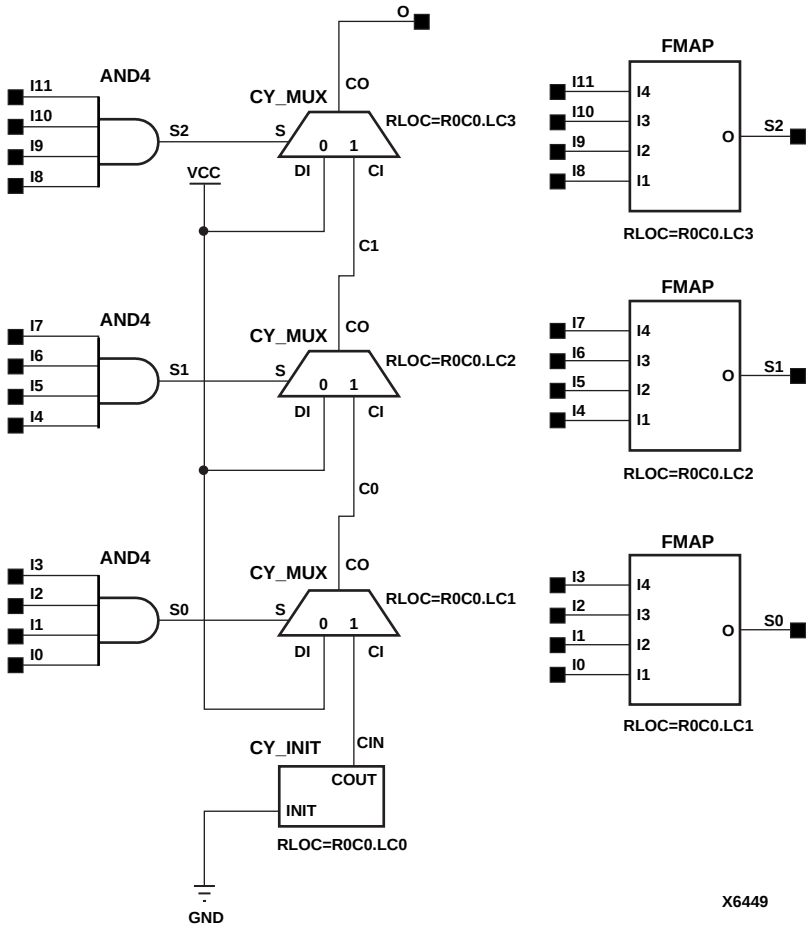


Figure 7-31 NAND12 Implementation XC5200

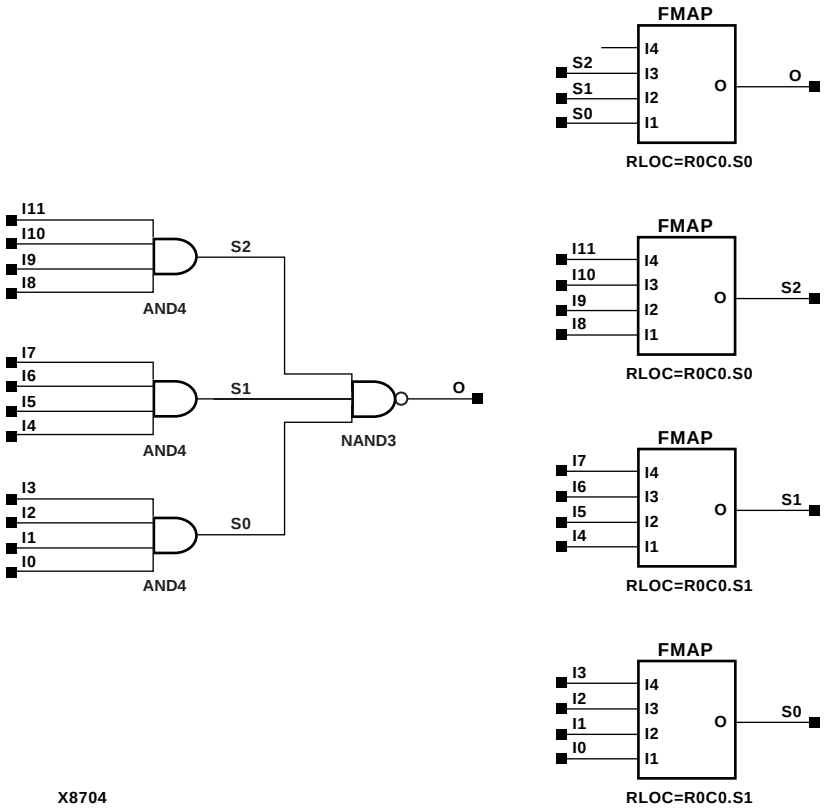


Figure 7-32 NAND12 Implementation Spartan2, Virtex

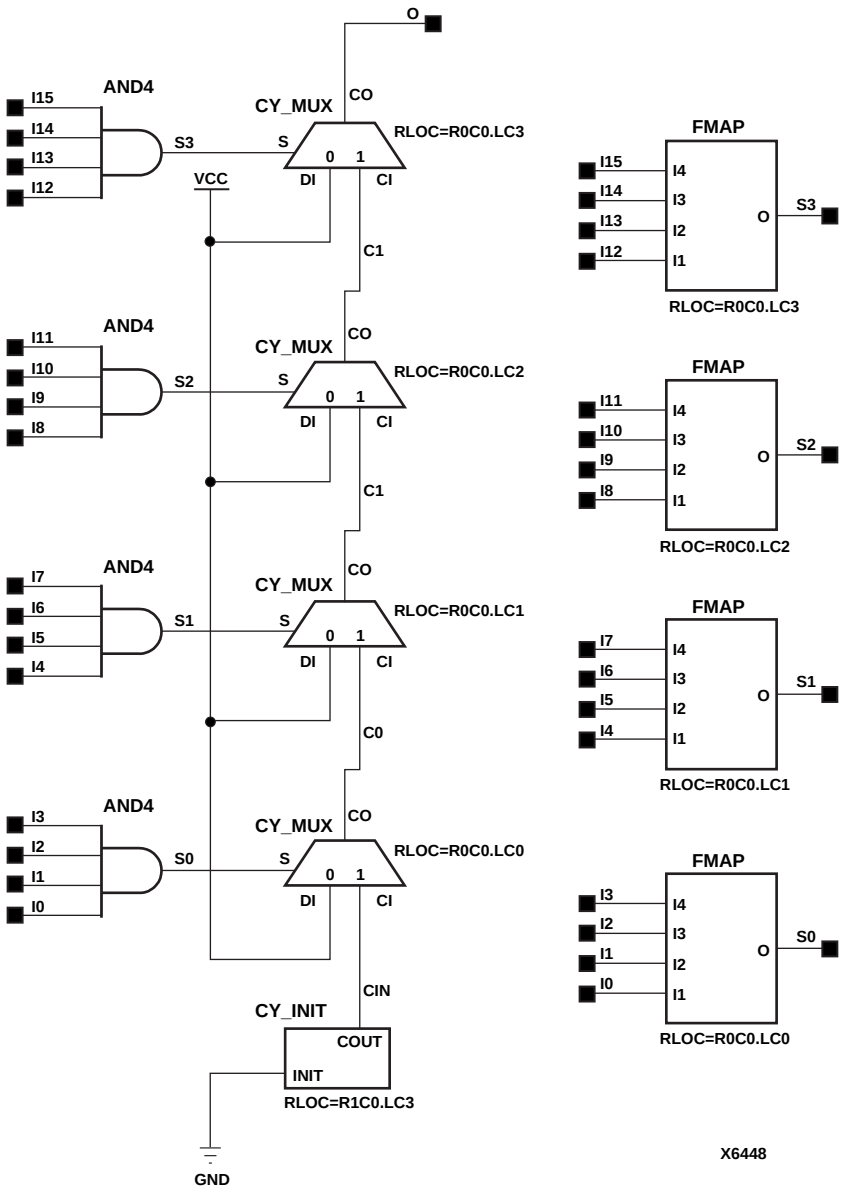


Figure 7-33 NAND16 Implementation XC5200

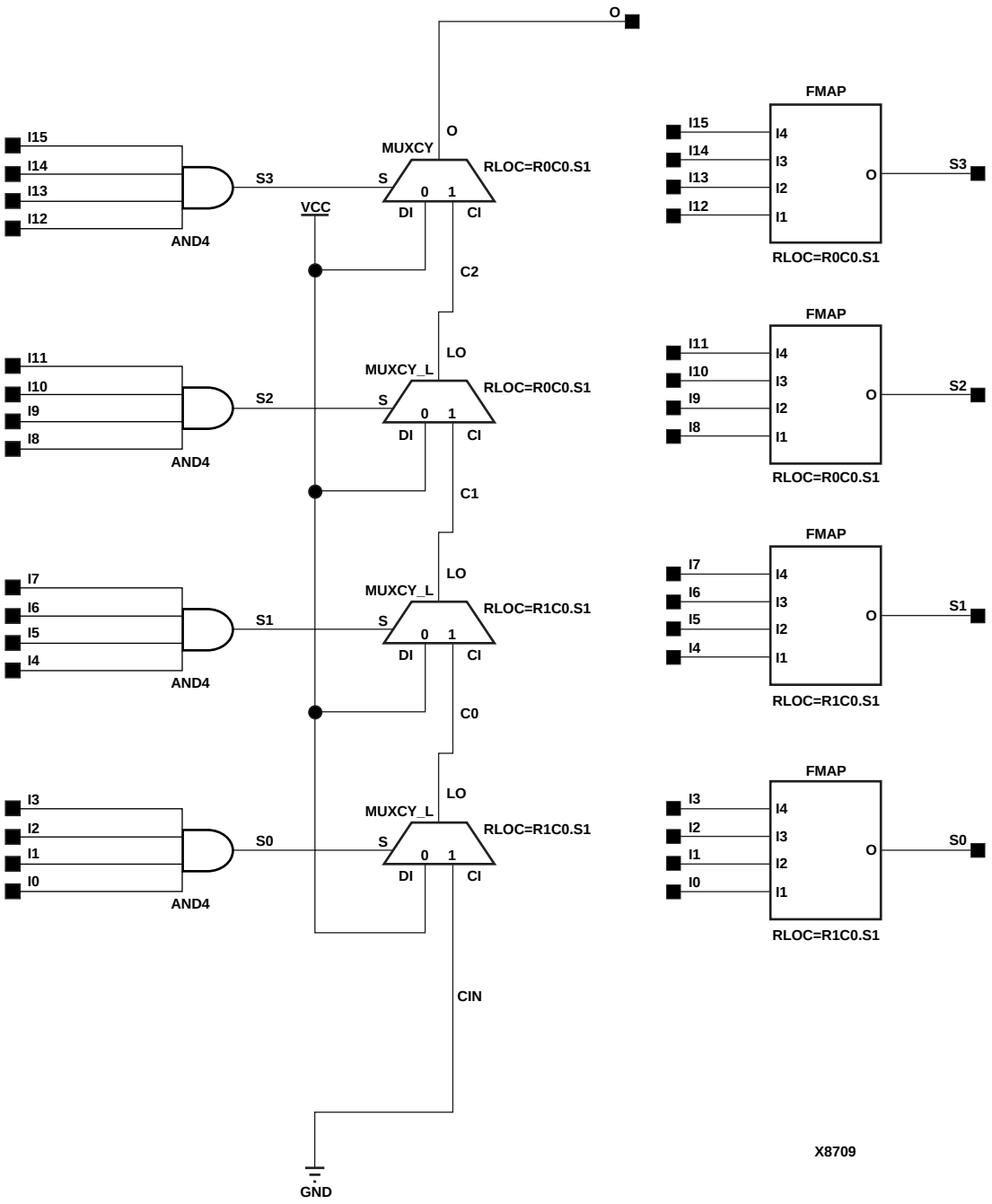


Figure 7-34 NAND16 Implementation Spartan2, Virtex

NOR2-9

2- to 9-Input NOR Gates with Inverted and Non-Inverted Inputs

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
NOR2, NOR2B1, NOR2B2, NOR3, NOR3B1, NOR3B2, NOR3B3, NOR4, NOR4B1, NOR4B2, NOR4B3, NOR4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5, NOR5B1, NOR5B2, NOR5B3, NOR5B4, NOR5B5	Primitive	Primitive	Primitive	Macro	Primitive	Primitive	Primitive	Primitive	Primitive
NOR6, NOR7, NOR8, NOR9	Macro	Macro	Macro	Macro	Primitive	Macro	Macro	Macro	Macro

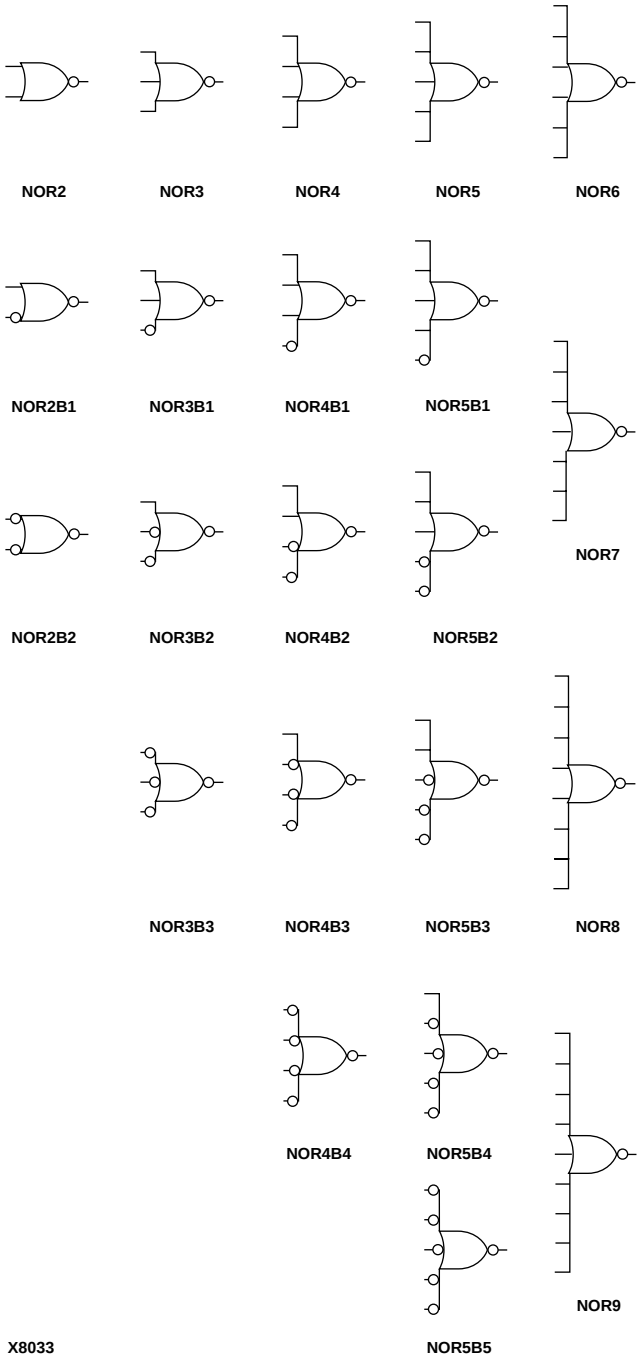


Figure 7-35 NOR Gate Representations

The NOR function is performed in the Configurable Logic Block (CLB) function generators for XC3000, XC4000, XC5200, Spartan, and SpartanXL. NOR gates of up to five inputs are available in any combination of inverting and non-inverting inputs. NOR gates of six to nine inputs are available with only non-inverting inputs. To invert some or all inputs, use external inverters. Since each input uses a CLB resource, replace gates with unused inputs with gates having the necessary number of inputs.

Refer to the “NOR12, 16” section for information on additional NOR functions for the XC5200, Spartan2, and Virtex.

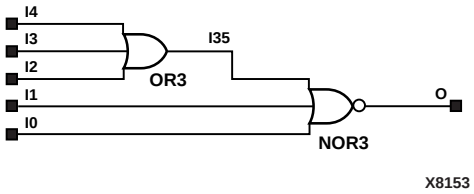


Figure 7-36 NOR5 Implementation XC5200

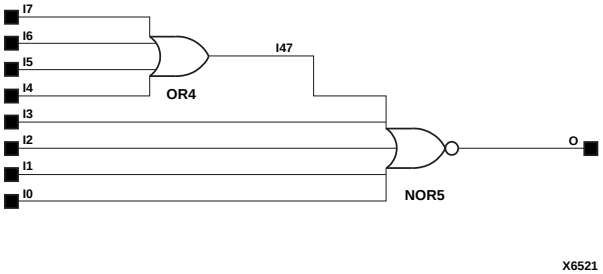


Figure 7-37 NOR8 Implementation XC3000

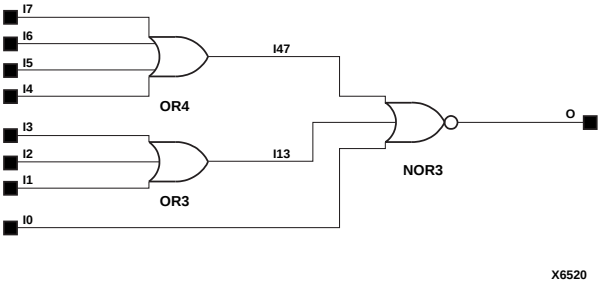


Figure 7-38 NOR8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

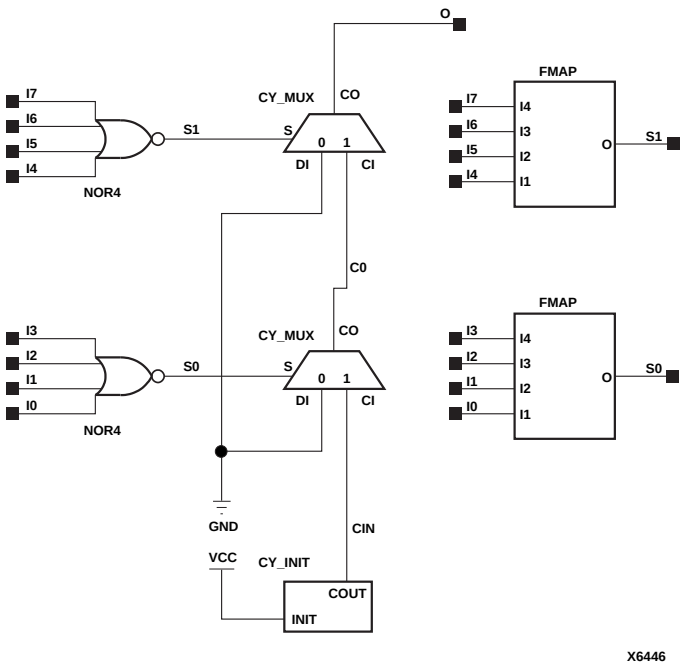


Figure 7-39 NOR8 Implementation XC5200

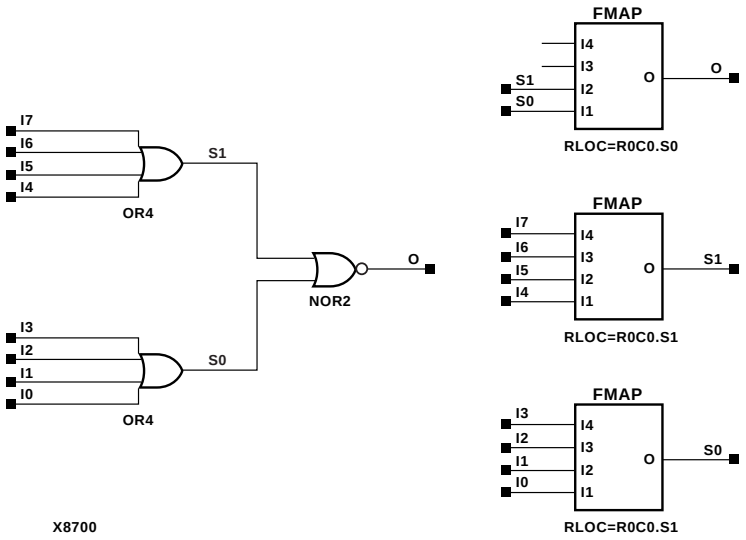
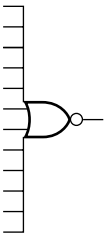


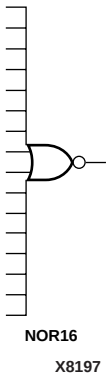
Figure 7-40 NOR8 Implementation Spartan2, Virtex

NOR12, 16

12- and 16-Input NOR Gates with Non-Inverted Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	Macro	N / A	N / A	N / A	Macro	Macro





The 12- and 16-input NOR functions are available only with non-inverting inputs. To invert some or all inputs, use external inverters.

Refer to the “NOR2-9” section for more information on NOR functions.

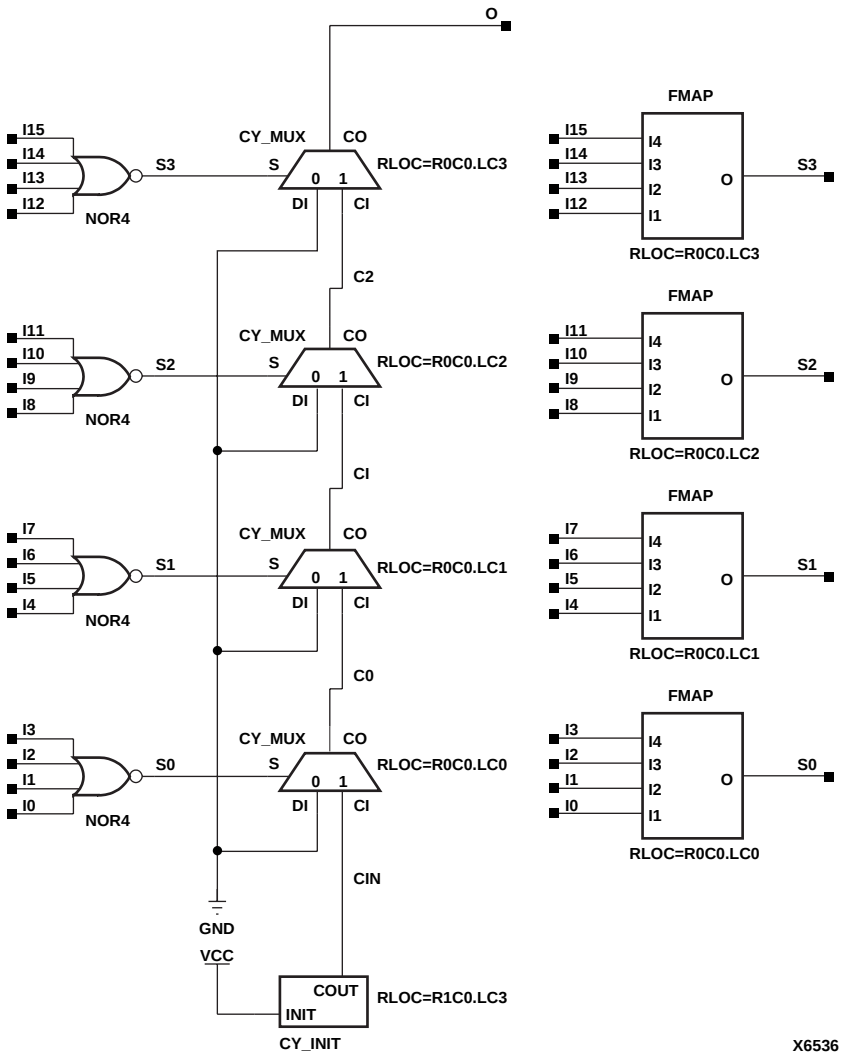


Figure 7-41 NOR16 Implementation XC5200

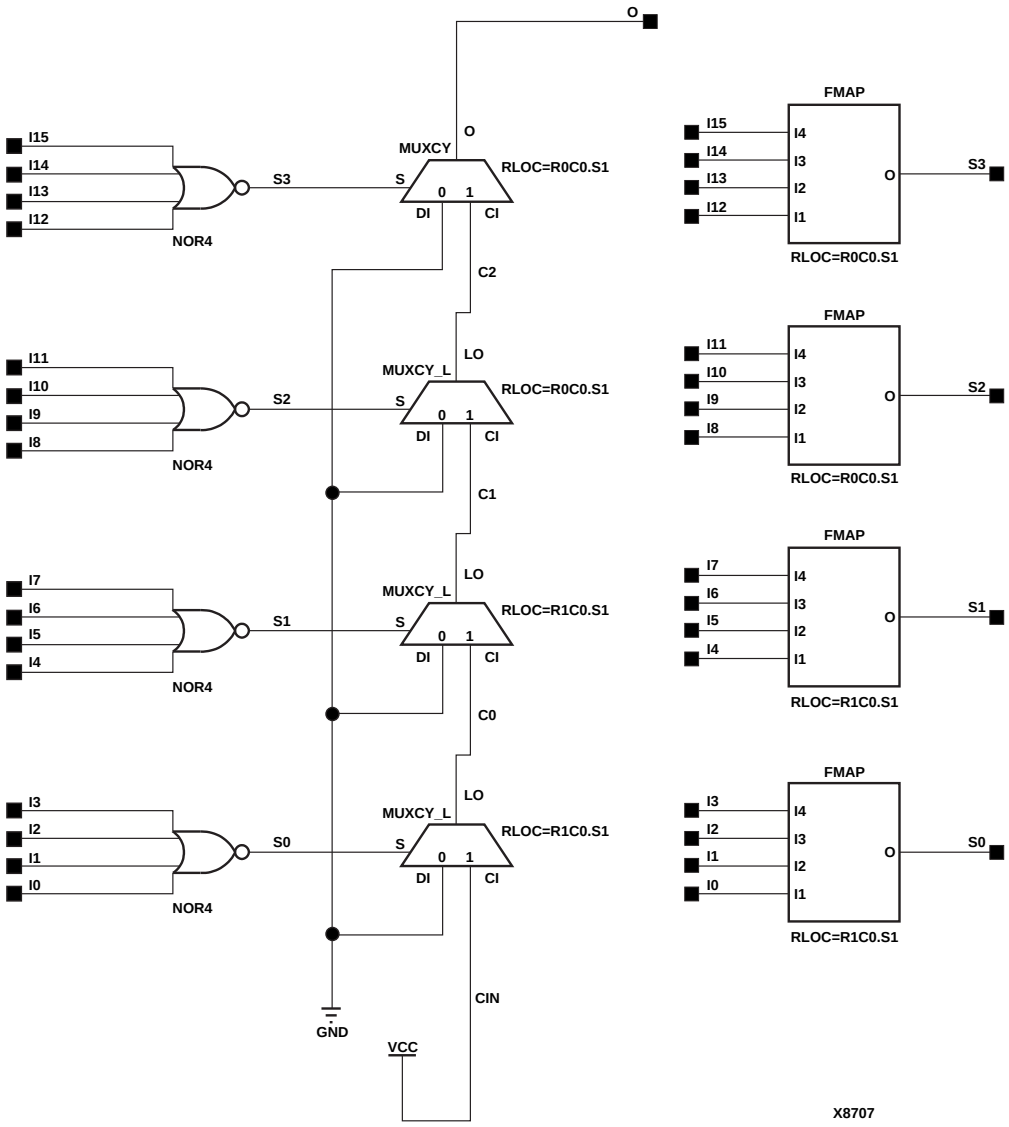


Figure 7-42 NOR16 Implementation Spartan2, Virtex

Chapter 8

Design Elements (OAND2 to OXOR2)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

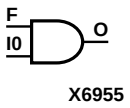
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

OAND2

2-Input AND Gate with Invertible Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



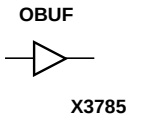
X6955

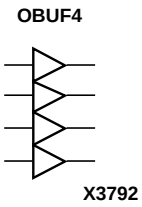
OAND2 is a 2-input AND gate that is implemented in the output multiplexer of the XC4000X IOB. The F pin is faster than I0. Input pins can be inverted even though there is no library component showing inverted inputs. The mapper will automatically bring any inverted input pins into the IOB.

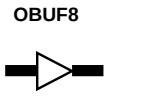
OBUF, 4, 8, 16

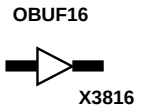
Single- and Multiple-Output Buffers

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OBUF	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUF4, OBUF8, OBUF16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









OBUF, OBUF4, OBUF8, and OBUF16 are single and multiple output buffers. An OBUF isolates the internal circuit and provides drive current for signals leaving a chip. OBUFs exist in input/output blocks (IOB). The output (O) of an OBUF is connected to an OPAD or an IOPAD.

For XC9000 CPLDs, if a high impedance (Z) signal from an on-chip 3-state buffer (like BUFE) is applied to the input of an OBUF, it is propagated to the CPLD device output pin.

For Virtex and Spartan2, refer to the "OBUF_selectIO" section for information on OBUF variants with selectable I/O interfaces. The I/O interface standard used by OBUF, 4, 8, and 16 is LVTTL. Also, Virtex and Spartan2 OBUF, 4, 8, and 16 have selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

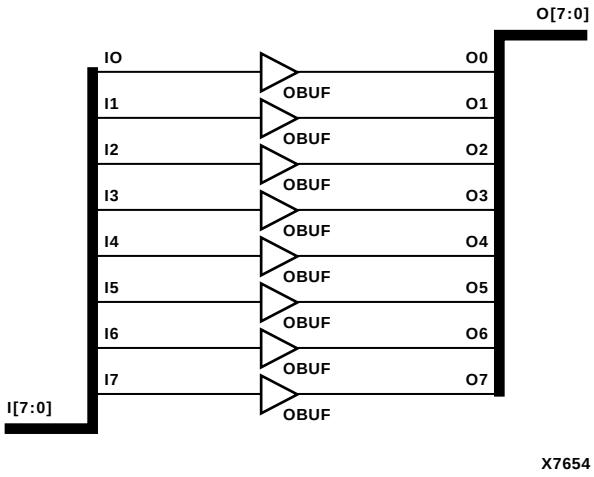
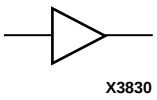


Figure 8-1 OBUF8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

OBUF_*selectIO*

Single Output Buffer with Selectable I/O Interface

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



OBUF and its variants (listed below) are single output buffers whose I/O interface corresponds to a specific I/O standard. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. The S, F, and 2, 4, 6, 8, 12, 16, 24 extensions specify the slew rate (SLOW or FAST) and the drive power (2, 4, 6, 8, 12, 16, 24 mA) for the LVTTL standard variants. For example, OBUF_F_12 is a single output buffer that uses the LVTTL I/O-signaling standard with a FAST slew and 12mA of drive power.

OBUF has selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

An OBUF isolates the internal circuit and provides drive current for signals leaving a chip. OBUFs exist in input/output blocks (IOB). The output (O) of an OBUF is connected to an OPAD or an IOPAD.

The hardware implementation of the I/O standard requires that you follow a set of usage rules for the SelectI/O buffer components. Refer to the "SelectI/O Usage Rules" section under the IBUF_*selectIO* section for information on using these components.

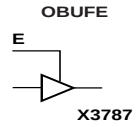
Component	I/O Standard	VCCO
OBUF	LVTTL	3.3
OBUF_S_2	LVTTL	3.3
OBUF_S_4	LVTTL	3.3
OBUF_S_6	LVTTL	3.3
OBUF_S_8	LVTTL	3.3
OBUF_S_12	LVTTL	3.3
OBUF_S_16	LVTTL	3.3
OBUF_S_24	LVTTL	3.3
OBUF_F_2	LVTTL	3.3
OBUF_F_4	LVTTL	3.3
OBUF_F_6	LVTTL	3.3
OBUF_F_8	LVTTL	3.3
OBUF_F_12	LVTTL	3.3
OBUF_F_16	LVTTL	3.3
OBUF_F_24	LVTTL	3.3
OBUF_LVCMOS2	LVCMOS2	2.5
OBUF_PCI33_3	PCI33_3	3.3
OBUF_PCI33_5	PCI33_5	3.3

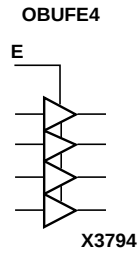
Component	I/O Standard	VCCO
OBUF_PCI66_3	PCI66_3	3.3
OBUF_GTL	GTL	N / A
OBUF_GTLP	GTL+	N / A
OBUF_HSTL_I	HSTL_I	1.5
OBUF_HSTL_III	HSTL_III	1.5
OBUF_HSTL_IV	HSTL_IV	1.5
OBUF_SSTL2_I	SSTL2_I	2.5
OBUF_SSTL2_II	SSTL2_II	2.5
OBUF_SSTL3_I	SSTL3_I	3.3
OBUF_SSTL3_II	SSTL3_II	3.3
OBUF_CTT	CTT	3.3
OBUF_AGP	AGP	3.3

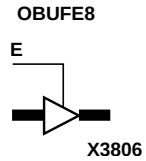
OBUFE, 4, 8, 16

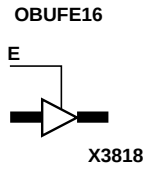
3-State Output Buffers with Active-High Output Enable

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OBUFE	Macro	Macro	Macro	Macro	Primitive	Macro	Macro	Macro	Macro
OBUFE4, OBUFE8, OBUFE16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









OBUFE, OBUFE4, OBUFE8, and OBUFE16 are 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15-I0, respectively; outputs O, O3 – O0, O7 – O0, and O15-O0, respectively; and active-High output enable (E). When E is High, data on the inputs of the buffers is transferred to the corresponding outputs. When E is Low, the output is High impedance (off or Z state). An OBUFE isolates the internal circuit and provides drive current for signals leaving a chip. An OBUFE output is connected to an OPAD or an IOPAD. An OBUFE input is connected to the internal circuit.

Inputs		Outputs
E	I	O
0	X	Z
1	1	1
1	0	0

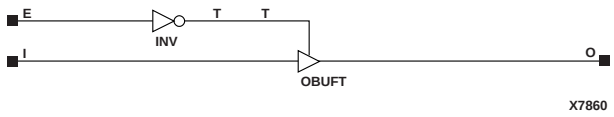


Figure 8-2 OBUFE Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

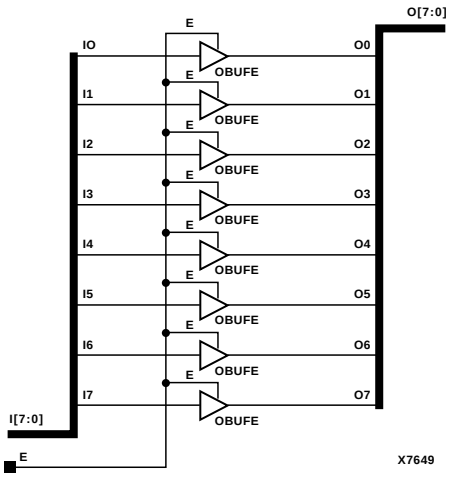
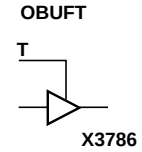


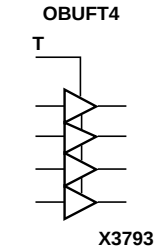
Figure 8-3 OBUFE8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

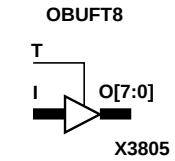
OBUFT, 4, 8, 16

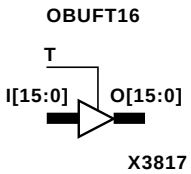
Single and Multiple 3-State Output Buffers with Active-Low Output Enable

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OBUFT	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUFT4, OBUFT8, OBUFT16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









OBUFT, OBUFT4, OBUFT8, and OBUFT16 are single and multiple 3-state output buffers with inputs I, I3 – I0, I7 – I0, I15 – I0, outputs O, O3 – O0, O7 – O0, O15 – O0, and active-Low output enables (T). When T is Low, data on the inputs of the buffers is transferred to the corresponding outputs. When T is High, the output is high impedance (off or Z state). OBUFTs isolate the internal circuit and provide extra drive current for signals leaving a chip. An OBUFT output is connected to an OPAD or an IOPAD.

For Virtex and Spartan2, refer to the "OBUFT_selectIO" section for information on OBUFT variants with selectable I/O interfaces. OBUFT, 4, 8, and 16 use the LVTTL standard. Also, Virtex and Spartan2 OBUFT, 4, 8, and 16 have selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0

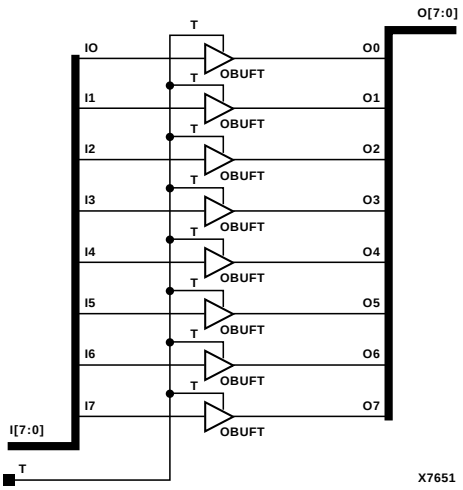
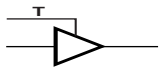


Figure 8-4 OBUFT8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

OBUFT_*selectIO*

Single 3-State Output Buffer with Active-Low Output Enable and Selectable I/O Interface

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



OBUFT and its variants (listed below) are single 3-state output buffers with active-Low output Enable whose I/O interface corresponds to a specific I/O standard. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. The S, F, and 2, 4, 6, 8, 12, 16, 24 extensions specify the slew rate (SLOW or FAST) and the drive power (2, 4, 6, 8, 12, 16, 24 mA) for the LVTTL standard. For example, OBUFT_S_4 is a 3-state output buffer with active-Low output enable that uses the LVTTL I/O signaling standard with a SLOW slew and 4mA of drive power.

OBUFT has selectable drive and slew rates using the DRIVE and FAST or SLOW constraints. The defaults are DRIVE=12 mA and SLOW slew.

When T is Low, data on the input of the buffer is transferred to the output. When T is High, the output is high impedance (off or Z state). OBUFTs isolate the internal circuit and provide extra drive current for signals leaving a chip. An OBUFT output is connected to an OPAD or an IOPAD.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffer components. Refer to the "SelectI/O Usage Rules" section under the IBUF_*selectIO* section for information on using these components.

Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0

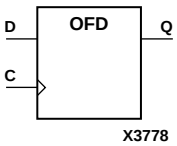
Component	I/O Standard	VCCO
OBUFT	LVTTL	3.3
OBUFT_S_2	LVTTL	3.3
OBUFT_S_4	LVTTL	3.3
OBUFT_S_6	LVTTL	3.3
OBUFT_S_8	LVTTL	3.3
OBUFT_S_12	LVTTL	3.3
OBUFT_S_16	LVTTL	3.3
OBUFT_S_24	LVTTL	3.3
OBUFT_F_2	LVTTL	3.3

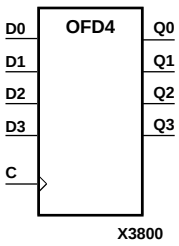
Component	I/O Standard	VCCO
OBUFT_F_4	LVTTL	3.3
OBUFT_F_6	LVTTL	3.3
OBUFT_F_8	LVTTL	3.3
OBUFT_F_12	LVTTL	3.3
OBUFT_F_16	LVTTL	3.3
OBUFT_F_24	LVTTL	3.3
OBUFT_LVCMOS2	LVCMOS2	2.5
OBUFT_PCI33_3	PCI33_3	3.3
OBUFT_PCI33_5	PCI33_5	3.3
OBUFT_PCI66_3	PCI66_3	3.3
OBUFT_GTL	GTL	N / A
OBUFT_GTLP	GTL+	N / A
OBUFT_HSTL_I	HSTL_I	1.5
OBUFT_HSTL_III	HSTL_III	1.5
OBUF_HSTL_IV	HSTL_IV	1.5
OBUFT_SSTL2_I	SSTL2_I	2.5
OBUFT_SSTL2_II	SSTL2_II	2.5
OBUFT_SSTL3_I	SSTL3_I	3.3
OBUFT_SSTL3_II	SSTL3_II	3.3
OBUFT_CTT	CTT	3.3
OBUFT_AGP	AGP	3.3

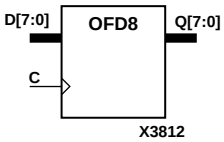
OFD, 4, 8, 16

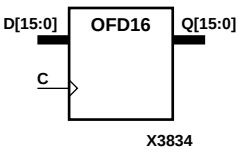
Single- and Multiple-Output D Flip-Flops

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OFD	Primitive	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro
OFD4, OFD8, OFD16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









OFD, OFD4, OFD8, and OFD16 are single and multiple output D flip-flops except for XC5200 and XC9000. The flip-flops exist in an input / output block (IOB) for XC3000, XC4000, Spartan, and SpartanXL. The outputs (for example, Q3 – Q0) are connected to OPADs or IOPADs. The data on the D inputs is loaded into the flip-flops during the Low-to-High clock (C) transition and appears on the Q outputs.

The flip-flops are asynchronously cleared with Low outputs when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs		Outputs
D	C	Q
D	↑	dn

dn = state of referenced input one setup time prior to active clock transition

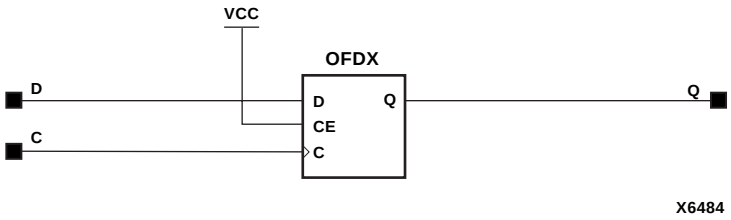


Figure 8-5 OFD Implementation XC4000E, XC4000X, Spartan, SpartanXL

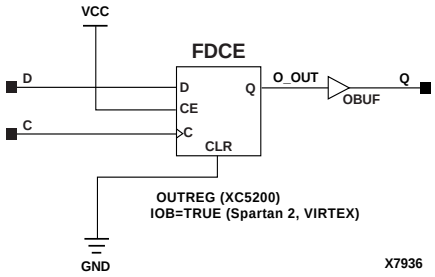


Figure 8-6 OFD Implementation XC5200, Spartan2, Virtex

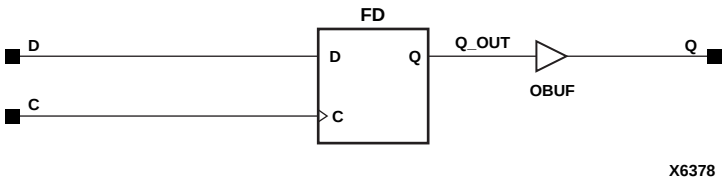


Figure 8-7 OFD Implementation XC9000

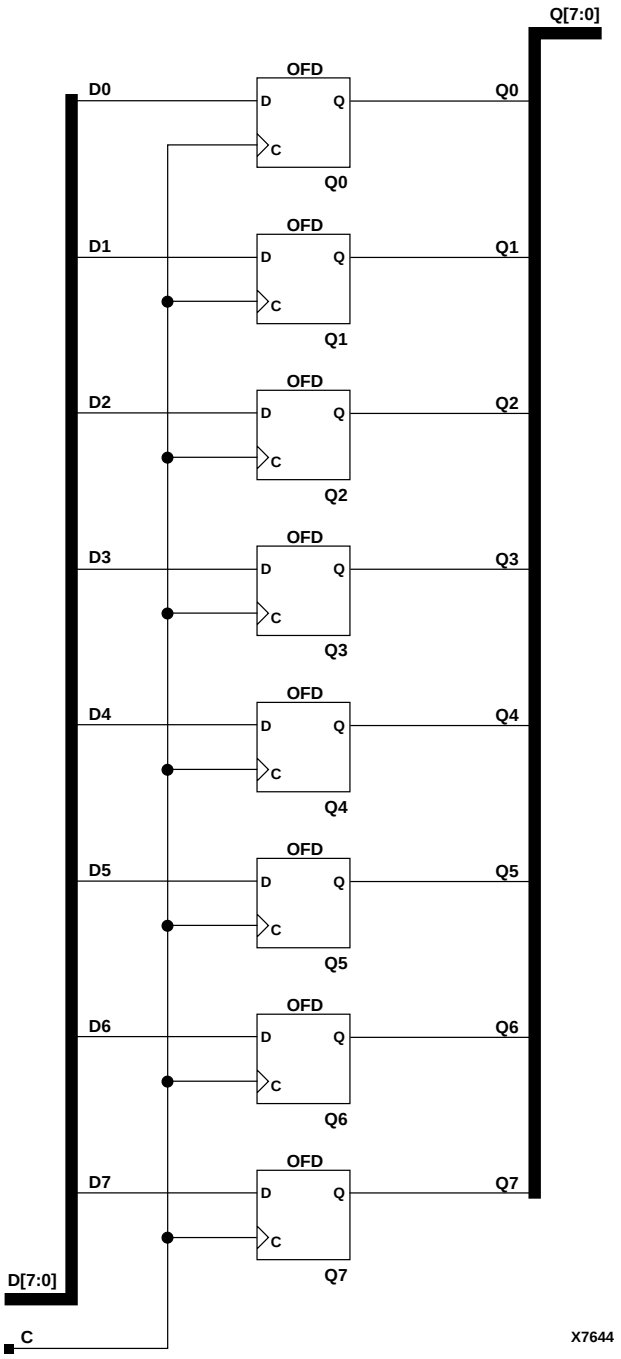


Figure 8-8 OFD8 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

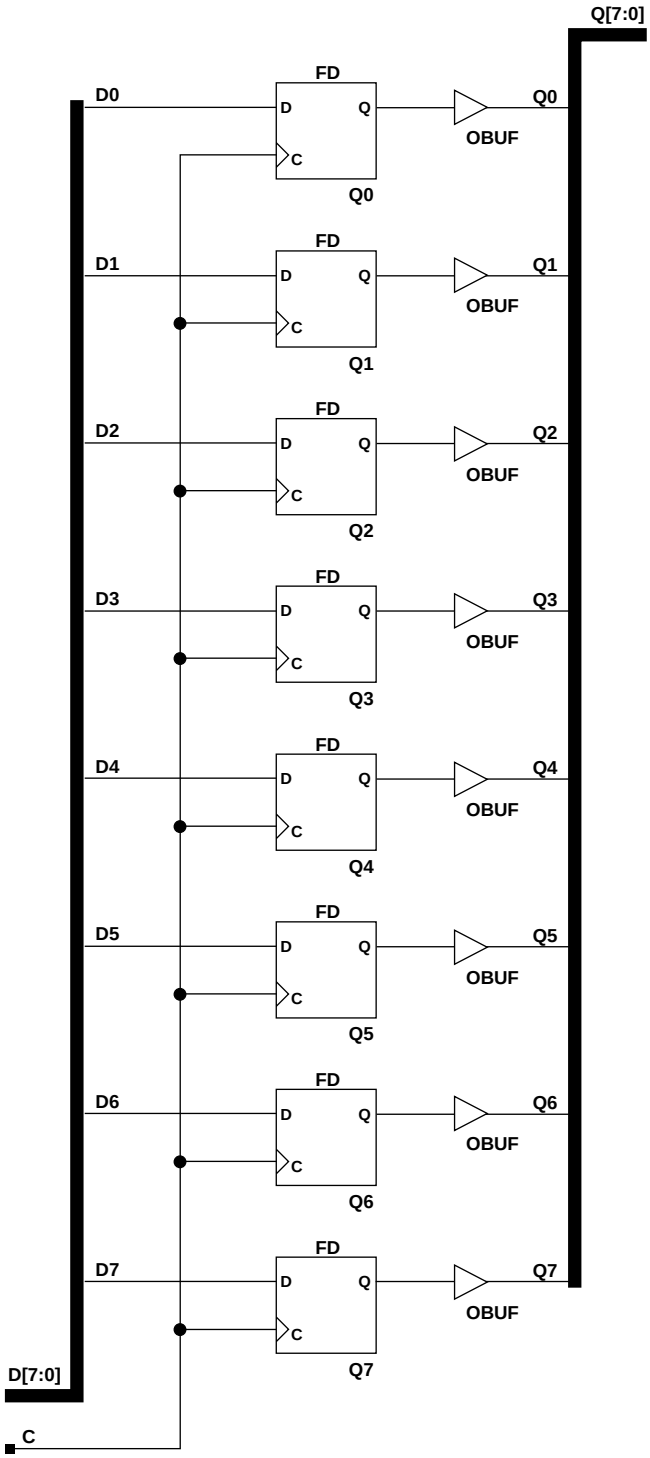
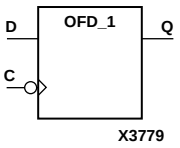


Figure 8-9 OFD8 Implementation XC9000

OFD_1

Output D Flip-Flop with Inverted Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Macro	Macro



OFD_1 is located in an input/output block (IOB) except for XC5200. The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output.

The flip-flop is asynchronously cleared, output Low, when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs		Outputs
D	C	Q
D	↓	d

d = state of referenced input one setup time prior to active clock transition

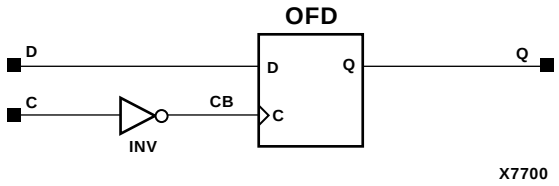


Figure 8-10 OFD_1 Implementation XC3000, XC4000E, XC4000X, Spartan, SpartanXL

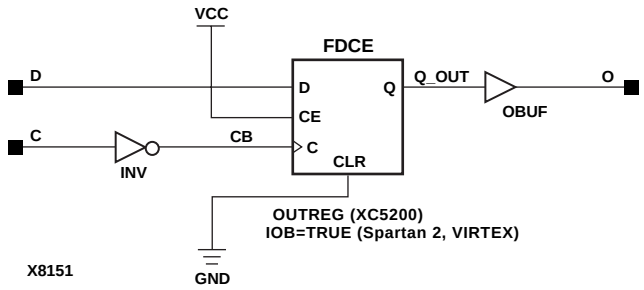
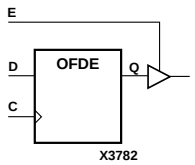


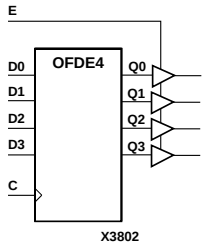
Figure 8-11 OFD_1 Implementation XC5200, Spartan2, Virtex

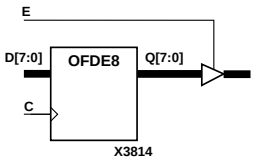
OFDE, 4, 8, 16

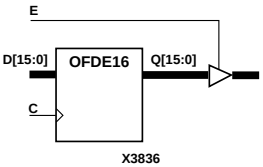
D Flip-Flops with Active-High Enable Output Buffers

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OFDE	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro
OFDE4, OFDE8, OFDE16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









OFDE, OFDE4, OFDE8, and OFDE16 are single or multiple D flip-flops whose outputs are enabled by tristate buffers. The flip-flop data outputs (Q) are connected to the inputs of output buffers (OBUFE). The OBUFE outputs (O) are connected to OPADs or IOPADs. These flip-flops and buffers are contained in input/output blocks (IOB) for XC3000 and XC4000. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-High enable inputs (E) are High, the data on the flip-flop outputs (Q) appears on the O outputs. When E is Low, outputs are high impedance (Z state or Off).

The flip-flops are asynchronously cleared with Low outputs when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
E	D	C	O
0	X	X	Z, not off
1	1	↑	1
1	0	↑	0

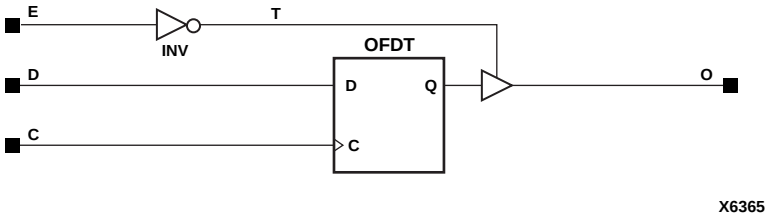


Figure 8-12 OFDE Implementation XC3000, XC4000E, XC4000X, Spartan, SpartanXL

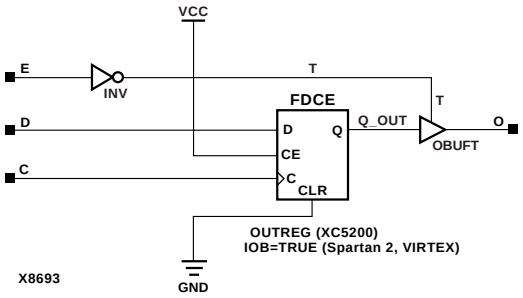


Figure 8-13 OFDE Implementation XC5200, Spartan2, Virtex

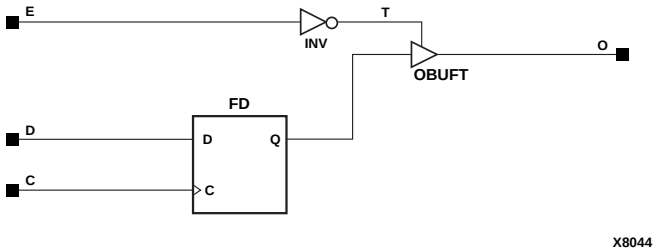


Figure 8-14 OFDE Implementation XC9000

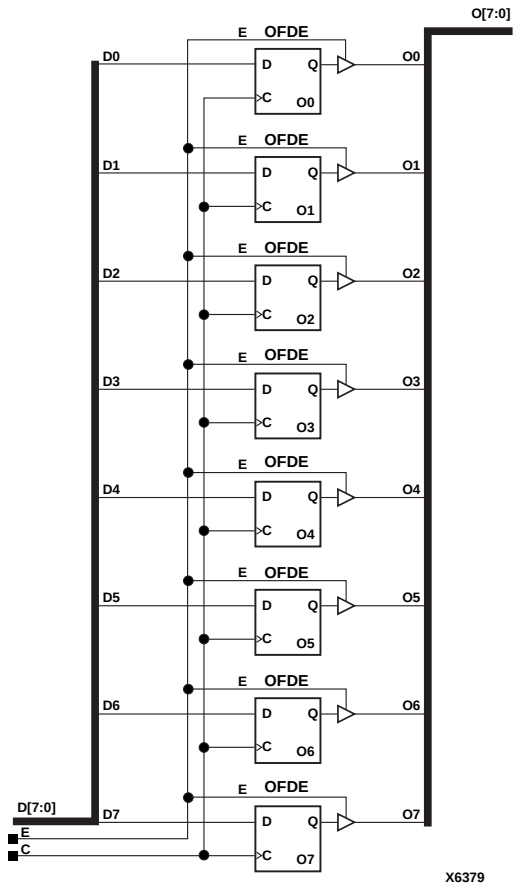
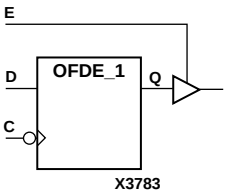


Figure 8-15 OFDE8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

OFDE_1

D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Macro	Macro



OFDE_1 and its output buffer are located in an input/output block (IOB) except for XC5200. The data output of the flip-flop (Q) is connected to the input of an output buffer or OBUFE. The output of the OBUFE is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or Off).

The flip-flop is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
E	**D**	**C**	**O**
0	X	X	Z
1	1	↓	1
1	0	↓	0

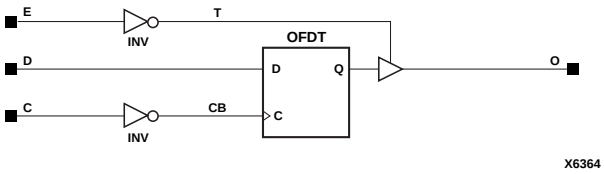


Figure 8-16 OFDE_1 Implementation XC3000, XC4000E, XC4000X, Spartan, SpartanXL

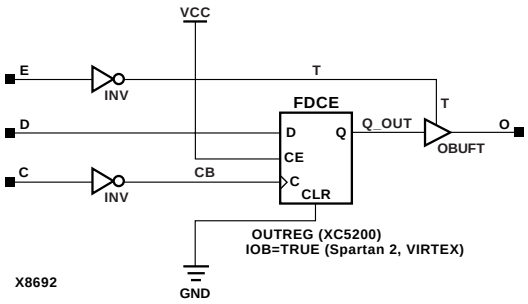
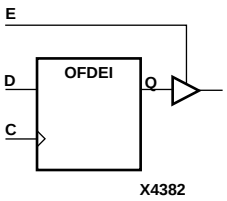


Figure 8-17 OFDE_1 Implementation XC5200, Spartan2, Virtex

OFDEI

D Flip-Flop with Active-High Enable Output Buffer (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDEI is a D flip-flop whose output is enabled by a 3-state buffer. The data output (Q) of the flip-flop is connected to the input of an output 3-state buffer or OBUFE. The output of the OBUFE (O) is connected to an OPAD or an IOPAD. These flip-flops and buffers are contained in input / output blocks (IOB). The data on the data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or off).

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs			Outputs
E	D	C	O
0	X	X	Z
1	1	↑	1
1	0	↑	0

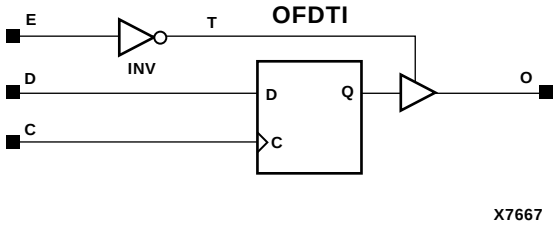
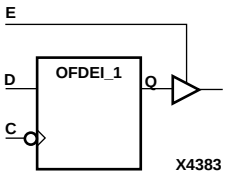


Figure 8-18 OFDEI Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDEI_1

D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDEI_1 and its output buffer exist in an input / output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer or OBUFE. The output of the OBUFE is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or off).

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs			Outputs
E	**D**	**C**	**O**
0	X	X	Z
1	1	↓	1
1	0	↓	0

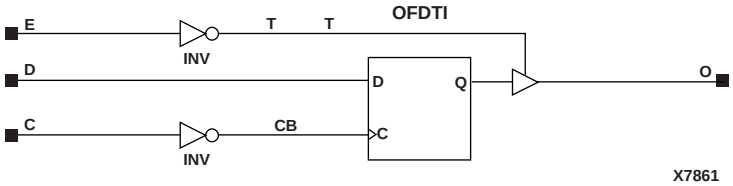
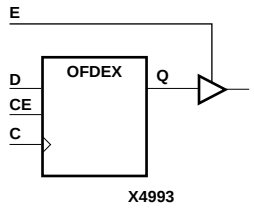


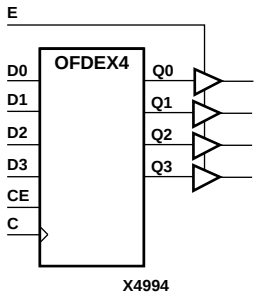
Figure 8-19 OFDEI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

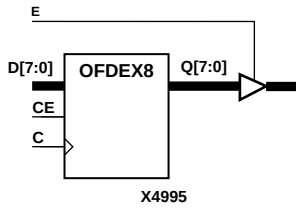
OFDEX, 4, 8, 16

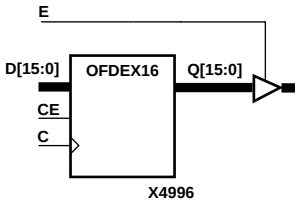
D Flip-Flops with Active-High Enable Output Buffers and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	N/A	N/A









OFDEX, OFDEX4, OFDEX8, and OFDEX16 are single or multiple D flip-flops whose outputs are enabled by tristate buffers. The flip-flop data outputs (Q) are connected to the inputs of output buffers (OBUFE). The OBUFE outputs (O) are connected to OPADs or IOPADs. These flip-flops and buffers are contained in input/output blocks (IOB). The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-High enable inputs (E) are High, the data on the flip-flop outputs (Q) appears on the O outputs. When E is Low, outputs are high impedance (Z state or Off). When CE is Low and E is High, the outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	E	D	C	O
X	0	X	X	Z, not off
1	1	1	↑	1
1	1	0	↑	0
0	1	X	X	No Chg

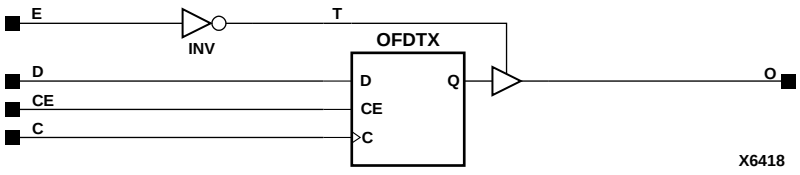


Figure 8-20 OFDEX Implementation XC4000E, XC4000X, Spartan, SpartanXL

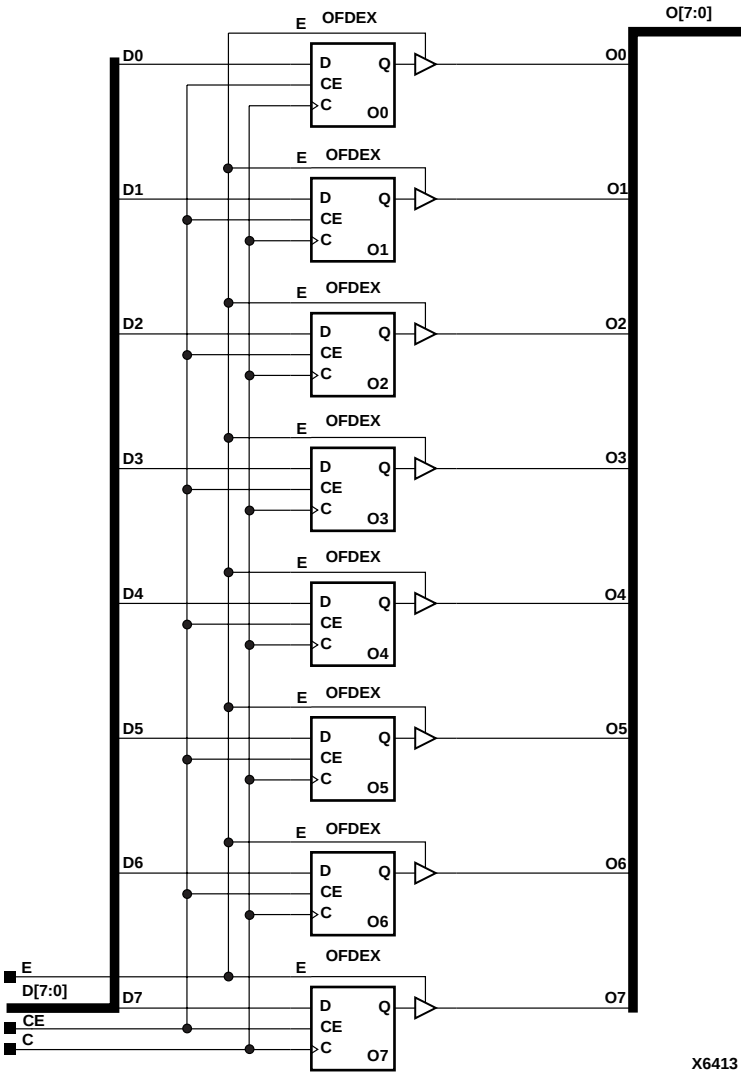
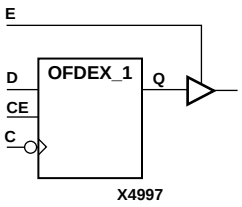


Figure 8-21 OFDEX8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDEX_1

D Flip-Flop with Active-High Enable Output Buffer, Inverted Clock, and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	N/A	N/A



OFDEX_1 and its output buffer are located in an input/output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer or OBUFE. The output of the OBUFE is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or Off). When CE is Low and E is High, the output does not change.

The flip-flop is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	E	D	C	O
X	0	X	X	Z
1	1	1	↓	1
1	1	0	↓	0
0	1	X	X	No Chg

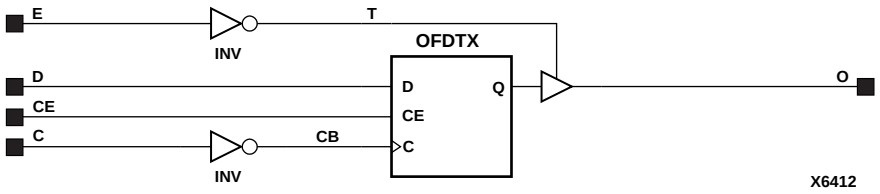
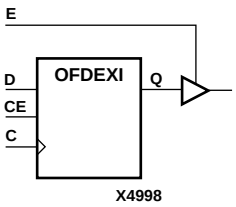


Figure 8-22 OFDEX_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDEXI

D Flip-Flop with Active-High Enable Output Buffer and Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDEXI is a D flip-flop whose output is enabled by a tristate buffer. The data output (Q) of the flip-flop is connected to the input of an output buffer or OBUFE. The output of the OBUFE (O) is connected to an OPAD or an IOPAD. These flip-flops and buffers are contained in input / output blocks (IOB). The data on the data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or Off). When CE is Low and E is High, the output does not change.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	E	D	C	O
X	0	X	X	Z
1	1	1	↑	1
1	1	0	↑	0
0	1	X	X	No Chg

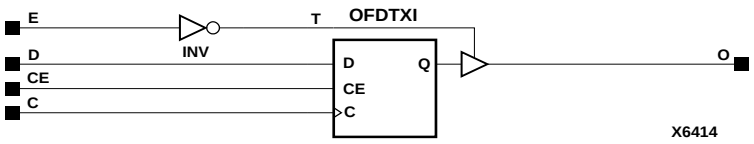
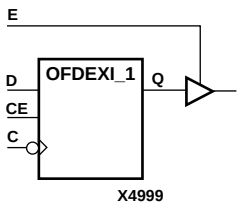


Figure 8-23 OFDEXI Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDEXI_1

D Flip-Flop with Active-High Enable Output Buffer, Inverted Clock, and Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	N/A	N/A



OFDEXI_1 and its output buffer are located in an input/output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer or OBUFE. The output of the OBUFE is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or Off). When CE is Low and E is High, the output does not change.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	E	D	C	O
X	0	X	X	Z
1	1	1	↓	1
1	1	0	↓	0
0	1	X	X	No Chg

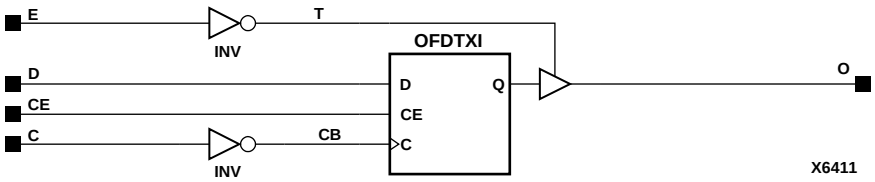
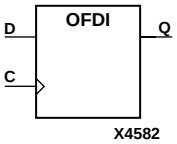


Figure 8-24 OFDEXI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDI

Output D Flip-Flop (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



OFDI is contained in an input / output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q).

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs		Outputs
D	C	Q
D	↑	d

d = state of referenced input one setup time prior to active clock transition

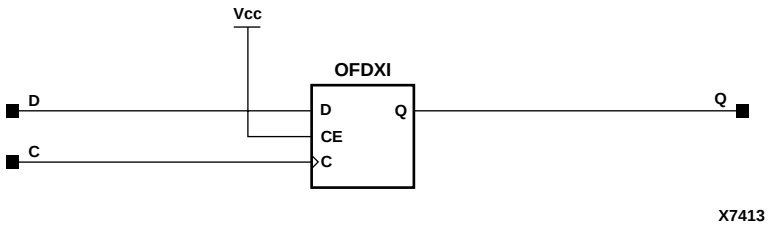


Figure 8-25 OFDI Implementation XC4000E, XC4000X, Spartan, SpartanXL

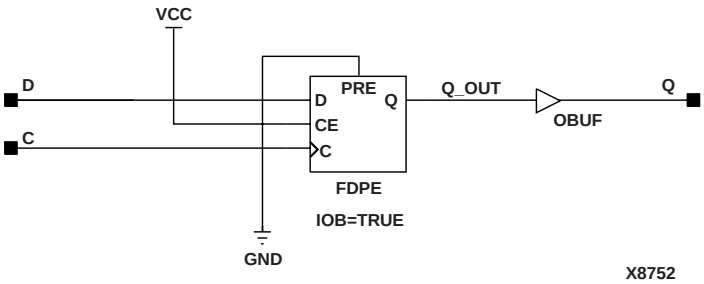
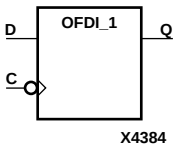


Figure 8-26 OFDI Implementation Spartan2, Virtex

OFDI_1

Output D Flip-Flop with Inverted Clock (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



OFDI_1 exists in an input / output block (IOB). The D flip-flop output (Q) is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output.

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs		Outputs
D	C	Q
D	↓	d

d = state of referenced input one setup time prior to the active clock transition

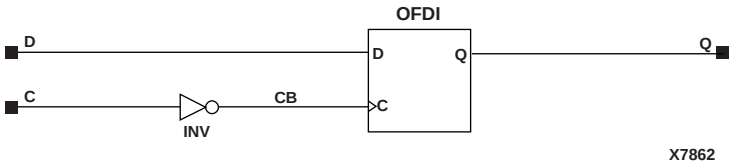


Figure 8-27 OFDI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

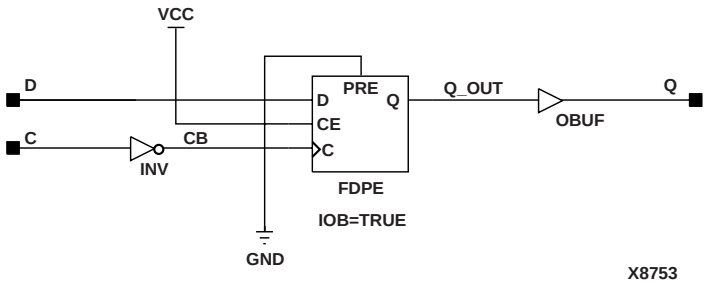
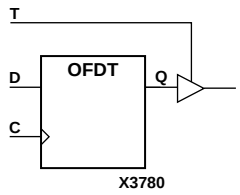


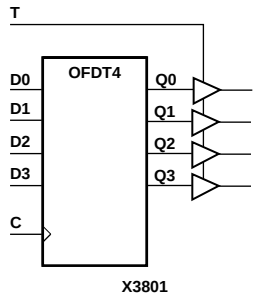
Figure 8-28 OFDI_1 Implementation Spartan2, Virtex

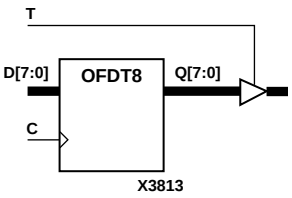
OFDT, 4, 8, 16

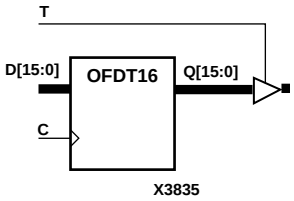
Single and Multiple D Flip-Flops with Active-Low 3-State Output Enable Buffers

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OFDT	Primitive	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro
OFDT4, OFDT8, OFDT16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro









OFDT, OFDT4, OFDT8, and OFDT16 are single or multiple D flip-flops whose outputs are enabled by a tristate buffers. The data outputs (Q) of the flip-flops are connected to the inputs of output buffers (OBUFT). The outputs of the OBUFTs (O) are connected to OPADs or IOPADs. These flip-flops and buffers are located in input/output blocks (IOB) for XC3000 and XC4000. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-Low enable inputs (T) are Low, the data on the flip-flop outputs (Q) appears on the O outputs. When T is High, outputs are high impedance (Off).

The flip-flops are asynchronously cleared with Low outputs, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	D	↑	d

d = state of referenced input one setup time prior to active clock transition

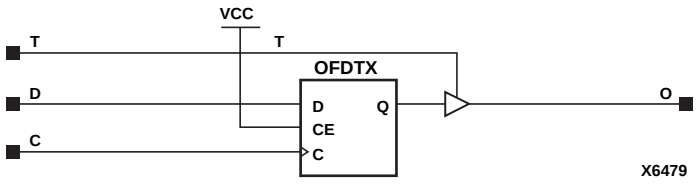


Figure 8-29 OFDT Implementation XC4000E, XC4000X, Spartan, SpartanXL

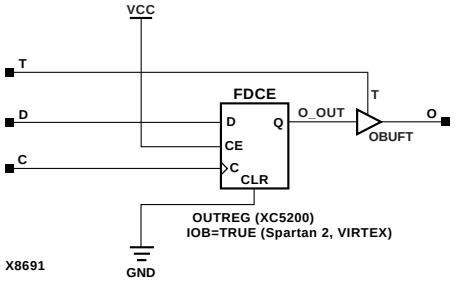


Figure 8-30 OFDT Implementation XC5200, Spartan2, Virtex

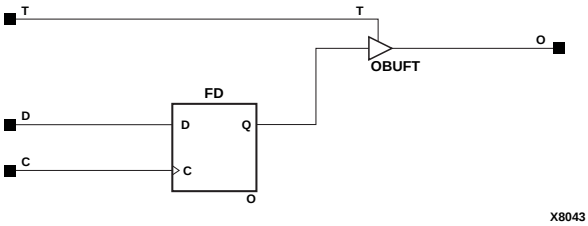


Figure 8-31 OFDT Implementation XC9000

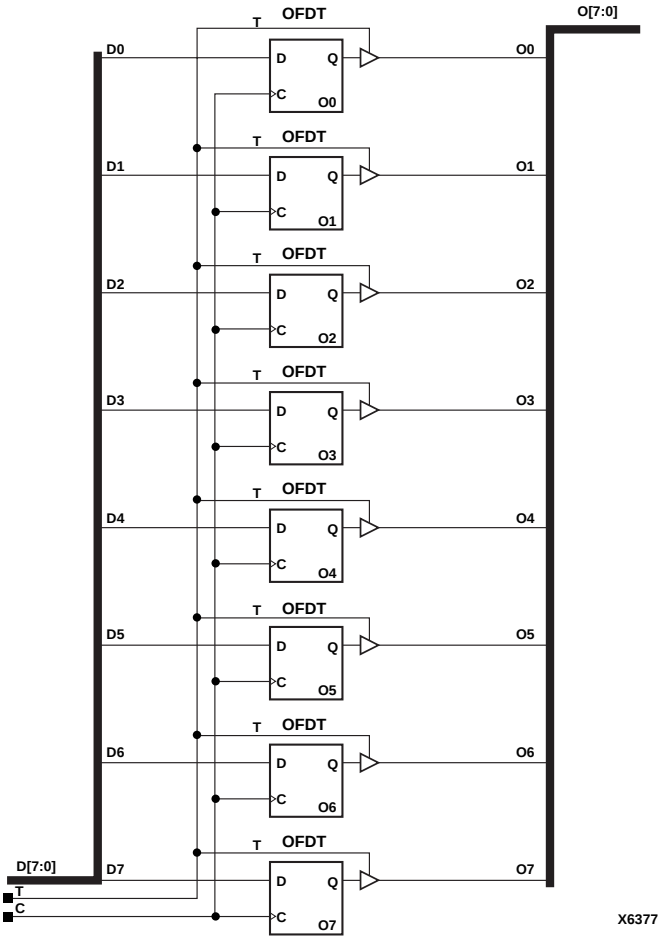
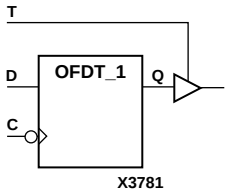


Figure 8-32 OFDT8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

OFDT_1

D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	N/A	Macro	Macro	Macro	Macro



OFDT_1 and its output buffer are located in an input/output block (IOB). The flip-flop data output (Q) is connected to the input of an output buffer (OBUFT). The OBUFT output is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the O output. When T is High, the output is high impedance (Off).

The flip-flop is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	1	↓	1
0	0	↓	0

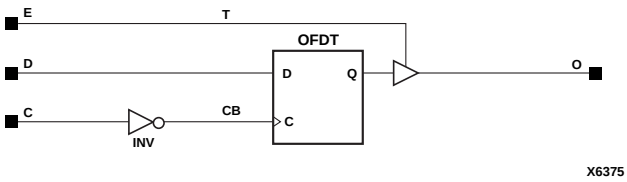


Figure 8-33 OFDT_1 Implementation XC3000, XC4000E, XC4000X, Spartan, SpartanXL

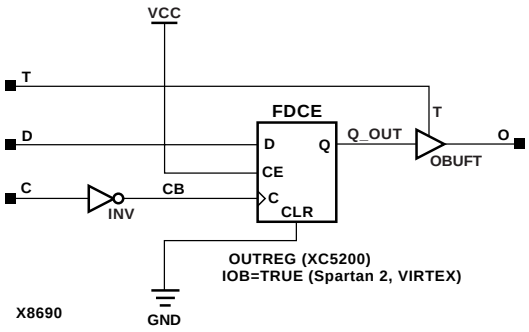
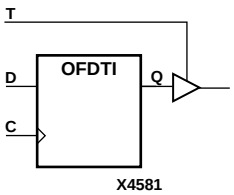


Figure 8-34 OFDT_1 Implementation XC5200, Spartan2, Virtex

OFDTI

D Flip-Flop with Active-Low 3-State Output Buffer (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDTI and its output buffer are contained in an input / output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer (OBUFT). The output of the OBUFT is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the Low-to-High clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the output (O). When T is High, the output is high impedance (off).

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	1	↑	1
0	0	↑	0

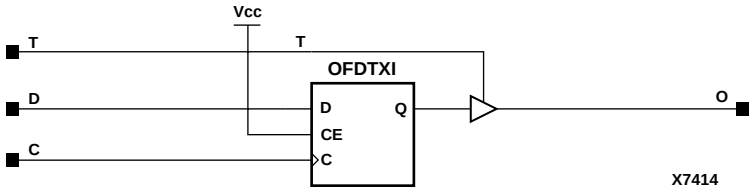
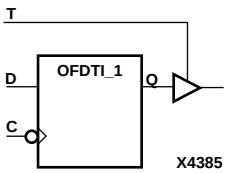


Figure 8-35 OFDTI Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDTI_1

D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDTI_1 and its output buffer are contained in an input / output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer (OBUFT). The OBUFT output is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the O output. When T is High, the output is high impedance (off).

The flip-flop is asynchronously preset, output High, when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	1	↓	1
0	0	↓	0

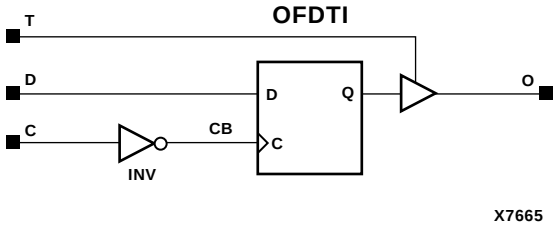
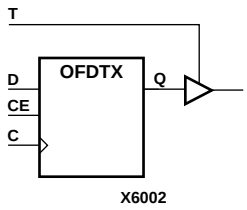


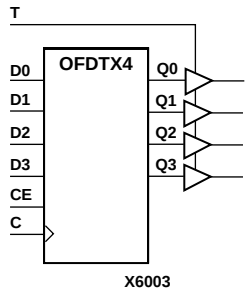
Figure 8-36 OFDTI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

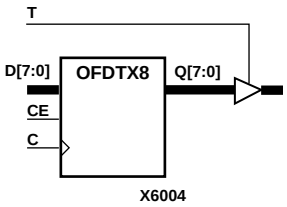
OFDTX, 4, 8, 16

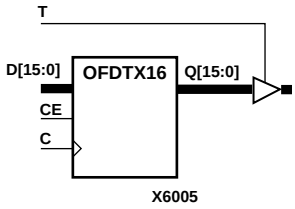
Single and Multiple D Flip-Flops with Active-Low 3-State Output Buffers and Clock Enable

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OFDTX	N / A	Primitive	Primitive	N / A	N / A	Primitive	Primitive	N / A	N / A
OFDTX4, OFDTX8, OFDTX16	N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A









OFDTX, OFDTX4, OFDTX8, and OFDTX16 are single or multiple D flip-flops whose outputs are enabled by a tristate buffers. The data outputs (Q) of the flip-flops are connected to the inputs of output buffers (OBUFT). The outputs of the OBUFTs (O) are connected to OPADs or IOPADs. These flip-flops and buffers are located in input / output blocks (IOB) for XC4000E. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-Low enable inputs (T) are Low, the data on the flip-flop outputs (Q) appears on the O outputs. When T is High, outputs are high impedance (Off). When CE is Low and T is Low, the outputs do not change.

The flip-flops are asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	T	D	C	Q
X	1	X	X	Z
1	0	D	↑	d
0	0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition

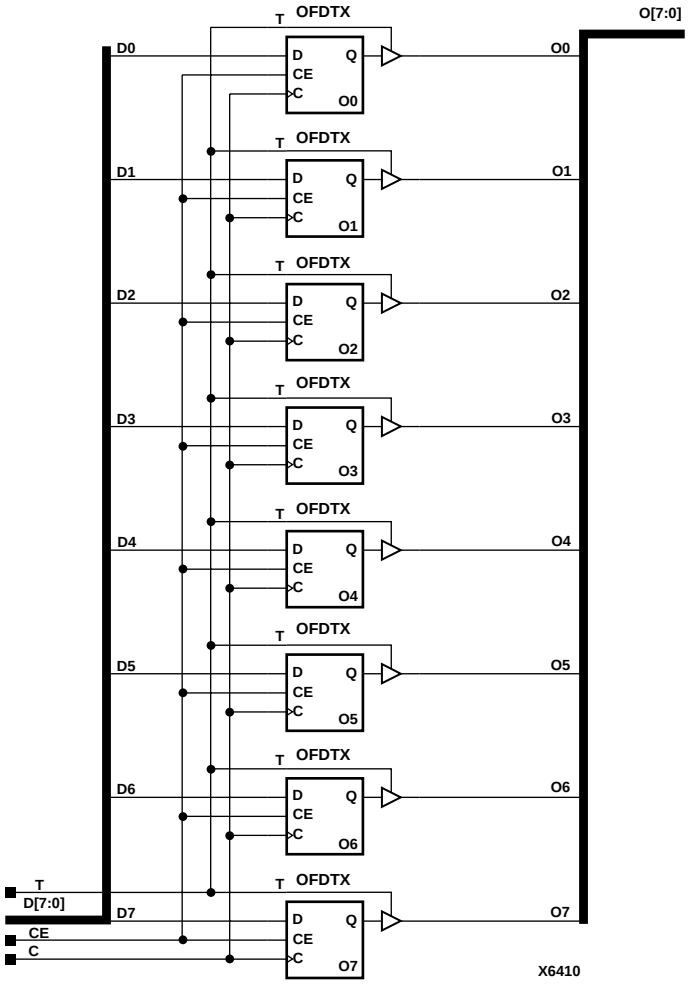
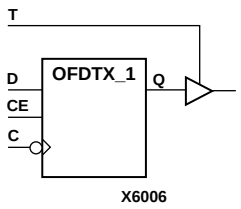


Figure 8-37 OFDTX8 Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDTX_1

D Flip-Flop with Active-Low 3-State Output Buffer, Inverted Clock, and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDTX_1 and its output buffer are located in an input / output block (IOB). The flip-flop data output (Q) is connected to the input of an output buffer (OBUFT). The OBUFT output is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the O output. When T is High, the output is high impedance (Off). When CE is High and T is Low, the outputs do not change.

The flip-flop is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	T	D	C	Q
X	1	X	X	Z
1	0	1	↓	0
1	0	0	↓	0
0	0	X	X	No Chg

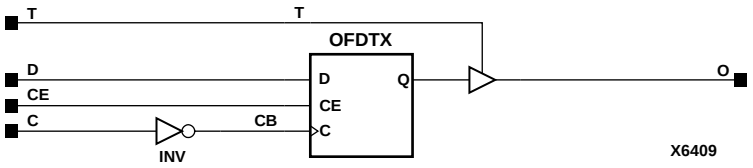
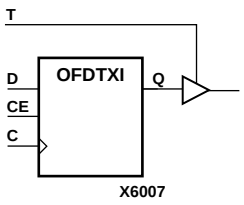


Figure 8-38 OFDTX_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

OFDTXI

D Flip-Flop with Active-Low 3-State Output Buffer and Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	N/A	N/A



OFDTXI and its output buffer are contained in an input/output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer (OBUFT). The output of the OBUFT is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the Low-to-High clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the output (O). When T is High, the output is high impedance (Off). When CE is Low and T is Low, the output does not change.

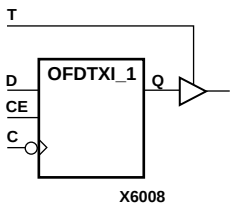
The flip-flop is asynchronously preset with High output when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	T	D	C	O
X	1	X	X	Z
1	0	1	↑	1
1	0	0	↑	0
0	0	X	X	No Chg

OFDTXI_1

D Flip-Flop with Active-Low 3-State Output Buffer, Inverted Clock, and Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	N / A	N / A



OFDTXI_1 and its output buffer are contained in an input / output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer (OBUFT). The OBUFT output is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the O output. When T is High, the output is high impedance (Off). When CE is Low and T is Low, the output does not change.

The flip-flop is asynchronously preset with High output when power is applied. FPGAs simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP symbol.

Inputs				Outputs
CE	T	D	C	Q
X	1	X	X	Z
1	0	1	↓	1
1	0	0	↓	0
0	0	X	X	No Chg

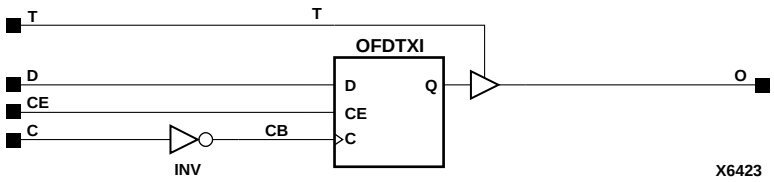
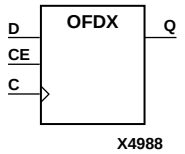


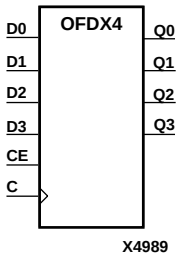
Figure 8-39 OFDTXI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

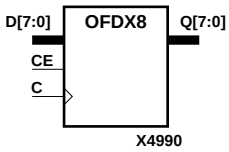
OFDX, 4, 8, 16

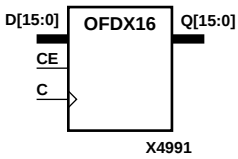
Single- and Multiple-Output D Flip-Flops with Clock Enable

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OFDX	N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Macro	Macro
OFDX4, OFDX8, OFDX16	N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro









OFDX, OFDX4, OFDX8, and OFDX16 are single and multiple output D flip-flops. The flip-flops are located in an input/output block (IOB) for XC4000E. The Q outputs are connected to OPADs or IOPADs. The data on the D inputs is loaded into the flip-flops during the Low-to-High clock (C) transition and appears on the Q outputs. When CE is Low, flip-flop outputs do not change.

The flip-flops are asynchronously cleared with Low outputs, when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↑	dn
0	X	X	No Chg

dn = state of referenced input one setup time prior to active clock transition

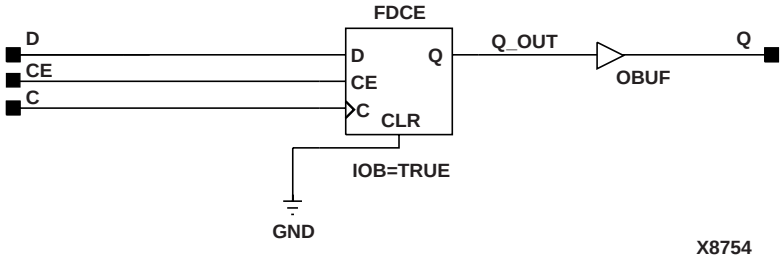


Figure 8-40 OFDX Implementation Spartan2, Virtex

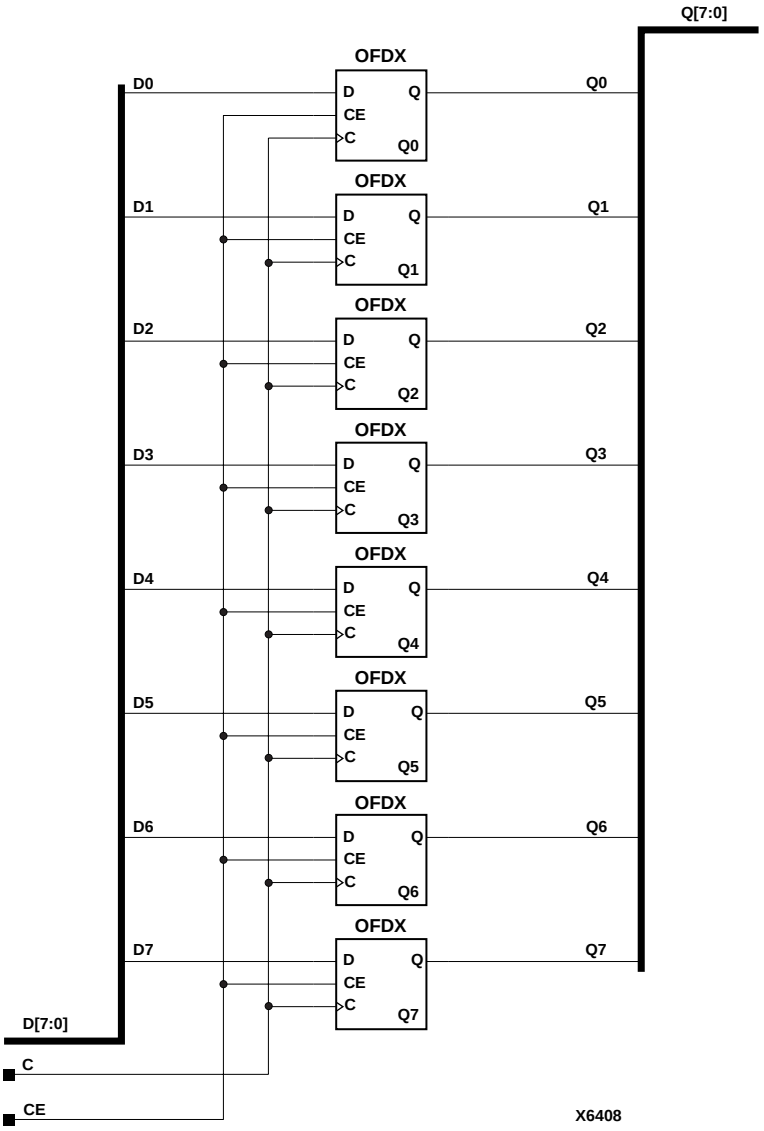
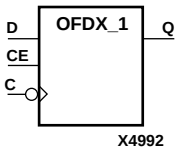


Figure 8-41 OFDX8 Implementation XC4000E, XC4000X, Spartan, SpartanXL, Spartan2, Virtex

OFDX_1

Output D Flip-Flop with Inverted Clock and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro



OFDX_1 is located in an input/output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously cleared with Low output when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition

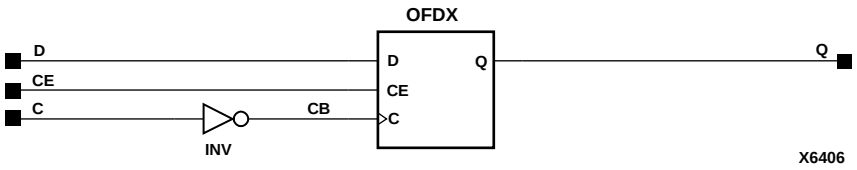


Figure 8-42 OFDX_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

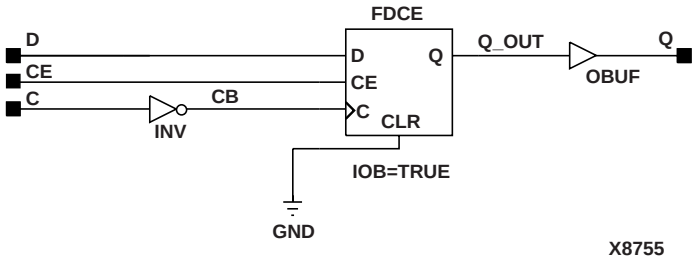
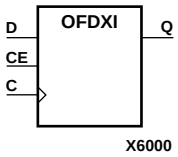


Figure 8-43 OFDX_1 Implementation Spartan2, Virtex

OFDXI

Output D Flip-Flop with Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Macro	Macro



OFDXI is contained in an input/output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). When CE is Low, the output does not change.

The flip-flop is asynchronously preset with High output when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↑	d
0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition

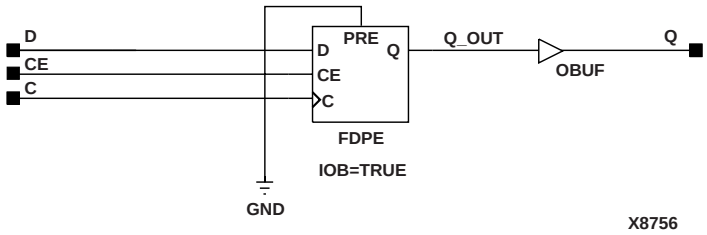
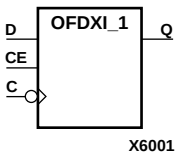


Figure 8-44 OFDXI Implementation Spartan2, Virtex

OFDXI_1

Output D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	N/A	N/A	Macro	Macro	Macro	Macro



OFDXI_1 is located in an input/output block (IOB). The D flip-flop output (Q) is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output. When CE is Low, the output (Q) does not change.

The flip-flop is asynchronously preset with High output when power is applied. FPGAs simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition

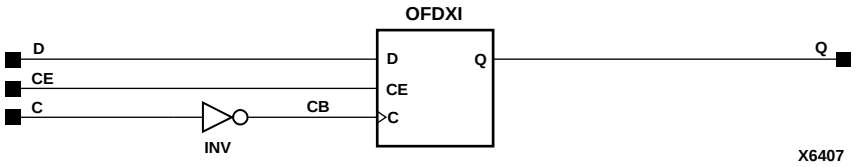


Figure 8-45 OFDXI_1 Implementation XC4000E, XC4000X, Spartan, SpartanXL

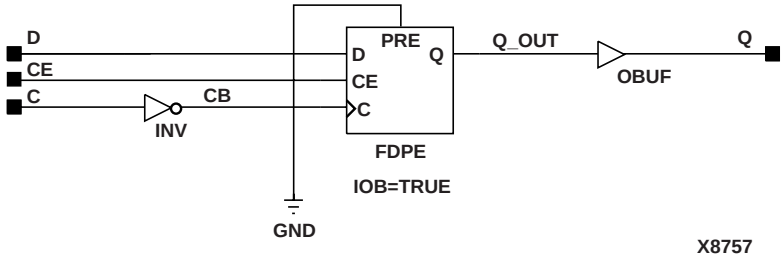
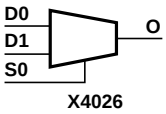


Figure 8-46 OFDXI_1 Implementation Spartan2, Virtex

OMUX2

2-to-1 Multiplexer

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



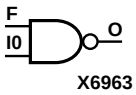
The OMUX2 multiplexer chooses one data bit from two sources (D1 or D0) under the control of the select input (S0). The output (O) reflects the state of the selected data input. When Low, S0 selects D0 and when High, S0 selects D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	1
0	X	0	0

ONAND2

2-Input NAND Gate with Invertible Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A

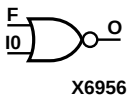


ONAND2 is a 2-input NAND gate that is implemented in the output multiplexer of the XC4000X IOB. The F pin is faster than I0. Input pins can be inverted even though there is no library component showing inverted inputs. The mapper will automatically bring any inverted input pins into the IOB.

ONOR2

2-Input NOR Gate with Invertible Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



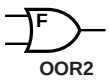
X6956

ONOR2 is a 2-input NOR gate that is implemented in the output multiplexer of the XC4000X IOB. The F pin is faster than I0. Input pins can be inverted even though there is no library component showing inverted inputs. The mapper will automatically bring any inverted input pins into the IOB.

OOR2

2-Input OR Gate with Invertible Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



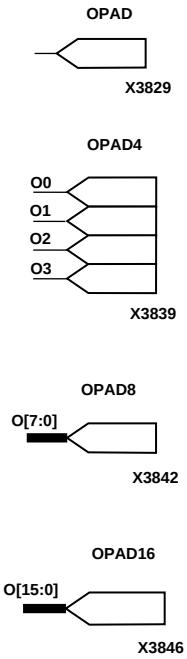
X8191

OOR2 is a 2-input OR gate that is implemented in the output multiplexer of the XC4000X IOB. The F pin is faster than I0. Input pins can be inverted even though there is no library component showing inverted inputs. The mapper will automatically bring any inverted input pins into the IOB.

OPAD, 4, 8, 16

Single- and Multiple-Output Pads

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OPAD	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OPAD4, OPAD8, OPAD16	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



OPAD, OPAD4, OPAD8, and OPAD16 are single and multiple output pads. An OPAD connects a device pin to an output signal of a PLD. It is internally connected to an input/output block (IOB), which is configured by the software as an OBUF, an OBUFT, an OBUFE, an OFD, or an OFDT.

Refer to the appropriate CAE tool interface user guide for details on assigning pin location and identification.

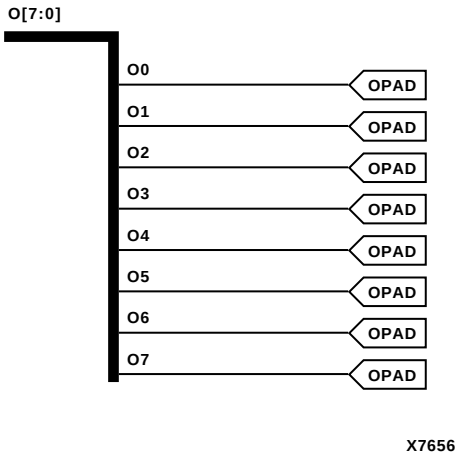


Figure 8-47 OPAD8 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

OR2-9

2- to 9-Input OR Gates with Inverted and Non-Inverted Inputs

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
OR2, OR2B1, OR2B2, OR3, OR3B1, OR3B2, OR3B3, OR4, OR4B1, OR4B2, OR4B3, OR4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5, OR5B1, OR5B2, OR5B3, OR5B4, OR5B5	Primitive	Primitive	Primitive	Macro	Primitive	Primitive	Primitive	Primitive	Primitive
OR6, OR7, OR8, OR9	Macro	Macro	Macro	Macro	Primitive	Macro	Macro	Macro	Macro

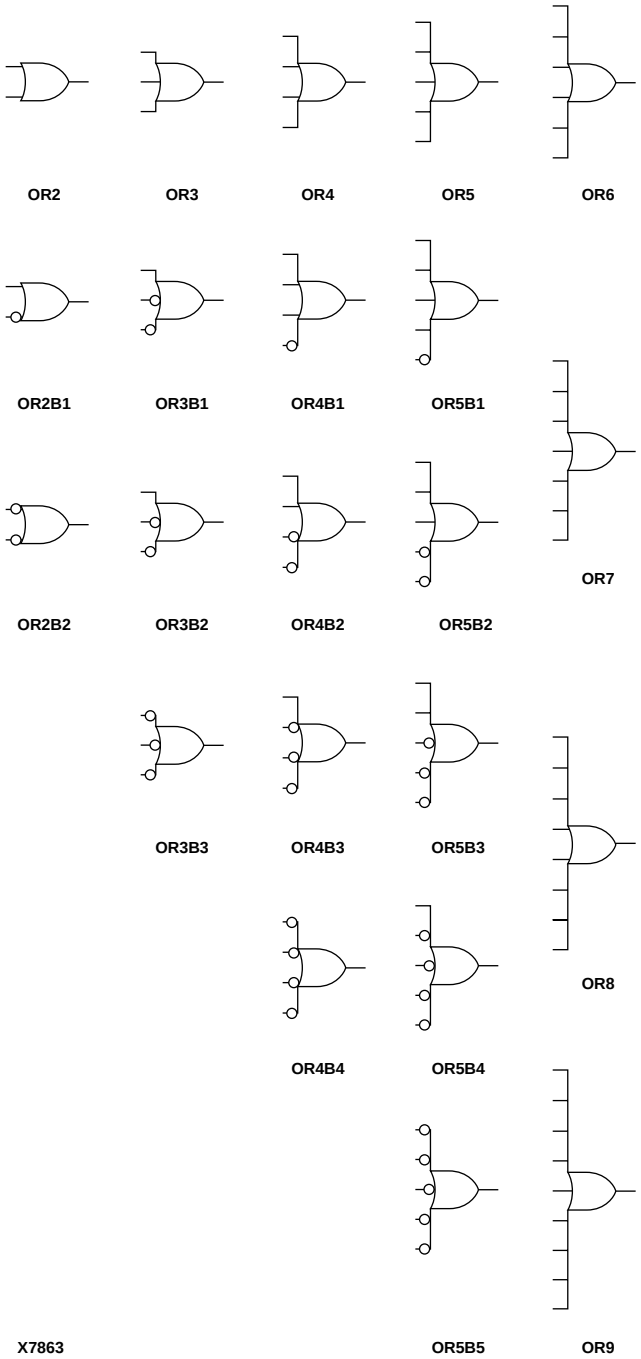


Figure 8-48 OR Gate Representations

The OR function is performed in the Configurable Logic Block (CLB) function generators for FPGAs. OR functions of up to five inputs are available in any combination of inverting and non-inverting inputs. OR functions of six to nine inputs are available with only non-inverting inputs. To invert some or all inputs, use external inverters. Since each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.

Refer to the “OR12, 16” section for information on additional OR functions for the XC5200, Spartan2, and Virtex.

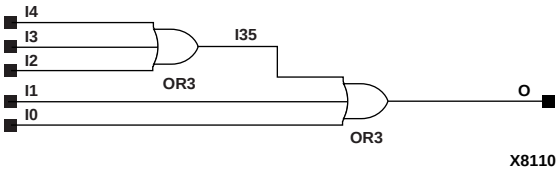


Figure 8-49 OR5 Implementation XC5200

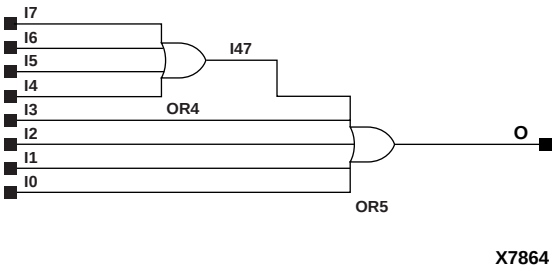


Figure 8-50 OR8 Implementation XC3000

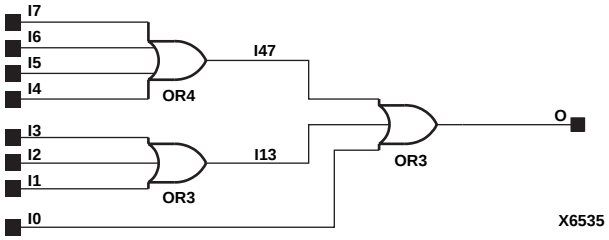


Figure 8-51 OR8 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

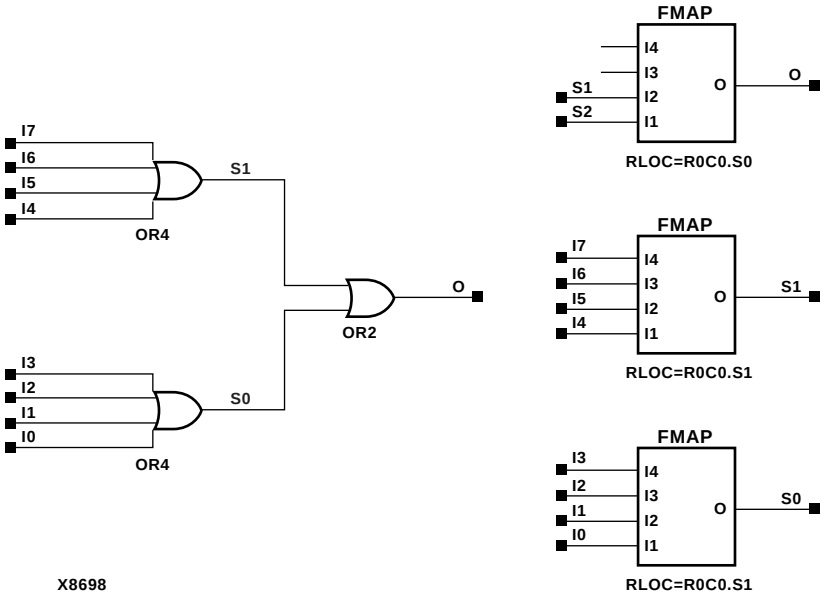


Figure 8-52 OR8 Implementation Spartan2, Virtex

OR12, 16

12- and 16-Input OR Gates with Non-Inverted Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	Macro	N / A	N / A	N / A	Macro	Macro

Refer to the “OR2-9” section for information on OR functions.

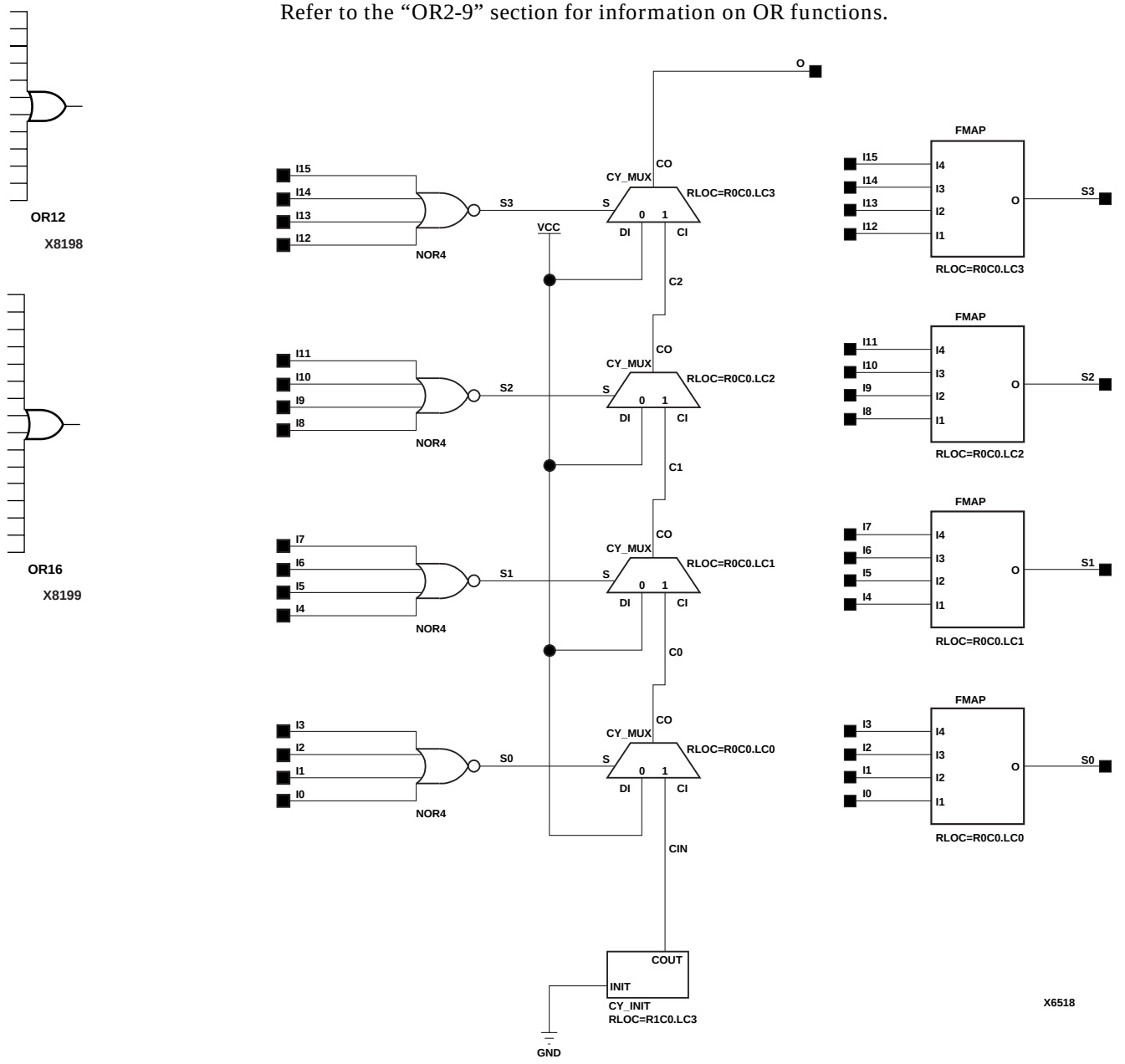


Figure 8-53 OR16 Implementation XC5200

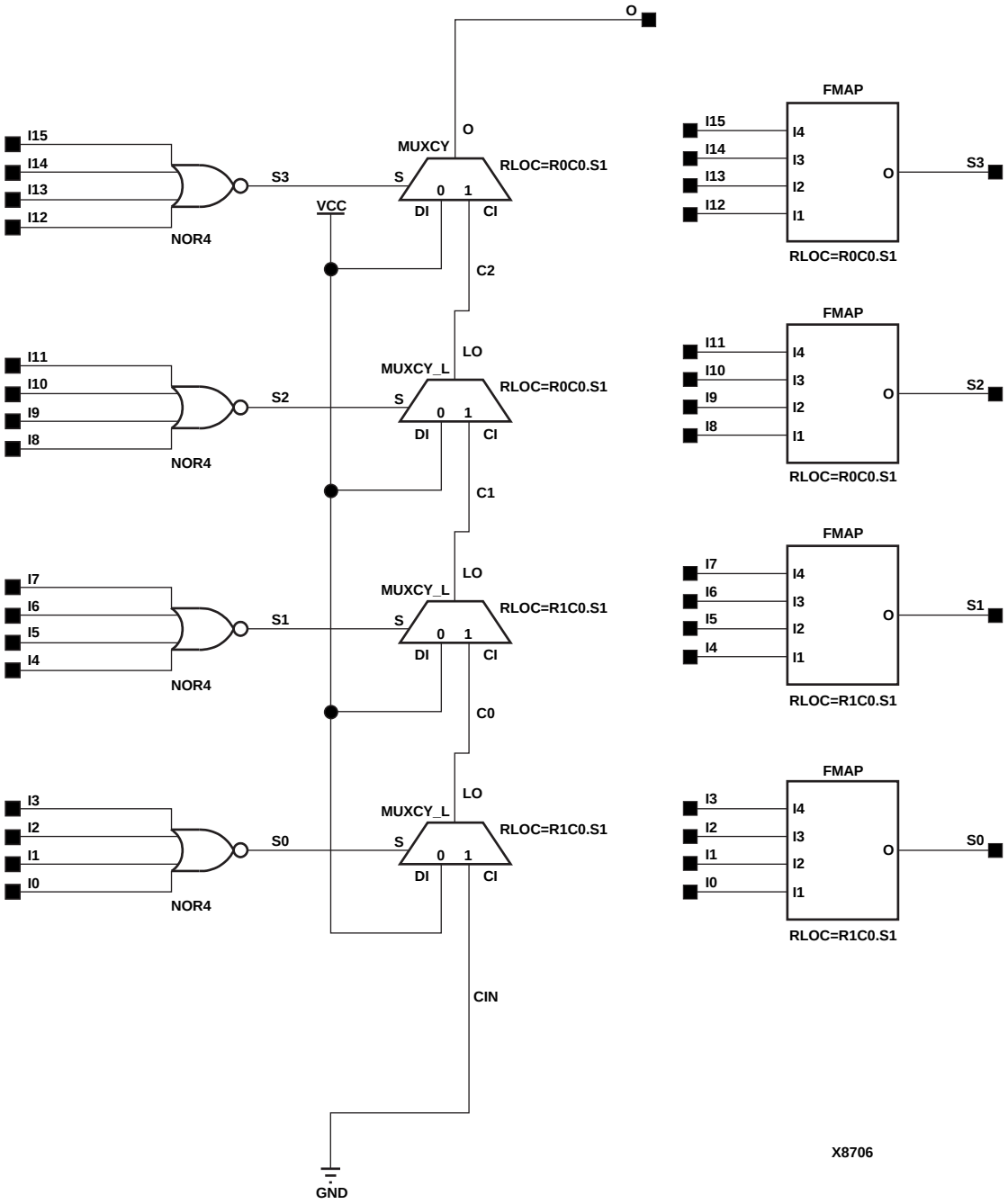
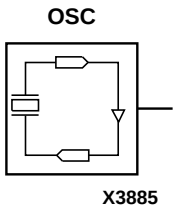


Figure 8-54 OR16 Implementation Spartan2, Virtex

OSC

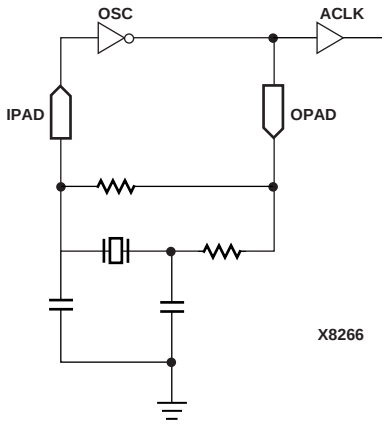
Crystal Oscillator Amplifier

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A



The OSC element's clock signal frequency is derived from an external crystal-controlled oscillator. The OSC output can be connected to an ACLK buffer, which is connected to an internal clock net.

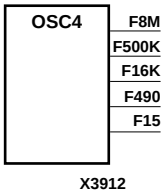
Two dedicated input pins (XTAL 1 and XTAL 2) on each FPGA device are internally connected to pads and input/output blocks that are connected to the OSC amplifier. The external components are connected as shown in the following example. Refer to *The Programmable Logic Data Book* for details on component selection and tolerances.



OSC4

Internal 5-Frequency Clock-Signal Generator

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	N/A	N/A

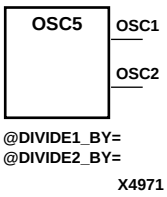


OSC4 provides internal clock signals in applications where timing is not critical. The available frequencies are determined by FPGA device components, which are process dependent. Therefore, the available frequencies vary from device to device. Nominal frequencies are 8 MHz, 500 kHz, 16 kHz, 490 Hz, and 15 Hz. Although there are five outputs, only three can be used at a time, with 8 MHz on one output and one frequency each on any two of the remaining four outputs. An error occurs if more than three outputs are used simultaneously.

OSC5

Internal Multiple-Frequency Clock-Signal Generator

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



OSC5 provides internal clock signals in applications where timing is not critical. The available frequencies are determined by FPGA device components that are process dependent. Therefore, the available frequencies vary from device to device. Use only one OSC5 per design. The OSC5 is not available if the CK_DIV element is used.

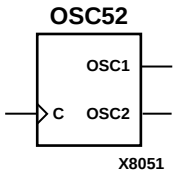
The clock frequencies of the OSC1 and OSC2 outputs are determined by specifying the DIVIDE1_BY=n_1 attribute for the OSC1 output and the DIVIDE2_BY=n_2 attribute for the OSC2 output. n_1 and n_2 are integer numbers by which the internal 16-MHz clock is divided to produce the desired clock frequency. The available frequency options are shown in the table.

n_1	OSC1 Frequency	n_2	OSC2 Frequency
4	4 MHz	2	8 MHz
16	1 MHz	8	2 MHz
64	250 kHz	32	500 kHz
256	63 kHz	128	125 kHz
		1,024	16 kHz
		4,096	4 kHz
		16,384	1 kHz
		65,536	244 Hz

OSC52

Internal Multiple-Frequency Clock-Signal Generator

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	Primitive	N/A	N/A	N/A	N/A	N/A



OSC52 provides internal clock signals in applications where timing is not critical. The available frequencies are determined by FPGA device components, which are process independent. Therefore, the available frequencies vary from device to device. Only one OSC52 may be used per design.

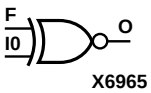
The oscillator frequencies of the OSC1 and OSC2 outputs are determined by specifying theDIVIDE1_BY=n_1 attribute for the OSC1 output and DIVIDE2_BY=n_2 attribute for the OSC2 output. n_1 and n_2 are integer numbers by which internal 16-MHz clock is divided to produce the desired clock frequency. The available frequency options appear in the table that follows.

n_1	OSC1 Frequency	n_2	OSC2 Frequency
4	4 MHz	2	8 MHz
16	1 MHz	8	2 MHz
64	250 kHz	32	500 kHz
256	63 kHz	128	125 kHz
		1,024	16 kHz
		4,096	4 kHz
		16,384	1 kHz
		65,536	244 Hz

OXNOR2

2-Input Exclusive-NOR Gate with Invertible Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A

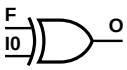


OXNOR2 is a 2-input exclusive NOR gate that is implemented in the output multiplexer of the XC4000X and SpartanXL IOB. The F pin is faster than I0. Input pins can be inverted even though there is no library component showing inverted inputs. The mapper will automatically bring any inverted input pins into the IOB.

OXOR2

2-Input Exclusive-OR Gate with Invertible Inputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	Primitive	N/A	N/A	N/A	Primitive	N/A	N/A



X6964

OXOR2 is a 2-input exclusive OR gate that is implemented in the output multiplexer of the XC4000X IOB. The F pin is faster than I0. Input pins can be inverted even though there is no library component showing inverted inputs. The mapper will automatically bring any inverted input pins into the IOB.

Chapter 9

Design Elements (PULLDOWN to ROM32X1)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

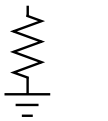
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

PULLDOWN

Resistor to GND for Input Pads

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	Primitive	Primitive



X3860

PULLDOWN resistor elements are available in each XC4000, Spartan, or SpartanXL Input/Output Block (IOB). They are connected to input, output, or bidirectional pads to guarantee a logic Low level for nodes that might float.

PULLUP

Resistor to VCC for Input PADs, Open-Drain, and 3-State Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	Primitive	Primitive

X3861

PULLUP resistor elements are available in each XC3000, XC4000, Spartan, and SpartanXL Input/Output Block (IOB). XC3000 IOBs only use PULLUP resistors on input pads. XC4000, Spartan, and SpartanXL IOBs connect PULLUP resistors to input, output, or bidirectional pads to guarantee a logic High level for nodes that might float.

The pull-up elements also establish a High logic level for open-drain elements and macros (DECODE, WAND, WORAND) or 3-state nodes (TBUF) when all the drivers are off.

The buffer outputs are connected together as a wired-AND to form the output (O). When all the inputs are High, the output is off. To establish an output High level, a PULLUP resistor(s) is tied to output (O). One PULLUP resistor uses the least power, two pull-up resistors achieve the fastest Low-to-High speed.

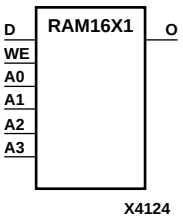
To indicate two PULLUP resistors, append a DOUBLE parameter to the pull-up symbol attached to the output (O) node. Refer to the appropriate CAE tool interface user guide for details.

The PULLUP element is ignored in XC9000 designs. Internal 3-state nodes (from BUFE or BUFT) in CPLD designs are always pulled up when not driven.

RAM16X1

16-Deep by 1-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A	N/A



RAM16X1 is a 16-word by 1-bit static read-write random access memory. When the write enable (WE) is High, the data on the data input (D) is loaded into the word selected by the 4-bit address (A3 – A0). The data output (O) reflects the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or input data transitions. Address inputs must be stable before the High-to-Low WE transition for predictable performance.

You can initialize RAM16X1 during configuration. See "Specifying Initial Contents of a RAM" in this section.

Mode selection is shown in the following truth table.

Inputs		Outputs
WE(mode)	D	O
0(read)	X	Data
1(write)	D	Data

Data = word addressed by bits A3 – A0

Specifying Initial Contents of a RAM

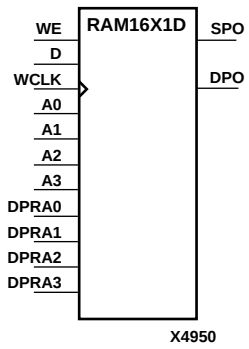
You can use the INIT attribute to specify an initial value directly on the symbol only if the RAM is 1 bit wide and 16 or 32 bits deep. The value must be a hexadecimal number, for example, INIT=ABAC.

If the INIT attribute is not specified, the RAM is initialized with zero.

RAM16X1D

16-Deep by 1-Wide Static Dual Port Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Primitive	Primitive



RAM16X1D is a 16-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM16X1D during configuration. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0
data_d = word addressed by bits DPRA3-DPRA0

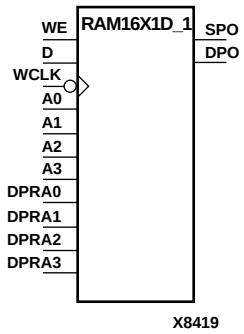
The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

RAM16X1D_1

16-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



RAM16X1D_1 is a 16-word by 1-bit static dual port random access memory with synchronous write capability and negative-edge clock. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM16X1D_1 during configuration. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A3-A0
data_d = word addressed by bits DPRA3-DPRA0

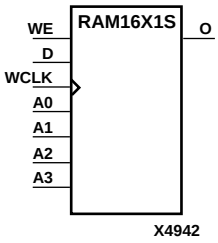
The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

RAM16X1S

16-Deep by 1-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Primitive	Primitive



RAM16X1S is a 16-word by 1-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM16X1S during configuration. See "Specifying Initial Contents of a RAM" in the "RAM16X1" section.

Mode selection is shown in the following truth table.

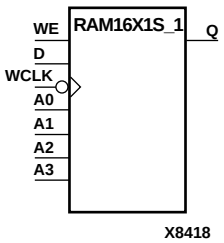
Inputs			Outputs
WE(mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	0	D
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

RAM16X1S_1

16-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	N / A	N / A	N / A	Primitive	Primitive



RAM16X1S_1 is a 16-word by 1-bit static random access memory with synchronous write capability and negative-edge clock. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM16X1S_1 during configuration. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

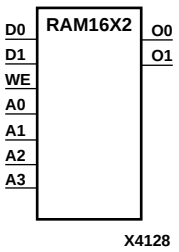
Inputs			Outputs
WE(mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	0	D
1 (read)	↑	X	Data

Data = word addressed by bits A3 – A0

RAM16X2

16-Deep by 2-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



RAM16X2 is a 16-word by 2-bit static read-write random access memory. When the write enable (WE) is High, the data on data inputs (D1 – D0) is loaded into the word selected by the 4-bit address (A3 – A0). The data outputs (O1 – O0) reflect the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or data input transitions. Address inputs must be stable before the High-to-Low WE transition for predictable performance.

The initial contents of RAM16X2 cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See "Specifying Initial Contents of a RAM" in the "RAM16X1" section.

Mode selection is shown in the following truth table.

Inputs		Outputs
WE (mode)	**D1 – D0**	**O1 – O0**
0 (read)	X	Data
1 (write)	D1 – D0	Data

Data = word addressed by bits A3 – A0

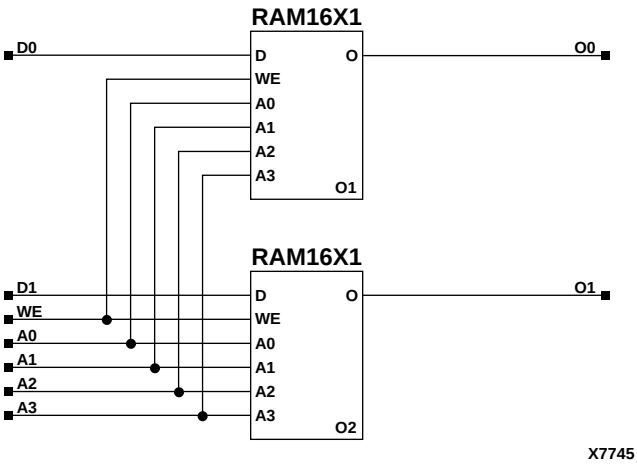
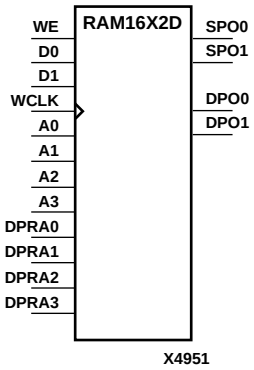


Figure 9-1 RAM16X2 Implementation XC4000E, XC4000X

RAM16X2D

16-Deep by 2-Wide Static Dual Port Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM16X2D is a 16-word by 2-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO1 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X2D cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D1-D0	SPO1-SPO0	DPO1-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D1-D0	D1-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0
data_d = word addressed by bits DPRA3-DPRA0

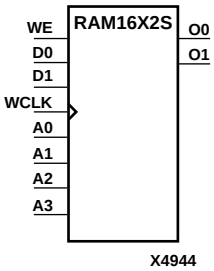
The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

RAM16X2S

16-Deep by 2-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM16X2S is a 16-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

The initial contents of RAM16X2S cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

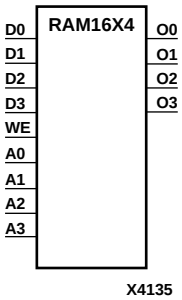
Inputs			Outputs
WE (mode)	**WCLK**	**D1-D0**	**O1-O0**
0 (read)	X	X	Data
1(read)	0	X	Data
1(read)	1	X	Data
1(write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

RAM16X4

16-Deep by 4-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



RAM16X4 is a 16-word by 4-bit static read-write random access memory. When the write enable (WE) is High, the data on data inputs (D3 – D0) is loaded into the word selected by the 4-bit address (A3 – A0). The data outputs (O3 – O0) reflect the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or data input transitions. Address inputs must be stable before the High-to-Low WE transition for predictable performance.

The initial contents of RAM16X4 cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs		Outputs
WE(mode)	D3 – D0	O3 – O0
0(read)	X	data
1(write)	D3 – D0	Data

Data = word addressed by bits A3 – A0

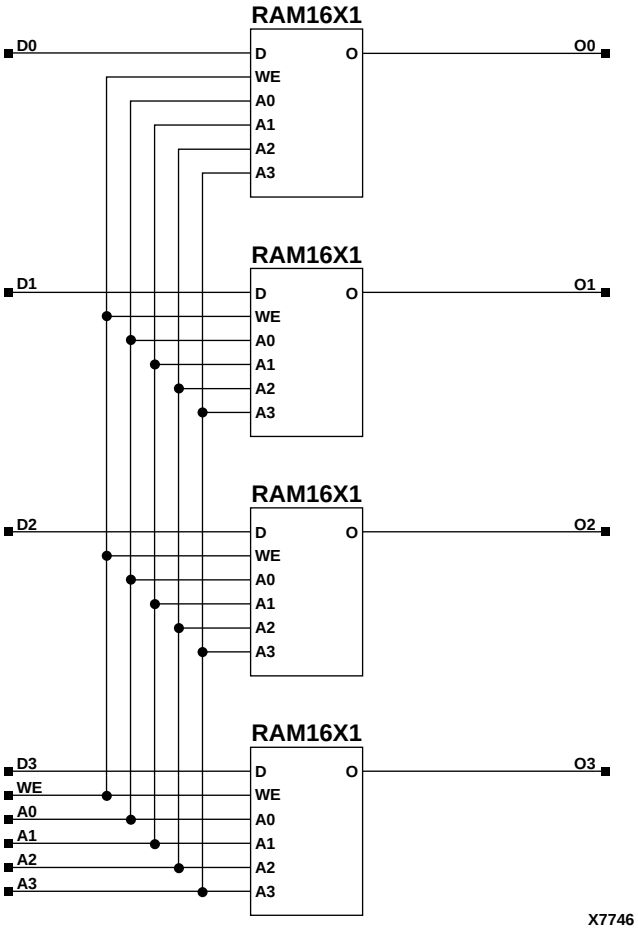
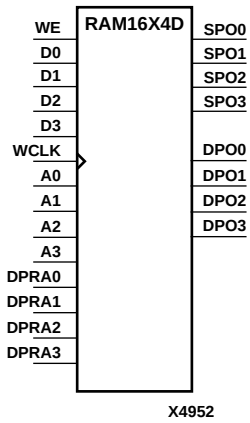


Figure 9-2 RAM16X4 Implementation XC4000E, XC4000X

RAM16X4D

16-Deep by 4-Wide Static Dual Port Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM16X4D is a 16-word by 4-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO3 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D3 – D0) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X4D cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D3-D0	SPO3-SPO0	DPO3-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D3-D0	D3-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0
data_d = word addressed by bits DPRA3-DPRA0

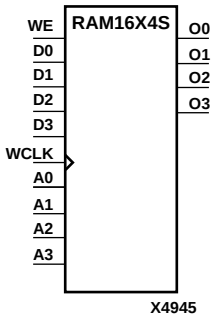
The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

RAM16X4S

16-Deep by 4-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM16X4S is a 16-word by 4-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D3 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O3 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

The initial contents of RAM16X4S cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

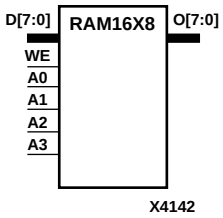
Inputs			Outputs
WE (mode)	**WCLK**	**D3 – D0**	**O3 – O0**
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3-D0	D3-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

RAM16X8

16-Deep by 8-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



RAM16X8 is a 16-word by 8-bit static read-write random access memory. When the write enable (WE) is High, the data on data inputs (D7 – D0) is loaded into the word selected by the 4-bit address (A3 – A0). The data outputs (O7 – O0) reflect the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or data input transitions. Address inputs must be stable before the High-to-Low WE transition for predictable performance.

The initial contents of RAM16X8 cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs		Outputs
WE(mode)	D7 – D0	O7 – O0
0(read)	X	Data
1(write)	D7 – D0	Data

Data = word addressed by bits A3 – A0

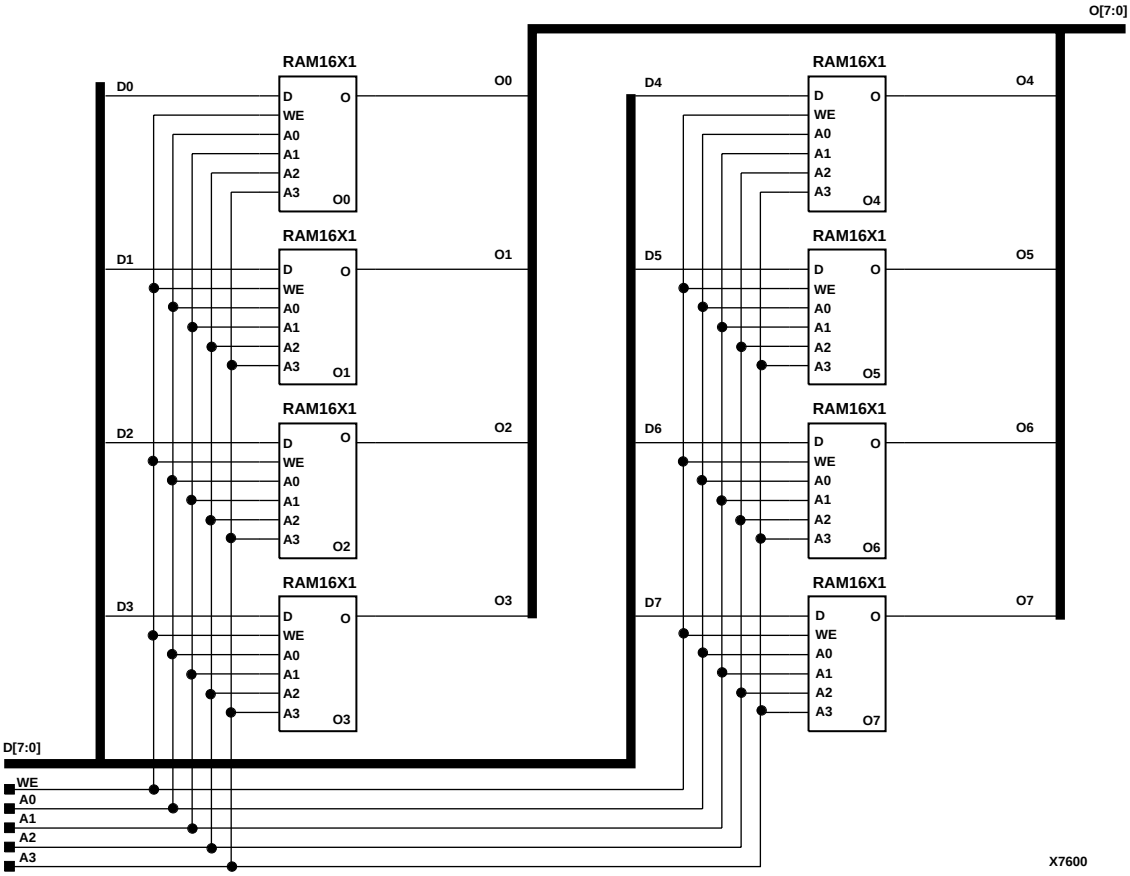
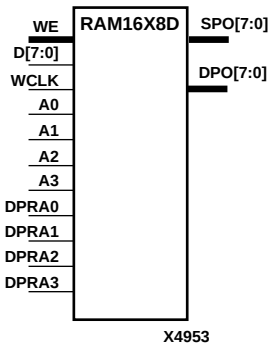


Figure 9-3 RAM16X8 Implementation XC4000E, XC4000X

RAM16X8D

16-Deep by 8-Wide Static Dual Port Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM16X8D is a 16-word by 8-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO7 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D7 – D0) into the word selected by the 4-bit write address (A3 – A0). For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X8D cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D7-D0	SP7-SPO0	DPO7-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D7-D0	D7-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0
data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

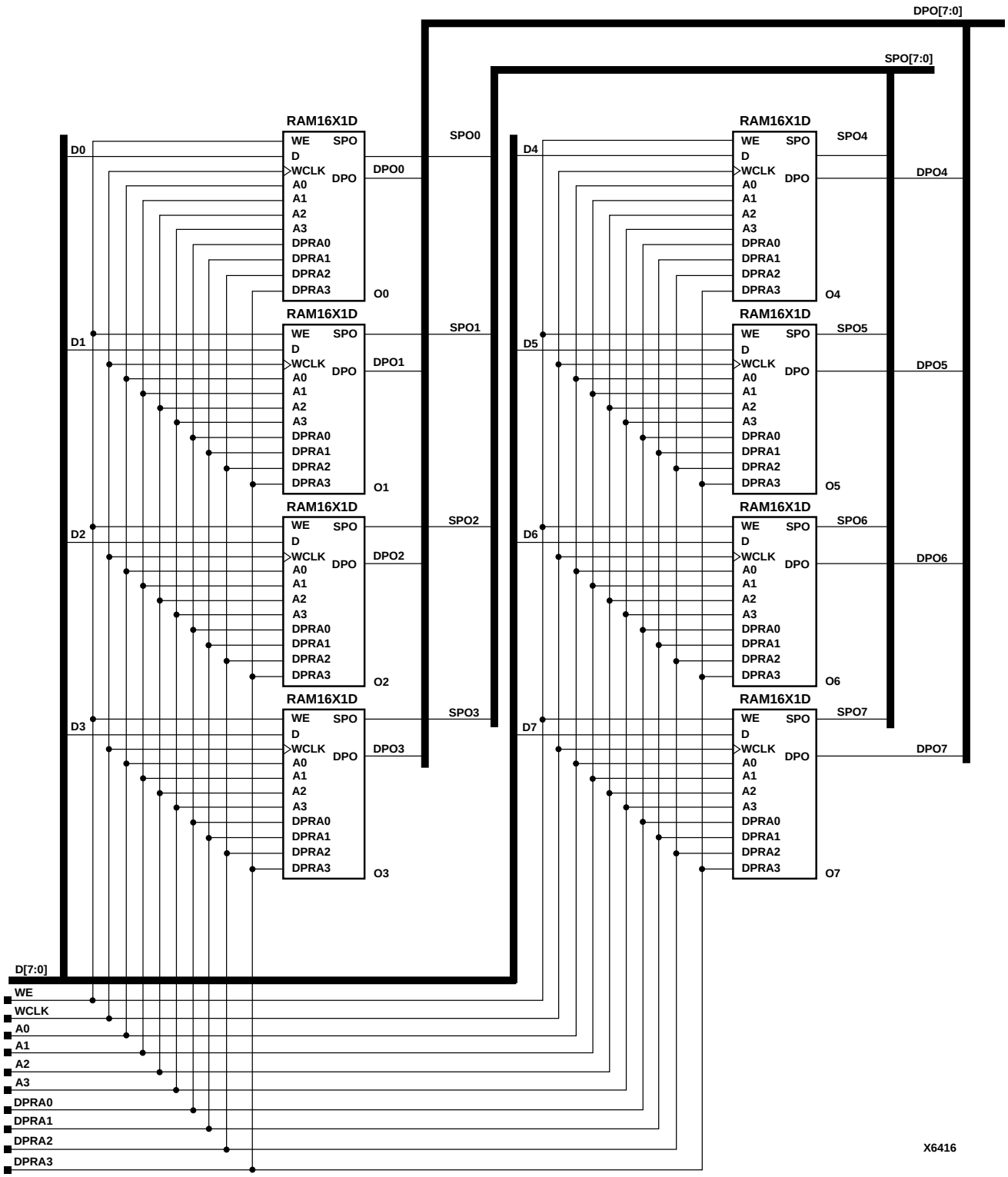
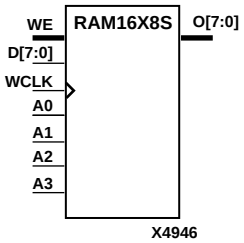


Figure 9-4 RAM16X8D Implementation XC4000E, XC4000X, Spartan, SpartanXL

RAM16X8S

16-Deep by 8-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM16X8S is a 16-word by 8-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on data inputs (D7 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O7 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

The initial contents of RAM16X8S cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D7-D0	O7-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D7-D0	D7-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

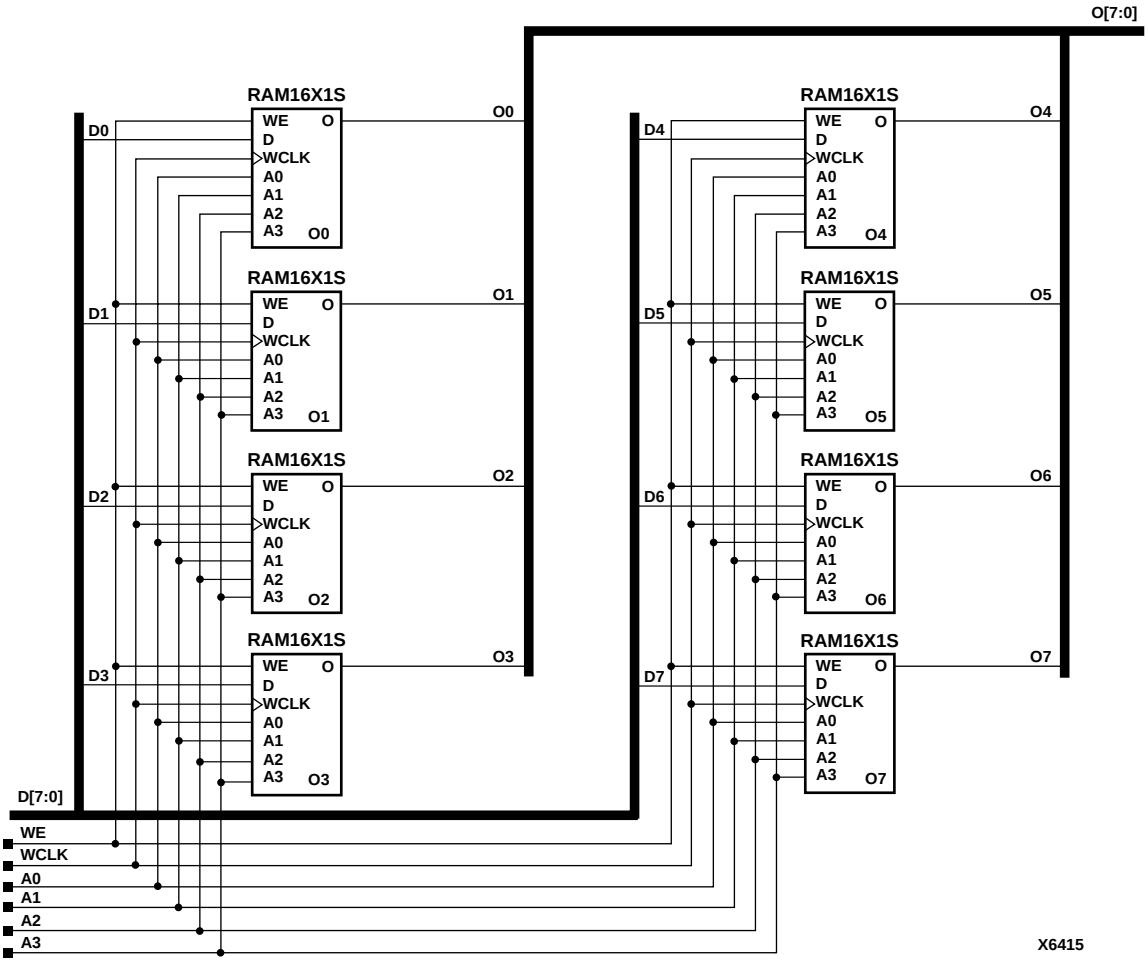
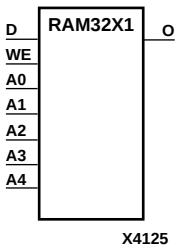


Figure 9-5 RAM16X8S Implementation XC4000E, XC4000X, Spartan, SpartanXL

RAM32X1

32-Deep by 1-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A	N/A



RAM32X1 is a 32-word by 1-bit static read-write random access memory. When the write enable (WE) is High, the data on the data input (D) is loaded into the word selected by the 5-bit address (A4 – A0). The data output (O) reflects the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or input data transitions. Address inputs must be stable before the High-to-Low WE transition for predictable performance.

You can initialize RAM32X1 during configuration. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

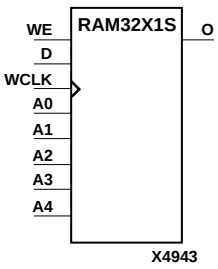
Inputs		Outputs
WE(mode)	D	O
0(read)	X	Data
1(write)	D	Data

Data = word addressed by bits A4 – A0

RAM32X1S

32-Deep by 1-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	Primitive	Primitive	Primitive	Primitive



RAM32X1S is a 32-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Specifying Initial Contents of a RAM32X1S

You can initialize RAM32X1S during configuration using the INIT attribute. The value must be a hexadecimal number, for example, INIT=ABAC. If the INIT attribute is not specified, the RAM is initialized with zero.

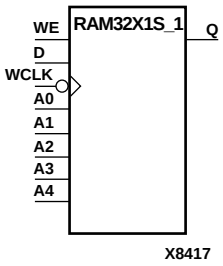
For XC4000E, XC4000X, Spartan, and SpartanXL, lower INIT values get mapped to the F function generator and upper INIT values get mapped to the G function generator.

For Virtex and Spartan2, lower INIT values get mapped to the G function generator and upper INIT values get mapped to the F function generator.

RAM32X1S_1

32-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



RAM32X1S_1 is a 32-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S_1 during configuration. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A4 – A0

Specifying Initial Contents of a RAM32X1S_1

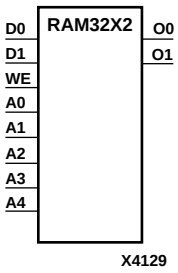
You can initialize RAM32X1S_1 during configuration using the INIT attribute. The value must be a hexadecimal number, for example, INIT=ABAC. If the INIT attribute is not specified, the RAM is initialized with zero.

For Virtex and Spartan2, lower INIT values get mapped to the G function generator and upper INIT values get mapped to the F function generator.

RAM32X2

32-Deep by 2-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



RAM32X2 is a 32-word by 2-bit static read-write random access memory. When the write enable (WE) is High, the data on the data inputs (D1 – D0) is loaded into the word selected by the address bits (A4 – A0). The data outputs (O1 – O0) reflect the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or input data transitions. Address inputs must be stable before the High-to- Low WE transition for predictable performance.

The initial contents of RAM32X2 cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

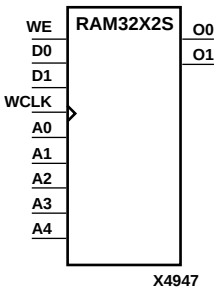
Inputs		Outputs
WE(mode)	**D1 – D0**	**O1 – O0**
0(read)	X	Data
1(write)	D1 – D0	Data

Data = word addressed by bits A4 – A0

RAM32X2S

32-Deep by 2-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM32X2S is a 32-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

The initial contents of RAM32X2S cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

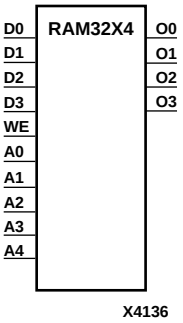
Inputs			Outputs
WE (mode)	WCLK	D0-D1	O0-O1
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

RAM32X4

32-Deep by 4-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



RAM32X4 is a 32-word by 4-bit static read-write random access memory. When the write enable (WE) is High, the data on the data inputs (D3 – D0) is loaded into the word selected by the address bits (A4 – A0). The data outputs (O3 – O0) reflect the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or input data transitions. Address inputs must be stable before the High-to- Low WE transition for predictable performance.

The initial contents of RAM32X4 cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

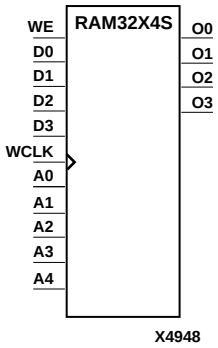
Inputs		Outputs
WE(mode)	**D3 – D0**	**O3 – O0**
0(read)	X	Data
1(write)	D3 – D0	Data

Data = word addressed by bits A4 – A0

RAM32X4S

32-Deep by 4-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM32X4S is a 32-word by 4-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data inputs (D3 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O3 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

The initial contents of RAM32X4S cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

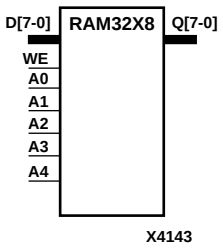
Inputs			Outputs
WE	**WCLK**	**D3-D0**	**O3-O0**
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3-D0	D3-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

RAM32X8

32-Deep by 8-Wide Static RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	N / A	N / A	N / A	N / A



RAM32X8 is a 32-word by 8-bit static read-write random access memory. When the write enable (WE) is High, the data on the data inputs (D7 – D0) is loaded into the word selected by the address bits (A4 – A0). The data outputs (O7 – O0) reflect the selected (addressed) word, whether WE is High or Low. When WE is Low, the RAM content is unaffected by address or input data transitions. The address inputs must be stable before the High-to- Low WE transition for predictable performance.

The initial contents of RAM32X8 cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs		Outputs
WE(mode)	**D7 – D0**	**O7 – O0**
0(read)	X	Data
1(write)	D7 – D0	Data

Data = word addressed by bits A4 – A0

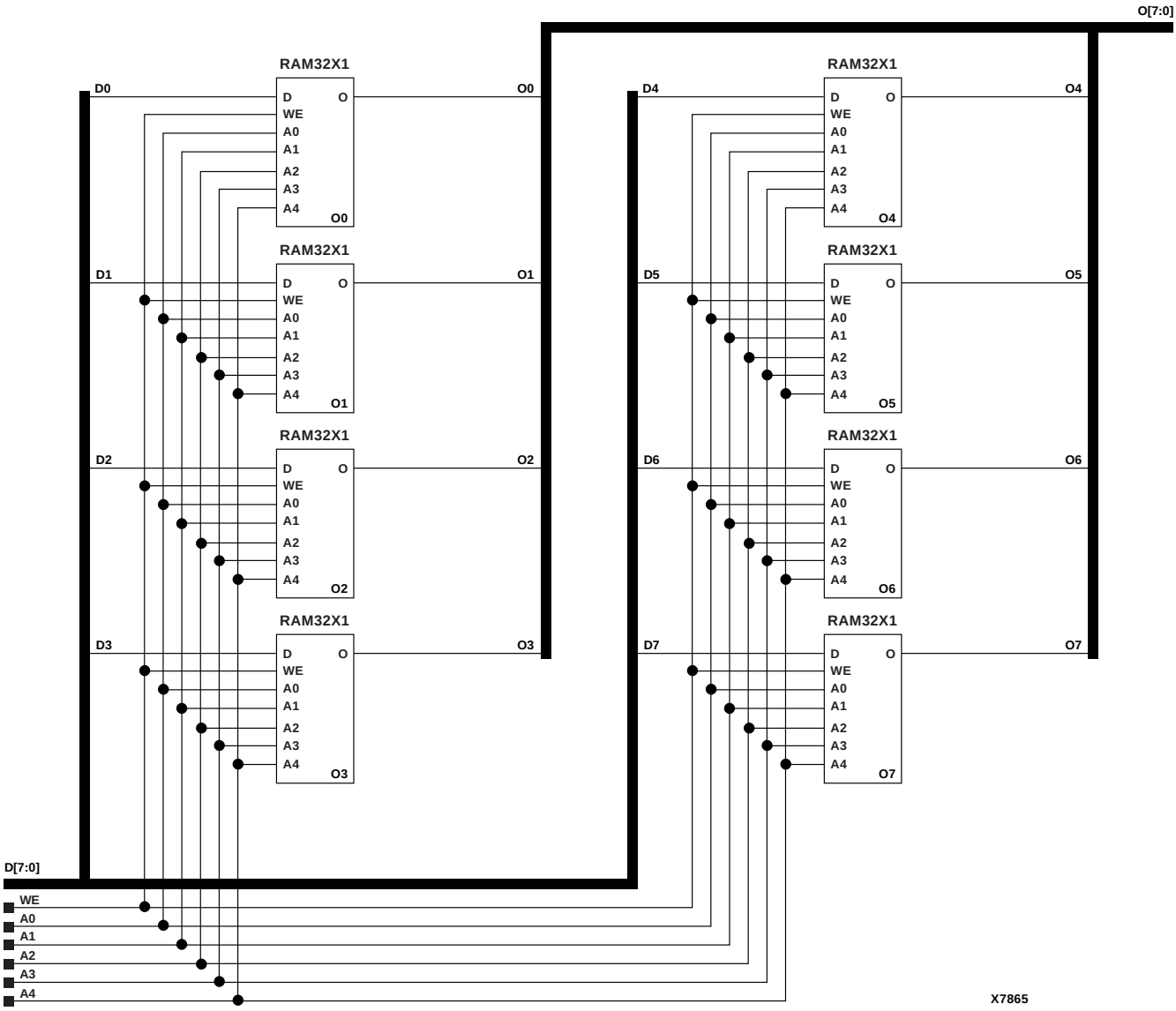
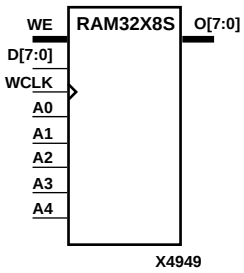


Figure 9-6 RAM32X8 Implementation XC4000E, XC4000X

RAM32X8S

32-Deep by 8-Wide Static Synchronous RAM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Macro	Macro	N / A	N / A	Macro	Macro	Macro	Macro



RAM32X8S is a 32-word by 8-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data inputs (D7 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O7 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

The initial contents of RAM32X8S cannot be specified directly. Initial contents may be specified only for RAMs that are 1-bit wide and 16 or 32 bits deep. See “Specifying Initial Contents of a RAM” in the “RAM16X1” section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	**WCLK**	**D7-D0**	**O7-O0**
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D7-D0	D7-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

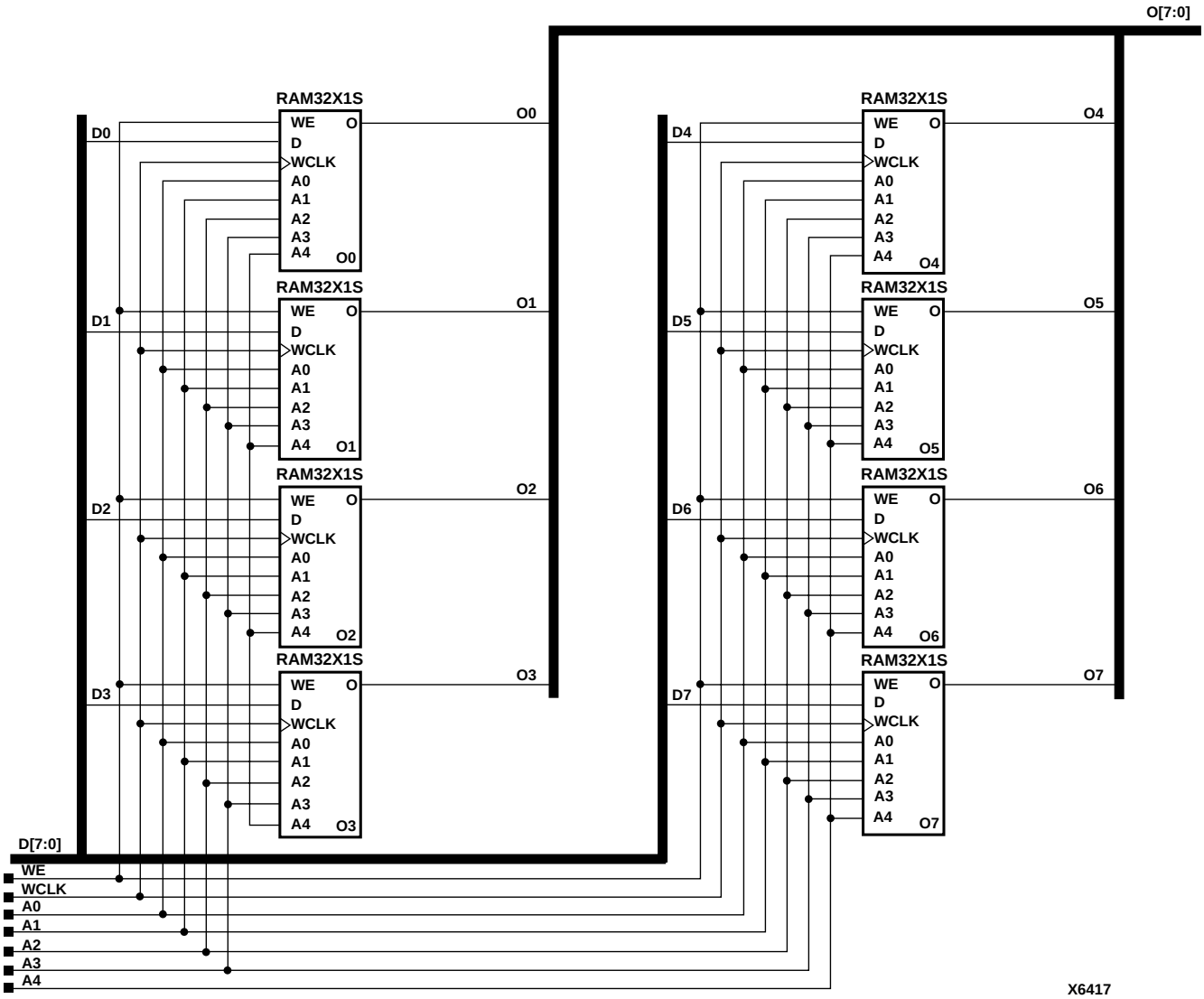
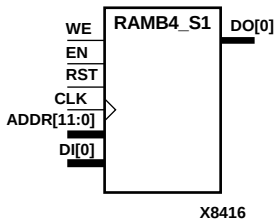


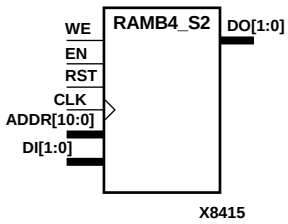
Figure 9-7 RAM32X8S Implementation XC4000E, XC4000X, Spartan, Spartan2, SpartanXL, Virtex

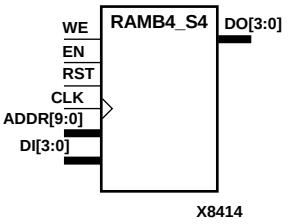
RAMB4_S*n*

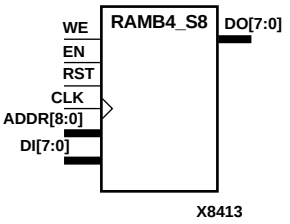
4096-Bit Single-Port Synchronous Block RAM with Port Width (*n*) Configured to 1, 2, 4, 8, or 16 Bits

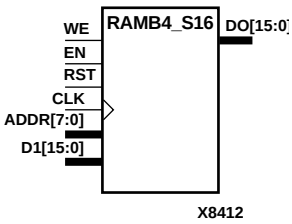
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive











RAMB4_S1, RAMB4_S2, RAMB4_S4, RAMB4_S8, and RAMB4_S16 are dedicated random access memory blocks with synchronous write capability. They provide the capability for fast, discrete, large blocks of RAM in each Virtex and Spartan2 device.The RAMB4_Sn cell configurations are listed in the following table.

Component	Depth	Width	Address Bus	Data Bus
RAMB4_S1	4096	1	(11:0)	(0:0)
RAMB4_S2	2048	2	(10:0)	(1:0)
RAMB4_S4	1024	4	(9:0)	(3:0)
RAMB4_S8	512	8	(8:0)	(7:0)
RAMB4_S16	256	16	(7:0)	(15:0)

The enable (EN) pin controls read, write, and reset. When EN is Low, no data is written and the output (DO) retains the last state. When EN is High and reset (RST) is High, DO is cleared during the Low-to-High clock (CLK) transition; if write enable (WE) is High, the memory contents reflect the data at DI. When EN is High and WE is Low, the data stored in the RAM address (ADDR) is read during the Low-to-High clock transition. When EN and WE are High, the data on the data input (DI) is loaded into the word selected by the write address (ADDR) during the Low-to-High clock transition and the data output (DO) reflects the selected (addressed) word.

The above description assumes an active High EN, WE, RST, and CLK. However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB4 port is absorbed into the block and does not use a CLB resource.

RAMB4_Sn's may be initialized during configuration. See the "Specifying Initial Contents of a Block RAM" section below.

Block RAM output registers are asynchronously cleared, output Low, when power is applied. The initial contents of the block RAM are not altered. Virtex and Spartan2 simulate power-on when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Mode selection is shown in the following truth table.

Inputs						Outputs	
EN	RST	WE	CLK	ADDR	DI	DO	RAM Contents
0	X	X	X	X	X	No Chg	No Chg
1	1	0	↑	X	X	0	No Chg
1	1	1	↑	addr	data	0	RAM(addr) <=data
1	0	0	↑	addr	X	RAM(addr)	No Chg
1	0	1	↑	addr	data	data	RAM(addr) <=data

addr=RAM address
RAM(addr)=RAM contents at address ADDR
data=RAM input data

Specifying Initial Contents of a Block RAM

You can use the INIT_0x attributes to specify an initial value during device configuration. The initialization of each RAMB4_Sn is set by 16 initialization attributes (INIT_00 through INIT_0F) of 64 hex values for a total of 4096 bits. See the “INIT_0x” section of the “Attributes, Constraints, and Carry Logic” chapter for more information on these attributes.

If any INIT_0x attribute is not specified, it is configured as zeros. Partial initialization strings are padded with zeros to the left.

RAMB4_S*n*_S*n*

4096-Bit Dual-Port Synchronous Block RAM with Port Width (*n*) Configured to 1, 2, 4, 8, or 16 Bits

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive

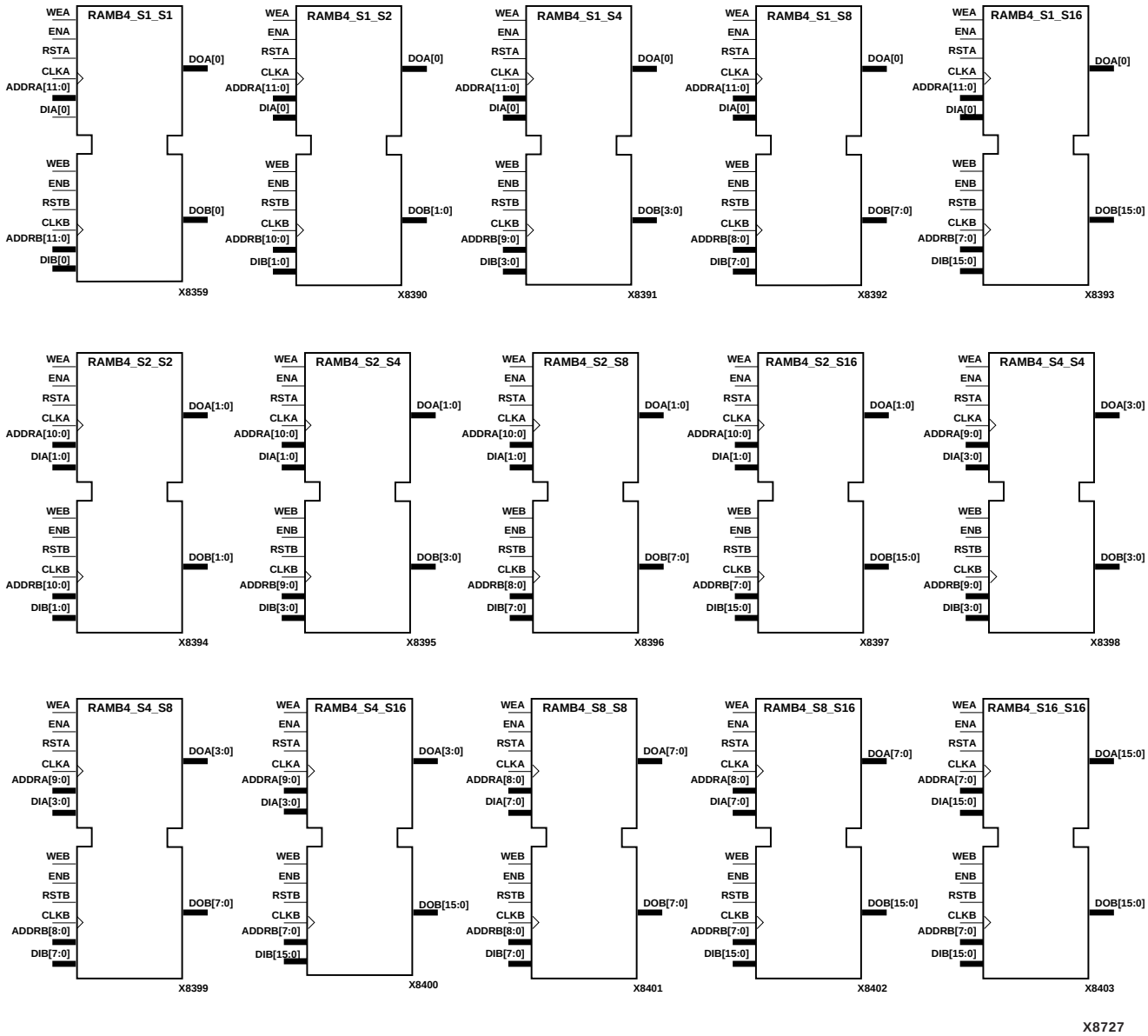


Figure 9-8 RAMB4_Sn_Sn Representations

The RAMB4_Sn_Sn components listed in the following table are 4096-bit dual-ported dedicated random access memory blocks with synchronous write capability. Each port is independent of the other while accessing the same set of 4096 memory cells. Each port is independently configured to a specific data width.

Component	Port A Depth	Port A Width	Port A ADDR	Port A DI	Port B Depth	Port B Width	Port B ADDR	Port B DI
RAMB4_S1_S1	4096	1	(11:0)	(0:0)	4096	1	(11:0)	(0:0)
RAMB4_S1_S2	4096	1	(11:0)	(0:0)	2048	2	(10:0)	(1:0)
RAMB4_S1_S4	4096	1	(11:0)	(0:0)	1024	4	(9:0)	(3:0)
RAMB4_S1_S8	4096	1	(11:0)	(0:0)	512	8	(8:0)	(7:0)
RAMB4_S1_S16	4096	1	(11:0)	(0:0)	256	16	(7:0)	(15:0)
RAMB4_S2_S2	2048	2	(10:0)	(1:0)	2048	2	(10:0)	(1:0)
RAMB4_S2_S4	2048	2	(10:0)	(1:0)	1024	4	(9:0)	(3:0)
RAMB4_S2_S8	2048	2	(10:0)	(1:0)	512	8	(8:0)	(7:0)
RAMB4_S2_S16	2048	2	(10:0)	(1:0)	256	16	(7:0)	(15:0)
RAMB4_S4_S4	1024	4	(9:0)	(3:0)	1024	4	(9:0)	(3:0)
RAMB4_S4_S8	1024	4	(9:0)	(3:0)	512	8	(8:0)	(7:0)
RAMB4_S4_S16	1024	4	(9:0)	(3:0)	256	16	(7:0)	(15:0)
RAMB4_S8_S8	512	8	(8:0)	(7:0)	512	8	(8:0)	(7:0)
RAMB4_S8_S16	512	8	(8:0)	(7:0)	256	16	(7:0)	(15:0)
RAMB4_S16_S16	256	16	(7:0)	(15:0)	256	16	(7:0)	(15:0)

ADDR=address bus for the port
DI=data input bus for the port

Each port is fully synchronous with independent clock pins. All port A input pins have setup time referenced to the CLKA pin and its data output bus DIA has a clock-to-out time referenced to the CLKA. All port B input pins have setup time referenced to the CLKB pin and its data output bus DIB has a clock-to-out time referenced to the CLKB.

The enable ENA pin controls read, write, and reset for port A. When ENA is Low, no data is written and the output (DOA) retains the last state. When ENA is High and reset (RSTA) is High, DOA is cleared during the Low-to-High clock (CLKA) transition; if write enable (WEA) is High, the memory contents reflect the data at DIA. When ENA is High and WEA is Low, the data stored in the RAM address (ADDRA) is read during the Low-to-High clock transition. When ENA and WEA are High, the data on the data input (DIA) is loaded into the word selected by the write address (ADDRA) during the Low-to-High clock transition and the data output (DOA) reflects the selected (addressed) word.

The enable ENB pin controls read, write, and reset for port B. When ENB is Low, no data is written and the output (DOB) retains the last state. When ENB is High and reset (RSTB) is High, DOB is cleared during the Low-to-High clock (CLKB) transition; if write enable (WEB) is High, the memory contents reflect the data at DIB. When ENB is High and WEB is Low, the data stored in the RAM address (ADDRB) is read during the Low-to-High clock transition. When ENB and WEB are High, the data on the data input (DIB) is loaded into the word selected by the write address (ADDRB) during the Low-to-High clock transition and the data output (DOB) reflects the selected (addressed) word.

The above descriptions assume active High control pins (ENA, WEA, RSTA, CLKA, ENB, WEB, RSTB, and CLKB). However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB4 port is absorbed into the block and does not use a CLB resource.

RAMB_Sn_Sn's may be initialized during configuration. See "Specifying Initial Contents of a Block RAM" section below.

Block RAM output registers are asynchronously cleared, output Low, when power is applied. The initial contents of the block RAM are not altered. Virtex and Spartan2 simulate power-on when global set / reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2 or STARTUP_VIRTEX symbol.

Mode selection is shown in the following truth table.

Inputs						Outputs	
EN(A/B)	**RST(A/B)**	**WE(A/B)**	**CLK(A/B)**	**ADDR(A/B)**	**DI(A/B)**	**DO(A/B)**	**RAM Contents**
0	X	X	X	X	X	No Chg	No Chg
1	1	0	↑	X	X	0	No Chg
1	1	1	↑	addr	data	0	RAM(addr) <=data
1	0	0	↑	addr	X	RAM(addr)	No Chg
1	0	1	↑	addr	data	data	RAM(addr) <=data

addr=RAM address of port A / B
RAM(addr)=RAM contents at address ADDRA / ADDRB
data=RAM input data at pins DIA / DIB

Address Mapping

Each port accesses the same set of 4096 memory cells using an addressing scheme that is dependent on the width of the port. The physical RAM location that is addressed for a particular width is determined from the following formula.

$$Start=((ADDR_{port}+1)*(Width_{port}))-1$$

$$End=(ADDR_{port})*(Width_{port})$$

The following table shows address mapping for each port width.

Table 9-1 Port Address Mapping

Port Width	Port Addresses																	
1	4096	<-----	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
2	2048	<-----	07		06		05		04		03		02		01		00	
4	1024	<-----	03				02				01				00			
8	512	<-----	01								00							
16	256	<-----	00															

Port A and Port B Conflict Resolution

A RAMB4_Sn_Sn component is a true dual-ported RAM in that it allows simultaneous reads of the same memory cell. When one port is performing a write to a given memory cell, the other port should not address that memory cell (for a write or a read) within the clock-to-clock setup window.

- If both ports write to the same memory cell simultaneously, violating the clock-to-setup requirement, the data stored will be invalid.
- If one port attempts to read from the same memory cell that the other is simultaneously writing to, violating the clock setup requirement, the write will be successful but the data read will be invalid.

Specifying Initial Contents of a Block RAM

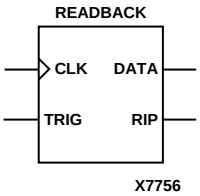
You can use the INIT_0x attributes to specify an initial value during device configuration. The initialization of each RAMB4_Sn_Sn is set by 16 initialization attributes (INIT_00 through INIT_0F) of 64 hex values for a total of 4096 bits. See the “INIT_0x” section of the “Attributes, Constraints, and Carry Logic” chapter for more information on these attributes.

If any INIT_0x attribute is not specified, it is configured as zeros. Partial initialization strings are padded with zeros to the left.

READBACK

FPGA Bitstream Readback Controller

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Macro	Macro	Macro	N/A	Macro	Macro	N/A	N/A



The READBACK macro accesses the bitstream readback function. A Low-to-High transition on the TRIG input initiates the readback process. The readback data appears on the DATA output. The RIP (readback-in-progress) output remains High during the readback process. If you use the ReadAbort:Enable option in BitGen, a High-to-Low transition on the TRIG input aborts the process. The signal on the CLK input clocks out the readback data; if no signal is connected to the CLK input, the internal CCLK is used. Set the ReadClk option in BitGen to indicate the readback clock source. (Refer to the *Development System Reference Guide* for information on BitGen.)

Typically, READBACK inputs are sourced by device-external input pins and outputs drive device-external output pins. If you want external input and output pins, connect READBACK pins through IBUFs or OBUFs to pads, as with any I/O device. However, you can connect READBACK pins to device-internal logic instead. For details on the READBACK process for each architecture, refer to *The Programmable Logic Data Book.*

Note: Virtex and Spartan2 provide the readback function through dedicated configuration port instructions, instead of with a READBACK component as in other FPGA architectures. For Virtex, refer to the "CAPTURE_VIRTEX" section for information on capturing register (flip-flop and latch) information for the Virtex readback function. For Spartan2, refer to the "CAPTURE_SPARTAN2" section.

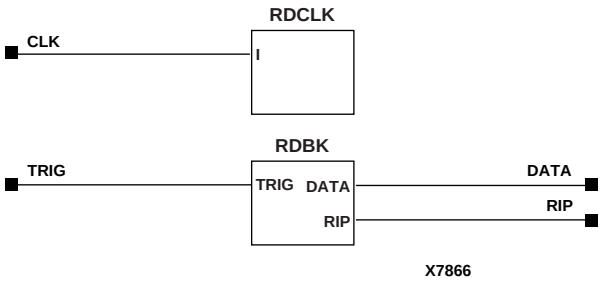
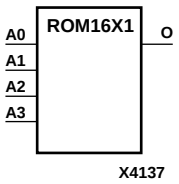


Figure 9-9 READBACK Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

ROM16X1

16-Deep by 1-Wide ROM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Primitive	Primitive	N / A	N / A	Primitive	Primitive	Primitive	Primitive



ROM16X1 is a 16-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 4-bit address (A3 – A0). The ROM is initialized to a known value during configuration with the INIT=*value* parameter. The *value* consists of four hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H. For example, the INIT=10A7 parameter produces the data stream

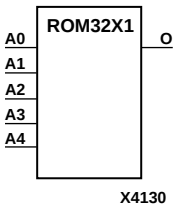
0001 0000 1010 0111

An error occurs if the INIT=*value* is not specified. Refer to the appropriate CAE tool interface user guide for details.

ROM32X1

32-Deep by 1-Wide ROM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	Primitive	Primitive	N / A	N / A	Primitive	Primitive	Primitive	Primitive



ROM32X1 is a 32-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 5-bit address (A4 – A0). The ROM is initialized to a known value during configuration with the INIT=*value* parameter. The *value* consists of eight hexadecimal digits that are written into the ROM from the most-significant digit A=1FH to the least-significant digit A=00H. For example, the INIT=10A78F39 parameter produces the data stream

0001 0000 1010 0111 1000 1111 0011 1001

An error occurs if the INIT=*value* is not specified. Refer to the appropriate CAE tool interface user guide for details.

Chapter 10

Design Elements (SOP3 to XORCY_L)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

SOP3-4

Sum of Products

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
SOP3, SOP3B1A, SOP3B1B, SOP3B2A, SOP3B2B, SOP3B3 SOP4, SOP4B1, SOP4B2A, SOP4B2B, SOP4B3, SOP4B4	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro

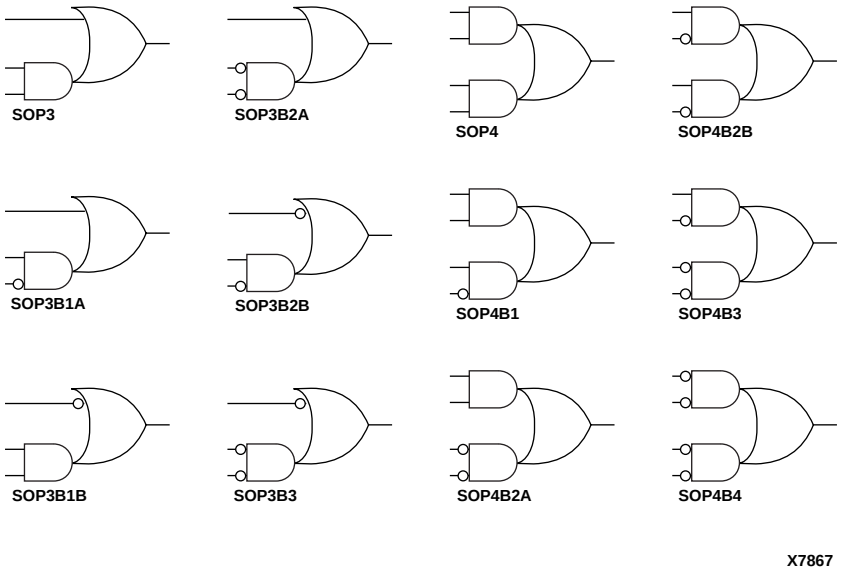


Figure 10-1 SOP Gate Representations

Sum Of Products (SOP) macros and primitives provide common logic functions by OR gating the outputs of two AND functions or the output of one AND function with one direct input. Variations of inverting and non-inverting inputs are available.

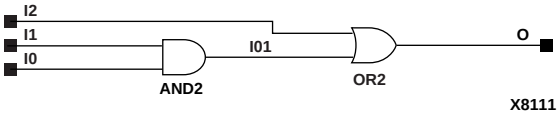


Figure 10-2 SOP3 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

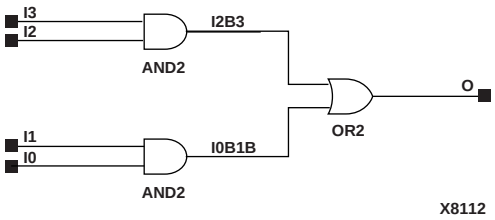
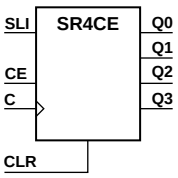


Figure 10-3 SOP4 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

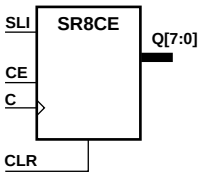
SR4CE, SR8CE, SR16CE

4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear

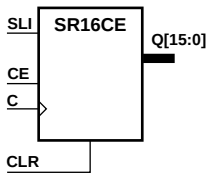
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



X4145



X4151



X4157

SR4CE, SR8CE, and SR16CE are 4-, 8-, and 16-bit shift registers, respectively, with a shift-left serial input (SLI), parallel outputs (Q), and clock enable (CE) and asynchronous clear (CLR) inputs. The CLR input, when High, overrides all other inputs and resets the data outputs (Q) Low. When CE is High and CLR is Low, the data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent Low-to-High clock transitions, when CE is High and CLR is Low, data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth). The register ignores clock transitions when CE is Low.

Registers can be cascaded by connecting the last Q output (Q3 for SR4CE, Q7 for SR8CE, or Q15 for SR16CE) of one stage to the SLI input of the next stage and connecting clock, CE, and CLR in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs	
CLR	CE	SLI	C	Q0	Qz – Q1
1	X	X	X	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	0	↑	0	qn-1

z = 3 for SR4CE; z = 7for SR8CE; z = 15 for SR16CE
qn-1 = state of referenced output one setup time prior to active clock transition

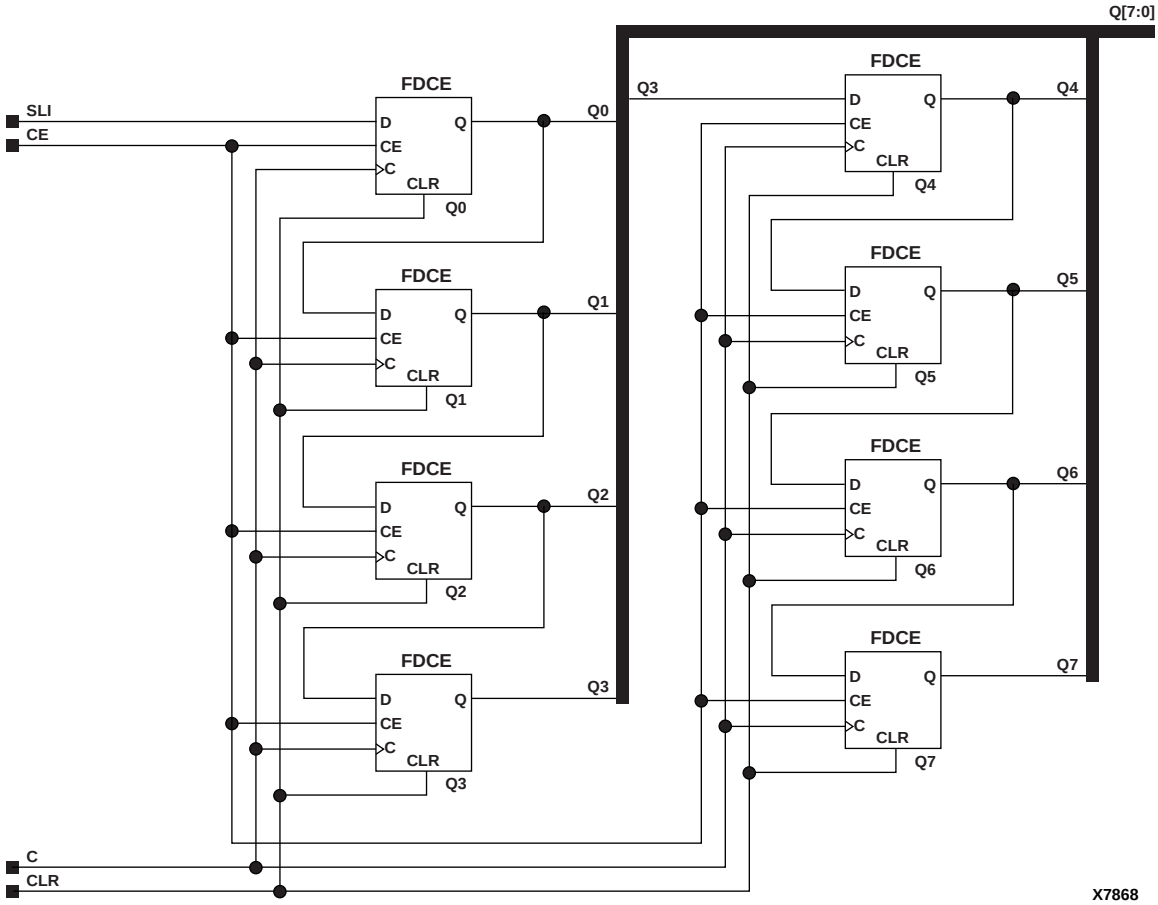
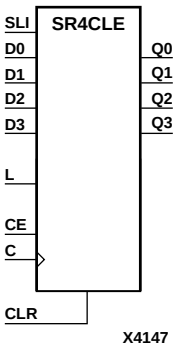


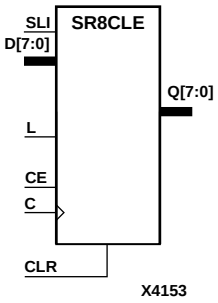
Figure 10-4 SR8CE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

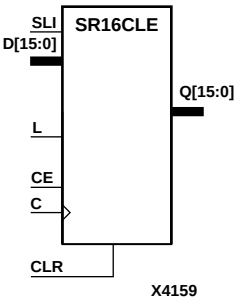
SR4CLE, SR8CLE, SR16CLE

4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







SR4CLE, SR8CLE, and SR16CLE are 4-, 8-, and 16-bit shift registers, respectively, with a shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and asynchronous clear (CLR).The register ignores clock transitions when L and CE are Low. The asynchronous CLR, when High, overrides all other inputs and resets the data outputs (Q) Low. When L is High and CLR is Low, data on the Dn – D0 inputs is loaded into the corresponding Qn – Q0 bits of the register. When CE is High and L and CLR are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and CLR are Low, the data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth).

Registers can be cascaded by connecting the last Q output (Q3 for SR4CLE, Q7 for SR8CLE, or Q15 for SR16CLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and CLR inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs	
CLR	L	CE	SLI	Dn – D0	C	Q0	Qz – Q1
1	X	X	X	X	X	0	0
0	1	X	X	Dn – D0	↑	d0	dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SR4CLE; z = 7 for SR8CLE; z = 15 for SR16CLE
dn = state of referenced input one setup time prior to active clock transition
qn-1 = state of referenced output one setup time prior to active clock transition

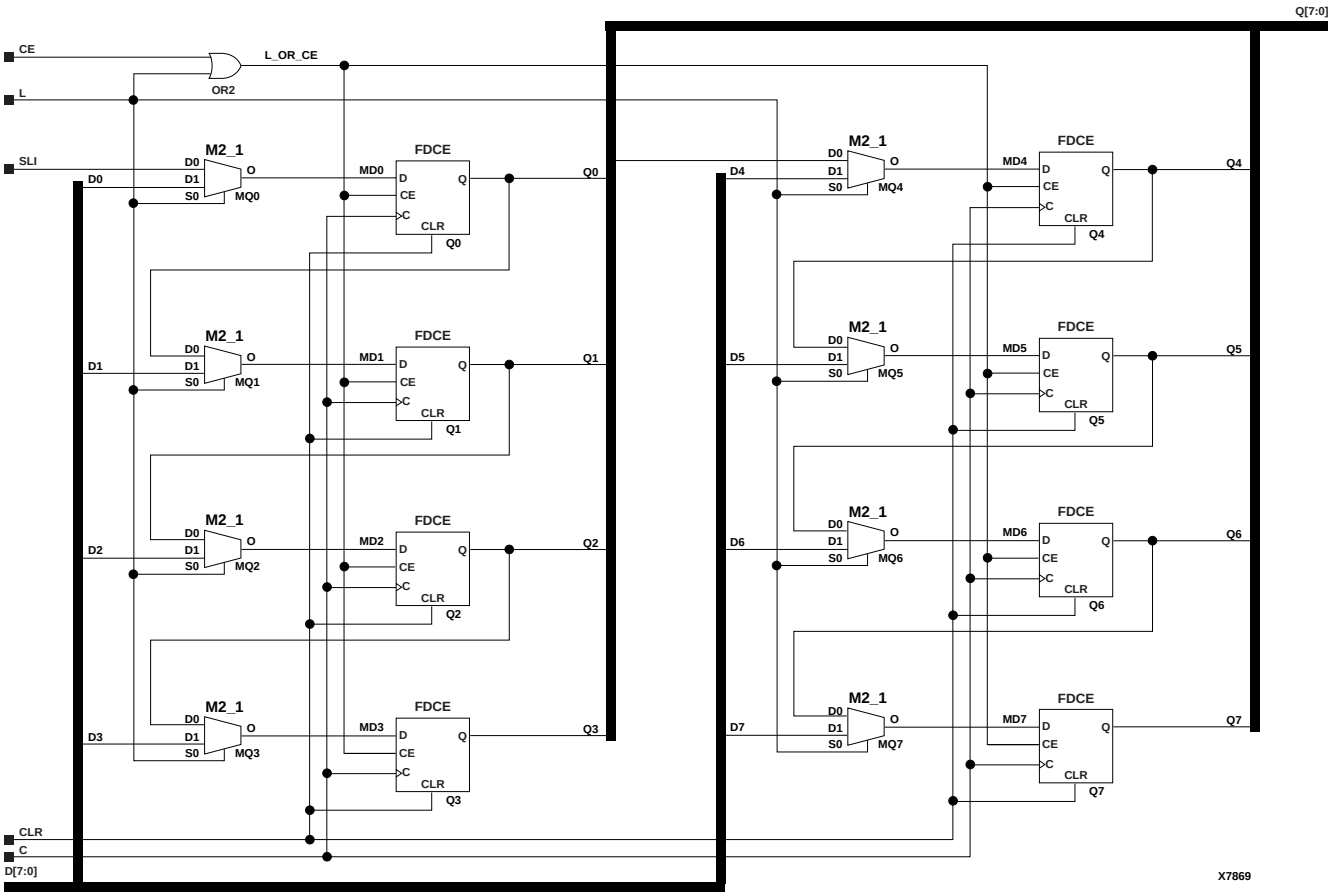
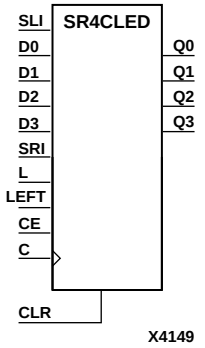


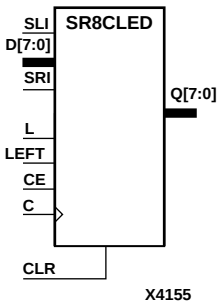
Figure 10-5 SR8CLE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

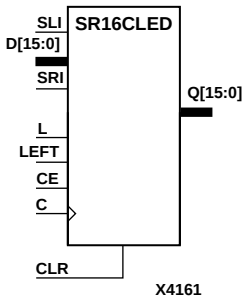
SR4CLED, SR8CLED, SR16CLED

4-, 8-, 16-Bit Shift Registers with Clock Enable and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







SR4CLED, SR8CLED, and SR16CLED are 4-, 8-, and 16-bit shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), and four control inputs: clock enable (CE), load enable (L), shift left / right (LEFT), and asynchronous clear (CLR). The register ignores clock transitions when CE and L are Low. The asynchronous clear, when High, overrides all other inputs and resets the data outputs (Qn) Low. When L is High and CLR is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and CLR are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on the SLI is loaded into Q0 during the Low-to-High clock transition and shifted left (to Q1, Q2, and so forth) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SR4CLED, Q7 for SR8CLED, or Q15 for SR16CLED) during the Low-to-High clock transition and shifted right (to Q2, Q1,... for SR4CLED; to Q6, Q5,... for SR8CLED; and to Q14, Q13,... for SR16CLED) during subsequent clock transitions. The truth tables for SR4CLED, SR8CLED, and SR16CLED indicate the state of the Q outputs under all input conditions for SR4CLED, SR8CLED, and SR16CLED.

The register is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Table 10-1 SR4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D3– D0	↑	d0	d3	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition
qn-1 and qn+1 = state of referenced output one setup time prior to active clock transition

Table 10-2 SR8CLED Truth Table

Inputs								Outputs		
CLR	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D7 – D0**	**C**	**Q0**	**Q7**	**Q6 – Q1**
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D7 – D0	↑	d0	d7	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

Table 10-3 SR16CLED Truth Table

Inputs								Outputs		
CLR	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D15 – D0**	**C**	**Q0**	**Q15**	**Q14 – Q1**
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D15 – D0	↑	d0	d15	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

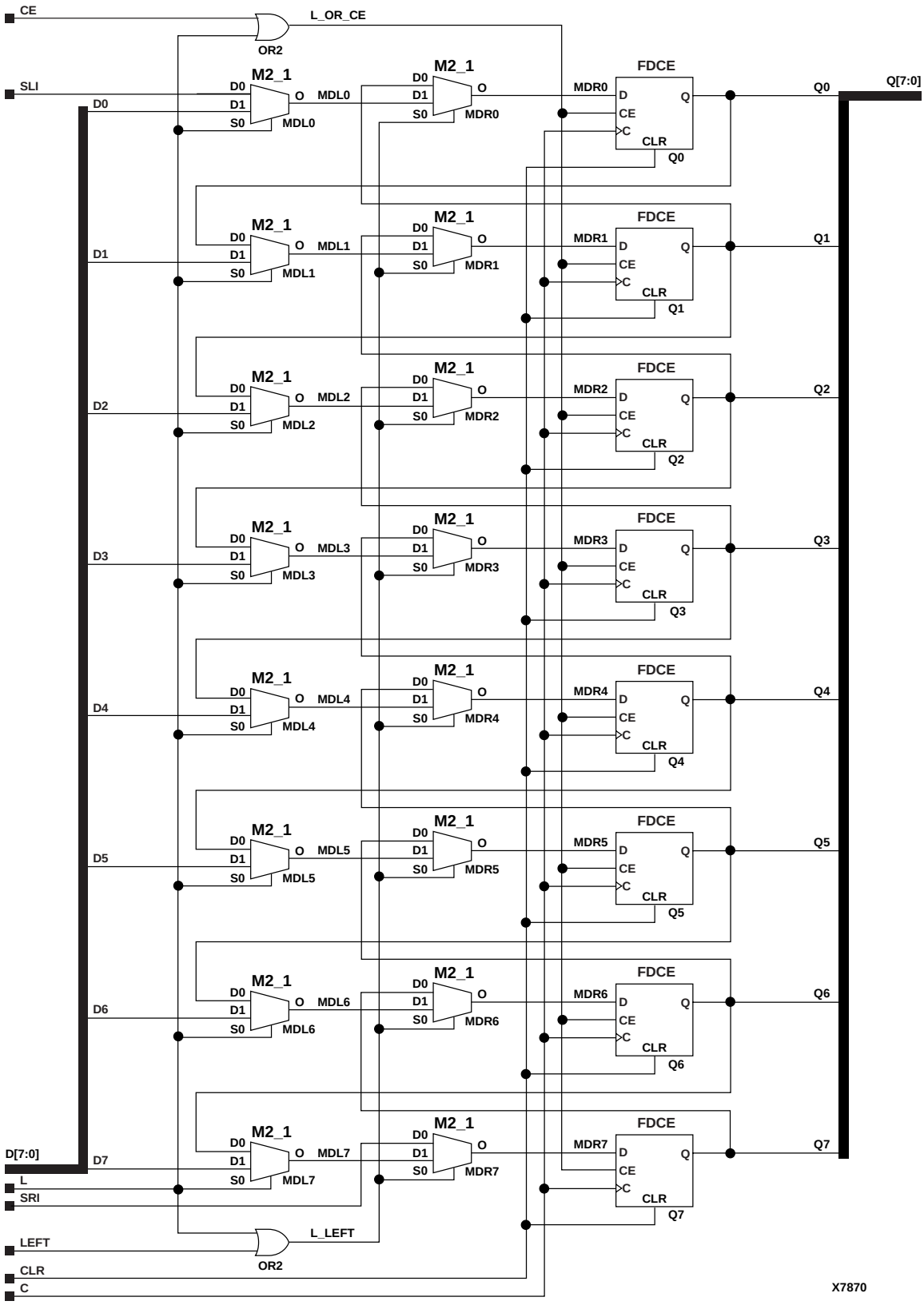


Figure 10-6 SR8CLED Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL, Spartan2, Virtex

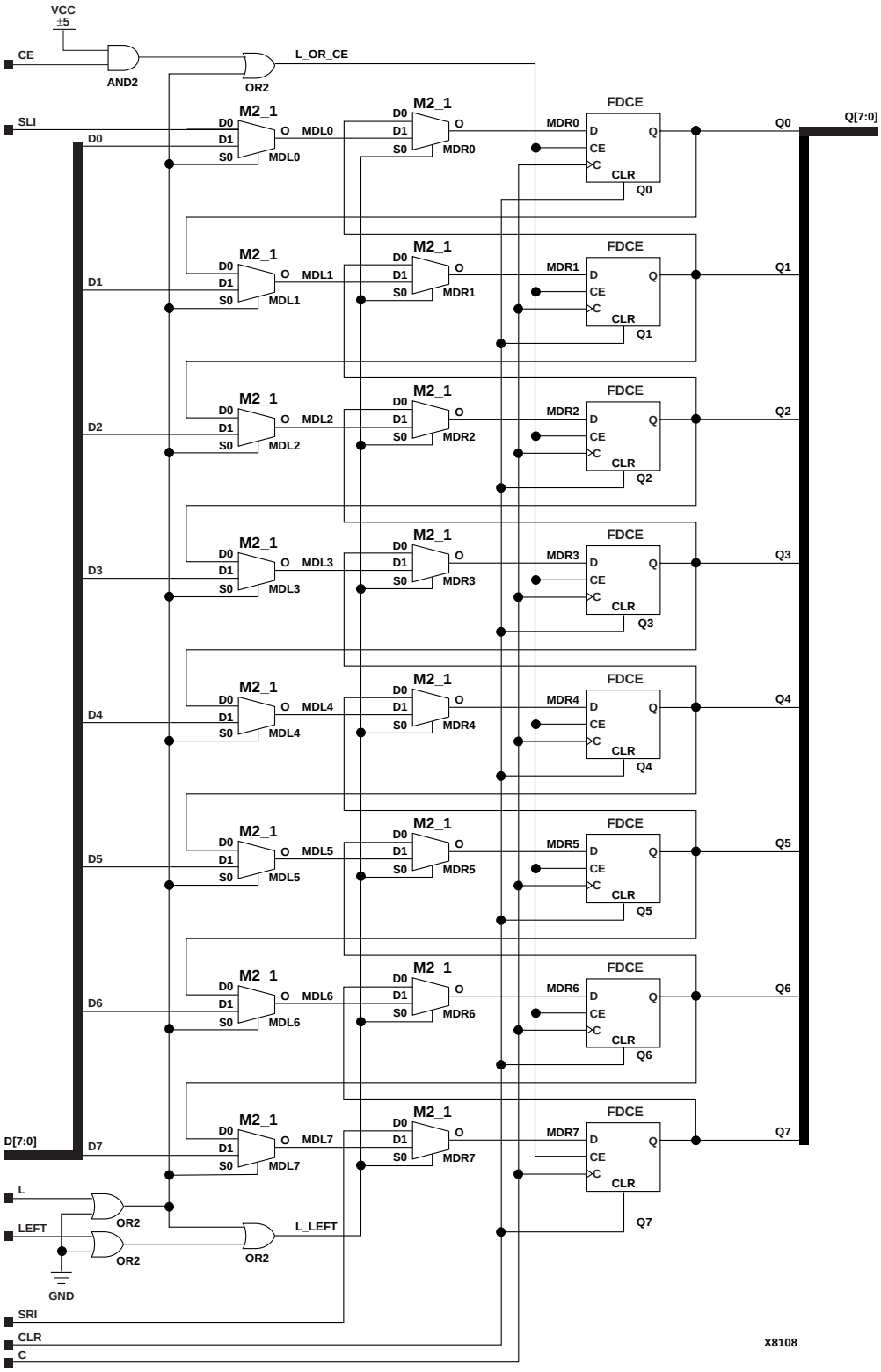
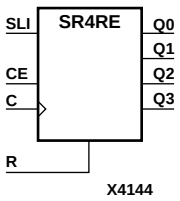


Figure 10-7 SR8CLED Implementation XC9000

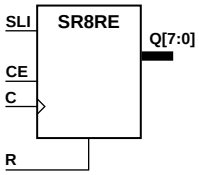
SR4RE, SR8RE, SR16RE

4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset

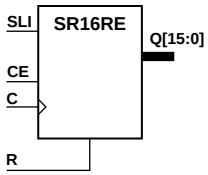
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro



X4144



X4150



X4156

SR4RE, SR8RE, and SR16RE are 4-, 8-, and 16-bit shift registers, respectively, with shift-left serial input (SLI), parallel outputs (Qn), clock enable (CE), and synchronous reset (R) inputs. The R input, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When CE is High and R is Low, the data on the SLI is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent Low-to-High clock transitions, when CE is High and R is Low, data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth). The register ignores clock transitions when CE is Low.

Registers can be cascaded by connecting the last Q output (Q3 for SR4RE, Q7 for SR8RE, or Q15 for SR16RE) of one stage to the SLI input of the next stage and connecting clock, CE, and R in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs				Outputs	
R	CE	SLI	C	Q0	Qz – Q1
1	X	X	↑	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	0	↑	0	qn-1

z = 3 for SR4RE; z = 7 for SR8RE; z = 15 for SR16RE
qn-1 = state of referenced output one setup time prior to active clock transition

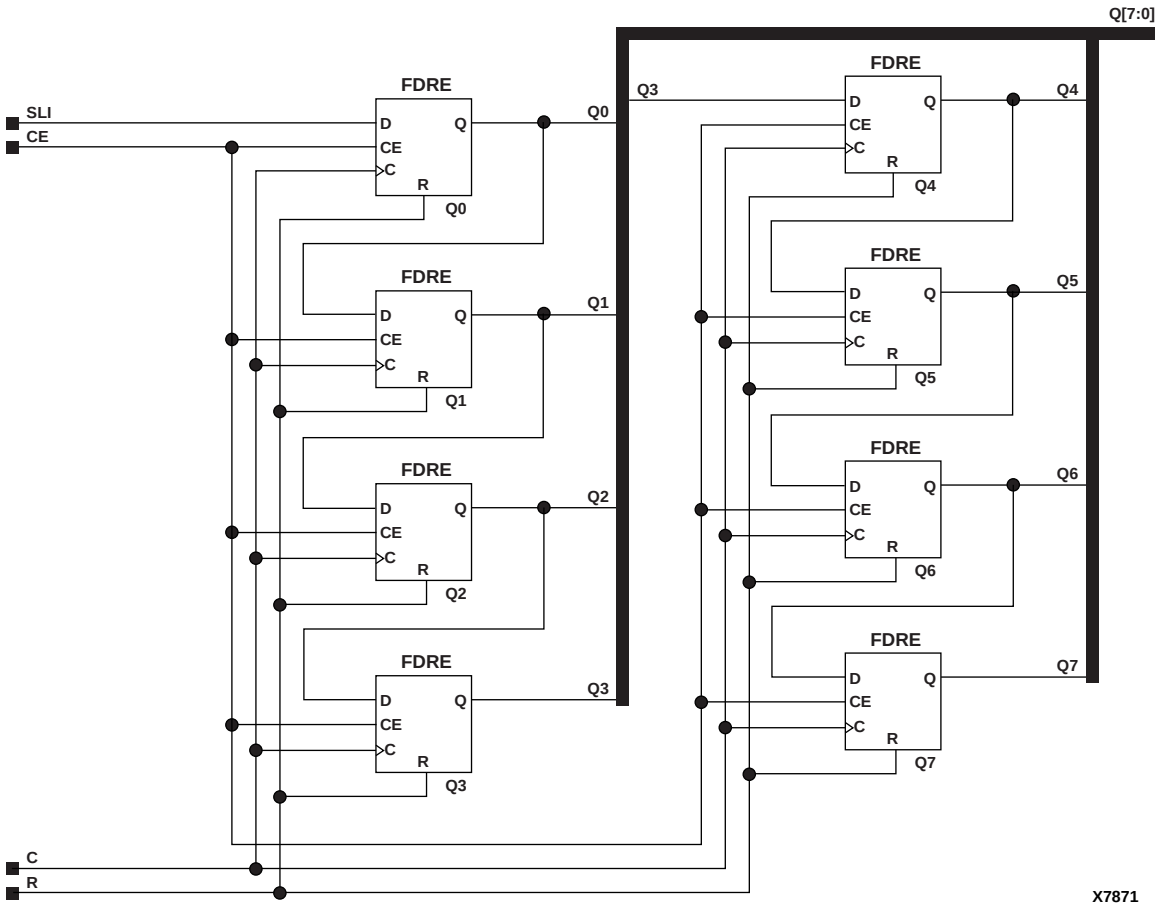
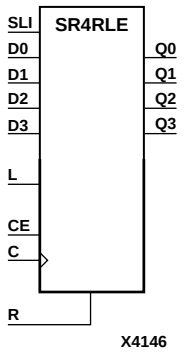


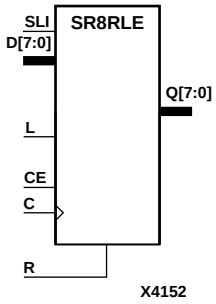
Figure 10-8 SR8RE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

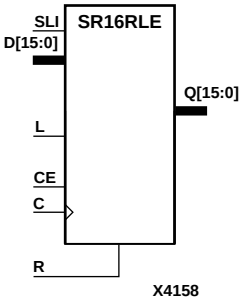
SR4RLE, SR8RLE, SR16RLE

4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







SR4RLE, SR8RLE, and SR16RLE are 4-, 8-, and 16-bit shift registers, respectively, with shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and synchronous reset (R). The register ignores clock transitions when L and CE are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low, data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and R are Low, the data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth).

Registers can be cascaded by connecting the last Q output (Q3 for SR4RLE, Q7 for SR8RLE, or 15 for SR16RLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and R inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set/reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR/GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Inputs						Outputs	
R	L	CE	SLI	Dz – D0	C	Q0	Qz – Q1
1	X	X	X	X	↑	0	0
0	1	X	X	Dz – D0	↑	d0	dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SR4RLE; z = 7 for SR8RLE; z = 15 for SR16RLE
dn = state of referenced input one setup time prior to active clock transition
qn-1 = state of referenced output one setup time prior to active clock transition

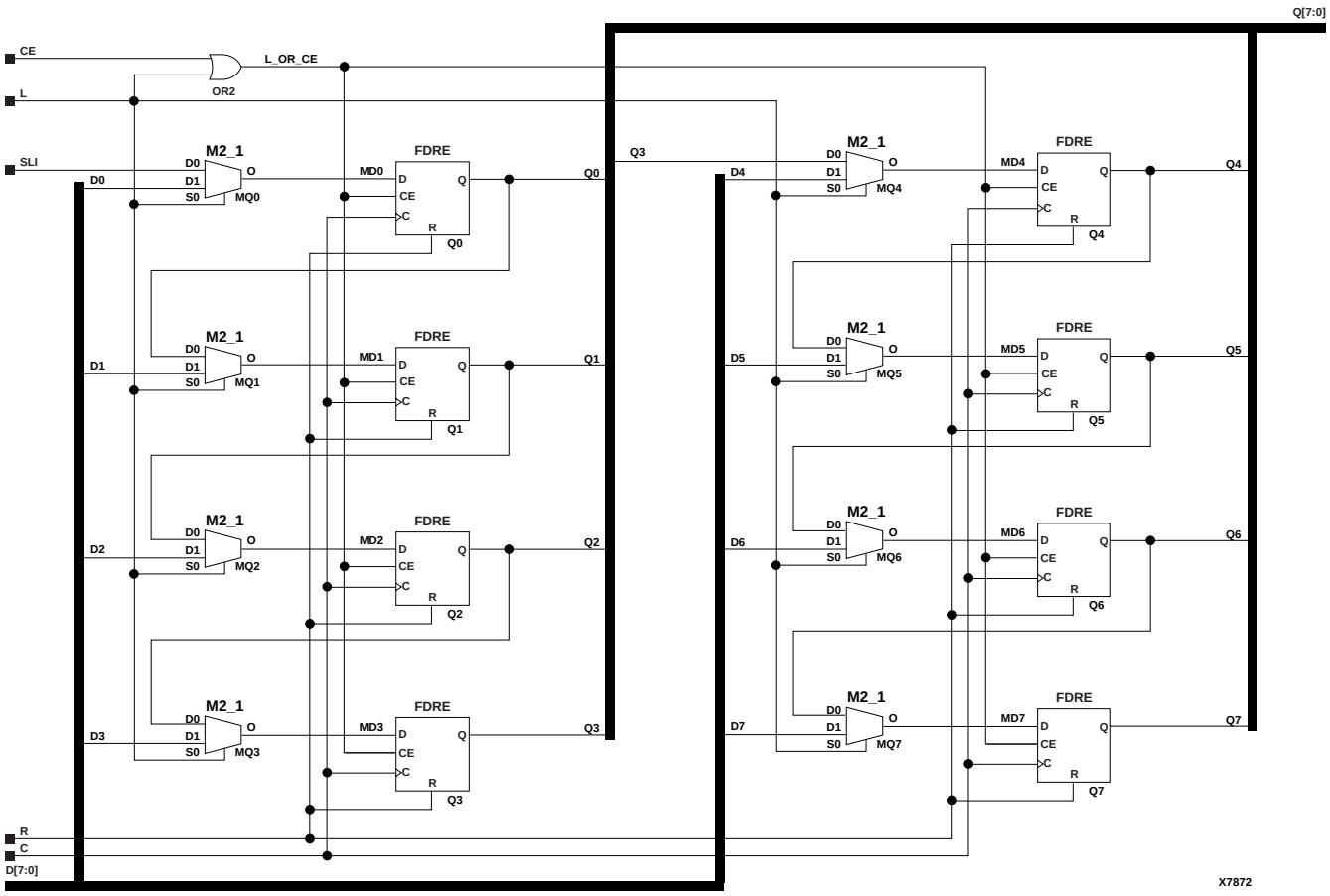
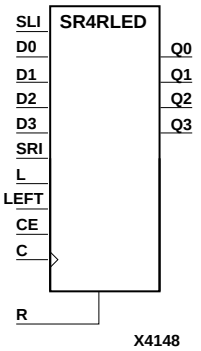


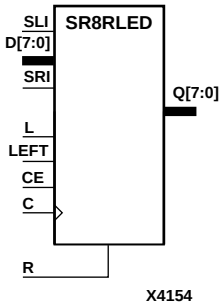
Figure 10-9 SR8RLE Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

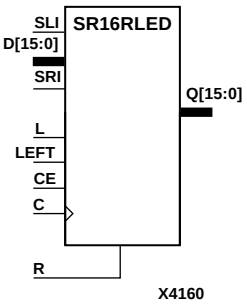
SR4RLED, SR8RLED, SR16RLED

4-, 8-, 16-Bit Shift Registers with Clock Enable and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro







SR4RLED, SR8RLED, and SR16RLED are 4-, 8-, and 16-bit shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), and four control inputs — clock enable (CE), load enable (L), shift left / right (LEFT), and synchronous reset (R). The register ignores clock transitions when CE and L are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on SLI is loaded into Q0 during the Low-to-High clock transition and shifted left (to Q1, Q2, and so forth) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SR4RLED, Q7 for SR8RLED, or Q15 for SR16RLED) during the Low-to-High clock transition and shifted right (to Q2, Q1,... for SR4RLED; to Q6, Q5,... for SR8RLED; or to Q14, Q13,... for SR16RLED) during subsequent clock transitions. The truth table indicates the state of the Q outputs under all input conditions.

The register is asynchronously cleared, outputs Low, when power is applied. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. FPGAs simulate power-on when global reset (GR) or global set / reset (GSR) is active. GR for XC3000 is active-Low. GR for XC5200 and GSR (XC4000, Spartans, Virtex) default to active-High but can be inverted by adding an inverter in front of the GR / GSR input of the STARTUP, STARTUP_SPARTAN2, or STARTUP_VIRTEX symbol.

Table 10-4 SR4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D3 – D0	↑	d0	d3	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

Table 10-5 SR8RLED Truth Table

Inputs								Outputs		
R	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D7– D0**	**C**	**Q0**	**Q7**	**Q6 – Q1**
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D7 – D0	↑	d0	d7	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

Table 10-6 SR16RLED Truth Table

Inputs								Outputs		
R	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D15 – D0**	**C**	**Q0**	**Q15**	**Q14 – Q1**
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D15 – D0	↑	d0	d15	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

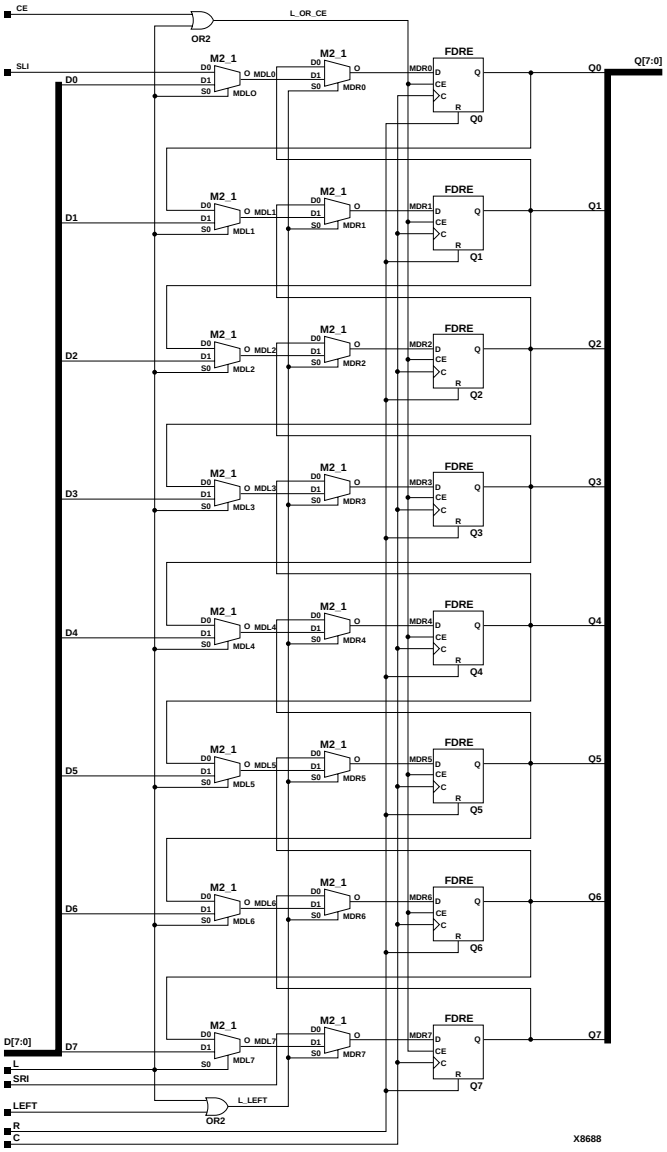
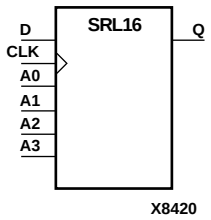


Figure 10-10 SR8RLED Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL, Spartan2, Virtex

SRL16

16-Bit Shift Register Look-Up-Table (LUT)

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



SRL16 is a shift register look up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

Static Length Mode

To get a fixed length shift register, drive the A3 through A0 inputs with static values. The length of the shift register can vary from 1 bit to 16 bits as determined from the following formula:

Length = (8*A3) +(4*A2) + (2*A1) + A0 +1

If A3, A2, A1, and A0 are all zeros (0000), the shift register is one bit long. If they are all ones (1111), it is 16 bits long.

Dynamic Length Mode

The length of the shift register can be changed dynamically by changing the values driving the A3 through A0 inputs. For example, if A2, A1, and A0 are all ones (111) and A3 toggles between a one (1) and a zero (0), the length of the shift register changes from 16 bits to 8 bits.

Internally, the length of the shift register is always 16 bits and the input lines A3 through A0 select which of the 16 bits reach the output.

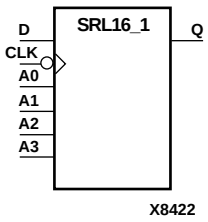
Inputs				Output
CLK	D	<SR(1)>	<SR(i)>	Q
1	X	No Chg	No Chg	No Chg
0	X	No Chg	No Chg	No Chg
↑	D	D	SR(i-1)	SR(L)

SR(1) = contents of first shift register
SR(i) = contents of the i'th shift register stage (2<= n <= L)
L = shift register length (1 through 16 determined by (8*A3) +(4*A2) + (2*A1) + A0 +1)

SRL16_1

16-Bit Shift Register Look-Up-Table (LUT) with Negative-Clock Edge

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



SRL16_1 is a shift register look up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. Refer to “Static Length Mode” and “Dynamic Length Mode” in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent Low-to-High clock transitions data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

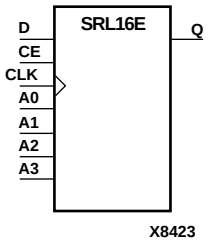
Inputs				Output
CLK	D	<SR(1)>	<SR(i)>	Q
1	X	No Chg	No Chg	No Chg
0	X	No Chg	No Chg	No Chg
↓	D	D	SR(i-1)	SR(L)

SR(1) = contents of first shift register
SR(i) = contents of the i’th shift register stage (2<= n <= L)
L = shift register length (1 through 16 determined by (8*A3) +(4*A2) + (2*A1) + A0 +1)

SRL16E

16-Bit Shift Register Look-Up-Table (LUT) with Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



SRL16E is a shift register look up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or dynamically adjusted. Refer to “Static Length Mode” and “Dynamic Length Mode” in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions, when CE is High, data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

When CE is Low, the register ignores clock transitions.

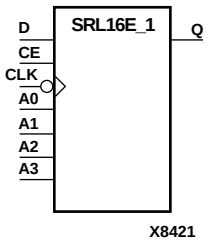
Inputs					Output
CE	CLK	D	<SR(1)>	<SR(i)>	Q
0	X	X	No Chg	No Chg	No Chg
1	1	X	No Chg	No Chg	No Chg
1	0	X	No Chg	No Chg	No Chg
1	↑	D	D	SR(i-1)	SR(L)

SR(1) = contents of first shift register
SR(i) = contents of the i’th shift register stage (2<= n <= L)
L = shift register length (1 through 16 determined by (8*A3) +(4*A2) + (2*A1) + A0 +1)

SRL16E_1

16-Bit Shift Register Look-Up-Table (LUT) with Negative-Edge Clock and Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N / A	N / A	N / A	N / A	N / A	N / A	N / A	Primitive	Primitive



SRL16E_1 is a shift register look up table (LUT) with clock enable (CE). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or dynamically adjusted. Refer to “Static Length Mode” and “Dynamic Length Mode” in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions, when CE is High, data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

When CE is Low, the register ignores clock transitions.

Inputs					Output
CE	CLK	D	<SR(1)>	<SR(i)>	Q
0	X	X	No Chg	No Chg	No Chg
1	1	X	No Chg	No Chg	No Chg
1	0	X	No Chg	No Chg	No Chg
1	↓	D	D	SR(i-1)	SR(L)

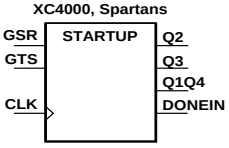
SR(1) = contents of first shift register
SR(i) = contents of the i’th shift register stage (2<= n <= L)
L = shift register length (1 through 16 determined by (8*A3) +(4*A2) + (2*A1) + A0 +1)

STARTUP

User Interface to Global Clock, Reset, and 3-State Controls

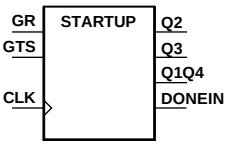
XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	N/A	N/A

XC4000, Spartans



X8060

XC5200

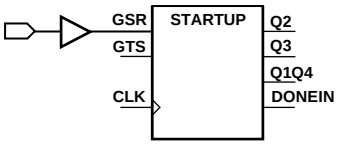


X8732

The STARTUP primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets every flip-flop in the device, depending on the initialization state (S or R) of the flip-flop. Following configuration, the global 3-state control (GTS), when High, forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

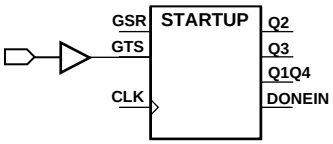
Including the STARTUP symbol in a design is optional. You must include the symbol under the following conditions.

- If you intend to exert external control over global set/reset, you must connect the GSR pin to an IPAD and an IBUF, as shown here. (For the XC5200, connect the GR pin to an IPAD and an IBUF.)



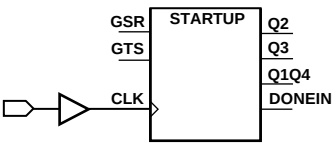
X6970

- If you intend to exert external control over global tristate, you must connect the GTS pin to an IPAD and IBUF, as shown here.



X3911

- If you wish to synchronize startup to a user clock, you must connect the user clock signal to the CLK input, as shown here. Furthermore, "user clock" must be selected in the BitGen program.



X6972

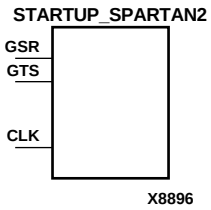
You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

The STARTUP outputs (Q2, Q3, Q1Q4, and DONEIN) display the progress/status of the start-up process following the configuration. Refer to *The Programmable Logic Data Book* for additional details.

STARTUP_SPARTAN2

Spartan2 User Interface to Global Clock, Reset, and 3-State Controls

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	N/A



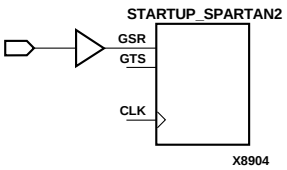
The STARTUP_SPARTAN2 primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAM (RAMB4) output register in the device, depending on the initialization state (S or R) of the component.

Note: Block RAMB4 content, LUT RAMs, delay locked loop elements (CLKDLL, CLKDLLHF, BUFGDLL), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1) are not set/reset.

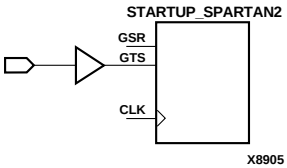
Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

Including the STARTUP_SPARTAN2 symbol in a design is optional. You must include the symbol under the following conditions.

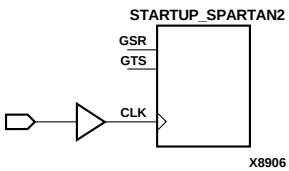
- If you intend to exert external control over global set/reset, you must connect the GSR pin to a top level port and an IBUF, as shown here.



- If you intend to exert external control over global tristate, you must connect the GTS pin to a top level port and IBUF, as shown here.



- If you wish to synchronize startup to a user clock, you must connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.

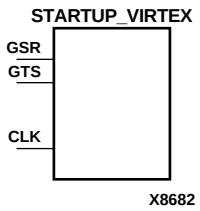


You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

STARTUP_VIRTEX

Virtex User Interface to Global Clock, Reset, and 3-State Controls

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive



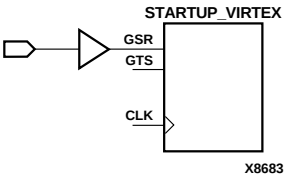
The STARTUP_VIRTEX primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAM (RAMB4) output register in the device, depending on the initialization state (S or R) of the component.

Note: Block RAMB4 content, LUT RAMs, delay locked loop elements (CLKDLL, CLKDLLHF, BUFGDLL), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1) are not set/reset.

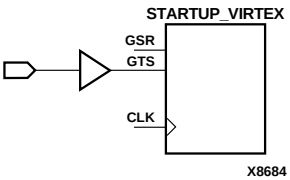
Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

Including the STARTUP_VIRTEX symbol in a design is optional. You must include the symbol under the following conditions.

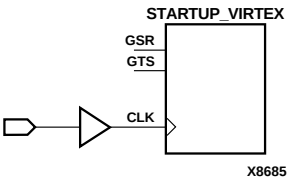
- If you intend to exert external control over global set/reset, you must connect the GSR pin to a top level port and an IBUF, as shown here.



- If you intend to exert external control over global tristate, you must connect the GTS pin to a top level port and IBUF, as shown here.



- If you wish to synchronize startup to a user clock, you must connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.

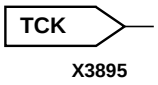


You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

TCK

Boundary Scan Test Clock Input Pad

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	N/A	N/A



The TCK input pad is connected to the boundary scan test clock, which shifts the serial data and instructions into and out of the boundary scan data registers. The function of the TCK pad is device configuration dependent and can be used as follows.

- During configuration TCK is connected to the boundary scan logic.
- After configuration, if boundary scan is not used, the TCK pad is unrestricted and can be used by the routing tool as an input/output pad.
- After configuration, if boundary scan is used, the TCK pad can be used for user-logic input by connecting it directly to the user logic.

TDI

Boundary Scan Test Data Input Pad

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	N/A	N/A

X3897

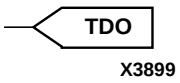
The TDI input pad is connected to the boundary scan TDI input. It loads instructions and data on the Low-to-High TCK transition. The function of the TDI pad is device configuration dependent and can be used as follows.

- During configuration, TDI is connected to the boundary scan logic.
- After configuration, if boundary scan is not used, the TDI pad is unrestricted and can be used by the routing tools as an input/output pad.
- After configuration, if boundary scan is used, the TDI pad can be used for user-logic input by connecting the TDI pad directly to the user logic.

TDO

Boundary Scan Data Output Pad

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	N/A	N/A



The TDO data output pad is connected to the boundary scan TDO output. It is connected to the external circuit to provide the boundary scan data for each Low-to-High TCK transition. The function of the TDO pad is device configuration dependent and can be used as follows.

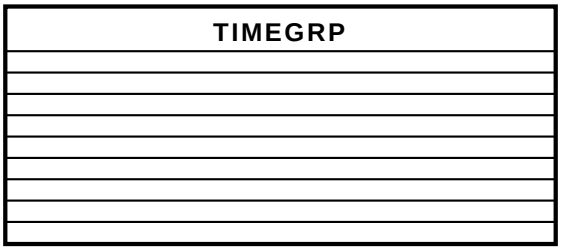
- During configuration, TDO is connected to the boundary scan logic.
- After configuration, if boundary scan is not used, the TDO pad can be used as a 3-state output pad by the routing tool.
- After configuration, if boundary scan is used, the TDO pad is still used as an output from the boundary scan logic.

TIMEGRP

Schematic-Level Table of Basic Timing Specification Groups

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

The TIMEGRP primitive table defines timing groups used in "from-to" TIMESPEC statements in terms of other groups. The TIMEGRP table is shown in the following figure.



X4699

These groups can include predefined groups, such as "ffs," groups created by using TNM attributes, such as TNM-reg on schematics, and other groups defined by a statement in the TIMEGRP symbol.

The following sample statement defines groups in a TIMEGRP symbol.

```
TIMEGRP=all_but_regs=ffs:except:regs
```

The table can contain up to 8 statements of any character length, but only 30 characters are displayed in the symbol.

Note: When entering timegroup properties into a TIMEGRP symbol, some property names should not be used because they cause a conflict with the predefined (reserved) property names of the TIMEGRP primitive.

The standard procedure for adding a property to a symbol is to use the following command.

PROPERTY = *property_name* **VALUE=***value*

For *property_name* you must not use any of the system reserved names LIBVER, INST, COMP, MODEL, or any other names reserved by your schematic capture program. Please consult your schematic capture documentation to familiarize yourself with reserved property names.

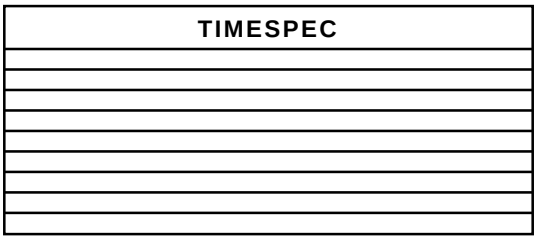
For more on time group attributes, see the "Time Group Attributes" section of the "Attributes, Constraints, and Carry Logic" chapter.

TIMESPEC

Schematic-Level Timing Requirement Table

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

The TIMESPEC primitive is a table that you can use to specify up to eight timing attributes (TS). TS attributes can be any length, but only 30 characters are displayed in the TIMESPEC window. The TIMESPEC table is displayed in the following figure.



X3866

For more information on "TS" timing attributes refer to the “TSidentifier” section of the “Attributes, Constraints, and Carry Logic” chapter.

TMS

Boundary Scan Test Mode Select Input Pad

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	N/A	N/A

X3901

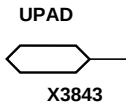
The TMS input pad is connected to the boundary scan TMS input. It determines which boundary scan operation is performed. The function of the TMS pad is device configuration dependent and can be used as follows.

- During configuration, TMS is connected to the boundary scan logic.
- After configuration, if boundary scan is not used, the TMS pad is unrestricted and can be used by the routing tools as an input/output pad.
- After configuration, if boundary scan is used, the TMS pad can be used for user-logic input by connecting the TMS pad directly to the user logic.

UPAD

Connects the I/O Node of an IOB to the Internal PLD Circuit

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	N/A	Primitive	Primitive	Primitive	Primitive



A UPAD allows the use of any unbonded IOBs in a device. It is used the same way as a IOPAD except that the signal output is not visible on any external device pins.

VCC

VCC-Connection Signal Tag

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

X8721

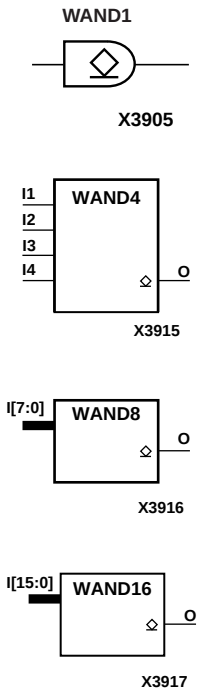
The VCC signal tag or parameter forces a net or input function to a logic High level. A net tied to VCC cannot have any other source.

When the placement and routing software encounters a net or input function tied to VCC, it removes any logic that is disabled by the VCC signal. The VCC signal is only implemented when the disabled logic cannot be removed.

WAND1, 4, 8, 16

Open-Drain Buffers

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
WAND1	N/A	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A	N/A
WAND4, WAND8. WAND16	N/A	Macro	Macro	N/A	N/A	N/A	N/A	N/A	N/A



WAND1, WAND4, WAND8, and WAND16 are single and multiple open-drain buffers. Each buffer has an input (I) and an open-drain output (O). When any of the inputs is Low, the output is Low. When all the inputs are High, the output is off. To obtain a High output, add pull-up resistors to the output (O). One pull-up resistor uses the least power, and two pull-up resistors achieve the fastest Low-to-High transition.

To indicate two pull-up resistors, add a DOUBLE parameter to the pull-up symbol attached to the output (O) node. Refer to the appropriate CAE tool interface user guide for details.

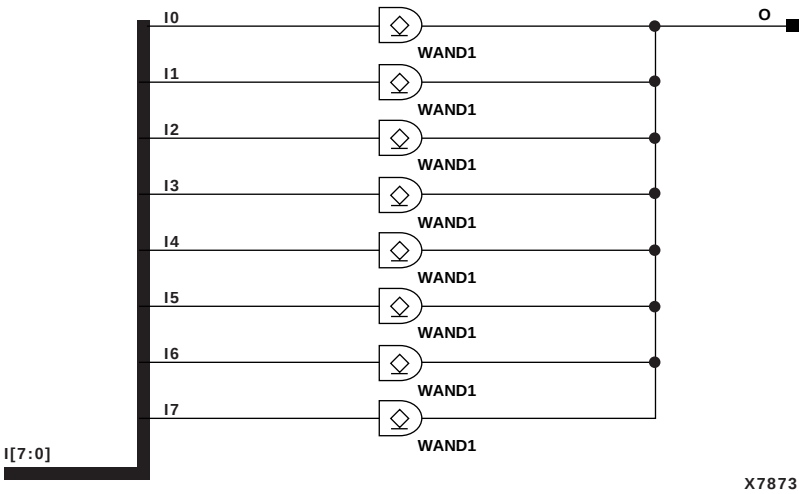
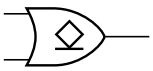


Figure 10-11 WAND8 Implementation XC4000E, XC4000X

WOR2AND

2-Input OR Gate with Wired-AND Open-Drain Buffer Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	Primitive	Primitive	N/A	N/A	N/A	N/A	N/A	N/A



X3906

WOR2AND is a 2-input (I1 and I2) OR gate/buffer with an open-drain output (O). It is used in bus applications by tying multiple open-drain outputs together. When both inputs (I1 and I2) are Low, the output (O) is Low. When either input is High, the output is off; WOR2AND cannot source or sink current. To establish an output High level, tie a pull-up resistor(s) to the output (O). One pull-up resistor uses the least power, two pull-up resistors achieve the fastest Low-to-High speed.

To indicate two pull-up resistors, append a DOUBLE parameter to the pull-up symbol attached to the output (O) node. Refer to the appropriate CAE tool interface user guide for details.

XNOR2-9

2- to 9-Input XNOR Gates with Non-Inverted Inputs

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
XNOR2, XNOR3, XNOR4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR5	Primitive	Primitive	Primitive	Macro	Macro	Primitive	Primitive	Primitive	Primitive
XNOR6, XNOR7, XNOR8, XNOR9	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro

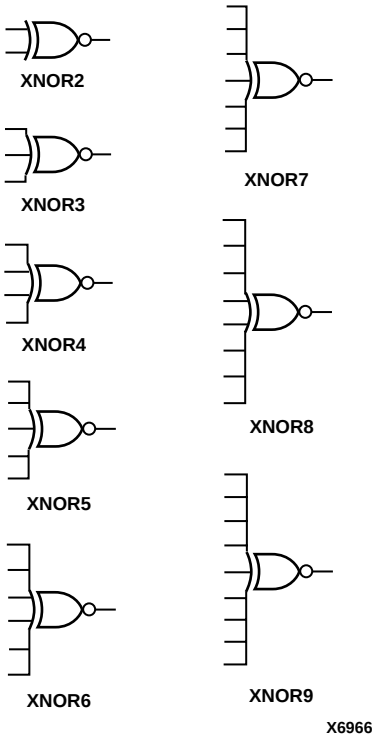


Figure 10-12 XNOR Gate Representations

The XNOR function is performed in the Configurable Logic Block (CLB) function generators in XC3000, XC4000, Spartan, and SpartanXL. XNOR functions of up to nine inputs are available. All inputs are non-inverting. Because each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.

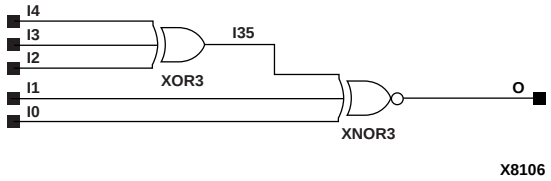


Figure 10-13 XNOR5 Implementation XC5200

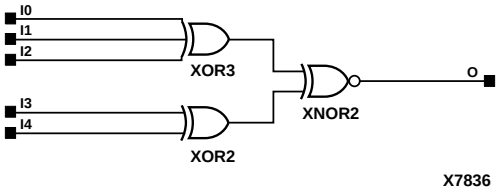


Figure 10-14 XNOR5 Implementation XC9000

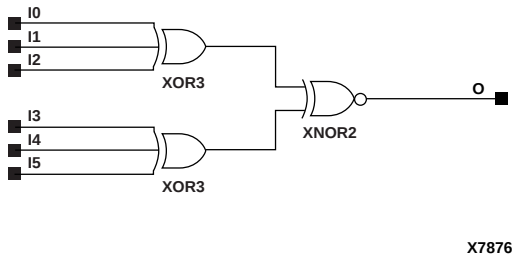


Figure 10-15 XNOR6 Implementation XC9000

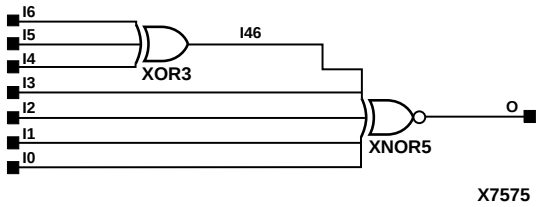


Figure 10-16 XNOR7 Implementation XC3000

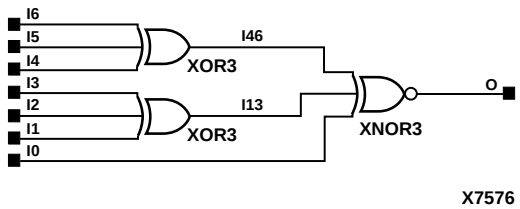


Figure 10-17 XNOR7 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

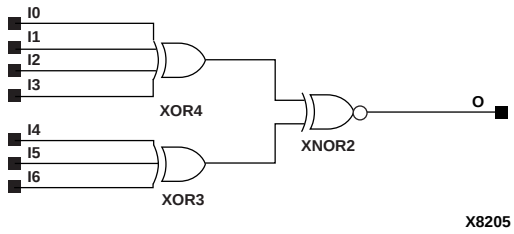


Figure 10-18 XNOR7 Implementation XC9000

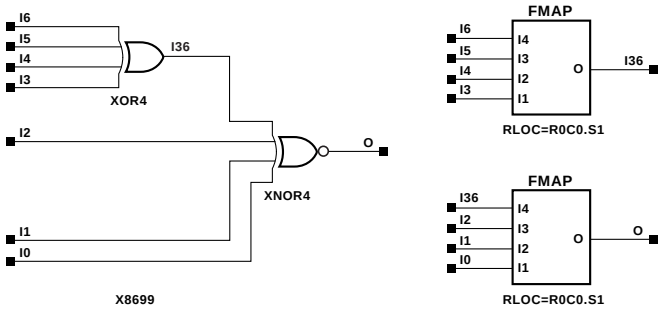


Figure 10-19 XNOR7 Implementation Spartan2, Virtex

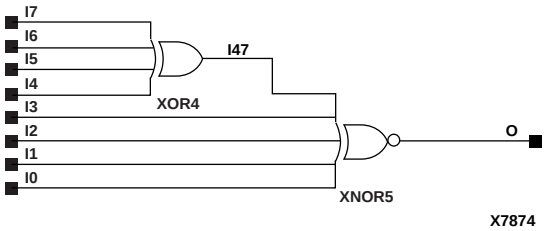


Figure 10-20 XNOR8 Implementation XC3000

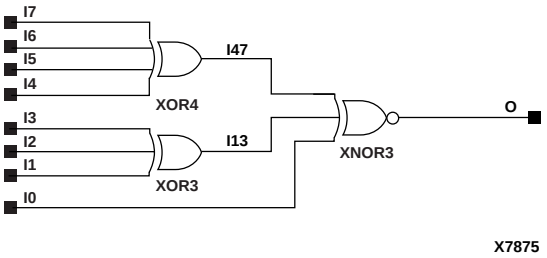


Figure 10-21 XNOR8 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

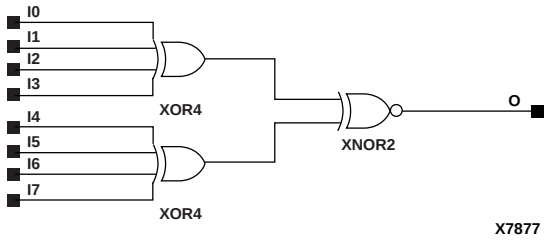


Figure 10-22 XNOR8 Implementation XC9000

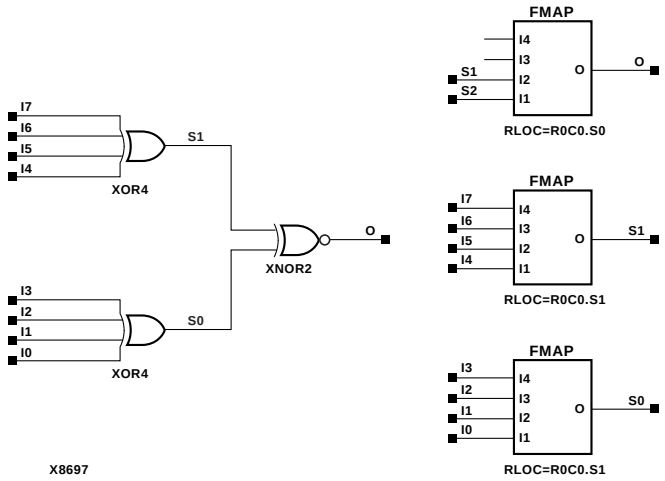


Figure 10-23 XNOR8 Implementation Spartan2, Virtex

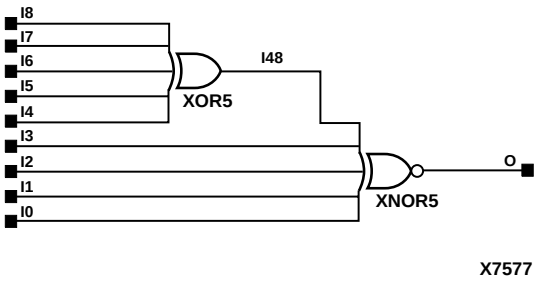


Figure 10-24 XNOR9 Implementation XC3000

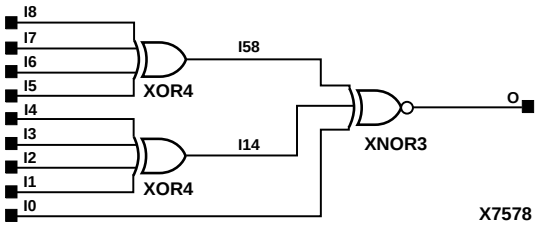


Figure 10-25 XNOR9 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

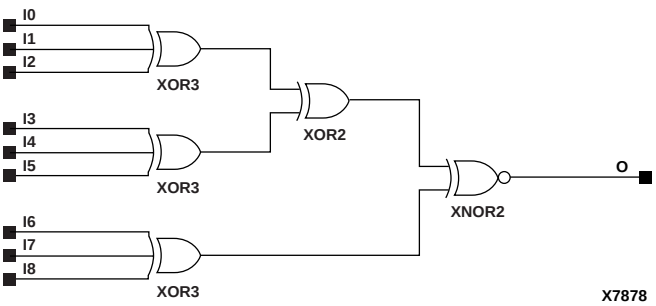


Figure 10-26 XNOR9 Implementation XC9000

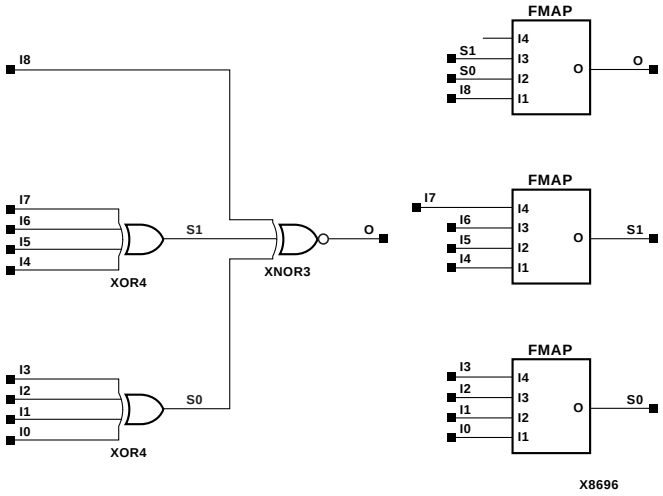


Figure 10-27 XNOR9 Implementation Spartan2, Virtex

XOR2-9

2- to 9-Input XOR Gates with Non-Inverted Inputs

Element	XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
XOR2, XOR3, XOR4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XOR5	Primitive	Primitive	Primitive	Macro	Macro	Primitive	Primitive	Primitive	Primitive
XOR6, XOR7, XOR8, XOR9	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro	Macro

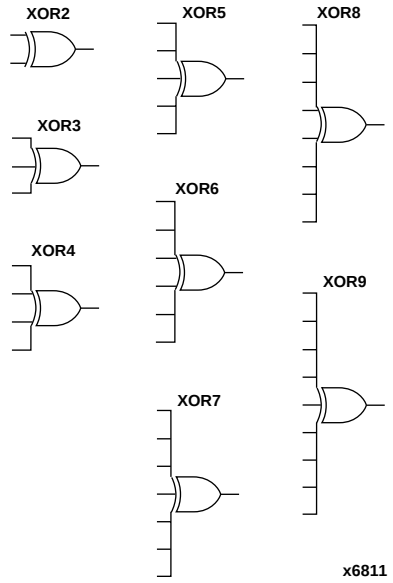


Figure 10-28 XOR Gate Representations

The XOR function is performed in the Configurable Logic Block (CLB) function generators in XC3000, XC4000, Spartan, and SpartanXL. XOR functions of up to nine inputs are available. All inputs are non-inverting. Because each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.

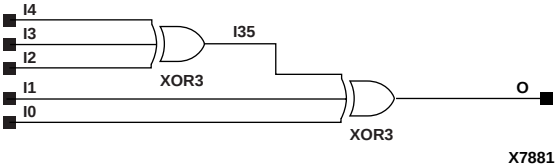


Figure 10-29 XOR5 Implementation XC5200

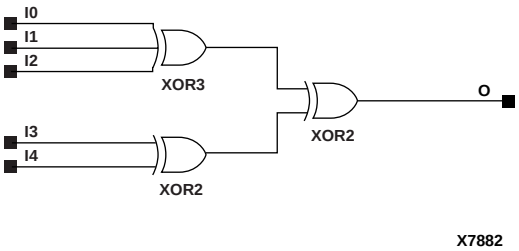


Figure 10-30 XOR5 Implementation XC9000

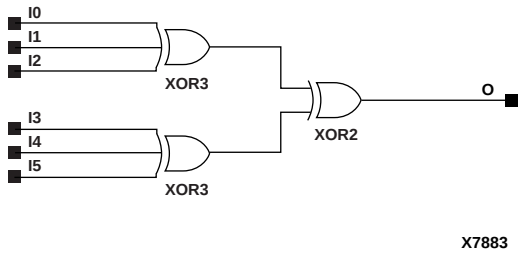


Figure 10-31 XOR6 Implementation XC9000

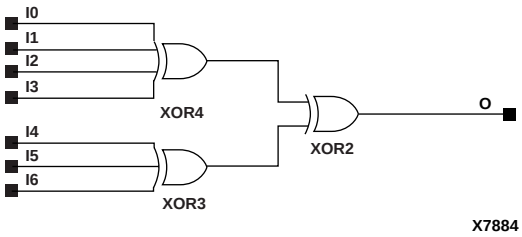


Figure 10-32 XOR7 Implementation XC9000

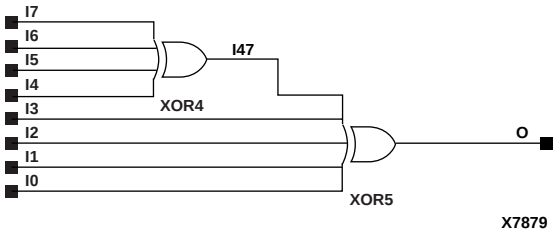


Figure 10-33 XOR8 Implementation XC3000

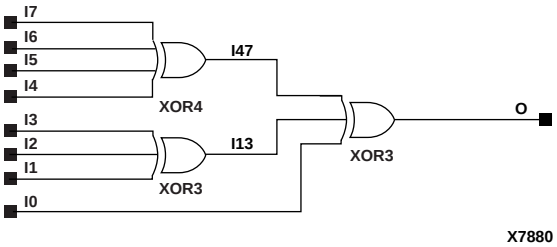


Figure 10-34 XOR8 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

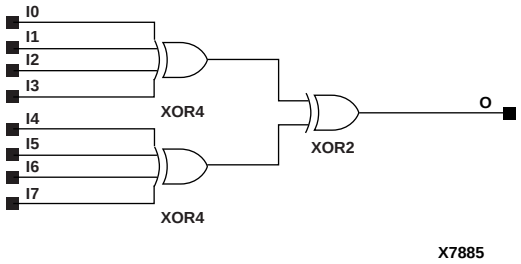


Figure 10-35 XOR8 Implementation XC9000

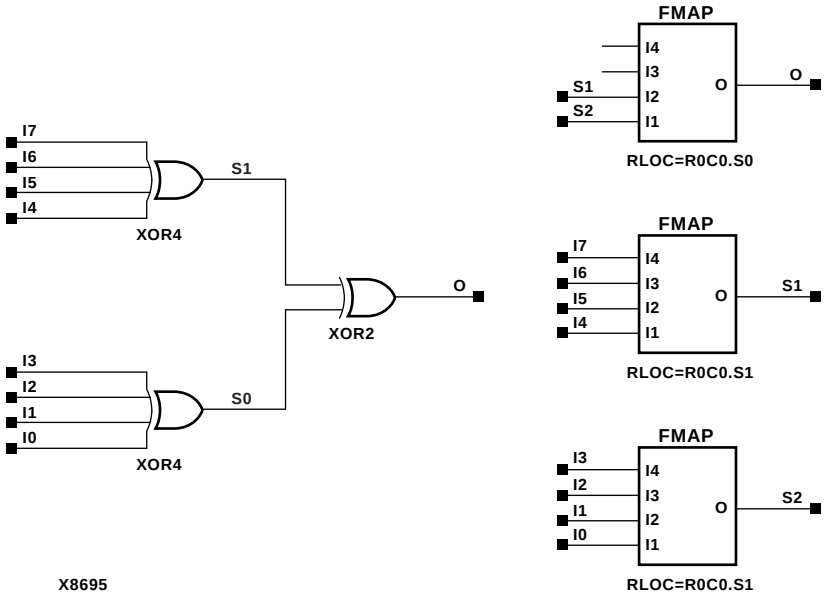


Figure 10-36 XOR8 Implementation Spartan2, Virtex

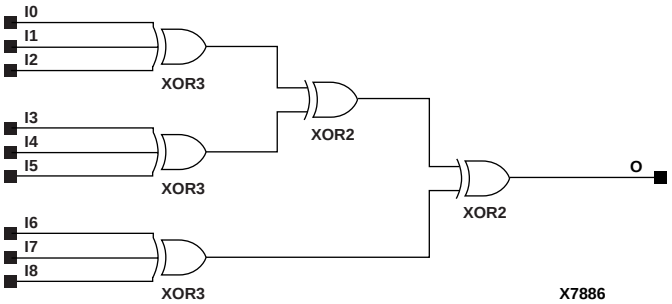
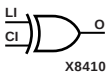


Figure 10-37 XOR9 Implementation XC9000

XORCY

XOR for Carry Logic with General Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



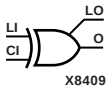
XORCY is a special XOR with general O output used for generating faster and smaller arithmetic functions.

Its O output is a general interconnect. See also“XORCY_D” and “XORCY_L.”

XORCY_D

XOR for Carry Logic with Dual Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



XORCY_D is a special XOR used for generating faster and smaller arithmetic functions.

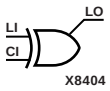
XORCY_D has two functionally identical outputs, O and LO. The O output is a general interconnect. The LO output is used to connect to another output within the same CLB slice.

See also “XORCY” and “XORCY_L.”

XORCY_L

XOR for Carry Logic with Local Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Primitive	Primitive



XORCY_L is a special XOR with local LO output used for generating faster and smaller arithmetic functions. The LO output is used to connect to another output within the same CLB slice.

See also “XORCY” an d “XORCY_D.”

Chapter 11

Design Elements (X74_42 to X74_521)

This chapter describes design elements included in the Unified Libraries. The elements are organized in alphanumeric order with all numeric suffixes in ascending order.

The library applicability table at the beginning of an element description identifies how the element is implemented in each library as follows.

- Primitive

  A primitive is a basic building block that cannot be broken up into smaller components.

- Macro

  A macro is constructed from primitives. Macros whose implementations contain relative location constraint (RLOC) information are known as Relationally Placed Macros (RPMs).

  Schematics for macro implementations are included at the end of the component description. Schematics are included for each library if the macro implementation differs. Design elements with bused or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

- N/A

  Certain design elements are not available in all libraries because they cannot be accommodated in all device architectures. These are marked as N/A (Not Available).

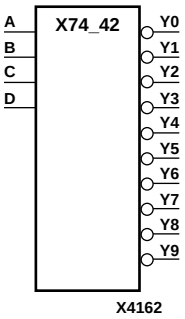
Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific architectures supported by each of the following libraries: XC3000 Library, XC4000E Library, XC4000X Library, XC5200 Library, XC9000 Library, Spartan Library, SpartanXL Library, Spartan2 Library, and Virtex Library.

Note: Wherever *XC4000* is used, the information applies to all architectures supported by the XC4000E and XC4000X libraries. Wherever *Spartans* is used, the information applies to all architectures supported by the Spartan, SpartanXL, and Spartan2 libraries.

X74_42

4- to 10-Line BCD-to-Decimal Decoder with Active-Low Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_42 decodes the 4-bit BCD number on the data inputs (A – D). Only one of the ten outputs (Y9 – Y0) is active (Low) at a time, which reflects the decimal equivalent of the BCD number on inputs A – D. All outputs are inactive (High) during any one of six illegal states, as shown in the truth table.

Inputs				Outputs
D	C	B	A	Selected (Low) Output
0	0	0	0	Y0
0	0	0	1	Y1
0	0	1	0	Y2
0	0	1	1	Y3
0	1	0	0	Y4
0	1	0	1	Y5
0	1	1	0	Y6
0	1	1	1	Y7
1	0	0	0	Y8
1	0	0	1	Y9
1	0	1	0	All Outputs High
1	0	1	1	All Outputs High
1	1	0	0	All Outputs High
1	1	0	1	All Outputs High
1	1	1	0	All Outputs High
1	1	1	1	All Outputs High

Selected output is Low (0) and all others are High

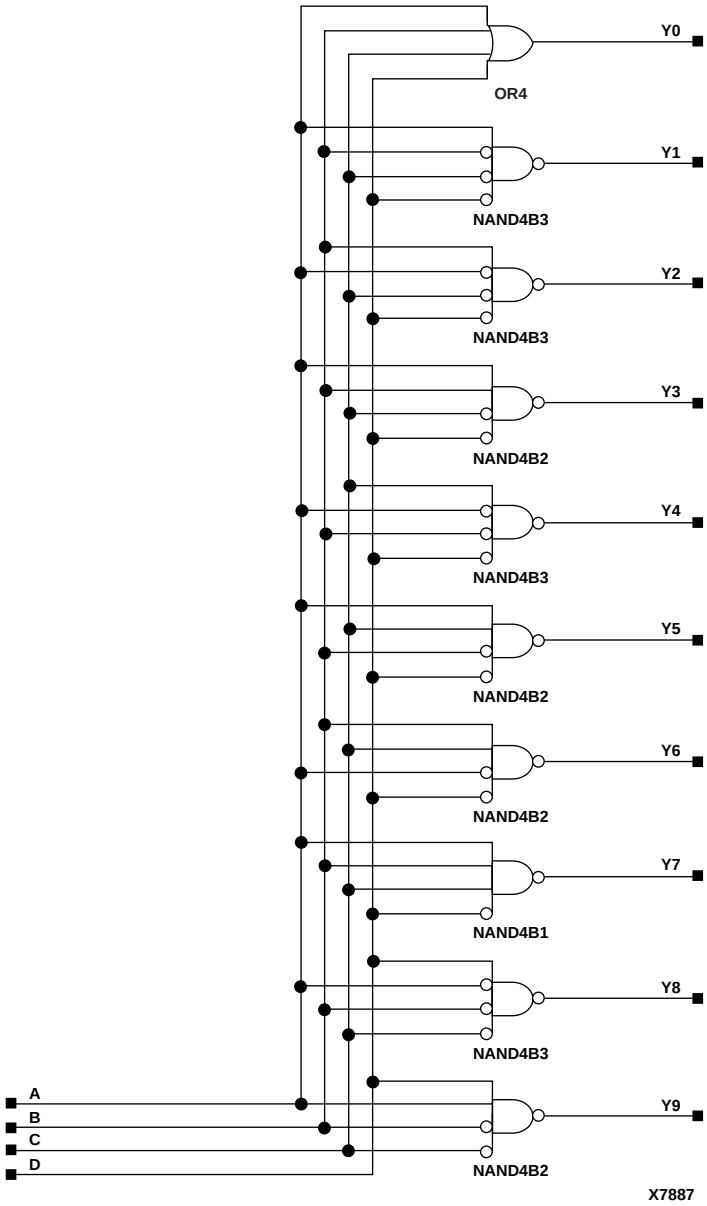
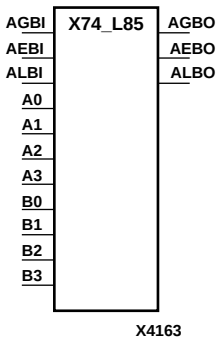


Figure 11-1 X74_42 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_L85

4-Bit Expandable Magnitude Comparator

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_L85 is a 4-bit magnitude comparator that compares two 4-bit binary-weighted words A3 – A0 and B3 – B0, where A3 and B3 are the most significant bits. The greater-than output, AGBO, is High when A>B. The less-than output, ALBO, is High when A<B, and the equal output, AEBO, is High when A=B. The expansion inputs, AGBI, ALBI, and AEBI, are the least significant bits. Words of greater length can be compared by cascading the comparators. The AGBO, ALBO, and AEBO outputs of the stage handling less-significant bits are connected to the corresponding AGBI, ALBI, and AEBI inputs of the next stage handling more-significant bits. For proper operation, the stage handling the least significant bits must have AGBI and ALBI tied Low and AEBI tied High.

Inputs							Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	AGBI	ALBI	AEBI	AGBO	ALBO	AEBO
A3>B3	X	X	X	X	X	X	1	0	0
A3<B3	X	X	X	X	X	X	0	1	0
A3=B3	A2>B2	X	X	X	X	X	1	0	0
A3=B3	A2<B2	X	X	X	X	X	0	1	0
A3=B3	A2=B2	A1>B1	X	X	X	X	1	0	0
A3=B3	A2=B2	A1<B1	X	X	X	X	0	1	0
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	1	0	0
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	1	0	0	1	0	0
A3=B3	A2=B2	A1=B1	A0=B0	0	1	0	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	0	0	1	0	0	1
A3=B3	A2=B2	A1=B1	A0=B0	0	1	1	0	1	1
A3=B3	A2=B2	A1=B1	A0=B0	1	0	1	1	0	1
A3=B3	A2=B2	A1=B1	A0=B0	1	1	1	1	1	1
A3=B3	A2=B2	A1=B1	A0=B0	1	1	0	1	1	0
A3=B3	A2=B2	A1=B1	A0=B0	0	0	0	0	0	0

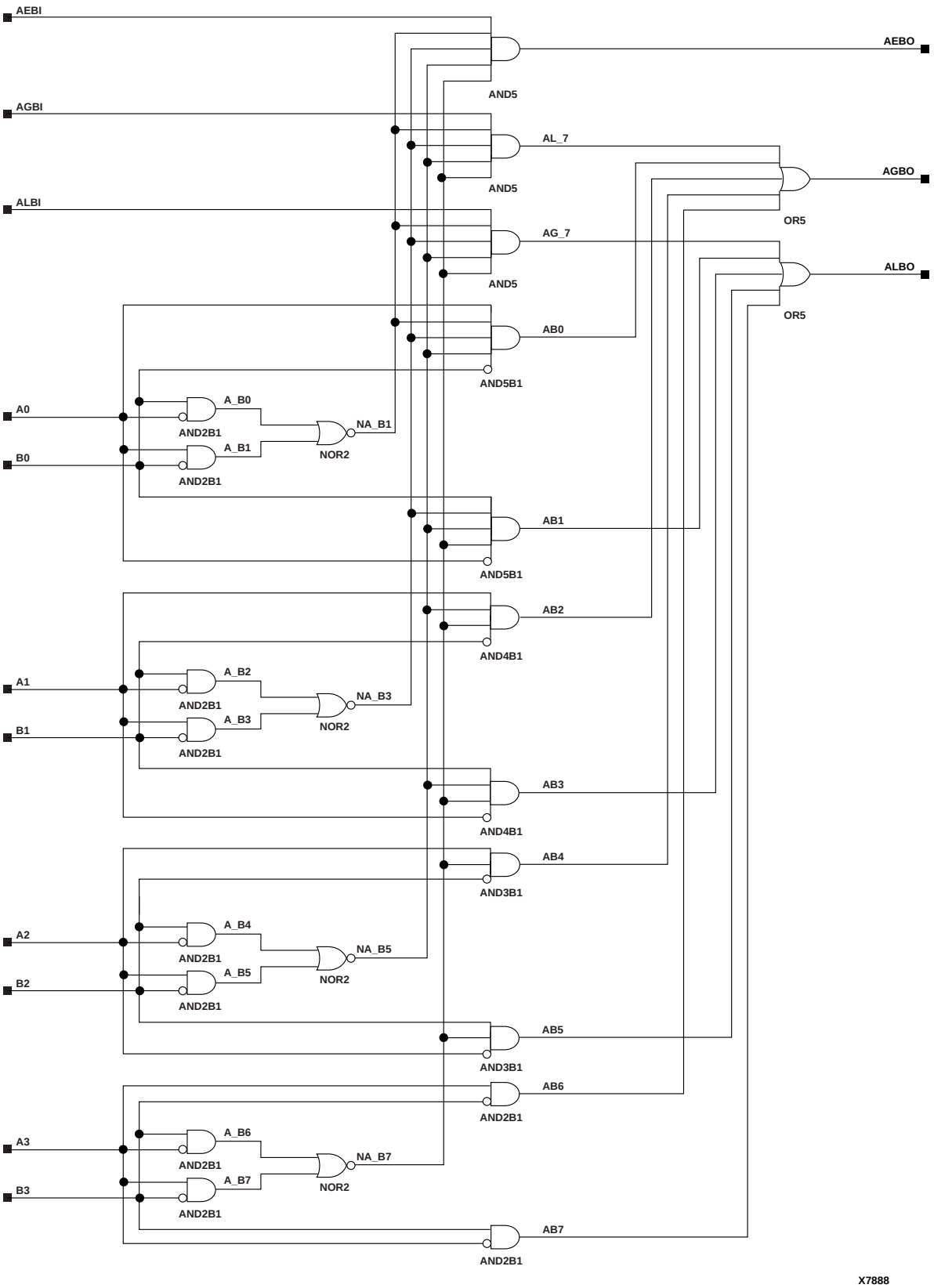


Figure 11-2 X74_L85 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

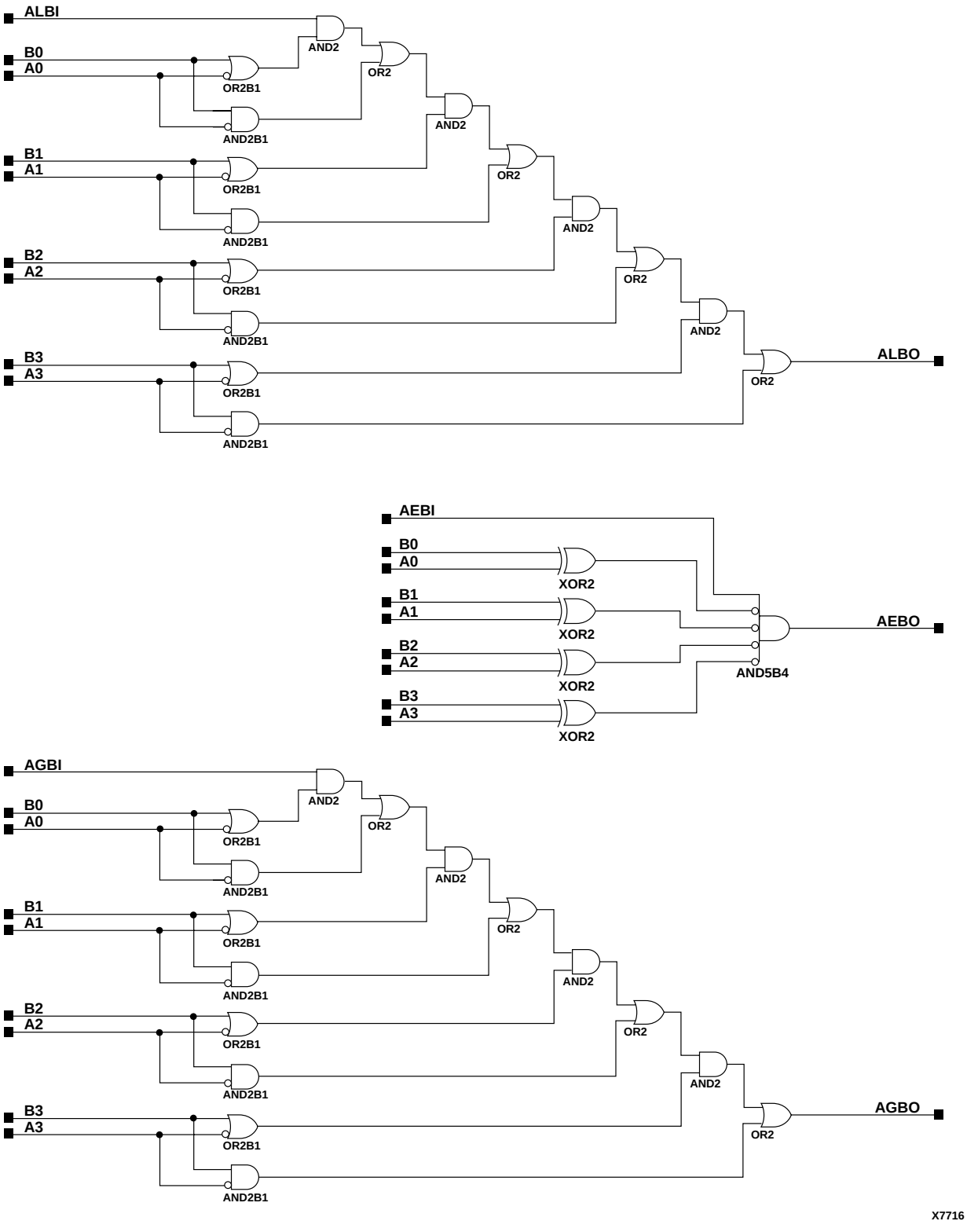
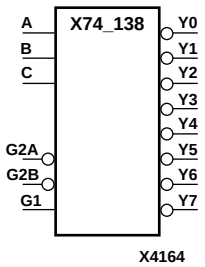


Figure 11-3 X74_L85 Implementation XC9000

X74_138

3- to 8-Line Decoder/Demultiplexer with Active-Low Outputs and Three Enables

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_138 is an expandable decoder / demultiplexer with one active-High enable input (G1), two active-Low enable inputs (G2A and G2B), and eight active-Low outputs (Y7 – Y0). When G1 is High and G2A and G2B are Low, one of the eight active-Low outputs is selected with a 3-bit binary address on address inputs A, B, and C. The non-selected outputs are High. When G1 is Low or when G2A or G2B is High, all outputs are High.

X74_138 can be used as an 8-output active-Low demultiplexer by tying the data input to one of the enable inputs.

Inputs						Outputs							
C	B	A	G1	G2A	G2B	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	1	0	0	1	1	1	1	1	1	1	0
0	0	1	1	0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	1	1	1	1	0	1	1
0	1	1	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	0	0	1	1	0	1	1	1	1	1
1	1	0	1	0	0	1	0	1	1	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1	1	1	1
X	X	X	0	X	X	1	1	1	1	1	1	1	1
X	X	X	X	1	X	1	1	1	1	1	1	1	1
X	X	X	X	X	1	1	1	1	1	1	1	1	1

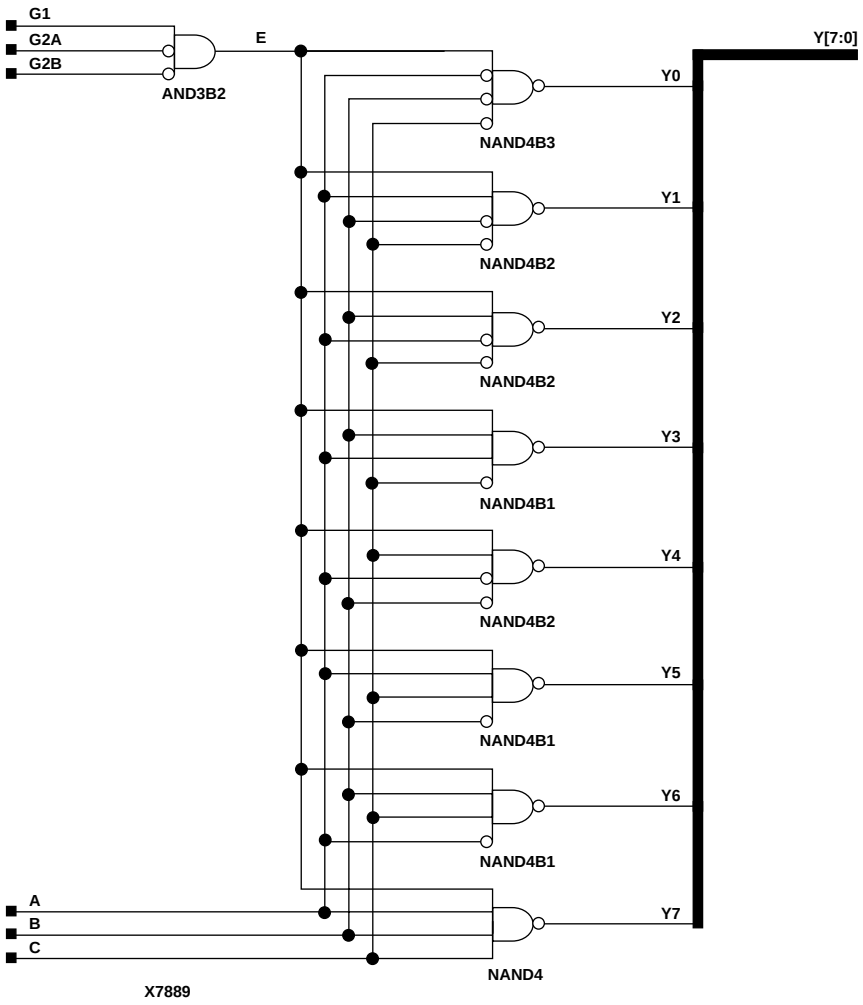
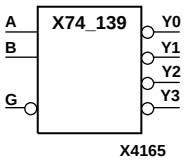


Figure 11-4 X74_138 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_139

2- to 4-Line Decoder/Demultiplexer with Active-Low Outputs and Active-Low Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_139 implements one half of a standard 74139 dual 2- to 4-line decoder / demultiplexer. When the active-Low enable input (G) is Low, one of the four active-Low outputs (Y3 – Y0) is selected with the 2-bit binary address on the A and B address input lines. B is the High-order address bit. The non-selected outputs are High. Also, when G is High all outputs are High.

X74_139 can be used as a 4-output active-Low demultiplexer by tying the data input to G.

Inputs			Outputs			
G	B	A	Y3	Y2	Y1	Y0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	X	X	1	1	1	1

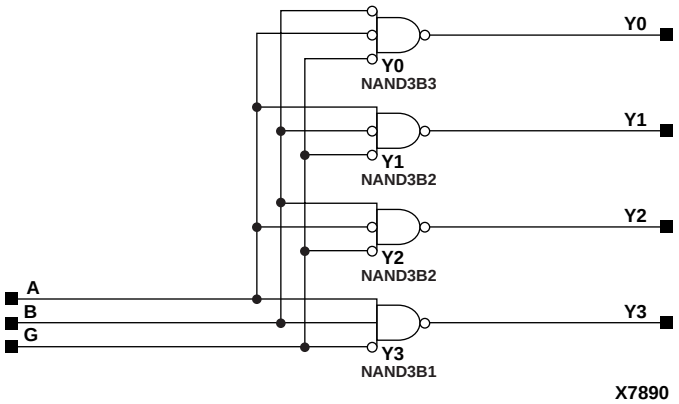
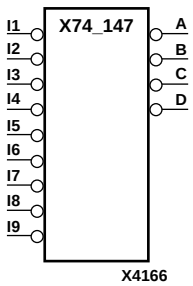


Figure 11-5 X74_139 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_147

10- to 4-Line Priority Encoder with Active-Low Inputs and Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_147 is a 10-line-to-BCD-priority encoder that accepts data from nine active-Low inputs (I9 – I1) and produces a binary-coded decimal (BCD) representation on the four active-Low outputs A, B, C, and D. The data inputs are weighted, so when more than one input is active, only the one with the highest priority is encoded, with I9 having the highest priority. Only nine inputs are provided, because the implied “zero” condition requires no data input. “Zero” is encoded when all data inputs are High.

Inputs									Outputs			
I9	I8	I7	I6	I5	I4	I3	I2	I1	D	C	B	A
1	1	1	1	1	1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	0	X	1	1	0	1
1	1	1	1	1	1	0	X	X	1	1	0	0
1	1	1	1	1	0	X	X	X	1	0	1	1
1	1	1	1	0	X	X	X	X	1	0	1	0
1	1	1	0	X	X	X	X	X	1	0	0	1
1	1	0	X	X	X	X	X	X	1	0	0	0
1	0	X	X	X	X	X	X	X	0	1	1	1
0	X	X	X	X	X	X	X	X	0	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1

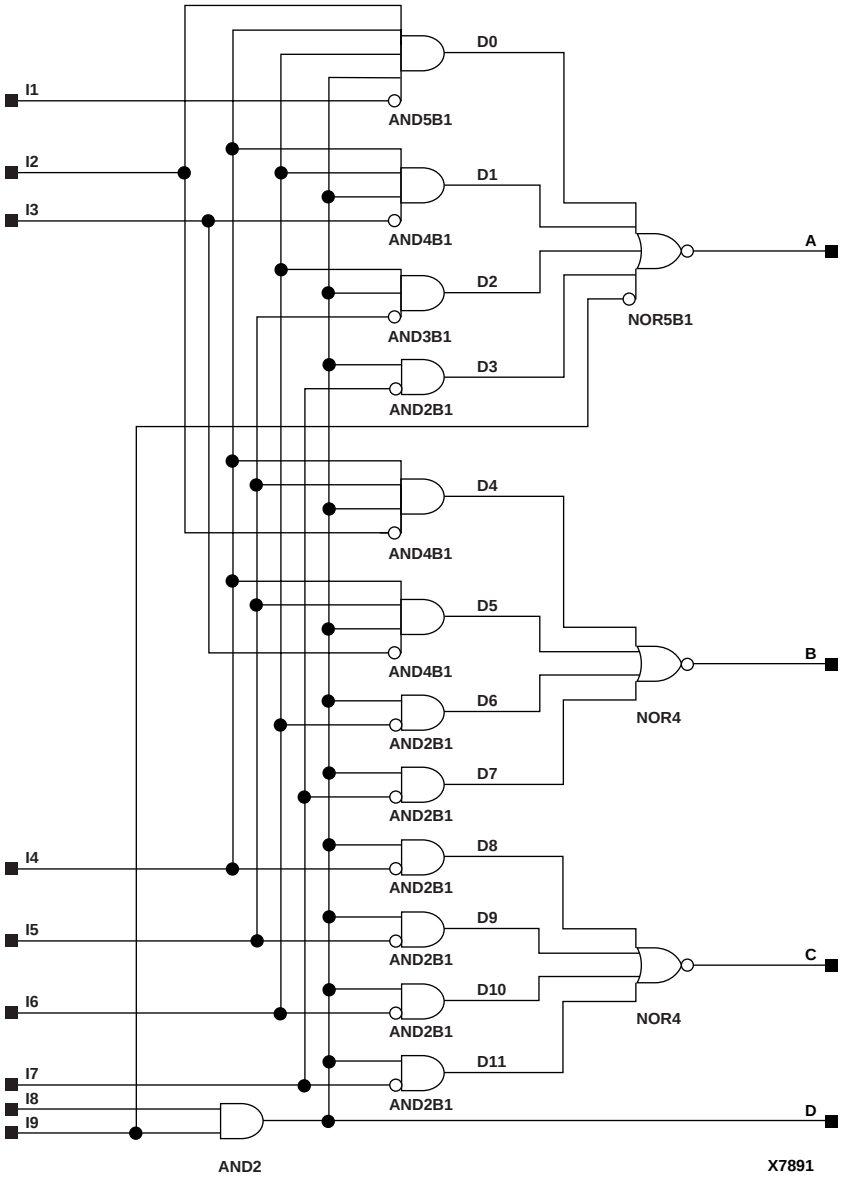
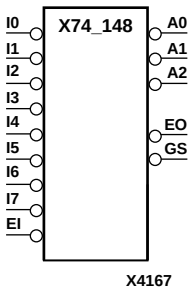


Figure 11-6 X74_147 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_148

8- to 3-Line Cascadable Priority Encoder with Active-Low Inputs and Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_148 8-input priority encoder accepts data from eight active-Low inputs (I7 – I0) and produces a binary representation on the three active-Low outputs (A2 – A0). The data inputs are weighted, so when more than one of the inputs is active, only the input with the highest priority is encoded, I7 having the highest priority. The active-Low group signal (GS) is Low whenever one of the data inputs is Low and the active-Low enable input (EI) is Low.

The active-Low enable input (EI) and active-Low enable output (EO) are used to cascade devices and retain priority control. The EO of the highest priority stage is connected to the EI of the next-highest priority stage. When EI is High, the data outputs and EO are High. When EI is Low, the encoder output represents the highest-priority Low data input, and the EO is High. When EI is Low and all the data inputs are High, the EO output is Low to enable the next-lower priority stage.

Inputs									Outputs				
EI	I7	I6	I5	I4	I3	I2	I1	I0	A2	A1	A0	GS	EO
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0	X	1	1	0	0	1
0	1	1	1	1	1	0	X	X	1	0	1	0	1
0	1	1	1	1	0	X	X	X	1	0	0	0	1
0	1	1	1	0	X	X	X	X	0	1	1	0	1
0	1	1	0	X	X	X	X	X	0	1	0	0	1
0	1	0	X	X	X	X	X	X	0	0	1	0	1
0	0	X	X	X	X	X	X	X	0	0	0	0	1

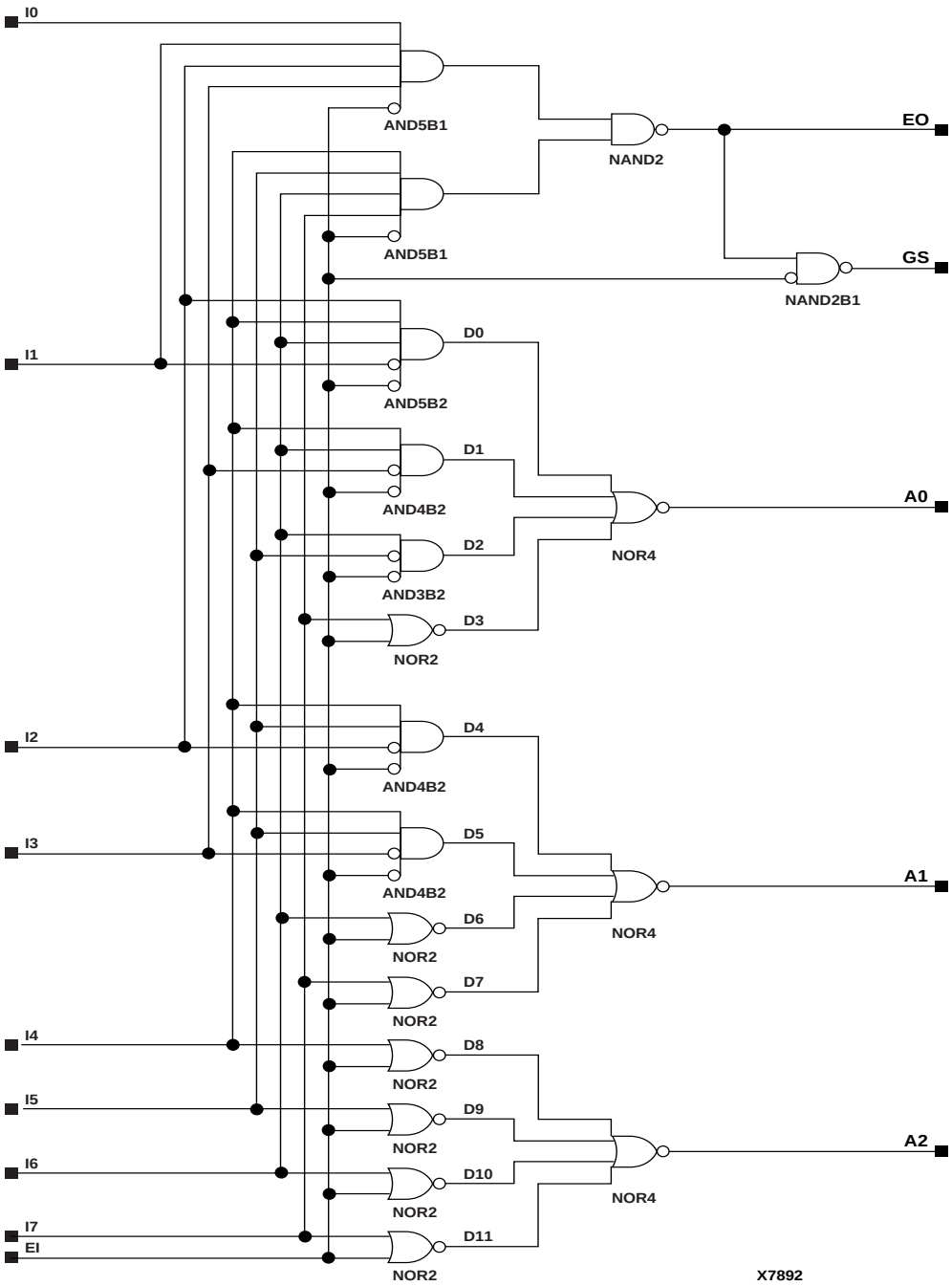
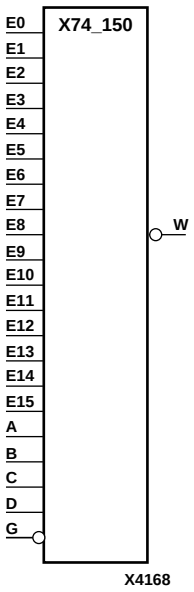


Figure 11-7 X74_148 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_150

16-to-1 Multiplexer with Active-Low Enable and Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low enable input (G) is Low, the X74_150 multiplexer chooses one data bit from 16 sources (E15 – E0) under the control of select inputs A, B, C, and D. The active-Low output (W) reflects the inverse of the selected input, as shown in the truth table. When the enable input (G) is High, the output (W) is High.

Inputs					Outputs
G	D	C	B	A	Selected Input Appears (Inverted) on W
1	X	X	X	X	1
0	0	0	0	0	$\overline{E0}$
0	0	0	0	1	$\overline{E1}$
0	0	0	1	0	$\overline{E2}$
0	0	0	1	1	$\overline{E3}$
0	0	1	0	0	$\overline{E4}$
0	0	1	0	1	$\overline{E5}$
0	0	1	1	0	$\overline{E6}$
0	0	1	1	1	$\overline{E7}$
0	1	0	0	0	$\overline{E8}$
0	1	0	0	1	$\overline{E9}$
0	1	0	1	0	$\overline{E10}$
0	1	0	1	1	$\overline{E11}$
0	1	1	0	0	$\overline{E12}$
0	1	1	0	1	$\overline{E13}$
0	1	1	1	0	$\overline{E14}$
0	1	1	1	1	$\overline{E15}$

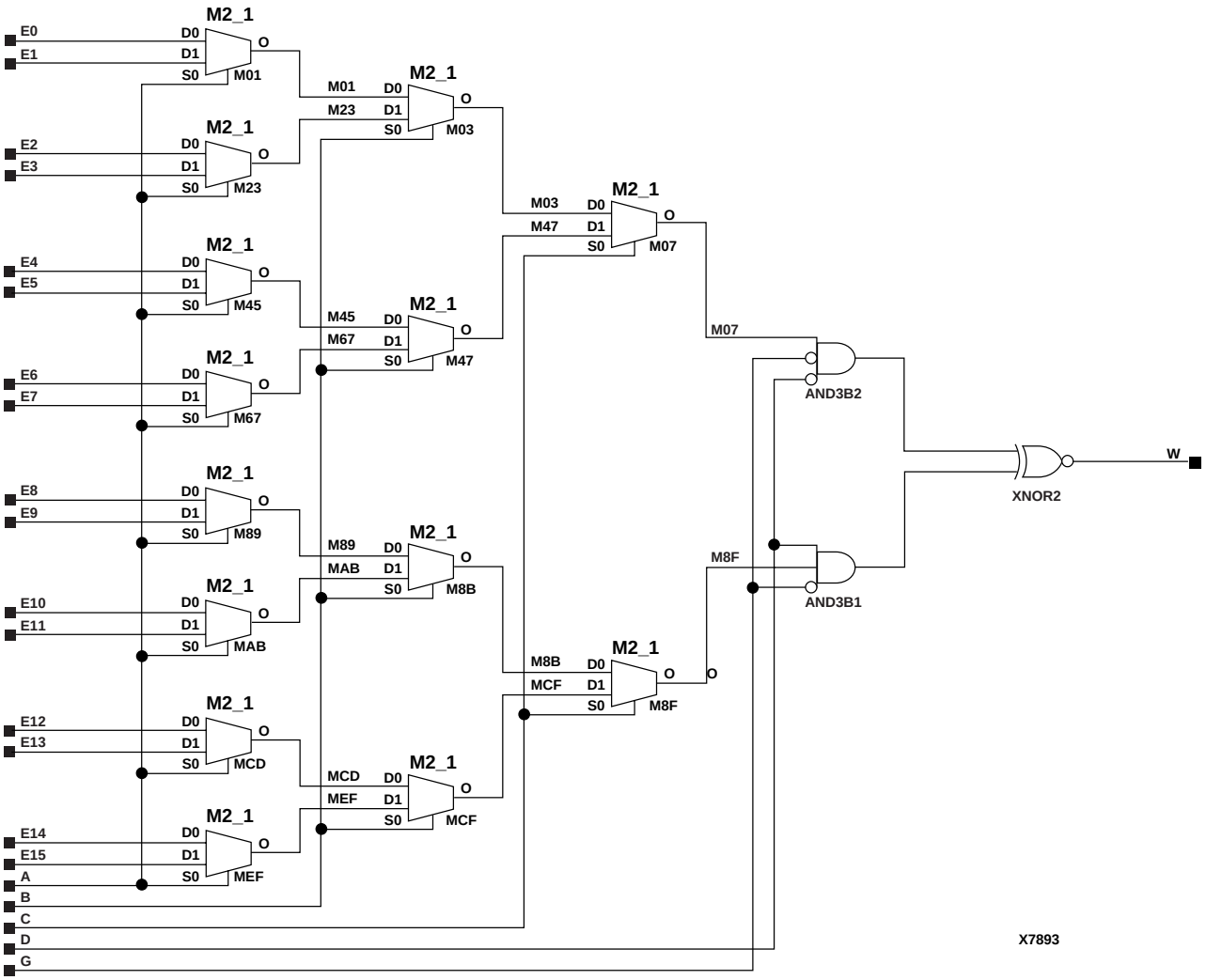
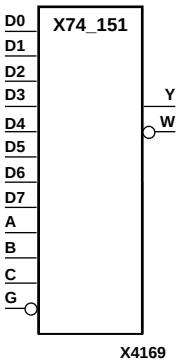


Figure 11-8 X74_150 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_151

8-to-1 Multiplexer with Active-Low Enable and Complementary Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low enable (G) is Low, the X74_151 multiplexer chooses one data bit from eight sources (D7 – D0) under control of the select inputs A, B, and C. The output (Y) reflects the state of the selected input, and the active-Low output (W) reflects the inverse of the selected input as shown in the truth table. When G is High, the Y output is Low, and the W output is High.

Inputs				Outputs	
G	C	B	A	Y	W
1	X	X	X	0	1
0	0	0	0	D0	$\overline{D0}$
0	0	0	1	D1	$\overline{D1}$
0	0	1	0	D2	$\overline{D2}$
0	0	1	1	D3	$\overline{D3}$
0	1	0	0	D4	$\overline{D4}$
0	1	0	1	D5	$\overline{D5}$
0	1	1	0	D6	$\overline{D6}$
0	1	1	1	D7	$\overline{D7}$

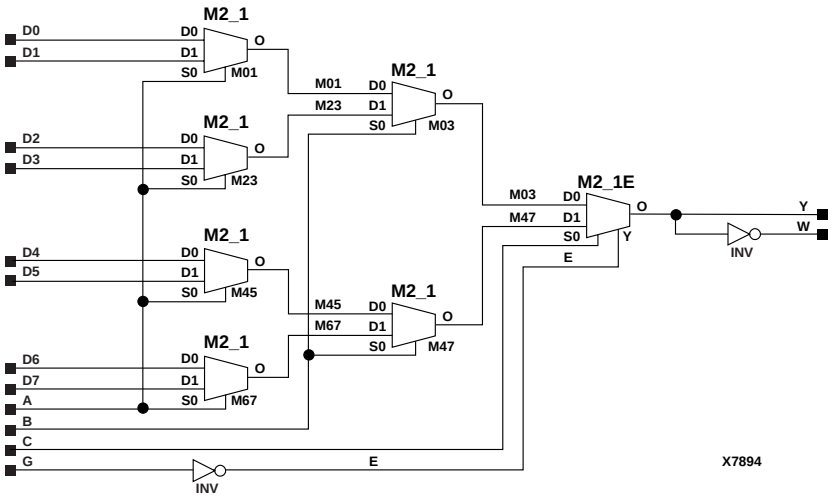
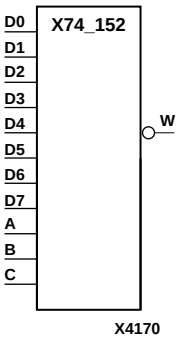


Figure 11-9 X74_151 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_152

8-to-1 Multiplexer with Active-Low Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_152 multiplexer chooses one data bit from eight sources (D7 – D0) under control of the select inputs A, B, and C. The active-Low output (W) reflects the inverse of the selected data input, as shown in the truth table.

Inputs			Outputs
C	B	A	W
0	0	0	$\overline{D0}$
0	0	1	$\overline{D1}$
0	1	0	$\overline{D2}$
0	1	1	$\overline{D3}$
1	0	0	$\overline{D4}$
1	0	1	$\overline{D5}$
1	1	0	$\overline{D6}$
1	1	1	$\overline{D7}$

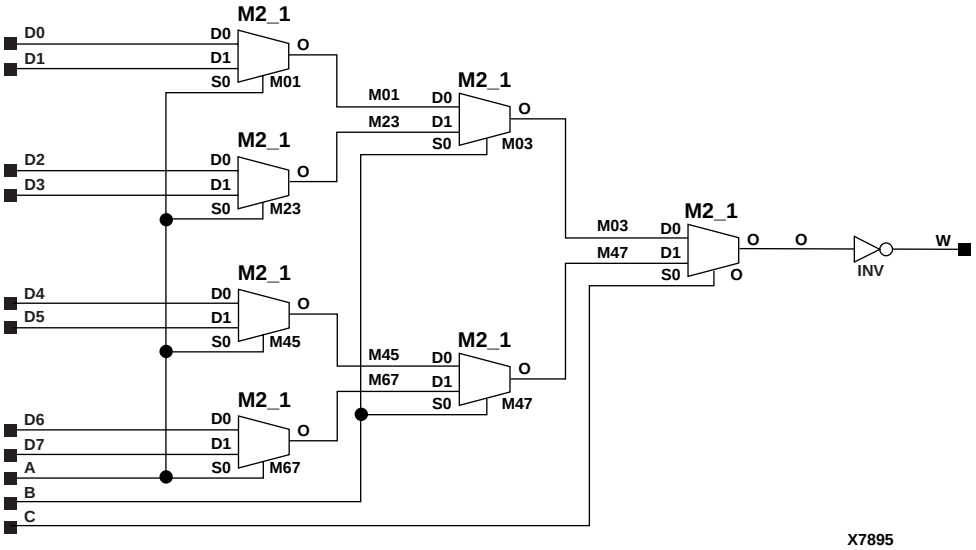
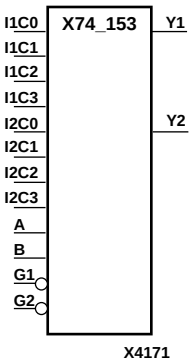


Figure 11-10 X74_152 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_153

Dual 4-to-1 Multiplexer with Active-Low Enables and Common Select Input

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low enable inputs G1 and G2 are Low, the data output Y1, reflects the data input chosen by select inputs A and B from data inputs I1C3 – I1C0. The data output Y2 reflects the data input chosen by select inputs A and B from data inputs I2C3 – I2C0. When G1 or G2 is High, the corresponding output, Y1 or Y2 respectively, is Low.

Inputs				Outputs	
G1	G2	B	A	Y1	Y2
1	1	X	X	0	0
1	0	0	0	0	I2C0
1	0	0	1	0	I2C1
1	0	1	0	0	I2C2
1	0	1	1	0	I2C3
0	1	0	0	I1C0	0
0	1	0	1	I1C1	0
0	1	1	0	I1C2	0
0	1	1	1	I1C3	0
0	0	0	0	I1C0	I2C0
0	0	0	1	I1C1	I2C1
0	0	1	0	I1C2	I2C2
0	0	1	1	I1C3	I2C3

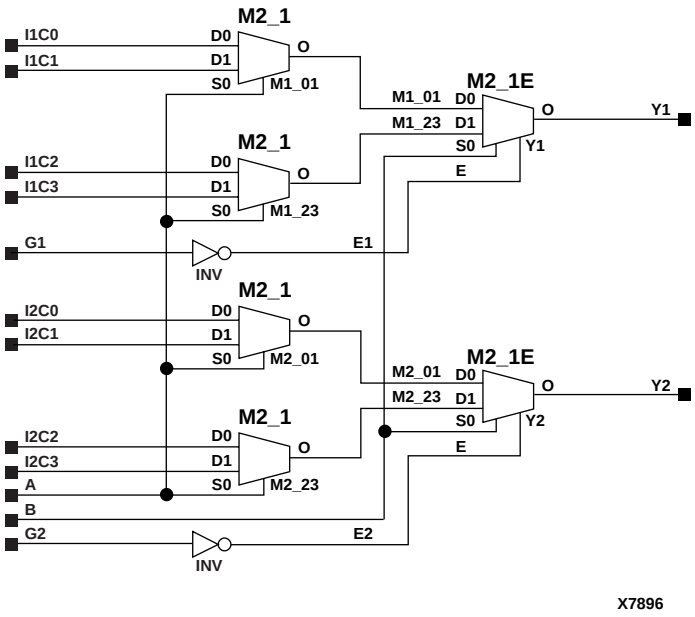
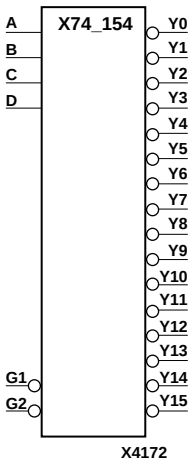


Figure 11-11 X74_153 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_154

4- to 16-Line Decoder/Demultiplexer with Two Enables and Active-Low Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low enable inputs G1 and G2 of the X74_154 decoder / demultiplexer are Low, one of 16 active-Low outputs, Y15 – Y0, is selected under the control of four binary address inputs A, B, C, and D. The non-selected inputs are High. Also, when either input G1 or G2 is High, all outputs are High.

The X74_154 can be used as a 16-to-1 demultiplexer by tying the data input to one of the G inputs and tying the other G input Low.

Inputs						Outputs							
G1	G2	D	C	B	A	Y15	Y14	Y13	Y12	Y11	Y10	Y9	... Y0
1	X	X	X	X	X	1	1	1	1	1	1	1	... 1
X	1	X	X	X	X	1	1	1	1	1	1	1	... 1
0	0	1	1	1	1	0	1	1	1	1	1	1	... 1
0	0	1	1	1	0	1	0	1	1	1	1	1	... 1
0	0	1	1	0	1	1	1	0	1	1	1	1	... 1
-	-	-	-	-	-	-	-	-	-	-	-	-	... -
-	-	-	-	-	-	-	-	-	-	-	-	-	... -
-	-	-	-	-	-	-	-	-	-	-	-	-	... -
0	0	0	0	0	0	1	1	1	1	1	1	1	... 0

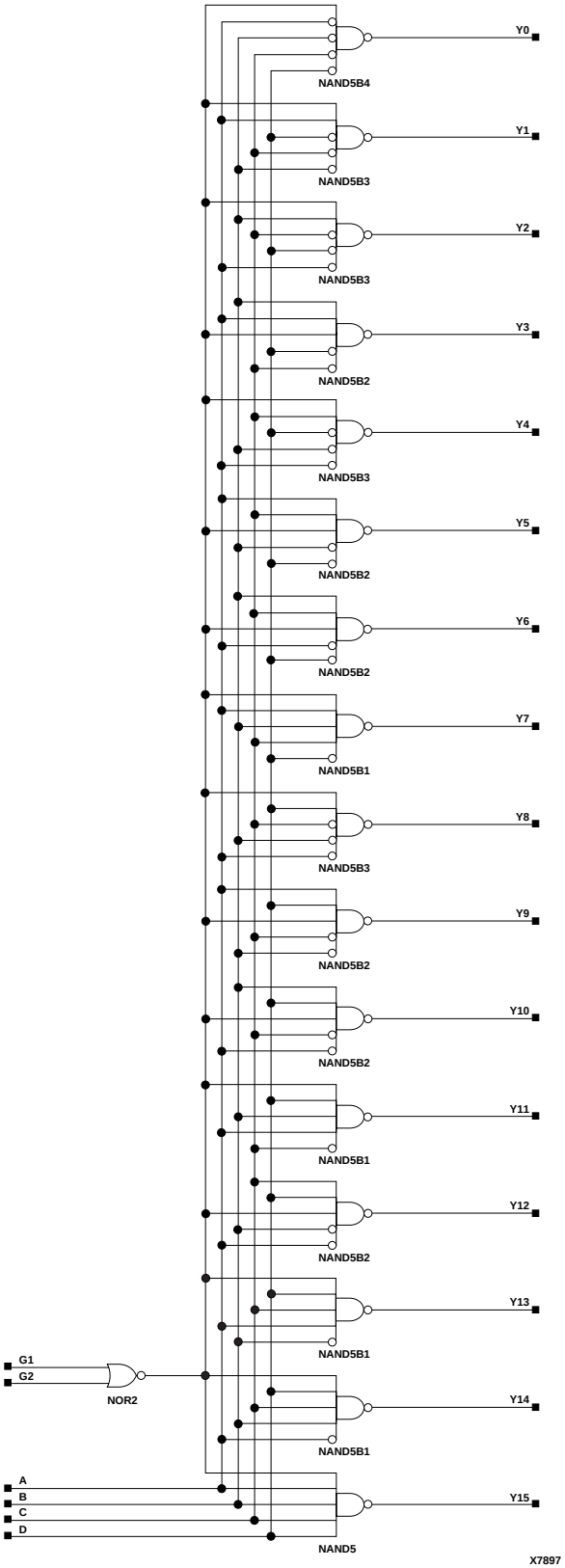
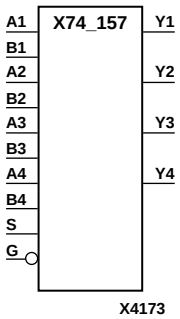


Figure 11-12 X74_154 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_157

Quadruple 2-to-1 Multiplexer with Common Select and Active-Low Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low enable input (G) of the X74_157 multiplexer is Low, a 4-bit word is selected from one of two sources (A3 – A0 or B3 – B0) under the control of the select input (S) and is reflected on the four outputs (Y4 – Y1). When S is Low, the outputs reflect A3 – A0; when S is High, the outputs reflect B3 – B0. When G is High, the outputs are Low.

Inputs				Outputs
G	S	B	A	Y
1	X	X	X	0
0	1	1	X	1
0	1	0	X	0
0	0	X	1	1
0	0	X	0	0

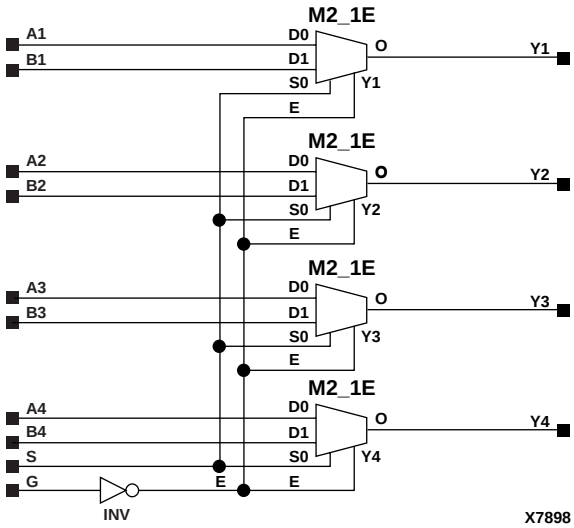
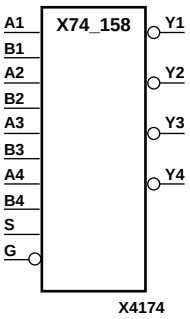


Figure 11-13 X74_157 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_158

Quadruple 2-to-1 Multiplexer with Common Select, Active-Low Enable, and Active-Low Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low enable (G) of the X74_158 multiplexer is Low, a 4-bit word is selected from one of two sources (A3 – A0 or B3 – B0) under the control of the common select input (S). The inverse of the selected word is reflected on the active-Low outputs (Y4 – Y1). When S is Low, $\overline{A3}$ – $\overline{A0}$ appear on the outputs; when S is High, $\overline{B3}$ – $\overline{B0}$ appear on the outputs. When G is High, the outputs are High.

Inputs				Outputs
G	S	B	A	Y
1	X	X	X	1
0	1	1	X	0
0	1	0	X	1
0	0	X	1	0
0	0	X	0	1

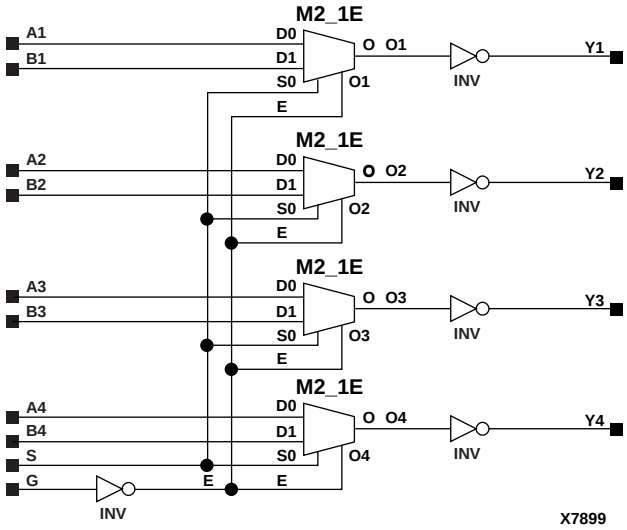
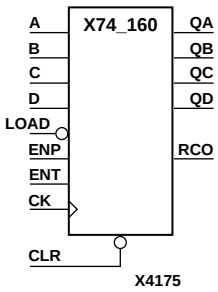


Figure 11-14 X74_158 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_160

4-Bit BCD Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_160 is a 4-stage, 4-bit, synchronous, loadable, resettable, cascadable, binary-coded decimal (BCD) counter. The active-Low asynchronous clear (CLR), when Low, overrides all other inputs and resets the data (QD, QC, QB, QA) and ripple carry-out (RCO) outputs Low. When the active-Low load enable input (LOAD) is Low and CLR is High, parallel clock enable (ENP), and trickle clock enable (ENT) are overridden and data on inputs A, B, C, and D are loaded into the counter during the Low-to-High clock transition. The data outputs (QD, QC, QB, QA) increment when ENP, ENT LOAD, and CLR are High during the Low-to-High clock transition. The counter ignores clock transitions when ENP or ENT are Low and LOAD is High. RCO is High when QD, QA, and ENT are High and QC and QB are Low.

Inputs						Outputs	
CLR	LOAD	ENP	ENT	D – A	CK	QD – QA	RCO
0	X	X	X	X	X	0	0
1	0	X	X	D – A	↑	d – a	RCO
1	1	0	X	X	X	No Chg	RCO
1	1	X	0	X	X	No Chg	0
1	1	1	1	X	↑	Inc	RCO

RCO = (QD • $\overline{QC}$ • $\overline{QB}$ • QA • ENT)

d – a = state of referenced input one set-up time prior to active clock transition

Carry-Lookahead Design

The carry-lookahead design allows cascading of large counters without extra gating. Both ENT and ENP must be High to count. ENT is fed forward to enable RCO, which produces a High output pulse with the approximate duration of the QA output. The following figure illustrates a carry-lookahead design.

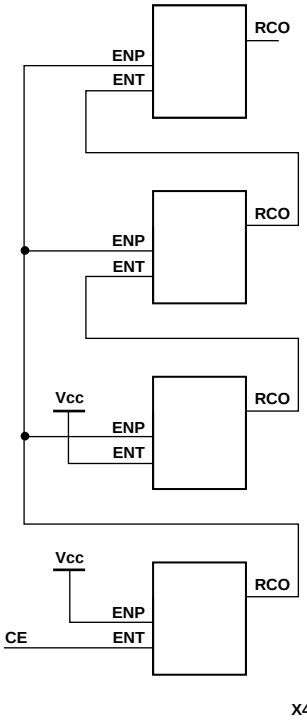


Figure 11-15 Carry-Lookahead Design

The RCO output of the first stage of the ripple carry is connected to the ENP input of the second stage and all subsequent stages. The RCO output of the second stage and all subsequent stages is connected to the ENT input of the next stage. The ENT of the second stage is always enabled / tied to VCC. CE is always connected to the ENT input of the first stage. This cascading method allows the first stage of the ripple carry to be built as a prescaler. In other words, the first stage is built to count very fast.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles.

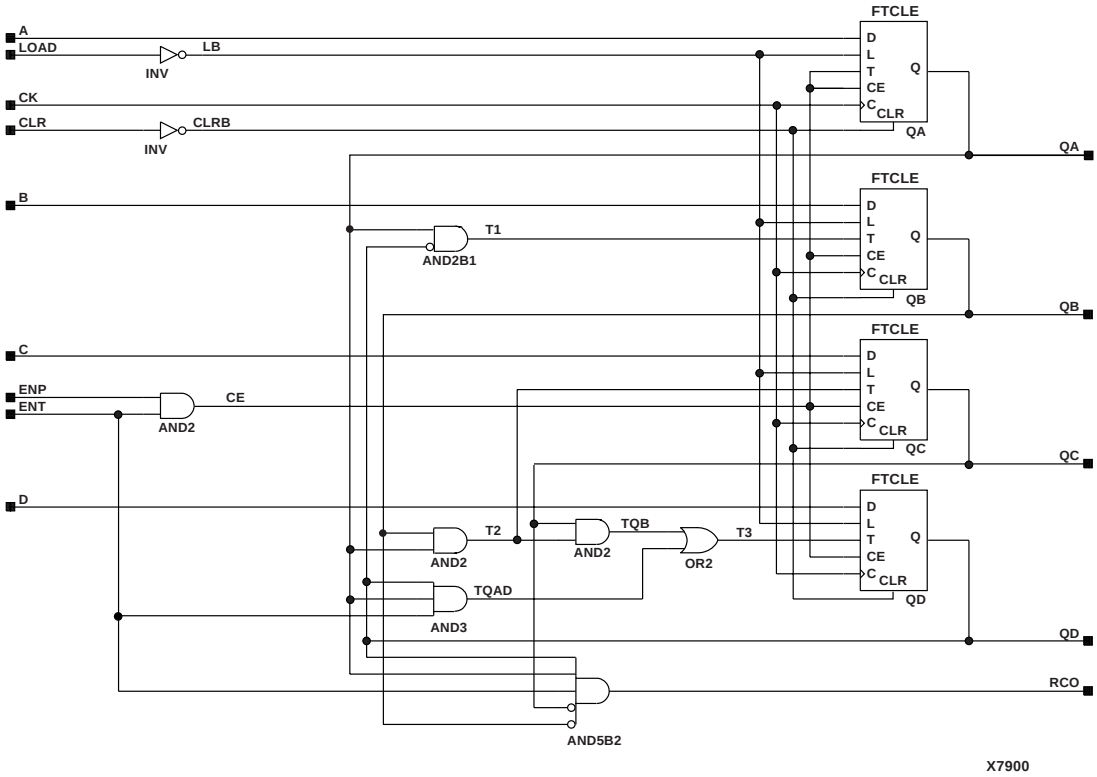


Figure 11-16 X74_160 Implementation XC3000

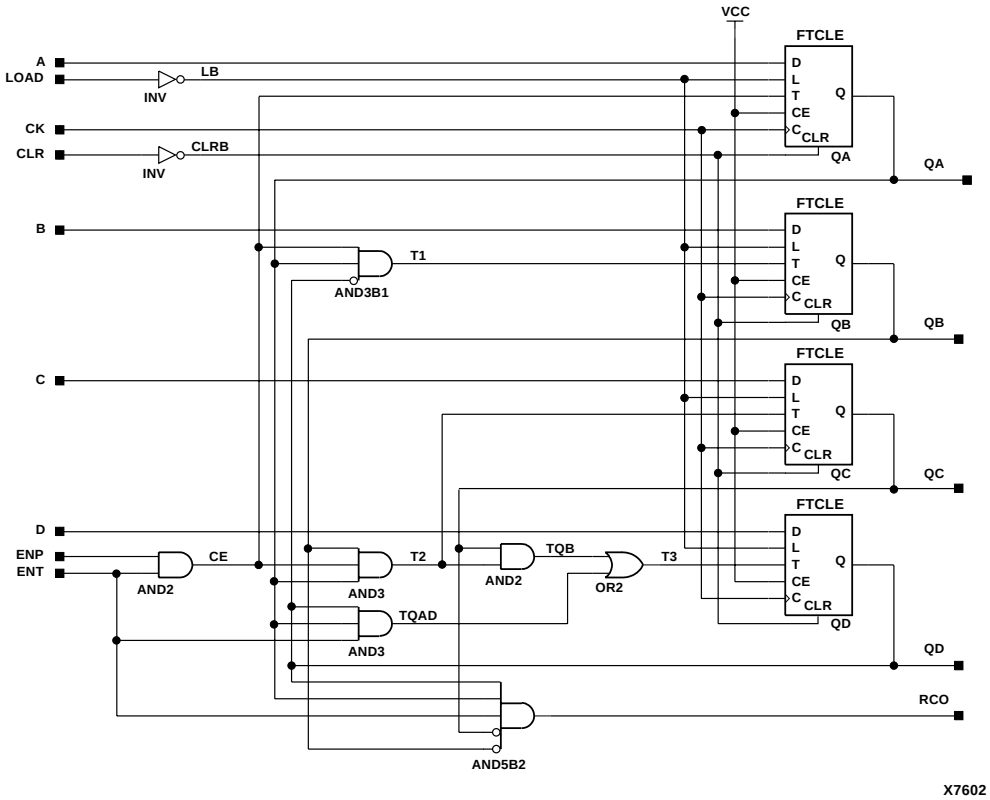


Figure 11-17 X74_160 Implementation XC4000E, XC4000X, XC5200, Spartan, SpartanXL

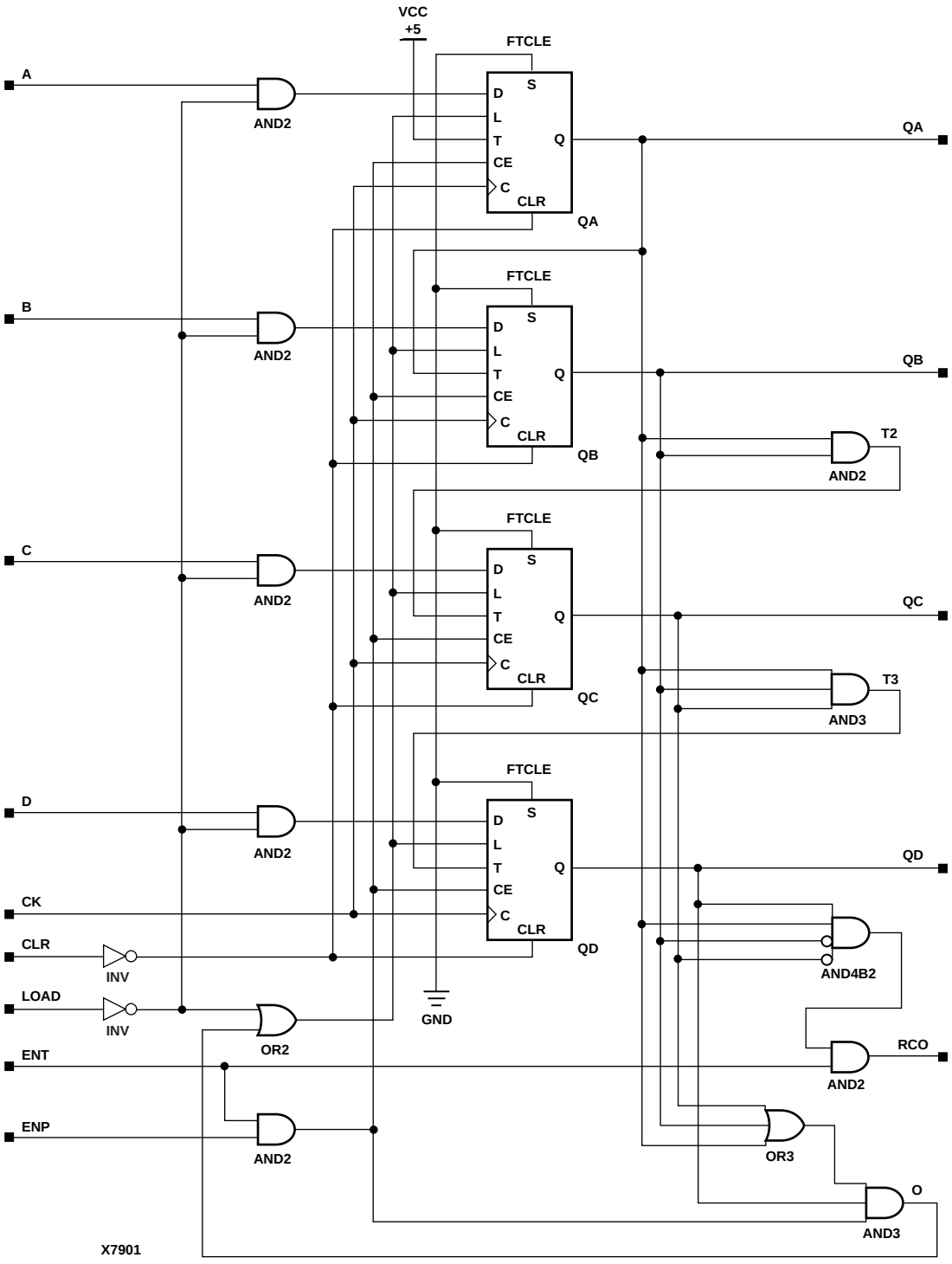
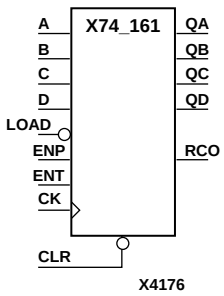


Figure 11-18 X74_160 Implementation XC9000

X74_161

4-Bit Binary Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_161 is a 4-stage, 4-bit, synchronous, loadable, resettable, cascadable binary counter. The active-Low asynchronous clear (CLR), when Low, overrides all other inputs and resets the data outputs (QD, QC, QB, QA) and the ripple carry-out output (RCO) Low. When the active-Low load enable (LOAD) is Low and CLR is High, parallel clock enable (ENP) and trickle clock enable (ENT) are overridden and the data on inputs A, B, C, and D is loaded into the counter during the Low-to-High clock (CK) transition. The data outputs (QD, QC, QB, QA) increment when LOAD, ENP, ENT, and CLR are High during the Low-to-High clock transition. The counter ignores clock transitions when LOAD is High and ENP or ENT are Low. RCO is High when QD – QA and ENT are High.

The carry-lookahead design accommodates large counters without extra gating. Refer to “Carry-Lookahead Design” in the “X74_160” section for more information.

Inputs						Outputs	
CLR	LOAD	ENP	ENT	D – A	CK	QD – QA	RCO
0	X	X	X	X	X	0	0
1	0	X	X	D – A	↑	d – a	RCO
1	1	0	X	X	X	No Chg	RCO
1	1	X	0	X	X	No Chg	0
1	1	1	1	X	↑	Inc	RCO

RCO = (QD • QC • QB • QA • ENT)

d – a = state of referenced input one setup time prior to active clock transition

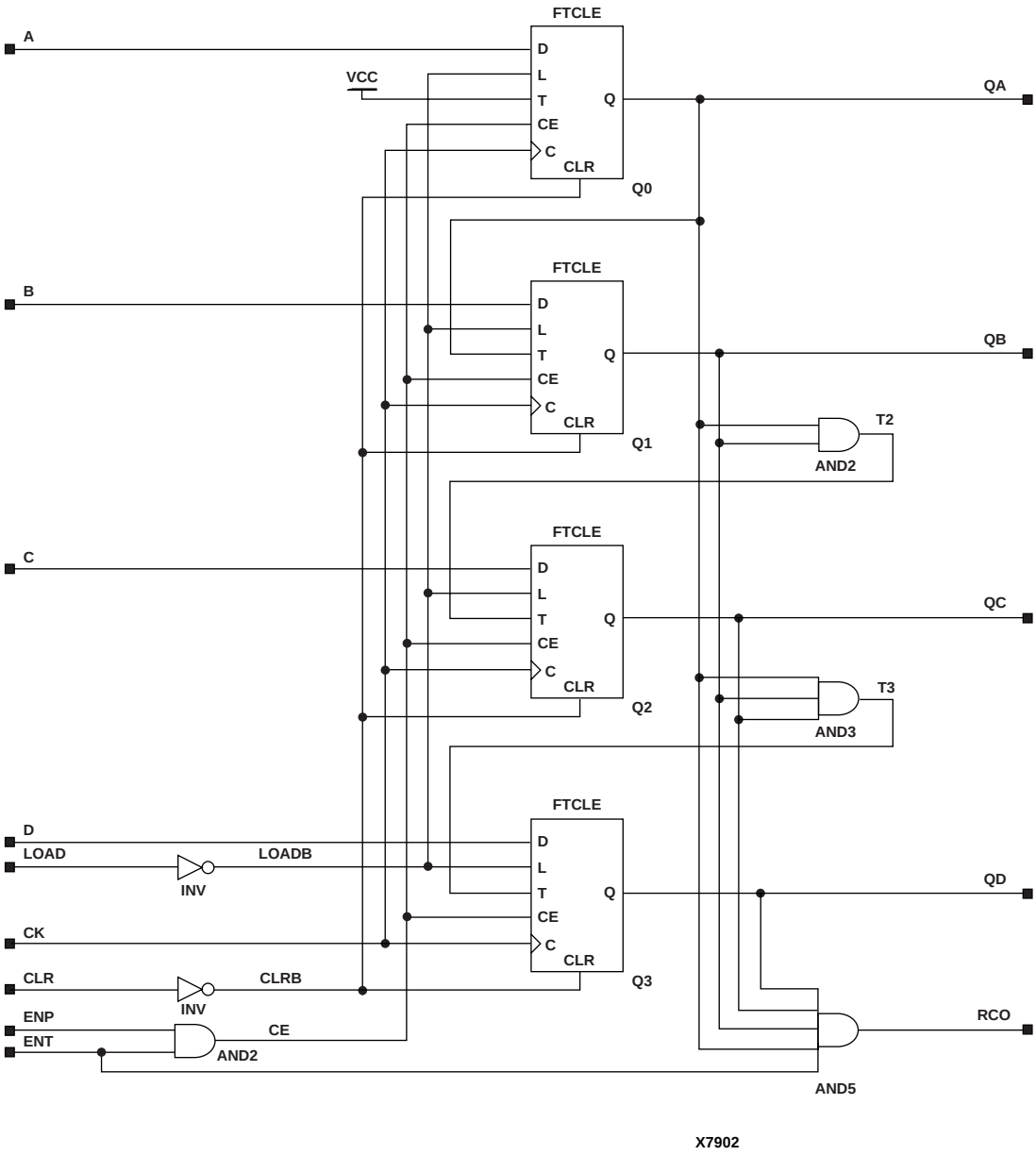


Figure 11-19 X74_161 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

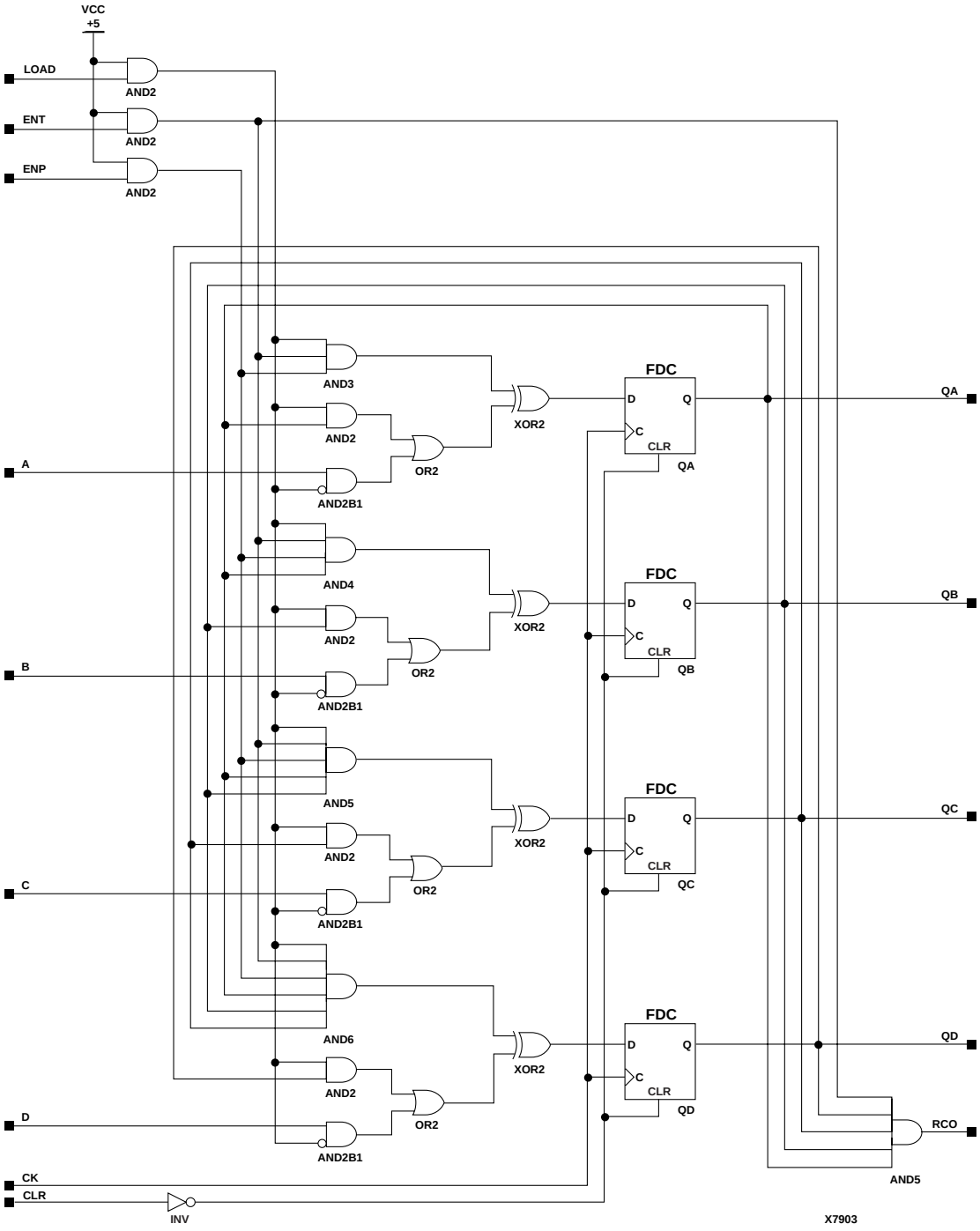
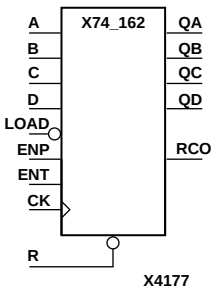


Figure 11-20 X74_161 Implementation XC9000

X74_162

4-Bit BCD Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_162 is a 4-stage, 4-bit, synchronous, loadable, resettable, cascadable binary-coded decimal (BCD) counter. The active-Low synchronous reset (R), when Low, overrides all other inputs and resets the data (QD, QC, QB, QA) and ripple carry-out (RCO) outputs Low during the Low-to-High clock (CK) transition. When the active-Low load enable input (LOAD) is Low and R is High, parallel clock enable (ENP) and trickle clock enable (ENT) are overridden and data on inputs A, B, C, and D is loaded into the counter during the Low-to-High clock transition. The data outputs (QD, QC, QB, QA) increment when ENP, ENT, LOAD, and R are High during the Low-to-High clock transition. The counter ignores clock transitions when ENP or ENT are Low and LOAD is High. RCO is High when QD, QA, and ENT are High and QC and QB are Low.

The carry-lookahead design accommodates cascading large counters without extra gating. Refer to “Carry-Lookahead Design” in the “X74_160” section for more information.

Inputs						Outputs	
R	LOAD	ENP	ENT	D – A	CK	QD – QA	RCO
0	X	X	X	X	↑	0	0
1	0	X	X	D – A	↑	d – a	RCO
1	1	0	X	X	X	No Chg	RCO
1	1	X	0	X	X	No Chg	0
1	1	1	1	X	↑	Inc	RCO

RCO = (QD • $\overline{QC}$ • $\overline{QB}$ • QA • ENT)

d – a = state of referenced input one setup time prior to active clock transition

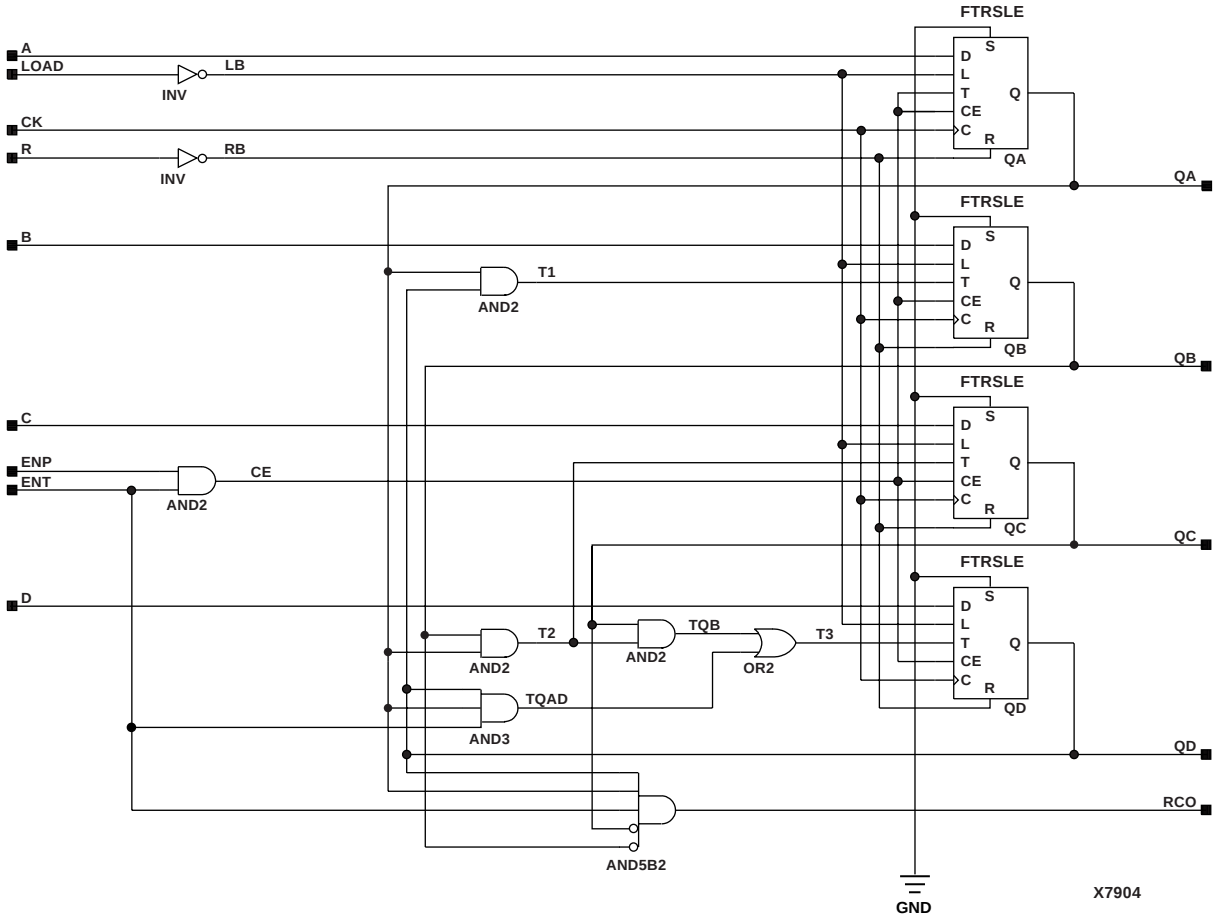


Figure 11-21 X74_162 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

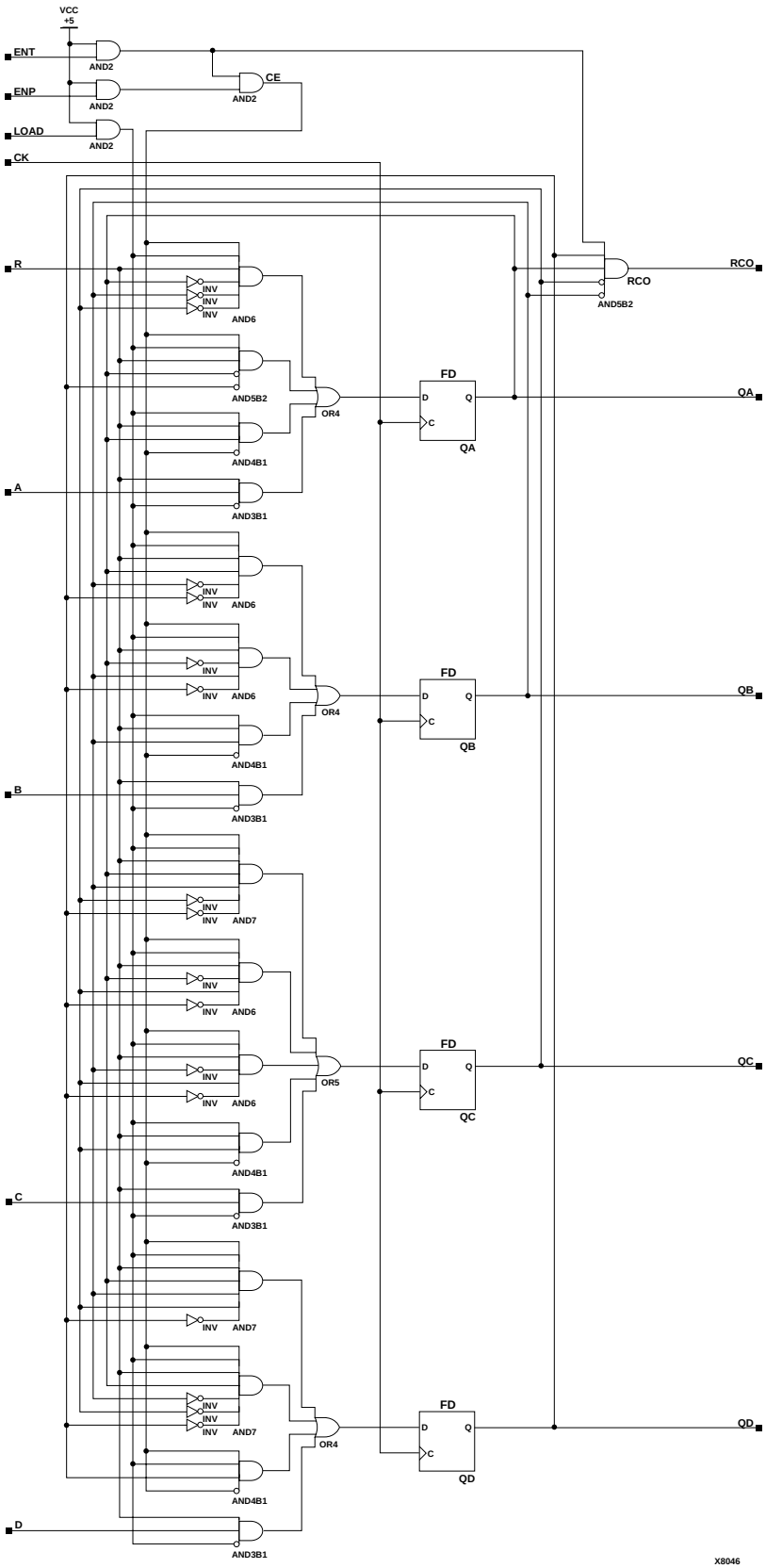
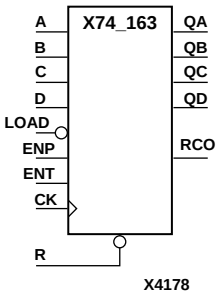


Figure 11-22 X74_162 Implementation XC9000

X74_163

4-Bit Binary Counter with Parallel and Trickle Enables, Active-Low Load Enable, and Synchronous Reset

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_163 is a 4-stage, 4-bit, synchronous, loadable, resettable, cascadable binary counter. The active-Low synchronous reset (R), when Low, overrides all other inputs and resets the data outputs (QD, QC, QB, QA) and the ripple carry-out output (RCO) Low during the Low-to-High clock (CK) transition. When the active-Low load enable (LOAD) is Low and R is High, parallel clock enable (ENP) and trickle clock enable (ENT) are overridden and the data on inputs (A, B, C, D) is loaded into the counter during the Low-to-High clock (CK) transition. The outputs (QD, QC, QB, QA) increment when LOAD, ENP, ENT, and R are High during the Low-to-High clock transition. The counter ignores clock transitions when LOAD is High and ENP or ENT are Low; RCO is High when QD – QA and ENT are High.

The carry-lookahead design accommodates large counters without extra gating. Refer to “Carry-Lookahead Design” in the “X74_160” section for more information.

Inputs						Outputs	
R	LOAD	ENP	ENT	D – A	CK	QD – QA	RCO
0	X	X	X	X	↑	0	0
1	0	X	X	D – A	↑	d – a	RCO
1	1	0	X	X	X	No Chg	RCO
1	1	X	0	X	X	No Chg	0
1	1	1	1	X	↑	Inc	RCO

RCO = (QD • QC • QB • QA • ENT)

d – a = state of referenced input one setup time prior to active clock transition

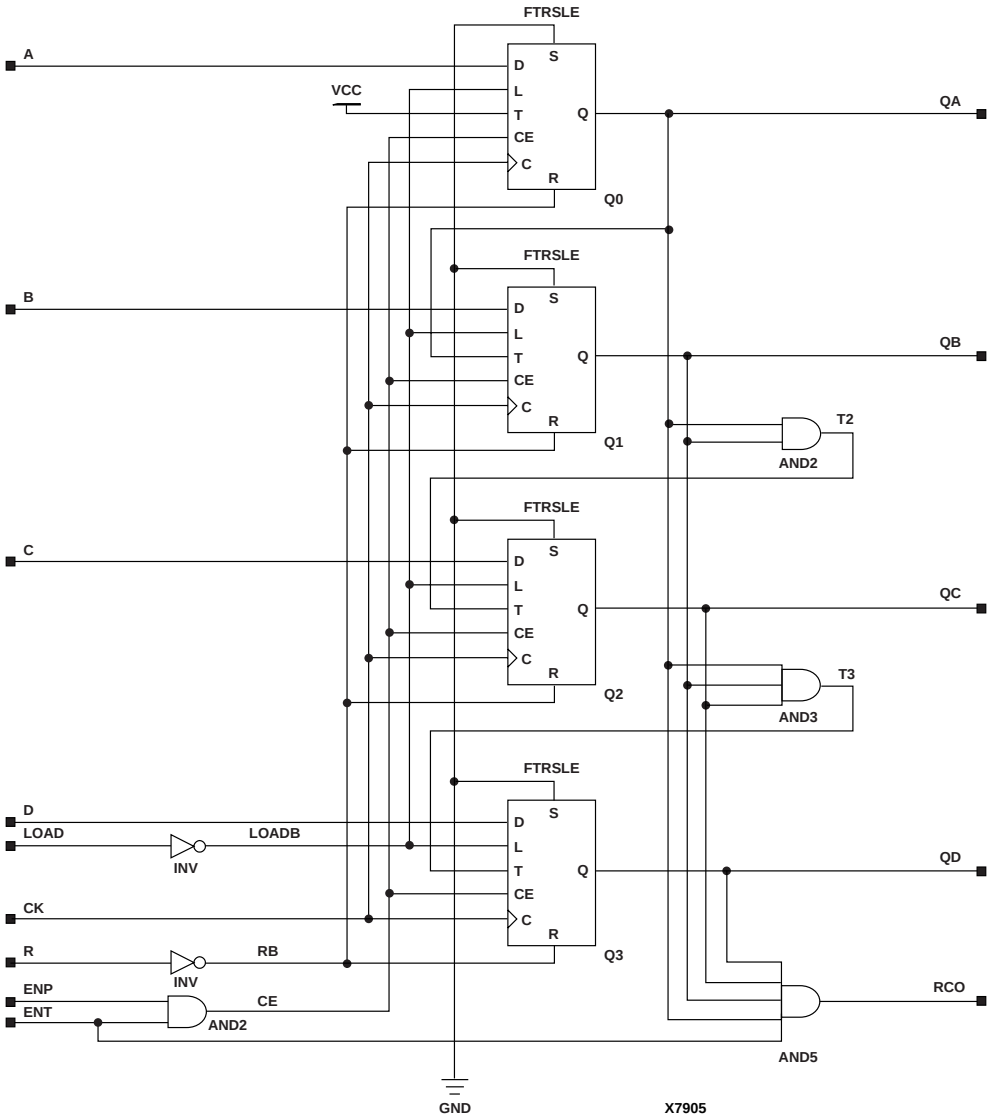


Figure 11-23 X74_163 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

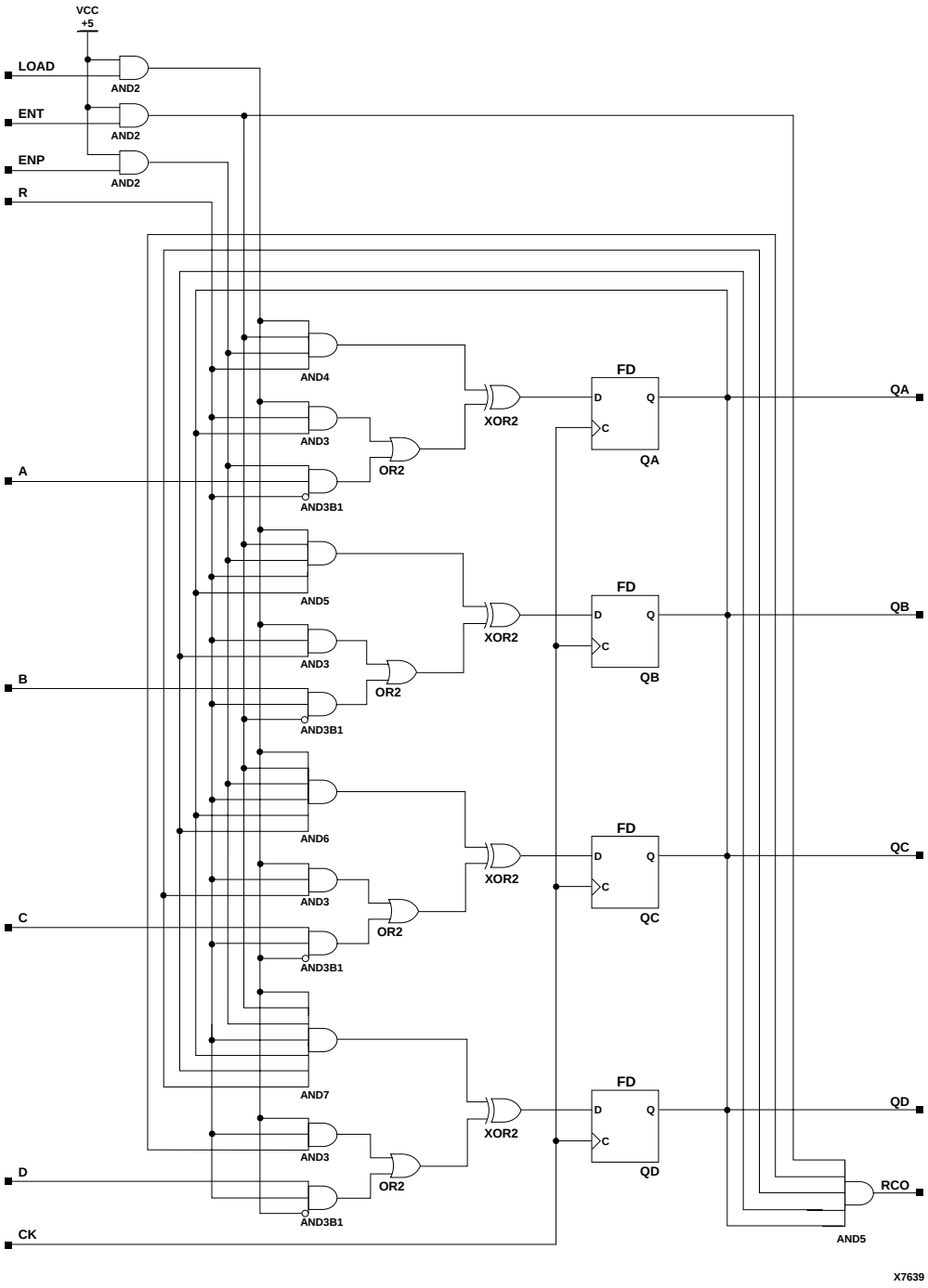
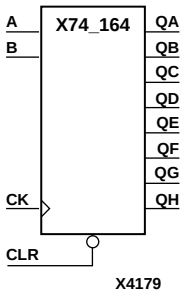


Figure 11-24 X74_163 Implementation XC9000

X74_164

8-Bit Serial-In Parallel-Out Shift Register with Active-Low Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_164 is an 8-bit, serial input (A and B), parallel output (QH – QA) shift register with an active-Low asynchronous clear (CLR) input. The asynchronous CLR, when Low, overrides the clock input and sets the data outputs (QH – QA) Low. When CLR is High, the AND function of the two data inputs (A and B) is loaded into the first bit of the shift register during the Low-to-High clock (CK) transition and appears on the QA output. During subsequent Low-to-High clock transitions, with CLR High, the data is shifted to the next-highest bit position as new data is loaded into QA (A and B→QA, QA→QB, QB→QC, and so forth).

Registers can be cascaded by connecting the QH output of one stage to the A input of the next stage, by tying B High, and by connecting the clock and CLR inputs in parallel.

Inputs				Outputs	
CLR	A	B	CK	QA	QB – QH
0	X	X	X	0	0
1	1	1	↑	1	qA – qG
1	0	X	↑	0	qA – qG
1	X	0	↑	0	qA – qG

qA – qG = state of referenced output one setup time prior to active clock transition

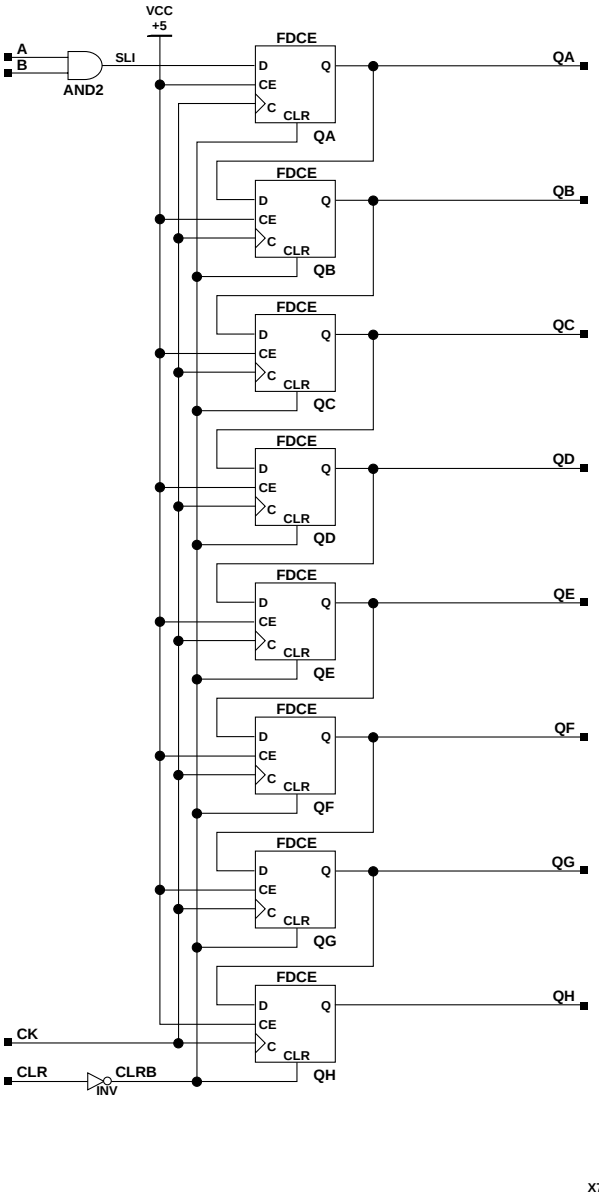
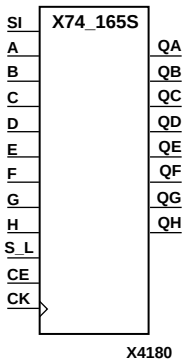


Figure 11-25 X74_164 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_165S

8-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_165S is an 8-bit shift register with serial-input (SI), parallel- inputs (H – A), parallel-outputs (QH – QA), and two control inputs – clock enable (CE) and active-Low shift / load enable (S_L). When S_L is Low, data on the H – A inputs is loaded into the corresponding QH – QA bits of the register on the Low-to-High clock (CK) transition. When CE and S_L are High, data on the SI input is loaded into the first bit of the register during the Low-to-High clock transition. During subsequent Low-to-High clock transitions, with CE and S_L High, the data is shifted to the next-highest bit position (shift right) as new data is loaded into QA (SI→ QA, QA→QB, QB→QC, and so forth). The register ignores clock transitions when CE is Low and S_L is High.

Registers can be cascaded by connecting the QH output of one stage to the SI input of the next stage and connecting clock, CE, and S_L inputs in parallel.

Inputs					Outputs	
S_L	CE	SI	A – H	CK	QA	QB – QH
0	X	X	A – H	↑	qa	qb – qh
1	0	X	X	X	No Chg	No Chg
1	1	SI	X	↑	si	qA – qG

si = state of referenced input one setup time prior to active clock transition
qn = state of referenced output one setup time prior to active clock transition

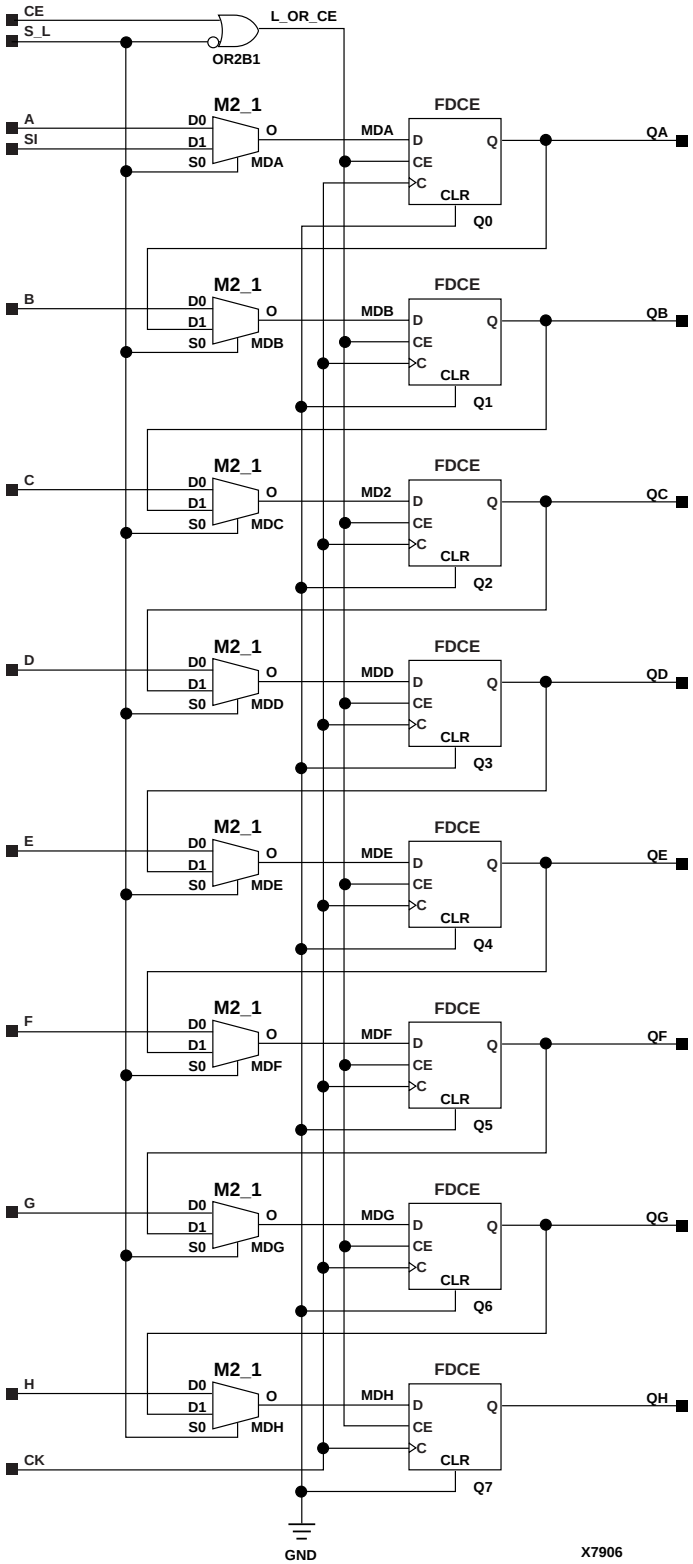
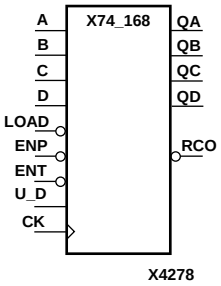


Figure 11-26 X74_165S Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_168

4-Bit BCD Bidirectional Counter with Parallel and Trickle Clock Enables and Active-Low Load Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_168 is a 4-stage, 4-bit, synchronous, loadable, cascadable, bidirectional binary-coded-decimal (BCD) counter. The data on the D – A inputs is loaded into the counter when the active-Low load enable (LOAD) is Low during the Low-to-High clock (CK) transition. The LOAD input, when Low, has priority over parallel clock enable (ENP), trickle clock enable (ENT), and the bidirectional (U_D) control. The outputs (QD – QA) increment when U_D and LOAD are High and ENP and ENT are Low during the Low-to-High clock transition. The outputs decrement when LOAD is High and ENP, ENT, and U_D are Low during the Low-to-High clock transition. The counter ignores clock transitions when LOAD and either ENP or ENT are High.

Inputs						Outputs	
LOAD	ENP	ENT	U_D	A – D	CK	QA – QD	RCO
0	X	X	X	A – D	↑	qa – qd	RCO
1	0	0	1	X	↑	Inc	RCO
1	0	0	0	X	↑	Dec	RCO
1	1	0	X	X	X	No Chg	RCO
1	X	1	X	X	X	No Chg	1

$RCO = (Q3 \bullet \overline{Q2} \bullet \overline{Q1} \bullet Q0 \bullet U\_D \bullet \overline{ENT}) + (\overline{Q3} \bullet \overline{Q2} \bullet \overline{Q1} \bullet \overline{Q0} \bullet \overline{U\_D} \bullet \overline{ENT})$

qa – qd = state of referenced input one setup time prior to active clock transition

The active-Low ripple carry-out output (RCO) is Low when QD, QA, and U_D are High and QC, QB, and ENT are Low. RCO is also Low when all outputs, ENT and U_D are Low. The following figure illustrates a carry-lookahead design.

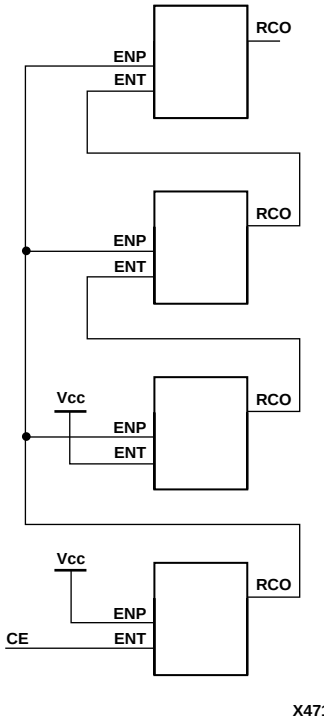


Figure 11-27 Carry-Lookahead Design

The RCO output of the first stage of the ripple carry is connected to the ENP input of the second stage and all subsequent stages. The RCO output of second stage and all subsequent stages is connected to the ENT input of the next stage. The ENT of the second stage is always enabled / tied to VCC. CE is always connected to the ENT input of the first stage. This cascading method allows the first stage of the ripple carry to be built as a prescaler. In other words, the first stage is built to count very fast.

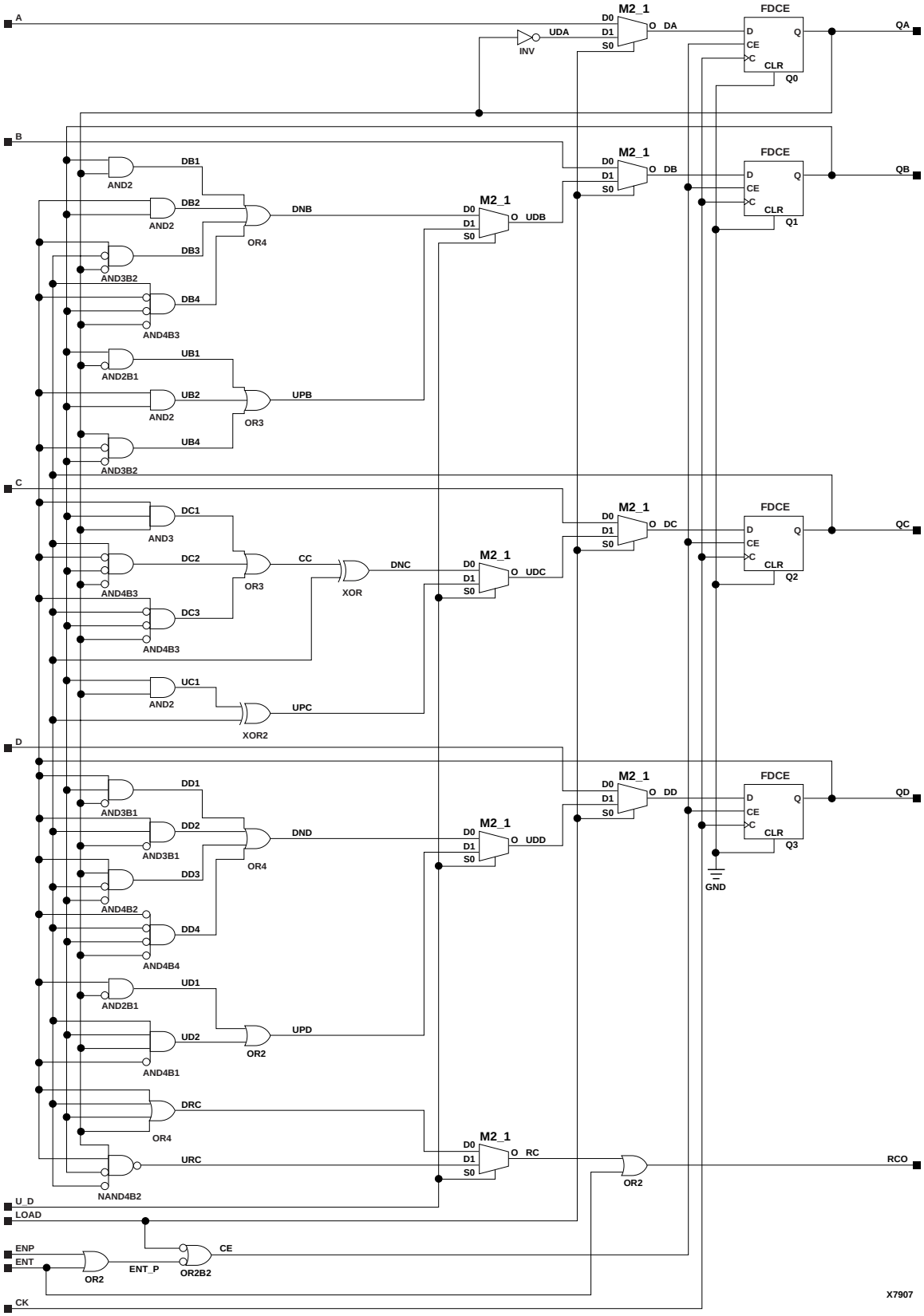


Figure 11-28 X74_168 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

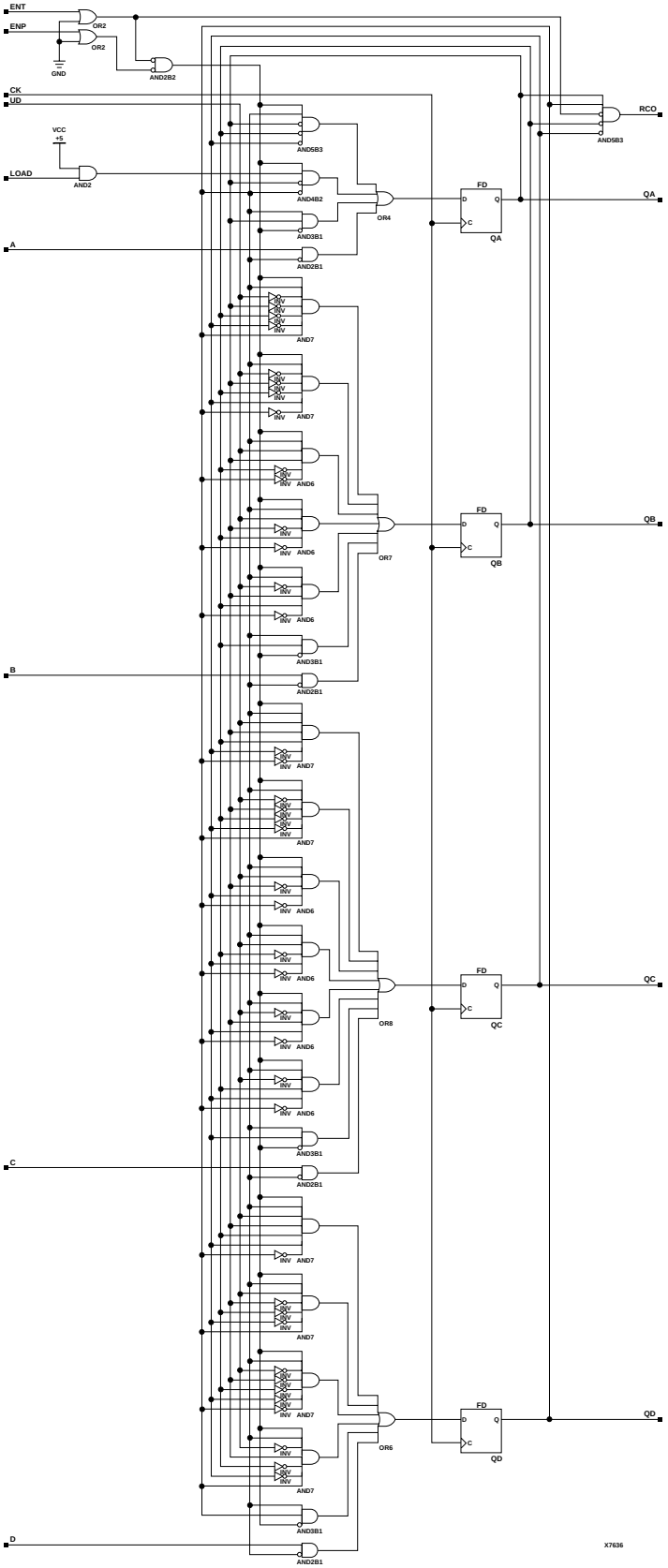
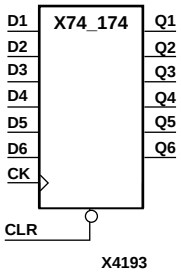


Figure 11-29 X74_168 Implementation XC9000

X74_174

6-Bit Data Register with Active-Low Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



The active-Low asynchronous clear input (CLR), when Low, overrides the clock and resets the six data outputs (Q6 – Q1) Low. When CLR is High, the data on the six data inputs (D6 – D1) is transferred to the corresponding data outputs on the Low-to-High clock (CK) transition.

Inputs			Outputs
CLR	D6 – D1	CK	Q6 – Q1
0	X	X	0
1	D6 – D1	↑	d6 – d1

dn = state of referenced input one setup time prior to active clock transition

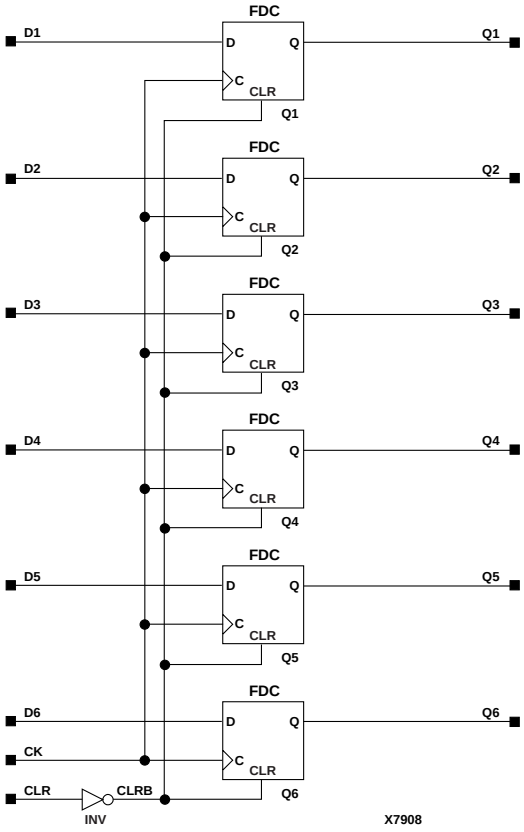
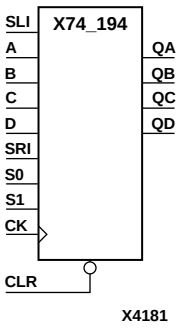


Figure 11-30 X74_174 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_194

4-Bit Loadable Bidirectional Serial/Parallel-In Parallel-Out Shift Register

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X4181

X74_194 is a 4-bit shift register with shift-right serial input (SRI), shift-left serial input (SLI), parallel inputs (D – A), parallel outputs (QD – QA), two control inputs (S1, S0), and active-Low asynchronous clear (CLR). The shift register performs the following functions.

Clear	When CLR is Low, all other inputs are ignore and outputs QD – QA go to logic state zero.
Load	When S1 and S0 are High, the data on inputs D –A is loaded into the corresponding output bits QD –QA during the Low-to-High clock (CK) transition.
Shift Right	When S1 is Low and S0 is High, the data is to the next-highest bit position (right) as new data is loaded into QA(SRI→QA,QA→QB, QB→QC, and so forth).
Shift Left	When S1 is High and S0 is Low, the data is shifted to the next-lowest bit position (left) as new data is loaded into QD (SLI→QD,QD→QC,QC→QB, and so forth).

Registers can be cascaded by connecting the QD output of one stage to the SRI input of the next stage, the QA output of one stage to the SLI input of the next stage, and connecting clock, S1, S0, and CLR inputs in parallel.

Inputs							Outputs			
CLR	S1	S0	SRI	SLI	A – D	CK	QA	QB	QC	QD
0	X	X	X	X	X	X	0	0	0	0
1	0	0	X	X	X	X	No Chg	No Chg	No Chg	No Chg
1	1	1	X	X	A – D	↑	a	b	c	d
1	0	1	SRI	X	X	↑	sri	qa	qb	qc
1	1	0	X	SLI	X	↑	qb	qc	qd	sli

Lowercase letters represent state of referenced input or output one setup time prior to active clock transition

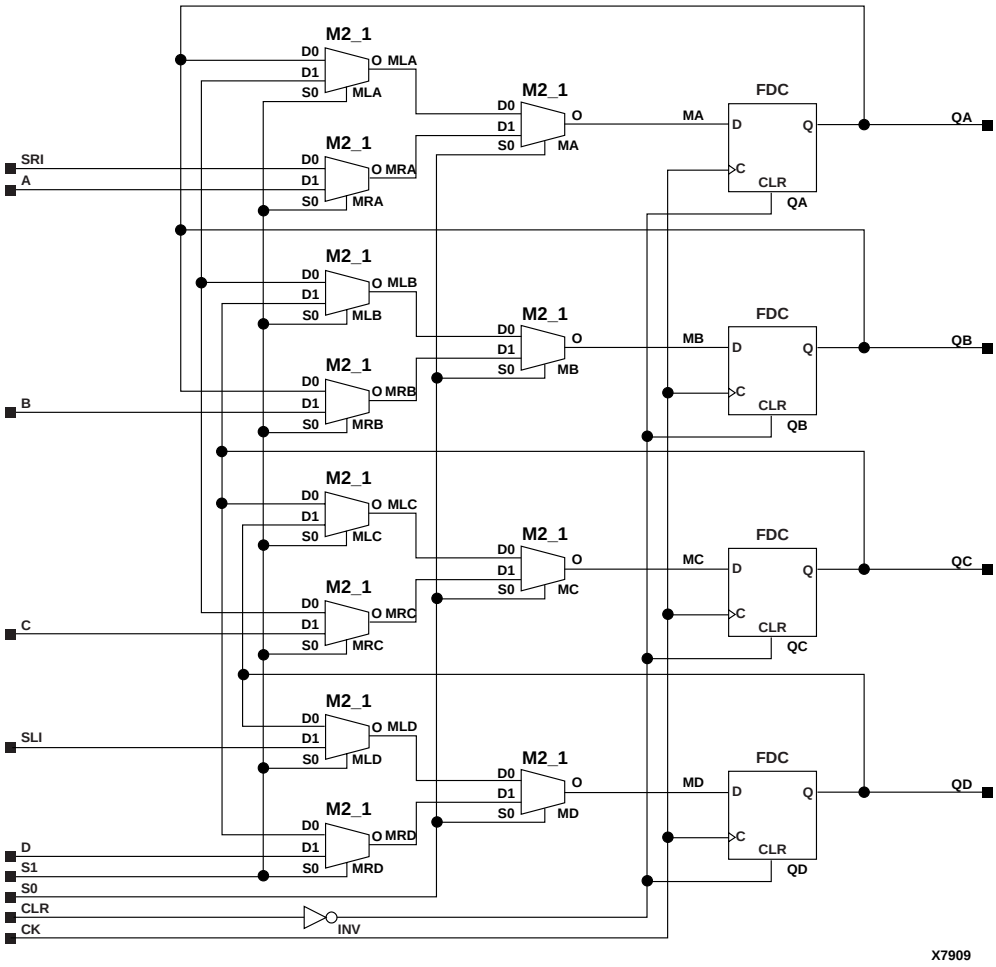
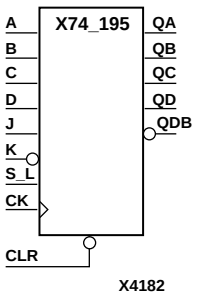


Figure 11-31 X74_194 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_195

4-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_195 is a 4-bit shift register with shift-right serial inputs (J, active High, and K, active Low), parallel inputs (D – A), parallel outputs (QD – QA) and QDB, shift / load control input (S_L), and active-Low asynchronous clear (CLR). Asynchronous CLR, when Low, overrides all other inputs and resets data outputs QD – QA Low and QDB High. When S_L is Low and CLR is High, data on the D – A inputs is loaded into the corresponding QD – QA bits of the register during the Low-to-High clock (CK) transition. When S_L and CLR are High, the first bit of the register (QA) responds to the J and K inputs during the Low-to-High clock transition, as shown in the truth table. During subsequent Low-to-High clock transitions, with S_L and CLR High, the data is shifted to the next-highest bit position (shift right) as new data is loaded into QA (J, K→QA, QA→QB, QB→QC, and so forth).

Registers can be cascaded by connecting the QD and QDB outputs of one stage to the J and K inputs, respectively, of the next stage and connecting clock, S_L and CLR inputs in parallel.

Inputs						Outputs				
CLR	S_L	J	K	A – D	CK	QA	QB	QC	QD	QDB
0	X	X	X	X	X	0	0	0	0	1
1	0	X	X	A – D	↑	a	b	c	d	$\overline{d}$
1	1	0	0	X	↑	0	qa	qb	qc	$\overline{qc}$
1	1	1	1	X	↑	1	qa	qb	qc	$\overline{qc}$
1	1	0	1	X	↑	qa	qa	qb	qc	$\overline{qc}$
1	1	1	0	X	↑	$\overline{qa}$	qa	qb	qc	$\overline{qc}$

Lowercase letters represent state of referenced input or output one setup time prior to active clock transition

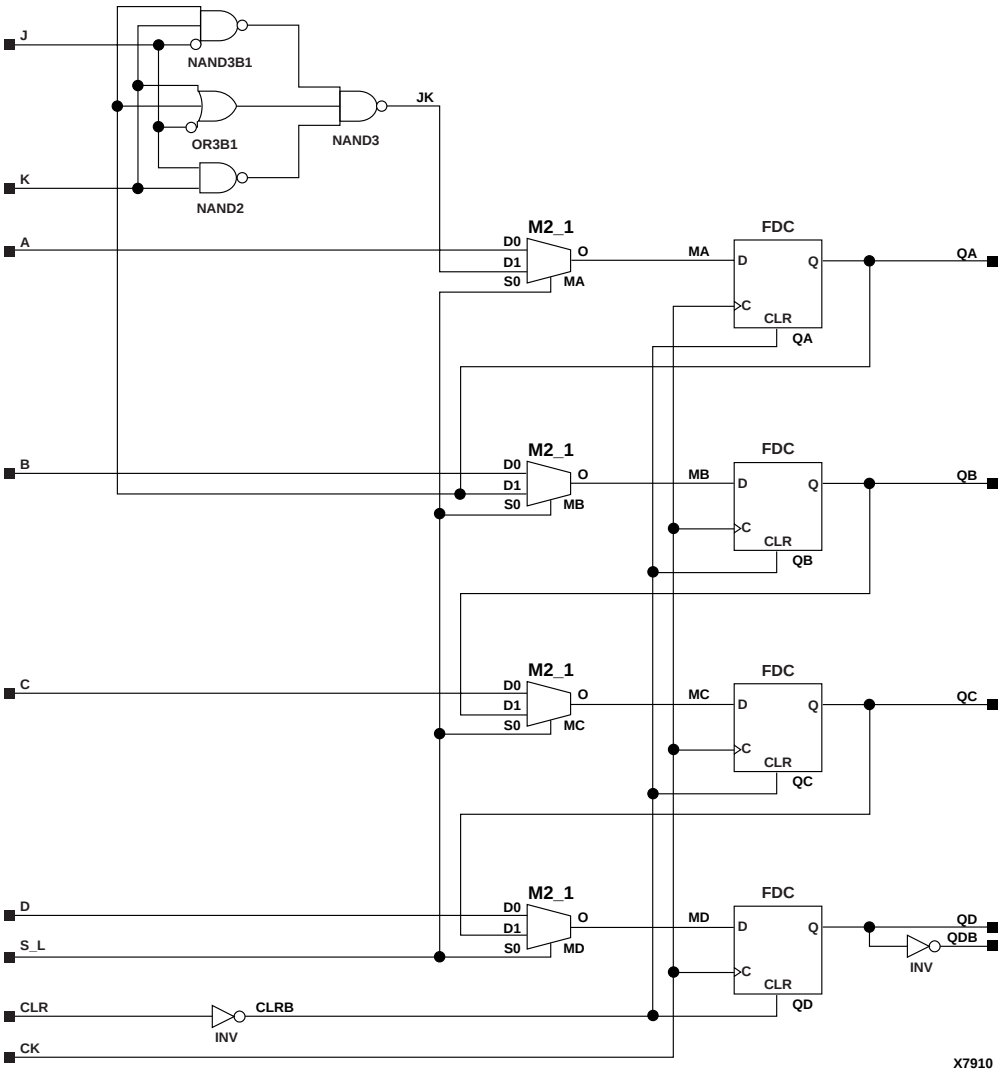
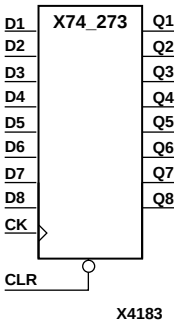


Figure 11-32 X74_195 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_273

8-Bit Data Register with Active-Low Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_273 is an 8-bit data register with active-Low asynchronous clear. The active-Low asynchronous clear (CLR), when Low, overrides all other inputs and resets the data outputs (Q8 – Q1) Low. When CLR is High, the data on the data inputs (D8 – D1) is transferred to the corresponding data outputs (Q8 – Q1) during the Low-to-High clock transition (CK).

Inputs			Outputs
CLR	D8 – D1	CK	Q8 – Q1
0	X	X	0
1	D8 – D1	↑	d8 – d1

dn = state of referenced input one setup time prior to active clock transition

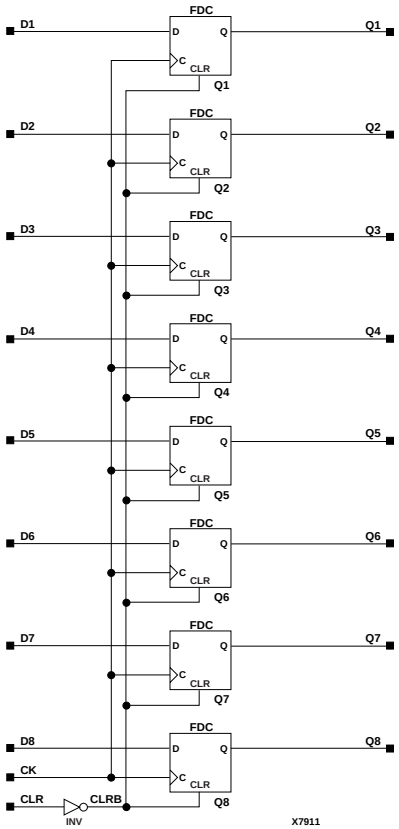
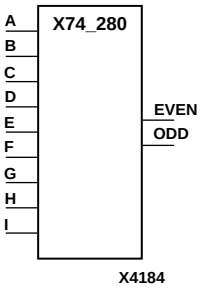


Figure 11-33 X74_273 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_280

9-Bit Odd/Even Parity Generator/Checker

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N/A	N/A



X74_280 parity generator/checker compares up to nine data inputs (I – A) and provides both even (EVEN) and odd parity (ODD) outputs. The EVEN output is High when an even number of inputs is High. The ODD output is High when an odd number of inputs is High.

Expansion to larger word sizes is accomplished by tying the ODD outputs of up to nine parallel components to the data inputs of one more X74_280; all other inputs are tied to ground.

Inputs	Outputs	
Number of Ones on A – I	EVEN	ODD
0, 2, 4, 6, or 8	1	0
1, 3, 5, 7, or 9	0	1

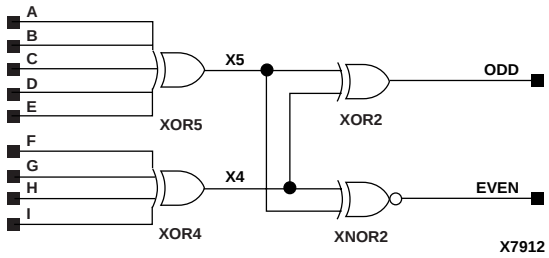


Figure 11-34 X74_280 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

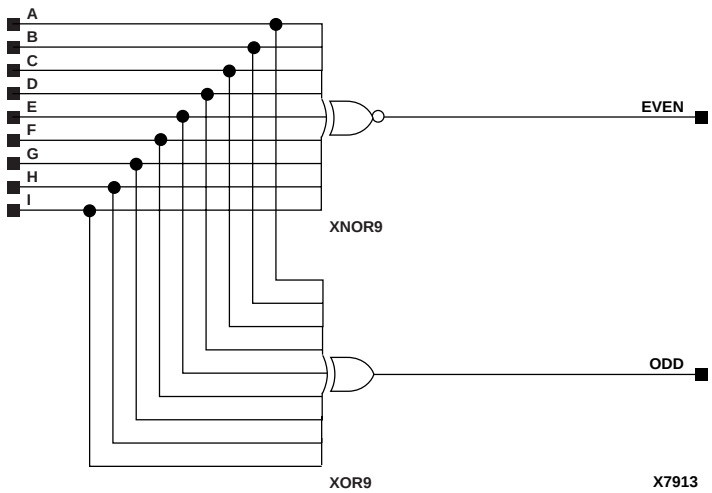
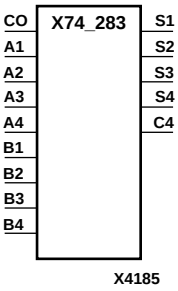


Figure 11-35 X74_280 Implementation XC9000

X74_283

4-Bit Full Adder with Carry-In and Carry-Out

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_283, a 4-bit full adder with carry-in and carry-out, adds two 4-bit words (A4 – A1 and B4 – B1) and a carry-in (C0) and produces a binary sum output (S4 – S1) and a carry-out (C4).

16(C4)+8(S4)+4(S3)+2(S2)+S1=8(A4+B4)+4(A3+B3)+2(A2+B2)+(A1+B1)+CO (where "+" = addition)

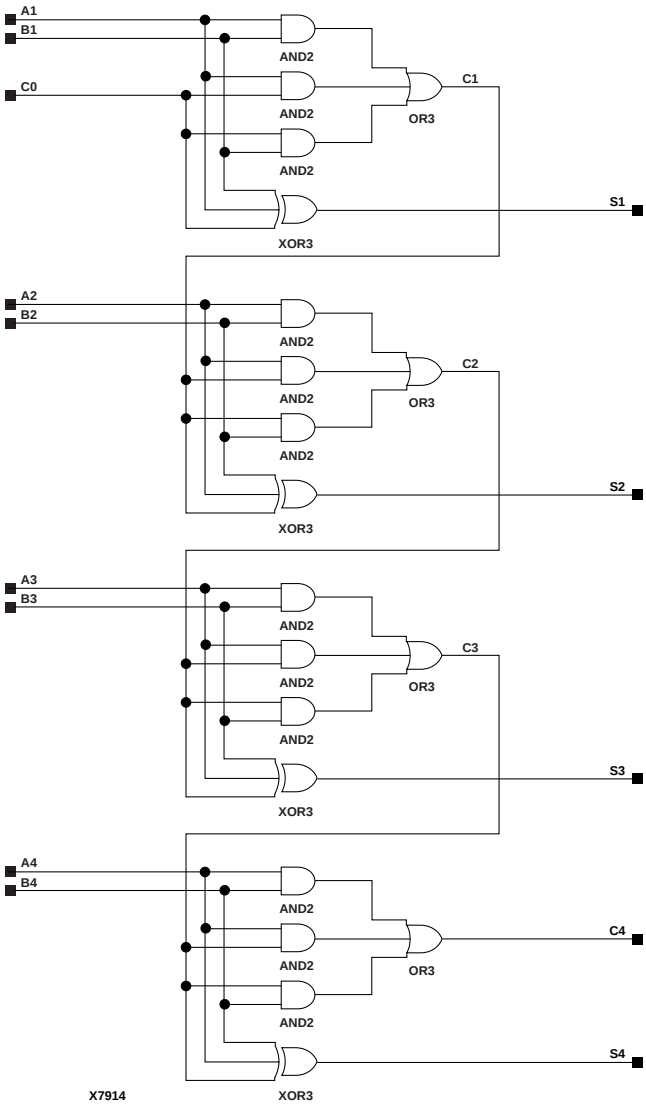


Figure 11-36 X74_283 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

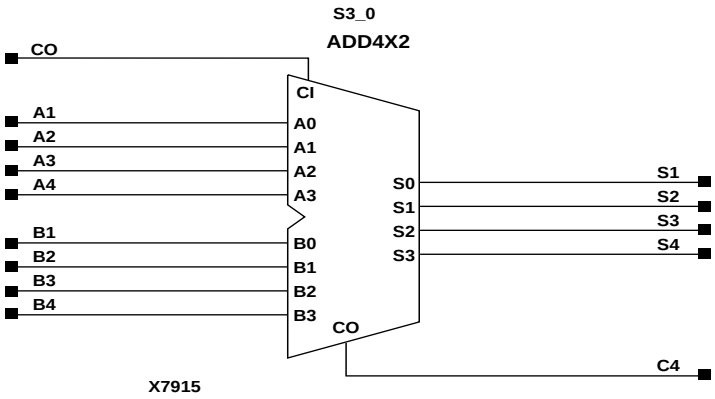
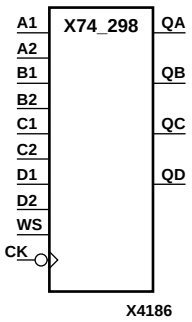


Figure 11-37 X74_283 Implementation XC9000

X74_298

Quadruple 2-Input Multiplexer with Storage and Negative-Edge Clock

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_298 selects 4-bits of data from two sources (D1 – A1 or D2 – A2) under the control of a common word select input (WS). When WS is Low, D1 – A1 is chosen, and when WS is High, D2 – A2 is chosen. The selected data is transferred into the output register (QD – QA) during the High-to-Low transition of the negative-edge triggered clock (CK).

Inputs				Outputs
WS	**A1 – D1**	**A2 – D2**	**CK**	**QA – QD**
0	A1 – D1	X	↓	a1 – d1
1	X	A2 – D2	↓	a2 – d2

an – dn = state of referenced input one setup time prior to active clock transition

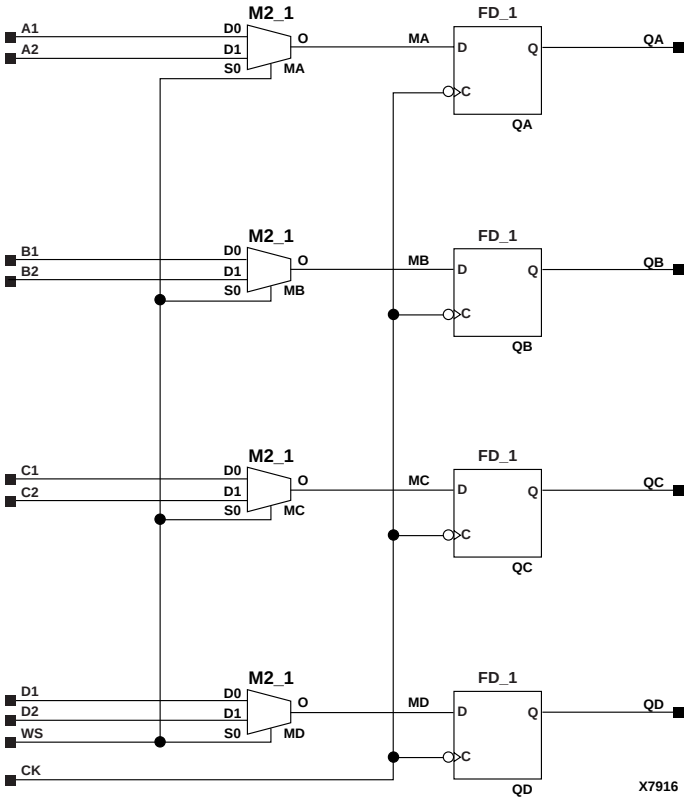


Figure 11-38 X74_298 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

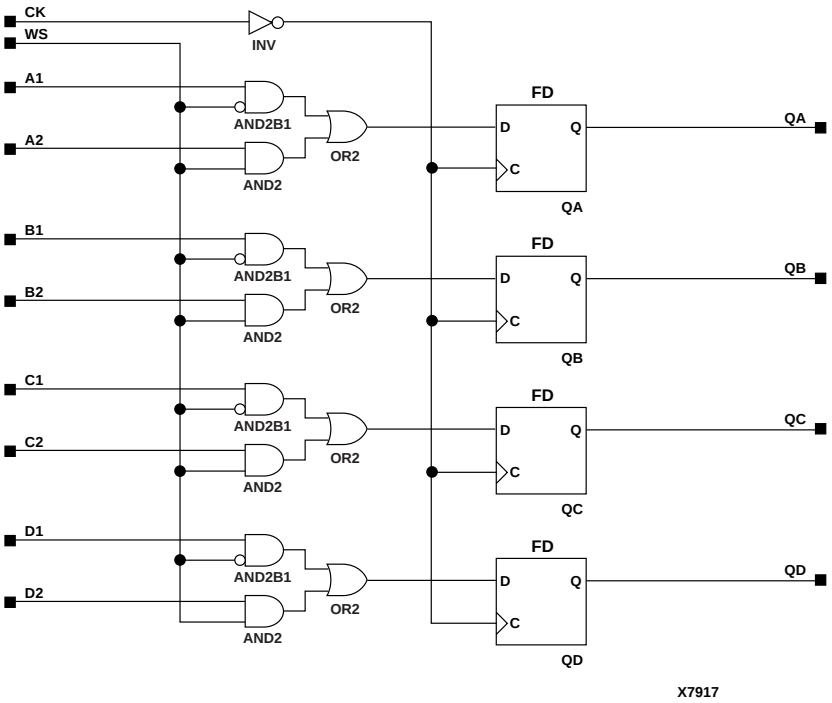
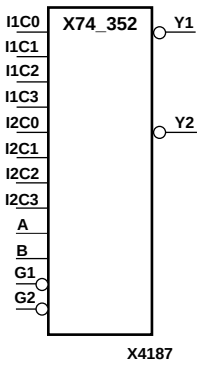


Figure 11-39 X74_298 Implementation XC9000

X74_352

Dual 4-to-1 Multiplexer with Active-Low Enables and Outputs

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_352 comprises two 4-to-1 multiplexers with separate enables (G1 and G2) but with common select inputs (A and B). When an active-Low enable (G1 or G2) is Low, the multiplexer chooses one data bit from the four sources associated with the particular enable (I1C3 – I1C0 for G1 and I2C3 – I2C0 for G2) under the control of the common select inputs (A and B). The active-Low outputs (Y1 and Y2) reflect the inverse of the selected data as shown in truth table. Y1 is associated with G1 and Y2 is associated with G2. When an active-Low enable is High, the associated output is High.

Inputs							Outputs
G	B	A	IC0	IC1	IC2	IC3	Y
1	X	X	X	X	X	X	1
0	0	0	IC0	X	X	X	$\overline{IC0}$
0	0	1	X	IC1	X	X	$\overline{IC1}$
0	1	0	X	X	IC2	X	$\overline{IC2}$
0	1	1	X	X	X	IC3	$\overline{IC3}$

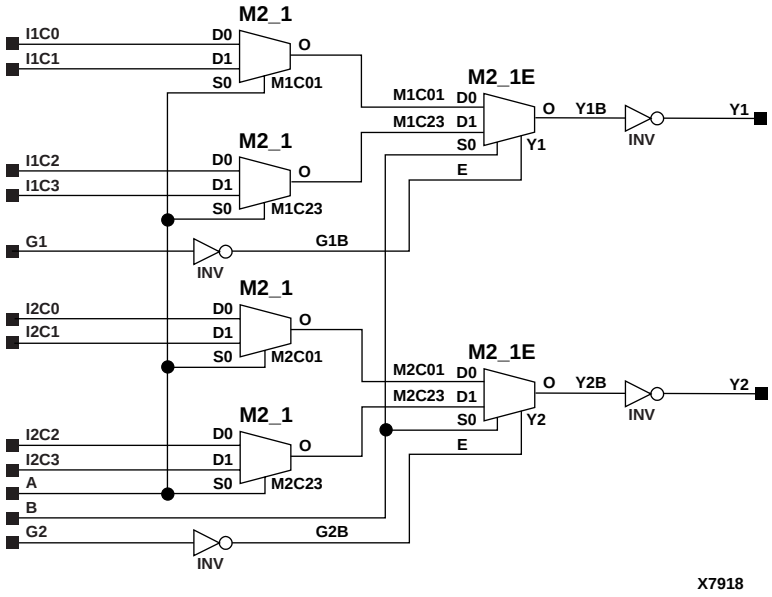
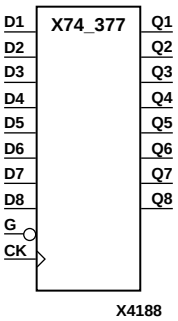


Figure 11-40 X74_352 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_377

8-Bit Data Register with Active-Low Clock Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



When the active-Low clock enable (G) is Low, the data on the eight data inputs (D8 – D1) is transferred to the corresponding data outputs (Q8 – Q1) during the Low-to-High clock (CK) transition. The register ignores clock transitions when G is High.

Inputs			Outputs
G	D8 – D1	CK	Q8 – Q1
1	X	X	No Chg
0	D8 – D1	↑	d8 – d1

dn = state of referenced input one setup time prior to active clock transition

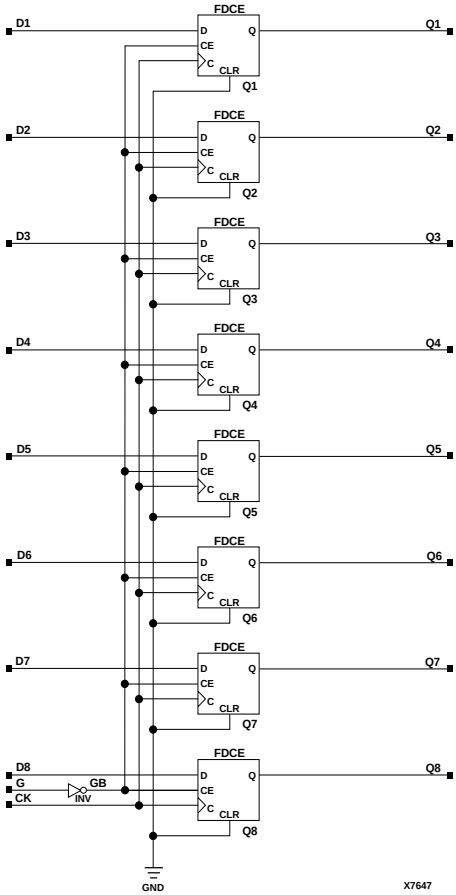
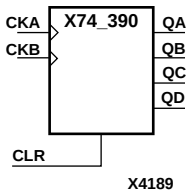


Figure 11-41 X74_377 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_390

4-Bit BCD/Bi-Quinary Ripple Counter with Negative-Edge Clocks and Asynchronous Clear

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_390 is a cascadable, resettable binary-coded decimal (BCD) or bi-quinary counter that can be used to implement cycle lengths equal to whole and / or cumulative multiples of 2 and / or 5. In BCD mode, the output QA is connected to negative-edge clock input (CKB), and data outputs (QD – QA) increment during the High-to-Low clock transition as shown in the truth table, provided asynchronous clear (CLR) is Low. In bi-quinary mode, output QD is connected to the negative-edge clock input (CKA). As shown in the truth table, in bi-quinary mode, QA supplies a divide-by-five output and QB supplies a divide-by-two output, provided asynchronous CLR is Low. When asynchronous CLR is High, the other inputs are overridden, and data outputs (QD – QA) are reset Low.

Larger ripple counters are created by connecting the QD output (BCD mode) or QA output (bi-quinary mode) of the first stage to the appropriate clock input of the next stage and connecting the CLR inputs in parallel.

Count	BCD				Bi-Quinary			
	QD	QC	QB	QA	QD	QC	QB	QA
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	1	0	0
3	0	0	1	1	0	1	1	0
4	0	1	0	0	1	0	0	0
5	0	1	0	1	0	0	0	1
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	1	0	1
8	1	0	0	0	0	1	1	1
9	1	0	0	1	1	0	0	1

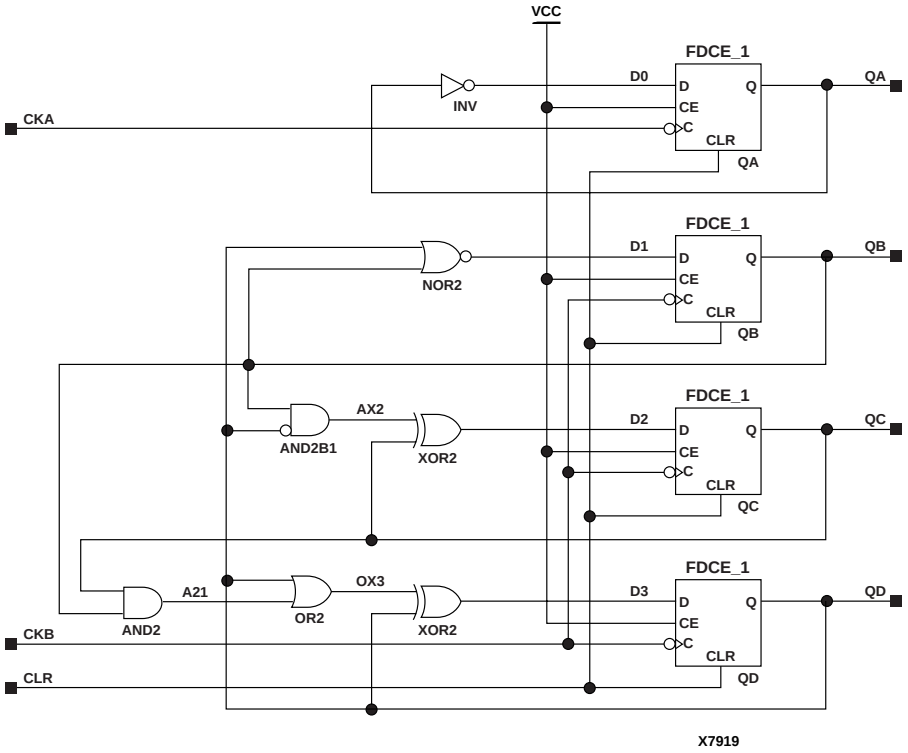


Figure 11-42 X74_390 Implementation XC3000, XC4000E, XC4000X, XC5200, Spartan, SpartanXL

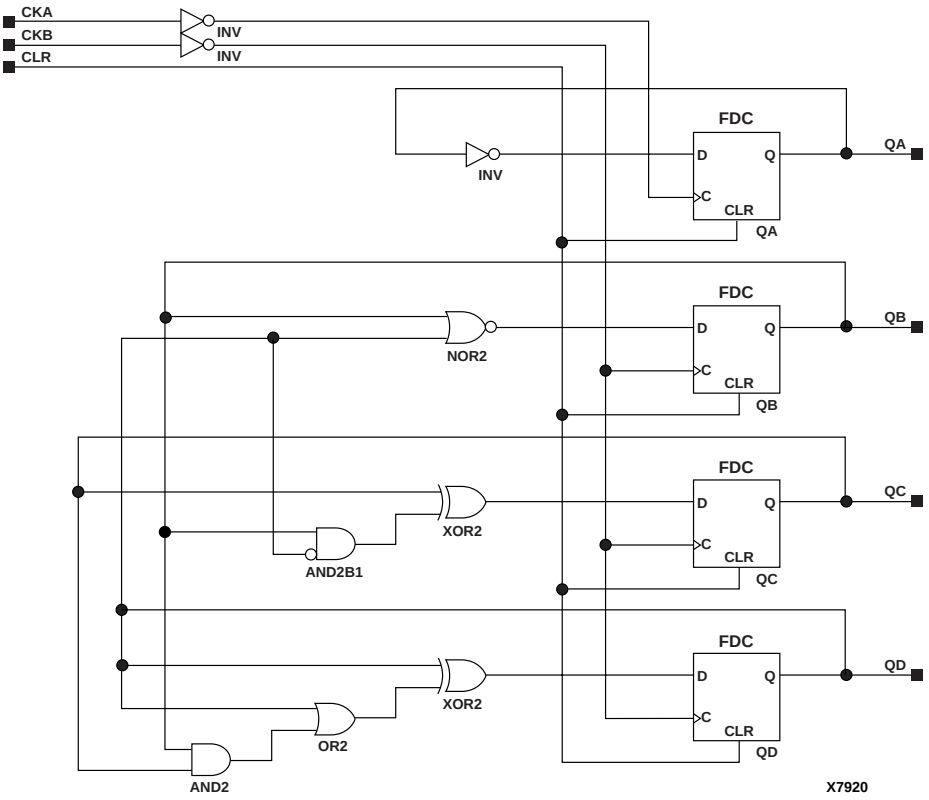
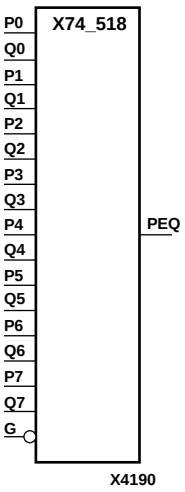


Figure 11-43 X74_390 Implementation XC9000

X74_518

8-Bit Identity Comparator with Active-Low Enable

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_518 is an 8-bit identity comparator with 16 data inputs for two 8-bit words (P7 – P0 and Q7 – Q0), data output (PEQ), and active-Low enable (G). When G is High, the PEQ output is Low. When G is Low and the two input words are equal, PEQ is High. Equality is determined by a bit comparison of the two words. When any of the two equivalent bits from the two words are not equal, PEQ is Low.

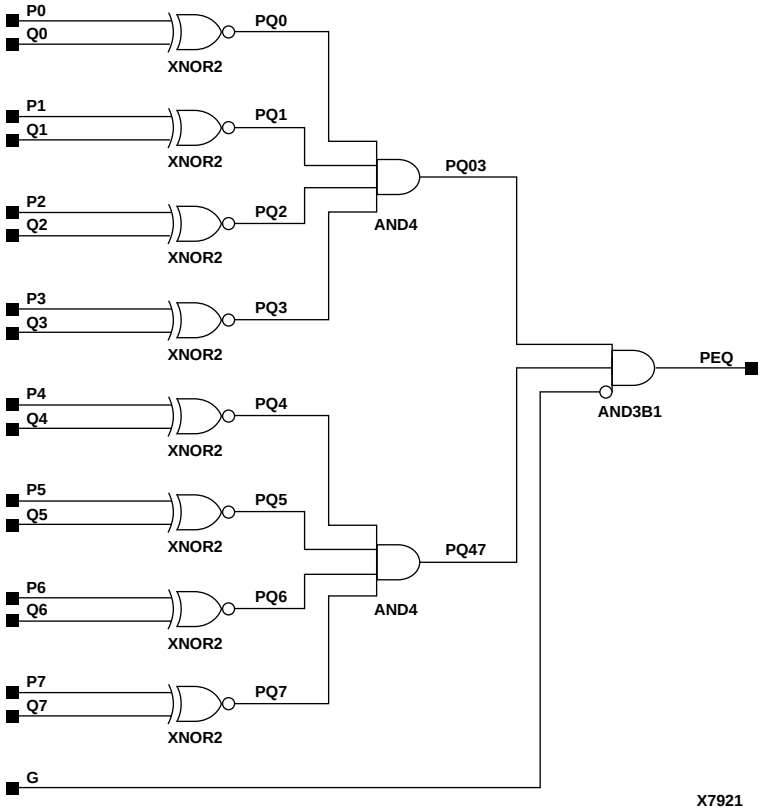
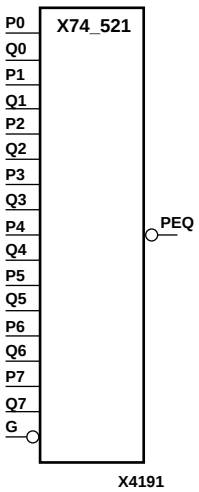


Figure 11-44 X74_518 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

X74_521

8-Bit Identity Comparator with Active-Low Enable and Output

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
Macro	Macro	Macro	Macro	Macro	Macro	Macro	N / A	N / A



X74_521 is an 8-bit identity comparator with 16 data inputs for two 8-bit words (P7 – P0 and Q7 – Q0), active-Low data output (PEQ), and active-Low enable (G). When G is High, the PEQ output is High. When G is Low and the two input words are equal, PEQ is Low. X74_521 does a bit comparison of the two words to determine equality. When any of the two equivalent bits from the two words are not equal, PEQ is High.

Inputs									Outputs
G	P7, Q7	P6, Q6	P5, Q5	P4, Q4	P3, Q3	P2, Q2	P1, Q1	P0, Q0	PEQ
1	X	X	X	X	X	X	X	X	1
0	P7≠Q7	X	X	X	X	X	X	X	1
0	X	P6≠Q6	X	X	X	X	X	X	1
0	X	X	P5≠Q5	X	X	X	X	X	1
0	X	X	X	P4≠Q4	X	X	X	X	1
0	X	X	X	X	P3≠Q3	X	X	X	1
0	X	X	X	X	X	P2≠Q2	X	X	1
0	X	X	X	X	X	X	P1≠Q1	X	1
0	X	X	X	X	X	X	X	P0≠Q0	1
0	P7=Q7	P6=Q6	P5=Q5	P4=Q4	P3=Q3	P2=Q2	P1=Q1	P0=Q0	0

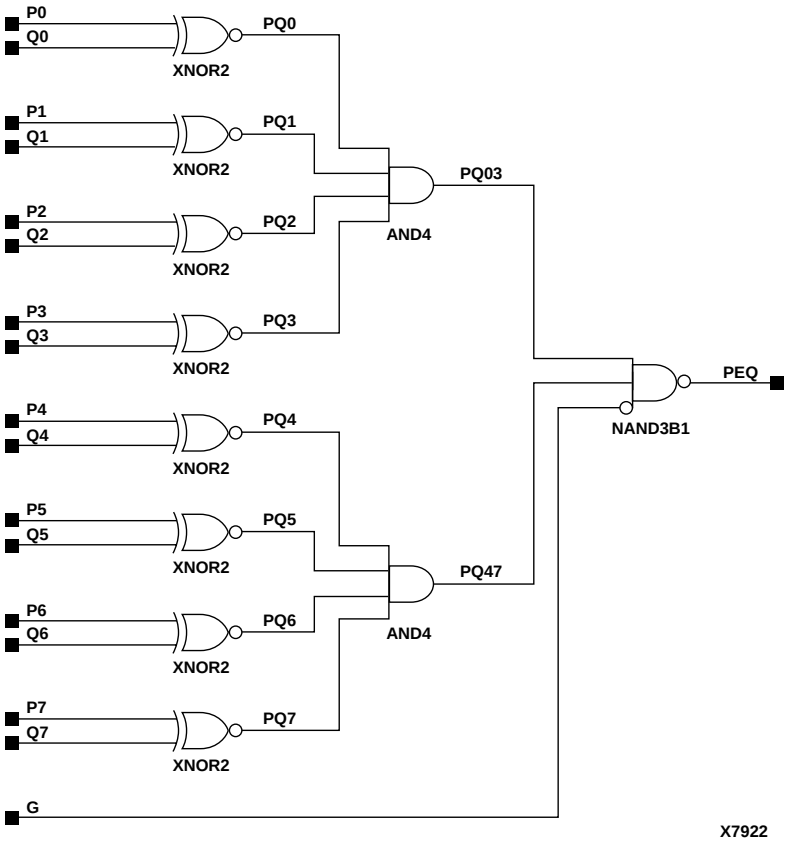


Figure 11-45 X74_521 Implementation XC3000, XC4000E, XC4000X, XC5200, XC9000, Spartan, SpartanXL

Chapter 12

Attributes, Constraints, and Carry Logic

This chapter lists and describes all the attributes that you can use with your design entry software and the constraints that are contained in machine- and user-generated files.

This chapter contains the following major sections.

- "Overview"
- "Information for Mentor Customers"
- "Schematic Syntax"
- "UCF / NCF File Syntax"
- "Constraints Editor"
- "Attributes / Logical Constraints" — This section contains alphabetical listings of the attributes and constraints as well as descriptions, syntax, and examples of each constraint.
- "Placement Constraints"
- "Relative Location (RLOC) Constraints"
- "Timing Constraints"
- "Physical Constraints"
- "Relationally Placed Macros (RPMs)"
- "Carry Logic in XC4000, Spartan, SpartanXL"
- "Carry Logic in XC5200"
- "Carry Logic in Virtex, Spartan2"

Overview

This section gives an overview of attributes, constraints, and carry logic.

Attributes

Attributes are instructions placed on symbols or nets in an FPGA or CPLD schematic to indicate their placement, implementation, naming, directionality, and so forth. This information is used by the design implementation software during placement and routing of a design. All the attributes listed in this chapter are available in the schematic entry tools directly supported by Xilinx unless otherwise noted, but some may not be available in textual entry methods such as VHDL.

Attributes applicable only to a certain schematic entry tool are described in the documentation for that tool. For third-party interfaces, consult the interface user guides for information on which attributes are available and how they are used.

Refer to the "Schematic Syntax" section in this chapter for guidelines on placing attributes on symbols on a schematic.

Constraints

Constraints, which are a type, or subset, of attributes, indicate where an element should be placed.

Logical Constraints

Constraints that are attached to elements in the design prior to mapping are referred to as logical constraints. Applying logical constraints helps you to adapt your design's performance to expected worst-case conditions. Later, when you choose a specific Xilinx architecture and place and route your design, the logical constraints are converted into physical constraints.

You can attach logical constraints using attributes in the input design, which are written into the Netlist Constraints File (NCF), or with a User Constraints File (UCF). Refer to the "UCF / NCF File Syntax" section for the rules for entering constraints in a UCF or NCF file.

Three categories of logical constraints are described in detail in the "Attributes / Logical Constraints" section: placement constraints, relative location constraints, and timing constraints.

The "Placement Constraints" section gives examples showing how to place constraints on the various types of logic elements in FPGA designs.

For FPGAs, relative location constraints (RLOCs) group logic elements into discrete sets and allow you to define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design. Refer to the "Relative Location (RLOC) Constraints" section for detailed information on RLOCs.

Timing constraints allow you to specify the maximum allowable delay or skew on any given set of paths or nets in your design. Refer to the "Timing Constraints" section for detailed information on using timing constraints in a UCF file.

Physical Constraints

Constraints can also be attached to the elements in the physical design, that is, the design after mapping has been performed. These constraints are referred to as physical constraints and are defined in the Physical Constraints File (PCF), which is created during mapping. See the "Physical Constraints" section.

Note: It is preferable to place any user-generated constraint in the UCF file — *not* in an NCF or PCF file.

Carry Logic

Dedicated fast carry logic increases the efficiency and performance of adders, subtracters, accumulators, comparators, and counters. See the "Carry Logic in XC4000, Spartan, SpartanXL" section, "Carry Logic in XC5200" section, and "Carry Logic in Virtex, Spartan2" section at the end of this chapter.

Information for Mentor Customers

In some software programs, particularly Mentor Graphics, attributes are called properties, but their functionality is the same as that of attributes. In this document, they are referred to as attributes.

There are two types of Mentor Graphics properties. In one, a property is equal to a value, for example, LOC=AA. In the other, the property name and the value are the same, for example, DECODE. In the first type, "attribute" refers to the property. In the second, "attribute" refers to the property and the value.

The Mentor netlist writer program (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. Because property names are processed in this way, you must enter the variable text of certain constraints in uppercase letters only. The most salient examples are the following.

- A TS*identifier* name should contain only uppercase letters on a Mentor Schematic (TSMAIN, for example, but not TSmain or TSMain). Also, if a TS*identifier* name is referenced in a property value, it must be entered in uppercase letters. For example, the TSID1 in the second constraint below must be entered in uppercase letters to match the TSID1 name in the first constraint.

  ```
  TSID1 = FROM: gr1: TO: gr2: 50;
  TSMAIN = FROM: here: TO: there: TSID1: /2;
  ```

- Group names should contain only uppercase letters on a Mentor schematic (MY_FLOPS, for example, but not my_flops or My_flops).
- If a group name appears in a property value, it must also be expressed in uppercase letters. For example, the GROUP3 in the first constraint below must be entered in uppercase letters to match the GROUP3 in the second constraint.

  ```
  TIMEGRP GROUP1 = gr2: GROUP3;
  TIMEGRP GROUP3 = FFS: except: grp5;
  ```

Schematic Syntax

This section describes how to place attributes on symbols on a schematic. The following guidelines are for specifying locations.

- To specify a single location, use the following syntax.

  attribute=location

- To specify multiple locations, use the following syntax.

  attribute=location,location,location

  A comma separates locations in a list of locations. (Spaces are ignored if entered.)

  When you specify a list of locations, PAR (Place and Route) chooses any one of the listed locations.

- To define a range by specifying the two corners of a bounding box, use the following syntax.

  attribute=location:*location* [SOFT]

  A colon is only used to separate the corners of a bounding box. The logic represented by the symbol is placed somewhere inside the bounding box. If SOFT is specified, PAR may place the attribute elsewhere to obtain better results.

Following are some examples of location attributes.

LOC=CLB_R1C2	Locates the element in the CLB in the first row, second column
LOC=CLB_R1C2.LC3	Locates the element in the top-most slice of the four slices of the XC5200 CLB located in the first row, second column (XC5200)
LOC=CLB_R1C2.S1	Locates the element in the right-most slice of the two slices of the Virtex or Spartan2 CLB located in the first row, second column
LOC=P12	Assigns the signal to the pin P12
RLOC=R4C4	Assigns the location relative to other elements in the set to row 4, column 4
RLOC=R0Cl.FFX	Assigns the location relative to other elements in the set to the X flip-flop in row 0, column 1
RLOC=R2C3.LC0	Assigns the location of the element to be one slice below another element in the set that has an RLOC=R2C3.LC1 attribute (XC5200)
RLOC=R2C3.S0	Assigns the location of the element to be the right-most slice of another element in the set that has an RLOC=R2C3.S1 attribute (Spartan2, Virtex)
RLOC_ORIGIN=R4C4	Fixes the reference CLB of a designated set at row 4, column 4
RLOC_RANGE=R4C4 : R10C10	Limits the members of a designated set to a range between row 4, column 4 and row 10, column 10

A name can be composed of any ASCII character except for control characters and characters that have special meanings.

Control characters are : (colon) ; (semi-colon) , (comma) and “(double quotes).

Characters with special meaning are / (forward slash) used as the hierarchy separator and . (period) used as the pin separator and for extensions.

For additional propagation rules for each constraint and attribute, refer to the “Macro and Net Propagation Rules” table in the "Attributes / Logical Constraints" section.

UCF/NCF File Syntax

Logical constraints are found in a Netlist Constraint File (NCF), an ASCII file generated by synthesis programs, and in a User Constraint File (UCF), an ASCII file generated by the user. This section describes the rules for entering constraints in a UCF or NCF file.

Note: It is preferable to place any user-generated constraint in the UCF file — *not* in an NCF or PCF file.

Following are some general rules for the UCF and NCF files.

- The UCF and NCF files are case sensitive. Identifier names (names of objects in the design, such as net names) must exactly match the case of the name as it exists in the source design netlist. However, any Xilinx constraint keyword (for

example, LOC, PERIOD, HIGH, LOW) may be entered in either all upper-case or all lower-case letters; mixed case is not allowed.

- Each statement is terminated by a semicolon (;).
- No continuation characters are necessary if a statement exceeds one line, since a semicolon marks the end of the statement.
- You can add comments to the UCF / NCF file by beginning each comment line with a pound (#) sign. Following is an example of part of a UCF / NCF file containing comments.

```
# file TEST.UCF
# net constraints for TEST design

NET $SIG_0 MAXDELAY 10 ;
NET $SIG_1 MAXDELAY 12 ns ;
```

- Statements do not have to be placed in any particular order in the UCF / NCF file.

The constraints in the UCF / NCF files and the constraints in the schematic or synthesis file are applied equally; it does not matter whether a constraint is entered in the schematic or synthesis file or in the UCF / NCF files. If the constraints overlap, however, UCF / NCF constraints override the schematic constraint. Refer to the "Using Timing Constraints" chapter of the *Development System Reference Guide* for additional details on constraint priorities.

If by mistake two or more elements are locked onto a single location, the mapper detects the conflict, issues a detailed error message, and stops processing so that you can correct the mistake.

The syntax for constraints in the UCF / NCF files is as follows.

{**NET** | **INST** | **PIN**} *full_name constraint* **;**

or

SET *set_name set_constraint* **;**

where

full_name is a full hierarchically qualified name of the object being referred to. When the name refers to a pin, the *instance* name of the element is also required.

constraint is a constraint in the same form as it would be used if it were attached as an attribute on a schematic object. For example, LOC=P38 or FAST, and so forth.

set_name is the name of an RLOC set. Refer to the “RLOC Sets” section for more information.

set_constraint is an RLOC_ORIGIN or RLOC_RANGE constraint.

Note: To specify attributes for the CONFIG or TIMEGRP primitives (tables), the keywords CONFIG or TIMEGRP precede the attribute definitions in the constraints files.

```
CONFIG PROHIBIT=CLB_R6C8 ;

TIMEGRP input_pads=pads EXCEPT output_pads ;
```

For the TIMESPEC primitive (table), the use of the keyword TIMESPEC in the constraints files is optional.

Note: In all of the constraints files (NCF, UCF, and PCF), instance or variable names that match internal reserved words will be rejected unless the names are enclosed in double quotes. It is good practice to enclose all names in double quotes.

For example, the following entry would *not* be accepted because the word net is a reserved word.

```
NET net OFFSET=IN 20 BEFORE CLOCK;
```

Following is the recommended way to enter the constraint.

```
NET “net” OFFSET=IN 20 BEFORE CLOCK;
```

or

```
NET “$SIG_0” OFFSET=IN 20 BEFORE CLOCK;
```

Inverted signal names, for example ~OUTSIG1, must always be enclosed in double quotes as shown in the following example.

```
NET “~OUTSIG1” OFFSET=IN 20 BEFORE CLOCK;
```

Wildcards

You can use the wildcard characters, * and ?, in constraint statements as follows. The asterisk (*) represents any string of zero or more characters. The question mark (?) indicates a single character.

In net names, the wildcard characters enable you to select a group of symbols whose output net names match a specific string or pattern. For example, the following constraint increases the output speed of the pads to which nets with names that begin with any series of characters followed by "AT" and end with one single characters are connected.

```
NET *AT? FAST ;
```

In an instance name, a wildcard character by itself represents every symbol of the appropriate type. For example, the following constraint initializes an entire set of ROMs to a particular hexadecimal value, 5555.

```
INST $1I3*/ROM2 INIT=5555 ;
```

If the wildcard character is used as part of a longer instance name, the wildcard represents one or more characters at that position.

In a location, you can use a wildcard character for either the row number or the column number. For example, the following constraint specifies placement of any instance under the hierarchy of loads_of_logic in any CLB in column 8.

```
INST /loads_of_logic/* LOC=CLB_r*c8 ;
```

Wildcard characters can be used in dot extensions.

```
CLB_R1C3.*
```

Wildcard characters cannot be used for both the row number and the column number in a single constraint, since such a constraint is meaningless.

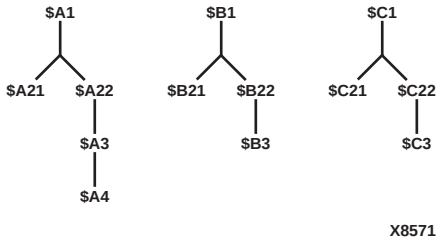
Traversing Hierarchies

Note: Top-level block names (design names) are ignored when searching for instance name matches.

You can use the asterisk wildcard character (*) to traverse the hierarchy of a design within a UCF or NCF file. The following syntax applies (where level1 is an example hierarchy level name).

*	Traverses all levels of the hierarchy
level1/*	Traverses all blocks in level1 and below
level1/*/	Traverses all blocks in the level1 hierarchy level but no further

Consider the following design hierarchy.



With the example design hierarchy, the following specifications illustrate the scope of the wildcard.

```
INST *          => <everything>
INST /*         => <everything>
INST /*/        => <$A1,$B1,$C1>
INST $A1/*      => <$A21,$A22,$A3,$A4>
INST $A1/*/     => <$A21,$A22>
INST $A1/*/*    => <$A3,$A4>
INST $A1/*/*/   => <$A3>
INST $A1/*/*/*  => <$A4>
INST $A1/*/*/*/ => <$A4>
INST /*/*22/    => <$A22,$B22,$C22>
INST /*/*22     => <$A22,$A3,$A4,$B22,$B3,$C22,$C3>
```

File Name

By default, NGDBuild reads the constraints file that carries the same name as the input design with a .ucf extension; however, you can specify a different constraints file name with the -uc option when running NGDBuild. NGDBuild automatically reads in the NCF file if it has the same base name as the input XNF or EDIF file and is in the same directory as the XNF or EDIF file.

Note: The implementation tools (NGDBuild, MAP, PAR, etc.) require file name extensions in all lowercase (.ucf, for example) in command lines.

Instances and Blocks

Because the statements in the constraints file concern instances and blocks, these entities are defined here.

An *instance* is a symbol on the schematic. An *instance name* is the symbol name as it appears in the EDIF or XNF netlist. A *block* is a CLB, an IOB, or a TBUF. You can specify the *block name* by using the BLKNM, HBLKNM, or the XBLKNM attribute; by default, the software assigns a block name on the basis of a signal name associated with the block.

Constraints Editor

The Xilinx Constraints Editor is a GUI tool provided in the Xilinx Development System for entering timing constraints and pin location constraints. The Constraints Editor's GUIs simplify constraint entry by guiding you through constraint creation without your needing to understand UCF file syntax.

The Constraints Editor automatically writes out a valid UCF file and a valid NGD file. These files are processed by the Map program, which generates a PCF file. Constraints created using the Xilinx Constraints Editor appear syntactically in the Editable Constraints list the same way they appear in the UCF.

A Constraints Editor "Example" section is included with each attribute/constraint description in the "Attributes and Logcal Constraints" section in this chapter. The Constraints Editor Example gives information on the appropriate Constraints Editor GUI, if one is available, to assist you with entry of that constraint. For a detailed description of how to use the editor, see the *Constraints Editor Guide.*

Attributes/Logical Constraints

Syntax Summary

This section summarizes attribute and logical constraints syntax. This syntax conforms to the conventions given in the "Conventions" section. A checkmark (√) appearing in a column means that the attribute/constraint is supported for architectures that use the indicated library. (Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific device architectures supported in each library.) A blank column means that the attribute/constraint is not supported for architectures using that library.

BASE		BASE = {F | FG | FGM | IO}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

BLKNM		BLKNM = *block_name*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

BUFG		BUFG = {CLK | OE | SR}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

CLKDV_DIVIDE		CLKDV_DIVIDE={ 1.5 | 2 | 2.5 | 3 | 4 | 5 | 8 | 16}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

COLLAPSE		COLLAPSE						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

CONFIG*		CONFIG = *tag value* [*tag value*]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

*The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

DECODE		DECODE						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√						

DIVIDE1_BY and DIVIDE2_BY		DIVIDE1_BY = {4 | 16 | 64 | 256} DIVIDE2_BY = {2 | 8 | 32 | 128 | 1024 | 4096 | 16384 | 65536}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

DOUBLE		DOUBLE						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√			√	√		

DRIVE		**XC4000X, SpartanXL:** DRIVE = {12 |24} **Spartan2, Virtex:** DRIVE = {2 | 4 | 6 | 8 | 12 | 16 | 24}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√*				√	√	√

* supported for XC4000XV and XC4000XLA designs only

DROP_SPEC		TS*identifier*=DROP_SPEC						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

DUTY_CYCLE_CORRECTION		DUTY_CYCLE_CORRECTION={TRUE | FALSE}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

EQUATE_F and EQUATE_G		EQUATE_F = *equation* EQUATE_G = *equation*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√								

FAST		FAST						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

FILE		FILE = *file_name* [.*extension*]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

HBLKNM		HBLKNM = *block_name*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√		

HU_SET		HU_SET = *set_name*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

INIT		INIT ={S | R | *value*}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√		√	√	√	√	√

INIT_0x		INIT_0x = *value*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

INREG		INREG						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

IOB		IOB={TRUE | FALSE}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

IOSTANDARD		IOSTANDARD=*iostandard_name*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

KEEP		KEEP						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√*	√*

*Only at BEL level

KEEPER		KEEPER						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

LOC		**FPGAs:** LOC=l*ocation1*[,*location2,... , locationn*] or: LOC=*location* : *location* [SOFT] **CPLDs:** LOC = {*pin_name* | FB*nn* | FB*nn_mm}*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

MAP		MAP = [PUC | PUO | PLC | PLO]*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

*Only PUC and PUO are observed. PLC and PLO are translated to PUC and PUO, respectively. The default is PUO.

MAXDELAY		MAXDELAY = *allowable_delay* [*units*]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

MAXSKEW		MAXSKEW = *allowable_skew* [*units*]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

MEDDELAY		MEDDELAY						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
		√				√		

NODELAY		NODELAY						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√			√	√	√	√

NOREDUCE		NOREDUCE						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√				√				

OFFSET		OFFSET={IN | OUT} offset_time [units] {BEFORE | AFTER} "clk_net" [TIMEGRP "reggroup"] or: NET "name" OFFSET={IN | OUT} offset_time [units] {BEFORE | AFTER} "clk_net" [TIMEGRP "reggroup"] or: TIMEGRP "group" OFFSET={IN | OUT} offset_time [units] {BEFORE | AFTER} "clk_net" [TIMEGRP "reggroup"] or: TSidentifier= [TIMEGRP name] OFFSET = {IN | OUT} offset_time [units] {BEFORE | AFTER} "clk_net" [TIMEGRP "reggroup"]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

ONESHOT		ONESHOT						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

OPT_EFFORT		OPT_EFFORT= {NORMAL | HIGH}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√		

OPTIMIZE		OPTIMIZE ={AREA | SPEED | BALANCE | OFF}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√		

OUTREG		OUTREG						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
			√					

PART		PART = *part_type*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

PERIOD		PERIOD = *period*[*units*] [{HIGH | LOW} [*high_or_low_time* [*hi_lo_units*]]] or: TS*identifier*=PERIOD *TNM_reference period*[*units*] [{HIGH | LOW} [*high_or_low_time* [*hi_lo_units*]]]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

PROHIBIT		PROHIBIT = *location1*[, *location2, ... , locationn*] or: PROHIBIT = *location* : *location*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

PULLDOWN		PULLDOWN						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

PULLUP		PULLUP						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

PWR_MODE		PWR_MODE ={LOW | STD}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

REG		REG={ CE | TFF }						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

RLOC		**XC4000E, XC4000X**: RLOC = R*m*C*n*[.*extension*] **XC5200, Spartan2, Virtex:** RLOC = R*m*C*n*.*extension*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

RLOC_ORIGIN		RLOC_ORIGIN = R*m*C*n*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

RLOC_RANGE		RLOC_RANGE = R*m1*C*n1*:R*m2*C*n2*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

S(ave) - Net Flag Attribute		S						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

SLOW		SLOW						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√				

STARTUP_WAIT		STARTUP_WAIT={TRUE | FALSE}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
							√	√

TEMPERATURE		TEMPERATURE=*value*[C | F | K]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√*	√*	√*	√*		√*	√*	√*	√*

*Availability depends on the release of characterization data

TIG		TIG or: TIG= TS*identifier1* [, TS*identifier2*, ... ,TS*identifiern]*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

Time Group Attributes		*new_group_name*=[RISING | FALLING] *group_name1* [EXCEPT g*roup_name2*... *group_namen*] or: *new_group_name*=[TRANSHI | TRANSLO] *group_name1* [EXCEPT *group_name2*... *group_namen*]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

TNM		TNM = [*predefined_group*:] *identifier*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

TNM_NET		TNM_NET = [*predefined_group*:] *identifier*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

TPSYNC		TPSYNC = *identifier*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

TPTHRU		TPTHRU = *identifier*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

TSidentifier		TS*identifier*=[MAXDELAY] FROM *source_group* TO *dest_group allowable_delay* [*units*] or: TS*identifier*=FROM *source_group* TO *dest_group allowable_delay* [*units*] or: TS*identifier*=FROM *source_group* THRU *thru_point* [THRU *thru_point1*... *thru_pointn*] TO *dest_group allowable_delay* [*units*] or: TS*identifier***=**FROM *source_group* TO *dest_group another_TSid*[**/** | *****] *number* or: TS*identifier*=PERIOD *TNM_reference period*[*units*] [{HIGH | LOW} [*high_or_low_time* [*hi_lo_units*]]] or: TS*identifier*=PERIOD *TNM_reference another_PERIOD_identifier* [/ | *] *number* [{HIGH | LOW} [*high_or_low_time* [*hi_lo_units*]]] or: TS*identifier*=FROM *source_group* TO *dest_group* TIG or: TS*identifier*=FROM *source_group* THRU *thru_point* [THRU *thru_point1*... *thru_pointn*] TO *dest_group* TIG NOTE: The use of a colon (:) instead of a space as a separator is optional.							
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√	√	√	√	√	√

U_SET		U_SET = *name*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

USE_RLOC		USE_RLOC = {TRUE | FALSE}						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
	√	√	√		√	√	√	√

VOLTAGE		VOLTAGE=*value*[V]						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√*	√*	√*	√*		√*	√*	√*	√*

*Availability depends on the release of characterization data

WIREAND		WIREAND						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
				√*				

* not supported for XC9500XL and XC9500XV designs

XBLKNM		XBLKNM = *block_name*						
XC3000	**XC4000E**	**XC4000X**	**XC5200**	**XC9000**	**Spartan**	**SpartanXL**	**Spartan2**	**Virtex**
√	√	√	√		√	√	√	√

Attributes/Constraints Applicability

Certain constraints can only be defined at the design level, whereas other constraints can be defined in the various configuration files. The following table lists the constraints and their applicability to the design, and the NCF, UCF, and PCF files.

The CE column indicates which constraints can be entered using the Xilinx Constraints Editor, a GUI tool in the Xilinx Development System. The Constraints Editor passes these constraints to the implementation tools through a UCF file.

A check mark (√) indicates that the constraint applies to the item for that column.

Table 12-1 Constraint Applicability Table

Attribute/Constraint	Design	NCF	UCF	CE	PCF
BASE	√				
BLKNM	√	√	√		
BUFG	√	√	√		
CLKDV_DIVIDE	√	√	√		
COLLAPSE	√	√	√		
COMPGRP					√
CONFIG**	√				
DECODE	√	√	√		
DIVIDE1_BY	√	√			
DIVIDE2_BY	√	√			
DOUBLE	√				
DRIVE	√	√	√	√	
DROP_SPEC		√	√		√*
DUTY_CYCLE_CORRECTION	√	√	√		
EQUATE_F	√				
EQUATE_G	√				
FAST	√	√	√	√	
FILE	√				
FREQUENCY					√
HBLKNM	√	√	√		
HU_SET	√	√	√		
INIT	√	√	√***		
INIT_0x	√	√	√		
INREG	√	√	√		√
IOB	√	√	√		
IOSTANDARD	√	√	√	√	
KEEP	√	√	√		
KEEPER	√	√	√	√	
LOC	√	√	√	√	√*

Table 12-1 Constraint Applicability Table

Attribute/Constraint	Design	NCF	UCF	CE	PCF
LOCATE					√
LOCK					√
MAP	√	√	√		
MAXDELAY	√	√	√		√*
MAXSKEW	√	√	√		√*
MEDDELAY	√	√	√		
NODELAY	√	√	√		
NOREDUCE	√	√	√		
OFFSET		√	√	√	√*
ONESHOT	√				
OPT_EFFORT	√	√	√		
OPTIMIZE	√	√	√		
OUTREG	√	√	√		√
PATH					√
PART	√	√	√		
PENALIZE TILDE					√
PERIOD	√	√	√	√	√*
PIN					√
PRIORITIZE					√
PROHIBIT	√	√	√	√	√*
PULLDOWN	√	√	√	√	
PULLUP	√	√	√	√	
PWR_MODE	√	√	√		
REG	√	√	√		
RLOC	√	√	√		
RLOC_ORIGIN	√	√	√		√
RLOC_RANGE	√	√	√		√
S(ave) - Net Flag attribute	√	√	√		
SITEGRP					√
SLOW	√	√	√	√	
STARTUP_WAIT	√	√	√		
TEMPERATURE	√	√	√	√	√
TIG	√	√	√	√	√*
Time group attributes	√	√	√	√	√
TNM	√	√	√	√	
TNM_NET	√	√	√	√	
TPSYNC	√	√	√		
TPTHRU	√	√	√	√	

Table 12-1 Constraint Applicability Table

Attribute/Constraint	Design	NCF	UCF	CE	PCF
TSidentifier	√	√	√	√	√*
U_SET	√	√	√		
USE_RLOC	√	√	√		
VOLTAGE	√	√	√	√	√
WIREAND	√	√	√		
XBLKNM	√	√	√		

*Use cautiously — although the constraint is available, there are differences between the UCF/NCF and PCF syntax.

**The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

***INIT is allowed in the UCF for CPLDs only.

Macro and Net Propagation Rules

Not all constraints can be attached to nets and macros. The following table lists the constraints and stipulates whether they can be attached to a net, a macro, or neither.

Table 12-2 Macro and Net Propagation Rules

Constraint/Attribute	Action when attached to a net	Action when attached to a macro
BASE	illegal	illegal
BLKNM	illegal	Note 4
BUFG	Note 2	Note 4
CLKDV_DIVIDE	illegal	illegal
COLLAPSE	Note 2	Note 4
CONFIG*	illegal	illegal
DECODE	Note 1	Note 4
DIVIDE1_BY and DIVIDE2_BY	illegal	illegal
DOUBLE	Note 1	Note 4
DRIVE	Note 6	Note4
DROP_SPEC	illegal	illegal
DUTY_CYCLE_CORRECTION	illegal	Note 4
EQUATE_F and EQUATE_G	illegal	illegal
FAST	Note 6	Note 4
FILE	illegal	Note 5
HBLKNM	illegal	Note 4
HU_SET	illegal	Note 4
INIT	FPGA: illegal CPLD: Note 1	Note 4
INIT_0x	illegal	illegal
INREG	illegal	illegal
IOB	illegal	Note 4
IOSTANDARD	Note 6	Note 4
KEEP	Note 3	illegal
KEEPER	Note 6	Note 4
LOC	All: Note 6 CPLD: Also Note 1	Note 4
MAP	illegal	illegal
MAXDELAY	Note 3	illegal
MAXSKEW	Note 3	illegal
MEDDELAY	Note 6	Note 4
NODELAY	Note 6	Note 4
NOREDUCE	Note 3	illegal
OFFSET	Note 3	illegal

Table 12-2 Macro and Net Propagation Rules

Constraint/Attribute	Action when attached to a net	Action when attached to a macro
ONESHOT	illegal	illegal
OPT_EFFORT	illegal	Note 5
OPTIMIZE	illegal	Note 5
OUTREG	illegal	illegal
PART	illegal	illegal
PERIOD	Note 3	illegal
PROHIBIT	illegal	illegal
PULLDOWN	Note 3	illegal
PULLUP	Note 3	illegal
PWR_MODE	Note 1	Note 4
REG	Note 1	Note 4
RLOC	illegal	Note 4
RLOC_ORIGIN	illegal	Note 4
RLOC_RANGE	illegal	Note 4
S(ave) - Net Flag Attribute	Note 3	illegal
SLOW	Note 6	Note 4
STARTUP_WAIT	illegal	Note 4
TEMPERATURE	illegal	illegal
TIG	Note 2	Note 4
Time Group Attributes	illegal	illegal
TNM	Note 2	Note 4
TNM_NET	Note 2	illegal
TPSYNC	Note 3	illegal
TPTHRU	Note 3	illegal
TSidentifier	illegal	illegal
U_SET	illegal	Note 4
USE_RLOC	illegal	Note 4
VOLTAGE	illegal	illegal
WIREAND	Note 3	illegal
XBLKNM	illegal	Note 4

Note 1: Attaches to all applicable elements that drive the net.

Note 2: The attribute has a net form and so no special propagation is required.

Note 3: Attribute is a net attribute and any attachment to a macro is illegal.

Note 4: Propagated to all applicable elements in the hierarchy below the module.

Note 5: Attribute is a macro attribute and any attachment to a net is illegal.

Note 6: Attribute is illegal when attached to a net except when the net is connected to a pad. In this case, the attribute is treated as attached to the pad instance.

*The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

Syntax Descriptions

The information that follows describes in alphabetical order the attributes and constraints. A checkmark (√) appearing in a column means that the attribute / constraint is supported for architectures that use the indicated library. (Refer to the "Applicable Architectures" section of the "Xilinx Unified Libraries" chapter for information on the specific device architectures supported in each library.) A blank column means that the attribute / constraint is not supported for architectures that use that library.

The description for each attribute contains a subsection entitled "Applicable Elements." This section describes the base primitives and circuit elements to which the constraint or attribute can be attached. In many cases, constraints or attributes can be attached to macro elements, in which case some resolution of the user's intent is required. Refer to the "Macro and Net Propagation Rules" section for a table describing the additional propagation rules for each constraint and attribute.

BASE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√								

Applicable Elements

CLB or IOB primitives

Description

The BASE attribute defines the base configuration of a CLB or an IOB. For an IOB primitive, it should always be set to IO. For a CLB primitive, it can be one of three modes in which the CLB function generator operates.

- F mode allows the CLB to implement any one function of up to five variables.
- FG mode gives the CLB any two functions of up to four variables. Of the two sets of four variables, one input (A) must be common, two (B and C) can be either independent inputs or feedback from the Qx and Qy outputs of the flip-flops within the CLB, and the fourth can be either of the two other inputs to the CLB (D and E).
- FGM mode is similar to FG, but the fourth input must be the D input. The E input is then used to control a multiplexer between the two four-input functions, allowing some six- and seven-input functions to be implemented.

CLB and IOB base configurations of the XC3000 family are illustrated in the "IOB and CLB Primitives for Base Configurations XC3000" figure. In this drawing, BASE F, FG, and FGM are for CLBs; BASE IO is for IOBs.

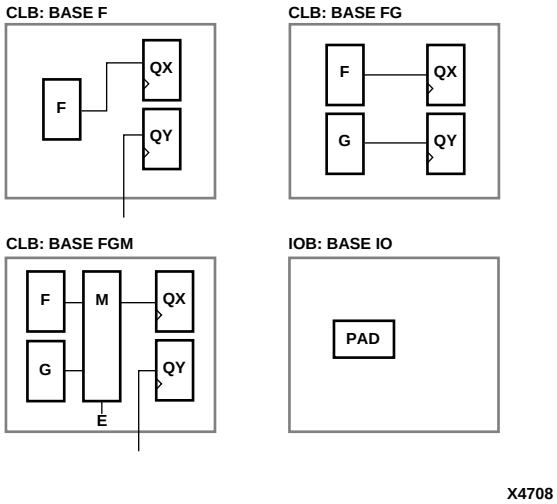


Figure 12-1 IOB and CLB Primitives for Base Configurations XC3000

In a schematic, BASE can be attached to any valid instance. Not supported for UCF, NCF, or PCF files.

Syntax

BASE=*mode*

where *mode* can be F, FG, or FGM for a CLB and IO for an IOB.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

N / A

Constraints Editor

N / A

BLKNM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√ 1, 2, 3, 7, 8	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 6, 7, 11		√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 1, 2, 3, 4, 7, 8, 9, 10, 11	√ 1, 2, 3, 4, 7, 8, 9, 10, 11

Applicable Elements

1. IOB, CLB and CLBMAP (See the Note at the end of this list)
2. Flip-flop and latch primitives
3. Any I/O element or pad
4. FMAP

5. HMAP
6. F5MAP
7. BUFT
8. ROM primitive
9. RAM primitives
10. RAMS and RAMD primitives
11. Carry logic primitives

Note: You can also attach the BLKNM constraint to the net connected to the pad component in a UCF file. NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **BLKNM=***property_value*

Description

BLKNM assigns block names to qualifying primitives and logic elements. If the same BLKNM attribute is assigned to more than one instance, the software attempts to map them into the same block. Conversely, two symbols with different BLKNM names are not mapped into the same block. Placing similar BLKNMs on instances that do not fit within one block creates an error.

Specifying identical BLKNM attributes on FMAP and/or HMAP symbols tells the software to group the associated function generators into a single CLB. Using BLKNM, you can partition a complete CLB without constraining the CLB to a physical location on the device.

BLKNM attributes, like LOC constraints, are specified from the schematic. Hierarchical paths are not prefixed to BLKNM attributes, so BLKNM attributes for different CLBs must be unique throughout the entire design. See the section on the "HBLKNM" attribute for information on attaching hierarchy to block names.

The BLKNM attribute allows any elements except those with a different BLKNM to be mapped into the same physical component. Elements without a BLKNM can be packed with those that have a BLKNM. See the section on the "XBLKNM" attribute for information on allowing only elements with the same XBLKNM to be mapped into the same physical component.

For XC5200, a given BLKNM string can only be used to group a logic cell (LC), which contains one register, one LUT (FMAP), and one F5_MUX element. An error will occur if two or more registers, two or more FMAPs, or two or more F5_MUX elements have the same BLKNM attribute.

Syntax

BLKNM=*block_name*

where *block_name* is a valid LCA block name for that type of symbol. For a list of prohibited block names, see the "Naming Conventions" section of each user interface manual.

For information on assigning hierarchical block names, see the "HBLKNM" section in this chapter.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement assigns an instantiation of an element named block1 to a block named U1358.

```
INST $1I87/block1 BLKNM=U1358;
```

Constraints Editor

N/A

BUFG

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

Applicable Elements

Any input buffer (IBUF), input pad net, or internal net that drives a CLK, OE, or SR pin

Description

When applied to an input buffer or input pad net, th BUFG attribute maps the tagged signal to a global net. When applied to an internal net, the tagged signal is brought out to a global device control pin and then routed to the connected internal control pins via a global net.

Syntax

BUFG={CLK | OE | SR}

where CLK, OE, and SR indicate clock, output enable, or set/reset, respectively.

Example

Schematic

Attach to an IBUF instance of the input pad connected to an IBUF input.

UCF/NCF file

This statement maps the signal named “fastclk” to a global clock net.

```
NET fastclk BUFG=CLK;
```

Constraints Editor

N/A

CLKDV_DIVIDE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

Any CLKDLL or CLKDLLHF instance

Description

CLKDV_DIVIDE specifies the extent to which the CLKDLL or CLKDLLHF clock divider (CLKDV output) is to be frequency divided.

Syntax

CLKDV_DIVIDE={1.5 | 2.0 | 2.5 | 3.0 | 4.0 | 5.0 | 8.0 | 16.0}

The default is 2.0 if no CLKDV_DIVIDE attribute is specified.

Note: The CLKDV_DIVIDE value must be entered as a real number.

Example

Schematic

Attach to a CLKDLL or CLKDLLHF instance.

UCF/NCF file

This statement specifies a frequency division factor of 8 for the clock divider foo/bar.

```
INST foo/bar CLKDV_DIVIDE=8;
```

Constraints Editor

N/A

COLLAPSE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

Applicable Elements

Any internal net

Description

COLLAPSE forces a combinatorial node to be collapsed into all of its fanouts.

Syntax

COLLAPSE

Example

Schematic

Attach to a logic symbol or its output net.

UCF/NCF file

This statement forces net $1N6745 to collapse into all its fanouts.

```
NET $1I87/$1N6745 COLLAPSE;
```

Constraints Editor

N / A

CONFIG

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√								

Applicable Elements

IOB and CLB primitives

Description

CONFIG defines the configuration of the internal options of a CLB or IOB.

Note: Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes. Refer to the “PROHIBIT” section for information on disallowing the use of a site and to the “PART” section for information on defining the part type for the design.

Syntax

CONFIG=*tag value* [*tag value*]

where *tag* and *value* are derived from the following tables.

Table 12-3 XC3000 CLB Configuration Options

Tag	BASE F	BASE FG	BASE FGM*
X	F, QX	F, QX	M, QX
Y	F, QY	G, QY	M, QY
DX	DI, F	DI, F, G	DI, M
DY	DI, F	DI, F, G	DI, M
CLK	K, NOT	K, NOT	K, NOT
RSTDIR	RD	RD	RD
ENCLK	EC	EC	EC
F	A,B,C,D,E,QX, QY	A,B,C,D,E,QX, QY	A,B,C,D,QX, QY
G	None	A,B,C,D,E,QX, QY	A,B,C,D,QX, QY

*For BASE FGM, M=F if E=0, and M=G if E=1

Table 12-4 XC3000 IOB Configuration Options

Tag	BASE IO
IN	I, IQ, IKNOT, FF, LATCH, PULLUP
OUT	O, OQ, NOT, OKNOT, FAST
TRI	T, NOT

Example

Schematic

Attach to a valid instance.

Following is an example of a valid XC3000 CLB CONFIG attribute value.

```
X:QX Y:QY DX:F DY:G CLK:K ENCLK:EC
```

UCF/NCF file

N/A

Constraints Editor

N/A

DECODE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√						

Applicable Elements

WAND1

Description

DECODE defines how a wired-AND (WAND) instance is created, either using a BUFT or an edge decoder. If the DECODE attribute is placed on a single-input WAND1 gate, the gate is implemented as an input to a wide-edge decoder in XC4000 designs.

Syntax

DECODE

DECODE attributes can only be attached to a WAND1 symbol.

Example

Schematic

Attach to a WAND1 symbol.

UCF/NCF file

This statement implements an instantiation of a wired AND using the edge decoder $COMP_1

```
INST address_decode/$COMP_1 DECODE;
```

Constraints Editor

N / A

DIVIDE1_BY and DIVIDE2_BY

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
			√					

Applicable Elements

OSC5, CK_DIV

Description

DIVIDE1_BY and DIVIDE2_BY define the division factor for the on-chip clock dividers.

Syntax

DIVIDE1_BY={4 | 16 | 64 | 256}

DIVIDE2_BY={2 | 8 | 32 | 128 | 1024 | 4096 | 16384 | 65536}

Examples

Schematic

Attach to a valid instance.

NCF file

This statement defines the division factor of 8 for the clock divider $1I45678.

```
INST clk_gen/$1I45678 divide2_by=8;
```

Note: DIVDE1_BY and DIVIDE2_BY are not supported in the UCF file.

Constraints Editor

N / A

DOUBLE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√			√	√		

Applicable Elements

PULLUP components

Description

DOUBLE specifies double pull-up resistors on the horizontal longline. On XC3000 parts, there are internal nets that can be set as 3-state with two programmable pull-up resistors available per line. If the DOUBLE attribute is placed on a PULLUP symbol, both pull-ups are used, enabling a fast, high-power line. If the DOUBLE attribute is not placed on a pull-up, only one pull-up is used, resulting in a slower, lower-power line.

When the DOUBLE attribute is present, the speed of the distributed logic is increased, as is the power consumption of the part. When only half of the longline is used, there is only one pull-up at each end of the longline.

While the DOUBLE attribute can be used for the XC4000, Spartan, and SpartanXL, it is not recommended. The mapper activates both pull-up resistors if the entire longline (not a half-longline) is used. When DOUBLE is used, PAR will not add an additional pull-up to achieve your timing constraints while routing XC4000, Spartan, or SpartanXL designs (refer to the "PAR - Place and Route" Chapter of the *Development System Reference Guide* for information on PAR and timing optimization).

Syntax

DOUBLE

Example

Schematic

Attach to a PULLUP component only.

UCF/NCF file

N / A

Constraints Editor

N / A

DRIVE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
		√* 1				√ 1	√ 2	√ 2

* supported for XC4000XV and XC4000XLA designs only

Applicable Elements

1. IOB output components (OBUF, OFD, etc.)
2. OBUF, OBUFT, IOBUF instances (with implied LVTTL standards)

Description

For the XC4000XV, XC4000XLA, and SpartanXL, DRIVE programs the output drive current from High (24 mA) to Low (12 mA).

For Virtex and Spartan2, DRIVE selects output drive strength (mA) for the components that use the LVTTL interface standard.

Syntax

XC4000XV, XC4000XLA, and SpartanXL

DRIVE={**12** | **24**}

Spartan2, Virtex

DRIVE={**2** | **4** | **6** | **8** | **12** | **16** | **24**}

where 12 mA is the default.

Example

Schematic

Attach to a valid IOB output component.

UCF/NCF file

This statement specifies a High drive.

```
INST /top/my_design/obuf DRIVE=24 ;
```

Constraints Editor

DRIVE current can be selected for any output pad signal in the Ports tab (I/O Configuration Options).

DROP_SPEC

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

All timing specifications. Should be used only in UCF or PCF files.

Description

DROP_SPEC allows you to specify that a timing constraint defined in the input design should be dropped from the analysis. This constraint can be used when new specifications defined in a constraints file do not directly override all specifications defined in the input design, and some of these input design specifications need to be dropped.

While this timing command is not expected to be used much in an input netlist (or NCF file), it is not illegal. If defined in an input design this attribute must be attached to a TIMESPEC primitive.

Syntax

TS*identifier***=DROP_SPEC**

where TS*identifier* is the identifier name used for the timing specification that is to be removed.

Example

Schematic

N/A

UCF/NCF file

This statement cancels the input design specification TS67.

```
TIMESPEC TSidentifier TS67=DROP_SPEC;
```

Constraints Editor

N/A

DUTY_CYCLE_CORRECTION

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

Any CLKDLL, CLKDLLHF, or BUFGDLL instance

Description

DUTY_CYCLE_CORRECTION corrects the duty cycle of the CLK0 output.

Syntax

```
DUTY_CYCLE_CORRECTION={TRUE | FALSE}
```

where TRUE corrects the duty cycle to be a 50_50 duty cycle and FALSE does not change the duty cycle. The default is FALSE.

Example

Schematic

Attach to a CLKDLL, CLKDLLHF, or BUFGDLL instance.

UCF/NCF file

This statement specifies a 50_50 duty cycle for foo/bar.

```
INST foo/bar DUTY_CYCLE_CORRECTION=TRUE;
```

Constraints Editor

N/A

EQUATE_F and EQUATE_G

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√								

Applicable Elements

CLB primitive

Description

EQUATE_F and EQUATE_G set the logic equations describing the F and G function generators of a CLB, respectively.

Syntax

EQUATE_F=*equation*

EQUATE_G=*equation*

where *equation* is a logical equation of the function generator inputs (A, B, C, D, E, QX, QY) using the boolean operators listed in the following table.

Table 12-5 Valid XC3000 Boolean Operators

Symbol	Boolean Equivalent
~	NOT
*	AND
@	XOR
+	OR
( )	Group expression

Example

Schematic

Attach to a valid instance.

Here are two examples illustrating the use of the EQUATE_F attribute.

```
EQUATE_F=F=((~A*B)+D))@Q
F=A@B+(C*~D)
```

UCF/NCF file

N / A

Constraints Editor

N / A

FAST

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Output primitives, output pads, bidirectional pads

Note: You can also attach the FAST attribute to the net connected to the pad component in a UCF file. NGDBuild transfers the attribute from the net to the pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **FAST**

Description

FAST increases the speed of an IOB output.

Note: The FAST attribute produces a faster output but may increase noise and power consumption.

Syntax

FAST

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement increases the output speed of the element y2.

```
INST $1I87/y2 FAST;
```

This statement increases the output speed of the pad to which net1 is connected.

```
NET net1 FAST;
```

Constraints Editor

FAST slew rate can be selected for any output pad signal in the Ports tab (I/O Configuration Options).

FILE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Macros that refer to underlying non-schematic designs

Description

FILE is attached to a macro (a custom symbol) that does not have an underlying schematic. It identifies the file to be looked at for the logic definition of that macro.The type of file to be searched for is defined by the search order of the program compiling the design.

Syntax

FILE=*file_name*[**.***extension*]

where *file_name* is the name of a file that represents the underlying logic for the element carrying the attribute. Example file types include EDIF, EDN, NMC.

Schematic

Attach to a valid instance.

UCF/NCF file

N/A

Constraints Editor

N/A

HBLKNM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√ 1, 2, 3, 7, 8, 9, 10,12	√ 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15	√ 2, 3, 4, 5, 7, 8, 10, 12, 13, 14, 15	√ 2, 3, 4, 6, 7, 10, 15		√ 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15	√ 2, 3, 4, 5, 7, 8, 10, 12, 13, 14, 15		

Applicable Elements

1. IOB, CLB and CLBMAP (See Note below)
2. Registers
3. I/O elements and pads
4. FMAP
5. HMAP
6. F5MAP
7. BUFT
8. PULLUP
9. ACLK, GCLK
10. BUFG
11. BUFGS, BUFGP
12. ROM
13. RAM

14. RAMS and RAMD
15. Carry logic primitives

Note: You can also attach the HBLKNM constraint to the net connected to the pad component in a UCF file. NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **HBLKNM=***property_value*

Description

HBLKNM assigns hierarchical block names to logic elements and controls grouping in a flattened hierarchical design. When elements on different levels of a hierarchical design carry the same block name and the design is flattened, NGDBuild prefixes a hierarchical path name to the HBLKNM value.

Like BLKNM, the HBLKNM attribute forces function generators and flip-flops into the same CLB. Symbols with the same HBLKNM attribute map into the same CLB, if possible. However, using HBLKNM instead of BLKNM has the advantage of adding hierarchy path names during translation, and therefore the same HBLKNM attribute and value can be used on elements within different instances of the same macro.

For XC5200, a given HBLKNM string can only be used to group a logic cell (LC), which contains one register, one LUT (FMAP), and one F5_MUX element. An error will occur if two or more registers, two or more FMAPs, or two or more F5_MUX elements have the same HBLKNM attribute.

Syntax

HBLKNM=*block_name*

where *block_name* is a valid LCA block name for that type of symbol.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement specifies that the element this_hmap will be put into the block named group1.

```
INST $I13245/this_hmap HBLKNM=group1;
```

This statement attaches the HBLKNM constraint to the pad connected to net1.

```
NET net1 HBLKNM=$COMP_0;
```

Note: Elements with the same HBLKNM are placed in the same logic block if possible. Otherwise an error occurs. Conversely, elements with different block names will not be put into the same block.

Constraints Editor

N / A

HU_SET

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√ 1, 2, 3, 5, 7, 8, 9, 10, 12	√ 1, 2, 3, 5, 7, 8, 9, 10, 12	√ 1, 2, 4, 6, 7, 8, 12		√ 1, 2, 3, 5, 7, 8, 9, 10, 12	√ 1, 2, 3, 5, 7, 8, 9, 10, 12	√ 1,2, 7, 11, 12	√ 1,2, 7, 11, 12

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. Macro Instance
8. EQN
9. ROM
10. RAM
11. RAMS, RAMD
12. BUFT

Description

The HU_SET constraint is defined by the design hierarchy. However, it also allows you to specify a set name. It is possible to have only one H_SET constraint within a given hierarchical element (macro) but by specifying set names, you can specify several HU_SET sets.

NGDBuild hierarchically qualifies the name of the HU_SET as it flattens the design and attaches the hierarchical names as prefixes. The difference between an HU_SET constraint and an H_SET constraint is that an HU_SET has an explicit user-defined and hierarchically qualified name for the set, but an H_SET constraint has only an implicit hierarchically qualified name generated by the design-flattening program. An HU_SET set “starts” with the symbols that are assigned the HU_SET constraint, but an H_SET set “starts” with the instantiating macro one level above the symbols with the RLOC constraints.

For background information about using the various set attributes, refer to the “RLOC Sets” section.

Syntax

HU_SET=*set_name*

where set_*name* is the identifier for the set; it must be unique among all the sets in the design.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement assigns an instance of the register FF_1 to a set named heavy_set.

```
INST $1I3245/FF_1 HU_SET=heavy_set;
```

Constraints Editor

N / A

INIT

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√ 1, 2, 3	√ 1, 2, 3		√ 3	√ 1, 2, 3	√ 1, 2, 3	√ 2, 3, 4	√ 2, 3, 4

Applicable Elements

1. ROM
2. RAM
3. Registers
4. LUTs, SRLs

Description

INIT Initializes ROMs, RAMs, registers, and look-up tables. The least significant bit of the value corresponds to the value loaded into the lowest address of the memory element. For register initialization, S indicates Set and R indicates Reset. The INIT attribute can be used to specify the initial value directly on the symbol with the following limitation. INIT may only be used on a RAM or ROM that is 1 bit wide and not more than 32 bits deep.

Syntax

INIT={*value* | **S** | **R**}

where *value* is a 4-digit or 8-digit hexadecimal number that defines the initialization string for the memory element, depending on whether the element is 16-bit or 32-bit. For example, INIT=ABAC1234. If the INIT attribute is not specified, the RAM is initialized with zero.

Note: For RAM32X1S and RAM32X1S_1, mapping of the upper and lower INIT values to the F and G function generators are handled differently for Virtex and Spartan2 than they are for XC4000E, XC4000X, Spartan, and SpartanXL. Lower INIT values get mapped to the F and upper INIT values get mapped to the G for XC4000E, XC4000X, Spartan, and SpartanXL. For Virtex and Spartan2, lower INIT values get mapped to the G function generator and upper INIT values get mapped to the F function generator.

S indicates Set and R indicates Reset for registers.

Note: For XC4000E, XC4000X, Spartan, and SpartanXL, INIT cannot specify a register as Set if the reset pin is being used or as Reset if the set pin is being used.

Example

Schematic

Attach to a net, pin, or instance.

UCF/NCF file

INIT={S | R} is supported in both the NCF and UCF files. It is allowed in the UCF to control the startup state of registers (primarily for CPLDs).

INIT=*value* is supported in the NCF file only. This statement defines the initialization string for an instantiation of the memory element ROM2 to be the 16-bit hexadecimal string 5555.

```
INST $1I3245/ROM2 INIT = 5555;
```

Note: INIT=*value* is not supported in the UCF file.

Constraints Editor

N/A

INIT_0x

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

Block RAMs

Description

INIT_0x specifies initialization strings for block RAM components.

Syntax

INIT_0*x***=***value*

where

x is any hexadecimal value 0 through F that specifies which 256 bits (see the following table) of the 4096-bit block RAM to initialize to the specified value.

value is a string of hexadecimal characters up to 64 digits wide. If the INIT_0x attribute has a value less than the required 64 hex digits, the value will be padded with zeros from the most significant bit (MSB) side. This fills the 256 bits in the initialization string (4 bits per hexadecimal character * 64 characters).

INIT_0x	Addresses				
	4096 x 1	**2048 x 2**	**1024 x 4**	**512 x 8**	**256 x 16**
INIT_00	255 — 0	127 — 0	63 — 0	31 — 0	15 — 0
INIT_01	511 — 256	255 — 128	127— 64	63 — 32	31 — 16
INIT_02	767 — 512	383 — 256	191 — 128	95 — 64	47 — 32
INIT_03	1023 — 768	511 — 384	255 — 192	127 — 96	63 — 48
INIT_04	1279 — 1024	639 — 512	319 — 256	159 — 128	79 — 64
INIT_05	1535 — 1280	767 — 640	383 — 320	191 — 160	95 — 80
INIT_06	1791 — 1536	895 — 768	447 — 384	223 — 192	111 — 96
INIT_07	2047 — 1792	1023 — 896	511 — 448	255 — 224	127 — 112
INIT_08	2303 — 2048	1151 — 1024	575 — 512	287 — 256	143 — 128
INIT_09	2559 — 2304	1279 — 1152	639 — 576	319 — 288	159 — 144
INIT_0A	2815 — 2560	1407 — 1280	703 — 640	351 — 320	175 — 160
INIT_0B	3071 — 2816	1535 — 1408	767 — 704	383 — 352	191 — 176
INIT_0C	3327 — 3072	1663 — 1536	831 — 768	415 — 384	207 — 192
INIT_0D	3583 — 3328	1791 — 1664	895 — 832	447 — 416	223 — 208
INIT_0E	3839 — 3584	1919 — 1792	959 — 896	479 — 448	239 — 224
INIT_0F	4095 — 3840	2047 — 1920	1023 — 960	511 — 480	255 — 240

INIT_0x Usage Rules

A summary of the rules for the INIT_0x attribute follows.

- If no INIT_0x attribute is attached to a block RAM, the contents of the RAM defaults to zero.
- Each initialization string defines 256 bits of the 4096-bit block RAM. For example, for a 4096-bit deep x 1-bit wide block RAM, INIT_00 assigns the 256 bits to addresses 0 through 255 and INIT_01 assigns the 256 bits to addresses 256 through 511. For a 2048-bit deep x 2-bit wide block RAMs, INIT_00 assigns the 256 bits to addresses 0 through 127 (a 2-bit value at each address) and INIT_01 assigns the 256 bits to addresses 128 through 255.
- If a subset of the INIT_00 through INIT_0F properties are specified for a block RAM, the remaining properties default to zero.
- In an initialization string, the least significant bit (LSB) is the right-most value.
- The least significant word of the block RAM address space specified by INIT_0x is composed of the least significant bits of the block RAM INIT_0x attribute.

INIT_0x on Block RAMs of Various Widths

The initialization string "fills" the block RAM beginning from the LSB of the 256 bits for the specified INIT_0x addresses. The size of the word filling each address depends on the width of the block RAM being initialized— 1, 2, 4, 8, or 16 bits.

For example, if INIT_0C=bcde7, the corresponding binary sequence is as follows:

1011	1100	1101	1110	0111	←LSB
b	c	d	e	7	

The appropriate addresses in the RAM are initialized with the binary string content depending on the width of the RAM as shown in the following table.

Block RAM (depth x width)	Address (INIT_0C)	Contents
4096 x 1	3072 3073 3074 3075 . 3327	1 1 1 0 . 0
2048 x 2	1536 1537 1538 1539 . 1663	11 01 10 11 . 00
1024 x 4	768 769 770 771 . 831	0111 1110 1101 1100 . 0000
512 x 8	384 385 386 387 . 415	11100111 11001101 00001011 00000000 . 00000000
256 x 16	192 193 194 195 . 207	1100110111101111 0000000000001011 0000000000000000 0000000000000000 . 0000000000000000

Example

Schematic

Attach to a block RAM instance.

UCF/NCF file

The following statement specifies that the INIT_03 addresses in instance foo/bar be initialized, starting from the LSB, to the hex value aaaaaaaaaaaaaaaaaaaa (padded with 44 zeros from the MSB side).

```
INST foo/bar INIT_03=aaaaaaaaaaaaaaaaaaaa;
```

Constraints Editor

N/A

INREG

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
			√					

Applicable Elements

Flip-flops, latches

Description

Because XC5200 IOBs do not have flip-flops or latches, you can apply the INREG attribute to meet fast setup timing requirements. If a flip-flop or latch is driven by an IOB, you can specify INREG to enable PAR (Place and Route) to place the flip-flop/latch close to the IOB so that the two elements can be connected using fast routes. See also the "OUTREG" section.

Syntax

INREG

Example

Schematic

Attach to a latch or flip-flop instance.

UCF/NCF file

This statement directs PAR to place the flip-flop $I1 near the IOB driving it.

```
INST $I1 INREG;
```

Constraints Editor

N/A

IOB

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

Non-INFF / OUTFF flip-flop and latch primitives, registers

Description

IOB indicates which flip-flops and latches can be moved into the IOB. The mapper supports a command line option (-pr i | o | b) that allows flip-flop / latch primitives to be pushed into the input IOB (i), output IOB (o), or input / output IOB (b) on a global scale. The IOB constraint, when associated with a flip-flop or latch, tells the mapper to pack that instance into an IOB type component if possible. The IOB constraint has precedence over the mapper "-pr" command line option.

Syntax

IOB={**TRUE** | **FALSE**}

where TRUE allows the flip-flop / latch to be pulled into an IOB and FALSE indicates not to pull it into an IOB.

Example

Schematic

Attach to a flip-flop or latch instance or to a register.

UCF/NCF file

This statement prevents the mapper from placing the foo / bar instance into an IOB component.

```
INST foo/bar IOB=TRUEE;
```

Constraints Editor

N / A

IOSTANDARD

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

IBUF, IBUFG, IOBUF, OBUF, OBUFT

Description

The IOSTANDARD attribute can be used to assign an I/O standard to an I/O primitive. All components with the IOSTANDARD attribute must follow the same placement rules as the SelectI/O components. Refer to the "SelectI/O Usage Rules" in the IBUF_*selectIO* section for information on the placement rules.

Syntax

IOSTANDARD=*iostandard_name*

where *iostandard_name* is one of the following:

AGP	HSTL_I	LVCMOS2	SSTL2_I
CTT	HSTL_III	PCI33_3	SSTL2_II
GTL	HSTL_IV	PCI33_5	SSTL3_I
GTLP	LVTTL	PCI66_3	SSTL3_II

The default is LVTTL if no IOSTANDARD attribute is specified.

Example

Schematic

Attach to an I/O primitive.

UCF/NCF file

These statements configure the IO to the GTL standard.

```
INST "pad_instance_name" IOSTANDARD=GTL;
NET "pad_net_name" IOSTANDARD=GTL;
```

Constraints Editor

An IOSTANDARD can be selected for an input or output pad signal in the Ports tab (I/O Configuration Options).

KEEP

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Nets

Description

When a design is mapped, some nets may be absorbed into logic blocks. When a net is absorbed into a block, it can no longer be seen in the physical design database. This may happen, for example, if the components connected to each side of a net are mapped into the same logic block. The net may then be absorbed into the block containing the components. The KEEP constraint prevents this from happening.

In Virtex and Spartan2, KEEP makes the signal visible at the BEL level, not the CLB level as in other architectures.

Note: The KEEP property is translated into an internal constraint known as NOMERGE when targeting an FPGA. Messaging from the implementation tools will therefore refer to the system property NOMERGE—not KEEP.

Syntax

KEEP

Example

Schematic

Attach to a net.

UCF/NCF file

This statement ensures that the net $SIG_0 will remain visible.

```
NET $1I3245/$SIG_0 KEEP;
```

Constraints Editor

N/A

KEEPER

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

Tri-state output pad nets: OBUFT, OBUFT_*selectIO*, OBUFE, and OBUFE_*selectIO* components

Description

KEEPER retains the value of the output net it is attached to. For example, if logic 1 is being driven onto the net, KEEPER drives a weak/resistive 1 onto the net. If the net driver is then tri-stated, KEEPER continues to drive a weak/resistive 1 onto the net.

Note: The KEEPER constraint must follow the same banking rules as the KEEPER component. Refer to the “SelectI/O Usage Rules” in the "IBUF_*selectIO*" section for information on the banking rules.

Syntax

KEEPER

Example

Attach to an output pad net.

UCF/NCF file

These statements configure the IO to use the KEEPER option.

```
NET "pad_net_name" KEEPER;
INST "pad_instance_name" KEEPER;
```

Constraints Editor

KEEPER can be selected for an output or bidirectional pad signal in the Ports tab (I/O Configuration Options).

LOC

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√ 1, 5, 6, 12	√ 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15	√ 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15	√ 1, 2, 4, 5, 8, 12, 14	√ 1, 5, 16	√ 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15	√ 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15	√ 1, 2, 5, 6, 10, 11, 12, 13, 14, 15, 16, 17	√ 1, 2, 5, 6, 10, 11, 12, 13, 14, 15, 16, 17

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. IO elements
6. CLB and IOB primitives, CLBMAP
7. CY4
8. CY_MUX
9. ROM
10. RAM
11. RAMS, RAMD
12. BUFT
13. WAND
14. Clock buffers
15. Edge decoders
16. Any instance
17. RAMB4s

Description for FPGAs

LOC defines where a symbol can be placed within an FPGA. It specifies the absolute placement of a design element on the FPGA die. It can be a single location, a range of locations, or a list of locations. The LOC constraint can be specified from the schematic and statements in a constraints file can also be used to direct placement.

You can specify multiple locations for the same symbol by using a comma (,) to separate each location within the field. It specifies that the symbols be placed in any of the locations specified. Also, you can specify an area in which to place a symbol or group of symbols.

The legal names are a function of the target part type. However, to find the correct syntax for specifying a target location, you can load an empty part into the FPGA Editor (the design editor). Place the cursor on any block and click to display its location in the FPGA Editor history area. Do not include the pin name such as .I, .O, or .T as part of the location.

You can use the LOC constraint for logic that uses multiple CLBs, IOBs, soft macros, or other symbols. To do this, use the LOC attribute on a soft macro symbol, which passes the location information down to the logic on the lower level. The location restrictions are applied to all blocks on the lower level for which LOCs are legal.

XC5200

The XC5200 CLB is divided into four physical site locations that each contain one flip-flop, one function generator, and one carry logic element. Therefore, for the XC5200, each LOC attribute can be used for only one register, one FMAP, one F5_MUX element, or one CY_MUX element. An error will occur if two or more registers, two or more FMAPs, two or more F5_MUX elements, or two or more CY_MUX elements have the same LOC attribute.

Spartan2, Virtex

The physical site specified in the location value is defined by the row and column numbers for the array, with an optional extension to define the slice for a given row / column location. A Virtex or Spartan2 slice is composed of two LUTs (that can be configured as RAM or shift registers), two flip-flops (that can also be configured as latches), two XORCYs, two MULT_ANDs, one F5MUX, one F6MUX, and one MUXCY. Only one F6MUX can be used between the two adjacent slices in a specific row / column location. The two slices at a specific row / column location are adjacent to one another.

The block RAMs (RAMB4s) have a different row / column grid specification than the CLB and TBUFs. A block RAM located at RAMB4_R3C1 is *not* located at the same site as a flip-flop located at CLB_R3C1. Therefore, the location value must start with "CLB," "TBUF," or "RAMB4." The location cannot be shortened to reference only the row, column, and extension.The optional extension specifies the left-most or right-most slice for the row / column. While the XC4000, Spartan, and SpartanXL allow extensions such as .FFX, .FFY, .F and .G to identify specific flip-flops and LUTs within the CLB, these extensions are not required or allowed for Virtex and Spartan2.

The location value for global buffers and DLL elements is the specific physical site names for available locations.

Description for CPLDs

For CPLDs, use the LOC=*pin_name* attribute on a PAD symbol or pad net to assign the signal to a specific pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD. You can use the LOC=FBnn attribute on any instance or its output net to assign the logic or register to a specific function block or macrocell, provided the instance is not collapsed.

Pin assignments and function block assignments are unconditional; that is, the software does not attempt to relocate a pin if it cannot achieve the specified assignment. You can apply the LOC constraint to as many symbols in your design as you like. However, each assignment further constrains the software as it automatically allocates logic and I / O resources to internal nodes and I / O pins with no LOC constraints.

The LOC=FB*nn_mm* attribute on any internal instance or output pad assigns the corresponding logic to a specific function block or macrocell within the CPLD. If a LOC is placed on a symbol that does not get mapped to a macrocell or is otherwise removed through optimization, the LOC will be ignored.

Note: Pin assignment using the LOC attribute is not supported for bus pad symbols such as OPAD8.

Location Types

Use the following location types to define the physical location of an element.

P12	IOB location (chip carrier)
A12	IOB location (pin grid or CSP package)
B, L, T, R	Indicates edge locations (bottom, left, top, right) — applies to edge decoders only
LB, RB, LT, RT, BR, TR, BL, TL	Indicates half edges (left bottom, right bottom, and so forth) — applies to edge decoders only
TL, TR, BL, BR	Indicates a corner for global buffer placement
AA	CLB location for XC3000
CLB_R4C3	CLB location for XC4000, XC5200, Spartan, SpartanXL
CLB_R4C3 (or .S0 or .S1)	CLB location for Spartan2, Virtex
CLB_R6C8.F (or .G)	Function generator, RAM, ROM, or RAMS location for XC4000, Spartan, SpartanXL
CLB_R6C8.LC0 (or .LC1, .LC2, .LC3)	Function generator or register location for XC5200
CLB_R6C8.S0 (or .S1)	Function generator or register slice for Spartan2, Virtex
CLB_R6C8.LC0 (or .LC2)	F5_MUX location for XC5200
CLB_R6C8.FFX (or.FFY)	Flip-flop location for XC4000, Spartan, SpartanXL
TBUF_R6C7.1 (or.2)	TBUF location for XC4000, Spartan, SpartanXL
TBUF_R6C7.0 (or .1, .2, or .3)	TBUF location for XC5200
TBUF_R6C7 (or .0 or .1)	TBUF location for Spartan2, Virtex
RAMB4_R3C1	Block RAM location for Spartan2, Virtex
GCLKBUF0 (or 1, 2, or 3)	Global clock buffer location for Spartan2, Virtex
GCLKPAD0 (or 1, 2, or 3)	Global clock pad location for Spartan2, Virtex
DLL0 (or 1, 2, or 3)	Delay Locked Loop element location for Spartan2, Virtex

The wildcard character (*) can be used to replace a single location with a range as shown in the following examples.

C*	Any CLB in row C of an XC3000 device
*D	Any CLB in column D of an XC3000 device

CLB_R*C5	Any CLB in column 5 of an XC4000, XC5200, Spartan, or SpartanXL device
CLB_R*C5	Any CLB in either slice in column 5 of a Virtex or Spartan2 device

Note: The wildcard character is *not* supported for Virtex or Spartan2 global buffer or DLL locations.

The following are *not* supported.

- Dot extensions on ranges. For example, LOC=CLB_R0C0:CLB_R5C5.G. However, for the XC5200, range locations will be expanded to include extensions, CLB_R0C0.*:CLB_R5C5.*, for example, when the mapper passes a range constraint to the PCF file.
- B, L, R, T used to indicate IO edge locations (bottom, left, top, right)
- LB, RB, LT, RT, BR, TR, BL, TL used to indicate IO half edges (left bottom, right bottom, etc.)
- Wildcard character for Virtex or Spartan2 global buffer, global pad, or DLL locations.

Syntax for FPGAs

Single location

LOC=*location*

where *location* is a legal LCA location for the LCA part type. Examples of the syntax for single LOC constraints are given in the "Single LOC Constraint Examples" table.

Table 12-6 Single LOC Constraint Examples

Attribute	Description
LOC=P12	Place I/O at location P12.
LOC=B	Place decode logic on the bottom edge.
LOC=TL	Place decode logic on the top left edge, or global buffer in the top left corner.
LOC=AA (XC3000)	Place logic in CLB AA.
LOC=TBUF.AC.2 (XC3000)	Place BUFT in TBUF above and one column to the right of CLB AC.
LOC=CLB_R3C5 (XC4000, Spartan, SpartanXL)	Place logic in the CLB in row 3, column 5.
LOC=CLB_R3C5 (Spartan2, Virtex)	Place logic in either slice of the CLB in row3, column 5.
LOC=CLB_R4C4.LC0 (XC5200)	Place logic in the lowest slice of the CLB in row 4, column 4.
LOC=CLB_R3C5.S0 (Spartan2, Virtex)	Place logic in the right-most slice of the CLB in row 3, column 5.
LOC=CLB_R4C5.ffx (XC4000, Spartan, SpartanXL)	Place CLB flip-flop in the X flip-flop of the CLB in row 4, column 5.

Table 12-6 Single LOC Constraint Examples

Attribute	Description
LOC=CLB_R4C5.F (XC4000, Spartan, SpartanXL)	Place CLB function generator in the F generator of row 4, column 5.
LOC=TBUF_R2C1.1 (XC4000, Spartan, SpartanXL)	Place BUFT in row 2, column 1, along the top.
LOC=TBUF_R4C4.3 (XC5200)	Place BUFT in the top buffer in row 4, column 4.
LOC=TBUF_R*C0 (XC4000, XC5200, Spartan, SpartanXL)	Place BUFT in any row in column 0.
LOC=TBUF_R1C2.* (Spartan2, Virtex)	Place both TBUFs in row 1, column 2.
RAMB4_R*C1 (Spartan2, Virtex)	Specifies any block RAM in column 1 of the block RAM array

Multiple locations

LOC=*location1*,*location2*,...,*locationn*

Repeating the LOC constraint and separating each such constraint by a comma specifies multiple locations for an element. When you specify multiple locations, PAR can use any of the specified locations. Examples of multiple LOC constraints are provided in the following table.

Table 12-7 Multiple LOC Constraint Examples

Attribute	Description
LOC=T,B (XC4000, Spartan, SpartanXL)	Place decoder (XC4000) on the top or bottom edge.
LOC=clb_r2c4, clb_r7c9 (XC4000, Spartan, SpartanXL)	Place the flip-flop in either CLB R2C4 or CLB R7C9.
LOC=clb_r4c5.s1, clb_r4c6.* (Spartan2, Virtex)	Place the flip-flop in the left-most slice of CLB R4C5 or in either slice of CLB R4C6

Range of locations

LOC=*location*:*location* [**SOFT**]

You can define a range by specifying the two corners of a bounding box. Specify the upper left and lower right corners of an area in which logic is to be placed. Use a colon (:) to separate the two boundaries. The logic represented by the symbol is placed somewhere inside the bounding box. The default is to interpret the constraint as a "hard" requirement and to place it within the box. If SOFT is specified, PAR may place the constraint elsewhere if better results can be obtained at a location outside the bounding box. Examples of LOC constraints used to specify an area (range) are given in the "Area LOC Constraint Examples" table.

Table 12-8 Area LOC Constraint Examples

Attribute	Description
`LOC=AA:FF` (XC3000)	Place CLB logic anywhere in the top left corner of the LCA bounded by row F and column F.
`LOC=CLB_R1C1:CLB_R5C5` (XC4000, Spartan, SpartanXL)	Place logic in the top left corner of the LCA in a 5 x 5 area bounded by row 5 and column 5.

`LOC=CLB_R1C1:CLB_R5C5`
`PROHIBIT=CLB_R5C5`
(must be specified in one continuous line)
(XC4000, Spartan, SpartanXL)
	Place CLB logic in the top left corner of the LCA in a 5 x 5 area, but not in the CLB in row 5, column 5.

`LOC=CLB_R1C1.LC3:CLB_R4C4.LC0` (XC5200)	Place logic in any slice in the top left corner of the LCA bounded by row 4, column 4.
`LOC=CLB_R1C1:CLB_R4C4` (Spartan2, Virtex)	Place logic in either slice in the top left corner of the LCA bounded by row 4, column 4.
`LOC=TBUF_R1C1:TBUF_R2C8` (XC4000, XC5200, Spartan, SpartanXL)	Place BUFT anywhere in the area bounded by row 1, column 1 and row 2, column 8.

Note: For area constraints, LOC ranges can be supplemented by the user with the keyword SOFT.

Syntax for CPLDs

LOC=*pin_name*

or

LOC=FB*nn*

or

LOC=FB*nn_mm*

where

pin_name is P*nn* for PC, PQ, or VQ packages; *nn* is a pin number. The pin name is *rc* (row number and column number) for CSP and BGA packages. See the appropriate data book for the pin package names, for example, p12. Examples are LOC=P24 and LOC=G2. This form is valid only on pad instances.

nn is a function block number and *mm* is a macrocell within a function block number. This form is valid on any instances.

Examples

Refer to the “Placement Constraints” section for multiple examples of legal placement constraints for each type of logic element (flip-flops, ROMs and RAMs, block RAMS, FMAPs and HMAPs, CLBMAPs, BUFTs, CLBs, IOBs, I/Os, edge decoders, global buffers) in FPGA designs.

Schematic

Attach to an instance.

UCF/NCF file

This specifies that an instance of the element BUF1 be placed above the CLB in row 6, column 9. For XC4000, Spartan, or SpartanXL devices, you can place the TBUF above or below the CLB. For XC5200 devices, you can place the TBUF in one of four locations (.0-.3).

```
INST /DESIGN1/GROUPS/BUF1 LOC=TBUF_R6C9.1 ;
```

This specifies that each instance found under “FLIP_FLOPS” is to be placed in any CLB in column 8.

```
INST /FLIP_FLOPS/* LOC=CLB_R*C8;
```

This specifies that an instantiation of MUXBUF_D0_OUT be placed in IOB location P110.

```
INST MUXBUF_D0_OUT LOC=P110 ;
```

This specifies that the net DATA<1> be connected to the pad from IOB location P111.

```
NET DATA<1> LOC=P111 ;
```

Constraints Editor

Location constraints for input and output pad signals can be entered in the Ports tab.

MAP

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√ 4	√ 1, 2	√ 1, 2	√ 1, 3		√ 1, 2	√ 1, 2	√ 1	√ 1

Applicable Elements

1. FMAP
2. HMAP
3. F5MAP
4. CLBMAP

Description

Place the MAP attribute on an FMAP, F5MAP, HMAP, or CLBMAP to specify whether pin swapping and the merging of other functions with the logic in the map are allowed. If merging with other functions is allowed, other logic can also be placed within the CLB, if space allows.

Syntax

MAP=[PUC | PUO | PLC | PLO]

where

PUC means that the CLB pins are unlocked, and the CLB is closed.

PUO means that the CLB pins are unlocked, and the CLB is open.

PLC means that the CLB pins are locked, and the CLB is closed.

PLO means that the CLB pins are locked, and the CLB is open.

"Unlocked" in these definitions means that the software can swap signals among the pins on the CLB; "locked" means that it cannot. "Open" means that the software can add or remove logic from the CLB; conversely, "closed" indicates that the software cannot add or remove logic from the function specified by the MAP symbol.

The default is PUO.

Note: Currently, only PUC and PUO are observed. PLC and PLO are translated into PUC and PUO, respectively.

Example

Schematic

Attach to a map symbol instance.

UCF/NCF file

This statement allows pin swapping and ensures that no logic other than that defined by the original map will be mapped into the function generators.

```
INST $1I3245/map_of_the_world map=puc;
```

Constraints Editor

N/A

MAXDELAY

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets

Description

The MAXDELAY attribute defines the maximum allowable delay on a net.

Syntax

MAXDELAY=*allowable_delay*[*units*]

where *units* may be ps, ns, us, ms, GHz, MHz, or kHz. The default is ns.

Example

Schematic

Attach to a net.

UCF/NCF file

This statement assigns a maximum delay of 1 us to the net $SIG_4.

```
NET $1I3245/$SIG_4 MAXDELAY=1us;
```

Constraints Editor

N/A

MAXSKEW

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets

Description

MAXSKEW defines the allowable skew on a net.

Syntax

MAXSKEW=*allowable_skew* [*units*]

where *units* may be ps, ns, us, ms, GHz, MHz, or kHz. The default is ns.

Example

Schematic

Attach to a net.

UCF/NCF file

This statement specifies a maximum skew of 3 ns on net $SIG_6.

```
NET $1I3245/$SIG_6 MAXSKEW=3;
```

Constraints Editor

N/A

MEDDELAY

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
		√				√		

Applicable Elements

Input register

Note: You can also attach the MEDDELAY constraint to a net that is connected to a pad component in a UCF file. NGDBuild transfers the constraint from the net to the

pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **MEDDELAY**

Description

MEDDELAY specifies a medium sized delay for the IOB register.

Syntax

MEDDELAY

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement specifies that the register in the IOB $COMP_6 will have a medium sized delay.

```
INST $1I87/$COMP_6 MEDDELAY;
```

This statement assigns a medium sized delay to the pad to which net1 is connected.

```
NET Net1 MEDDELAY ;
```

Constraints Editor

N/A

NODELAY

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√			√	√	√	√

Applicable Elements

Input register

Note: You can also attach the NODELAY constraint to a net connected to a pad component in a UCF file. NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **NODELAY**

Description

The default configuration of IOB flip-flops in XC4000, Spartan, and SpartanXL designs includes an input delay that results in no external hold time on the input data path. However, this delay can be removed by placing the NODELAY attribute on input flip-flops or latches, resulting in a smaller setup time but a positive hold time.

The NODELAY attribute can be attached to the I/O symbols and the special function access symbols TDI, TMS, and TCK.

Syntax

NODELAY

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement specifies that IOB register inreg67 not have an input delay.

```
INST $1I87/inreg67 NODELAY;
```

This statement specifies that there be no input delay to the pad that is attached to net1.

```
NET net1 NODELAY ;
```

Constraints Editor

N/A

NOREDUCE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

Applicable Elements

Any net

Description

NOREDUCE prevents minimization of redundant logic terms that are typically included in a design to avoid logic hazards or race conditions. NOREDUCE also identifies the output node of a combinatorial feedback loop to ensure correct mapping. When constructing combinatorial feedback latches in a design, always apply NOREDUCE to the latch's output net and include redundant logic terms when necessary to avoid race conditions.

Syntax

NOREDUCE

Example

Schematic

Attach to a net.

UCF/NCF file

This statement specifies that there be no Boolean logic reduction or logic collapse from the net named $SIG_12 forward.

```
NET $SIG_12 NOREDUCE;
```

Constraints Editor

N / A

OFFSET

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Global, nets, time groups

Description

OFFSET specifies the timing relationship between an external clock and its associated data-in or data-out pin. Used only for pad-related signals and cannot be used to extend the arrival time specification method to the internal signals in a design.

OFFSET constraints allow you to do the following.

- Calculate whether a setup time is being violated at a flip-flop whose data and clock inputs are derived from external nets.
- Specify the delay of an external output net derived from the Q output of an internal flip-flop being clocked from an external device pin.

For CPLD designs, clock inputs referenced by OFFSET constraints must be explicitly assigned to a global clock pin (using either the BUFG symbol or applying the BUFG=CLK attribute to an ordinary input). Otherwise, the OFFSET constraint will not be used during timing-driven optimization of the design.

Syntax

Global method

The OFFSET constraint can be a "global" constraint that applies to all data pad nets in the design for the specified clock.

OFFSET={**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} "*clk_net*" [**TIMEGRP** "*reggroup*"]

Net-Specific method

When the NET "name" specifier is used, the constraint is associated with a specific net.

NET "*name*" **OFFSET=**{**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} "*clk_net*" [**TIMEGRP** "*reggroup*"]

Group method

When the TIMEGRP "group" specifier is used, the constraint is associated with a group of data pad nets.

TIMEGRP "*group*" **OFFSET=**{**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} "*clk_net*" [**TIMEGRP** "*reggroup*"]

Alternate method

Because the global and group OFFSET constraints are not associated with a single data net or component, these two types can also be entered on a TIMESPEC symbol in the design netlist with TS*identifier*.

TS*identifier*=[**TIMEGRP** *name*] **OFFSET =** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *"clk_net"* [**TIMEGRP** *"reggroup"*]

where

group is the name of a time group containing IOB components or pad BELs.

offset_time is the external offset.

units is an optional field to indicate the units for the offset time. The default is nano-seconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

IN or OUT specifies that the offset is computed with respect to an input IOB or an output IOB. For a bidirectional IOB, the IN or OUT lets you specify the flow of data (input output) on the IOB.

BEFORE or AFTER indicates whether the data is to arrive (input) or leave (output) the device before or after the clock input.

clk_net is the fully hierarchical netname of the clock net between the pad and its input buffer. All inputs / outputs are offset relative to *clk_net*.

reggroup is a previously defined time group of register bels. Only registers in the time group clocked by the specified IOB component is checked against the specified offset time. The optional time group qualifier, TIMEGRP "reggroup," can be added to any OFFSET constraint to indicate that the offset applies only to registers specified in the qualifying group. When used with the "Group method," the "register time" group lists the synchronous elements that qualify which register elements clocked by "clk_net" get analyzed.

Note: CPLD designs do not support the "Group Method" or the TIMEGRP options in the other methods described above.

Example

Schematic

N / A

UCF/NCF file

This statement specifies that the data will be present on input43 at least 20 ns before the triggering edge of the clock signal CLOCK.

```
NET input43 OFFSET=IN 20 BEFORE CLOCK;
```

For a detailed description of OFFSET, please see the "Using Timing Constraints" chapter in the *Development System Reference Guide*.

Constraints Editor

OFFSET IN BEFORE and OFFSET OUT AFTER constraints can be entered in the Advanced tab. Global offsets can be entered in the Global tab. Pad-to-Setup and Clock-to-Pad offsets can be entered in the Ports tab.

ONESHOT

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√ 1	√ 2

Applicable Elements

1. CAPTURE_SPARTAN2

2. CAPTURE_VIRTEX

Description

ONESHOT limits capture of registers for readback to a single capture of all register information. After a trigger (transition on CLK while CAP is asserted), all register information is captured and no additional captures can occur until the readback operation is completed. Without the ONESHOT attribute, data is captured after every trigger.

Syntax

ONESHOT

Example

Schematic

Attach to a CAPTURE_SPARTAN2 or CAPTURE_VIRTEX instance.

UCF/NCF file

N / A

Constraints Editor

N / A

OPT_EFFORT

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√		

Applicable Elements

Any macro or hierarchy level

Description

OPT_EFFORT defines an effort level to be used by the optimizer.

Syntax

OPT_EFFORT={NORMAL | HIGH}

Example

Schematic

Attach to a macro.

UCF/NCF file

This statement attaches a High effort of optimization to all of the logic contained within the module defined by instance $1I678/adder.

```
INST $1I678/adder OPT_EFFORT=HIGH;
```

Constraints Editor

N/A

OPTIMIZE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√		

Applicable Elements

Any macro or hierarchy level

Description

OPTIMIZE defines whether optimization is performed on the flagged hierarchical tree. The OPTIMIZE attribute has no effect on any symbol that contains no combinational logic, such as an input/output buffer.

Syntax

```
OPTIMIZE={AREA | SPEED | BALANCE | OFF}
```

Example

Schematic

Attach to a macro.

UCF/NCF file

This statement specifies that no optimization be performed on an instantiation of the macro CTR_MACRO.

```
INST /$1I678/CTR_MACRO OPTIMIZE=OFF;
```

Constraints Editor

N/A

OUTREG

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
			√					

Applicable Elements

Flip-flops, latches

Description

Because XC5200 IOBs do not have flip-flops or latches, you can apply the OUTREG attribute to meet fast setup requirements. If a flip-flop or latch is driving an IOB, you can specify OUTREG to enable PAR (Place and Route) to place the flip-flop / latch close to the IOB so that the two elements can be connected using fast routes. See also the "INREG" section.

Syntax

OUTREG

Example

Schematic

Attach to a latch or flip-flop instance.

UCF/NCF file

This statement directs PAR to place the flip-flop $I1 near the IOB that it is driving.

```
INST $I1 OUTREG;
```

Constraints Editor

N / A

PART

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

1. Global
2. Attached to CONFIG symbol in schematics

Description

PART defines the part type used for the design.

Syntax

PART=*part_type*

where *part_type* can be device-speed-package or device-package-speed. For example, 4028EX-PG299-3 or 4028EX-3-PG299

The package string must always begin with an alphabetic character — *never* with a number.

The speed string must always begin with an numeric character —*never* with an alphabetic character.

The text XC is an optional prefix to the whole part_type string.

In a constraints file, the PART specification must be preceded by the keyword CONFIG.

Example

Schematic

Place in a blank area of the schematic for global definition or attach to the CONFIG symbol.

UCF/NCF file

This statement specifies a 4005E device, a PQ160C package, with a speed of 5.

```
CONFIG PART=4005E-PQ160C-5;
```

Constraints Editor

N/A

PERIOD

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Nets that feed forward to drive flip-flop clock pins

Description

PERIOD provides a convenient way of defining a clock period for registers attached to a particular clock net.

PERIOD controls pad-to-setup and clock-to-setup paths but not clock-to-pad paths.

Refer to the "Using Timing Constraints" chapter in the *Development System Reference Guide* for more information on clock period specifications.

Special rules apply when using TNM and TNM_NET with the PERIOD constraint for Virtex and Spartan2 CLKDLLs and CLKDLLHFs. These rules are explained in the "PERIOD Specifications on CLKDLLs" section of the "Using Timing Constraints" chapter in the *Development System Reference Guide*.

Syntax

Simple Method

PERIOD=*period*[*units*] [{**HIGH** | **LOW**} [*high_or_low_time*[*hi_lo_units*]]]

where

period is the required clock period.

units is an optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ns, or us to indicate the intended units.

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

Alternate Method

TS*identifier*=**PERIOD** *TNM_reference period* [*units*] [{**HIGH** | **LOW**} [*high_or_low_time* [*hi_lo_units*]]]

where

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net (or a net in the clock path) using the TNM or TNM_NET attribute.

Note: When a TNM_NET attribute is traced into the CLKIN input of a CLKDLL or CLKDLLHF component, new PERIOD specifications may be created at the CLKDLL / CLKDLLHF outputs. If new PERIOD specifications are created, new TNM_NET groups to use in those specifications are also created. Each new TNM_NET group is named the same as the corresponding CLKDLL / CLKDLLHF output net (*outputnetname*). The new PERIOD specification becomes "TS_*outputnetname*=PERIOD *outputnetname*." The new TNM_NET groups are then traced forward from the CLKDLL / CLKDLLHF output net to tag all flip-flops, latches, and RAMs controlled by that clock signal. The new groups and specifications are shown in the timing analysis reports. Refer to the "PERIOD Specifications on CLKDLLs" section in the "Development System Reference Guide" for detailed information.

period is the required clock period.

units is an optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ms, us, or % to indicate the intended units.

HIGH or LOW indicates whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

Example

The following examples are for the "simple method."

Schematic

Attach to a net. Following is an example of the syntax format.

```
PERIOD=40 HIGH 25
```

UCF/NCF file

This statement assigns a clock period of 40 ns to the net named $SIG_24, with the first pulse being High and having a duration of 25 nanoseconds.

```
NET $SIG_24 PERIOD=40 HIGH 25;
```

Constraints Editor

Period timing constraints can be entered in the Global tab for each input pad signal used as a clock.

PROHIBIT

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Attached to CONFIG symbol

Description

PROHIBIT disallows the use of a site within PAR, FPGA Editor, and the CPLD fitter.

Location Types

Use the following location types to define the physical location of an element.

P12	IOB location (chip carrier)
A12	IOB location (pin grid or CSP package)
B, L, R, T	Indicates edge locations (bottom, left, top, right) — applies to edge decoders only
LB, RB, LT, RT, BR, TR, BL, TL	Indicates half edges (left bottom, right bottom, and so forth) — applies to edge decoders only
TL, TR, BL, BR	Indicates a corner for global buffer placement
AA	CLB location for XC3000
CLB_R4C3	CLB location for XC4000 or XC5200
CLB_R4C3 (or .S0 or .S1)	CLB location for Spartan2, Virtex
CLB_R6C8.LC0 (or 1, 2, 3)	Function generator or register location for XC5200

CLB_R6C8.S0 (or .S1)	Function generator or register location for Spartan2, Virtex
CLB_R6C8.LC0 (or 2)	F5_MUX location for XC5200
TBUF_R6C7.1 (or.2)	TBUF location for XC4000
TBUF_R6C7.0 (or.1,.2, or.3)	TBUF location for XC5200
TBUF_R6C7 (or .0 or .1)	TBUF location for Spartan2, Virtex
RAMB4_R3C1	Block RAM location for Spartan2, Virtex
GCLKBUF0 (or 1, 2, or 3)	Global clock buffer location for Spartan2, Virtex
GCLKPAD0 (or 1, 2, or 3)	Global clock pad location for Spartan2, Virtex
DLL0 (or 1, 2, or 3)	Delay Locked Loop element location for Spartan2, Virtex

The wildcard character (*) can be used to replace a single location with a range as shown in the following examples.

C*	Any CLB in row C of an XC3000 device
*D	Any CLB in column D of an XC3000 device
CLB_R*C5	Any CLB in column 5 of an XC4000 or XC5200 device
CLB_R*C5	Any CLB in either slice in column 5 of a Spartan2, Virtex device

Note: The wildcard character is *not* supported for Virtex and Spartan2 global buffer or DLL locations.

The following are *not* supported.

- Dot extensions on ranges. For example, LOC=CLB_R0C0:CLB_R5C5.G. However, for the XC5200, range locations will be expanded to include extensions, CLB_R0C0.*:CLB_R5C5.*, for example, when the mapper passes a range constraint to the PCF file.
- B, L, R, T used to indicate IO edge locations (bottom, left, top, right)
- LB, RB, LT, RT, BR, TR, BL, TL used to indicate IO half edges (left bottom, right bottom, etc.)
- .F or .G extension for function generator, RAM, ROM, or RAMS location for XC4000
- .FFX or .FFY extension for flip-flop location for XC4000
- Wildcard character for Virtex and Spartan2 global buffer, global pad, or DLL locations.

Syntax

Single location

PROHIBIT=*location*

Multiple single locations

PROHIBIT=*location1, location2, . . . , locationn ;*

Range of locations

PROHIBIT=*location***:***location*

In a constraints file, the PROHIBIT specification must be preceded by the keyword CONFIG.

Note: CPLDs do not support the "Range of locations" form of PROHIBIT.

Example

Schematic

Place on the schematic as an unattached attribute or attach to a CONFIG symbol.

UCF/NCF file

This statement prohibits use of the site P45.

```
CONFIG PROHIBIT=P45;
```

This statement prohibits use of the CLB located in Row 6, Column 8.

```
CONFIG PROHIBIT=CLB_R6C8 ;
```

This statement prohibits use of the site TBUF_R5C2.2.

```
CONFIG PROHIBIT=TBUF_R5C2.2 ;
```

Constraints Editor

PROHIBIT constraints can be entered using a dialog box provided in the Ports tab.

PULLDOWN

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√	√	√

Applicable Elements

Input, output, and bidirectional pads and pad nets

Description

PULLDOWN guarantees a logic Low level to allow tri-stated nets to avoid floating when not being driven.

Syntax

PULLDOWN

Example

Schematic

Attach to a pad or pad net.

UCF/NCF file

These statements configure the IO to use a PULLDOWN.

```
NET "pad_net_name" PULLDOWN;
INST "pad_instance_name" PULLDOWN;
```

Constraints Editor

PULLDOWN can be entered for an output or bidirectional pad signal through the Ports tab (I/O Configuration Options).

PULLUP

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Input, output, and bidirectional pads and pad nets

Description

PULLUP guarantees a logic High level to allow tri-stated nets to avoid floating when not being driven.

Syntax

PULLUP

Example

Schematic

Attached to a pad or pad net.

UCF/NCF file

These statements configure the IO to use a PULLUP.

```
NET "pad_net_name" PULLUP;
INST "pad_instance_name" PULLUP;
```

Constraints Editor

PULLUP can be entered for an output or bidirectional pad signal through the Ports tab (I/O Configuration Options).

PWR_MODE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

Applicable Elements

1. Nets
2. Any instance

Description

PWR_MODE defines the mode, Low power or High performance (standard power), of the macrocell that implements the tagged element.

Note: If the tagged function is collapsed forward into its fanouts, the attribute is not applied.

Syntax

```
PWR_MODE={LOW | STD}
```

Example

Schematic

Attach to a net or an instance.

UCF/NCF file

This statement specifies that the macrocell that implements the net $SIG_0 will be in Low power mode.

```
NET $1187/$SIG_0 PWR_MODE=LOW;
```

Constraints Editor

N/A

REG

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

Applicable Elements

Registers

Description

REG specifies how a register is to be implemented in the CPLD macrocell.

Syntax

```
REG = {CE | TFF}
```

where

CE, when applied to an FDCE or FDPE primitive, forces the CE pin input to be implemented using the clock enable product term of the XC9500XL or XC9500XV macrocell. (XC9500 macrocells do not support clock enable p-terms. For the XC9500, REG=CE attributes are ignored.)

TFF indicates that the register is to be implemented as a T-type flip-flop in the CPLD macrocell. If applied to a D-flip-flop primitive, the D-input expression is transformed to T-input form and implemented with a T-flip-flop. Automatic transformation between D and T flip-flops is normally performed by the CPLD fitter.

Example

Schematic

Attach to a flip-flop instance or macro containing flip-flops.

UCF/NCF file

This statement specifies that the CE pin input be implemented using the clock enable product term of the XC9500XL or XC9500XV macrocell.

```
INST Q1 REG=CE;
```

Constraints Editor

N/A

RLOC

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√ 1, 2, 3, 5, 7, 8, 9, 10, 11	√ 1, 2, 3, 5, 7, 8, 9, 10, 11	√ 1, 2, 4, 6, 10		√ 1, 2, 3, 5, 7, 8, 9, 10	√ 1, 2, 3, 5, 7, 8, 9, 10	√ 1, 2, 8, 9, 10, 12	√ 1, 2, 8, 9, 10, 12

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. ROM
8. RAM
9. RAMS, RAMD
10. BUFT (Can only be used if the associated RPM has an RLOC_ORIGIN that causes the RLOC values in the RPM to be changed to LOC values.)

11. WAND primitives that do not have a DECODE attribute attached
12. LUTs, F5MUX, F6MUX, MUXCY, XORCY, MULT_AND, SRL16, SRL16E

Description

Relative location (RLOC) constraints group logic elements into discrete sets and allow you to define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design. See the “Physical Constraints” section for detailed information about this type of constraint.

For XC5200, the RLOC attribute must include the extension that defines in which of the four slices of a CLB the element will be placed (.LC0, .LC1, .LC2, .LC3). This defines the relationship of the elements in the set and also specifies in which of the four slices the element will eventually be placed.

For Virtex and Spartan2, the RLOC attribute must include the extension that defines in which of the two slices of a CLB the element will be placed (.S0, .S1).

Syntax

XC4000E, XC4000X, Spartan, SpartanXL

RLOC=R*m***C***n*[**.***extension*]

XC5200, Spartan2, Virtex

RLOC=R*m***C***n***.***extension*

where

m and *n* are integers (positive, negative, or zero) representing relative row numbers and column numbers, respectively.

extension uses the LOC extension syntax as appropriate; it can take all the values that are available with the current absolute LOC syntax.

For the XC4000, Spartan, and SpartanXL, the available extensions are FFX, FFY, F, G, H, 1, and 2. The 1 and 2 values are available for BUFT primitives, and the rest are available for primitives associated with CLBs. See the “LOC” section for more details.

For the XC5200, *extension* is required to define in which of the four slices of a CLB the element will be placed (.LC0, .LC1, .LC2, .LC3).

For Virtex and Spartan2, *extension* is required to define the spatial relationships (.S0 is the right-most slice; .S1 is the left-most slice) of the objects in the RPM.

The RLOC value cannot specify a range or a list of several locations; it must specify a single location. See the “Guidelines for Specifying Relative Locations” section for more information.

Example

Schematic

Attach to an instance.

UCF/NCF file

This statement specifies that an instantiation of FF1 be placed in the CLB at row 4, column 4.

```
INST /4K/design/FF1 RLOC=R4C4;
```

This statement specifies that an instantiation of elemA be placed in the X flip-flop in the CLB at row 0, column 1.

```
INST /$1I87/elemA RLOC=r0cl.FFX;
```

Constraints Editor

N / A

RLOC_ORIGIN

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√	√	√

Applicable Elements

Instances or macros that are members of sets

Description

An RLOC_ORIGIN constraint fixes the members of a set at exact die locations. This constraint must specify a single location, not a range or a list of several locations. For more information about this constraint, refer to the “Fixing Members of a Set at Exact Die Locations” section.

The RLOC_ORIGIN constraint is required for a set that includes BUFT symbols. The RLOC_ORIGIN constraint cannot be attached to a BUFT instance.

Syntax

RLOC_ORIGIN=R*m***C***n*

where *m* and *n* are positive integers (including zero) representing relative row and column numbers, respectively.

Example

Schematic

Attach to an instance that is a member of a set.

UCF/NCF file

This statement specifies that an instantiation of FF1, which is a member of a set, be placed in the CLB at R4C4 relative to FF1. For example, if RLOC=R0C2 for FF1, then the instantiation of FF1 is placed in the CLB that occupies row 4 (R0 + R4) , column 6 (C2 + C4).

```
INST /archive/designs/FF1 RLOC_ORIGIN=R4C4;
```

Constraints Editor

N / A

RLOC_RANGE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√	√	√

Applicable Elements

Instances or macros that are members of sets

Description

The RLOC_RANGE constraint is similar to the RLOC_ORIGIN constraint except that it limits the members of a set to a certain range on the die. The range or list of locations is meant to apply to all applicable elements with RLOCs, not just to the origin of the set.

Syntax

RLOC_RANGE=R*m1***C***n1***:R***m2***C***n2*

where the relative row numbers (*m1* and *m2*) and column numbers (*n1* and *n2*) can be positive integers (including zero) or the wildcard (*) character. This syntax allows three kinds of range specifications, which are defined in the “Fixing Members of a Set at Exact Die Locations” section.

Example

Schematic

Attach to an instance that is a member of a set.

UCF/NCF file

This statement specifies that an instantiation of the macro MACRO4 be placed within a region that is enclosed by the rows R4-R10 and the columns C4-C10.

```
INST /archive/designs/MACRO4 RLOC_RANGE=R4C4:R10C10;
```

Constraints Editor

N/A

S(ave) - Net Flag Attribute

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets

Description

Attaching the SAVE net flag attribute to nets affects the mapping, placement, and routing of the design by preventing the removal of unconnected signals.

Syntax

S

The S (save) net flag attribute prevents the removal of unconnected signals. If you do not have the S attribute on a net, any signal not connected to logic and/or an I/O primitive is removed.

Example

Schematic

Attach to a net.

UCF/NCF file

This statement specifies that the net named $SIG_9 will not be removed.

```
NET $SIG_9 S;
```

Constraints Editor

N/A

SLOW

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√				

Applicable Elements

Output primitives, output pads, bidirectional pads

Note: You can also attach the SLOW constraint to the net connected to the pad component in a UCF file. NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **SLOW**

Description

SLOW stipulates that the slew rate limited control should be enabled.

Syntax

SLOW

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement establishes a slow slew rate for an instantiation of the element y2.

```
INST $1I87/y2 SLOW;
```

This statement establishes a slow slew rate for the pad to which net1 is connected.

```
NET net1 SLOW;
```

Constraints Editor

SLOW slew rate can be selected for any output pad signal in the Ports tab (I/O Configuration Options).

STARTUP_WAIT

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
							√	√

Applicable Elements

Any CLKDLL, CLKDLLHF, or BUGDGLL instance

Description

STARTUP_WAIT controls whether the DONE signal (device configuration) can go HIGH (indicating that the device is fully configured).

Syntax

STARTUP_WAIT={**TRUE** | **FALSE**}

where

TRUE specifies that the DONE signal cannot go High until the instance assigned this property locks.

FALSE, the default, specifies that the locking of the instance has no impact on the DONE signal.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement specifies that the DONE signal cannot go High until the foo/bar instance locks.

```
INST foo/bar STARTUP_WAIT=TRUE;
```

Constraints Editor

N/A

TEMPERATURE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√*	√*	√*	√*		√*	√*	√*	√*

*Availability depends on the release of characterization data

Applicable Elements

Global

Description

The TEMPERATURE constraint allows the specification of the operating junction temperature. This provides a means of prorating device delay characteristics based on the specified temperature. Prorating is a scaling operation on existing speed file delays and is applied globally to all delays. (Prorating is applicable only to commercial operating temperature ranges. It is not intended for use by industrial and military customers.)

Note: Each architecture has its own specific range of valid operating temperatures. If the entered temperature does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead. Also note that the error message for this condition does not appear until PCF processing.

Syntax

TEMPERATURE=*value*[**C |F| K**]

where

value is real number specifying the temperature.

C, K, and F are the temperature units. F is degrees Fahrenheit, K is degrees Kelvin, and C is degrees Celsius, the default.

Example

Schematic

Place on the schematic as an unattached attribute.

UCF/NCF file

This statement specifies that the analysis for everything relating to speed file delays assumes a junction temperature of 25 degrees Celsius.

```
TEMPERATURE=25C;
```

Constraints Editor

Temperature prorating constraints can be specified from the Advanced tab.

TIG

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets, pins

Description

TIG (Timing IGnore) causes paths that fan forward from the point of application (of TIG) to be treated as if they do not exist (for the purposes of the timing model) during implementation.

A TIG may be applied relative to a specific timing specification.

Syntax

TIG

or

TIG=TS*identifier1*, . . . , **TS***identifiern*

where *identifier* refers to a timing specification that should be ignored.

Example

Schematic

Attach to a net or pin.

UCF/NCF file

This statement specifies that the timing specifications TS_fast and TS_even_faster will be ignored on all paths fanning forward from the net $Sig_5.

```
NET $1I567/$Sig_5 TIG=TS_fast, TS_even_faster;
```

For more on TIG, see the "Using Timing Constraints" chapter in the *Development System Reference Guide.*

Constraints Editor

Nets to be ignored for timing purposes can be specified in the Advanced tab.

Time Group Attributes

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

1. Global in constraints file (preceded by the keyword TIMEGRP)
2. Time group primitive

Description

Time group properties (attributes) are a set of grouping mechanisms that use existing TNMs (Timing Names) to create new groups or to define new groups based on the output net that the group sources. The timing group primitive (TIMEGRP) exists for the purpose of hosting these properties. In a constraints file, the specification of these properties must be preceded with the keyword TIMEGRP.

Note: When entering time group properties into a TIMEGRP symbol, some property names may conflict with the predefined property names of the TIMEGRP primitive.

The standard procedure for adding a property to a symbol is to use the following format.

PROPERTY=*property_name* VALUE=*value*

However, some property names are reserved, and should not be used because they cause a conflict. Hence, for *property_name* you must not use any of the system reserved names LIBVER, INST, COMP, MODEL, or any other names reserved by your schematic capture program. Please consult your schematic capture documentation to become familiar with reserved property names.

Note: For more on the TIMEGRP symbol, see the “TIMEGRP” section in the "Design Elements" chapter.

Syntax

new_group_name=[**RISING** | **FALLING**] *group_name1* [**EXCEPT** *group_name2*... *group_namen*]

or

new_group_name=[**TRANSHI** | **TRANSLO**] *group_name1* [**EXCEPT** *group_name2*... *group_namen*]

where

group_names can be

- the name assigned to a previously defined group.
- all of the members of a predefined group using the keywords FFS, RAMS, PADS or LATCHES as follows:
  - FFS refers to all CLB and IOB flip-flops. (Flip-flops built from function generators are not included. Shift register LUTs in Virtex and Spartan2 are not included.)
  - RAMS refers to all RAMs for architectures with RAMS. For Virtex and Spartan2, LUT RAMS and Block RAMS are included.
  - PADS refers to all I/O pads.
  - LATCHES refers to all CLB or IOB latches. (Latches built from function generators are not included.)
- a subset of elements in a group predefined by name matching using the following syntax.

  predefined_group name qualifier1... *name_qualifiern*

RISING or FALLING creates timing subgroups from the rising or falling edge sensitive flip-flops in a timing group. If the timing group contains non-flip-flop elements, these elements are also included in the subgroup; they are not filtered based on edge

sensitivity. (Refer to the preceding *group_names* description for information on creating flip-flop only timing groups.)

TRANSHI or TRANSLO is the form of the constraint applied to latches.

EXCEPT excludes the object group.

Example 1

Schematic

The following attribute would be attached to a TIMEGRP primitive to combine the elements in two groups to form a new group.

```
big_group=little_group other_group
```

UCF/NCF file

The same constraint could appear in a User Constraints File (UCF) as follows.

```
TIMEGRP big_group=little_group other_group;
```

Constraints Editor

New timing groups can be created in the Advanced tab.

Example 2

Schematic

The following constraints would be attached to a TIMEGRP primitive to define new groups by exclusion.

```
input_pads=pads except output_pads
```

UCF/NCF file

The same constraint could appear in a UCF as follows.

```
TIMEGRP input_pads=pads EXCEPT output_pads;
```

For more on Time Group Attributes, see the "Using Timing Constraints" chapter in the *Development System Reference Guide.*

Constraints Editor

New timing groups can be created in the Advanced tab.

TNM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

Nets, instances, macros

Note: You can attach the TNM constraint to the net connected to the pad component in a UCF file. NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper. Use the following syntax.

NET *net_name* **TNM=***property_value*

Description

TNM (Timing Name) tags specific flip-flops, RAMs, pads, and latches as members of a group to simplify the application of timing specifications to the group.

TNMs applied to pad nets do not propagate forward through the IBUF / OBUF. The TNM is applied to the external pad. This case includes the net attached to the D input of an IFD. See the "TNM_NET" section if you want the TNM to trace forward from an input pad net.

TNMs applied to the input pin of an IBUF / OBUF will cause the TNM to be attached to the pad.

TNMs applied to the output pin of an IBUF / OBUF will propagate the TNM to the next appropriate element.

TNMs applied to an IBUF or OBUF element stay attached to that element.

TNMs applied to a clock-pad-net will not propagate forward through the clock buffer.

When TNM is applied to a macro, all the elements in the macro will have that timing name.

Special rules apply when using TNM with the PERIOD constraint for Virtex and Spartan2 CLKDLLs and CLKDLLHFs.

See the "Using Timing Constraints" chapter in the *Development System Reference Guide* for detailed information about this attribute.

Syntax

TNM=[*predefined_group*:] *identifier;*

where

predefined_group can be

- the name assigned to a previously defined group.
- all of the members of a predefined group using the keywords FFS, RAMS, PADS or LATCHES as follows:
  - FFS refers to all CLB and IOB flip-flops. (Flip-flops built from function generators are not included. Shift register LUTs in Virtex and Spartan2 are not included.)
  - RAMS refers to all RAMs for architectures with RAMS. For Virtex and Spartan2, LUT RAMS and Block RAMS are included.
  - PADS refers to all I/O pads.
  - LATCHES refers to all CLB or IOB latches. (Latches built from function generators are not included.)
- a subset of elements in a group predefined by name matching using the following syntax.

  predefined_group name_qualifier1 . . . name_qualifiern

identifier can be any combination of letters, numbers, or underscores. Do not use reserved words, such as FFS, LATCHES, RAMS, or PADS for TNM identifiers.

Example

Schematic

Attach to a net or a macro.

UCF/NCF file

This statement identifies the element register_ce as a member of the timing group the_register.

```
NET $1I87/register_ce TNM=the_register;
```

Constraints Editor

Timing group names can be assigned to pad and flip-flop elements in the Advanced tab.

TNM_NET

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets

Description

TNM_NET (Timing Name - Net) tags specific flip-flops, RAMs, pads, and latches as members of a group to simplify the application of timing specifications to the group. NGDBuild never transfers a TNM_NET constraint from the attached net to a pad, as it does with the TNM constraint.

TNM_NETs applied to pad nets propagate forward through the IBUF/ OBUF.

TNM_NETs applied to a clock-pad-net propagate forward through the clock buffer.

When TNM_NET is applied to a macro, all the elements in the macro will have that timing name.

Special rules apply when using TNM_NET with the PERIOD constraint for Virtex and Spartan2 CLKDLLs and CLKDLLHFs.

See the "Using Timing Constraints" chapter in the *Development System Reference Guide* for detailed information about this attribute.

Syntax

TNM_NET=[*predefined_group*:]*identifier*

where

predefined_group can be

- the name assigned to a previously defined group.
- all of the members of a predefined group using the keywords FFS, RAMS, PADS or LATCHES. FFS refers to all flip-flops. RAMS refers to all RAMs. PADS refers to all I/O pads. LATCHES refers to all latches.

- a subset of elements in a group predefined by name matching using the following syntax.

  predefined_group name_qualifier1 . . . name_qualifiern

identifier can be any combination of letters, numbers, or underscores. Do not use reserved words, such as FFS, LATCHES, RAMS, or PADS for TNM identifiers.

Example

Schematic

Attach to a net.

UCF/NCF file

This statement identifies all flip-flops fanning out from the PADCLK net as a member of the timing group FFGRP.

```
NET PADCLK TNM_NET=FFS(*):FFGRP;
```

Constraints Editor

Timing group names can be assigned to net elements in the Advanced tab.

TPSYNC

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets, instances, pins

Description

TPSYNC flags a particular point or a set of points with an identifier for reference in subsequent timing specifications. You can use the same identifier on several points, in which case timing analysis treats the points as a group. See the “Time Group Attributes” section.

Defining synchronous points

When the timing of a design must be designed from or to a point that is not a flip-flop, latch, RAM, or I/O pad, the following rules apply if a TPSYNC timing point is attached to a net, macro pin, output or input pin of a primitive, or an instance.

- A net — the source of the net is identified as a potential source or destination for timing specifications.
- A macro pin — all of the sources inside the macro that drive the pin to which the attribute is attached are identified as potential sources or destinations for timing specifications. If the macro pin is an input pin (that is, if there are no sources for the pin in the macro), then all of the load pins in the macro are flagged as synchronous points.
- The output pin of a primitive — the primitive’s output is flagged as a potential source or destination for timing specifications.

- The input pin of a primitive — the primitive's input is flagged as a potential source or destination for timing specifications.
- An instance — the output of that element is identified as a potential source or destination for timing specifications.

Syntax

TPSYNC=*identifier*

where *identifier* is a name that is used in timing specifications in the same way that groups are used.

All flagged points are used as a source or destination or both for the specification where the TPSYNC identifier is used.

Note: The name for the identifier must be different from any identifier used for a TNM attribute.

Example

Schematic

Attach to a net, instance, or pin.

UCF/NCF file

This statement identifies latch as a potential source or destination for timing specifications for the net logic_latch.

```
NET $1I87/logic_latch TPSYNC=latch;
```

Constraints Editor

N/A

TPTHRU

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√		√	√	√	√

Applicable Elements

Nets, pins, instances

Description

TPTHRU flags a particular point or a set of points with an identifier for reference in subsequent timing specifications. You can use the same identifier on several points, in which case timing analysis treats the points as a group. See the "Time Group Attributes" section.

Defining through points

The TPTHRU attribute is used when it is necessary to define intermediate points on a path to which a specification applies. See the "TSidentifier" section.

Syntax

TPTHRU=*identifier*

where *identifier* is a name used in timing specifications for further qualifying timing paths within a design.

Note: The name for the identifier must be different from any identifier used for a TNM attribute.

Example

Schematic

Attach to a net, instance, or pin.

UCF/NCF file

This statement identifies the net on_the_way as an intermediate point on a path to which the timing specification named “here” applies.

```
NET $1I87/on_the_way TPTHRU=here;
```

Constraints Editor

Timing THRU points can be created in the Advanced tab.

TSidentifier

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√	√	√	√	√	√	√	√	√

Applicable Elements

1. Global in constraints file
2. TIMESPEC primitive

Description

TSidentifier properties beginning with the letters “TS” are placed on the TIMESPEC symbol. In a constraints file, the specification of these properties can be preceded with the optional keyword TIMESPEC. The value of the *TSidentifier* attribute corresponds to a specific timing specification that can then be applied to paths in the design.

Syntax

Note: All the following syntax definitions use a space as a separator. The use of a colon (:) as a separator is optional.

Defining a maximum allowable delay

TS*identifier*=[**MAXDELAY**] **FROM** *source_group* **TO** *dest_group allowable_delay* [*units*]

or

TS*identifier***=FROM** *source_group* **TO** *dest_group allowable_delay* [*units*]

Defining intermediate points

Note: This form is not supported for CPLDs.

TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point1* . . . *thru_pointn*] **TO** *dest_group allowable_delay* [*units*]

where

identifier is an ASCII string made up of the characters A-Z, a-z, 0-9, and _.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. The default units are nanoseconds (ns), but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

Defining a linked specification

This allows you to link the timing number used in one specification to another specification.

TS*identifier***=FROM** *source_group* **TO** *dest_group another_TSid*[**/** | *****] *number*

where

identifier is an ASCII string made up of the characters A-Z, a-z, 0-9, and _.

source_group and *dest_group* are user-defined or predefined groups.

another_Tsid is the name of another timespec.

number is a floating point number.

Defining a clock period

This allows more complex derivative relationships to be defined as well as a simple clock period.

TS*identifier***=PERIOD** *TNM_reference period*[*units*] [{**HIGH** | **LOW**} [*high_or_low_time* [*hi_lo_units*]]]

where

identifier is a reference identifier with a unique name.

TNM_reference is the identifier name attached to a clock net (or a net in the clock path) using a TNM attribute.

period is the required clock period.

units is an optional field to indicate the units for the allowable delay. The default units are nanoseconds (ns), but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

Specifying derived clocks

TS*identifier***=PERIOD** *TNM_reference another_PERIOD_identifier* [**/** | *****] *number* [{**HIGH** | **LOW**} [*high_or_low_time* [*hi_lo_units*]]]

where

TNM_reference is the identifier name attached to a clock net (or a net in the clock path) using a TNM attribute.

another_PERIOD_identifier is the name of the identifier used on another period specification.

number is a floating point number.

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

Ignoring paths

Note: This form is not supported for CPLDs.

There are situations in which a path that exercises a certain net should be ignored because all paths through the net, instance, or instance pin are not important from a timing specification point of view.

TS*identifier***=FROM** *source_group* **TO** *dest_group* **TIG**

or

TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point1*. . . *thru_pointn*]**TO** *dest_group* **TIG**

where

identifier is an ASCII string made up of the characters A-Z, a-z 0-9, and _.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

Example

Schematic

Attach to a TIMESPEC primitive.

UCF/NCF file

This statement says that the timing specification TS_35 calls for a maximum allowable delay of 50 ns between the groups “here” and “there”.

```
TIMESPEC TS_35=FROM here TO there 50;
```

This statement says that the timing specification TS_70 calls for a 25 ns clock period for clock_a, with the first pulse being High for a duration of 15 ns.

```
TIMESPEC TS_70=PERIOD “clock_a” 25 high 15;
```

For more information, see the “Timing Constraints” section.

Note: In either example above, a colon can be used instead of a space as the separator. (Additional spaces entered before or after the colon are ignored.) The statements then become as follows.

```
TIMESPEC TS_35=FROM:here:TO:there:50;
TIMESPEC TS_70=PERIOD:”clock_a”:25:high:15;
```

Constraints Editor

Clock period timing constraints can be entered in the Global tab. Input setup time and clock-to-output delay can be entered for specific pads in the Ports tab, or for all pads related to a given clock in the Global tab. Combinational pad-to-pad delays can be entered in the Advanced tab, or for all pad-to-pad paths in the Global tab.

U_SET

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√ 1, 2, 3, 5, 7, 8, 9, 10, 11, 12	√ 1, 2, 3, 5, 7, 8, 9, 10, 11, 12	√ 1, 2, 4, 6, 7, 8, 12		√ 1, 2, 3, 5, 7, 8, 9, 10, 11, 12	√ 1, 2, 3, 5, 7, 8, 9, 10, 11, 12	√ 1, 2, 7, 8, 10, 11, 12, 13	√ 1, 2, 7, 8, 10, 11, 12, 13

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. Macro instance
8. EQN
9. ROM
10. RAM
11. RAMS, RAMD
12. BUFT (Can only be used for Virtex and Spartan2 if the associated RPM has an RLOC_ORIGIN that causes the RLOC values in the RPM to be changed to LOC values.)
13. LUTs, F5MUX, F6MUX, MUXCY, XORCY, MULT_AND, SRL16, SRL16E

Description

The U_SET constraint groups design elements with attached RLOC constraints that are distributed throughout the design hierarchy into a single set. The elements that are members of a U_SET can cross the design hierarchy; that is, you can arbitrarily select objects without regard to the design hierarchy and tag them as members of a U_SET. For detailed information about this attribute, refer to the “RLOC Sets” section.

Syntax

U_SET=*name*

where *name* is the identifier of the set. This name is absolute. It is not prefixed by a hierarchical qualifier.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement specifies that the design element ELEM_1 be in a set called JET_SET.

```
INST $1I3245/ELEM_1 U_SET=JET_SET;
```

Constraints Editor

N/A

USE_RLOC

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
	√	√	√		√	√	√	√

Applicable Elements

Instances or macros that are members of sets

Description

USE_RLOC turns the RLOC constraint on or off for a specific element or section of a set. For detailed information about this constraint, refer to the “Toggling the Status of RLOC Constraints” section.

Syntax

USE_RLOC={TRUE | FALSE}

where TRUE turns on the RLOC attribute for a specific element, and FALSE turns it off. The default is TRUE.

Example

Schematic

Attach to a member of a set.

UCF/NCF file

```
INST $1I87/big_macro USE_RLOC=FALSE;
```

Constraints Editor

N / A

VOLTAGE

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√*	√*	√*	√*		√*	√*	√*	√*

*Availability depends on the release of characterization data

Applicable Elements

Global

Description

The VOLTAGE constraint allows the specification of the operating voltage. This provides a means of prorating delay characteristics based on the specified voltage. Prorating is a scaling operation on existing speed file delays and is applied globally to all delays. (Prorating is applicable only to commercial operating voltage ranges. It is not intended for use by industrial and military customers.)

Note: Each architecture has its own specific range of supported voltages. If the entered voltage does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead. Also note that the error message for this condition appears during PCF processing.

Syntax

VOLTAGE=*value*[**V**]

where

value is a real number specifying the voltage.

V indicates volts, the default voltage unit.

Example

Schematic

Place on the schematic as an unattached attribute.

UCF/NCF file

This statement specifies that the analysis for everything relating to speed file delays assumes an operating power of 5 volts.

```
VOLTAGE=5;
```

Constraints Editor

Voltage prorating constraints can be specified in the Advanced tab.

WIREAND

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
				√*				

*Not supported for XC9500XL and XC9500XV designs

Applicable Elements

Any net

Description

WIREAND forces a tagged node to be implemented as a wired AND function in the interconnect (UIM and Fastconnect).

Syntax

WIREAND

Example

Schematic

Attach to a net.

UCF/NCF file

This statement specifies that the net named SIG_11 be implemented as a wired AND when optimized.

```
NET $I16789/SIG_11 WIREAND;
```

Constraints Editor

N/A

XBLKNM

XC3000	XC4000E	XC4000X	XC5200	XC9000	Spartan	SpartanXL	Spartan2	Virtex
√ 1,2, 3, 7, 8	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 6, 7, 11		√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 1, 2, 3, 4, 7, 8, 9, 10, 11	√ 1, 2, 3, 4, 7, 8, 9, 10, 11	√ 1, 2, 3, 4, 7, 8, 9, 10, 11

Applicable Elements

1. IOB, CLB, and CLBMAP
2. Flip-flop and latch primitives
3. Any I/O element or pad
4. FMAP
5. HMAP
6. F5MAP

7. BUFT
8. ROM primitive
9. RAM primitives
10. RAMS and RAMD primitives
11. Carry logic primitives

Description

XBLKNM assigns LCA block names to qualifying primitives and logic elements. If the same XBLKNM attribute is assigned to more than one instance, the software attempts to map them into the same LCA block. Conversely, two symbols with different XBLKNM names are not mapped into the same block. Placing similar XBLKNMs on instances that do not fit within one LCA block creates an error.

Specifying identical XBLKNM attributes on FMAP and/or HMAP symbols tells the software to group the associated function generators into a single CLB. Using XBLKNM, you can partition a complete CLB without constraining the CLB to a physical location on the device.

XBLKNM attributes, like LOC constraints, are specified from the schematic. Hierarchical paths are not prefixed to XBLKNM attributes, so XBLKNM attributes for different CLBs must be unique throughout the entire design.

The BLKNM attribute allows any elements except those with a different BLKNM to be mapped into the same physical component. XBLKNM, however, allows only elements with the same XBLKNM to be mapped into the same physical component. Elements without an XBLKNM cannot be not mapped into the same physical component as those with an XBLKNM.

For XC5200, a given XBLKNM string can only be used to group a logic cell (LC), which contains one register, one LUT (FMAP), and one F5_MUX element. An error will occur if two or more registers, two or more FMAPs, or two or more F5_MUX elements have the same XBLKNM attribute.

Syntax

XBLKNM=*block_name*

where *block_name* is a valid LCA block name for that type of symbol. For a list of prohibited block names, see the “Naming Conventions” section of each user interface manual.

Example

Schematic

Attach to a valid instance.

UCF/NCF file

This statement assigns an instantiation of an element named flip_flop2 to a block named U1358.

```
INST $1I87/flip_flop2 XBLKNM=U1358;
```

Constraints Editor

N/A

Placement Constraints

This section describes the legal placement constraints for each type of logic element, such as flip-flops, ROMs and RAMs, FMAPs, F5MAPs, and HMAPs, CLBMAPs, BUFTs, CLBs, IOBs, I/Os, edge decoders, and global buffers in FPGA designs. Individual logic gates, such as AND or OR gates, are mapped into CLB function generators before the constraints are read and therefore cannot be constrained. However, if gates are represented by an FMAP, F5MAP, HMAP, or CLBMAP symbol, you can put a placement constraint on that symbol.

You can use the following constraints (described earlier in the "Attributes/Logical Constraints" section) to control mapping and placement of symbols in a netlist.

- BLKNM
- HBLKNM
- XBLKNM
- LOC
- PROHIBIT
- RLOC
- RLOC_ORIGIN
- RLOC_RANGE

Most constraints can be specified either in the schematic or in the UCF file.

In a constraints file, each placement constraint acts upon one or more symbols. Every symbol in a design carries a unique name, which is defined in the input file. Use this name in a constraint statement to identify the symbol.

Note: The UCF and NCF files are case sensitive. Identifier names (names of objects in the design, such as net names) must exactly match the case of the name as it exists in the source design netlist. However, any Xilinx constraint keyword (for example, LOC, PROHIBIT, RLOC, BLKNM) can be entered in either all upper-case or all lower-case letters; mixed case is not allowed.

The following sections describe various types of placement constraints, explains the method of determining the symbol name for each, and provides examples.

BUFT Constraint Examples

You can constrain internal 3-state buffers (BUFTs) to an individual BUFT location, a list of BUFT locations, or a rectangular block of BUFT locations. BUFT constraints all refer to locations with a prefix of TBUF, which is the name of the physical element on the device.

BUFT constraints can be assigned from the schematic or through the UCF file. From the schematic, LOC constraints are attached to the target BUFT. The constraints are then passed into the EDIF netlist file and after mapping are read by PAR. Alternatively, in a constraints file a BUFT is identified by a unique instance name.

In the XC3000, BUFT locations are not straightforward. View the device in the FPGA Editor to determine the exact BUFT names.

In XC4000, Spartan, or SpartanXL, BUFT locations are identified by the adjacent CLB. Thus, TBUF_R1C1.1 is just above CLB_R1C1, and TBUF_R1C1.2 is just below it. For XC4000, Spartan, or SpartanXL, use the following syntax to denote fixed locations.

TBUF_R*row***C***col*{**.1 | .2**}

where *row* is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

- 1 indicates that the instance should be placed above the CLB.
- 2 indicates that the instance should be placed below the CLB.

In the XC5200, BUFT locations are identified by the adjacent slice. From bottom to top, they are number 0, 1, 2, and 3. Thus, TBUF_R1C1.0 is located toward the bottom of the row. TBUF_R1C1.3 is located toward the top of the row. For an XC5200, Use the following syntax to denote fixed locations.

TBUF_R*row***C***col*{**.0 | .1 | .2 | .3**}

where *row* is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

- 0 indicates that the instance should be placed in the bottom buffer.
- 1 indicates that the instance should be placed in the buffer that is second from bottom.
- 2 indicates that the instance should be placed in the buffer that is second from top.
- 3 indicates that the instance should be placed in the top buffer.

For Virtex or Spartan2, use the following syntax to denote fixed locations.

TBUF_R*row***C***col*{**.0 | .1**}

where *row* is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

- 0 indicates one TBUF at the specific row / column.
- 1 indicates the second TBUF at the specific row / column.

For the XC4000, XC5200, Spartan, Spartan2, SpartanXL, or Virtex, use the following syntax to denote a range of locations from the lowest to the highest.

TBUF_R*row***C***col* **TBUF_R***row***C***col*

The following examples illustrate the format of BUFT LOC constraints. Specify LOC= and the BUFT location.

The following statements place the BUFT in the designated location.

`LOC=TBUF.AA.1`	(XC3000)
`LOC=TBUF_R1C1.1 (or .2)`	(XC4000, Spartan, SpartanXL)
`LOC=TBUF_R1C1.3 (or .0, .1, .2)`	(XC5200)
`LOC=TBUF_R1C1.0 (or .1)`	(Spartan2, Virtex)

The next statements place BUFTs at any location in the first column of BUFTs. The asterisk (*) is a wildcard character.

`LOC=TBUF.*A`	(XC3000)
`LOC=TBUF_R*C0`	(XC4000, XC5200, Spartan, SpartanXL, Spartan2, Virtex)

The following statements place BUFTs within the rectangular block defined by the first specified BUFT in the upper left corner and the second specified BUFT in the lower right corner.

`LOC=TBUF.AA:TBUF.BH`	(XC3000)
`LOC=TBUF_R1C1:TBUF_R2C8`	(XC4000, XC5200, Spartan, SpartanXL, Spartan2, Virtex)

In the following examples, the instance names of two BUFTs are /top-72/rd0 and /top-79/ed7.

Example 1

This example specifies a BUFT adjacent to a specific CLB.

Schematic	`LOC=TBUF_r1c5`
UCF	`INST /top-72/rd0 LOC=TBUF_r1c5 ;`

Place the BUFT adjacent to CLB R1C5. In XC4000, Spartan, and SpartanXL, PAR uses either the longline above the row of CLBs or the longline below. In an XC5200, PAR places the BUFT in one of the four slices of the CLB at row 1, column 5. In Virtex and Spartan2, PAR places the BUFT in one of two slices of the CLB at row 1, column 5.

Example 2

The following example places a BUFT in a specific location.

Schematic	`LOC=TBUF_r1c5.1`
UCF	`INST /top-72/rd0 LOC=TBUF_r1c5.1 ;`

Place the BUFT adjacent to CLB R1C5. In an XC4000, Spartan, or SpartanXL device, .1 tag specifies the longline above the row of CLBs; the .2 tag specifies the longline below it. In an XC5200 device, the .1 tag specifies the longline associated with the slice above the bottom-most slice in the CLB at the location; the .1, .2, .3 tags specify slices above the .0 slice for the specified row and column. In Virtex and Spartan2, the .1 tag specifies the second TBUF in CLB R1C5.

BUFTs that drive the same signal must carry consistent constraints. If you specify .1 or .2 for one of the BUFTs that drives a given signal, you must also specify .1 or .2 on the other BUFTs on that signal; otherwise, do not specify any constraints at all.

Example 3

The next example specifies a column of BUFTs.

Schematic	`LOC=TBUF_r*c3`
UCF	`INST /top-72/rd0 /top-79/ed7 LOC=TBUF_r*c3 ;`

Place BUFTs in column 3 on any row. This constraint might be used to align BUFTs with a common enable signal. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of BUFTs.

Example 4

This example specifies a row of BUFTs .

Schematic	`LOC=TBUF_r7c*`
UCF	`INST /top-79/ed7 LOC=TBUF_r7c* ;`

Place the BUFT on one of the longlines in row 7 for any column. You can use the wild-card (*) character in place of either the row or column number to specify an entire row or column of BUFTs.

CLB Constraint Examples

You can assign soft macros and flip-flops to a single CLB location, a list of CLB locations, or a rectangular block of CLB locations. You can also specify the exact function generator or flip-flop within a CLB. CLB locations are identified as CLB_R*row*C*col* for XC4000, XC5200, Spartan, Spartan2, SpartanXL, and Virtex or *aa* for XC3000, where *aa* is a two-letter designator. The upper left CLB is CLB_R1C1 (for XC4000, XC5200, Spartan, Spartan2, SpartanXL, and Virtex) or AA (for XC3000).

CLB locations can be a fixed location or a range of locations. Use the following syntax to denote fixed locations.

For XC4000, Spartan, SpartanXL:

CLB_R *row***C***col*{**.F** | **.G** | **.FFX** | **.FFY** }

For XC5200:

CLB_R *row***C***col*{**.LC0** | **.LC1** | **.LC2** | **.LC3** }

For Spartan2, Virtex:

CLB_R *row***C***col*{**.S0** | **.S1**}

where

row is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive, or *. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

.F means the CLB is mapped into the F function generator.

.G means the CLB is mapped into the G function generator.

.FFX indicates the X flip-flop in the CLB.

.FFY indicates the Y flip-flop in the CLB.

.LC0 means the bottom-most slice in the XC5200 CLB.

.LC1 means the slice above the .LC0 slice in the XC5200 CLB.

.LC2 means the slice above the .LC1 slice in the XC5200 CLB.

.LC3 means top-most slice in the XC5200 CLB.

.S0 means the right-most slice in the Virtex or Spartan2 CLB.

.S1 means the left-most slice in the Virtex or Spartan2 CLB.

Use the following syntax to denote a range of locations from the highest to the lowest.

CLB_R*row1***C***col***:CLB_R***row2***C***col2*

The following examples illustrate the format of CLB constraints. Enter LOC= and the pin or CLB location. If the target symbol represents a soft macro, the LOC constraint is applied to all appropriate symbols (flip-flops, maps) contained in that macro. If the indicated logic does not fit into the specified blocks, an error is generated.

The following statements place logic in the designated CLB.

`LOC=AA`	(XC3000)
`LOC=CLB_R1C1`	(XC4000, Spartan, SpartanXL)
`LOC=CLB_R1C1.LC0`	(XC5200)
`LOC=CLB_R1C1.S0`	(Spartan2, Virtex)

The following statements place logic within the first column of CLBs. The asterisk (*) is a wildcard character.

`LOC=*A`	(XC3000)
`LOC=CLB_R*C1`	(XC4000, Spartan, SpartanXL)
`LOC=CLB_R*C1.LC0`	(XC5200)
`LOC=CLB_R*C1.S0`	(Spartan2, Virtex)

The next two statements place logic in any of the three designated CLBs. There is no significance to the order of the LOC statements.

`LOC=AA,AB,AC`	(XC3000)
`LOC=CLB_R1C1,CLB_R1C2,CLB_R1C3`	(XC4000, Spartan, SpartanXL, XC5200, Spartan2, Virtex)

The following statements place logic within the rectangular block defined by the first specified CLB in the upper left corner and the second specified CLB towards the lower right corner.

`LOC=AA:HE`	(XC3000)
`LOC=CLB_R1C1:CLB_R8C5`	(XC4000, XC5200, Spartan, SpartanXL, Spartan2, Virtex)

The next statement places logic in the X flip-flop of CLB_R2C2. For the Y flip-flop, use the FFY tag.

`LOC=CLB_R2C2.FFX`	(XC4000, Spartan, SpartanXL)

You can prohibit PAR from using a specific CLB, a range of CLBs, or a row or column of CLBs. Such prohibit constraints can be assigned only through the User Constraints File (UCF). CLBs are prohibited by specifying a PROHIBIT constraint at the design level, as shown in the following examples.

Example 1

Schematic	`None`
UCF	`CONFIG PROHIBIT=clb_r1c5 ;`

Do not place any logic in the CLB in row 1, column 5. CLB R1C1 is in the upper left corner of the device.

Example 2

Schematic	None
UCF	CONFIG PROHIBIT=clb_r1c1:clb_r5c7 ;

Do not place any logic in the rectangular area bounded by the CLB R1C1 in the upper left corner and CLB R5C7 in the lower right.

Example 3

Schematic	None
UCF	CONFIG PROHIBIT=clb_r*c3 ;

Do not place any logic in any row of column 3. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of CLBs.

Example 4

Schematic	None
UCF	CONFIG PROHIBIT=clb_r2c4, clb_r7c9 ;

Do not place any logic in either CLB R2C4 or CLB R7C9.

Delay Locked Loop (DLL) Constraint Examples (Virtex and Spartan2 Only)

You can constrain Virtex and Spartan2 DLL elements—CLKDLL and CLKDLLHF—to a specific physical site name. Specify LOC=DLL and a numeric value (0 through 3) to identify the location.

Following is an example.

Schematic	LOC=DLL1
UCF	INST buf1 LOC=DLL1;

Edge Decoder Constraint Examples (XC4000 Only)

In an XC4000 design, you can assign the decode logic to a specified die edge or half-edge. All elements of a single decode function must lie along the same edge; they cannot be split across two edges of the die. If you use decoder constraints, you must assign all decode inputs for a given function to the same edge. From the schematic, attach LOC constraints to the decode logic — either a DECODE macro or a WAND gate with the DECODE attribute. The constraints are then passed into the EDIF netlist and after mapping is read by PAR.

The format of decode constraints is LOC= and the decode logic symbol location. If the target symbol represents a soft macro containing only decode logic, for example, DECODE8, the LOC constraint is applied to all decode logic contained in that macro. If the indicated decode logic does not fit into the specified decoders, an error is generated.

To constrain decoders to precise positions within a side, constrain the associated pads. However, because PAR determines decoder edges before processing pad constraints, it is not enough to constrain the pads alone. To constrain decoders to a specific die

side, use the following rule. For every output net that you want to constrain, specify the side for at least one of its input decoders (WAND gates), using one of the following.

```
LOC=L        LOC=T
LOC=R        LOC=B
```

The “Legal Edge Designations for Edge Decoders” table shows the legal edge designations.

Example 1

```
Schematic    LOC=T
UCF          INST dec1/$1I1 LOC=T ;
```

Place the decoder along the top edge of the die.

Example 2

```
Schematic    LOC=L
UCF          INST dec1/$1I1 LOC=L ;
```

Place the decoder logic along the left edge of the die.

Example 3

```
Schematic    LOC=LT
UCF          INST dec1/$1I1 LOC=LT ;
```

Place decoders along the top half of the left edge of the die. The first letter in this code represents the die edge, and the second letter represents the desired half of that edge.

Table 12-9 Legal Edge Designations for Edge Decoders

Edge Code	Edge Location
T	Top edge
B	Bottom edge
L	Left edge
R	Right edge
TL	Left half of top edge
TR	Right half of top edge
BL	Left half of bottom edge
BR	Right half of bottom edge
LT	Top half of left edge
LB	Bottom half of left edge
RT	Top half of right edge
RB	Bottom half of right edge

Note: The edges referred to in these constraints are die edges, which do not necessarily correspond to package edges. View the device in the FPGA Editor to determine which pins are on which die edge.

Flip-Flop Constraint Examples

Flip-flops can be constrained to a specific CLB, a range of CLBs, a row or column of CLBs, a specific half-CLB, or one of four specific slices of the XC5200 CLB. Flip-flop constraints can be assigned from the schematic or through the UCF file.

From the schematic, attach LOC constraints to the target flip-flop. The constraints are then passed into the EDIF netlist and are read by PAR after the design is mapped.

The following examples show how the LOC constraint is applied to a schematic and to a UCF (User Constraints File). The instance names of two flip-flops, /top-12/fdrd and /top-54/fdsd, are used to show how you would enter the constraints in the UCF.

Example 1

Schematic	`LOC=clb_rlc5`
UCF	`INST /top-12/fdrd LOC=clb_r1c5 ;`

Place the flip-flop in the CLB in row 1, column 5. CLB R1C1 is in the upper left corner of the device.

Example 2

Schematic	`LOC=clb_r1c1:clb_r5c7`
UCF	`INST /top-12/fdrd LOC=clb_r1c1:clb_r5c7 ;`

Place the flip-flop in the rectangular area bounded by the CLB R1C1 in the upper left corner and CLB R5C7 in the lower right corner.

Example 3

Schematic	`LOC=clb_r*c3`
UCF	`INST /top-12/fdrd/top-54/fdsd LOC=clb_r*c3 ;`

Place the flip-flops in any row of column 3. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of CLBs.

In the following example, repeating the LOC constraint and separating each such constraint by a comma specifies multiple locations for an element. When you specify multiple locations, PAR can use any of the specified locations.

Example 4

Schematic	`LOC=clb_r2c4,clb_r7c9`
UCF	`INST /top-54/fdsd LOC=clb_r2c4,clb_r7c9 ;`

Place the flip-flop in either CLB R2C4 or CLB R7C9.

Example 5

Schematic	`LOC=clb_r3c5.ffx`
UCF	`INST /top-12/fdrd LOC=clb_r3c5.ffx ;`

Place the flip-flop in CLB R3C5 and assign the flip-flop output to the XQ pin. (Use the FFY tag to indicate the YQ pin of the CLB.) If either the FFX or FFY tags are specified, the wildcard (*) character cannot be used for the row or column numbers.

Example 6

Schematic	`PROHIBIT=clb_r5c*`
UCF	`CONFIG PROHIBIT=clb_r5c* ;`

Do not place the flip-flop in any column of row 5. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of CLBs.

The XC5200 CLB is divided into four specific slices for every row and column location on the array. In order to place a flip-flop in a specific slice, use the .LC0, .LC1, .LC2, or .LC3 extension on the location constraint as shown in the following example.

Example 7

Schematic	`LOC=clb_r1c5.LC3`
UCF	`INST /top-12/fdrd LOC=clb_r1c5.LC3 ;`

Place the flip-flop in the top slice of the XC5200 CLB in row 1, column 5.

Global Buffer Constraint Examples

XC3000

You cannot assign placement to the GCLK or ACLK buffers in the XC3000 family, since there is only one each, and their placements are fixed on the die.

XC4000, XC5200, Spartan, SpartanXL

For the XC4000, XC5200, Spartan, and SpartanXL, you can constrain a global buffer — BUFG, BUFGP, BUFGS, BUFGLS, BUFGE, or BUFFCLK— to a corner of the die. From the schematic, attach LOC constraints to the global buffer symbols; specify LOC= and the global clock buffer location. The constraints are then passed into the EDIF netlist and after mapping are read by PAR.

Following is an example.

Schematic	`LOC=TL`
UCF	`INST buf1 LOC=TL ;`

Place the global buffer in the top left corner of the die. The following table shows the legal corner designations.

Table 12-10 Legal Corner Designations for Global Buffers

Corner Code	Corner Location
TL	Top left corner
TR	Top right corner
BL	Bottom left corner
BR	Bottom right corner

If a global buffer is sourced by an external signal, the dedicated IOB for that buffer must not be used by any other signal. For example, if a BUFGP is constrained to TL, the PGCK1 pin must be used to source it, and no other I/O can be assigned to that pin.

Virtex, Spartan2

You can constrain a Virtex or Spartan2 global buffer—BUFGP, and IBUFG_selectIO variants—to a specific buffer site name or dedicated global clock pad in the device model. From the schematic, attach LOC constraints to the global buffer symbols. Specify LOC= and GCLKBUF plus a number (0 through 3) to create a specific buffer site name in the device model. Or, specify LOC= and GCLKPAD plus a number (0 through 3) to create a specific dedicated global clock pad in the device model.The constraints are then passed into the EDIF netlist and after mapping are read by PAR.

Following is an example.

Schematic	`LOC=GCLKBUF1`
UCF	`INST buf1 LOC=GCLKBUF1;`
Schematic	`LOC=GCLKPAD1`
UCF	`INST buf1 LOC=GCLKPAD1;`

I/O Constraint Examples

You can constrain I/Os to a specific IOB. You can assign I/O constraints from the schematic or through the UCF file.

From the schematic, attach LOC constraints to the target PAD symbol. The constraints are then passed into the netlist file and read by PAR after mapping.

Alternatively, in the UCF file a pad is identified by a unique instance name. The following example shows how the LOC constraint is applied to a schematic and to a UCF (User Constraints File). In the examples, the instance names of the I/Os are /top-102/data0_pad and /top-117/q13_pad. The example uses a pin number to lock to one pin.

Schematic	`LOC=p17`
UCF	`INST /top-102/data0_pad LOC=p17 ;`

Place the I/O in the IOB at pin 17. For pin grid arrays, a pin name such as B3 or T1 is used.

IOB Constraint Examples

You can assign I/O pads, buffers, and registers to an individual IOB location. IOB locations are identified by the corresponding package pin designation.

The following examples illustrate the format of IOB constraints. Specify LOC= and the pin location. If the target symbol represents a soft macro containing only I/O elements, for example, INFF8, the LOC constraint is applied to all I/O elements contained in that macro. If the indicated I/O elements do not fit into the specified locations, an error is generated.

The following statement places the I/O element in location P13. For PGA packages, the letter-number designation is used, for example, B3.

```
LOC=P13
```

You can prohibit the mapper from using a specific IOB. You might take this step to keep user I/O signals away from semi-dedicated configuration pins. Such prohibit constraints can be assigned only through the UCF file.

IOBs are prohibited by specifying a PROHIBIT constraint preceded by the CONFIG keyword, as shown in the following example.

Schematic	None
UCF	CONFIG PROHIBIT=p36, p37, p41 ;

Do not place user I/Os in the IOBs at pins 36, 37, or 41. For pin grid arrays, pin names such as D14, C16, or H15 are used.

Mapping Constraint Examples

Mapping constraints control the mapping of logic into CLBs. They have two parts. The first part is a FMAP, HMAP, or CLBMAP component placed on the schematic. The second is a LOC constraint that can be placed on the schematic or in the constraints file.

CLBMAP (XC3000 Only)

With the CLBMAP symbol, you can specify logic mapping at the schematic level for all XC3000 designs. It is used in conjunction with standard logic elements, such as gates and flip-flops. It implicitly specifies the configuration of a CLB by defining the signals on its pins. Use the CLBMAP symbol to control mapping when the default mapping is not acceptable.

Enter the CLBMAP symbol on the schematic and assign signals to its pins. MAP processes this symbol and maps the appropriate logic, as defined by the input and output signals, into one CLB. The easiest way to define a CLBMAP is to connect a labeled wire segment to each pin, which connects that pin to the net carrying the same label.

If a CLBMAP specifies an illegal CLB configuration, MAP issues an error explaining why the CLBMAP is illegal.

A CLBMAP can be either closed or open. A closed CLBMAP must specify both the input and output signals for that CLB. MAP maps a closed CLBMAP exactly as specified, unless the indicated configuration is illegal. MAP does not add any logic to a CLB specified with a closed CLBMAP.

An open CLBMAP specifies the minimum amount of logic to place within a CLB. MAP attempts to place more logic within the CLB as long as the CLB remains valid. MAP only adds logic on the inputs to the CLB. It does not add logic on the output signals. MAP assigns those signals to the CLB output pins and maps the source logic into the CLB as appropriate. Use an open CLBMAP to specify the minimum function of a CLB.

Specify whether a CLBMAP is open or closed by attaching the appropriate MAP attribute to the symbol. See the “Map Attributes for CLBMAP Symbols” table for the exact conventions.

The default configuration for a CLBMAP is unlocked and open.

Table 12-11 Map Attributes for CLBMAP Symbols

	Closed CLB	Open CLB
Pins locked	MAP=PLC	MAP=PLO
Pins unlocked	MAP=PUC	MAP=PUO (default)

Note: Currently, pin locking is not supported. PLC and PLO are translated into PUC and PUO, respectively.

Example 1

Schematic	`LOC=CLB_R1C1`
UCF	`INST top/cntq7 LOC=CLB_R1C1 ;`

Place the CLBMAP in CLB CLB_R1C1.

Example 2

Schematic	`LOC=AA:EE`
UCF	`INST reg/bit7 LOC=AA:EE ;`

Place the CLBMAP in the area bounded by CLB AA in the upper left corner and CLB EE in the lower right.

FMAP and HMAP

The FMAP and HMAP symbols control mapping in an XC4000, Spartan, or SpartanXL design. They are similar to the XC3000 CLBMAP symbol. The FMAP may also be used to control mapping XC5200, Spartan2, or Virtex designs.

FMAP and HMAP control the mapping of logic into function generators. These symbols do not define logic on the schematic; instead, they specify how portions of logic shown elsewhere on the schematic should be mapped into a function generator.

The FMAP symbol defines mapping into a four-input (F) function generator. The mapper assigns this function to an F or G function generator for XC4000, Spartan, and SpartanXL, so you are not required to specify whether it belongs in F or G. For the XC5200, the four-input function generator defined by the FMAP will be assigned to one of the four slices of the CLB. For Virtex and Spartan2, the four-input function generator defined by the FMAP will be assigned to one of the two slices of the CLB.

The HMAP symbol defines mapping into a three-input (H) function generator for XC4000, Spartan, and SpartanXL. If the HMAP has two FMAP outputs and, optionally, one normal (non-FMAP) signal as its inputs, The mapper places all the logic related to these symbols into one CLB.

An example of how to use these symbols in your schematic appears in the “FMAP and HMAP Schematics” figure and the “Implementation of FMAP and HMAP” figure.

For the FMAP symbol as with the CLBMAP primitive, MAP=PUC or PUO is supported, as well as the LOC constraint. (Currently, pin locking is not supported. MAP=PLC or PLO is translated into PUC and PUO, respectively.)

For the HMAP symbol, only MAP=PUC is supported.

Example 1

Schematic	`LOC=clb_r7c3`
UCF	`INST $1I323 LOC=clb_r7c3;`

Place the FMAP or HMAP symbol in the CLB at row 7, column 3.

Example 2

Schematic	`LOC=clb_r2c4,clb_r3c4`
UCF	`INST top/dec0011 LOC=clb_r2c4,clb_r3c4;`

Place the FMAP or HMAP symbol in either the CLB at row 2, column 4 or the CLB at row 3, column 4.

Example 3

Schematic	`LOC=clb_r5c5:clb_r10c8`
UCF	`INST $3I27 LOC=clb_r5c5:clb_r10c8;`

Place the FMAP or HMAP symbol in the area bounded by CLB R5C5 in the upper left corner and CLB R10C8 in the lower right.

Example 4 (XC4000, Spartan, SpartanXL)

Schematic	`LOC=clb_r10c11.f`
UCF	`INST top/done LOC=clb_r1011.f ;`

Place the FMAP in the F function generator of CLB R10C11. The .G extension specifies the G function generator. An HMAP can only go into the H function generator, so there is no need to specify this placement explicitly.

The XC5200 CLB is divided into four specific slices for every row and column location in the array. In order to place a function generator in a specific slice, use the .LC0, .LC1, .LC2., or LC3 extension on the location constraint on the FMAP as shown in the following example.

Example 5 (XC5200)

Schematic	`LOC=clb_r10c11.LC3`
UCF	`INST /top/done LOC=clb_r10c11.LC3 ;`

Place the FMAP in the top slice of the XC5200 CLB in row 10, column 11.

The Virtex or Spartan2 CLB is divided into two specific slices for every row and column location in the array. In order to place a function generator in a specific slice, use the .S0 (right-most slice) or .S1 (left-most slice) extension on the location constraint on the FMAP as shown in the following example.

Example 6 (Virtex, Spartan2)

Schematic	`LOC=clb_r10c11.S0`
UCF	`INST /top/done LOC=clb_r10c11.S0 ;`

Place the FMAP in the right-most slice of the Virtex or Spartan2 CLB in row 10, column 11.

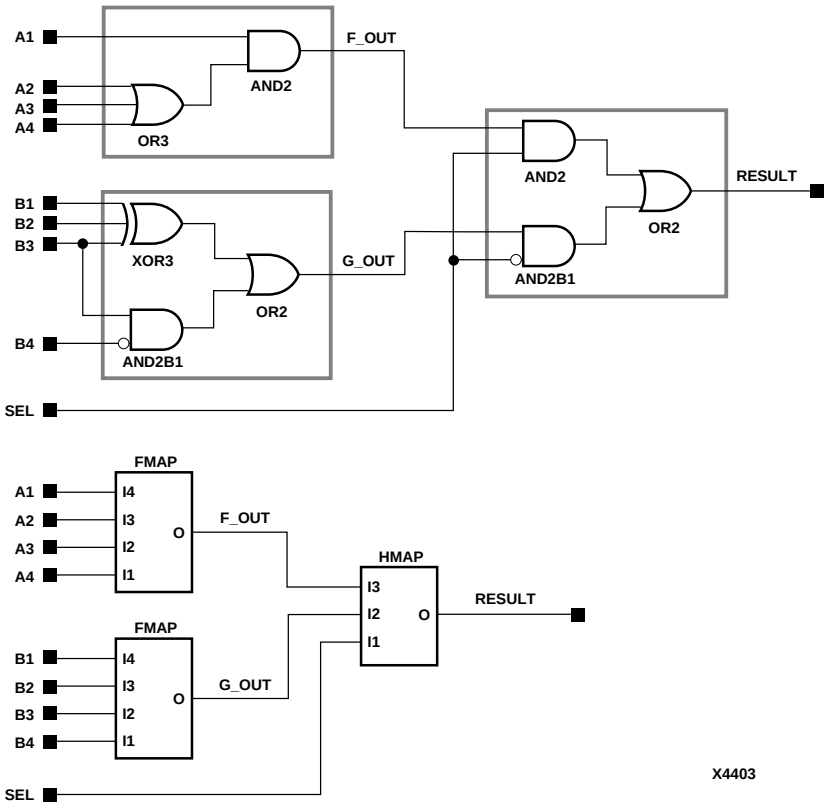


Figure 12-2 FMAP and HMAP Schematics

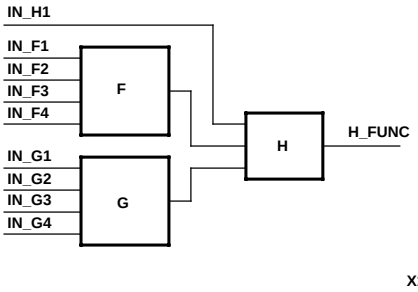


Figure 12-3 Implementation of FMAP and HMAP

RAM and ROM Constraint Examples

You can constrain a ROM or RAM to a specific CLB, a range of CLBs, or a row or column of CLBs. Memory constraints can be assigned from the schematic or through the UCF file.

From the schematic, attach the LOC constraints to the memory symbol. The constraints are then passed into the netlist file and after mapping they are read by PAR. For more information on attaching LOC constraints, see the appropriate interface user guide.

Alternatively, in the constraints file a memory is identified by a unique instance name. One or more memory instances of type ROM or RAM can be found in the input file. All memory macros larger than 16 x 1 or 32 x 1 are broken down into these basic elements in the netlist file.

In the following examples, the instance name of the ROM primitive is /top-7/rq. The instance name of the RAM primitive, which is a piece of a RAM64X8 macro, is /top-11-ram64x8/ram-3.

Example 1

Schematic	`LOC=clb_r1c5`
UCF	`INST /top-7/rq LOC=clb_r1c5 ;`

Place the memory in the CLB in row 1, column 5. CLB R1C1 is in the upper left corner of the device. You can only apply a single-CLB constraint such as this to a 16 x 1 or 32 x 1 memory.

Example 2

Schematic	`LOC=clb_r2c4, clb_r7c9`
UCF	`INST /top-7/rq LOC=clb_r2c4, clb_r7c9 ;`

Place the memory in either CLB R2C4 or CLB R7C9.

Example 3

Schematic	`LOC=clb_r1c1:clb_r5c7`
UCF	`INST /top-17/bigram/*` `LOC=clb_r1c1:clb_r5c7 ;`

Place the LogiBlox module in the rectangular area bounded by the CLB R1C1 in the upper left corner and CLB R5C7 in the lower right.

From the schematic, attach the LOC constraint to the LogiBlox symbol for the bigram block.

In the UCF file, the /* is appended to the end of the LogiBlox symbol instance. The wildcard (*) character here specifies all instances that begin with the /top-17/bigram/ prefix, that is, all RAM elements within the LogiBlox block.

Example 4

Schematic	`PROHIBIT clb_r5c*`
UCF	`CONFIG PROHIBIT=clb_r5c* ;`

Do not place the memory in any column of row 5. You can use the wildcard (*) character in place of either the row or column number in the CLB name to specify an entire row or column of CLBs.

RAMB4 (Block RAM) Constraint Examples (Virtex, Spartan2 Only)

You can constrain a Virtex or Spartan2 block RAM to a specific CLB, a range of CLBs, or a row or column of CLBs. Memory constraints can be assigned from the schematic or through the UCF file. From the schematic, attach the LOC constraints to the memory symbol. The constraints are then passed into the netlist file and after mapping they are read by PAR. For more information on attaching LOC constraints, see the appropriate interface user guide. Alternatively, in the constraints file a memory is identified by a unique instance name.

A Virtex or Spartan2 block RAM has a different row/column grid specification than CLBs and TBUFs. It is specified using RAMB4_RnCn where the numeric row and column numbers refer to the block RAM grid array. A block RAM located at RAMB4_R3C1 is not located at the same site as a flip-flop located at CLB_R3C1.

For example, assume you have a device with two columns of block RAM, each column containing four blocks, where one column is on the right side of the chip and the other is on the left. The block RAM located in the upper left corner is RAMB4_R0C0. Because there are only two columns of block RAM, the block located in the upper right corner is RAMB4_R0C1.

Schematic	`LOC=RAMB4_R0C0`
UCF	`INST /top-7/rq LOC=RAMB4_R0C0 ;`

Relative Location (RLOC) Constraints

Note: This section applies to all FPGA families except XC3000.

The RLOC constraint groups logic elements into discrete sets. You can define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design. For example, if RLOC constraints are applied to a group of eight flip-flops organized in a column, the mapper maintains the columnar order and moves the entire group of flip-flops as a single unit. In contrast, absolute location (LOC) constraints constrain design elements to specific locations on the FPGA die with no relation to other design elements.

Benefits and Limitations of RLOC Constraints

RLOC constraints allow you to place logic blocks relative to each other to increase speed and use die resources efficiently. They provide an order and structure to related design elements without requiring you to specify their absolute placement on the FPGA die. They allow you to replace any existing hard macro with an equivalent that can be directly simulated.

In the Unified Libraries, you can use RLOC constraints with BUFT- and CLB-related primitives, that is, DFF, HMAP, FMAP, and CY4 primitives. You can also use them on non-primitive macro symbols. There are some restrictions on the use of RLOC constraints on BUFT symbols; for details, see the "Fixing Members of a Set at Exact Die Locations" section. You cannot use RLOC constraints with decoders, clocks, or I/O primitives. LOC constraints, on the other hand, can be used on all primitives: BUFTs, CLBs, IOBs, decoders, and clocks.

The following symbols (primitives) accept RLOCs.

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. ROM
8. RAM
9. RAMS, RAMD
10. BUFT
11. WAND primitives that do not have a DECODE attribute attached
12. LUTs, F5MUX, F6MUX, MUXCY, XORCY, MULT_AND, SRL16, SRL16E

Guidelines for Specifying Relative Locations

General syntax for assigning elements to relative locations is

RLOC=R*m***C***n* [**.***extension*]

where *m* and *n* are relative row numbers and column numbers, respectively.

The extension uses the LOC extension syntax as appropriate; for example .1 and .2 for TBUF location.

The extension is required for XC5200 designs in order to fully specify the order of the elements (.LC0, .LC1, .LC2, .LC3). It is required for Virtex and Spartan2 designs to specify the spatial relationship of the objects in the RPM (.S0, .S1).

The row and column numbers can be any positive or negative integer including zero. Absolute die locations, in contrast, cannot have zero as a row or column number. Because row and column numbers in RLOC constraints define only the order and relationship between design elements and not their absolute die locations, their numbering can include zero or negative numbers. Even though you can use any integer in numbering rows and columns for RLOC constraints, it is recommended that you use small integers for clarity and ease of use.

It is not the absolute values of the row and column numbers that is important in RLOC specifications but their relative values or differences. For example, if design element A has an RLOC=R3C4 constraint and design element B has an RLOC=R6C7 constraint, the absolute values of the row numbers (3 and 6) are not important in themselves. However, the difference between them is important; in this case, 3 (6 -3) specifies that the location of design element B is three rows away from the location of design element A. To capture this information, a normalization process is used at some point in the design implementation. In the example just given, normalization would reduce the RLOC on design element A to R0C0, and the RLOC on design element B to R3C3.

In Xilinx programs, rows are numbered in increasing order from top to bottom, and columns are numbered in increasing order from left to right. RLOC constraints follow this numbering convention.

The “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000, Spartan, or SpartanXL Design” figure demonstrates the use of RLOC constraints. Four flip-flop primitives named A, B, C, and D are assigned RLOC constraints as shown. These RLOC constraints require each flip-flop to be placed in a different CLB in the same column and stacked in the order shown — A above B, C, and D. Within a CLB, however, they can be placed either in the FFX or FFY position.

If you wish to place more than one of these flip-flop primitives per CLB, you can specify the RLOCs as shown in the “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000, Spartan, or SpartanXL Design” figure. The arrangement in the figure requires that A and B be placed in a single CLB and that C and D be placed in another CLB immediately below the AB CLB. However, within a CLB, the flip-flops can be placed in either of the two flip-flop positions, FFX or FFY.

To control the ordering of these flip-flop primitives specifically, you can use the extension field, as shown in the “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000, Spartan, or SpartanXL Design” figure. In this figure, the same four flip-flops are constrained to use specific resources in the CLBs. This specification always ensures that these elements are arranged exactly as shown— A must be placed in the FFX spot, B in the same CLB at the FFY spot, and so on.

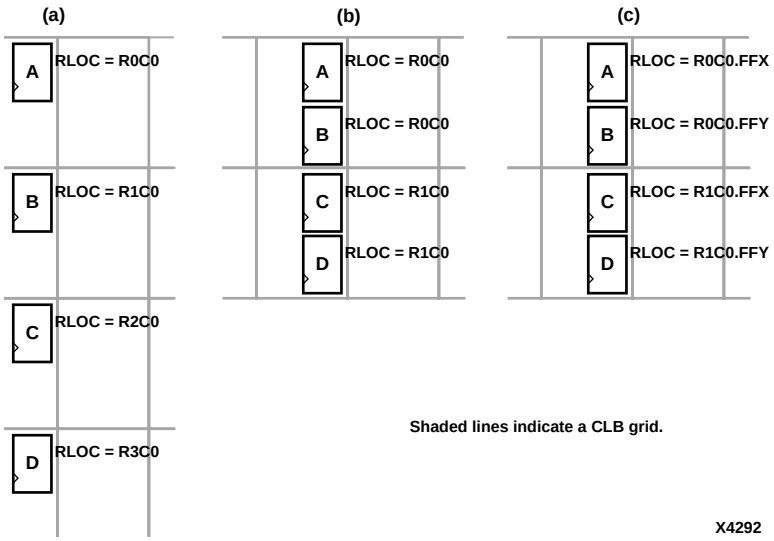


Figure 12-4 Different RLOC Specifications for Four Flip-flop Primitives for an XC4000, Spartan, or SpartanXL Design

RLOC Sets

RLOC constraints give order and structure to related design elements. This section describes RLOC sets, which are groups of related design elements to which RLOC constraints have been applied. For example, the four flip-flops in the "Different RLOC Specifications for Four Flip-flop Primitives for an XC4000, Spartan, or SpartanXL Design" figure are related by RLOC constraints and form a set. Elements in a set are related by RLOC constraints to other elements in the same set. Each member of a set must have an RLOC constraint, which relates it to other elements in the same set. You can create multiple sets, but a design element can belong to one set only.

Sets can be defined explicitly through the use of a set parameter or implicitly through the structure of the design hierarchy.

Four distinct types of rules are associated with each set.

- Definition rules define the requirements for membership in a set.
- Linkage rules specify how elements can be linked to other elements to form a single set.
- Modification rules dictate how to specify parameters that modify RLOC values of all the members of the set.
- Naming rules specify the nomenclature of sets.

These rules are discussed in the sections that follow.

The following sections discuss three different set constraints— U_SET, H_SET, and HU_SET. Elements must be tagged with both the RLOC constraint and one of these set constraints to belong to a set.

U_SET

U_SET constraints enable you to group into a single set design elements with attached RLOC constraints that are distributed throughout the design hierarchy. The letter U in the name U_SET indicates that the set is user-defined. U_SET constraints allow you to group elements, even though they are not directly related by the design hierarchy. By

attaching a U_SET constraint to design elements, you can explicitly define the members of a set. The design elements tagged with a U_SET constraint can exist anywhere in the design hierarchy; they can be primitive or non-primitive symbols. When attached to non-primitive symbols, the U_SET constraint propagates to all the primitive symbols with RLOC constraints that are below it in the hierarchy.

The syntax of the U_SET constraint is the following.

U_SET=*set_name*

where *set_name* is the user-specified identifier of the set. All design elements with RLOC constraints tagged with the same U_SET constraint name belong to the same set. Names therefore must be unique among all the sets in the design.

H_SET

In contrast to the U_SET constraint, which you explicitly define by tagging design elements, the H_SET (hierarchy set) is defined implicitly through the design hierarchy. The combination of the design hierarchy and the presence of RLOC constraints on elements defines a hierarchical set, or H_SET set. You do not use an HSET constraint to tag the design elements to indicate their set membership. The set is defined automatically by the design hierarchy.

All design elements with RLOC constraints at a single node of the design hierarchy are considered to be in the same H_SET set unless they are tagged with another type of set constraint such as RLOC_ORIGIN or RLOC_RANGE. If you explicitly tag any element with an RLOC_ORIGIN, RLOC_RANGE, U_SET, or HU_SET constraint, it is removed from an H_SET set. Most designs contain only H_SET constraints, since they are the underlying mechanism for relationally placed macros. The RLOC_ORIGIN or RLOC_RANGE constraints are discussed further in the “Fixing Members of a Set at Exact Die Locations” section.

NGDBuild recognizes the implicit H_SET set, derives its name, or identifier, attaches the H_SET constraint to the correct members of the set, and writes them to the output file.

The syntax of the H_SET constraint as generated by NGDBuild follows.

H_SET=*set_name*

set_name is the identifier of the set and is unique among all the sets in the design. The base name for any H_SET is “hset,” to which NGDBuild adds a hierarchy path prefix to obtain unique names for different H_SET sets in the NGDBuild output file.

HU_SET

The HU_SET constraint is a variation of the implicit H_SET (hierarchy set). Like H_SET, HU_SET is defined by the design hierarchy. However, you can use the HU_SET constraint to assign a user-defined name to the HU_SET.

The syntax of the HU_SET constraint is the following.

HU_SET=*set_name*

where *set_name* is the identifier of the set; it must be unique among all the sets in the design. You must define the base names to ensure unique hierarchically qualified names for the sets after the mapper resolves the design and attaches the hierarchical names as prefixes.

This user-defined name is the base name of the HU_SET set. Like the H_SET set, in which the base name of "hset" is prefixed by the hierarchical name of the lowest common ancestor of the set elements, the user-defined base name of an HU_SET set is prefixed by the hierarchical name of the lowest common ancestor of the set elements.

The HU_SET constraint defines the start of a new set. All design elements at the same node that have the same user-defined value for the HU_SET constraint are members of the same HU_SET set. Along with the HU_SET constraint, elements can also have an RLOC constraint. The presence of an RLOC constraint in an H_SET constraint links the element to all elements tagged with RLOCs above and below in the hierarchy. However, in the case of an HU_SET constraint, the presence of an RLOC constraint along with the HU_SET constraint on a design element does not automatically link the element to other elements with RLOC constraints at the same hierarchy level or above.

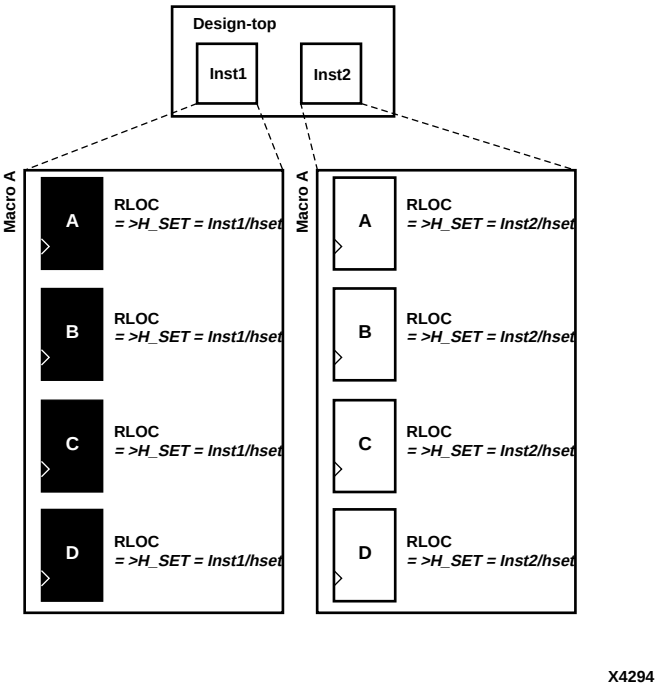


Figure 12-5 Macro A Instantiated Twice

Note: In "Macro A Instantiated Twice" figure and the other related figures shown in the subsequent sections, the italicized text prefixed by => is added by NGDBuild during the design flattening process. You add all other text.

"Macro A Instantiated Twice" figure demonstrates a typical use of the implicit H_SET (hierarchy set). The figure shows only the first "RLOC" portion of the constraint. In a real design, the RLOC constraint must be specified completely with RLOC=*RmCn*. In this example, macro A is originally designed with RLOC constraints on four flip-flops — A, B, C, and D. The macro is then instantiated twice in the design — Inst1 and Inst2. When the design is flattened, two different H_SET sets are recognized because two distinct levels of hierarchy contain elements with RLOC constraints. NGDBuild creates and attaches the appropriate H_SET constraint to the set members: H_SET=Inst1/hset for the macro instantiated in Inst1, and H_SET=Inst2/hset for the macro instantiated in Inst2. The design implementation programs place each of the

two sets individually as a unit with relative ordering within each set specified by the RLOC constraints. However, the two sets are regarded to be completely independent of each other.

The name of the H_SET set is derived from the symbol or node in the hierarchy that includes all the RLOC elements. In the "Macro A Instantiated Twice" figure, Inst1 is the node (instantiating macro) that includes the four flip-flop elements with RLOCs shown on the left of the figure. Therefore, the name of this H_SET set is the hierarchically qualified name of "Inst1" followed by "hset." The Inst1 symbol is considered the "start" of the H_SET, which gives a convenient handle to the entire H_SET and attaches constraints that modify the entire H_SET. Constraints that modify sets are discussed in the "Set Modifiers" section.

The "Macro A Instantiated Twice" figure demonstrates the simplest use of a set that is defined and confined to a single level of hierarchy. Through linkage and modification, you can also create an H_SET set that is linked through two or more levels of hierarchy. Linkage allows you to link elements through the hierarchy into a single set. On the other hand, modification allows you to modify RLOC values of the members of a set through the hierarchy.

RLOC Set Summary

The following table summarizes the RLOC set types and the constraints that identify members of these sets.

Table 12-12 Summary of Set Types

Type	Definition	Naming	Linkage	Modification
Set	A set is a collection of elements to which relative location constraints are applied.			
U_SET= *name*	All elements with the same user-tagged U_SET constraint value are members of the same U_SET set.	The name of the set is the same as the user-defined name without any hierarchical qualification.	U_SET links elements to all other elements with the same value for the U_SET constraint.	U_SET is modified by applying RLOC_ORIGIN or RLOC_RANGE constraints on, at most, one of the U_SET constraint-tagged elements.

Table 12-12 Summary of Set Types

Type	Definition	Naming	Linkage	Modification
H_SET (implicit through hierarchy) is not available as a constraint that you can attach to symbols.	RLOC on the node. Any other constraint removes a node from the H_SET set.	The lowest common ancestor of the members defines the start of the set. The name is the hierarchically qualified name of the start followed by the base name, "hset."	H_SET links elements to other elements at the same node that do not have other constraints. It links down to all elements that have RLOC constraints and no other constraints. Similarly, it links to other elements up the hierarchy that have RLOC constraints but no other constraints.	H_SET is modified by applying RLOC_ORIGIN and RLOC_RANGE at the start of the set: the lowest common ancestor of all the elements of the set.
HU_SET= *name*	All elements with the same hierarchically qualified name are members of the same set.	The lowest common ancestor of the members is prefixed to the user-defined name to obtain the name of the set.	HU_SET links to other elements at the same node with the same HU_SET constraint value. It links to elements with RLOC constraints below.	The start of the set is made up of the elements on the same node that are tagged with the same HU_SET constraint value. An RLOC_ORIGIN or an RLOC_RANGE can be applied to, at most, one of these start elements of an HU_SET set.

Set Linkage

The example in the "Three H_SET Sets" figure explains and illustrates the process of linking together elements through the design hierarchy. Again, the complete RLOC specification, RLOC=R*m*C*n*, is required for a real design.

Note: In this and other illustrations in this section, the sets are shaded differently to distinguish one set from another.

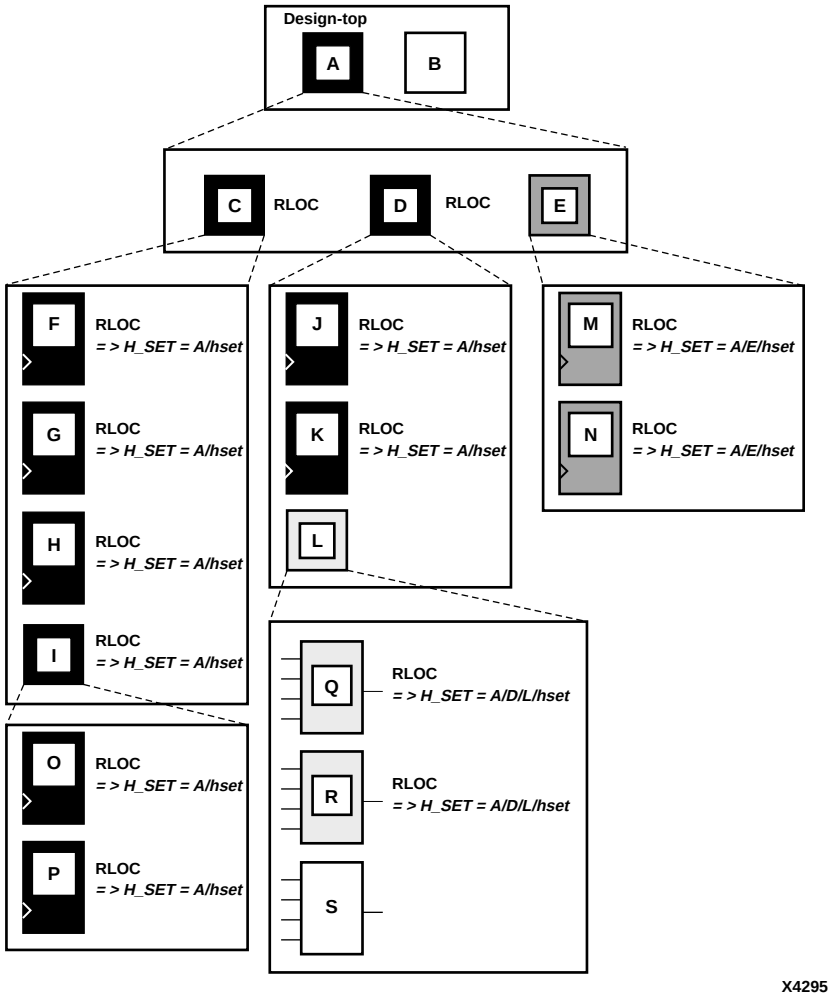


Figure 12-6 Three H_SET Sets

As noted previously, all design elements with RLOC constraints at a single node of the design hierarchy are considered to be in the same H_SET set unless they are assigned another type of set constraint, an RLOC_ORIGIN constraint, or an RLOC_RANGE constraint. In the “Three H_SET Sets” figure, RLOC constraints have been added on primitives and non-primitives C, D, F, G, H, I, J, K, M, N, O, P, Q, and R. No RLOC constraints were placed on B, E, L, or S. Macros C and D have an RLOC constraint at node A, so all the primitives below C and D that have RLOCs are members of a single H_SET set. Furthermore, the name of this H_SET set is “A/hset” because it is at node A that the set starts. The start of an H_SET set is the lowest common ancestor of all the RLOC-tagged constraints that constitute the elements of that H_SET set. Because element E does not have an RLOC constraint, it is not linked to the A/hset set. The RLOC-tagged elements M and N, which lie below element E, are therefore in their own H_SET set. The start of that H_SET set is A/E, giving it the name “A/E/hset.”

Similarly, the Q and R primitives are in their own H_SET set because they are not linked through element L to any other design elements. The lowest common ancestor for their H_SET set is L, which gives it the name “A/D/L/hset.” After the flattening, NGDBuild attaches H_SET=A/hset to the F, G, H, O, P, J, and K primitives; H_SET=A/D/L/hset to the Q and R primitives; and H_SET=A/E/hset to the M and N primitives.

Consider a situation in which a set is created at the top of the design. In “Three H_SET Sets” figure, there would be no lowest common ancestor if macro A also had an RLOC constraint, since A is at the top of the design and has no ancestor. In this case, the base name “hset” would have no hierarchically qualified prefix, and the name of the H_SET set would simply be “hset.”

Set Modification

The RLOC constraint assigns a primitive an RLOC value (the row and column numbers with the optional extensions), specifies its membership in a set, and links together elements at different levels of the hierarchy. In “Three H_SET Sets” figure, the RLOC constraint on macros C and D links together all the objects with RLOC constraints below them. An RLOC constraint is also used to modify the RLOC values of constraints below it in the hierarchy. In other words, RLOC values of elements affect the RLOC values of all other member elements of the same H_SET set that lie below the given element in the design hierarchy.

The Effect of the Hierarchy on Set Modification

When the design is flattened, the row and column numbers of an RLOC constraint on an element are added to the row and column numbers of the RLOC constraints of the set members below it in the hierarchy. This feature gives you the ability to modify existing RLOC values in submodules and macros without changing the previously assigned RLOC values on the primitive symbols. This modification process also applies to the optional extension field. However, when using extensions for modifications, you must ensure that inconsistent extensions are not attached to the RLOC value of a design element that may conflict with RLOC extensions placed on underlying elements. For example, if an element has an RLOC constraint with the FFX extension, all the underlying elements with RLOC constraints must either have the same extension, in this case FFX, or no extension at all; any underlying element with an RLOC constraint and an extension different from FFX, such as FFY or F, is flagged as an error.

After resolving all the RLOC constraints, extensions that are not valid on primitives are removed from those primitives. For example, if NGDBuild generates an FFX extension to be applied on a primitive after propagating the RLOC constraints, it applies the extension if and only if the primitive is a flip-flop. If the primitive is an element other than a flip-flop, the extension is ignored. Only the extension is ignored in this case, not the entire RLOC constraint.

“Adding RLOC Values Down the Hierarchy” figure illustrates the process of adding RLOC values down the hierarchy. The row and column values between the parentheses show the addition function performed by the mapper. The italicized text prefixed by => is added by MAP during the design resolution process and replaces the original RLOC constraint that you added.

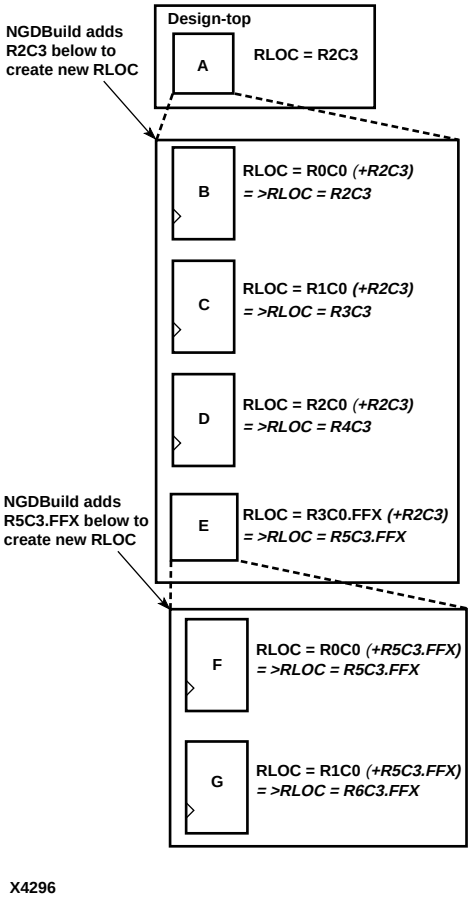


Figure 12-7 Adding RLOC Values Down the Hierarchy

The ability to modify RLOC values down the hierarchy is particularly valuable when instantiating the same macro more than once. Typically, macros are designed with RLOC constraints that are modified when the macro is instantiated. The “Modifying RLOC Values of Same Macro and Linking Together as One Set” figure is a variation of the sample design in the “Macro A Instantiated Twice” figure. The RLOC constraint on Inst1 and Inst2 now link all the objects in one H_SET set. Because the RLOC=R0C0 modifier on the Inst1 macro does not affect the objects below it, the mapper only adds the H_SET tag to the objects and leaves the RLOC values as they are. However, the RLOC=R0C1 modifier on the Inst2 macro causes MAP to change the RLOC values on objects below it, as well as to add the H_SET tag, as shown in the italicized text.

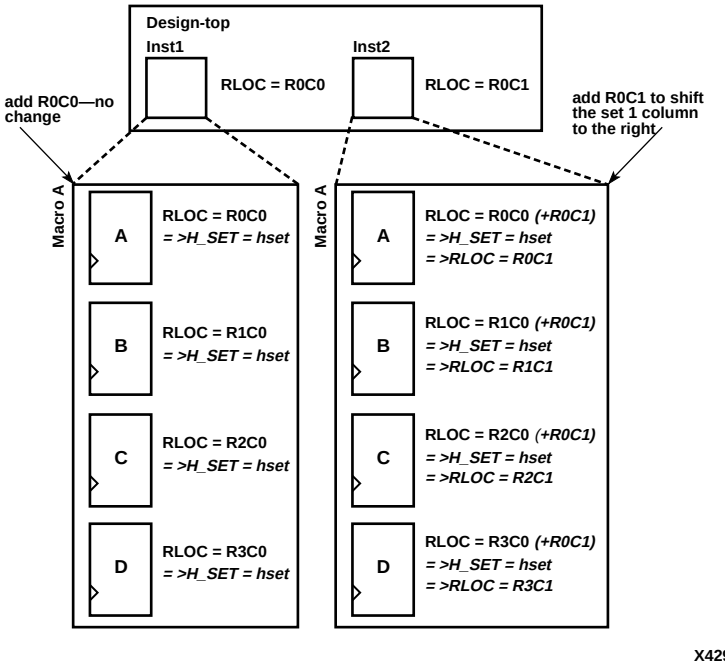


Figure 12-8 Modifying RLOC Values of Same Macro and Linking Together as One Set

Separating Elements from H_SET Sets

The HU_SET constraint is a variation of the implicit H_SET (hierarchy set). The HU_SET constraint defines the start of a new set. Like H_SET, HU_SET is defined by the design hierarchy. However, you can use the HU_SET constraint to assign a user-defined name to the HU_SET.

The “HU_SET Constraint Linking and Separating Elements from H_SET Sets” figure demonstrates how HU_SET constraints designate elements as set members, break links between elements tagged with RLOC constraints in the hierarchy to separate them from H_SET sets, and generate names as identifiers of these sets.

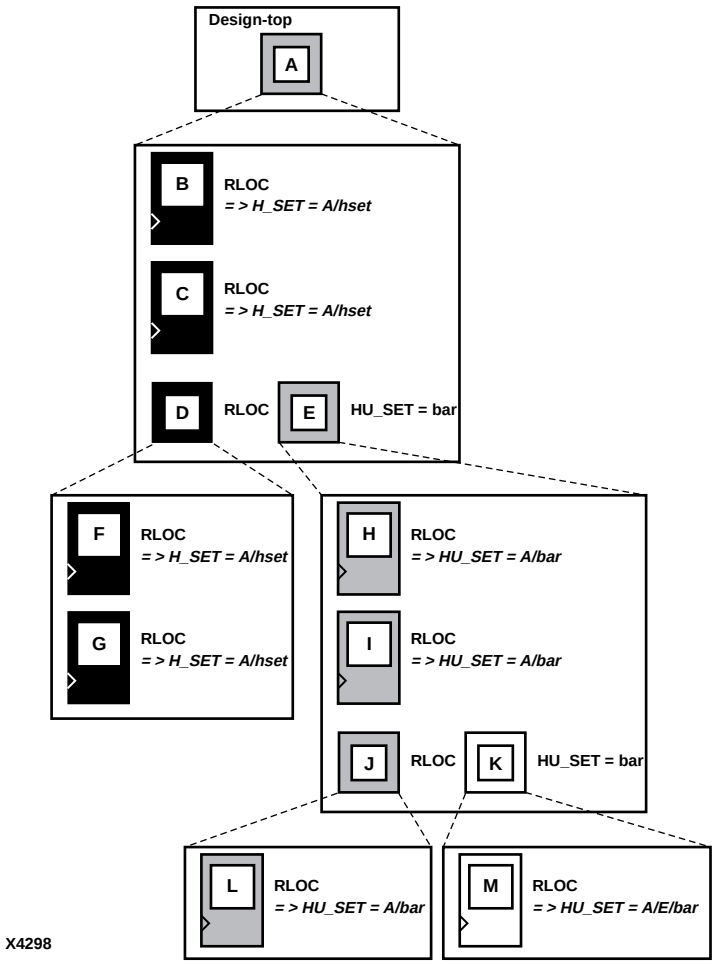


Figure 12-9 HU_SET Constraint Linking and Separating Elements from H_SET Sets

The user-defined HU_SET constraint on E separates its underlying design elements, namely H, I, J, K, L, and M from the implicit H_SET=A/hset that contains primitive members B, C, F, and G. The HU_SET set that is defined at E includes H, I, and L (through the element J). The mapper hierarchically qualifies the name value “bar” on element E to be A/bar, since A is the lowest common ancestor for all the elements of the HU_SET set, and attaches it to the set member primitives H, I, and L. An HU_SET constraint on K starts another set that includes M, which receives the HU_SET=A/E/bar constraint after processing by the mapper. In the “HU_SET Constraint Linking and Separating Elements from H_SET Sets” figure, the same name field is used for the two HU_SET constraints, but because they are attached to symbols at different levels of the hierarchy, they define two different sets.

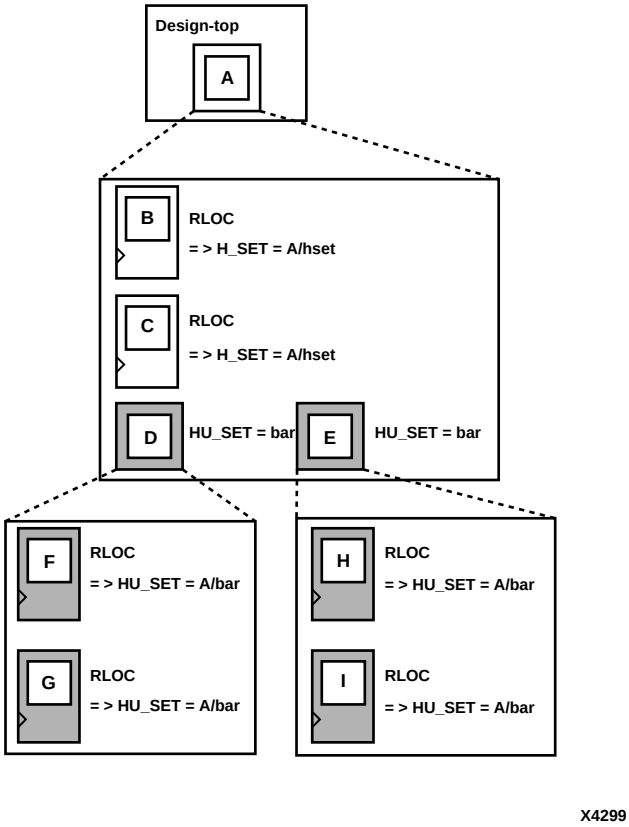


Figure 12-10 Linking Two HU_SET Sets

The “Linking Two HU_SET Sets” figure shows how HU_SET constraints link elements in the same node together by naming them with the same identifier. Because of the same name, “bar,” on two elements, D and E, the elements tagged with RLOC constraints below D and E become part of the same HU_SET.

Set Modifiers

A modifier, as its name suggests, modifies the RLOC constraints associated with design elements. Since it modifies the RLOC constraints of all the members of a set, it must be applied in a way that propagates it to all the members of the set easily and intuitively. For this reason, the RLOC modifiers of a set are placed at the start of that set. The following set modifiers apply to RLOC constraints.

- RLOC

  The RLOC constraint associated with a design element modifies the values of *other* RLOC constraints below the element in the hierarchy of the set. Regardless of the set type, RLOC row, column, and extension values on an element always propagate down the hierarchy and are added at lower levels of the hierarchy to RLOC constraints on elements in the same set.

- RLOC_ORIGIN (see the “RLOC_ORIGIN” section)
- RLOC_RANGE (see the “RLOC_RANGE” section)

Using RLOCs with Xilinx Macros

Xilinx-supplied flip-flop macros include an RLOC=R0C0 constraint on the underlying primitive, which allows you to attach an RLOC to the macro symbol. This symbol links the underlying primitive to the set that contains the macro symbol. Simply attach an appropriate RLOC constraint to the instantiation of the actual Xilinx flip-flop macro. The mapper adds the RLOC value that you specified to the underlying primitive so that it has the desired value.

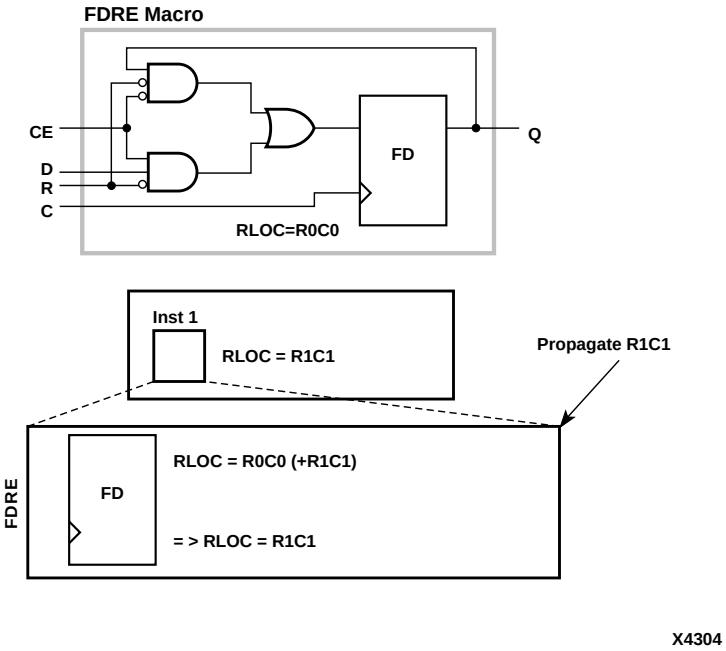


Figure 12-11 Typical Use of a Xilinx Macro

For example, in the “Typical Use of a Xilinx Macro” figure, the RLOC = R1C1 constraint is attached to the instantiation (Inst1) of the FDRE macro. It is added to the R0C0 value of the RLOC constraint on the flip-flop within the macro to obtain the new RLOC values.

If you do not put an RLOC constraint on the flip-flop macro symbol, the underlying primitive symbol is the lone member of a set. The mapper removes RLOC constraints from a primitive that is the only member of a set or from a macro that has no RLOC objects below it.

LOC and RLOC Propagation through Design Flattening

NGDBuild continues to propagate LOC constraints down the design hierarchy. It adds this constraint to appropriate objects that are not members of a set. While RLOC constraint propagation is limited to sets, the LOC constraint is applied from its start point all the way down the hierarchy.

When the design is flattened, the row and column numbers of an RLOC constraint on an element are added to the row and column numbers of the RLOC constraints of the set members below it in the hierarchy. This feature gives you the ability to modify existing RLOC values in submodules and macros without changing the previously assigned RLOC values on the primitive symbols.

Specifying RLOC constraints to describe the spatial relationship of the set members to themselves allows the members of the set to float anywhere on the die as a unit. You can, however, fix the exact die location of the set members. The RLOC_ORIGIN constraint allows you to change the RLOC values into absolute LOC constraints that respect the structure of the set.

The design resolution program, NGDBuild, translates the RLOC_ORIGIN constraint into LOC constraints. The row and column values of the RLOC_ORIGIN are added individually to the members of the set after all RLOC modifications have been made to their row and column values by addition through the hierarchy. The final values are then turned into LOC constraints on individual primitives.

Fixing Members of a Set at Exact Die Locations

As noted in the previous section, you can fix the members of a set at exact die locations with the RLOC_ORIGIN constraint. You must use the RLOC_ORIGIN constraint with sets that include BUFT symbols. However, for sets that do not include BUFT symbols, you can limit the members of a set to a certain range on the die. In this case, the set could "float" as a unit within the range until a final placement. Since every member of the set must fit within the range, it is important that you specify a range that defines an area large enough to respect the spatial structure of the set.

The syntax of this constraint is the following.

RLOC_RANGE=R*m1***C***n1***:R***m2***C***n2*

where the relative row numbers (*m1*, *m2)* and column numbers (*n1*, *n2)* can be non-zero positive numbers, or the wildcard (*) character. This syntax allows for three kinds of range specifications as follows.

- *Rr1Cc1:Rr2Cc2* — A rectangular region enclosed by rows r1, r2, and columns c1, c2
- *R*Cc1:R*Cc2* — A region enclosed by the columns c1 and c2 (any row number)
- *Rr1C*:Rr2C** — A region enclosed by the rows r1 and r2 (any column number)

For the second and third kinds of specifications with wildcards, applying the wildcard character (*) differently on either side of the separator colon creates an error. For example, specifying R*C1:R2C* is an error since the wildcard asterisk is applied to rows on one side and to columns on the other side of the separator colon.

Specifying a Range or Area

To specify a range or area, use the following syntax, which is equivalent to placing an RLOC_RANGE constraint on the schematic.

set_name **RLOC_RANGE=R***m1***C***n1***:R***m2***C***n2*

The range identifies a rectangular area. You can substitute a wildcard (*) character for either the row number or the column number of both corners of the range.

Note: The bounding rectangle applies to all elements in a relationally placed macro, not just to the origin of the set. See the "Relationally Placed Macros (RPMs)" section for more information.

The values of the RLOC_RANGE constraint are not simply added to the RLOC values of the elements. In fact, the RLOC_RANGE constraint does not change the values of the RLOC constraints on underlying elements. It is an additional constraint that is attached automatically by the mapper to every member of a set. The RLOC_RANGE constraint is attached to design elements in exactly the same way as the

RLOC_ORIGIN constraint. The values of the RLOC_RANGE constraint, like RLOC_ORIGIN values, must be non-zero positive numbers since they directly correspond to die locations.

If a particular RLOC set is constrained by an RLOC_ORIGIN or an RLOC_RANGE constraint in the design netlist and is also constrained in the UCF file, the UCF file constraint overrides the netlist constraint.

Toggling the Status of RLOC Constraints

Another important set modifier is the USE_RLOC constraint. It turns the RLOC constraints on and off for a specific element or section of a set. RLOC can be either TRUE or FALSE.

The application of the USE_RLOC constraint is strictly based on hierarchy. A USE_RLOC constraint attached to an element applies to all its underlying elements that are members of the same set. If it is attached to a symbol that defines the start of a set, the constraint is applied to all the underlying member elements, which represent the entire set. However, if it is applied to an element below the start of the set (for example, E in the "Using the USE_RLOC Constraint to Control RLOC Application on H_SET and HU_SET Sets" figure), only the members of the set (H and I) below the specified element are affected.You can also attach the USE_RLOC constraint directly to a primitive symbol so that it affects only that symbol.

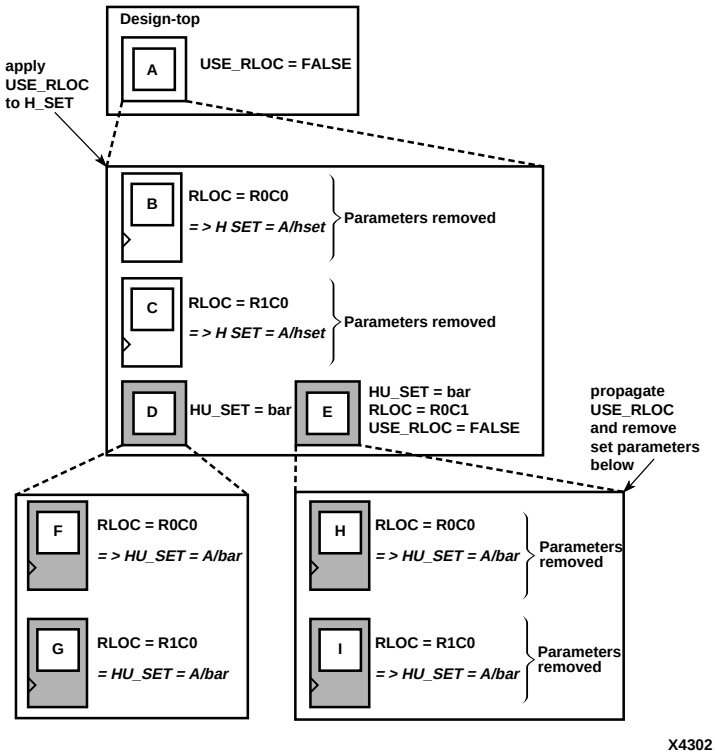


Figure 12-12 Using the USE_RLOC Constraint to Control RLOC Application on H_SET and HU_SET Sets

When the USE_RLOC=FALSE constraint is applied, the RLOC and set constraints are removed from the affected symbols in the NCD file. This process is different than that followed for the RLOC_ORIGIN constraint. For RLOC_ORIGIN, the mapper generates and outputs a LOC constraint in addition to all the set and RLOC constraints in the PCF file. The mapper does not retain the original constraints in the presence of a

USE_RLOC=FALSE constraint because these cannot be turned on again in later programs.

The “Using the USE_RLOC Constraint to Control RLOC Application on H_SET and HU_SET Sets” figure illustrates the use of the USE_RLOC constraint to mask an entire set as well as portions of a set.

Applying the USE_RLOC constraint on U_SET sets is a special case because of the lack of hierarchy in the U_SET set. Because the USE_RLOC constraint propagates strictly in a hierarchical manner, the members of a U_SET set that are in different parts of the design hierarchy must be tagged separately with USE_RLOC constraints; no single USE_RLOC constraint is propagated to all the members of the set that lie in different parts of the hierarchy. If you create a U_SET set through an instantiating macro, you can attach the USE_RLOC constraint to the instantiating macro to allow it to propagate hierarchically to all the members of the set. You can create this instantiating macro by placing a U_SET constraint on a macro and letting the mapper propagate that constraint to every symbol with an RLOC constraint below it in the hierarchy.

The “Using the USE_RLOC Constraint to Control RLOC Application on U_SET Sets” figure illustrates an example of the use of the USE_RLOC=FALSE constraint. The USE_RLOC=FALSE on primitive E removes it from the U_SET set, and USE_RLOC=FALSE on element F propagates to primitive G and removes it from the U_SET set.

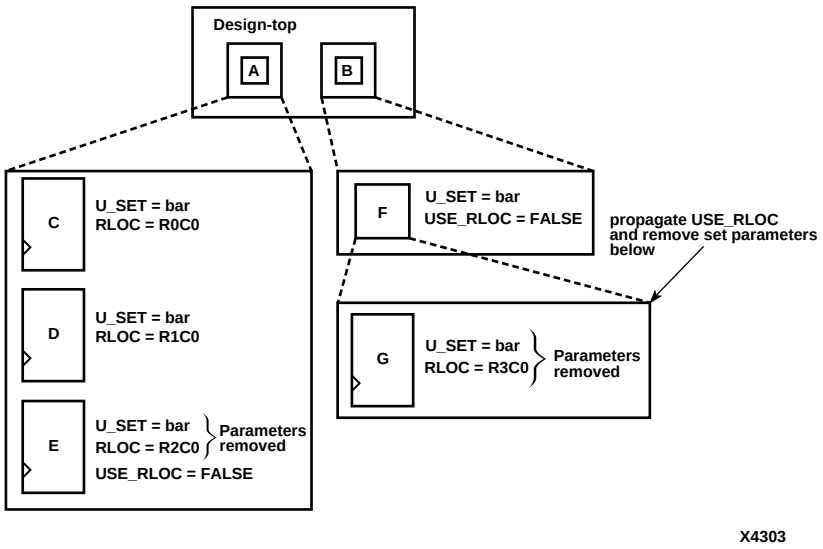


Figure 12-13 Using the USE_RLOC Constraint to Control RLOC Application on U_SET Sets

While propagating the USE_RLOC constraint, the mapper ignores underlying USE_RLOC constraints if it encounters elements higher in the hierarchy that already have USE_RLOC constraints. For example, if the mapper encounters an underlying element with a USE_RLOC=TRUE constraint during the propagation of a USE_RLOC=FALSE constraint, it ignores the newly encountered TRUE constraint.

Choosing an RLOC Origin when Using Hierarchy Sets

To specify a single origin for an RLOC set, use the following syntax, which is equivalent to placing an RLOC_ORIGIN attribute on the schematic.

set_name **RLOC_ORIGIN=R***m***C***n*

The *set_name* can be the name of any type of RLOC set — a U_SET, an HU_SET, or a system-generated H_SET.

The origin itself is expressed as a row number and a column number representing the location of the elements at RLOC=R0C0.

When the RLOC_ORIGIN constraint is used in conjunction with an implicit H_SET (hierarchy set), it must be placed on the element that is the start of the H_SET set, that is, on the lowest common ancestor of all the members of the set.

If you apply an RLOC_ORIGIN constraint to an HU_SET constraint, place it on the element at the start of the HU_SET set, that is, on an element with the HU_SET constraint. However, since there could be several elements linked together with the HU_SET constraint at the same node, the RLOC_ORIGIN constraint can be applied to only one of these elements to prevent more than one RLOC_ORIGIN constraint from being applied to the HU_SET set.

Similarly, when used with a U_SET constraint, the RLOC_ORIGIN constraint can be placed on only one element with the U_SET constraint. If you attach the RLOC_ORIGIN constraint to an element that has only an RLOC constraint, the membership of that element in any set is removed, and the element is considered the start of a new H_SET set with the specified RLOC_ORIGIN constraint attached to the newly created set.

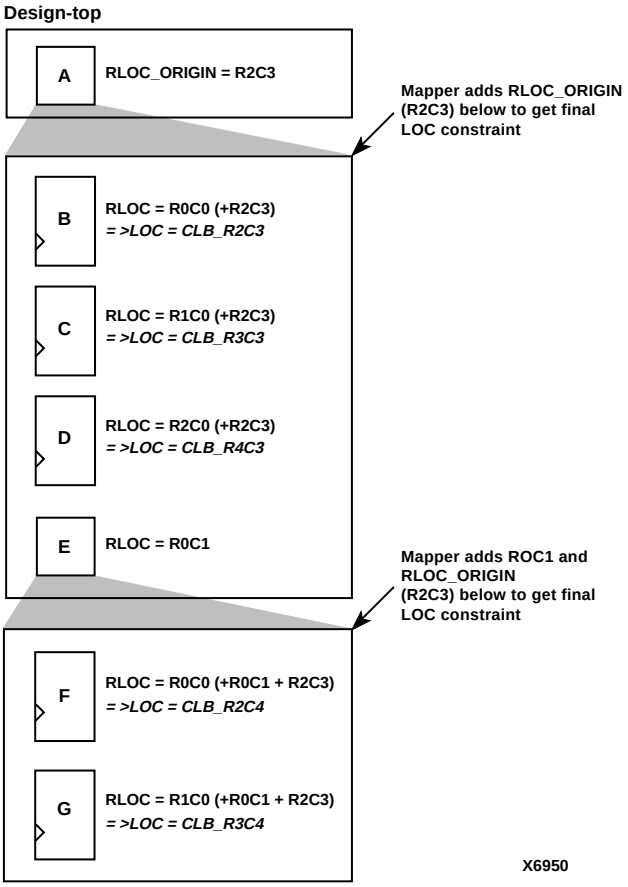


Figure 12-14 Using an RLOC_ORIGIN Constraint to Modify an H_SET Set

In the "Using an RLOC_ORIGIN Constraint to Modify an H_SET Set" figure, the elements B, C, D, F, and G are members of an H_SET set with the name A/hset. This figure is the same as the "Adding RLOC Values Down the Hierarchy" figure except for the presence of an RLOC_ORIGIN constraint at the start of the H_SET set (at A). The RLOC_ORIGIN values are added to the resultant RLOC values at each of the member elements to obtain the values that are then converted by the mapper to LOC constraints. For example, the RLOC value of F, given by adding the RLOC value at E (R0C1) and that at F (R0C0), is added to the RLOC_ORIGIN value (R2C3) to obtain the value of (R2C4), which is then converted to a LOC constraint, LOC = CLB_R2C4.

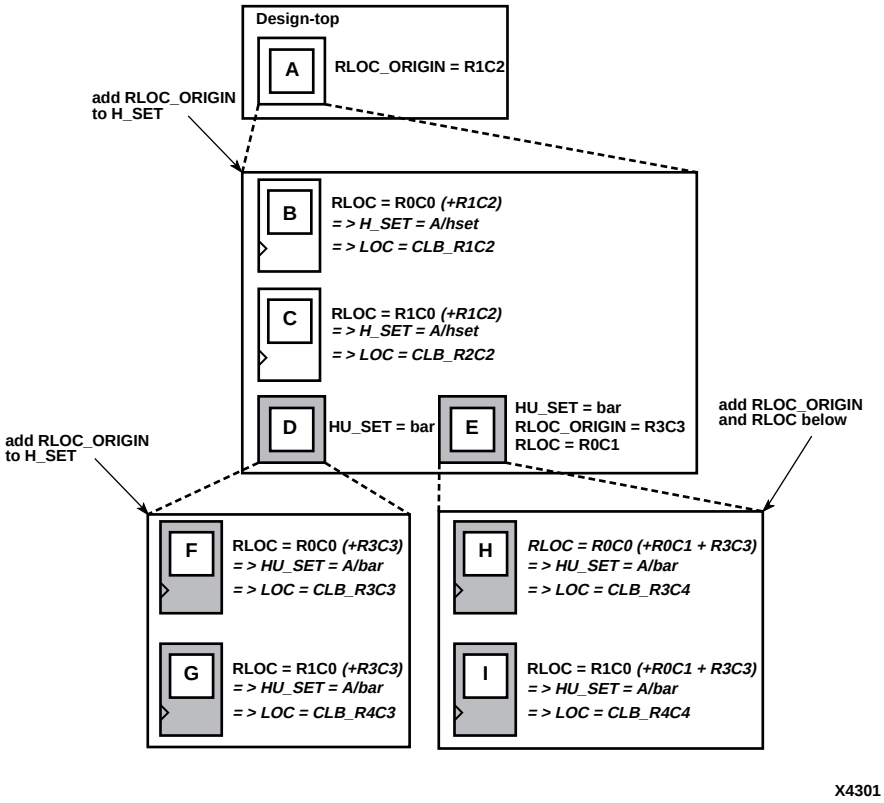


Figure 12-15 Using an RLOC_ORIGIN to Modify H_SET and HU_SET Sets

The “Using an RLOC_ORIGIN to Modify H_SET and HU_SET Sets” figure shows an example of an RLOC_ORIGIN constraint modifying an HU_SET constraint. The start of the HU_SET A / bar is given by element D or E. The RLOC_ORIGIN attached to E, therefore, applies to this HU_SET set. On the other hand, the RLOC_ORIGIN at A, which is the start of the H_SET set A / hset, applies to elements B and C, which are members of the H_SET set.

Timing Constraints

This section describes the syntax for using timing constraints in a UCF file. Timing constraints allow you to specify the maximum allowable delay or skew on any given set of paths or nets in your design.

There are three steps for applying timing specifications to a design.

1. Add TNM attributes to symbols on your schematic to group them into sets. This step is not necessary if you are using only predefined sets. This step can be performed in the schematic or in a constraints file. See the "Using Timing Constraints" chapter in the *Development Systems Reference Guide* for instructions.
2. Add a TIMEGRP symbol and add attributes to the symbol. These attributes can combine the sets defined in step 1 or by pattern matching into additional, more complex, sets, or they can match patterns. This step is optional. You can define these groups on the schematic or in the constraints file.
3. Add a TIMESPEC symbol and add attributes to the symbol, defining the timing requirements for the sets defined in steps 1 and 2. You can define the timing requirements on the schematic or in the constraints file.

TNM Attributes

Timing name (TNM) attributes can be used to identify the elements that make up a group and give them a name that can later be used in an actual timing specification. The value of the attribute can take several forms and there are several attachment mechanisms by which the attribute can identify the elements that make up a group.

TNM attributes can be attached to a net, an element pin, a primitive, or a macro.

TNMs on Nets

The TNM attribute can be placed on any net in the design. It is used to indicate that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged net. Forward tracing stops at any flip-flop, latch, RAM or pad. TNMs do not propagate across IBUFs if they are attached to the input pad net. (Use TNM_NET if you want to trace forward from an input pad net.)

TNMs on Macro or Primitive Pins

The TNM attribute can be placed on any macro or component pin in the design if the design entry package allows placement of attributes on macro or primitive pins. It is used to indicate that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged pin. Forward tracing stops at any flip-flop, latch, RAM or pad.

TNMs on Primitives

Attaching a TNM attribute directly to a primitive defines that primitive as part of the named group.

TNMs on Macro Symbols

A TNM attribute attached to a macro indicates that all elements inside the macro (at all levels of hierarchy below the tagged macro) are part of the named group.

TIMEGRP Constraints

It is sometimes convenient to use existing TNMs to create new groups or to define a group based on the output nets that the group sources. A set of grouping mechanisms has been created to do this. The Timing Group primitive (TIMEGRP) serves as the host for these attributes. Because they contain no keyword, the attributes make no sense when used alone.

You can either attach a TIMEGRP constraint to the TIMEGRP schematic symbol or specify it with the TIMEGRP keyword in the UCF file. In the UCF file, the statement syntax is as follows.

TIMEGRP *timegrp_name*=*timegrp_parameter*

where *timegrp_parameter* is identical to the text you would attach to the TIMEGRP schematic symbol.

You can create groups using the following four methods.

1. Combine multiple groups into one; use the following syntax.

   new_group=group1:group2:. . . groupn

   where *new_group* is the group being defined; *group1*, *group2*, and so forth can be a valid TNM-defined group, predefined group (FFS, PADS, RAMS, LATCHES), or group defined with another TIMEGRP attribute. You can create a time group attribute that references another TIMEGRP attribute that appears after the initial definition. Do not use reserved words such as FFS, PADS, RISING, FALLING, or EXCEPT as group names.

   Example

Schematic	`NEWGRP=OLD1:OLD2`
UCF	`TIMEGRP NEWGRP=OLD1:OLD2 ;`

2. Create groups by exclusion; use the following syntax.

   new_group=group1`:`**EXCEPT** *group2*

   where *new_group* is the group being defined; *group1* and *group2* can be a valid TNM-defined group, predefined group (FFS, PADS, RAMS, LATCHES), or group defined with another TIMEGRP attribute. Do not use reserved words such as FFS, PADS, RISING, FALLING, or EXCEPT as group names.

   Example

Schematic	`FFGRP2=FFS:EXCEPT FFGRP1`
UCF	`TIMEGRP FFGRP2=FFS:EXCEPT FFGRP1 ;`

   You can also specify multiple groups to include or exclude when creating the new group.

   new_group=group1`:`*group2*`:`**EXCEPT** *group3*`:. . .` *groupn*

   where *group1*, *group2*, *group3*, and *groupn* can be a valid TNM-defined group, predefined group (FFS, PADS, RAMS, LATCHES), or group defined with another TIMEGRP attribute. Do not use reserved words such as FFS, PADS, RISING, FALLING, or EXCEPT as group names.

3. Define groups of flip-flops triggered by rising and falling clock edges; use the following syntax.

   new_group=**{RISING** | **FALLING** *group* | **ffs**}

   where *group* must be a group that includes only flip-flops. FFS is a predefined group.

   Example

   Defining a group of flip-flops that switch on the falling edge of the clock.

Schematic	`newfall=FALLING ffs`
UCF	`TIMEGRP newfall=FALLING ffs ;`

4. Use wildcard characters to define groups of symbols whose associated signal names match a specific pattern; use this syntax.

   group=predefined_group pattern

   where *predefined_group* can be one of the following predefined groups: FFS, PADS, RAMS, LATCHES.

   pattern is the string characterizing the output net names of the blocks that you want to include in the new group. It can be any string of characters used with one or more wildcard characters, which can be either of the following.

   An asterisk (*) matches any string of zero or more characters.

   A question mark (?) matches one character.

   Example

   Group created by pattern matching.

Schematic	`newfall=FALLING ffs(A*)`
UCF	`TIMEGRP newfall=FALLING ffs(A*) ;`

TIMESPEC Constraints

After you have defined appropriate groups by attaching TNM attributes to symbols and, optionally, by combining these groups using the TIMEGRP symbol, the next step is to add the timing specifications to the constraints file with the TS*identifier* constraint. You can define these timing requirements by the following means.

The actual timing specifications take the form of attributes that are attached to a timing specification (TIMESPEC) primitive. The TIMESPEC primitive acts as a place to attach attributes and keeps the attributes together. More than one TIMESPEC primitive can be used in a design at any level of the hierarchy.

The sources and destinations can be any synchronous point in the design. The timing allowance specified is used explicitly by the timing analysis tools. There is no hidden accounting for any clock inversions between registers clocked by the same clock, etc.

If paths are not specified, they are ignored for the purposes of timing analysis. The forms described here require the definition of a source and a destination for a specification.

Basic Form

Syntax for defining a maximum allowable delay is as follows.

TS*identifier***=FROM:***source_group***:TO:***dest_group* *allowable_delay*[*units*]

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

In a schematic the timespec attribute is attached to the TIMESPEC symbol.

Defining Intermediate Points on a Path

It is sometimes convenient to define intermediate points on a path to which a specification applies. This defines the maximum allowable delay and has the following syntax.

TS*identifier***=FROM:***source_group* **THRU** *thru_point*[**THRU** *thru_point1*. . . *thru_pointn*]**:TO:***dest_group* *allowable_delay*[*units*]

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

Worst Case Allowable Delay (MAXDELAY)

Syntax for maximum delay is as follows.

TS*identifier***=MAXDELAY FROM:***source_group***:TO:***dest_group* *allowable_delay*[*units*]

Syntax for maximum delay using a through point is as follows.

TS*identifier***=MAXDELAY FROM:***source_group* **THRU** *thru_point* [**THRU** *thru_point1*. . . *thru_pointn*]**:TO:***dest_group* *allowable_delay*[*units*]

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay.

Linked Specifications

This allows you to link the timing number used in one specification to another specification in terms of fractions or multiples.

Note: Circular links are not allowed.

Syntax is as follows.

TS*identifier***=FROM:***source_group***:TO:***dest_group* *another_Tsid* **[/ | *]***number*

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

another_Tsid is a the name of another timespec.

number is a floating point number.

Defining Priority for Equivalent Level Specifications

A conflict between two specifications at the same level of priority can be resolved by defining their priority. You can do this by adding the following text to each of the conflicting specifications.

normal_timespec_syntax **PRIORITY** *integer*

where

normal_timespec_syntax is the timing specification.

integer represents the priority. The smaller the number, the higher the priority.

Note: The PRIORITY keyword cannot be used with the MAXDELAY, MAXSKEW, or PERIOD constraint.

Ignoring Paths

Paths exercising a certain net can be ignored because from a timing specification point of view, all paths through a net, instance, or instance pin may not be important.

Syntax is as follows.

TIG=TS*identifier*

where *identifier* is the timing specification name of the specific timespec for which any paths through the tagged object should be ignored. The attribute can be attached to a net, macro pin or primitive pin. Paths that fan forward from the attribute's point of application are treated as if they don't exist from the viewpoint of timing analysis against the timing specification.

Examples

The following attribute would be attached to a net to inform the timing analysis tools that it should ignore paths through the net for specification TS43.

```
TIG=TS43
```

The following attribute would be created in a UCF file to inform the timing analysis tools that it should ignore paths through the net $1I567/sometimes_slow for specification TS_fast and TS_really_fast.

```
NET $1I567/sometimes_slow TIG=TS_fast , TS_really_fast;
```

Ignoring Paths Through Primitives

The tracing rules for how PAR's timing analysis handles the traversal of primitives are the same as those used for user driven timing analysis. If a user wishes to override the default behavior for an element, the element can be tagged with an override attribute in the PCF file. For more information, see the "Using Timing Constraints" chapter in the *Development System Reference Guide*.

Defining a Clock Period

A clock period specification is used to define to the timing analysis tools the allowable time for paths between elements clocked by the flagged clock signal.

Note: The definition of a clock period is different from a FROM:TO style specification, because the timing analysis tools will automatically take into account any inversions of the clock signal at register clock pins.

There are two methods for specifying clock periods.

Method 1

The quick, convenient way to define the clock period for registers attached to a particular clock net is to attach the following parameter directly to a net in the path that drives the register clock pin(s).

PERIOD=*period*[*units*] [{**HIGH** | **LOW**} [*high_or_low_time* [*hi_lo_units*]]]

where

period is the required clock period.

units is an optional field to indicate the units for the clock period. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms to indicate the intended units.

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time depending on the preceding keyword. If an actual time is specified it must be less than the period. If no High or Low time is specified the default duty cycle is 50%.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

The PERIOD constraint is forward-traced in exactly the same fashion as a TNM would be and attaches itself to all of the flip-flops that the forward tracing reaches. There are no rules about not tracing through certain elements. If you need a more complex form of tracing behavior, for example, where gated clocks are used in the design, you must place the PERIOD on a particular net, or use the preferred method as described in the following paragraphs.

Method 2

The preferred method for defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period. The following attribute is attached to a TIMESPEC symbol in conjunction with a TNM attribute attached to the relevant clock net.

TS*identifier***=PERIOD** *TNM_reference period*[*units*][{**HIGH** | **LOW**} [*high_or_low_time* [*hi_lo_units*]]]

where

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net (or a net in the clock path) using a TNM attribute.

period is the required clock period.

units is an optional field to indicate the units for the clock period. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms to indicate the intended units.

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time depending on the preceding keyword. If an actual time is specified it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is ns, but the High or Low time number can be followed by ps, ns, us, ms, or % if the High or Low time is an actual time measurement.

Example

Clock net sys_clk has the attribute tnm=master_clk attached to it and the following attribute is attached to a TIMESPEC primitive.

```
TS_master=PERIOD master_clk 50 HIGH 30
```

Specifying Derived Clocks

The preferred method of defining a clock period uses an identifier, allowing another clock period specification to reference it. To define the relationship in the case of a derived clock, use the following syntax.

TS*identifier***=PERIOD** *TNM_reference another_PERIOD_identifier* [*/ | **] *number* [{**HIGH | LOW**} [*high_or_low_time* [*hi_lo_units*]]]

where

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net or a net in the clock path using a TNM attribute.

another_PERIOD_identifier is a the name of the identifier used on another period specification.

number is a floating point number.

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time. This must be less than the period, depending on the preceding keyword. The default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

Example

Clock net sub_clk has the attribute tnm=slave_clk attached to it and the following attribute is attached to a TIMESPEC primitive.

```
ts_slave1=PERIOD slave_clk master_clk * 4
```

Controlling Net Skew

You can control the maximum allowable skew on a net by attaching the MAXSKEW attribute directly to the net. Syntax is as follows.

MAXSKEW=*allowable_skew* [*units*]

where

allowable_skew is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

Controlling Net Delay

You can control the maximum allowable delay on a net by attaching the MAXDELAY attribute directly to the net. Syntax is as follows.

MAXDELAY=*allowable_delay* [*units*]

where

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

Physical Constraints

Note: The information in this section applies only to FPGA families.

When a design is mapped, the logical constraints found in the netlist and the UCF file are translated into physical constraints; that is, constraints that apply to a specific architecture. These constraints are found in a mapper-generated file called the Physical Constraints File (PCF). The file contains two sections, the schematic section and the user section. The schematic section contains the physical constraints based on the logical constraints found in the netlist and the UCF file. The user section can be used to add any physical constraints.

PCF File Syntax

The structure of the PCF file is as follows.

```
schematic start;
```

translated schematic and UCF or NCF constraints in PCF format

```
schematic end;
```

user-entered physical constraints

You should put all user-entered physical constraints after the “schematic end” statement.

Note: Do not edit the schematic constraints. They are overwritten every time the mapper generates a new PCF file.

Global constraints need not be attached to any object but should be entered in a constraints file.

The end of each constraint statement must be indicated with a semi-colon.

Note: In all of the constraints files (NCF, UCF, and PCF), instance or variable names that match internal reserved words will be rejected unless the names are enclosed in double quotes. It is good practice to enclose all names in double quotes. For example, the following entry would *not* be accepted because the word net is a reserved word.

```
NET net FAST;
```

Following is the recommended way to enter the constraint.

```
NET “net” FAST;
```

or

```
NET “$SIG_0” FAST ;
```

Syntax Descriptions

A description of each legal physical constraint follows.

Note: Although this section describes the constraint’s syntax for the PCF file, it is preferable to place any user-generated constraint in the UCF file — *not* in an NCF or PCF file.

COMPGRP

Description

Identifies a group of components.

Syntax

COMPGRP *"group_name"=comp_item1... comp_itemn* [**EXCEPT** *comp_group*]**;**

where

comp_item is one of the following,

- COMPONENT "*comp_name*"
- COMPGRP *"group_name"*

FREQUENCY

Description

Identifies the minimum operating frequency for all input pads and sequential output to sequential input pins clocked by the specified net. If no net name is given, the constraint applies to all clock nets in the design that do not have a specific clock frequency constraint.

Syntax

TS*identifier***=FREQUENCY** *frequency_item frequency_value* **;**

frequency_item **FREQUENCY=***frequency_value***;**

where

frequency_item is one of the following,

- NET "*net_name*"
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

frequency_value is one of the following,

- *frequency_number units*
  - *units* can be GHz, MHz, or kHz (gigahertz, megahertz, or kilohertz*)*
- TS*identifier* [{/ |*} *real_number*]

INREG

Description

Forces the placement of a flip-flop or latch close to the IOB so that the two elements can be connected using fast routes. Because XC5200 IOBs do not have flip-flops or latches, you can apply this attribute to meet fast setup timing requirements if a flip-flop or latch is driven by an IOB.

Syntax

NET "*net_name***" INREG ;**

where *net_name* is the name of the net that connects the IOB to the INREG instance.

LOCATE

Description

Specifies a single location, multiple single locations, or a location range.

Syntax

Single or multiple single locations

COMP "*comp_name***" LOCATE=**[**SOFT**] *site_item1*... *site_itemn* [**LEVEL** *n*];

COMPGRP "*group_name***" LOCATE=**[**SOFT**] *site_item1*... *site_itemn* [**LEVEL** *n*];

MACRO "*name***" LOCATE=**[**SOFT**] *site_item1*... *site_itemn* [**LEVEL** *n*];

Range of locations

COMP "*comp_name***" LOCATE=**[**SOFT**] **SITE** "*site_name*" **: "***site_name***"** [**LEVEL** *n*];

COMPGRP "*group_name***" LOCATE=**[**SOFT**] **SITE "***site_name***" : "***site_name***"** [**LEVEL** *n*];

MACRO "*macro_name***" LOCATE=**[**SOFT**] **SITE "***site_name***" : "***site_name***"** [**LEVEL** *n*];

where

site_name is a component site (that is, a CLB or IOB location).

site_item is one of the following,

- SITE "*site_name*"
- SITEGRP "*site_group_name*"

n is 0, 1, 2, 3, or 4.

LOCK

Description

Locks a net that has been previously placed or routed (that is, cannot be unplaced, unrouted, moved, swapped, or deleted). Can also be used to lock all nets.

Syntax

A specific net

"*net_name*" **LOCK;**

All nets

ROUTING LOCK;

MAXDELAY

Description

Identifies a maximum total delay for a net or path in the design. If a net is specified, the maximum delay constraint applies to all driver-to-load connections on the net. If a path is specified, the delay value is the constraint for the path including net and component delays.

Syntax

TS*identifier***=MAXDELA**Y *path path_value***;**

path **MAXDELAY=***path_value***;**

net_delay_item **MAXDELAY=***delay_time* [*units*]**;**

where

path is one of the following,

- PATH "*path_name*"
- ALLPATHS
- FROM *group_item* THRU *group_item1... group_itemn*
- FROM *group_item* THRU *group_item1... group_itemn* TO *group_item*
- THRU *group_item1... group_itemn* TO *group_item*.

path_value is one of the following:

- *delay_time [units]*
  - *units* defaults to nanoseconds, but the delay time number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to specify the units
- *frequency units*
  - *units* can be specified as GHz, MHz, or kHz (gigahertz, megahertz, or kilohertz*)*
- TS*identifier* [{/ |*} *real_number*]

net_delay_item is one of the following:

- NET "*net_name*"
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

MAXSKEW

Description

Specifies a maximum signal skew between a driver and loads on a specified clock signal. Skew is the difference between minimum and maximum load delays on a clock net. If no signal is specified, this constraint applies to all signals which have clock pins as loads and do not have a specified skew constraint.

Syntax

skew_item **MAXSKEW=***time* [*units*]**;**

where

skew_item is one of the following,

- NET *"net_name"*
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

units defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.

OFFSET

Description

Specifies the timing relationship between an external clock and its associated data-in- or data-out-pin.

Can be used on a group of one or more data components or pads.

The OFFSET constraint can be a "global" constraint that applies to all data pad nets in the design for the specified clock. When the COMP "name" specifier is used, the constraint is associated with a single clock IOB component. When the TIMEGRP "group" specifier is used, the constraint is associated with a group of data pad nets.

Optionally, except for CPLDs, a time group qualifier, TIMEGRP "reggroup," can be added to any OFFSET constraint to indicate that the offset applies only to registers specified in the qualifying group. When used with the "Group method," the "register time" group indicates to which design registers clocked by the clock IOB the offset applies.

Syntax

Global method

OFFSET={**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} **COMP** ["*clk_iob_name*"] [**TIMEGRP** "*reggroup*"];

Single net method

NET "*name*" **OFFSET=**{**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} **COMP** ["*clk_iob_name*"] [**TIMEGRP** "*reggroup*"];

Group method

TIMEGRP "*group*" **OFFSET=**{**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} **COMP** ["*clk_iob_namet*"] [**TIMEGRP** "*reggroup*"];

where

group is the name of a time group containing IOB components or PAD bels.

offset_time is the external offset.

units defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.

clk_iob_name is the block name of the clock IOB.

reggroup is a previously defined time group of register BELs. Only registers in the time group clocked by the specified IOB component is checked against the specified offset time.

OUTREG

Description

Forces the placement of a flip-flop or latch close to the IOB so that the two elements can be connected using fast routes. Because XC5200 IOBs do not have flip-flops or latches, you can apply this attribute to meet fast setup timing requirements if a flip-flop or latch is driving an IOB.

Syntax

NET "*net_name*" **OUTREG;**

where *net_name* is the name of the net that connects the IOB to the OUTREG instance.

PATH

Description

Assigns a path specification to a path.

Syntax

PATH *"path_name"=path_spec***;**

where

path_spec is one of the following,

- FROM *group_item* THRU *group_item1... group_itemn*
- FROM *group_item* THRU *group_item1... group_itemn* TO *group_item*
- THRU *group_item1... group_itemn* TO *group_item*.

group_item is one of the following,

- PIN "*pin_name*"
- NET "*net_name*"
- COMP "*comp_name*"
- MACRO "*macro_name*"
- TIMEGRP "*group_name*"
- BEL "*instance_name*"

  BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

PENALIZE TILDE

Description

Penalizes those delays that are reported as only approximate (signified with a tilde (~) in delay reports) by a user-specified percentage. When the penalize tilde constraint is applied to an approximate delay, the delay will be penalized by the designated percentage in subsequent timing checks. Default for percent value is zero.

Syntax

PENALIZE TILDE=*percent*

PERIOD

Description

Assigns a timing period to a timing specification.

Syntax

TS*identifer***=PERIOD** *period_item period_value* [{**LOW** | **HIGH**}{*time* [*units*] | *percent*}];

period_item **PERIOD=***period_value* [{**LOW** | **HIGH**}{*time* [*units*] | *percent*}];

where

period_item is one of the following,

- NET "*net_name*"
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

period_value is one of the following,

- *time* [*units*]
  - *units* defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.
- TS *identifier* [{ / | *} *real_number*]

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

PIN

Description

Identifies a specific pin.

Syntax

PIN *"pin_name"=pin_spec*;

where

pin_spec is one of the following,

- NET *"net_name"* BEL "*instance_name"*
- NET *"net_name*" COMP "*comp_name"*
- COMP "*comp_name"* NET "*net_name"*
- NET *"net_name*" MACRO *"macro_name"*
- MACRO "*macro_name"* NET *"net_name"*
- BEL "*instance_name"* NET "*net_name"*

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

PRIORITIZE

Description

Assigns a weighted importance to a net or bus. Values range from 0 through 100, with 100 being the highest priority and 0 the lowest. The default is 3. Any net with a priority of 3 is not considered critical; no constraint will be generated. The prioritize constraint is used by PAR, which assigns longlines by net priority and routes higher-priority nets before routing lower-priority nets. The prioritize constraint is also used by BITGEN to determine which nets not to use for tiedown. A net with a priority greater than 3 will only be used for tiedown as a last resort.

Syntax

NET *"net_name"* **PRIORITIZE=***integer*;

PROHIBIT

Description

Disallows the use of a site or multiple sites within PAR, FPGA Editor, and the CPLD fitter.

Syntax

Single or multiple single locations

PROHIBIT= *site_group*;

PROHIBIT= *site_group1*... ,*site_groupn*;

Range of locations

PROHIBIT= *site_group* **:** *site_group***;**

where

site_group is one of the following,

- SITE *"site_name"*
- SITEGRP *"site_group_name"*

site_name must be a valid site for the targeted device. (For example, CLB_R1C1.FFX is not a valid site for the XC4000X or SpartanXL.)

Note: CPLDs do not support the "Range of locations" form of PROHIBIT.

SITEGRP

Description

Identifies a group of sites.

Syntax

SITEGRP *site_group_name=site_group1. . . site_groupn* **;** [**EXCEPT** *site_group*]**;**

where

site_group is one of the following,

- SITE "*site_name"*
- SITEGRP *"site_group_name"*

site_name must be a valid site for the targeted device. (For example, CLB_R1C1.FFX is not a valid site for the XC4000X or SpartanXL.)

TEMPERATURE

Description

Allows the specification of the operating temperature for commercial ranges.

Note: Each architecture has its own specific range of valid operating temperatures. If the entered temperature does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead.

Syntax

TEMPERATURE=*value*[**C** |**F**| **K**]

where

value is an integer or a real number specifying the temperature.

C, K, and F are the temperature units. F is degrees Fahrenheit, K is degrees Kelvin, and C is degrees Celsius, the default.

TIMEGRP (Timing Group)

Description

Defines objects that are to be treated as a group for timing considerations.You can refer to a group of flip-flops, input latches, pads, or RAMs by using the corresponding keywords.

Keyword	Description
FFS	CLB or IOB flip-flops only; not flip-flops built from function generators; shift register LUTs in Virtex and Spartan2 are not included
LATCHES	CLB or IOB latches only; not latches built from function generators
PADS	Input/output pads
RAMS	For architectures with RAMS. For Virtex and Spartan2, LUT RAMS and Block RAMS are included.

Syntax

TIMEGRP *"group_name"*=[*qualifier1*] *group_spec1*... [*qualifiern*] *group_specn* [**EXCEPT** *group_spec1*... *group_specn*];

where

qualifier is RISING or FALLING.

group_spec is one of the following,

- PIN "*pin_name*"
- NET *"net_name*"
- BEL "*instance_name"*
- COMP *"comp_name"*
- MACRO *"macro_name"*
- TIMEGRP *"group_name"*
- FFS [*"pattern"*]
- LATCHES [*"pattern"*]
- RAMS [*“pattern”*]
- PADS [*“pattern”*]

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

This example shows you one way to use the TIMEGRP attribute. If you have some outputs that can be slower than others, you can create timespecs similar to those shown below for output signals obc_data(7:0) and ingr_irq_n.

First create the Timegroups.

```
TIMEGRP slow_outs=PADS(obc_data* : ingr_irq_n) ;
TIMEGRP fast_outs=PADS : EXCEPT : slow_outs ;
```

Then apply a timing spec to the Timegroups.

```
TIMESPEC TS08=FROM : FFS : TO : fast_outs : 22 ;
TIMESPEC TS09=FROM : FFS : TO : slow_outs : 75 ;
```

TIG (Timing Ignore)

Description

Identifies paths that can be ignored for timing purposes.

Syntax

ignore_item **TIG** [**=TS***identifier1*. . . **TS***identifiern*];

where

ignore_item is one of the following,

- PIN "*pin_name*"
- NET "*net_name*"
- COMP "*comp_name*"
- MACRO "*macro_name*"
- PATH "*path_name*"
- BEL "*instance_name*"
- FROM *group_item* THRU *group_item1... group_itemn*
- FROM *group_item* THRU *group_item1... group_itemn*TO *group_item*
- THRU *group_item... group_itemn* TO *group_item* }

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

For a detailed description of TIG, see the "Using Timing Constraints" chapter in the *Development System Reference Guide.*

TSidentifier

Description

Assigns a timing period or frequency to a timing specification.

Syntax

Period

TS*identifer***=PERIOD** *period_item period_value* [{**LOW** | **HIGH**}{*time* [*units*] | *percent*}];

period_item **PERIOD=***period_value* [{**LOW** | **HIGH**}{*time* [*units*] | *percent*}];

where

period_item is one of the following,

- NET "*net_name*"
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

period_value is one of the following,

- *time* [*units*]
  - *units* defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.
- TS *identifier* [{ / | *} *real_number*]

HIGH or LOW can be optionally specified to indicate whether the first pulse is to be High or Low.

high_or_low_time is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.

hi_lo_units is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, us, ms, or % if the High or Low time is an actual time measurement.

Frequency

TS*identifier***=FREQUENCY** *frequency_item frequency_value* **;**

frequency_item **FREQUENCY=***frequency_value***;**

where

frequency_item is one of the following,

- NET "*net_name"*
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

frequency_value is one of the following,

- *frequency_number units*
  - *units* can be GHz, MHz, or kHz (gigahertz, megahertz, or kilohertz*)*
- TS*identifier* [{ / |*} *real_number*]

VOLTAGE

Description

Allows the specification of the operating voltage for commercial ranges. This provides a means of prorating delay characteristics based on the specified voltage.

Note: Each architecture has its own specific range of supported voltages. If the entered voltage does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead.

Syntax

VOLTAGE=*value*[**V**]

where

value is an integer or real number specifying the voltage.

V specifies volts, the default voltage unit.

Relationally Placed Macros (RPMs)

The Xilinx libraries contain three types of elements.

- Primitives are basic logical elements such as AND2 and OR2 gates
- Soft macros are schematics made by combining primitives and sometimes other soft macros
- Relationally placed macros (RPMs) are soft macros that contain relative location constraint (RLOC) information, carry logic symbols, and FMAP / HMAP symbols, where appropriate

The last item mentioned above, RPMs, applies only to FPGA families.

The relationally placed macro (RPM) library uses RLOC constraints to define the order and structure of the underlying design primitives. Because these macros are built upon standard schematic parts, they do not have to be translated before simulation. The components that are implemented as RPMs are listed in the “Relationally Placed Macros” section of the “Selection Guide” chapter.

Designs created with RPMs can be functionally simulated. RPMs can, but need not, include all the following elements.

- FMAPs, HMAPs, and CLB-grouping attributes to control mapping. FMAPs and HMAPs have pin-lock attributes, which allow better control over routing. FMAPs and HMAPs are described in the “Mapping Constraint Examples” section.
- Relative location (RLOC) constraints to provide placement structure. They allow positioning of elements relative to each other. They are discussed in the “Benefits and Limitations of RLOC Constraints” section.
- Carry logic primitive symbols. Carry logic is discussed in the “Carry Logic in XC4000, Spartan, SpartanXL” section.

The RPM library offers the functionality and precision of the hard macro library with added flexibility. You can optimize RPMs and merge other logic within them. The elements in the RPM library allow you to access carry logic easily and to control mapping and block placement. Because RPMs are a superset of ordinary macros, you can design them in the normal design entry environment. They can include any primitive logic. The macro logic is fully visible to you and can be easily back-annotated with timing information.

Carry Logic in XC4000, Spartan, SpartanXL

In the XC4000, Spartan, and SpartanXL, the CLB contains a feature called dedicated carry logic. This carry logic is independent of the function generators, although it shares some of the same input pins. Dedicated interconnect propagates carry signals through a column of CLBs.

This section describes the use of carry logic in XC4000, Spartan, and SpartanXL CLBs and lists all the carry logic configuration mnemonics available.

Carry Logic Overview

The carry chain in XC4000E devices can run either up or down. At the top and bottom of columns where there are no CLBs above and below, the carry is propagated to the right as shown in the figure below.

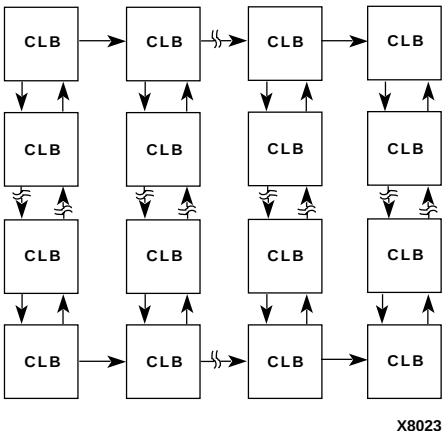


Figure 12-16 Available XC4000E Carry Propagation Paths

In XC4000X, Spartan, and SpartanXL devices the carry chain travels upward only. Standard interconnect can be used to route a signal in the downward direction. See the figure below.

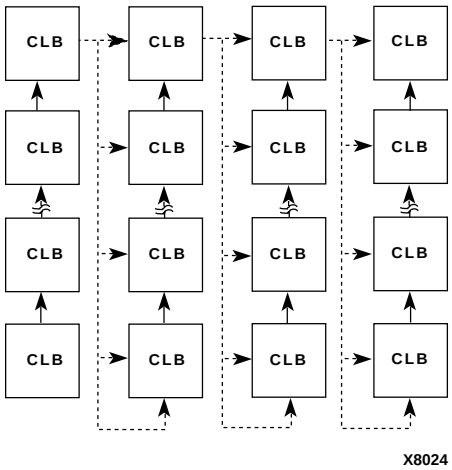


Figure 12-17 Available XC4000X, Spartan, and SpartanXL Carry Propagation Paths (dotted lines use general interconnect)

The CY4_43 carry mode component (Force-G4) forces the signal on the G4 pin to pass through to the COUT pin. This component is available only for XC4000X and SpartanXL devices.

Carry logic in each CLB can implement approximately 40 different functions, which you can use to build faster and more efficient adders, subtracters, counters, comparators, and so forth. The "XC4000, Spartan, SpartanXL Carry Logic" figure shows the carry logic in an XC4000, Spartan, or SpartanXL CLB.

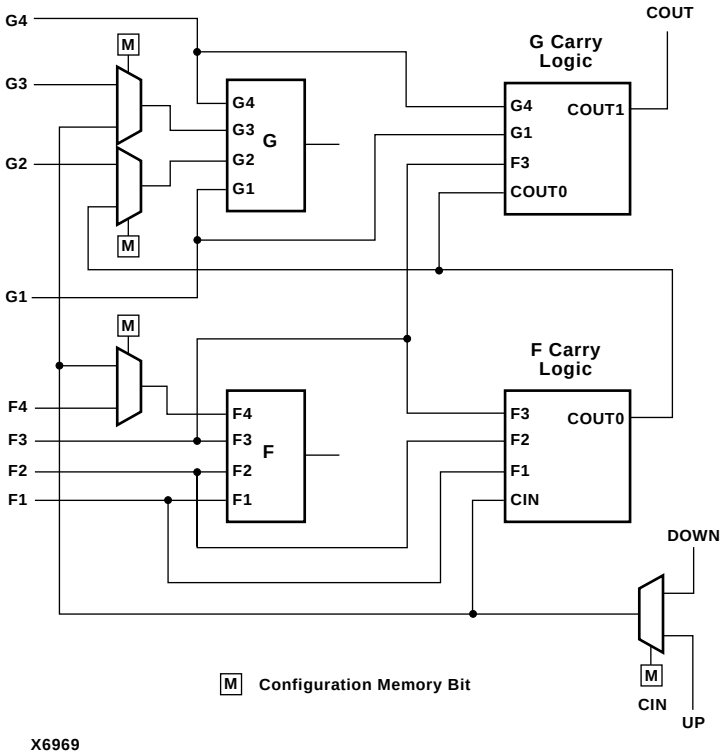


Figure 12-18 XC4000, Spartan, SpartanXL Carry Logic

Carry Logic Primitives and Symbols

The schematic capture libraries that Xilinx supports contain one generic carry logic primitive and several specific carry mode primitive symbols. The generic carry logic primitive represents the complete carry logic in a single CLB and is shown in the "Representative Carry Logic Symbol" figure.

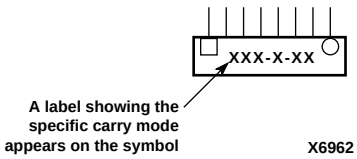


Figure 12-19 Representative Carry Logic Symbol

The carry mode primitive symbols represent unique carry modes, such as ADD-FG-CI. The "Carry Modes" table lists the carry mode names and symbols.

To specify the particular mode that you wish, connect a carry mode symbol to the C0-C7 mode pins of the carry logic symbol. It is the pair of symbols that defines the specific kind of carry logic desired.

A carry logic symbol requires you to place either a LOC or an RLOC constraint on it. If a LOC constraint is used, it must be a single LOC= constraint; it cannot be an area or prohibit LOC constraint or use wildcards in its syntax.

Table 12-13 Carry Modes

Carry Mode Name	Symbol
ADD-F-CI	cy4_01
ADD-FG-CI	cy4_02
ADD-G-F1	cy4_03
ADD-G-CI	cy4_04
ADD-G-F3-	cy4_05
ADDSUB-F-CI	cy4_12
ADDSUB-FG-CI	cy4_13
ADDSUB-G-CI	cy4_15
ADDSUB-G-F1	cy4_14
ADDSUB-G-F3-	cy4_16
FORCE-0	cy4_37
FORCE-1	cy4_38
FORCE-CI	cy4_40
FORCE-F1	cy4_39
FORCE-F3-	cy4_41
FORCE-G4	cy4_43*
EXAMINE-CI	cy4_42
DEC-F-CI	cy4_24
DEC-FG-0	cy4_26
DEC-FG-CI	cy4_25
DEC-G-0	cy4_27
DEC-G-CI	cy4_29
DEC-G-F1	cy4_28
DEC-G-F3-	cy4_30
INC-F-CI	cy4_17
INC-FG-1	cy4_19
INC-FG-CI	cy4_18
INC-G-1	cy4_20
INC-G-CI	cy4_22
INC-G-F1	cy4_21
INC-G-F3-	cy4_23
SUB-F-CI	cy4_06
SUB-FG-CI	cy4_07
SUB-G-1	cy4_08

Table 12-13 Carry Modes

Carry Mode Name	Symbol
SUB-G-CI	cy4_09
SUB-G-F1	cy4_10
SUB-G-F3-	cy4_11
INCDEC-F-CI	cy4_31
INCDEC-FG-1	cy4_33
INCDEC-FG-CI	cy4_32
INCDEC-G-0	cy4_34
INCDEC-G-CI	cy4_36
INCDEC-G-F1	cy4_35

*Available only for XC4000X and SpartanXL devices

Carry Logic Handling

The mapper checks for legal connections between carry logic symbols and also performs simple trimming on some carry modes. CY4 symbols might be trimmed as follows.

- If neither the COUT0 pin nor the COUT pin is used, the CY4 symbol is removed from the design. However, if the signal on the CIN pin connects to other logic, the mapper converts the CY4 to the EXAMINE-CI mode. An EXAMINE-CI mode CY4 is trimmed only if there is no other load on the signal on the CIN pin.
- If the specified mode does not require any of the A0, B0, A1, B1, and / or ADD CY4 inputs, signals are removed from these pins, which may save routing resources.

Carry Mode Configuration Mnemonics

The first step in configuring a CLB for carry logic is to choose the appropriate carry mode configuration mnemonic. Each of the 43 possible configurations of the carry logic has been assigned a three-part mnemonic code, for example:

```
ADD-FG-CI
```

- The first field (ADD) describes the operation performed in the CLB function generators, in this case, a binary addition. By implication, the carry logic in this CLB calculates the carry for this addition.
- The second field (FG) indicates which of the two function generators is used in the specified operation, in this case, both F and G.
- The last field (CI) specifies the source of the carry-in signal to the CLB, in this case, the CIN pin itself.

Consider another example:

```
INCDEC-G-F1
```

This mnemonic describes a CLB in which the G function generator performs an increment / decrement function. The carry-in to this CLB is sourced by the F1 pin.

All available carry mode configuration mnemonics are listed in the next section, the “Carry Logic Configurations” section.

To determine which carry mode primitive corresponds to which mnemonic, see the "Carry Modes" table.

Carry Logic Configurations

This section lists and describes all the available carry mode configuration mnemonics. The following information is given for each mnemonic.

- The name of the mode mnemonic.
- A brief description of the CLB function.
- The COUT0 and COUT equations performed by the carry logic.
- Default equations for the F and G function generators.
- Default assignments for the F4, G2, and G3 inputs.

The default F and G functions and default F4, G2, and G3 inputs are based on the generic CLB function described. You can change these defaults as required, allowing for features such as parallel enable or synchronous reset. However, if these defaults are changed, the CLB may no longer function as the mnemonic describes.

The COUT0 and COUT equations are absolutely determined by the carry mode configuration mnemonic. The only way to change these carry logic outputs is by selecting a different mnemonic.

ADD-F-CI

The ADD-F-CI configuration performs a 1-bit addition of A+B in the F function generator, with the A and B inputs on the F1 and F2 pins. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. This configuration can be used as the MSB of an adder, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.

F=(F1@F2)@F4

COUT0=(F1*F2) + CIN*(F1+F2)

G=

COUT=COUT0

F4=CIN

G2=G2I (COUT0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)

G3=G3I (CIN for overflow, OFL=G2@G3)

ADD-FG-CI

The ADD-FG-CI configuration performs a 2-bit addition of A+B in both the F and G function generators, with the lower-order A and B inputs on the F1 and F2 pins, and the higher-order A and B inputs on the G1 and G4 pins. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of an adder.

F=(F1@F2)@F4

COUT0=(F1*F2) + CIN*(F1+F2)

G=(G4@G1)@G2

COUT=(G4*G1) + COUT0*(G4+G1)

F4=CIN

G2=COUT0

G3=G3I

ADD-G-F1

The ADD-G-F1 configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an adder, where the carry-in signal is routed to F1. The F function generator is not used.

F=

COUT0=F1

G=(G4@G1)@G2

COUT=(G4*G1) + COUT0*(G4+G1)

F4=F4I

G2=COUT0

G3=G3I

ADD-G-CI

The ADD-G-CI configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of an adder, where the F function generator is reserved for another purpose.

F=

COUT0=CIN

G=(G4@G1)@G2

COUT=(G4*G1) + COUT0*(G4+G1)

F4=F4I

G2=COUT0

G3=G3I

ADD-G-F3-

The ADD-G-F3- configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an adder, where the inverted carry-in signal is routed to F3. The F function generator is not used.

F=

COUT0=~F3

G=(G4@G1)@G2

COUT=(G4*G1) + COUT0*(G4+G1)

F4=F4I

G2=COUT0

G3=G3I

SUB-F-CI

The SUB-F-CI configuration performs a 1-bit twos-complement subtraction of A-B in the F function generator, with the A input on F1 and the B input on F2. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. This configuration can be used as the MSB of a subtracter, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.

F=(F1@F2)@~F4=~(F1@F2@F4)

COUT0=(F1*~F2) + CIN*(F1+~F2)

G=

COUT=COUT0

F4=CIN

G2=G2I (COUT0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)

G3=G3I (CIN for overflow, OFL=G2@G3)

SUB-FG-CI

The SUB-FG-CI configuration performs a 2-bit twos-complement subtraction of A-B in both the F and G function generators. For the lower bit, the A input is on F1 and the B input is on F2. For the upper bit, the A input is on G4 and the B input is on G1. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of a subtracter.

F=(F1@F2)@~F4=~(F1@F2@F4)

COUT0=(F1*~F2) + CIN*(F1+~F2)

G=(G4@G1)@~G2=~(G4@G1@G2)

COUT=(G4*~G1) +COUT0*(G4+~G1)

F4=CIN

G2=COUT0

G3=G3I

SUB-G-1

The SUB-G-1 configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry-in is tied High (no borrow). The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of a subtracter with no carry-in. The F function generator is not used.

F=

COUT0=1

G=(G4@G1)

COUT=(G4+~G1)

F4=F4I

G2=G2I

G3=G3I

SUB-G-CI

The SUB-G-CI configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of a subtracter, where the F function generator is reserved for another purpose.

F=

COUT0=CIN

G=(G4@G1)@~G2=~(G4@G1@G2)

COUT=(G4*~G1) + COUT0*(G4+~G1)

F4=F4I

G2=COUT0

G3=G3I

SUB-G-F1

The SUB-G-F1 configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a subtracter, where the carry-in signal is routed to F1. The F function generator is not used.

F=

COUT0=F1

G=(G4@G1)@~G2=~(G4@G1@G2)

COUT=(G4*~G1) + COUT0*(G4+~G1)

F4=F4I

G2=COUT0

G3=G3I

SUB-G-F3-

The SUB-G-F3- configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a subtracter, where the inverted carry-in signal is routed to F3. The F function generator is not used.

F=

COUT0=~F3

G=(G4@G1)@~G2=~(G4@G1@G2)

COUT=(G4*~G1) + COUT0*(G4+~G1)

F4=F4I

G2=COUT0

G3=G3I

ADDSUB-F-CI

The ADDSUB-F-CI configuration performs a 1-bit twos-complement add/subtract of A+B in the F function generator, with the A input on F1 and the B input on F2. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The F3 input indicates add (F3=1) or subtract (F3=0). This configuration can be used as the MSB of an adder/subtracter, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.

F=(F1@F2)@F4@~F3=~(F1@F2@F4@F3)

COUT0=F3*((F1*F2) + CIN*(F1+F2)) + ~F3*((F1*~F2) + CIN*(F1+~F2))

G=

COUT=COUT0

F4=CIN

G2=G2I (COUT0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)

G3=G3I (CIN for overflow, OFL=G2@G3)

ADDSUB-FG-CI

The ADDSUB-FG-CI configuration performs a 2-bit twos- complement add/subtract of A+B in both the F and G function generators. For the lower bit, the A input is on F1 and the B input is on F2. For the upper bit, the A input is on G4 and the B input is on G1. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/subtract control signal must be routed to both the F3 and G3 pins. This configuration comprises the middle bits of an adder/subtracter.

F=(F1@F2)@F4@~F3=~(F1@F2@F4@F3)

COUT0=F3*((F1*F2) + CIN*(F1+F2)) + ~F3*((F1*~F2) + CIN*(F1+~F2))

G=(G4@G1)@G2@~G3=~(G4@G1@G2@G3)

COUT=F3*((G4*G1)+COUT0*(G4+G1))+~F3*((G4*~G1)+COUT0*(G4+~G1))

F4=CIN

G2=COUT0

G3=G3I

ADDSUB-G-CI

The ADDSUB-G-CI configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/subtract control signal must be routed to both the F3 and G3 pins. This

configuration is for the middle bit of an adder / subtracter, where the F function generator is reserved for another purpose.

F=

COUT0=CIN

G=(G4@G1)@G2@~G3=~(G4@G1@G2@G3)

COUT=F3*((G4*G1)+COUT0*(G4+G1))+~F3*((G4*~G1)+COUT0*(G4+~G1))

F4=F4I

G2=COUT0

G3=G3I

ADDSUB-G-F1

The ADDSUB-G-F1 configuration performs a 1-bit twos-complement add / subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add / subtract control signal must be routed to both the F3 and G3 pins. This configuration comprises the LSB of an adder / subtracter, where the carry-in signal is routed to F1. The F function generator is not used.

F=

COUT0=F1

G=(G4@G1)@G2@~G3=~(G4@G1@G2@G3)

COUT=F3*((G4*G1)+COUT0*(G4+G1))+~F3*((G4*~G1)+COUT0*(G4+~G1))

F4=F4I

G2=COUT0

G3=G3I

ADDSUB-G-F3-

The ADDSUB-G-F3- configuration performs a 1-bit twos-complement add / subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. Because the F3 input also indicates add (F3=1) or subtract (F3=0), the carry-in is always null (0 for add, 1 for subtract). This configuration comprises the LSB of an adder / subtracter with no carry-in. The F function generator is not used.

F=

COUT0=~F3

G=(G4@G1)

COUT=F3*G4*G1 + ~F3(G4+~G1)

F4=F4I

G2=COUT0

G3=G3I

INC-F-CI

The INC-F-CI configuration performs a 1-bit increment in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The G function generator can be used to output the terminal count of a counter.

F=(F1@F4)

COUT0=CIN*F1

G=

COUT=COUT0

F4=CIN

G2=G2I (COUT0 for terminal count, TC=G2)

G3=G31

INC-FG-1

The INC-FG-1 configuration performs a 2-bit increment in both the F and G function generator, with the lower-order A input on the F1 pin and the higher-order A input on the G4 pin. The carry-in is tied High. The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of an incrementer that is always enabled.

F=~(F1)

COUT0=F1

G=G2@G4

COUT=COUT0*G4

F4=F4I or CIN

G2=COUT0

G3=G3I or CIN

INC-FG-CI

The INC-FG-CI configuration performs a 2-bit increment in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of an incrementer.

F=(F1@F4)

COUT0=CIN*F1

G=(G4@G2)

COUT=COUT0*G4

F4=CIN

G2=COUT0

G3=G3I

INC-G-1

The INC-G-1 configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry-in is tied High. The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of an incrementer that is always enabled. The F function generator is not used. This configuration is identical to DEC-G-0, since the LSB of an incrementer is identical to the LSB of a decrementer.

F=

COUT0=0

G=~(G4)

COUT=G4

F4=F4I

G2=G2I

G3=G3I

INC-G-F1

The INC-G-F1 configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer where F1 is an active-High enable. The F function generator is not used.

F=

COUT0=F1

G=(G4@G2)

COUT=COUT0*G4

F4=F4I

G2=COUT0

G3=G3I

INC-G-CI

The INC-G-CI configuration does a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of an incrementer where the F function generator is reserved for another purpose.

F=

COUT0=CIN

G=(G4@G2)

COUT=COUT0*G4

F4=F4I

G2=COUT0

G3=G3I

INC-G-F3-

The INC-G-F3- configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the F3 pin, is inverted in the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer where F3 is an active-Low enable. The F function generator is not used.

F=

COUT0=~F3

G=(G4@G2)

COUT=COUT0*G4=~F3*G4

F4=F4I

G2=COUT0

G3=G3I

DEC-F-CI

The DEC-F-CI configuration performs a 1-bit decrement in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The G function generator can be used to output the terminal count of a counter.

F=~(F1@F4)

COUT0=F1+CIN*~F1

G=

COUT=COUT0

F4=CIN

G2=G2I (COUT0 for terminal count, TC=G2)

G3=G3I

DEC-FG-0

The DEC-FG-0 configuration performs a 2-bit decrement in both the F and G function generator, with the lower-order input on the F1 pin and the higher order input on the G4 pin. The carry-in is tied Low. The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of a decrementer that is always enabled.

F=~(F1)

COUT0=F1

G=~(G4@G2)

COUT=COUT=(COUT0*~G4) + G4

F4=F4I

G2=COUT0

G3=G3I

DEC-FG-CI

The DEC-FG-CI configuration performs a 2-bit decrement in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of a decrementer.

F=~(F1@F4)

COUT0=F1+CIN*~F1

G=~(G4@G2)

COUT=G4+COUT0*~G4

F4=CIN

G2=COUT0

G3=G3I

DEC-G-0

The DEC-G-0 configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry-in is tied High (no borrow). The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of a decrementer that is always enabled. The F function generator is not used. This configuration is identical to INC-G-1, since the LSB of an incrementer is identical to the LSB of a decrementer.

F=

COUT0=0

G=~(G4)

COUT=G4

F4=F4I

G2=G2I

G3=G3I

DEC-G-CI

The DEC-G-CI configuration does a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of a decrementer, where the F function generator is reserved for another purpose.

F=

COUT0=CIN

G=~(G4@G2)

COUT=G4+COUT0*~G4

F4=F4I

G2=COUT0

G3=G3I

DEC-G-F1

The DEC-G-F1 configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a decrementer where F1 is an active-Low enable. The F function generator is not used.

F=

COUT0=F1

G=~(G4@G2)

COUT=COUT0 + G4

F4=F4I

G2=COUT0

G3=G3I

DEC-G-F3-

The DEC-G-F3- configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F3 pin, is inverted in the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a decrementer, where F3 is an active-High enable. The F function generator is not used.

F=

COUT0=~F3

G=~(G4@G2)

COUT=COUT0 + G4

F4=F4I

G2=COUT0

G3=G3I

INCDEC-F-CI

The INCDEC-F-CI configuration performs a 1-bit increment / decrement in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The F3 input indicates increment (F3=1) or decrement (F3=0). The G function generator can be used to output the terminal count of a counter.

F=(F1@F4)@~F3

COUT0=~F3*(F1+ CIN) + F3*F1*CIN

G=

COUT=COUT0

F4=CIN

G2=G2I (COUT0 for terminal count, TC=G2)

G3=G31

INCDEC-FG-1

The INCDEC-FG-1 configuration performs a 2-bit increment / decrement in both the F and G function generator, with the lower- order input on the F1 pin and the higher-order input on the G4 pin. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment / decrement control signal must be routed to both the F3 and G3 pins. The carry-in is always active (High in increment mode and Low in decrement mode). The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of an incrementer / decrementer that is always enabled.

F=~(F1)

COUT0=F1

G=(G2@G4)@~G3

COUT=COUT=~F3*((COUT0*~G4)+G4) + F3*(G4*COUT0)

F4=F4I

G2=COUT0

G3=G3I

INCDEC-FG-CI

The INCDEC-FG-CI configuration performs a 2-bit increment / decrement in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment / decrement control signal must be routed to both the F3 and G3 pins. This configuration comprises the middle bits of an incrementer / decrementer.

F=(F1@F4)@~F3

COUT0=~F3*(F1+ CIN) + F3*F1*CIN

G=(G4@G2)@~G3

COUT=~F3*(G4+ COUT0) + F3*G4*COUT0

F4=CIN

G2=COUT0

G3=G3I

INCDEC-G-0

The INCDEC-G-0 configuration performs a 1-bit increment / decrement in the G function generator, with the input on the G4 pin. The carry-in is tied High. The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of an incrementer / decrementer that is always enabled. The F function generator is not used. F3 is not required for increment / decrement control, since the LSB of an incrementer is identical to the LSB of a decrementer; this configuration is identical to INC-G-1 and DEC-G-0.

F=

COUT0=0

G=~(G4)

COUT=G4

F4=F4I

G2=G2I

G3=G3I

INCDEC-G-CI

The INCDEC-G-CI configuration performs a 1-bit increment / decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment / decrement control signal must be routed to both the F3 and G3 pins. This configuration is for the middle bit of an incrementer / decrementer, where the F function generator is reserved for another purpose, although the F3 pin is used by the carry logic.

F=

COUT0=CIN

G=(G4@G2)@~G3

COUT=~F3*(G4+ COUT0) + F3*G4*COUT0

F4=F4I

G2=COUT0

G3=G3I

INCDEC-G-F1

The INCDEC-G-F1 configuration performs a 1-bit increment / decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer / decrementer where the carry-in signal is routed to F1. The carry-in is active-High for an increment operation and active-Low for a decrement operation. The F function generator is not used. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment / decrement control signal must be routed to both the F3 and G3 pins.

F=

COUT0=F1

G=(G4@G2)@~G3

COUT=F3*(G4*COUT0) + ~F3*(G4+COUT0)

F4=F4I

G2=COUT0

G3=G3I

FORCE-0

The FORCE-0 configuration forces the carry-out signal on the COUT pin to be 0.

COUT0=0

COUT=0

FORCE-1

The FORCE-1 configuration forces the carry-out signal on the COUT pin to be 1.

COUT0=1

COUT=1

FORCE-CI

The FORCE-CI configuration forces the signal on the CIN pin to pass through to the COUT pin.

COUT0=CIN

COUT=COUT0=CIN

FORCE-F1

The FORCE-F1 configuration forces the signal on the F1 pin to pass through to the COUT pin.

COUT0=F1

COUT=COUT0=F1

FORCE-F3-

The FORCE-F3- configuration forces the signal on the F3 pin to pass inverted to the COUT pin.

COUT0=~F3

COUT=COUT0=~F3

FORCE-G4

The FORCE-G4 configuration forces the signal on the G4 pin to pass through to the COUT pin (XC4000X and SpartanXL only).

COUT0=0

COUT=G4

EXAMINE-CI

The EXAMINE-CI configuration allows the carry signal on the CIN pin to be used in the F or G function generators. This configuration forces the signal on the CIN pin to pass through to the COUT pin and is equivalent to the FORCE-CI configuration. EXAMINE-CI is provided for CLBs in which the carry logic is unused but the CIN signal is required.

COUT0=CIN

COUT=COUT0=CIN

Carry Logic in XC5200

The XC5200 CLB contains a dedicated carry logic feature. This enhances the performance of arithmetic functions such as adders, subtracters, counters, comparators, and so forth. A carry multiplexer (CY_MUX) represents the dedicated 2:1 multiplexer in each logic cell. The multiplexer performs a 1-bit high speed carry propagate per logic cell (four bits per CLB).

In addition to providing a high speed carry propagate function, each CY_MUX can be connected to the CY_MUX in the adjacent logic cell to provide cascadable decode logic. The "XC5200 Carry Logic" figure illustrates how the four-input function generators can be configured to take advantage of the four cascaded CY_MUXes.

Note: AND and OR cascading are specific cases of a generic decode.

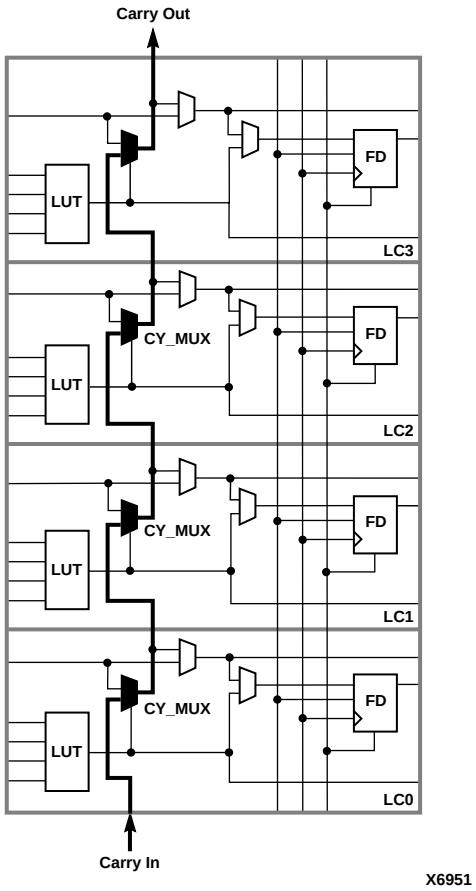


Figure 12-20 XC5200 Carry Logic

XC5200 Carry Logic Library Support

The design entry library contains one carry logic primitive and one carry logic macro. The carry multiplexer primitive (CY_MUX) represents the dedicated 2:1 multiplexer that performs the high speed carry propagate function. The carry initialize (CY_INIT) macro is used to initialize the carry chain for all arithmetic functions. The CY_INIT is implemented by forcing a zero onto the select line of the CY_MUX such that the DI pin of the CY_MUX is selected to drive the CO pin. See the "Carry Initialize Function XC5200" figure.

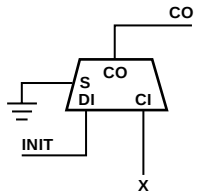


Figure 12-21 Carry Initialize Function XC5200

Note: The XC5200 library contains a set of RPMs designed to take advantage of the logic. Using the macros as they are or modifying them makes it much easier to take advantage of this feature.

Cascade Function

Each CY_MUX can be connected to the CY_MUX in the adjacent logic cell to provide cascadable decode logic. The “CY_MUX Used for Decoder Cascade Logic XC5200” figure illustrates how the 4-input function generators can be configured to take advantage of these four cascaded CY_MUXes.

Note: AND and OR cascading are specific cases of a general decode. In AND cascading, all bits are decoded equal to logic one. In OR cascading, all bits are decoded equal to logic zero. The flexibility of the LUT achieves this result.

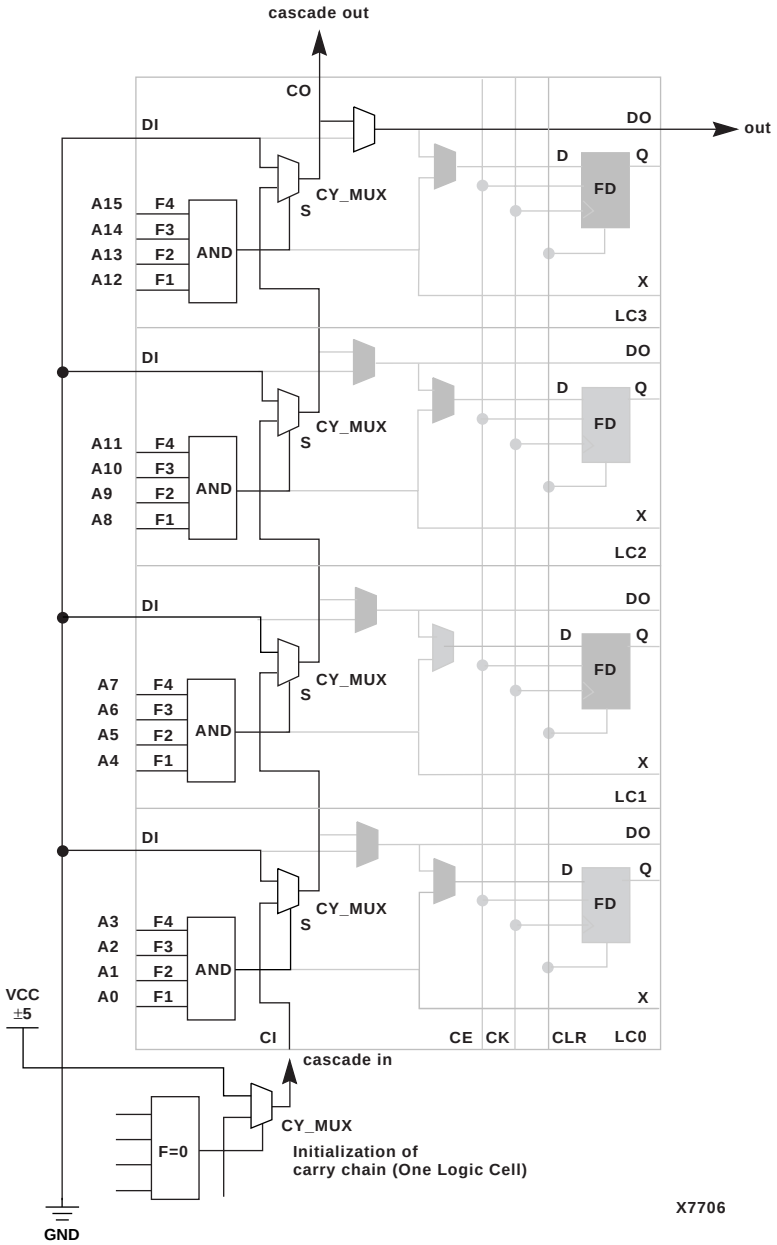


Figure 12-22 CY_MUX Used for Decoder Cascade Logic XC5200

Carry Logic in Virtex, Spartan2

A Virtex or Spartan2 CLB contains a dedicated carry logic feature. This enhances the performance of arithmetic functions such as adders, subtracters, counters, comparators, and so forth. For detailed information on Carry Logic in Virtex and Spartan2, refer to the Xilinx web site, http://support.xilinx.com.