TMS320C54x Application Code Examples

C5000 Applications Team

Digital Signal Processing Solutions

ABSTRACT

This application report contains complete code examples for routines that have been developed using a TMS320C54x™ EVM platform. These programs demonstrate applications that use a host interface and run in real time.

Contents

List of Tables

List of Examples

1 Running the Applications

The host communicates to the ’54x EVM through 16-bit I/O locations. Each I/O location is defined by an offset to an I/O page 0 address. The offset used for these applications is 0x240 + 0x800 for channel A, 0x240 + 0x804 for channel B, and 0x240 + 0x808 for the target/status control register, where 0x240 is the base address of the host. Check your PC system documentation to make sure that I/O space 0x240 does not conflict with other I/O devices. If the EVM is mapped to other than space 0x240, the base addresses of the control and status registers must be modified in the file host.h.

The ’54x assembler assembles code and generates object files. The linker command file links the object files and generates a file named main.out, using common object file format (COFF). You must load main.out into the EVM debugger command window with the LOAD command and compile the host software using a Borland C compiler. This generates the file, master.exe. that contains graphic routines that display data transferred from the target to the host.

To run the target application, load main.out into the EVM debugger. To start the program on the target side, press the F5 function key. If you halt the program with the escape key or use the halt command in the debugger window, the program remains in the handshake loop waiting for the host to send a command. Press the F5 function key to continue. To run the host application, execute master.exe at the DOS prompt or a window command line. When master.exe is executed by the host, it displays the message:

```
Graphics: No error. Press any key to halt.
```

When you press this key, the graphics window opens and displays data for the task the target has initiated. The default task is an oscilloscope routine. To change to a different task, go to the debugger window, halt the program, and in the command window, type:

```
e *present_command = x
```

where x = 1, 2, 3, 4, 5 or 6, and present_command has one of the following values:

1 = oscilloscope
2 = Low-pass finite impulse response (FIR) filter using MAC instruction
3 = Low-pass infinite impulse response (IIR) filter using biquad sections
4 = Lowpass FIR filtering using FIRS instruction
5 = System identification using adaptive filtering with least mean squares (LMS) instruction
6 = 256-point real fast Fourier transform (FFT)

You can view the output of the present task in the graphics window.

To exit the host application, press F3. Communication is lost if at any time the target code is reloaded or reset in the command window while the host executable is running in the background. This means that if you attempt to reset or reload the code in the debugger window and you press F5, the computer locks up. This occurs because there is no handshake between the host and the target. To unlock, reload and run the code (press F5) on the target side. On the host side, quit the window and rerun the executable.

The adaptive filter can be tested in two steps. The initial step size d_mu = 0 in the first step. If the present task is changed at the debugger window with the command e *present_command = 5, runs with d_mu = 0. Thus, the system is not identified since the coefficients of the adaptive filter are not updated. In the second step the step size can be changed by typing e *d_mu = 0x1000 at the command window. In this case, the system is identified and the filter coefficients are adapted using the LMS algorithm. In both cases, the error signal can be observed both on host and also from the output of the ’AC01.

2 Application Code

Table 1 lists programs appropriate for running on a target system and tells you where to look for them in this chapter.

Table 1. Target Files

Title	File Name	Page
Vector Table Initialization	vectors.asm	4
Main Program That Calls Different Functions	main.asm	13
Memory Allocation for Entire Application	memory.asm	7
Processor Initialization	init_54x.asm	18
Initialization of Variables, Pointers, and Buffers	prcs_int.asm	23
Initialization of Serial Port 1	init_ser.asm	27
’AC01 Initialization	init_aic.asm	31
’AC01 Register Configuration	aic_cfg.asm	35
Receive Interrupt Service Routine	rcv_int1.asm	38
Task Scheduling	task.asm	42
Echo the Input Signal	echo.asm	47
Low-Pass FIR Filtering Using MAC Instruction	fir.asm	49
Low-Pass Biquad IIR Filter	iir.asm	58
Low-Pass Symmetric FIR Filtering Using FIRS Instruction	sym_fir.asm	54
Adaptive Filtering Using LMS Instruction	adapt.asm	63
256-Point Real FFT Initialization	rfft.asm	71
Bit Reversal Routine	bit_rev.asm	74
256-Point Real FFT Routine	fft.asm	77
Unpack 256-Point Real FFT Output	unpack.asm	82
Compute the Power Spectrum of the Complex Output of the 256-Point Real FFT	power.asm	87

Table 2 lists programs appropriate for running on a host system and tells you where to look for them in this chapter.

Table 2. Communication Interface Files

Title	File Name	Page
Handshake Between Host and Target	hand_shk.asm	20
Interrupt 1 Service Routine	hst_int1.asm	93
Data Transfer from FIFO	fifo.asm	89
Main Function Call on Host Side	master.c	98
Function Calls on Host Side	host.c	97
Display the Data on the Screen	view2.c	102
Graphic Drivers Routine	graphic2.c	100

Example 28 on page 102 shows the linker command that links all object files together to produce a single executable COFF object module. This file establishes the memory configuration for the entire application, using the ’541’s memory map. Example 29 on page 104 shows the configuration of the memory map for a ’541 device used by an EVM debugger.

Example 1. Vector Table Initialization

```
;  TEXAS INSTRUMENTS INCORPORATED
;     DSP Data Communication System Development / ASP
;
;  Archives:   PVCS
;  Filename:   vectors.asm
;  Version:    1.0
;  Status:     draft              ( )
;                proposal         (X)
;                accepted         ( )        dd–mm–yy/?acceptor.
;
;     AUTHOR     Padma P. Mallela
;
;                Application Specific Products
;                Data Communication System Development
;                12203 SW Freeway, MS 701
;                Stafford, TX 77477;{
;                IPR statements description (can be collected).
;}
;    (C)         Copyright 1996. Texas Instruments.
;                All rights reserved.
;
;{
;                Change history:
;
;     VERSION    DATE    /         AUTHORS             COMMENT
;      1.0       July–24–96  /     P.Mallela           original created
;
;}
;{
```

Example 1. Vector Table Initialization (Continued)

```
;     1. ABSTRACT
;
;    1.1 Function Type
;        a. Core Routine
;        b. Subroutine
;
;    1.2 Functional Description
;        This file contains vector table of 541
;
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:   TMS320C54X ASSEMBLER
;        Version:    1.02 (PC)
;        Activation: asm500 –s vectors.asm
;
;    1.6 Notes and Special Considerations
;{
;            This code is written for 541 device. The code is tested on
;            C54x EVM
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;          –
;    2.2 Local Compiler Flags
;          –
;    2.3 Local Constants
;          –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include   ”init_54x.inc”
        .include   ”main.inc”
;    3.2 External Data
        .ref       SYSTEM_STACK
;    3.3 Import Functions
        .ref       main_start
        .ref       receive_int1
        .ref       host_command_int1
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;          –
;    4.2 Global Static Data
;          –
;    4.3 Dynamic Data
;          –
;    4.4 Temporary Data
;          –
```

Example 1. Vector Table Initialization (Continued)

```
;    4.5 Export Functions
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;===========================================================================
;    5.1 reset
;
;    5.2 Functional Description
;        This function initializes the vector table of 541 device
;---------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;
;        Reentrancy:  No
;        Recursive:   No
;
;    5.4 Inputs
;
;    5.5 Outputs
;
;    5.6 Global
;
;    5.7 Special considerations for data structure
;            –
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;in  |0 | 0 | 1 | 0 | 0  | 0 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;out |0 | 0 | 1 | 0 | 0  | 0 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;               Execution time: ?cycles
;               Call rate:  not applicable for this application
;
;===========================================================================
;HeaderEnd
;    5.10 Code
                .sect            ”vectors”
reset:          BD main_start                                ;RESET vector
                STM          #SYSTEM_STACK,SP
nmi:            RETE
                NOP
                NOP
                NOP                                          ;NMI
; software interrupts
sint17                  .space  4*16
sint18                  .space  4*16
sint19                  .space  4*16
sint20                  .space  4*16
sint21                  .space  4*16
sint22                  .space  4*16
sint23                  .space  4*16
sint24                  .space  4*16
sint25                  .space  4*16
sint26                  .space  4*16
```

Example 1. Vector Table Initialization (Continued)

```
sint27                  .space  4*16
sint28                  .space  4*16
sint29                  .space  4*16
sint30                  .space  4*16
int0:       RETE
            NOP
            NOP                                 ; INT0
            NOP
int1:       BD          host_command_int1       ; Host interrupt
            PSHM        ST0
            PSHM        ST1                     ; INT1
int2:       RETE
            NOP
            NOP
            NOP
tint:       RETE
            NOP
            NOP                                  ; TIMER
            NOP
rint0:      RETE                                 ; Serial Port Receive
            NOP                                  ; Interrupt 0
            NOP
            NOP
xint0:      RETE                                 ; Serial Port Transmit
            NOP                                  ; Interrupt 0
            NOP
            NOP
rint1:      BD           receive_int1            ; Serial Port Receive
            PSHM         ST0                     ;Interrupt 1
            PSHM         ST1
xint1:      RETE                                 ; Serial Port Transmit
            NOP                                  ; Interrupt 1
            NOP
            NOP
int3:       RETE
            NOP
            NOP                                  ;INT3
            NOP
           .end
*-------------------------------------------------------------------------------
```

Example 2. Memory Allocation for Entire Application

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:  PVCS
; Filename:  memory.asm
; Version:   1.0
; Status :   draft      ( )
;            proposa    (X)
;            accepted   ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
```

Example 2. Memory Allocation for Entire Application (Continued)

```
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;            VERSION   DATE     /    AUTHORS            COMMENT
;             1.0   July–24–96 /    P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains main function
;
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s memory.asm
;
;    1.6 Notes and Special Considerations
;{
; This code is written for 541 device. The code is tested on C54x EVM
;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;           –
;    2.2 Local Compiler Flags
;           –
;    2.3 Local Constants
;           –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
             .mmregs
             .include ”defines.inc”
             .include ”main.inc”
;    3.2 External Data
;
;    3.3 Import Functions
;}
;{
```

Example 2. Memory Allocation for Entire Application (Continued)

```
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;           –
;    4.2 Global Static Data
;           –
;    4.3 Dynamic Data
;           –
;    4.4 Temporary Data
;           –
;    4.5 Export Functions
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 main_start
;
;    5.2 Functional Description
;        Memory configuration of the application
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;
;           Reentrancy:            No
;           Recursive :            No
;
;    5.4 Inputs
;    5.5 Outputs
;
;    5.6 Global
;
;    5.7 Special considerations for data structure
;
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;in  |U | 1 |NU |NU |NU  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU |NU|NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;out |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;              Execution time: ?cycles
;              Call rate:    not applicable for this application
;
;============================================================================
;HeaderEnd
;    5.10 Code
STACK          .usect   ”stack”,K_STACK_SIZE
SYSTEM_STACK   .set     K_STACK_SIZE+STACK

input_data     .usect   ”inpt_buf”,K_FRAME_SIZE*2 ; input data array
output_data    .usect   ”outdata”,K_FRAME_SIZE*2  ; output data array
; this section of variables are used in receive_int1 routine and related routines
RCV_INT1_DP    .usect   ”rcv_vars”,0
```

Example 2. Memory Allocation for Entire Application (Continued)

```
d_rcv_in_ptr   .usect   ”rcv_vars”,1              ; save/restore input bffr ptr
d_xmt_out_ptr  .usect   ”rcv_vars”,1              ; save/restore output bffr ptr
d_frame_flag   .usect   ”rcv_vars”,1
d_index_count  .usect   ”rcv_vars”,1
; System Coefficients
scoff          .sect    ”coeffh”
               .include ”impulse.h”
; RAM location for the System coefficient
hcoff          .usect   ”bufferh”, H_FILT_SIZE
wcoff          .usect   ”bufferw”, ADPT_FILT_SIZE ;
; RAM location for the input data
xh             .usect   ”bufferx”, H_FILT_SIZE    ; input data for system
xw             .usect   ”bufferp”, ADPT_FILT_SIZE ; input data for adaptive filter
; RAM location for filter outputs, residual error
; and temporary location for the new input sample.
ADAPT_DP       .usect   ”adpt_var”,0
d_primary      .usect   ”adpt_var”,1
d_output       .usect   ”adpt_var”,1
d_error        .usect   ”adpt_var”,1
d_mu           .usect   ”adpt_var”,1
d_mu_e         .usect   ”adpt_var”,1
d_new_x        .usect   ”adpt_var”,1
d_adapt_count  .usect   ”adpt_var”,1
                                          ; the 16 tap FIR coefficients
COFF_FIR_START .sect    ”coff_fir”                ; filter coefficients
   .word 6Fh
   .word 0F3h
   .word 269h
   .word 50Dh
   .word 8A9h
   .word 0C99h
   .word 0FF8h
   .word 11EBh
   .word 11EBh
   .word 0FF8h
   .word 0C99h
   .word 8A9h
   .word 50Dh
   .word 269h
   .word 0F3h
   .word 6Fh
COFF_FIR_END
; circular buffers for coefficients and data buffers
fir_coff_table  .usect    ”fir_coff”, 20
d_data_buffer   .usect    ”fir_bfr”, 40
; variables used in FIR routine
FIR_DP          .usect    ”fir_vars”,0
d_filin         .usect    ”fir_vars”,1
d_filout        .usect    ”fir_vars”,1
; variables used in IIR routine
IIR_DP          .usect    ”iir_vars”,0
d_iir_d        . usect    ”iir_vars”,3*2
d_iir_y         .usect    ”iir_vars”,1
                .sect     ”iir_coff”
iir_table_start
*
* second–order section # 01
*
   .word –26778                  ;A2
   .word 29529                   ;A1/2
```

Example 2. Memory Allocation for Entire Application (Continued)

```
   .word 19381                  ;B2
   .word –23184                 ;B1
   .word –19381                 ;B0
*
* second–order section # 02
*
   .word –30497                 ;A2
   .word 31131                  ;A1/2
   .word 11363                  ;B2
   .word –20735                 ;B1
   .word 11363                  ;B0
iir_table_end
iir_coff_table .usect   ”coff_iir”,16
; symmetric FIR filter coeffs
FIR_COFF       .sect      ”sym_fir”              ; filter coefficients
   .word 6Fh
   .word 0F3h
   .word 269h
   .word 50Dh
   .word 8A9h
   .word 0C99h
   .word 0FF8h
   .word 11EBh
; circular buffers used in symmetric filter routine
d_datax_buffer   .usect  ”cir_bfr”,20
d_datay_buffer   .usect  ”cir_bfr1”,20
   .include ”ref_tsk.inc”
task_list        .sect   ”task_tbl”             ; calls the tasks itself
   .word do_nothing
   .word echo_task               ; Echo routine
   .word fir_task                ; FIR routine
   .word iir_task
   .word sym_fir_task
   .word adapt_task
   .word rfft_task
task_init_list   .sect   ”task_int”             ; has the initialization of tasks
   .word do_nothing
   .word no_echo_init_task       ; there is no init in this case
   .word fir_init
   .word iir_init
   .word sym_fir_init
   .word adapt_init
   .word do_nothing
; variables used in task handling routine
TASK_VAR_DP      .usect  ”tsk_vars”,0
present_command  .usect  ”tsk_vars”,1
last_command     .usect  ”tsk_vars”,1
d_task_addr      .usect  ”tsk_vars”,1
d_task_init_addr .usect  ”tsk_vars”,1
d_buffer_count   .usect  ”tsk_vars”,1
d_output_addr    .usect  ”tsk_vars”,1
d_input_addr     .usect  ”tsk_vars”,1
; Set start addresses of buffers
fft_data         .usect  ”fft_bffr”, 4*K_FFT_SIZE  ; fft data processing buffer
; Copy twiddle tables
                 .sect   ”sin_tbl”
sine_table       .copy   twiddle1                  ; sine table
sine             .usect  ”twid_sin”,K_FFT_SIZE
                 .sect   ”cos_tbl”
cos_table  .copy   twiddle2                        ; cosine table
```

Example 2. Memory Allocation for Entire Application (Continued)

```
cosine          .usect  ”twid_cos”,K_FFT_SIZE
; Define variables for indexing input data and twiddle tables
FFT_DP          .usect  ”fft_vars”,0
d_grps_cnt      .usect  ”fft_vars”,1            ; (# groups in current stage)–1
d_twid_idx      .usect  ”fft_vars”,1
; index of twiddle tables
d_data_idx      .usect  ”fft_vars”,1            ; index of input data table
;variables used for host interface
FIFO_DP         .usect  ”fifo_var”,0
d_command_reg   .usect  ”fifo_var”,1
d_command_value .usect  ”fifo_var”,1
d_fifo_count    .usect  ”fifo_var”,1
d_fifo_ptr      .usect  ”fifo_var”,1
    .end
; Filename : defines.inc
; this include file defines all the variables, buffers and pointers used for the entire
; application
.def        STACK,SYSTEM_STACK
.def        input_data,output_data
.def        scoff,hcoff,wcoff,xh,xw
.def        ADAPT_DP,d_primary,d_output,d_error
.def        d_mu,d_mue,d_new_x,d_adapt_count
.def        fir_coff_table,d_data_buffer
.def        FIR_DP,d_filin,d_filout
.def        COFF_FIR_START,COFF_FIR_END
.def        IIR_DP,d_iir_d,iir_y
.def        iir_coff_table
.def        COFF_FIR_START,COFF_FIR_END
.def.       d_datax_buffer,d_datay_buffer
.def        FIR_COFF
.def        TASK_VAR_DP,present_command,last_command
.def        d_task_addr,d_task_init_addr,d_buffer_count,d_output_addr
.def        RCV_INTL_DP
.def        d_rcv_in_ptr,d_xmt_out_ptr
.def        d_frame_flag,d_index_count
.def        fft_data,sine, cosine
.def        FFT_DP,d_grps_cnt, d_twid_idx, d_data_idx
.def        cos_table,sine_table
.def        FIFO_DP
.def        d_command_reg
.def        d_fifo_count
.def        d_fifo_ptr
; Filename:   ref_tsk.inc
; this includes all the task scheduling table referenced labels
.ref        do_nothing,echo_task,fir_task,iir_task
.ref        sym_fir_task,fir_init,iir_init,sym_fir_init
.ref        no_echo_init_task,fir_init,iir_init,sym_fir_init
.ref        adapt_init
.def        task_init_list,task_list
```

Example 3. Main Program That Calls Different Functions

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    main.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR       Padma P. Mallela
;
;              Application Specific Products
;              Data Communication System Development
;              12203 SW Freeway, MS 701
;              Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /     AUTHORS           COMMENT
;          1.0     July-24-96  /    P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1  Function Type
;         a.Core Routine
;         b.Subroutine
;
;    1.2  Functional Description
;         This file contains main function
;
;
;    1.3  Specification/Design Reference (optional)
;
;    1.4  Module Test Document Reference
;         Not done
;
;    1.5  Compilation Information
;         Compiler:       TMS320C54X  ASSEMBLER
;         Version:        1.02 (PC)
;         Activation:     asm500 -s main.asm
;
;    1.6 Notes and Special Considerations
;{
; This code is written for 541 device. The code is tested on C54x EVM
;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
```

Example 3. Main Program That Calls Different Functions (Continued)

```
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include    ”init_54x.inc”
         .include    ”main.inc”
;    3.2 External Data
         .ref        d_frame_flag
         .ref        RCV_INT1_DP
;    3.3 Import Functions
         .ref        aic_init,serial_init,init_54,init_bffr_ptr_var
         .ref        task_handler,evm_handshake,fifo_host_transfer
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def        main_start
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 main_start
;
;    5.2 Functional Description
;        This is the main function that calls other functions.
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;            BD      main_start
;            PSHM    ST0
;            PSHM    ST1
;
;        Reentrancy:    No
;        Recursive :    No
;
;    5.4 Inputs
;
;    5.5 Outputs
;
;    5.6 Global
;
;
```

Example 3. Main Program That Calls Different Functions (Continued)

```
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.10 Code
    .sect ”main_prg”
*******************************************************************************
* The code initializes the 541 device, handshake between Target (DSP)
* and the host (PC). Zeros all buffers, variables and init. pointers
* Initializes serial port, programs AC01 registers for selecting sampling
* rate, gains etc..
 ******************************************************************************
main_start:
    CALL    init_54                     ; initialize ST0,ST1 PMST and
                                        ; other registers
    .if     K_HOST_FLAG = 1
    CALL    evm_handshake               ; EVM host handshake
    .endif
    CALL    init_bffr_ptr_var           ; init tables,vars,bffrs,ptr
    CALL    serial_init                 ; initialize serial_port 1
    CALLD   aic_init                    ; Configures AC01
    LD      #0,DP
    NOP
*******************************************************************************
* After enabling interrupts from the above, the real processing starts here.
* After collecting 256 samples from AC01 a flag(d_frame_flag is set).
* Handles the task initiated by the user and transfers the data to the
* host. Keeps the sequence forever !!!!!
*******************************************************************************
start_loop
    LD      #RCV_INT1_DP,DP             ; restore the DP loop:
    BITF    d_frame_flag,1              ; if 256 samples are received
    BC      loop,NTC                    ; if not just loop back
    CALL    task_handler                ; handles task scheduling
    CALL    fifo_host_transfer          ; EVM HOST interface
    B       loop
    .end
* Includes all the constants – main.inc
K_0                     .set    0               ; constant
K_FIR_INDEX             .set    1               ; index count
K_FIR_BFFR              .set    16              ; FIR buffer size
K_neg1                  .set    –1h             ; index count
K_BIQUAD                .set    2               ; there are 2 bi–quad sections
K_IIR_SIZE              .set    10              ; each bi–quad has 5 coeffs
K_STACK_SIZE            .set    200             ; stack size
K_FRAME_SIZE            .set    256             ; PING/PONG buffer size
```

Example 3. Main Program That Calls Different Functions (Continued)

```
K_FRAME_FLAG            .set    1               ; set after 256 collected
H_FILT_SIZE             .set    128             ; H(z) filter size
ADPT_FILT_SIZE          .set    128             ; W(z) filter size
K_mu                    .set    0h              ; initial step constant
K_HOST_FLAG             .set    1               ; Enable EVM_HOST interface
K_DEFAULT_AC01          .set    1h              ; default AC01 init
*  This include file sets the FFT size for the 'C54x Real FFT code
*  Note that the Real FFT size (i.e. the number of points in the
*  original real input sequence) is 2N; whereas the FFT size is
*  the number of complex points formed by packing the real inputs,
*  which is N.  For example, for a 256-pt Real FFT, K_FFT_SIZE
*  should be set to 128 and K_LOGN should be set to 7.
K_FFT_SIZE              .set    128             ; # of complex points (=N)
K_LOGN                  .set    7               ; # of stages (=logN/log2)
K_ZERO_BK               .set    0
K_TWID_TBL_SIZE         .set    128             ; Twiddle table size
K_DATA_IDX_1            .set    2               ; Data index for Stage 1
K_DATA_IDX_2            .set    4               ; Data index for Stage 2
K_DATA_IDX_3            .set    8               ; Data index for Stage 3
K_FLY_COUNT_3           .set    4               ; Butterfly counter for Stage 3
K_TWID_IDX_3            .set    32              ; Twiddle index for Stage 3
*******************************************************************************
* FILENAME: INIT54x.INC
* This include file contains all the initial values of ST0, ST1, PMST, SWWSR, BSCR
registers
* ST0 Register Organization
*
*  ------------------------------------------------
*  | 15    13 | 12 | 11 | 10 |  9  | 8            0  |
*  |----------|----|----|----|-----|-----------------|
*  |   ARP    | TC | C  | OVA| OVB |       DP        |
*  ------------------------------------------------
*
*******************************************************************************
K_ARP           .set    000b<<13                ; ARP can be addressed from 00b -111b
                                                ; reset value
K_TC            .set    1b<<12                  ; TC = 1 at reset
K_C             .set    1b<<11                  ; C = 1 at reset
K_OVA           .set    1b<<10                  ; OVA = 0 at reset, Set OVA
K_OVB           .set    1b<< 9                  ; OVB = 0 at reset, Set OVB
K_DP            .set    000000000b<<0           ; DP is cleared to 0 at reset
K_ST0           .set    K_ARP|K_TC|K_C|K_OVA|K_OVB|K_DP
*****************************************************************
*ST1 Register Organization
*
*  ---------------------------------------------------------
*  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |  6 |  5 | 4    0 |
*  |---------------------------------------------------------|
*  |BRAF| CPL| XF | HM |INTM|  0 |OVM|SXM|C16|FRCT|CMPT|  ASM   |
*  ---------------------------------------------------------
*
*******************************************************************************
K_BRAF          .set    0b << 15                ; BRAF = 0 at reset
K_CPL           .set    0b << 14                ; CPL = 0 at reset
K_XF            .set    1b << 13                ; XF = 1 at reset
K_HM            .set    0b << 12                ; HM = 0 at reset
K_INTM          .set    1b << 11                ; INTM
K_ST1_RESR      .set    0b << 10                ; reserved
K_OVM           .set    1b << 9                 ; OVM = 0 at reset
K_SXM           .set    1b << 8                 ; SXM = 1 at reset
```

Example 3. Main Program That Calls Different Functions (Continued)

```
K_C16        .set    0b << 07              ; C16 = 0 at reset
K_FRCT       .set    1b << 06              ; FRCT = 0 at reset, Set FRCT
K_CMPT       .set    0b << 05              ; CMPT = 0 at reset
K_ASM        .set    00000b << 00          ; ASM = 0 at reset
K_ST1_HIGH   .set    K_BRAF|K_CPL|K_XF|K_HM|K_INTM|K_ST1_RESR|K_OVM|K_SXM
K_ST1_LOW    .set    K_C16|K_FRCT|K_CMPT|K_ASM
K_ST1        .set    K_ST1_HIGH|K_ST1_LOW
*******************************************************************************
*PMST Register Organization
*  ------------------------------------------------
*  | 15    7 |   6  | 5  | 4  | 3  |   2  | 1      0  |
*  |---------|------|----|----|----|------|-----------
*  |   IPTR  |MP/MC |OVLY|AVIS|DROM|CLKOFF| Reserved  |
*  ------------------------------------------------
K_IPTR       .set   111111111b <<  07       ; 111111111b at reset
K_MP_MC      .set   1b << 06                ; 1 at reset
K_OVLY       .set   Ob << 05                ; OVLY = 0 at reset
K_AVIS       .set   Ob << 04                ; AVIS = 0 at reset
K_DROM       .set   Ob << 03                ; DROM = 0 at reset
K_CLKOFF     .set   Ob << 02                ; CLKOFF = 0 at reset
K-PMST_RESR  .set   OOb << 0                ; reserved
                                            ; for 548  bit 0 = SMUL
                                            ; saturation on multiply
                                            ; bit 1 = SST = saturation on store
K_PMST       .set   K_IPTR|K_MP_MC|K_OVLY|K_AVIS|K_DROM|K_CLKOFF|K_PMST_RESR
*******************************************************************************
*SWWSR Register Organization
*  ----------------------------------------------------
*  |     15    |14      12|11    9|8      6| 5  3   | 2  0  |
*  ----------------------------------------------------
*  |  Reserved |  I/O     |  Data   |  Data |Program|Program |
*  ----------------------------------------------------
*******************************************************************************
K_SWWSR_IO   .set                  2000h                    ; set the I/O space
*******************************************************************************
*Bank Switching Control Register (BSCR)Organization
*  ------------------------------------------
*  |  15    12 |  11    | 10        2 |  1 |  0      |
*  ------------------------------------------
*  |  BNKCMP      |  PS-DS |  Reserved   | BH | EXIO |
*  ------------------------------------------
*******************************************************************************
K_BNKCMP     .set    0000b << 12           ; bank size = 64K
K_PS_DS      .set    0b << 11
K_BSCR_RESR  .set    000000000b <<2        ; reserved space
K_BH         .set    0b << 1               ; BH = 0 at reset
K_EXIO       .set    0b << 0               ; EXIO = 0 at reset
K_BSCR       .set    K_BNKCMP|K_PS_DS|K_BSCR_RESR|K_BH|K_EXIO
```

Example 4. Processor Initialization

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    init_54x.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR       Padma P. Mallela
;
;              Application Specific Products
;              Data Communication System Development
;              12203 SW Freeway, MS 701
;              Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /     AUTHORS           COMMENT
;          1.0     July–29–96  /    P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains initialization of the processor
;
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s init_54x.asm
;
;    1.6 Notes and Special Considerations
;{
;    This code is written for 541 device. The code is tested on C54x EVM
;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
```

Example 4. Processor Initialization (Continued)

```
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include        ”init_54x.inc”
;    3.2 External Data
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def    init_54
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 init_54
;
;    5.2 Functional Description
;        Initializes the processor from a reset state
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;                CALL    init_54
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;
;    5.5 Outputs
;
;    5.6 Global
;
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
```

Example 4. Processor Initialization (Continued)

```
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in  |NU|NU |NU |NU |NU  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out |NU|NU |NU |NU |NU  |NU |NU |NU |NU |NU |UM |UM |NU |NU |NU|NU|NU|NU |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:    not applicable for this application
;
;=============================================================================
;        HeaderEnd
;    5.10 Code
    .sect       ”main_prg”
    init_54:
; Init.the s/w wait state reg.for 2 wait states for I/O operations
    STM          #K_SWWSR_IO, SWWSR           ; 2 wait states for I/O operations
; wait states for Bank Switch
    STM          #K_BSCR, BSCR                ; 0 wait states for BANK SWITCH
; initialize the status and control registers
    STM          #K_ST0, ST0
    STM          #K_ST1, ST1
    RETD
    STM          #K_PMST,PMST
    .end
```

Example 5. Handshake Between Host and Target

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    hand_shk.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR       Padma P. Mallela
;
;              Application Specific Products
;              Data Communication System Development
;              12203 SW Freeway, MS 701
;              Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
; (C) Copyright 1996. Texas Instruments. All rights reserved.
;{
; Change history:
;
;        VERSION       DATE     /     AUTHORS           COMMENT
;          1.0     July–26–96   /    P.Mallela      original created
;
;}
;{
```

Example 5. Handshake Between Host and Target (Continued)

```
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutine:
;        1) evm_handshake
;    1.3 Specification/Design Reference (optional)
;        called by main.asm depending upon if K_HOST_FLAG is set
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s hand_shk.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include        ”target.inc”
        .include        ”init_54x.inc”
        .include        ”interrpt.inc”
;    3.2 External Data
        .ref            FIFO_DP
        .ref            d_command_reg
        .ref            d_command_value
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
       .def    evm_handshake
```

Example 5. Handshake Between Host and Target (Continued)

```
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;===========================================================================
;
;---------------------------------------------------------------------------
;    5.1 evm_handshake
;
;    5.2 Functional Description
;
;        This initiates the handshake between the host(PC) and the target (DSP).
;        The host writes a command to CH A. This generates an INT1 on the target.
;        The AXST bit on HCR is set to 1.The bit in IFR is polled if it is set
;        then it is cleared to clear pending interrupts. The FIFO is cleared
;        by reading from the FIFO. The command from host is read thru CH A and
;        ARST on TCR is cleared. Another command from target is written to CH A,
;        which sets AXST. Also sets XF low. The host polls XF line. The host reads
;        CH A which clears ARST on host side and AXST on target side.
;
;---------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    evm_handshake
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;            NONE
;    5.5 Outputs
;            NONE
;
;    5.6 Global
;        Data structure:     d_command_reg
;        Data Format:        16-bit variable
;        Modified:           Yes
;        Description:        command from host is read thru CH A
;
;        Data structure:     d_command_value
;        Data Format:        16-bit variable
;        Modified:           Yes
;        Description:        holds the command value

;    5.7 Special considerations for data structure
;        -
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |UM|NU|NU|NU |NU|NU |
;
; Note : UM - Used & Modified, U - Used, NU - Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
```

Example 5. Handshake Between Host and Target (Continued)

```
;=============================================================================
;HeaderEnd
;    5.10 Code
     .sect    ”handshke”
evm_handshake:
     LD       #0,DP
     BITF     IFR,02h                        ; Poll for INT1
     BC       evm_handshake,NTC              ; ARST = 1
     STM      #K_INT1,IFR                    ; clear the pending interrupt
     LD       #FIFO_DP,DP
     RPT      #K_FIFO_SIZE–1
     PORTR    K_CHB,d_command_reg            ; assures that FIFO is empty to
     PORTR    K_CHA,d_command_value          ; ARST = 0
target_handshake_command:                    ; read the command from HOST
                                             ;to acknowledge INT1
     PORTR    K_TRGCR_ADDR,d_command_reg     ; while (port14 & ARST)
     BITF     d_command_reg,K_ARST           ; check FIFO empty
     BC       target_handshake_command,TC    ; branch occurs
     LD       #K_HANDSHAKE_CMD,A             ; indicate of FIFO empty
     SUB      d_command_value,A
bad_handshake_command
     BC       bad_handshake_command,ANEQ     ; read the command send by hosts
     ST       #K_AXST_CLEAR,d_command_reg    ; send to a command to clear AXST
     PORTW    d_command_reg, K_CHA           ; write command to command reg A
                                             ; AXST = 1
     RSBX     XF                             ; XF = 0
     RET
    .end
```

Example 6. Initialization of Variables, Pointers, and Buffers

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    prcs_int.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR       Padma P. Mallela
;
;              Application Specific Products
;              Data Communication System Development
;              12203 SW Freeway, MS 701
;              Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
; (C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /     AUTHORS           COMMENT
;          1.0     July–29–96  /    P.Mallela      original created
;
;}
;{
```

Example 6. Initialization of Variables, Pointers, and Buffers (Continued)

```
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains initialization of buffers,pointers and variables
;
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s prcs_int.asm
;
;    1.6 Notes and Special Considerations
;{
; This code is written for 541 device. The code is tested on C54x EVM
;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include   ”main.inc”
;    3.2 External Data
        .ref       input_data,output_data
        .ref       present_command
        .ref       d_rcv_in_ptr,d_xmt_out_ptr
        .ref       RCV_INT1_DP
        .ref       d_buffer_count
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
```

Example 6. Initialization of Variables, Pointers, and Buffers (Continued)

```
;    4.5 Export Functions
         .def    init_bffr_ptr_var
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;===============================================================================
;
;-------------------------------------------------------------------------------
;    5.1 init_bffr_ptr
;
;    5.2 Functional Description
;        This routine initializes all the buffers, pointers and variables
;-------------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    init_bffr_ptr_var
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;    5.6 Global
;
;        Data structure:     AR2
;        Data Format:        16-bit input buffer pointer
;        Modified:           Yes
;        Description:        initialize to the starting address
;
;        Data structure:     AR3
;        Data Format:        16-bit output buffer pointer
;        Modified:           Yes
;        Description:        initialize to the starting address
;
;        Data structure:     present_command
;        Data Format:        16-bit variable
;        Modified:           Yes
;        Description:        holds the present command
;
;        Data structure:     input_data
;        Data Format:        16-bit array
;        Modified:           Yes
;        Description:        address of the input data buffer
;
;        Data structure:     output_data
;        Data Format:        16-bit array
;        Modified:           Yes
;        Description:        address of the output data buffer
;
;        Data structure:     d_rcv_in_ptr
;        Data Format:        16-bit var
;        Modified:           Yes
;        Description:        holds the staritng address of input bffr
;
;        Data structure:     d_xmt_out_ptr
;        Data Format:        16-bit variable
;        Modified:           Yes
;        Description:        holds the starting address of output bffr
```

Example 6. Initialization of Variables, Pointers, and Buffers (Continued)

```
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in  |0 | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|0 |NU |NU|NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out |2 | 1 | 1 |NU | 1  |NU |NU |UM |UM |UM |NU |NU |NU |NU |UM|NU|0 |NU |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
; ;=============================================================================
;HeaderEnd
;  5.10 Code
         .asg     AR1,ZRPAD_P                  ; zero pad pointer
         .asg     AR2,GETFRM_IN_P              ; get frame input data pointer
         .asg     AR3,GETFRM_OUT_P             ; get frame output data pointer
         .asg     AR2,COFF_P
         .sect    ”zeropad”
init_bffr_ptr_var:
         STM      #RCV_INT1_DP,AR2             ; init all vars to 0
         RPTZ     A,K_FRAME_SIZE/2–1           ; this may need mods if all vars
         STL      A, *AR2+                     ; are not in 1 page
         STM      #input_data,GETFRM_IN_P      ; input buffer ptr
         STM      #output_data,GETFRM_OUT_P    ; output buffer ptr
         LD       #RCV_INT1_DP,DP
         MVKD     GETFRM_IN_P,d_rcv_in_ptr     ; holds present in. bffr ptr
         MVKD     GETFRM_OUT_P,d_xmt_out_ptr   ; holds present out bffr ptr
         ST       #3,present_command           ; initialize present command
         ST       #K_0, d_buffercount          ; reset the buffer count
         STM      #input_data,ZRPAD_P
         RPTZ     A,2*K_FRAME_SIZE–1           ; zeropad both bottom 256
input_data
         STL      A, *ZRPAD_P+                 ; and fft_data buffers
         STM      #output_data,ZRPAD_P
         RPTZ     A,2*K_FRAME_SIZE–1
         STL      A, *ZRPAD_P+
         RET
```

Example 7. Initialization of Serial Port 1

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    init_ser.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR     Padma P. Mallela/Ramesh A Iyer
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
; (C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS            COMMENT
;          1.0     July–29–96  /   P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains the initialization of the serial port 1
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s init_ser.asm
;
;    1.6 Notes and Special Considerations
;{
;    This code is written for 541 device. The code is tested on C54x EVM ;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;
```

Example 7. Initialization of Serial Port 1 (Continued)

```
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include    ”interrpt.inc”
         .include    ”init_ser.inc”
;    3.2 External Data
;        NONE
;    3.3 Import Functions
;        NONE
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
         AIC_VAR_DP      .usect  ”aic_vars”,0
         aic_in_rst      .usect  ”aic_vars”,1
         aic_out_of_rst  .usect  ”aic_vars”,1
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def    serial_init
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 serial_init
;
;    5.2 Functional Description
;        This routine initializes the serial port 1 of 541. The serial port is put
;        in reset by writting 0’s to RRST and XRST bits and pulled out of reset by
;        writting 1’s to both RRST and XRST bits. This routine also puts the AC01
;        in reset and after 12 cycles the AC01 is pulled out of reset. The serial
;        port initialization is done during the 12 cylce latency of the AC01 init.
;
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    serial_init
;
;        Reentrancy:    No
;        Recursive :    No
;
;    5.4 Inputs
;
;    5.5 Outputs
;
;    5.6 Global
;
```

Example 7. Initialization of Serial Port 1 (Continued)

```
;        Data structure: aic_in_rst
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds the value to put AC01 in reset state
;
;        Data structure: aic_out_of_reset
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds the value to put AC01 out of reset state
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |0 | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.10 Code
            .sect       ”ser_cnfg”
serial_init:
         LD       #AIC_VAR_DP,DP               ; initialize DP for aic_reset
         ST       #K_0, aic_in_rst             ; bit 15 = 0 of TCR resets AIC
         PORTW    aic_in_rst,K_TRGCR_ADDR      ;write to address 14h (TCR)
*****************************************************************************
*We need at least 12 cycles to pull the AIC out of reset.
*****************************************************************************
         STM      #K_SERIAL_RST, SPC1          ;reset the serial port with
                                               ;0000 0000 0000 1000
         STM      #K_SERIAL_OUT_RST, SPC1      ;bring ser.port out of reset
                                               ;0000 0000 1100 1000
         RSBX     INTM
         LD       #0,DP
         ORM      #(K_RINT1|K_INT1),IMR        ; Enable RINT1,INT1
                                               ; 0000 0000 0100 0010
         LD       #AIC_VAR_DP,DP               ; restore DP
         STM      #(K_RINT1),IFR               ; clear RINT1
         STM      #K_0,DXR1                    ; 0000 0000 0100 0000
; Pull the AC01 out of reset – the AC01 requires that it be held in reset for
; 1 MCLK, which is equivalent to 96.45ns (based on an MCLK of 10.368MHz)
         ST       #K_8000, aic_out_of_rst      ; bit 15 = 1 brings AIC from reset
         RETD
         PORTW aic_out_of_rst, K_TRGCR_ADDR    ; AIC out of reset
         .end
```

Example 7. Initialization of Serial Port 1 (Continued)

```
*************************************************************************
* FILENAME: “INIT_SER.INC“
*  ---------------------------------------------------------
*  | 15  | 14  |   13   |   12    | 11   |  10  | 9  |  8  |
*  |-----|-----|--------|---------|------|------|----|-----|
*  |FREE | SOFT| RSRFULL| XSREMPTY| XRDY | RRDY | IN1| IN0 |
*  ---------------------------------------------------------
*  -------------------------------------------------
*  | 7   | 6   |   5 | 4  |  3  |  2 | 1   |  0  |
*  |-----|-----|-----|----|-----|----|-----|-----|
*  |RRST | XRST| TXM | MCM| FSM | FO | DLB | RES |
*  -------------------------------------------------
* This include file includes the SPC1 register configuration
*************************************************************************
;Bit     Name        Function
;0    Reserved    Always read as 0
;1    DLB         Digital loop back: 0 –> Disabled, 1_.Enabled
;2    FO          Format bit: 0 –> data transfered as 8 bit bytes, 1 –> 16 bit
                  words
;3    FSM         Frame sync pulse: 0 –> serial port in continuous mode, 1 –> FSM
                  is required
;4    MCM         Clock mode bit: 0 –> CLKX obtained from CLKX pin 1 –> CLKX
                  obtained from CLKX
;5    TXM         Transmit mode bit: 0 –> Frame sync pulses generated externally
                  and supplied on FSX pin, 1 –> Internally generated frame sync
                  pulses out on FSX pin
;6    XRST        Transmit reset bit: 0 –> reset the serial port, 1 –> bring
                  serial port out of reset
;7    RRST        Receive reset bit: 0 –> reset the serial port, 1 –> bring
                  serial port out of reset
;8    IN0         Read–only bit reflecting the state of the CLKR pin
;9    IN1         Read–only bit reflecting the state of the CLKX pin
;10   RRDY        Transition from 0 to 1 indicates data is ready to be read
;11   XRDY        Transition from 0 to 1 indicates data is ready to be sent
;12   XSREMPTY    Transmit shift register empty  ( Read–only) 0 –> tramsitter
                  has experienced underflow
;13   RSRFUL      Receive shift register full flag (Read–only): 0 –> Receiver
                  has experienced overrun
;14   SOFT        Soft bit – 0 –> immdeiate stop, 1–> stop after word completion
;15   FREE        Free run bit: 0 –> behaviour depends on SOFT bit, 1–> free run
                  regardless of SOFT bit
; The system has the following configuration:
;     Uses 16–bit data => FO = 0
;        Operates in burst mode => FSM = 1
;     CLKX is derived from CLKX pin => MCM = 0
;     Frame sync pulses are generated externally by the AIC => TXM = 0
; Therefore, to reset the serial port, the SPC field would have
;     0000 0000 0000 1000
; To pull the serial port out of reset, the SPC field would have
;     0000 0000 1100 1000
K_0                 .set     00000000b << 8  ; bits 15–8 to 0 at reset
K_RRST              .set     0b << 7         ; First write to SPC1 is 0
                                             ; second write is 1
K_XRST              .set     0b << 6         ; First write to SPC1 is 0
                                             ; second write is 1
K_TXM               .set     0b << 5
K_MCM               .set     0b << 4
K_FSM               .set     1b << 3         ; Frame Sync mode
K_ZERO              .set     000b << 0
```

Example 7. Initialization of Serial Port 1 (Continued)

```
K_SERIAL_RST      .set      K_0|K_RRST|K_XRST|K_TXM|K_MCM|K_FSM|K_ZERO
                                              ; first write to SPC1 register
K_RRST1           .set    1b << 7             ; second write to SPC1
K_XRST1           .set    1b << 6             ; second write to SPC1
K_SERIAL_OUT_RST  .set    K_0|K_RRST1|K_XRST1|K_TXM|K_MCM|K_FSM|K_ZERO

K_TRGCR_ADDR      .set    14h                 ; Target/Status I/O address

K_0               .set    0h
K_8000            .set    8000h               ; set bit 15 to pull AIC out
                                              ; of reset
* FILENAME: INTERRUPT.INC
*  ------------------------------------------------------------------------
*  | 15     9 |  8  |  7   |   6  |  5   |  4   | 3   | 2   | 1   |  0  |
*  |----------|-----|------|------|------|------|-----|-----|-----|-----|
*  | Reserved | INT3| XINT1| RINT1| XINT0| RINT0| TINT| INT2| INT1| INT0|
*  ------------------------------------------------------------------------
* This file includes the IMR and IFR configuration
*************************************************************************
K_IMR_RESR        .set    0000000b << 9       ; reserved space
K_INT3            .set    1b << 8             ; disable INT3
K_XINT1           .set    1b << 7             ; disable transmit interrupt 1
K_RINT1           .set    1b << 6             ; enable receive interrupt 1
K_XINT0           .set    1b << 5             ; disable transmit interrupt 0
K_RINT0           .set    1b << 4             ; disable receive interrupt
K_TINT            .set    1b << 3             ; disbale timer interrupt
K_INT2            .set    1b << 2             ; disable INT2
K_INT1            .set    1b << 1             ; disbale INT1
K_INT0            .set    1b << 1             ; enable INT0
```

Example 8. 'AC01 Initialization

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    init_aic.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR       Padma P. Mallela/Ramesh A. Iyer
;
;              Application Specific Products
;              Data Communication System Development
;              12203 SW Freeway, MS 701
;              Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS          COMMENT
;          1.0     July-29-96/     P.Mallela      original created
;
;}
;{
```

Example 8. ’AC01 Initialization (Continued)

```
;     1.  ABSTRACT
;
;     1.1 Function Type
;         a.Core Routine
;         b.Subroutine
;
;     1.2 Functional Description
;         This file contains the initialization of AC01
;
;
;     1.3 Specification/Design Reference (optional)
;
;     1.4 Module Test Document Reference
;         Not done
;
;     1.5 Compilation Information
;         Compiler:       TMS320C54X  ASSEMBLER
;         Version:        1.02 (PC)
;         Activation:     asm500 –s init_aic.asm
;
;     1.6 Notes and Special Considerations
;{
;         This code is written for 541 device. The code is tested on C54x EVM ;
;}
;{
;     2. VOCABULARY
;
;     2.1 Definition of Special Words, Keywords (optional)
;         –
;     2.2 Local Compiler Flags
;         –
;     2.3 Local Constants
;         –
;}
;{
;     3. EXTERNAL RESOURCES
;
;     3.1 Include Files
          .mmregs
          .include        ”interrpt.inc”
;     3.2 External Data
;         NONE
;     3.3 Import Functions
          .ref    wrt_cnfg                        ; initializes AC01
;}
;{
;     4. INTERNAL RESOURCES
;
;     4.1 Local Static Data
;         –
;     4.2 Global Static Data
;         –
;     4.3 Dynamic Data
;         –
;     4.4 Temporary Data
;         –
;     4.5 Export Functions
          .def    aic_init
;}
```

Example 8. ’AC01 Initialization (Continued)

```
;    5. SUBROUTINE CODE
;       HeaderBegin
;===============================================================================
;
;
;-------------------------------------------------------------------------------
;    5.1 aic_init
;
;    5.2 Functional Description
;        This routine disables IMR and clears any pending interrupts before
;        initializing AC01. The wrt_cnfg function configures the AC01
;-------------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;                CALL    aic_init
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;
;    5.5 Outputs
;
;    5.6 Global
;
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |  |0  | 1 | 1 |NU  | 1 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|  |0  | 1 | 1 |NU  | 1 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;===============================================================================
;       HeaderEnd
;    5.10 Code
        .sect    ”aic_cnfg”
aic_init:
        CALLD    wrt_cnfg                       ; initialize AC01
        ANDM     #(~K_RINT1|K_INT1),IMR         ; disable receive_int1,INT1
        ORM      #(K_RINT1|K_INT1),IMR          ; enable the RINT1, INT1
        RETD
        STM      (K_RINT1),IFR                  ; service any pending interrupt
       .end
```

Example 8. 'AC01 Initialization (Continued)

```
*********************************************************************
* This file includes the AC01 registers initialization
* All registers have 2 control bits that initiates serial communication
* There are 2 communication modes – primary and secondary communications
* During primary communication the control bits D00 and D01 are 11 to request
* for a secondary communication. In the secondary serial communications the
* control bits D15 and D14 perform same control function as primary.
* The R/W~ bit at reset is set to 0 placing the device in write mode.
*********************************************************************
K_NOP_ADDR         .set   0 << 8
K_REG_0            .set   K_NOP_ADDR
K_A_ADDR           .set   1 << 8               ; REG 1 address
K_A_REG            .set   36                   ;
K_REG_1            .set   K_A_ADDR|K_A_REG     ; FCLK = 144KHz => A =24h
K_B_ADDR           .set   2 << 8               ; REG 2 address
K_B_REG            .set   18                   ;
K_REG_2            .set   K_B_ADDR|K_B_REG     ; Sampling rate = 8KHz
K_AA_ADDR          .set   3 << 8               ; Register 3 address
K_AA_REG           .set   0                    ;
K_REG_3            .set   K_AA_ADDR|K_AA_REG;  ; no shift
K_GAIN_ADDR        .set   4 << 8               ; Register 4 address
K_MONITOR_GAIN     .set   00b << 4             ; Monitor output gain = squelch
K_ANLG_IN_GAIN     .set   01b << 2             ; Analog input gain = 0dB
K_ANLG_OUT_GAIN    .set   01b << 0             ; Analog output gain = 0dB
K_REG_4            .set   K_GAIN_ADDR|K_MONITOR_GAIN|K_ANLG_IN_GAIN|K_ANLG_OUT_GAIN
K_ANLG_CNF_ADDR    .set   5 << 8               ; Register 5 address
K_ANLG_RESRV       .set   0 << 3               ; Must be set to 0K_HGH_FILTER .set 0 << 2
                                               ; High pass filter is enabled
K_ENBL_IN          .set   01b << 0             ; Enables IN+ and IN–
K_REG_5            .set   K_ANLG_CNF_ADDR|K_ANLG_RESRV|K_HGH_FILTER|K_ENBL_IN
K_DGTL_CNF_ADDR    .set   6 << 8               ; Register 6 address
K_ADC_DAC          .set   0 << 5               ; ADC and DAC is inactive
K_FSD_OUT          .set   0 << 4               ; Enabled FSD output
K_16_BIT_COMM      .set   0 << 3               ; Normal 16–bit mode
K_SECND_COMM       .set   0 << 2               ; Normal secondary communication
K_SOFT_RESET       .set   0 << 1               ; Inactive reset
K_POWER_DWN        .set   0 << 0               ; Power down external
K_REG_HIGH_6       .set   K_DGTL_CNF_ADDR|K_ADC_DAC|K_FSD_OUT|K_16_BIT_COMM
K_REG_LOW_6        .set   K_SECND_COMM|K_SOFT_RESET|K_POWER_DWN
K_REG_6            .set   K_REG_HIGH_6|K_REG_LOW_6
K_FRME_SYN_ADDR    .set   7 << 8               ; Register 7 address
K_FRME_SYN         .set   0 << 8               ;
K_REG_7            .set   K_FRME_SYN_ADDR|K_FRME_SYN
K_FRME_NUM_ADDR .set  8 << 8                   ; Register 8 address
K_FRME_NUM      .set  0 << 8                   ;
K_REG_8         .set  K_FRME_NUM_ADDR|K_FRME_NUM
; primary word with D01 and D00 bits set to 11 will cause a
; secondary communications interval to start when the frame
; sync goes low next
K_SCND_CONTRL  .set  11b << 0                  ; Secondary comm.bits
AIC_REG_START_LIST .sect   "aic_reg"           ; includes the aic table
     .word         AIC_REG_END_LIST–AIC_REG_START_LIST–1
     .word         K_REG_1
     .word         K_REG_2
     .word         K_REG_3
     .word         K_REG_4
     .word         K_REG_5
     .word         K_REG_6
     .word         K_REG_7
     .word         K_REG_8
```

Example 8. ’AC01 Initialization (Continued)

```
AIC_REG_END_LIST
K_XRDY  .set 0800h                          ; XRDY bit in SPC1
    .sect       ”aic_cnfg”
aic_init:
    CALLD       wrt_cnfg                    ; initialize AC01
    ANDM        #(~K_RINT1),IMR             ; disable receive_int1
    ORM         #(K_RINT1|K_INT1),IMR       ; enable the RINT1, INT1
    RETD
    STM         #(K_RINT1),IFR              ; service any pending interrupt
    .end
```

Example 9. ’AC01 Register Configuration

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    aic_cfg.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR     Padma P. Mallela/Ramesh A. Iyer
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX  77477
;
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;VERSION      DATE     /    AUTHORS           COMMENT
;  1.0     July–25–96  /   P.Mallela      original created
;
;}
;{
;    1.  ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains the AC01 initialization
;
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
```

Example 9. ’AC01 Register Configuration (Continued)

```
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s aic_cfg.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include        ”aic_cfg.inc”
;    3.2 External Data
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
       .def    wrt_cnfg
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;===============================================================================
;
;-------------------------------------------------------------------------------
;    5.1 wrt_cnfg
;
;    5.2 Functional Description
;        Writes new configuration data into the AC01. Assuming a system
;        which processes speech signals and *  requires the following parameters
;        Low pass filter cut–off frequency = 3.6 kHz
;        Sampling rate = 8000 Hz
;        Assume the Master clock MCLK = 10.368 MHz
;        This example demonstrates how to program these parameters –
;        the registers affected are:
;        Register A which determines the division of the MCLK frequency
;        to generate the internal filter clock FCLK.
```

Example 9. ’AC01 Register Configuration (Continued)

```
;        It also determines the –3 dB corner frequency of the low–pass filter
;        Register B  which determines the division of FCLK to generate
;        the sampling (conversion) frequency
;        It also determines the –3dB corner frequency of the high–pass filter
;-------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;                CALLD   wrt_cnfg
;                STM     #K_RINT1,IFR
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;        Data structure: AR1
;        Data Format:    16–bit pointer
;        Modified:       No
;        Description:    indexes the table
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;

;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |NU |UM |NU |NU |UM |NU |NU |NU |UM|NU|NU|UM |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
; HeaderEnd
;   5.10 Code
;
;=========================================================================
    .asg    AR1,AIC_REG_P
    .sect   ”aic_cnfg”
wrt_cnfg:
    STM     #aic_reg_tble,AIC_REG_P             ; init AR1
    RPT     #AIC_REG_END_LIST–AIC_REG_START_LIST
    MVPD    #AIC_REG_START_LIST,*AIC_REG_P+     ; move the table
    STM     #aic_reg_tble,AIC_REG_P             ; init AR1
    STM     #K_REG_0,DXR1                       ; primary data word –
                                                ; a jump start!
```

Example 9. 'AC01 Register Configuration (Continued)

```
wait_xrdy
    BITF    SPC1,K_XRDY                         ; test XRDY bit in SPC1
    BC      wait_xrdy,NTC                       ; loop if not set
    STM     #K_SCND_CONTRL,DXR1                 ; send primary word with
                                                ; D01–D00  = 11 to
                                                ; signify secondary communication
    LD      *AIC_REG_P+,A
    STLM    A,BRC                               ; gives the # of registers to be
                                                ; initialized
            NOP
    RPTB    aic_cfg_complte–1
wait_xrdy1
    BITF    SPC1,K_XRDY                         ; test XRDY bit in SPC1
    BC      wait_xrdy1,NTC                      ; loop if not set
    LD      *AIC_REG_P+,A                       ; Read the register contents
    STLM    A, DXR1
wait_xrdy2
    BITF    SPC1,K_XRDY                         ; test XRDY bit in SPC1
    BC      wait_xrdy2,NTC                      ; loop if not set
    STM     #K_SCND_CONTRL,DXR1                 ; set to read the next register
aic_cfg_complte                                 ; contents
wait_xrdy3
    BITF    SPC1,K_XRDY                         ; test XRDY bit in SPC1
    BC      wait_xrdy3,NTC                      ; loop if not set
    RET
```

Example 10. Receive Interrupt Service Routine

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    rcv_int1.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR     Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS          COMMENT
;          1.0     July–29–96  /   P.Mallela      original created
;
;}
;{
```

Example 10. Receive Interrupt Service Routine (Continued)

```
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains interrput service routine INT1:
;        receive_int1
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s rcv_int1.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
; 3 EXTERNAL RESOURCES
;
;     3.1 Include Files
          .mmregs
          .include    ”INTERRPT.INC”
          .include    ”main.inc”
;     3.2 External Data
          .ref        d_frame_flag
          .ref        d_index_count
          .ref        d_rcv_in_ptr,d_xmt_out_ptr
          .ref        RCV_INT1_DP
;     3.3 Import Functions
;}
;{
; 4 INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
          .def        receive_int1
;}
```

Example 10. Receive Interrupt Service Routine (Continued)

```
;    5. SUBROUTINE CODE
;       HeaderBegin
;===========================================================================
;
;---------------------------------------------------------------------------
;    5.1 receive_int1
;
;    5.2 Functional Description
;        This routine services receive interrupt1. Accumulator A, AR2 and AR3
;        are pushed onto the stack since AR2 and AR3 are used in other
;        applications. A 512 buffer size of both input and output uses circular
;        addressing. After every 256 collection of input samples a flag is set to
;        process the data. A PING/PONG buffering scheme is used such that upon
;        processing PING buffer, samples are collected in the PONG buffer and vice
;        versa.
;---------------------------------------------------------------------------
;    5.3 Activation
;        Activation example:
;            BD    receive_int1
;            PSHM  ST0
;            PSHM  ST1
;
;        Reentrancy:    No
;        Recursive :    No
;
;    5.4 Inputs
;        NONE
;
;    5.5 Outputs
;        NONE
;
;   5.6 Global
;
;        Data structure:     AR2
;        Data Format:        16–bit input buffer pointer
;        Modified:           Yes
;        Description:        either point to PING/PONG buffer. Upon entering
;                            AR2 is pushed onto stack and the address is restored
;                            through d_rcv_in_ptr
;
;        Data structure:     AR3
;        Data Format:        16–bit output buffer pointer
;        Modified:           Yes
;        Description:        either point to PING/PONG buffer. Upon entering
;                            AR3 is pushed onto stack and the address is restored
;                            through d_rcv_in_ptr
;
;        Data structure:     d_index_count
;        Data Format:        16–bit var
;        Modified:           Yes
;        Description:        holds the number samples that has been collected from
;                            AC01
;
;        Data structure:     d_frame_flag
;        Data Format:        16–bit variable
;        Modified:           Yes
;        Description:        flag is set if 256 samples are collected
;
```

Example 10. Receive Interrupt Service Routine (Continued)

```
;        Data structure:     d_rcv_in_ptr
;        Data Format:        16–bit var
;        Modified:           Yes
;        Description:        holds the input buffer address where the newest
;                            sample is stored
;
;        Data structure:     d_xmt_out_ptr
;        Data Format:        16–bit variable
;        Modified:           Yes
;        Description:        holds the output buffer address where the oldest
;                            sample is sent as output
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |U  |U  |NU |NU |NU |NU |U |NU|NU|NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |U  |U  |NU |NU |NU |NU |U |NU|U |NU |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
; 5.10 Code
         .asg    AR2,GETFRM_IN_P              ; get frame input data pointer
         .asg    AR3,GETFRM_OUT_P             ; get frame output data pointer
         .asg    AR2,SAVE_RSTORE_AR2
         .asg    AR3,SAVE_RSTORE_AR3
         .sect   ”main_prg”
receive_int1:
         PSHM    AL
         PSHM    AH
         PSHM    AG
         PSHM    SAVE_RSTORE_AR2
         PSHM    SAVE_RSTORE_AR3
         PSHM    BK
         PSHM    BRC
         STM     #2*K_FRAME_SIZE,BK           ; circular buffr size of in,out
                                              ; arrays
         LD      #RCV_INT1_DP,DP              ; init. DP
         MVDK    d_rcv_in_ptr,GETFRM_IN_P     ; restore input circular bffr ptr
         MVDK    d_xmt_out_ptr,GETFRM_OUT_P   ; restore output circular bffr ptr
         ADDM    #1,d_index_count             ; increment the index count
         LD      #K_FRAME_SIZE,A
         SUB     d_index_count, A
         BC      get_samples,AGT              ;check for a frame of samples
frame_flag_set
          ST      #K_FRAME_FLAG,d_frame_flag  ; set frame flag
          ST      #0,d_index_count            ; reset the counter
```

Example 10. Receive Interrupt Service Routine (Continued)

```
get_samples
          LDM     DRR1,A                          ; load the input sample
          STL     A,*GETFRM_IN_P+%                ; write to buffer
          LD      *GETFRM_OUT_P+%,A               ; if not true, then the filtered
          AND     #0fffch,A                       ; signal is send as output
          STLM    A,DXR1                          ; write to DXR1
          MVKD    GETFRM_IN_P,d_rcv_in_ptr        ; save input circualr buffer ptr
          MVKD    GETFRM_OUT_P,d_xmt_out_ptr      ; save out circular bffr ptr
          POPM    BRC
          POPM    BK
          POPM    SAVE_RSTORE_AR3
          POPM    SAVE_RSTORE_AR2
          POPM    AG
          POPM    AH
          POPM    AL
          POPM    ST1
          POPM    ST0
          RETE                                    ; return and enable interrupts
```

Example 11. Task Scheduling

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    task.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS           COMMENT
;          1.0     July-29-96  /   P.Mallela      original created
;
;}
;{
;    1. ABSTRACT;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file initiates task scheduling
;
;
;    1.3 Specification/Design Reference (optional)
;
```

Example 11. Task Scheduling (Continued)

```
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s task.asm
;
;    1.6 Notes and Special Considerations
;{
; This code is written for 541 device. The code is tested on C54x EVM
;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .include        ”init_54x.inc”
         .include        ”main.inc”
;    3.2 External Data
         .ref            d_task_addr,d_task_init_addr
         .ref            d_buffer_count
         .ref            present_command
         .ref            last_command
         .ref            d_output_addr,d_input_addr
         .ref            input_data,output_data
         .ref            d_frame_flag
         .ref            task_init_list,task_list
;    3.3 Import Functions
         .ref            echo_task
         .ref            fir_init,fir_task
         .ref            do_nothing,no_echo_init_task
         .ref            fir_init,fir_task
         .ref            iir_init,iir_task
         .ref            sym_fir_task,sym_fir_init
         .ref            adapt_init,adapt_task
         .ref            rfft_task
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
```

Example 11. Task Scheduling (Continued)

```
;    4.5 Export Functions
        .def    task_handler
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 task_handler
;
;    5.2 Functional Description
;        This routine handles the task scheduling. The present_command
;        can take values 1,2,3,4,5,6. If
;        present_command = 1 – Echo program is enabled
;        present_command = 2 – FIR is enabled
;        present_command = 3 – IIR is enabled
;        present_command = 4 – Symmetric FIR is enabled
;        present_command = 5 – Adaptive fitler is enabled
;        present_command = 6 – FFT is enabled
;        For every cycle the program checks if the current task is same
;        as previous task if it is, then no initialization is done.
;        If its not then cirular buffers, variables pointers are intialized
;        depending upon the task.
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    task_handler
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;
;        Data structure: present_command
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds the present command
;
;    5.5 Outputs
;
;        Data structure: AR6
;        Data Format:    16–bit input buffer pointer
;        Modified:       Yes
;        Description:    either point to PING/PONG buffer
;
;        Data structure: AR7
;        Data Format:    16–bit output buffer pointer
;        Modified:       Yes
;        Description:    either point to PING/PONG buffer
;
;    5.6 Global
;
;        Data structure: last_command
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds the last command
;
```

Example 11. Task Scheduling (Continued)

```
;        Data structure: d_frame_flag
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    gets reset after 256 samples
;
;        Data structure: d_buffer_count
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    used to load either PING/PONG bffr addresses
;
;        Data structure: input_data
;        Data Format:    16–bit array
;        Modified:       Yes
;        Description:    address of the input data buffer
;
;        Data structure: output_data
;        Data Format:    16–bit array
;        Modified:       Yes
;        Description:    address of the output data buffer
;
;        Data structure: d_input_addr
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds either PING/PONG address
;
;        Data structure: d_output_addr
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds either PING/PONG address
;
;        Data structure: d_task_addr
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds the task program address
;
;        Data structure: d_task_init_addr
;        Data Format:    16–bit variable
;        Modified:       Yes
;        Description:    holds the task init. address
;
;    5.7 Special considerations for data structure
;    –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |  |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |U  |NU|NU|NU|NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |  |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |U  |U  |UM|UM|NU|NU |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
```

Example 11. Task Scheduling (Continued)

```
;    5.10 Code
         .asg     AR6,INBUF_P                    ; PING/PONG input buffer
         .asg     AR7,OUTBUF_P                   ; PING/PONG output buffer
         .sect    ”task_hnd”
task_handler:
;        LD          #TASK_VAR_DP,DP
         ST          #K_0,d_frame_flag             ; reset the frame flag
         ADDM        #1,d_buffer_count
         LD          #input_data,A                 ; load PING input address
         LD          #output_data,B                ; load PING output address
         BITF        d_buffer_count,2              ; check if PING/PONG address
         BC          reset_buffer_address,NTC      ; needs to be loaded
         ADD         #K_FRAME_SIZE,A               ; PONG input address
         ADD         #K_FRAME_SIZE,B               ; PONG output address
         ST          #K_0,d_buffer_count           ; reset counter
reset_buffer_address
         STLM        A,INBUF_P                     ; input buffer address
         STL         A,d_input_addr                ; restore either PING/PONG bffr
         STLM        B,OUTBUF_P                    ; output buffer address
         STL         B,d_output_addr               ; restore either PING/PONG bffr
         LD          present_command,A
         SUB         last_command,A
         BC          new_task,ANEQ                 ; check if PC = LC
         LD          d_task_addr,A                 ; task_addr should
                                                   ; contain previous
                                                   ; PC = LC
         CALA        A                              ; call present task
         RET
new_task:
      MVKD        present_command,last_command  ; restore the present command
      LD          #task_init_list,A             ; loads PC init task
      ADD         present_command,A             ; computes the present task
      REDA        d_task_init_addr              ; save the PC into task_addr
      LD          d_task_init_addr,A
      CALA        A                             ; initializes the present task
      LD          #task_list,A;
         ADD         present_command,A             ; computes the present task
      READA       d_task_addr                   ; save the PC into
                                              ; task_addr
      LD       d_task_addr,A
      CALA     A
      RET
do_nothing:
      RET
no_echo_init_task:
      RET
      .end
```

Example 12. Echo the Input Signal

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    echo.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS           COMMENT
;          1.0     July-24-96  /   P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutines:
;            echo_task
;    1.3 Specification/Design Reference (optional)
;        called by task.asm depending upon the task
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s echo.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
;    2.2 Local Compiler Flags
;        -
```

Example 12. Echo the Input Signal (Continued)

```
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include        ”main.inc”
;    3.2 External Data
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def    echo_task
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 echo_task
;
;    5.2 Functional Description
;        This function reads a sample from either PING/PONG buffer and puts it
;        back in the output buffer. This is repeated 256 times i.e., size of the
;        frame. The present command in this case is 1.
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;            CALL    echo_task
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;
;                    Data structure: AR6
;                    Data Format:    16–bit input buffer pointer
;                    Modified:       Yes
;                    Description:    either point to PING/PONG buffer
;
;    5.5 Outputs
;
;                    Data structure: AR7
;                    Data Format:    16–bit output buffer pointer
;                    Modified:       Yes
;                    Description:    either point to PING/PONG buffer
;
```

Example 12. Echo the Input Signal (Continued)

```
;    5.6 Global
;
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;in  |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |U  |U  |NU|NU|NU|NU |NU|NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;out |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |UM |UM |UM|NU|NU|UM |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;            Execution time: ?cycles
;            Call rate:      not applicable for this application
; ;=============================================================================
;HeaderEnd
;    5.10 Code
          .asg      AR6,INBUF_P                   ; PING/PONG input buffer
          .asg      AR7,OUTBUF_P                  ; PING/PONG output buffer
          .sect     ”echo_prg”
echo_task:
          STM       #K_FRAME_SIZE–1,BRC           ; frame size of 256
          RPTB      echo_loop–1
          LD        *INBUF_P+, A                  ; load the input value
          STL       A, *OUTBUF_P+
echo_loop
          RET                                     ; output buffer
```

Example 13. Low-Pass FIR Filtering Using MAC Instruction

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    fir.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR    Padma P. Mallela
;
;                  Application Specific Products
;                  Data Communication System Development
;                  12203 SW Freeway, MS 701
;                  Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
```

Example 13. Low-Pass FIR Filtering Using MAC Instruction (Continued)

```
;         VERSION       DATE      /        AUTHORS              COMMENT
;           1.0       July–26–96  /       P.Mallela         original created
;
;}
;{
;     1. ABSTRACT
;
;     1.1 Function Type
;          a.Core Routine
;          b.Subroutine
;
;     1.2 Functional Description
;         This file contains two subroutines:
;         1) fir_init
;         2) fir_task
;     1.3 Specification/Design Reference (optional)
;         called by task.asm depending upon the task thru CALA
;
;     1.4 Module Test Document Reference
;         Not done
;
;     1.5 Compilation Information
;         Compiler:       TMS320C54X  ASSEMBLER
;         Version:        1.02 (PC)
;         Activation:     asm500 –s fir.asm
;
;     1.6 Notes and Special Considerations
;         –
;}
;{
;    2. VOCABULARY
;
;     2.1 Definition of Special Words, Keywords (optional)
;         –
;     2.2 Local Compiler Flags
;         –
;     2.3 Local Constants
;         –
;}
;{
;     3. EXTERNAL RESOURCES
;
;     3.1 Include Files
          .mmregs
          .include    ”main.inc”
;     3.2 External Data
          .ref        d_filin                    ; filter input
          .ref        d_filout                   ; filter output
          .ref        d_data_buffer
          .ref        fir_task
          .ref        COFF_FIR_START,COFF_FIR_END
          .ref        fir_coff_table
;     3.3 Import Functions
;}
;{
;     4. INTERNAL RESOURCES
;
;     4.1 Local Static Data
;         –
```

Example 13. Low-Pass FIR Filtering Using MAC Instruction (Continued)

```
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
        .def        fir_init                        ; initialize FIR filter
        .def        fir_filter                      ; perform FIR filtering
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;*****************************************************************************
;
;-----------------------------------------------------------------------------
;    5.1 fir_init
;
;    5.1.1 Functional Description
;          This routine initializes cicular buffers both for data and coeffs.
;-----------------------------------------------------------------------------
;
;    5.1.2 Activation
;        Activation example:
;            CALL    fir_init
;        Reentrancy:     No
;        Recursive :     No
;
;    5.1.3 Inputs
;          NONE
;    5.1.4 Outputs
;          NONE
;
;    5.1.5 Global
;
;          Data structure:     AR0
;          Data Format:        16–bit index pointer
;          Modified:           No
;          Description:        uses in circular addressing mode for indexing
;
;          Data structure:     AR4
;          Data Format:        16–bit x(n) data buffer pointer
;          Modified:           Yes
;          Description:        initializes the pointer
;
;          Data structure:     AR5
;          Data Format:        16–bit w(n) coeff buffer pointer
;          Modified:           Yes
;          Description:        initializes the pointer
;
;    5.1.6 Special considerations for data structure
;          –
;    5.1.7 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC|  T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |   |   |
;in  |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |UM |NU |NU|NU |NU |NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |   |   |
;out |U | 1 | 1 |NU | 1  |NU |UM |NU |NU |NU |UM |UM |NU |NU |UM |NU |NU|NU |NU |NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
```

Example 13. Low-Pass FIR Filtering Using MAC Instruction (Continued)

```
;
;    5.1.8 Execution
;            Execution time: ?cycles
;            Call rate:      not applicable for this application
; ;=============================================================================
;    HeaderEnd
;    5.1.9 Code
            .asg    AR0, FIR_INDEX_P
            .asg    AR4,FIR_DATA_P
            .asg    AR5,FIR_COFF_P
            .sect   ”fir_prog”
fir_init:
            STM     #fir_coff_table,FIR_COFF_P
            RPT     #K_FIR_BFFR–1                   ; move FIR coeffs from program
            MVPD    #COFF_FIR_START,*FIR_COFF_P+    ; to data
            STM     #K_FIR_INDEX,FIR_INDEX_P
            STM     #d_data_buffer,FIR_DATA_P       ; load cir_bfr address for the
                                                    ; recent samples
            RPTZ    A,#K_FIR_BFFR
            STL     A,*FIR_DATA_P+                  ; reset the buffer
            STM     #(d_data_buffer+K_FIR_BFFR–1), FIR_DATA_P
            RETD
            STM     #fir_coff_table, FIR_COFF_P
;    5. SUBROUTINE CODE
;       HeaderBegin
;=============================================================================
;
;-----------------------------------------------------------------------------
;    5.2 fir_task
;
;    5.2.1 Functional Description
;
;          This subroutine performs FIR filtering using MAC instruction.
;          accumulator A (filter output) = h(n)*x(n–i) for i = 0,1...15
;-----------------------------------------------------------------------------
;
;    5.2.2 Activation
;          Activation example:
;          CALL    fir_task
;          Reentrancy:    No
;          Recursive :    No
;
;    5.2.3 Inputs
;
;          Data structure:     AR6
;          Data Format:        16–bit input buffer pointer
;          Modified:           Yes
;          Description:        either point to PING/PONG buffer
;
;          Data structure:     AR4
;          Data Format:        16–bit data buffer pointer
;          Modified:           Yes
;          Description:        uses circular buffer addressing mode to filter
;                              16 tap Low–Pass filter – init. in fir_init
;
;          Data structure:     AR5
;          Data Format:        16–bit coefficient buffer pointer
;          Modified:           Yes
;          Description:        The 16 tap coeffs comprise the low–pass filter
;                                init. in fir_init
```

Example 13. Low-Pass FIR Filtering Using MAC Instruction (Continued)

```
;    5.2.4 Outputs
;
;          Data structure:     AR7
;          Data Format:        16–bit output buffer pointer
;          Modified:           Yes
;          Description:        either point to PING/PONG buffer
;
;
;    5.2.5 Global
;          NONE
;

;    5.2.6 Special considerations for data structure
;          –
;    5.2.7 Entry and Exit conditions
;
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |U  |NU |U  | U |UM|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |UM |NU |NU |NU |UM |UM |UM |UM |UM|NU|UM|UM |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.2.8 Execution
;          Execution time: ?cycles
;          Call rate:      not applicable for this application
; ;=============================================================================
;HeaderEnd
;    5.2.9 Code
     .asg    AR6,INBUF_P
     .asg    AR7,OUTBUF_P
     .asg    AR4,FIR_DATA_P
     .asg    AR5,FIR_COFF_P
     .sect   ”fir_prog”
fir_task:
;    LD      #FIR_DP,DP
     STM     #K_FRAME_SIZE–1,BRC              ; Repeat 256 times
     RPTBD   fir_filter_loop–1
     STM     #K_FIR_BFFR,BK                   ; FIR circular bffr size
     LD      *INBUF_P+, A                     ; load the input value
fir_filter:
     STL     A,*FIR_DATA_P+%                  ; replace oldest sample with newest
                                              ; sample
     RPTZ    A,(K_FIR_BFFR–1)
     MAC     *FIR_DATA_P+0%,*FIR_COFF_P+0%,A  ; filtering
     STH      A, *OUTBUF_P+                   ; replace the oldest bffr value
fir_filter_loop
     RET
```

Example 14. Low-Pass Symmetric FIR Filtering Using FIRS Instruction

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    sym_fir.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS           COMMENT
;          1.0     July-26-96  /   P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains two subroutines:
;        1) sym_fir_init
;        2) sym_fir_task
;
;    1.3 Specification/Design Reference (optional)
;        called by task.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s sym_fir.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
```

Example 14. Low-Pass Symmetric FIR Filtering Using FIRS Instruction (Continued)

```
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include   ”main.inc”
;    3.2 External Data
         .ref       d_datax_buffer
         .ref       d_datay_buffer
         .ref       FIR_COFF
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def       sym_fir_init                    ; initialize symmetric FIR
         .def       sym_fir_task
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1   sym_fir_init
;
;    5.1.1 Functional Description
;          This routine initializes cicular buffers both for data and coeffs.
;----------------------------------------------------------------------------
;
;    5.1.2 Activation
;          Activation example:
;          CALL            sym_fir_init
;          Reentrancy:     No
;          Recursive :     No
;
;    5.1.3 Inputs
;          NONE
;    5.1.4 Outputs
;          NONE
;
;    5.1.5 Global
;
;          Data structure: AR0
;          Data Format:    16–bit index pointer
;          Modified:       No
;          Description:    uses in circular addressing mode for indexing
```

Example 14. Low-Pass Symmetric FIR Filtering Using FIRS Instruction (Continued)

```
;
;          Data structure: AR4
;          Data Format:    16–bit x(n) data buffer pointer for 8 latest samples
;          Modified:       Yes
;          Description:    initializes the pointer
;
;          Data structure: AR5
;          Data Format:    16–bit x(n) data buffer pointer for 8 oldest samples
;          Modified:       Yes
;          Description:    initializes the pointer
;
;    5.1.6 Special considerations for data structure
;          –
;    5.1.7 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |UM|NU|NU|NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |
;out|U | 1 | 1 |NU | 1  |NU |UM |NU |NU |NU |U  |UM |NU |NU |UM|NU|NU|NU |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.1.8 Execution
;          Execution time: ?cycles
;          Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.1.9 Code
          .asg     AR0,SYMFIR_INDEX_P
          .asg     AR4,SYMFIR_DATX_P
          .asg     AR5,SYMFIR_DATY_P
          .sect    ”sym_fir”
sym_fir_init:
          STM      #d_datax_buffer,SYMFIR_DATX_P      ; load cir_bfr address
                                                      ; for the 8 most
                                                      ; recent samples
          STM      #d_datay_buffer+K_FIR_BFFR/2–1,SYMFIR_DATY_P
                                                      ; load cir_bfr1 address
                                                      ; for the 8 old samples


          STM      #K_neg1,SYMFIR_INDEX_P             ; index offset –
                                                      ; whenever the pointer
                                                      ; hits the top of the bffr,
                                                      ; it automatically hits
                                                      ; bottom address of
                                                      ; buffer and decrements
                                                      ; the counter
          RPTZ     A,#K_FIR_BFFR
          STL      A,* SYMFIR_DATX_P+
          STM      #d_datax_buffer, SYMFIR_DATX_P
          RPTZ     A,#K_FIR_BFFR
          STL      A,* SYMFIR_DATY_P–
          RETD
          STM      #d_datay_buffer+K_FIR_BFFR/2–1, SYMFIR_DATY_P
;    5. SUBROUTINE CODE
;       HeaderBegin
;=============================================================================
;
```

Example 14. Low-Pass Symmetric FIR Filtering Using FIRS Instruction (Continued)

```
;-----------------------------------------------------------------------------
;    5.2 sym_fir_init
;
;    5.2.1 Functional Description
;          This program uses the FIRS instruction to implement symmetric FIR
;          filter. Circular addressing is used for data buffers.The input scheme
;          for the data samples is divided into two circular buffers. The first
;          buffer contains samples from X(–N/2) to X(–1) and the second buffer
;          contains samples from X(–N) to X(–N/2–1).
;-----------------------------------------------------------------------------
;
;    5.2.2 Activation
;          Activation example:
;          CALL    sym_fir_init
;          Reentrancy:    No
;          Recursive :    No
;
;    5.2.3 Inputs
;
;          Data structure: AR6
;          Data Format:    16–bit input buffer pointer
;          Modified:       Yes
;          Description:    either point to PING/PONG buffer
;
;          Data structure: AR4
;          Data Format:    16–bit data buffer pointer
;          Modified:       Yes
;          Description:    uses circular buffer addressing mode to filter
;                          16 tap Low–Pass filter – init. in sym_fir_init
;
;          Data structure: AR5
;          Data Format:    16–bit coefficient buffer pointer
;          Modified:       Yes
;          Description:    The 16 tap coeffs comprise the low–pass filter
;                          init. in sym_fir_init
;    5.2.4 Outputs
;
;          Data structure: AR7
;          Data Format:    16–bit output buffer pointer
;          Modified:       Yes
;          Description:    either point to PING/PONG buffer
;
;    5.2.5 Global
;          NONE
;
;    5.2.6 Special considerations for data structure
;          –
;    5.2.7 Entry and Exit conditions
;
;     |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;     |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;in   |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |U  |U  |U  | U |UM|NU|NU|NU |NU|NU
;     |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;out  |U | 1 | 1 |NU | 1  |NU |UM |NU |NU |NU |UM |UM |NU |NU |UM|UM|NU|UM |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.2.8 Execution
;          Execution time: ?cycles
;          Call rate:    not applicable for this application
```

Example 14. Low-Pass Symmetric FIR Filtering Using FIRS Instruction (Continued)

```
;
;=============================================================================
;HeaderEnd
;    5.2.9 Code
           .asg       AR6,INBUF_P
           .asg       AR7,OUTBUF_P
           .asg       AR4,SYMFIR_DATX_P
           .asg       AR5,SYMFIR_DATY_P
           .sect      ”sym_fir”
sym_fir_task:
   STM   #K_FRAME_SIZE–1,BRC
   RPTBD sym_fir_filter_loop–1
   STM   #K_FIR_BFFR/2,BK
   LD    *INBUF_P+, B
symmetric_fir:
   MVDD  *SYMFIR_DATX_P,*SYMFIR_DATY_P+0%      ; move X(–N/2) to X(–N)
   STL   B,*SYMFIR_DATX_P                      ; replace oldest sample with newest
                                               ; sample
   ADD   *SYMFIR_DATX_P+0%,*SYMFIR_DATY_P+0%,A ; add X(0)+X(–N/2–1)
   RPTZ  B,#(K_FIR_BFFR/2–1)
   FIRS  *SYMFIR_DATX_P+0%,*SYMFIR_DATY_P+0%,FIR_COFF
   MAR   *+SYMFIR_DATX_P(2)%                   ; to load the next newest sample
   MAR   *SYMFIR_DATY_P+%                      ; position for the X(–N/2) sample
   STH   B, *OUTBUF_P+
sym_fir_filter_loop
   RET
   .end
```

Example 15. Low-Pass Biquad IIR Filter

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    iir.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR:    Padma P. Mallela
;
;                   Application Specific Products
;                   Data Communication System Development
;                   12203 SW Freeway, MS 701
;                   Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /         AUTHORS                COMMENT
;          1.0     July–26–96  /        P.Mallela            original created
;
;}
;{
```

Example 15. Low-Pass Biquad IIR Filter (Continued)

```
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains two subroutines:
;        1) iir_init
;        2) iir_task
;    1.3 Specification/Design Reference (optional)
;        called by task.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s iir.asm
;
;    1.6 Notes and Special Considerations
;
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include   ”main.inc”
;    3.2 External Data
        .ref       d_iir_y
        .ref       d_iir_d
        .ref       iir_table_start,iir_coff_table
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
        .def    iir_init
        .def    iir_task
```

Example 15. Low-Pass Biquad IIR Filter (Continued)

```
;}
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 iir_init
;
;    5.1.1 Functional Description
;          A) This routine initializes buffers both for data and coeffs.
;----------------------------------------------------------------------------
;
;    5.1.2 Activation
;          Activation example:
;          CALL    iir_init
;          Reentrancy:     No
;          Recursive :     No
;
;    5.1.3 Inputs
;          NONE
;    5.1.4 Outputs
;          NONE
;    5.1.5 Global
;          NONE
;    5.1.6 Special considerations for data structure
;          –
;    5.1.7 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |UM|NU|NU|NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |UM |UM |NU |NU |UM|NU|NU|UM |UM|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.1.8 Execution
;          Execution time: ?cycles
;          Call rate:      not applicable for this application
;
;============================================================================
;HeaderEnd
;    5.1.9 Code
    .asg AR5,IIR_DATA_P                          ; data samples pointer
    .asg AR4,IIR_COFF_P                          ; IIR filter coffs pointer
    .sect   ”iir”
iir_init:
      STM   #iir_coff_table,IIR_COFF_P
      RPT   #K_IIR_SIZE–1                        ; move IIR coeffs from program
      MVPD  #iir_table_start,*IIR_COFF_P+        ; to data
      LD    #IIR_DP,DP
      STM   #d_iir_d,IIR_DATA_P                  ;AR5:d(n),d(n–1),d(n–2)
      RPTZ  A,#5                                 ;initial d(n),d(n–1),d(n–2)=0
      STL   A,*IIR_DATA_P+
      RET
;    5. SUBROUTINE CODE
;       HeaderBegin
;============================================================================
;
```

Example 15. Low-Pass Biquad IIR Filter (Continued)

```
;-------------------------------------------------------------------------
; 5.2 iir_task
;
; 5.2.1 Functional Description
;
;       This subroutine performs IIR filtering using biquad sections
;       IIR Low pass filter design
;       Filter type : Elliptic Filter
;       Filter order : 4 order (cascade: 2nd order + 2nd order)
;       cut freq. of pass band : 200 Hz
;       cut freq. of stop band : 500 Hz
;                              B0
;        ... -----> + ------> d(n) ---- x -> + -----....
;                |           |    |
;                |    A1     |        B1    |
;               + <- x -- d(n-1) -- x -> +
;                |           |    |
;                |    A2     |        B2    |
;               + <- x -- d(n-2) -- x -> +
;
;                    second order IIR
;-------------------------------------------------------------------------
;
;    5.2.2 Activation
;          Activation example:
;          CALL    iir_task
;          Reentrancy:    No
;          Recursive :    No
;
;    5.2.3 Inputs
;
;          Data structure:    AR6
;          Data Format:       16-bit input buffer pointer
;          Modified:          Yes
;          Description:       either point to PING/PONG buffer
;
;    5.2.4 Outputs
;
;          Data structure:    AR7
;          Data Format:       16-bit output buffer pointer
;          Modified:          Yes
;          Description:       either point to PING/PONG buffer
;
;    5.2.5 Global
;          Data structure:    AR1
;          Data Format:       16-bit index counter
;          Modified:          Yes
;          Description:       checks if 256 samples are processed
;
;          Data structure:    AR5
;          Data Format:       16-bit data buffer pointer
;          Modified:          Yes
;          Description:       inlcudes both feed forward and feedback paths
;
;          Data structure:    AR4
;          Data Format:       16-bit coefficient buffer pointer
;          Modified:          Yes
;          Description:       contains 2 biquad sections
;
```

Example 15. Low-Pass Biquad IIR Filter (Continued)

```
;          Data structure:    d_iir_y
;          Data Format:       16-bit variable
;          Modified:          Yes
;          Description:       holds the output of the 2 biquad sections
;    5.2.6 Special considerations for data structure
;          -
;    5.2.7 Entry and Exit conditions
;
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |U  |U  |UM|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |NU |UM |NU |NU |UM |UM |UM |UM |UM|NU|NU|UM |UM|NU |
;
; Note : UM - Used & Modified, U - Used, NU - Not Used
;
;    5.2.8 Execution
;          Execution time: ?cycles
;          Call rate:    not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.2.9 Code
      .asg        AR5,IIR_DATA_P              ; data samples pointer
      .asg        AR4,IIR_COFF_P              ; IIR filter coffs pointer
      .asg        AR6,INBUF_P
   .asg           AR7,OUTBUF_P
      .asg        AR1,IIR_INDEX_P
      .sect       "iir"
iir_task:
      STM      #K_FRAME_SIZE-1,BRC            ; Perform filtering for 256 samples
      RPTB     iir_filter_loop-1
      LD       *INBUF_P+,8,A                  ; load the input value
iir_filter:
      STM      #d_iir_d+5,IIR_DATA_P          ;AR5:d(n),d(n-1),d(n-2)
      STM      #iir_coff_table,IIR_COFF_P     ;AR4:coeff of IIR filter
A2,A1,B2,B1,B0
      STM      #K_BIQUAD-1,IIR_INDEX_P
feedback_path:
      MAC      *IIR_COFF_P+,*IIR_DATA_P-,A    ;input+d(n-2)*A2
      MAC      *IIR_COFF_P,*IIR_DATA_P,A      ;input+d(n-2)*A2+d(n-1)*A1/2
      MAC      *IIR_COFF_P+,*IIR_DATA_P-,A    ; A = A+d(n-1)*A1/2
      STH      A,*IIR_DATA_P+                 ;d(n) = input+d(n-2)*A2+d(n-1)*A1
      MAR      *IIR_DATA_P+
* Forward path
      MPY      *IIR_COFF_P+,*IIR_DATA_P-,A    ;d(n-2)*B2
      MAC      *IIR_COFF_P+,*IIR_DATA_P,A     ;d(n-2)*B2+d(n-1)*B1
      DELAY    *IIR_DATA_P-                   ;d(n-2)=d(n-1)
eloop:
      BANZD    feedback_path, *IIR_INDEX_P-
      MAC      *IIR_COFF_P+,*IIR_DATA_P,A     ;d(n-2)*B2+d(n-1)*B1+d(n)*B0
      DELAY    *IIR_DATA_P-                   ;d(n-1)=d(n)
      STH      A,d_iir_y                      ;output=d(n-2)*B2+d(n-1)*B1+d(n)*B0
      LD       d_iir_y,2,A                    ; scale the output
      STL      A, *OUTBUF_P+                  ; replace the oldest bffr value
iir_filter_loop
      RET
      .end
```

Example 16. Adaptive Filtering Using LMS Instruction

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
;
; Archives:    PVCS
; Filename:    adapt.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /    AUTHORS           COMMENT
;          1.0     July-24-96  /   P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains two subroutines:
;        1) adapt_init
;        2) adapt_task
;    1.3 Specification/Design Reference (optional)
;        called by task.asm depending upon the task
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s adapt.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include    ”main.inc”
;    3.2 External Data
;        .ref        ADAPT_DP
         .ref        d_mu,d_error,d_primary,d_output,d_mu,d_mu_e,d_new_x
         .ref        scoff,hcoff,wcoff
         .ref        xh,xw,d_adapt_count
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
       .def        adapt_init,adapt_task
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
; ---------------------------------------------------------------------------
;    5.1 adapt_init
;
;    5.1.1 Functional Description
;
;        This subroutine moves filter coeffcients from program to data space.
;        Initializes the adaptive coefficients, buffers,vars,and sets the circular
;        buffer address for processing.
;----------------------------------------------------------------------------
;
;    5.1.2 Activation
;          Activation example:
;          CALL    adapt_init
;          Reentrancy:     No
;          Recursive :     No
;
;    5.1.3 Inputs
;          NONE
;
;    5.1.4 Outputs
;          NONE
;
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
;    5.1.5 Global
;          Data structure: AR0
;          Data Format:    16–bit index pointer
;          Modified:       No
;          Description:    uses in circular addressing mode for indexing
;
;          Data structure: AR1
;          Data Format:    16–bit pointer
;          Modified:       Yes
;          Description:    used in initializing buffers and vars
;
;          Data structure: AR3
;          Data Format:    16–bit x(n) data buffer pointer for H(z)
;          Modified:       Yes
;          Description:    initializes the pointer
;
;          Data structure: AR5
;          Data Format:    16–bit x(n) data buffer pointer for W(z)
;          Modified:       Yes
;          Description:    initializes the pointer
;
;    5.1.6 Special considerations for data structure
;          –
;    5.1.7 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |UM|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |UM |UM |NU |UM |NU |UM |NU |NU |UM|UM|NU|NU |NU|NU |
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.1.8 Execution
;          Execution time: ?cycles
;          Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.1.9 Code
       .asg   AR0,INDEX_P
       .asg   AR1,INIT_P                    ; initialize buffer pointer
       .asg   AR3,XH_DATA_P                 ; data coeff buffer pointer
       .asg   AR5,XW_DATA_P                 ; data coeff buffer pointer
                                            ; for cal.y output
       .sect  ”filter”
adapt_init:
;   initialize input data location, input to hybrid, with Zero.
       STM    #xh,INIT_P
       RPTZ   A,#H_FILT_SIZE–1
       STL    A,*INIT_P+
;   initialize input data location, input to adaptive filter, with Zero.
       STM    #xw,INIT_P
       RPTZ   A,#ADPT_FILT_SIZE–1
       STL    A,*INIT_P+
;   initialize adaptive coefficient with Zero.
       STM    #wcoff,INIT_P
       RPTZ   A,#ADPT_FILT_SIZE–1
       STL    A,*INIT_P+
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
;   initialize temporary storage loacations with zero
       STM    #d_primary,INIT_P
       RPTZ   A,#6
       STL    A,*INIT_P+
;   copy system coefficient into RAM location, Rverse order
       STM    #hcoff,INIT_P
       RPT    #H_FILT_SIZE–1
       MVPD   #scoff,*INIT_P+
;      LD     #ADAPT_DP,DP                  ;set DP now and not worry about it
       ST     #K_mu,d_mu
       STM    #1,INDEX_P                    ; increment value to be used by
                                            ; dual address
;   associate auxilary registers for circular computation
       STM    #xh+H_FILT_SIZE–1,XH_DATA_P    ; last input of hybrid buffer
       RETD
       STM    #xw+ADPT_FILT_SIZE–1,XW_DATA_P ;last element of input buffer
;    5. SUBROUTINE CODE
;       HeaderBegin
 ;=============================================================================
;
;-----------------------------------------------------------------------------
;    5.2    adapt_task
;
;    5.2.1 Functional Description
;
;     This subroutine performs the adaptive filtering.The newest sample is
;     stored in a seperate location since filtering and adaptation are performed
;     at the same time. Otherwise the oldest sample is over written before
;     up dating the w(N–1) coefficient.
;
;     d_primary = xh *hcoff
;     d_output = xw *wcoff
;     LMS algorithm:
;        w(i+1) = w(i)+d*mu_error*xw(n–i) for i = 0,1,...127 and n = 0,1,.....
;     This program can run in two steps
;     1. Initial stepsize, d_mu = 0x0. At this point, the system is not
;     identified since the coefficients are not adapted and the error
;     signal e (n) is d (n). This is the default mode
;     2. At the EVM debugger command window change the step size
;     d_mu – 0x000, with the command e * d_mu = 0x1000
;     This changes the stepsize. The error signal e(n) in this case
;     is approximately 0 (theoretically) and the system is identified.
;-----------------------------------------------------------------------------;
;    5.2.2 Activation
;       Activation example:
;       CALL    adapt_task
;       Reentrancy:     No
;       Recursive :     No
;
;    5.2.3 Inputs
;
;        Data structure: AR3
;        Data Format:    16–bit x(n) data buffer pointer for H(Z)
;        Modified:       Yes
;        Description:    uses circular buffer addressing mode of size 128
;
;        Data structure: AR5
;        Data Format:    16–bit x(n) data buffer pointer for W(z)
;        Modified:       Yes
;        Description:    uses circular buffer addressing mode of size 128
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
;
;         Data structure: AR6
;         Data Format:    16–bit input buffer pointer
;         Modified:       Yes
;         Description:    either point to PING/PONG buffer
;
;     5.2.4 Outputs
;
;         Data structure: AR7
;         Data Format:    16–bit output buffer pointer
;         Modified:       Yes
;         Description:    either point to PING/PONG buffer
;
;     5.2.5 Global
;
;         Data structure: AR2
;         Data Format:    16–bit H(z) coeff buffer pointer
;         Modified:       Yes
;         Description:    uses circular buffer addressing mode of size 128
;
;         Data structure: AR4
;         Data Format:    16–bit W(z) coeff buffer pointer
;         Modified:       Yes
;         Description:    uses circular buffer addressing mode of size 128
;
;         Data structure: d_adapt_count
;         Data Format:    16–bit variable
;         Modified:       Yes
;         Description:    counter to check for processing 256 samples
;
;         Data structure: d_new_x
;         Data Format:    16–bit variable
;         Modified:       Yes
;         Description:    holds the newest sample
;
;         Data structure: d_primary
;         Data Format:    16–bit variable
;         Modified:       Yes
;         Description:    d_primary = xh * hcoeff
;
;         Data structure: d_output
;         Data Format:    16–bit variable
;         Modified:       Yes
;         Description:    d_output = xw * wcoff
;
;         Data structure: d_error
;         Data Format:    16–bit variable
;         Modified:       Yes
;         Description:    d_error = d_primary–d_output
;
;         Data structure: d_mu_e
;         Data Format:    16–bit variable
;         Modified:       Yes
;         Description:    d_mu_e = mu*d_error
;
;     5.2.6 Special considerations for data structure
;           –
;     5.2.7 Entry and Exit conditions
;
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC| T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;in  |U | 1 | 1 |NU | 1  |NU |U  |NU |NU |U  |NU | U |U  | U |UM |NU|NU|NU |NU|NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |   |   |  |   |  |   |
;out |U | 1 | 1 |NU | 1  |NU |U  |NU |UM |UM |UM |UM |UM |UM |UM |UM|UM|UM |UM|NU |
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.2.8 Execution
;          Execution time: ?cycles
;          Call rate:      not applicable for this application
;
;=============================================================================
;    HeaderEnd
; 5.2.9 Code
        .asg     AR2,H_COFF_P                  ; H(Z) coeff buffer pointer
        .asg     AR3,XH_DATA_P                 ; data coeff buffer pointer
        .asg     AR6,INBUF_P                   ; input buffer address pointer
        .asg     AR7,OUTBUF_P                  ; output buffer address pointer
                                               ; for cal. primary input
        .asg     AR4,W_COFF_P                  ; W(z) coeff buffer pointer
        .asg     AR5,XW_DATA_P                 ; data coeff buffer pointer
        .sect    ”filter”
adapt_task:
        STM      #H_FILT_SIZE,BK               ; first circular buffer size
        STM      #hcoff,H_COFF_P               ; H_COFF_P --> last of sys coeff
        ADDM     #1,d_adapt_count
        LD       *INBUF_P+, A                  ; load the input sample
        STM      #wcoff,W_COFF_P               ; reset coeff buffer
        STL      A,d_new_x                     ; read in new data
        LD       d_new_x,A                     ;
        STL      A,*XH_DATA_P+0%               ; store in the buffer
        RPTZ     A,#H_FILT_SIZE–1              ; Repeat 128 times
        MAC      *H_COFF_P+0%,*XH_DATA_P+0%,A  ; mult & acc:a = a + (h * x)
        STH      A,d_primary                   ; primary signal
;       start simultaneous filtering and updating the adaptive filter here.
        LD       d_mu_e,T                      ; T = step_size*error
        SUB      B,B                           ; zero acc B
        STM      #(ADPT_FILT_SIZE–2),BRC       ; set block repeat counter
        RPTBD    lms_end–1
        MPY      *XW_DATA_P+0%, A              ; error * oldest sample
        LMS      *W_COFF_P, *XW_DATA_P         ; B = filtered output (y)
                                               ; Update filter coeff
        ST       A, *W_COFF_P+                 ; save updated filter coeff
        ||       MPY*XW_DATA_P+0%,A            ; error *x[n–(N–1)]
        LMS      *W_COFF_P, *XW_DATA_P          ; B = accum filtered output y
                                               ; Update filter coeff
lms_end
        STH     A, *W_COFF_P                   ; final coeff
        MPY     *XW_DATA_P,A                   ; x(0)*h(0)
        MVKD    #d_new_x,*XW_DATA_P            ; store the newest sample
        LMS     *W_COFF_P,*XW_DATA_P+0%
        STH     B, d_output                    ; store the fitlered output
        LD      d_primary,A
        SUB     d_output,A
        STL     A, d_error                     ; store the residual error signal
        LD      d_mu,T
        MPY     d_error,A                      ; A=u*e
        STH     A,d_mu_e                       ; save the error *step_size
        LD      d_error,A                      ; residual error signal
        STL     A, *OUTBUF_P+
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
        LD      #K_FRAME_SIZE,A              ; check if a frame of samples
        SUB     d_adapt_count,A              ; have been processed
        BC      adapt_task,AGT
        RETD
        ST      #K_0,d_adapt_count           ; restore the count
       .end
* This is an input file used by the adaptive filter program.
* The transfer function is the system to be identifued by the adaptive filter
    .word    0FFFDh
    .word    24h
    .word    6h
    .word    0FFFDh
    .word    3h
    .word    3h
    .word    0FFE9h
    .word    7h
    .word    12h
    .word    1Ch
    .word    0FFF3h
    .word    0FFE8h
    .word    0Ch
    .word    3h
    .word    1Eh
    .word    1Ah
    .word    22h
    .word    0FFF5h
    .word    0FFE5h
    .word    0FFF1h
    .word    0FFC5h
    .word    0Ch
    .word    0FFE8h
    .word    37h
    .word    0FFE4h
    .word    0FFCAh
    .word    1Ch
    .word    0FFFDh
    .word    21h
    .word    0FFF7h
    .word    2Eh
    .word    28h
    .word    0FFC6h
    .word    53h
    .word    0FFB0h
    .word    55h
    .word    0FF36h
    .word    5h
    .word    0FFCFh
    .word    0FF99h
    .word    64h
    .word    41h
    .word    0FFF1h
    .word    0FFDFh
    .word    0D1h
    .word    6Ch
    .word    57h
    .word    36h
    .word    0A0h
    .word    0FEE3h
    .word    6h
    .word    0FEC5h
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
        .word   0ABh
        .word   185h
        .word   0FFF6h
        .word   93h
        .word   1Fh
        .word   10Eh
        .word   59h
        .word   0FEF0h
        .word   96h
        .word   0FFBFh
        .word   0FF47h
        .word   0FF76h
        .word   0FF0Bh
        .word   0FFAFh
        .word   14Bh
        .word   0FF3Bh
        .word   132h
        .word   289h
        .word   8Dh
        .word   0FE1Dh
        .word   0FE1Bh
        .word   0D4h
        .word   0FF69h
        .word   14Fh
        .word   2AAh
        .word   0FD43h
        .word   0F98Fh
        .word   451h
        .word   13Ch
        .word   0FEF7h
        .word   0FE36h
        .word   80h
        .word   0FFBBh
        .word   0FC8Eh
        .word   10Eh
        .word   37Dh
        .word   6FAh
        .word   1h
        .word   0FD89h
        .word   198h
        .word   0FE4Ch
        .word   0FE78h
        .word   0F215h
        .word   479h
        .word   749h
        .word   289h
        .word   0F667h
        .word   304h
        .word   5F8h
        .word   34Fh
        .word   47Bh
        .word   0FF7Fh
        .word   85Bh
        .word   0F837h
        .word   0F77Eh
        .word   0FF80h
        .word   0B9Bh
        .word   0F03Ah
        .word   0EE66h
        .word   0FE28h
```

Example 16. Adaptive Filtering Using LMS Instruction (Continued)

```
    .word    0FAD0h
    .word    8C3h
    .word    0F5D6h
    .word    14DCh
    .word    0F3A7h
    .word    0E542h
    .word    10F2h
    .word    566h
    .word    26AAh
    .word    15Ah
    .word    2853h
    .word    0EE95h
    .word    93Dh
    .word    20Dh
    .word    1230h
    .word    238Ah
```

Example 17. 256-Point Real FFT Initialization

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    rfft.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Simon Lau and Nathan Baltz
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /      AUTHORS             COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains core routine:
;        rfft
;
;    1.3 Specification/Design Reference (optional)
;
```

Example 17. 256-Point Real FFT Initialization (Continued)

```
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s rfft.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include        ”main.inc”
        .include        ”init_54x.inc”
;    3.2 External Data
        .ref            bit_rev, fft, unpack
        .ref            power
        .ref            sine,cosine
        .ref            sine_table,cos_table
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
        .def            rfft_task
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 rfft
;
;    5.2 Functional Description
;        The following code implements a Radix–2, DIT, 2N–point Real FFT for the
;        TMS320C54x.  This main program makes four function calls, each
```

Example 17. 256-Point Real FFT Initialization (Continued)

```
;        corresponds to a different phase of the algorithm. For more details about
;        how each phase is implemented, see bit_rev.asm, fft.asm, unpack.asm, and
;        power.asm assembly files.
;-------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    rfft
;        Reentrancy:     No
;        Recursive :     No
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;
;        Data structure: AR1
;        Data Format:    16–bit pointer
;        Modified:       No
;        Description:    used for moving the twiddle tables from
;                        program to data
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;out|U | 1 | 1 |NU | 1  |U  |NU |UM |NU |NU |NU |NU |NU |NU |NU|NU|NU|UM |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;          Execution time: ?cycles
;          Call rate:      not applicable for this application
;
;=========================================================================
;HeaderEnd
;    5.10 Code
          .asg    AR1,FFT_TWID_P
          .sect   ”rfft_prg”
rfft_task:
          STM     #sine,FFT_TWID_P
          RPT     #K_FFT_SIZE–1                ; move FIR coeffs from program
          MVPD    #sine_table,*FFT_TWID_P+     ; to data
          STM     #cosine,FFT_TWID_P
          RPT     #K_FFT_SIZE–1                ; move FIR coeffs from program
          MVPD    #cos_table,*FFT_TWID_P+      ; to data
          CALL    bit_rev
          CALL    fft
          CALL    unpack
          CALLD   power
          STM     #K_ST1,ST1                   ; restore the original contents of
                                               ; ST1 since ASM field has changed
          RET                                  ; return to main program
          .end
```

Example 18. Bit Reversal Routine

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP ;
;
; Archives:    PVCS
; Filename:    bit_rev.asm
; Version:     1.0
; Status :     draft         ( )
;              proposal      (X)
;              accepted      ( ) dd-mm-yy/?acceptor.
;
; AUTHOR     Simon Lau and Nathan Baltz
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutine:
;        bit_rev
;
;    1.3 Specification/Design Reference (optional)
;        called by rfft.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s bit_rev.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
;    2.2 Local Compiler Flags
;        -
```

Example 18. Bit Reversal Routine (Continued)

```
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include    ”main.inc”
;    3.2 External Data
         .ref        d_input_addr, fft_data
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def        bit_rev
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 bit_rev
;
;    5.2 Functional Description
;        This function is called from the main module of the ’C54x Real FFT code.
;        It reorders the original 2N–point real input sequence by using
;        bit–reversed addressing.  This new sequence is stored into the data
;        processing buffer of size 2N, where FFT will be performed in–place
;        during Phase Two.
;
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    bit_rev
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;
;        Data structure: AR0
;        Data Format:    16–bit index pointer
;        Modified:       No
;        Description:    used for bit reversed addressing
```

Example 18. Bit Reversal Routine (Continued)

```
;
;        Data structure: AR2
;        Data Format:    16–bit pointer
;        Modified:       Yes
;        Description:    pointer to processed data in bit–reversed order
;
;        Data structure: AR3
;        Data Format:    16–bit pointer
;        Modified:       Yes
;        Description:    pointer to original input data in natural order
;
;        Data structure: AR7
;        Data Format:    16–bit pointer
;        Modified:       Yes
;        Description:    starting addressing of data processing buffer
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |UM |NU |UM |UM |NU |NU |NU |UM |NU|NU|NU|UM |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.10 Code
      .asg    AR2,REORDERED_DATA
      .asg    AR3,ORIGINAL_INPUT
      .asg    AR7,DATA_PROC_BUF
      .sect   ”rfft_prg”
bit_rev:
      SSBX    FRCT                                ; fractional mode is on
      MVDK    d_input_addr,ORIGINAL_INPUT         ; AR3 –> 1 st original input
      STM     #fft_data,DATA_PROC_BUF             ; AR7 –> data processing buffer
      MVMM    DATA_PROC_BUF,REORDERED_DATA        ; AR2 –> 1st bit–reversed data
      STM     #K_FFT_SIZE–1,BRC
      RPTBD   bit_rev_end–1
      STM     #K_FFT_SIZE,AR0                     ; AR0 = 1/2 size of circ buffer
      MVDD    *ORIGINAL_INPUT+,*REORDERED_DATA+
      MVDD    *ORIGINAL_INPUT–,*REORDERED_DATA+
      MAR     *ORIGINAL_INPUT+0B
bit_rev_end:
      RET                                         ; return to Real FFT main module
      end
```

Example 19. 256-Point Real FFT Routine

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:   PVCS
; Filename:   fft.asm
; Version:    1.0
; Status :    draft         ( )
;             proposal      (X)
;             accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR    Simon Lau and Nathan Baltz
;
;             Application Specific Products
;             Data Communication System Development
;             12203 SW Freeway, MS 701
;             Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July–17–96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutine:
;        fft
;
;    1.3 Specification/Design Reference (optional)
;        called by rfft.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s fft.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;
```

Example 19. 256-Point Real FFT Routine (Continued)

```
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include    ”main.inc”
;    3.2 External Data
         .ref        fft_data, d_grps_cnt, d_twid_idx, d_data_idx, sine, cosine
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def        fft
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 fft
;
;    5.2 Functional Description
;        PHASE TWO   (LogN)–Stage Complex FFT
;        This function is called from main module of the ’C54x Real FFT code.
;        Here we assume that the original 2N–point real input sequence is al
;        ready packed into an N–point complex sequence and stored into the
;        data processing buffer in bit–reversed order (as done in Phase One).
;        Now we perform an in–place, N–point complex FFT on the data proces
;        sing buffer, dividing the outputs by 2 at the end of each stage to
;        prevent overflow. The resulting N–point complex sequence will be un–
;        packed into a 2N–point complex sequencein Phase Three & Four.
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    fft
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;
;        Data structure:    AR0
```

Example 19. 256-Point Real FFT Routine (Continued)

```
;        Data Format:       16–bit index pointer
;        Modified:          No
;        Description:       index to twiddle tables
;
;        Data structure:    AR1
;        Data Format:       16–bit counter
;        Modified:          No
;        Description:       group counter
;
;        Data structure:    AR2
;        Data Format:       16–bit pointer
;        Modified:          Yes
;        Description:       pointer to 1st butterfly data PR,PI
;
;        Data structure:    AR3
;        Data Format:       16–bit pointer
;        Modified:          Yes
;        Description:       pointer to 2nd butterfly data QR,QI
;
;        Data structure:   AR4
;        Data Format:      16–bit pointer
;        Modified:         Yes
;        Description:      pointer to cosine value WR
;
;        Data structure:   AR5
;        Data Format:      16–bit pointer
;        Modified:         Yes
;        Description:      pointer to cosine value WI
;
;        Data structure:   AR6
;        Data Format:      16–bit counter
;        Modified:         Yes
;        Description:      butterfly counter
;
;        Data structure:   AR7
;        Data Format:      16–bit pointer
;        Modified:         Yes
;        Description:      start address of data processing buffer
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  | 0 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |–1 |UM |UM |UM |UM |UM |UM |UM |UM |UM|UM|UM|UM |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate: not applicable for this application
;
;=============================================================================
;    HeaderEnd
;    5.10 Code
          .asg     AR1,GROUP_COUNTER
          .asg     AR2,PX
```

Example 19. 256-Point Real FFT Routine (Continued)

```
         .asg      AR3,QX
         .asg     AR4,WR
         .asg     AR5,WI
         .asg      AR6,BUTTERFLY_COUNTER
         .asg     AR7,DATA_PROC_BUF                ; for Stages 1 & 2
         .asg     AR7,STAGE_COUNTER                ; for the remaining stages
         .sect    "rfft_prg"
fft:
;   Stage 1 ------------------------------------------------------------------
         STM      #K_ZERO_BK,BK                    ; BK=0 so that *ARn+0% == *ARn+0
         LD       #-1,ASM                          ; outputs div by 2 at each stage
         MVMM     DATA_PROC_BUF,PX                 ; PX -> PR
         LD       *PX,A                            ; A  :=  PR
         STM      #fft_data+K_DATA_IDX_1,QX        ; QX -> QR
         STM      #K_FFT_SIZE/2-1,BRC
         RPTBD    stage1end-1
         STM      #K_DATA_IDX_1+1,AR0
         SUB      *QX,16,A,B                       ; B  :=  PR-QR
         ADD      *QX,16,A                         ; A  :=  PR+QR
         STH      A,ASM,*PX+                       ; PR':= (PR+QR)/2
         ST       B,*QX+                           ; QR':= (PR-QR)/2
         ||LD     *PX,A                            ; A  :=  PI
         SUB      *QX,16,A,B                       ; B  :=  PI-QI
         ADD      *QX,16,A                         ; A  :=  PI+QI
         STH      A,ASM,*PX+0                      ; PI':= (PI+QI)/2
         ST       B,*QX+0%                         ; QI':= (PI-QI)/2
         ||LD     *PX,A                            ; A  :=  next PR
stage1end:
;   Stage 2 ------------------------------------------------------------------
         MVMM        DATA_PROC_BUF,PX              ; PX -> PR
         STM         #fft_data+K_DATA_IDX_2,QX     ; QX -> QR
         STM         #K_FFT_SIZE/4-1,BRC
         LD          *PX,A                         ; A  :=  PR
         RPTBD       stage2end-1
         STM         #K_DATA_IDX_2+1,AR0
; 1st butterfly
         SUB         *QX,16,A,B                    ; B  :=  PR-QR
         ADD         *QX,16,A                      ; A  :=  PR+QR
         STH         A,ASM,*PX+                    ; PR':= (PR+QR)/2
         ST          B,*QX+                        ; QR':= (PR-QR)/2
         ||LD        *PX,A                         ; A  :=  PI
         SUB         *QX,16,A,B                    ; B  :=  PI-QI
         ADD         *QX,16,A                      ; A  :=  PI+QI
         STH         A,ASM,*PX+                    ; PI':= (PI+QI)/2
         STH         B,ASM,*QX+                    ; QI':= (PI-QI)/2
; 2nd butterfly
         MAR         *QX+
         ADD         *PX,*QX,A                     ; A  :=  PR+QI
         SUB         *PX,*QX-,B                    ; B  :=  PR-QI
         STH         A,ASM,*PX+                    ; PR':= (PR+QI)/2
         SUB         *PX,*QX,A                     ; A  :=  PI-QR
         ST          B,*QX                         ; QR':= (PR-QI)/2
         ||LD        *QX+,B                        ; B  :=     QR
            ST       A, *PX                        ; PI':= (PI-QR)/2
         ||ADD       *PX+0%,A                      ; A  :=  PI+QR
         ST          A,*QX+0%                      ; QI':= (PI+QR)/2
         ||LD        *PX,A                         ; A  :=  PR
stage2end:
; Stage 3 thru Stage logN-1 ------------------------------------------------
      STM   #K_TWID_TBL_SIZE,BK                    ; BK = twiddle table size always
```

Example 19. 256-Point Real FFT Routine (Continued)

```
      ST    #K_TWID_IDX_3,d_twid_idx              ; init index of twiddle table
      STM   #K_TWID_IDX_3,AR0                     ; AR0 = index of twiddle table
      STM   #cosine,WR                            ; init WR pointer
      STM   #sine,WI                              ; init WI pointer
      STM   #K_LOGN-2-1,STAGE_COUNTER             ; init stage counter
      ST    #K_FFT_SIZE/8-1,d_grps_cnt            ; init group counter
      STM   #K_FLY_COUNT_3-1,BUTTERFLY_COUNTER    ; init butterfly counter
      ST    #K_DATA_IDX_3,d_data_idx              ; init index for input data
stage:
      STM   #fft_data,PX                          ; PX -> PR
      LD    d_data_idx, A
      ADD   *(PX),A
      STLM  A,QX                                  ; QX -> QR
      MVDK  d_grps_cnt,GROUP_COUNTER              ; AR1 contains group counter
group:
      MVMD  BUTTERFLY_COUNTER,BRC                 ; # of butterflies in each grp
      RPTBD butterflyend-1
      LD    *WR,T                                 ; T  :=  WR
      MPY   *QX+,A                                ; A  :=  QR*WR  || QX->QI
      MACR  *WI+0%,*QX-,A                         ; A  :=  QR*WR+QI*WI
                                                  ; || QX->QR
      ADD   *PX,16,A,B                            ; B  := (QR*WR+QI*WI)+PR
      ST    B,*PX                                 ; PR':=((QR*WR+QI*WI)+PR)/2
      ||SUB *PX+,B                                ; B  :=  PR-(QR*WR+QI*WI)
                                                  ; || PX->PI
      ST    B,*QX                                 ; QR':= (PR-(QR*WR+QI*WI))/2
      ||MPY *QX+,A                                ; A  :=  QR*WI [T=WI]
                                                  ; || QX->QI
      MASR  *QX,*WR+0%,A                          ; A  :=  QR*WI-QI*WR
      ADD   *PX,16,A,B                            ; B  := (QR*WI-QI*WR)+PI
      ST    B,*QX+                                ; QI':=((QR*WI-QI*WR)+PI)/2
                                                  ; || QX->QR
      ||SUB *PX,B                                 ; B  :=  PI-(QR*WI-QI*WR)
      LD    *WR,T                                 ; T  :=  WR
      ST    B,*PX+                                ; PI':= (PI-(QR*WI-QI*WR))/2
                                                  ; || PX->PR
      ||MPY *QX+,A                                ; A  :=  QR*WR  || QX->QI
butterflyend:
; Update pointers for next group
      PSHM  AR0                                   ; preserve AR0
      MVDK  d_data_idx,AR0
     MAR    *PX+0                                 ; increment PX for next group
     MAR    *QX+0                                 ; increment QX for next group
     BANZD   group,*GROUP_COUNTER-
     POPM   AR0                                   ; restore AR0
     MAR    *QX-
; Update counters and indices for next stage
     LD     d_data_idx,A
     SUB    #1,A,B                                ; B = A-1
     STLM   B,BUTTERFLY_COUNTER                   ; BUTTERFLY_COUNTER = #flies-1
     STL    A,1,d_data_idx                        ; double the index of data
     LD     d_grps_cnt,A
     STL    A,ASM,d_grps_cnt                      ; 1/2 the offset to next group
     LD     d_twid_idx,A
     STL    A,ASM,d_twid_idx                      ; 1/2 the index of twiddle table
     BANZ   D stage,*STAGE_COUNTER-
     MVDK   d_twid_idx,AR0                        ; AR0 = index of twiddle table
fft_end:
     RET                                          ; return to Real FFT main module
    .end
```

Example 20. Unpack 256-Point Real FFT Output

```
;    TEXAS INSTRUMENTS INCORPORATED
;    DSP Data Communication System Development / ASP
;    Archives:    PVCS
;    Filename:    unpack.asm
;    Version:     1.0
;    Status :     draft         ( )
;                 proposal      (X)
;                 accepted      ( ) dd-mm-yy/?acceptor.
;
;    AUTHOR    Simon Lau and Nathan Baltz
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
;    IPR statements description (can be collected).
;}
;    (C) Copyright 1996. Texas Instruments. All rights reserved.
;{
;    Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutine:
;        unpack
;
;    1.3 Specification/Design Reference (optional)
;        called by rfft.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;

;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s unpack.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
;    2.2 Local Compiler Flags
;        -
```

Example 20. Unpack 256-Point Real FFT Output (Continued)

```
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include        ”main.inc”
;    3.2 External Data
         .ref            fft_data,sine, cosine
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def        unpack
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;    5.1  unpack
;
;    5.2 Functional Description
;
;        PHASE THREE & FOUR      Unpacking to 2N Outputs
;        This function is called from the main module of the ’C54x Real FFT
;        code. It first computes four intermediate sequences (RP,RM, IP, IM)
;        from the resulting complex sequence at the end of the previous phase.
;        Next, it uses the four intermediate sequences to form the FFT of the
;        original 2N–point real input. Again, the outputs are divided by 2 to
;        prevent overflow
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;          CALL    unpack
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;
```

Example 20. Unpack 256-Point Real FFT Output (Continued)

```
;    5.6.1 Phase Three Global
;
;           Data structure:    AR0
;           Data Format:       16–bit index pointer
;           Modified:          No
;           Description:       index to twiddle tables
;
;           Data structure:    AR2
;           Data Format:       16–bit pointer
;           Modified:          Yes
;           Description:       pointer to R[k], I[k], RP[k], IP[k]
;
;           Data structure:    AR3
;           Data Format:       16–bit pointer
;           Modified:          Yes
;           Description:       pointer to R[N–k], I[N–k], RP[N–k], IP[N–k]
;
;           Data structure:    AR6
;           Data Format:       16–bit pointer
;           Modified:          Yes
;           Description:       pointer to RM[k], IM[k]
;
;           Data structure:    AR7
;           Data Format:       16–bit pointer
;           Modified:          Yes
;           Description:       pointer to RM[n–k], IM[n–k]
;
;    5.6.2 Phase Four Global
;
;           Data structure: AR0
;           Data Format:       16–bit index pointer
;           Modified:          No
;           Description:       index to twiddle tables
;
;           Data structure: AR2
;           Data Format:       16–bit counter
;           Modified:          No
;           Description:       pointer to RP[k], IP[k], AR[k], AI[k], AR[0]
;
;           Data structure: AR3
;           Data Format:    16–bit pointer
;           Modified:       Yes
;           Description:    pointer to RM[k], IM[k], AR[2N–k], AI[2N–k]
;
;           Data structure: AR4
;           Data Format:    16–bit pointer
;           Modified:       Yes
;           Description:    pointer to cos(k*pi/N), AI[0]
;
;           Data structure: AR5
;           Data Format:    16–bit pointer
;           Modified:       Yes
;           Description:    pointer to sin(k*pi/N), AR[N], AI[N]
;
;    5.7 Special considerations for data structure
;        –
```

Example 20. Unpack 256-Point Real FFT Output (Continued)

```
;    5.8 Entry and Exit conditions
;
;    5.8.1 Phase Three Entry and Exit Conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;in |2 | 1 | 1 | 0 | 1  | 0 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|0 |NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;out|2 | 1 | 1 | 0 | 1  |–1 |UM |NU |UM |UM |NU |NU |UM |UM |UM|UM|UM|UM |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.8.2 Phase Four Entry and Exit Conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;in |U | 1 | 1 | 0 | 1  |–1 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |  |
;out|U | 1 | 1 | 0 | 1  |–1 |UM |NU |UM |UM |UM |UM |NU |NU |UM|UM|UM|UM |NU|NU
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;         HeaderEnd
;    5.10  Code
    .sect        ”rfft_prg”
unpack:
; Compute intermediate values RP, RM, IP, IM
    .asg    AR2,XP_k
    .asg    AR3,XP_Nminusk
    .asg    AR6,XM_k
    .asg    AR7,XM_Nminusk
     STM    #fft_data+2,XP_k                       ; AR2 –> R[k] (temp RP[k])
     STM    #fft_data+2*K_FFT_SIZE–2,XP_Nminusk    ; AR3 –> R[N–k] (temp
                                                           RP[N–k])
     STM    #fft_data+2*K_FFT_SIZE+3,XM_Nminusk    ; AR7 –> temp RM[N–k]
     STM    #fft_data+4*K_FFT_SIZE–1,XM_k          ; AR6 –> temp RM[k]
     STM    #–2+K_FFT_SIZE/2,BRC
     RPTBD  phase3end–1
     STM    #3,AR0
     ADD    *XP_k,*XP_Nminusk,A                    ; A := R[k]+R[N–k] =
                                                           2*RP[k]
     SUB    *XP_k,*XP_Nminusk,B                    ; B := R[k]–R[N–k] =
                                                           2*RM[k]
     STH    A,ASM,*XP_k+                           ; store RP[k] at AR[k]
     STH    A,ASM,*XP_Nminusk+                     ; store RP[N–k]=RP[k] at
                                                           AR[N–k]
     STH    B,ASM,*XM_k–                           ; store RM[k] at AI[2N–k]
     NEG    B                                      ; B := R[N–k]–R[k] =
                                                           2*RM[N–k]
     STH    B,ASM,*XM_Nminusk–                     ; store RM[N–k] at AI[N+k]
     ADD    *XP_k,*XP_Nminusk,A                    ; A := I[k]+I[N–k] =
                                                           2*IP[k]
     SUB    *XP_k,*XP_Nminusk,B                    ; B := I[k]–I[N–k] =
                                                           2*IM[k]
```

Example 20. Unpack 256-Point Real FFT Output (Continued)

```
     STH    A,ASM,*XP_k+                          ; store IP[k] at AI[k]
     STH    A,ASM,*XP_Nminusk-0                   ; store IP[N-k]=IP[k] at
                                                          AI[N-k]
     STH    B,ASM,*XM_k-                          ; store IM[k] at AR[2N-k]
     NEG    B                                     ; B := I[N-k]-I[k] =
                                                          2*IM[N-k]
     STH    B,ASM,*XM_Nminusk+0                   ; store IM[N-k] at AR[N+k]
phase3end:
     ST     #0,*XM_k-                             ; RM[N/2]=0
     ST     #0,*XM_k                              ; IM[N/2]=0
; Compute AR[0],AI[0], AR[N], AI[N]
     .asg   AR2,AX_k
     .asg   AR4,IP_0
     .asg   AR5,AX_N
     STM    #fft_data,AX_k                        ; AR2 -> AR[0] (temp
                                                          RP[0])
     STM    #fft_data+1,IP_0                      ; AR4 -> AI[0] (temp
                                                          IP[0])
     STM    #fft_data+2*K_FFT_SIZE+1,AX_N         ; AR5 -> AI[N]
     ADD    *AX_k,*IP_0,A                         ; A := RP[0]+IP[0]
     SUB    *AX_k,*IP_0,B                         ; B := RP[0]-IP[0]
     STH    A,ASM,*AX_k+                          ; AR[0] = (RP[0]+IP[0])/2
     ST     #0,*AX_k                                  ; AI[0] = 0
     MVDD   *AX_k+,*AX_N-                             ; AI[N] = 0
     STH     B,ASM,*AX_N                              ; AR[N] = (RP[0]-IP[0])/2
; Compute final output values AR[k], AI[k]
     .asg     AR3,AX_2Nminusk
     .asg     AR4,COS
     .asg     AR5,SIN
     STM      #fft_data+4*K_FFT_SIZE-1,AX_2Nminusk    ; AR3 -> AI[2N-1]
                                                          (temp RM[1])
     STM      #cosine+K_TWID_TBL_SIZE/K_FFT_SIZE,COS  ; AR4 -> cos(k*pi/N)
     STM      #sine+K_TWID_TBL_SIZE/K_FFT_SIZE,SIN    ; AR5 -> sin(k*pi/N)
     STM      #K_FFT_SIZE-2,BRC
     RPTBD    phase4end-1
     STM      #K_TWID_TBL_SIZE/K_FFT_SIZE,AR0         ; index of twiddle
                                                          tables
     LD       *AX_k+,16,A                             ; A := RP[k] ||
                                                          AR2->IP[k]
     MACR     *COS,*AX_k,A                            ; A :=A+cos(k*pi/N)
                                                           *IP[k]
     MASR     *SIN,*AX_2Nminusk-,A                    ; A := A-sin(k*pi/N)
                                                          *RM[k]
                                                      ; || AR3->IM[k]
     LD       *AX_2Nminusk+,16,B                      ; B := IM[k] ||
                                                          AR3->RM[k]
     MASR     *SIN+0%,*AX_k-,B                        ; B := B-sin(k*pi/N)
                                                          *IP[k]
                                                      ; || AR2->RP[k]
     MASR     *COS+0%,*AX_2Nminusk,B                  ; B := B-cos(k*pi/N)
                                                          *RM[k]
     STH      A,ASM,*AX_k+                            ; AR[k] = A/2
     STH      B,ASM,*AX_k+                            ; AI[k] = B/2
     NEG      B                                       ; B := -B
     STH      B,ASM,*AX_2Nminusk-                     ; AI[2N-k] = -AI[k]
                                                            = B/2
     STH      A,ASM,*AX_2Nminusk-                     ; AR[2N-k] = AR
                                                          [k] = A/2
phase4end:
     RET                                              ; returntoRealFFTmain module
     .end
```

Example 21. Compute the Power Spectrum of the Complex Output of the 256-Point Real FFT

```
;    TEXAS INSTRUMENTS INCORPORATED
;    DSP Data Communication System Development / ASP
;
;    Archives:    PVCS
;    Filename:    power.asm
;    Version:     1.0
;    Status :     draft         ( )
;                 proposal      (X)
;                 accepted      ( ) dd-mm-yy/?acceptor.
;
;    AUTHOR    Simon Lau and Nathan Baltz
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
;    IPR statements description (can be collected).
;}
;   (C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
;    Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutine:
;        power
;
;    1.3 Specification/Design Reference (optional)
;        called by rfft.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s power.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
;    2.2 Local Compiler Flags
;        -
;
```

Example 21. Compute the Power Spectrum of the Complex Output of the 256-Point Real FFT (Continued)

```
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include     ”main.inc”
;    3.2 External Data
         .ref        fft_data, d_output_addr
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
         .def        power
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 power
;
;    5.2 Functional Description
;        PHASE FIVE  Power Spectrum
;        This function is called from the main module of the ’C54x Real FFT
;        code. It computes the power spectrum of the Real FFT output.
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        CALL    power
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;        Data structure: AR2
;        Data Format:    16–bit pointer
;        Modified:       Yes
;        Description:    pointer to AR[k], AI[k]
;
```

Example 21. Compute the Power Spectrum of the Complex Output of the 256-Point Real FFT (Continued)

```
;        Data structure: AR3
;        Data Format:    16–bit pointer
;        Modified:       Yes
;        Description:    pointer to output buffer
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |UM |UM |NU |NU |NU |NU |UM|NU|NU|UM |NU|NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:    not applicable for this application
;
;=============================================================================
;    HeaderEnd
;    5.10 Code
         .asg       AR2,AX
         .asg       AR3,OUTPUT_BUF
         .sect      ”pwr_prog” power:
         MVDK       d_output_addr,OUTPUT_BUF      ; AR3 points to output buffer
           STM          #K_FFT_SIZE*2–1,BRC
         RPTBD      power_end–1
         STM        #fft_data,AX                  ; AR2 points to AR[0]
         SQUR       *AX+,A                        ; A := AR^2
         SQURA      *AX+,A                        ; A := AR^2 + AI^2
         STH        A,*OUTPUT_BUF+
   power_end:
         RET                                      ; return to main program
         .end
```

Example 22. Data Transfer from FIFO

```
;    TEXAS INSTRUMENTS INCORPORATED
;    DSP Data Communication System Development / ASP
;
;    Archives:    PVCS
;    Filename:    fifo.asm
;    Version:     1.0
;    Status :     draft         ( )
;                 proposal      (X)
;                 accepted      ( ) dd–mm–yy/?acceptor.
;
;    AUTHOR    Padma P. Mallela
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
;    IPR statements description (can be collected).
;}
;    (C) Copyright 1996. Texas Instruments. All rights reserved.
```

Example 22. Data Transfer from FIFO (Continued)

```
;
;{
;    Change history:
;
;        VERSION      DATE     /     AUTHORS           COMMENT
;          1.0     July–25–96  /    P.Mallela      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutines:
;        fifo_host_transfer
;    1.3 Specification/Design Reference (optional)
;        called by main.asm depending upon if K_HOST_FLAG is set
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s fifo.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include        ”target.inc”
;    3.2 External Data
        .ref            d_command_reg
        .ref            d_fifo_count
        .ref            d_command_value
        .ref            d_fifo_ptr
        .ref            d_output_addr
;    3.3 Import Functions
;}
;{
```

Example 22. Data Transfer from FIFO (Continued)

```
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
;        .def            fifo_host_transfer
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;============================================================================
;
;----------------------------------------------------------------------------
;    5.1 fifo_host_transfer
;
;    5.2 Functional Description
;        This routine transfers a FIFO(64) of data to host thru CH B.
;        In the process, after transferring data from DSP to FIFO sends a com–
;        mand to host thru CH A.The host acknowledges and sends a command to
;        target (DSP) thru CH A. The host transfer can be disabled by setting
;        the K_HOST_FLAG =0
;----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;                     CALL    fifo_host_transfer
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;                      Data structure: d_output_addr
;                      Data Format:    16–bit variable
;                      Modified:       NO
;    Description:   holds the starting addr of either PING/PONG addr.
;
;                      Data structure: d_fifo_count
;                      Data Format:    16–bit var
;                      Modified:       Yes
;                      Description:    counter for # of transfers
;
;                      Data structure: d_fifo_ptr
;                      Data Format:    16–bit variable
;                      Modified:       Yes
;                      Description: holds the output bffr addr. and incremented by
;                     32 for every transfer
;
;    5.5 Outputs
;
;                      Data structure: AR7
;                      Data Format:    16–bit output buffer pointer
;                      Modified:       Yes
;                      Description:    either point to PING/PONG buffer
;
```

Example 22. Data Transfer from FIFO (Continued)

```
;    5.6 Global
;                    Data structure: d_command_reg
;                    Data Format:    16-bit variable
;                    Modified:       Yes
;                    Description:    command from host is read thru CH A
;                    Data structure: d_command_value
;                    Data Format:    16-bit variable
;                    Modified:       Yes
;                    Description:    holds the command value
;
;    5.7 Special considerations for data structure
;        -
;
;    5.8 Entry and Exit conditions
;
;   |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;   |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |   |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |U  |NU|NU|NU|NU |NU|NU |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |U  |UM|NU|NU|NU |NU|NU |
;
;    Note : UM - Used & Modified, U - Used, NU - Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;HeaderEnd
;    5.10 Code
;         fifo_host_transfer:
;         LD       #FIFO_DP,DP
          .if      K_HOST_FLAG =1
          PORTR    K_TRGCR_ADDR,d_command_reg    ; while (port14 & BXST)
          BITF     d_command_reg,K_BXST
          BC       fifo_discard,TC               ; FIFO discard
          MVDK     d_output_addr,OUTBUF_P        ; load PING/PONG bffr address
          RPT      #K_FIFO_SIZE-1                ; write first set of 64 data
                                                 ; to FIFO
          PORTW    *OUTBUF_P+,K_CHB              ; Fill FIFO
          ST       #K_FIFO_FULL,d_command_value
          PORTW    d_command_value, K_CHA        ; write command to comnd reg A
          ST       #1,d_fifo_count               ; start counting for tranfers
          MVKD    OUTBUF_P,d_fifo_ptr            ; save the fifo_ptr
fifo_discard
          .endif
          RET
          .end
*****************************************************************************
* This file includes the TCR register configuration of EVM
*****************************************************************************
K_AIC_RST             .set     0b << 15       ; if AICRST=0, aic is reset
K_USR_BOT             .set     000b << 12     ; User discrete output bits
                                              ; 0,1,2
K_RESRV               .set     0000b << 8     ; Reserved bits
K_USR_BIN             .set     00b << 6       ; User discrete input bits 0,1
K_RCV_BRST            .set     00b << 4       ; Channel B receive status regs
                                              ; buffer half or more
K_XMT_BXST            .set     11b << 2       ; Ch B trasnmit status register
                                              ; buffer half or more
```

Example 22. Data Transfer from FIFO (Continued)

```
K_RCV_ARST          .set    0b << 1         ; Ch A receive register
K_XMT_AXST          .set    0b << 1         ; Ch A transmit register
K_TCR_HIGH          .set    K_AIC_RST|K_USR_BOT|K_RESRV
K_TCR_LOW           .set    K_USR_BIN|K_RCV_BRST|K_XMT_BXST|K_RCV_ARST|K_XMT_AXST
K_TCR               .set    K_TCR_HIGH|K_TCR_LOW
*******************************************************************************
* this includes I/O address of CH_A, CH_B and different commands that's been
* passed between host and the target
*******************************************************************************
K_0                 .set    0h              ; constant 0
K_FIFO_FULL         .set    0xFF            ; Full FIFO command written by
                                            ; target
K_FIFO_EMPTY        .set    0xEE            ; Empty FIFO command
                                              ; written by host
K_AXST_CLEAR        .set    0xAE            ; Clear AXST empty command
                                            ; written by the target
K_HANDSHAKE_CMD     .set    0xAB            ; handshake CMD written by host
K_CHB               .set    12h             ; Use Channel B as I/O interface
                                            ; to 54x EVM for sending data
K_CHA               .set    10h             ; Use Channel A as I/O interface
                                            ; to 54x EVM for send command
                                            ; to host
K_TRGCR_ADDR        .set    14h               ; Target status control register
                                             I/O address location
K_AXST              .set    1h              ; 0h
K_ARST              .set    2h              ; used to check the control bits
K_BXST              .set    3h              ; check if K_FIFO_SIZE
                    .set    64              ; its a 64 FIFO
K_FRAME_SIZE        .set    256             ; Frame size
K_HOST_FLAG         .set    1               ; if 0, then host interface
                                            ; is disabled
```

Example 23. Interrupt 1 Service Routine

```
;    TEXAS INSTRUMENTS INCORPORATED
;    DSP Data Communication System Development / ASP
;
;    Archives:    PVCS
;    Filename:    hst_int1.asm
;    Version:     1.0
;    Status :     draft         ( )
;                 proposal      (X)
;                 accepted      ( ) dd-mm-yy/?acceptor.
;
;    AUTHOR    Padma P. Mallela
;
;             Application Specific Products
;             Data Communication System Development
;             12203 SW Freeway, MS 701
;             Stafford, TX 77477
;{
;    IPR statements description (can be collected).
;}
;   (C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
;    Change history:
;
;        VERSION      DATE     /     AUTHORS           COMMENT
;          1.0     July-25-96  /    P.Mallela      original created
```

Example 23. Interrupt 1 Service Routine (Continued)

```
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains interrput service routine INT1:
;        1) host_command_int1
;    1.3 Specification/Design Reference (optional)
;        INT1 is serviced whenever host writes to CH A
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 –s hst_int1.asm
;
;    1.6 Notes and Special Considerations
;        –
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        –
;    2.2 Local Compiler Flags
;        –
;    2.3 Local Constants
;        –
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
        .mmregs
        .include    ”target.inc”
;    3.2 External Data
;        .ref        FIFO_DP
        .ref        d_command_reg
        .ref        d_fifo_count
        .ref        FIFO_DP
        .ref        d_command_value
        .ref        d_fifo_ptr
        .ref        d_output_addr
;        3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        –
;    4.2 Global Static Data
;        –
```

Example 23. Interrupt 1 Service Routine (Continued)

```
;    4.3 Dynamic Data
;        –
;    4.4 Temporary Data
;        –
;    4.5 Export Functions
        .def        host_command_int1
;}
;    5. SUBROUTINE CODE
;    HeaderBegin
;=============================================================================
;
;-----------------------------------------------------------------------------
;    5.1 host_command_int1
;
;    5.2 Functional Description
;        The host generates INT1 DSP whenever it writes to CH A. In INT1
;        service routine, the command from host is read whether the FIFO
;        has been empty. Writes another 32 data from target to FIFO.
;        Sends a command to host. The host acknowledges the command and read
;        the 32 data from the FIFO and sends a command to the target for
;        another set of 32 data. This process continues for 6 times till all
;        256 processed samples are transferred to host. Processing INT1 is
;        done background, i.e., INT1 is globally enabled.
;-----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;        BD    host_command_int1
;        PSHM    ST0
;        PSHM    ST1
;
;        Reentrancy:    No
;        Recursive :    No
;
;    5.4 Inputs
;        Data structure:  d_fifo_count
;        Data Format:     16–bit var
;        Modified:        Yes
;        Description:     counter for # of transfers
;
;        Data structure:  d_fifo_ptr
;        Data Format:     16–bit variable
;        Modified:        Yes
;        Description:     holds the output bffr addr. and incremented by
;                         32 for every transfer
;
;    5.5 Outputs
;
;        Data structure:  AR7
;        Data Format:     16–bit output buffer pointer
;        Modified:        Yes
;        Description:     either point to PING/PONG buffer
;
;    5.6 Global
;        Data structure:  d_command_reg
;        Data Format:     16–bit variable
;        Modified:        Yes
;        Description:     command from host is read thru CH A
;
```

Example 23. Interrupt 1 Service Routine (Continued)

```
;        Data structure:  d_command_value
;        Data Format:     16–bit variable
;        Modified:        Yes
;        Description:     holds the command value
;
;    5.7 Special considerations for data structure
;        –
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |
;in |U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |U  |U  |NU|NU|NU|NU |NU|NU
;    |  |   |   |   |    |   |   |   |   |   |   |   |   |   |  |  |  |   |  |
;out|U | 1 | 1 |NU | 1  |NU |NU |NU |NU |NU |NU |NU |NU |U  |UM|NU|NU|NU |NU|NU
;
;     Note : UM – Used & Modified, U – Used, NU – Not Used
;
;    5.9 Execution
;        Execution time: ?cycles
;        Call rate:      not applicable for this application
;
;=============================================================================
;    HeaderEnd
;    5.10 Code
          .asg    AR7,OUTBUF_P                ; output buffer pointer
          .asg    AR7,SV_RSTRE_AR7
          .sect   ”fifo_fil”
host_command_int1:
          PSHM    AL
          PSHM    AH
          PSHM    AG
          PSHM    SV_RSTRE_AR7                ; AR7 is used as a poiner for
                                              ; output buffer
          LD      #FIFO_DP,DP                 ; restore the DP
          PORTR   K_CHA,d_command_value       ; read command from host
wait_host_receive_data
          PORTR   K_TRGCR_ADDR,d_command_reg  ; while (port14 & AXST)
          BITF    d_command_reg,K_ARST        ; check FIFO empty
          BC      wait_host_receive_data,TC   ; branch occurs
          LD      #K_FIFO_EMPTY,A             ; indicate of FIFO empty
          SUB     d_command_value,A
bad_command
          BC      bad_command,ANEQ            ; read the command send by host
          LD      #(K_FRAME_SIZE/(K_FIFO_SIZE/2))–1,A
          SUB     d_fifo_count,A              ; check for complete transfer of
                                              ; 256 samples
          BC      start_remain_fifo_transfer,AGT
          BD      transfer_over
          ST      #0,d_fifo_count             ; reset the fifo count
                                              ; start_remain_fifo_transfer
          MVDK    d_fifo_ptr,OUTBUF_P         ; load PING/PONG bffr address
          RPT     #(K_FIFO_SIZE/2)–1          ; write 32 data to FIFO
          PORTW   *OUTBUF_P+,K_CHB            ; Fill FIFO half
          ST      #K_FIFO_FULL,d_command_value
          PORTW   d_command_value, K_CHA      ; write command to cmmnd reg A
                                              ; for FIFO half
          ADDM    #1,d_fifo_count
          MVKD    OUTBUF_P,d_fifo_ptr         ; save the fifo_ptr
```

Example 23. Interrupt 1 Service Routine (Continued)

```
transfer_over:
          POPM    SV_RSTRE_AR7                ; restore AR7
          POPM    AG
          POPM    AH
          POPM    AL
          POPM    ST1
          POPM    ST0
          RETE
          .end
```

Example 24. Function Calls on Host Side

```
    Host Action
/*************************************************************************/
/*                              FILE NAME: HOST.C                        */
/*                   C54x EVM/HOST COMMUNICATION FUNCTIONS -- HOST SIDE   */
/*************************************************************************/
    #include ”graphic2.c”
    #include ”host.h”         /* flag names, constants       */
/*_______________________________________________________________________
    This function initializes the data buffer and reads the FIFO so that FIFO
    is empty when the real data transfers start
-------------------------------------------------------------------------*/
    void initialize_slave(void)
{
        int j;
        for (j=0;j < 64; j++)
        dataa[j] = inport(BDAT_REG)  ; /* read data from data reg.    */
        for (j=0;j <256; j++)
        dataa[j] = 0;
        outport(CONT_REG, inport(CONT_REG) & 0xf7ff);
 }
/*_______________________________________________________________________
 The target sends a command to the host after collecting 32 word data from DSP
    memory to FIFO. The host checks if the command has been received
 ------------------------------------------------------------------------*/
int receive_command_FIFO_FULL(void)
                        /* RECEIVE COMMAND FROM EVM    */
{
while(!(inport(CONT_REG) & ARST))    ; /* wait for evm to send command*/

reply = inport(ADAT_REG)             ; /* read command into reply*/

while ((reply & 0xFF) !=0xFF)        ; return(reply)
                                     ; /* return command for process’g*/
 } /*_____________________________________________________________________*/
/*    This function sends a command to target for a new set of data from FIFO*/
      void send_command_new_FIFO(command unsigned int command;
 {
               command = 0xEE;
               outport (ADAT_REG,command);
               while(inport(CONT_REG) & AXST);
  }
/*-------------------------------------------------------------------------
      This initiates the handshake between the target and host. The host writes
      a command to target which sets the AXST flag to 1. The INT1 is generated
      whenever host writes to CH A. On the target side, INT1 is polled and reads
      the CH A.This clears ARST on target side. A command is written to Ch A on
      target after emptying the FIFO that sets AXSt =1. Later sets XF to go low.
      On host XF is polled and then reads CH A that clears ARST to 0 and AXST to 0
      on the target side
```

Example 24. Function Calls on Host Side (Continued)

```
*/
   int receive_clear_AXST(void)
                                    /* RECEIVE COMMAND FROM EVM    */
{
    command = 0xAB;
    outport (ADAT_REG,command);
    while(inport(CONT_REG) & AXST);      /* write command to evm */
    while((inport(CONT_REG) & XF));
    while(!(inport(CONT_REG) & ARST));  /* wait for evm to send command*/
    reply = inport(ADAT_REG);           /* read command into reply     */
    while ((reply & 0xAE) !=0xAE);
    return(reply);                                /* return command for process'g*/
    }
/*--------------------------------------------------------------------------*/
```

Example 25. Main Function Call on Host Side

```
/**************************************************************************/
/*                              FILE NAME: MASTER.C                       */
/*                    C54x EVM/HOST COMMUNICATION FUNCTIONS -- HOST SIDE      */
/**************************************************************************/
    #define   F1    15104
    #define   F2    15360
    #define   F3    15616
    #define   DATA_FRAME  256
    #include <bios.h>
    #include "view2.c"
    int get_kbhit(void);
    extern void send_command_new_FIFO(unsigned int);
    void main(void)
{
    int count=0,n,fifo_size;
    int main_done =0;
    int done = 0;
    int hit;
    initialize_slave();
/*   a command is written to CH A to initiate handshake */
     command_AXST = receive_clear_AXST()
;    while (!main_done)
{
      done =0;
      init_graphics()
    ; while(!done)
{
      count =0;
      if(kbhit())
{
      hit = get_kbhit();
      setviewport(120,433,290,445,0);
      clearviewport();
      switch(hit)
{
    case    (F1):    amplitude = amplitude * 2    ; outtextxy(1,1,"Amplitude
                                                  ; decreased")
                                                  ; break;
//  case    (F2):    if (amplitude>=2) amplitude = amplitude / 2
                                                  ; outtextxy(1,1,"Amplitude
                                                  ; increased")
                                                  ; break;
    case    (F2):    done =1                      ; closegraph()
                                                  ; break;
```

Example 25. Main Function Call on Host Side (Continued)

```
    case    (F3):    done =1                        ;main_done=1
                                                    ;closegraph()
                                                    ;break;
}
}
else
{
    command_FIFO = receive_command_FIFO_FULL();
    for (fifo_size=0; fifo_size < 64; fifo_size++)
    dataa[fifo_size+count] = inport(BDAT_REG);
    send_command_new_FIFO(command);
    for (count=64; count< 256; count++)
{
    command_FIFO = receive_command_FIFO_FULL();     /* command from target*/
    for (fifo_size=0; fifo_size < 32; fifo_size++)
    dataa[fifo_size+count] = inport(BDAT_REG);      /* read 32 word fifo*/
    count = count+31;
    send_command_new_FIFO(command);                /* send command to target*/
}
    screen();
}
}
}
    closegraph();
}
int get_kbhit(void)
{
    unsigned int key = bioskey(0);
    fflush(stdin);
    return(key);
}
/*****************************************************************************/
/*        FILE NAME: HOST.H  C54x                                            */
/*      HOST SIDE FLAGS, CONSTANTS, COMMAND NUMBERS, AND GLOBAL VARIABLES    */
/*                                                                           */
/*****************************************************************************/
/* The numbers I've picked for the file I/O constants, command numbers, and  */
/* basic control constants are not important. These numbers could really     */
/* be anything, as long as two of them are not the same.  Please notice the  */
/* pattern I used for file commands. Masks and pointers CANNOT be changed.   */
#include <stdio.h>
/*---------------------- FILE I/O CONSTANTS AND COMMAND --------------------*/
#define  MAX_FRAME        256      /* size of data frame to be passed */
/*------------------------ BASIC CONTROL CONSTANTS -------------------------*/
#define  STOP        99
#define  NO          98
#define  YES         97
#define  READY       96
#define  CLEAR       95
#define  ACKNOWLEDGE  0
/*------------- POINTERS TO DATA, COMMAND, AND CONTROL REGISTERS -------------*/
unsigned int    ADAT_REG     = (unsigned int )(0x240 + 0x800);
unsigned int    BDAT_REG     = (unsigned int )(0x240 + 0x804);
unsigned int    CONT_REG     = (unsigned int )(0x240 + 0x808);
```

Example 25. Main Function Call on Host Side (Continued)

```
/*------------ MASKS FOR READING MESSAGE FLAGS OF CONTROL REGISTER -----------*/
unsigned int    XF          = (unsigned int ) 0x0020;
unsigned int    ARST        = (unsigned int ) 0x0002;
unsigned int    AXST        = (unsigned int ) 0x0001;
unsigned int    BRST_MASK   = (unsigned int ) 0x0200;
unsigned int    BXST_MASK   = (unsigned int ) 0x0008;
/*---------------- GLOBAL VARIABLES USED BY EVM.C AND MASTER.C ---------------*/
FILE*file[20];            /* stores ptrs to files in files.dat*/
int     reply;            /* integer for whatever           */
int     reply1[128];
int     data[256];
int     index;
unsigned int command;
unsigned int command1;
unsigned int command_new_data;
unsigned int command_FIFO;
unsigned int command_AXST;
unsigned int command_HANDSHAKE;
int amplitude = –10;
```

Example 26. Graphic Drivers Routine

```
/*************************************************************************/
/*                          FILE NAME: GRAPHIC2.C                        */
/*                GRAPHICS DRIVER INITIALIZATION ROUTINE                 */
/*                                                                       */
/*************************************************************************/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void init_graphics(void)
{
           int i;
           int gdriver = DETECT, gmode, errorcode; int left, top, right, bottom;
           initgraph(&gdriver, &gmode, ””); errorcode = graphresult();
           if (errorcode != grOk);
           {
                      printf(”Graphics error: %s\n”, grapherrormsg(errorcode));
                      printf(”Press any key to halt.\n”);
                      getch();
   }
           cleardevice();
           setlinestyle(0,0,1);
           setcolor(1);
           setfillstyle(1,3);
           rectangle(1,1,getmaxx()–1,getmaxy()–1);
           setcolor(7);
           left = 5;
           top = getmaxy()/2 + – 123;
           right = 108;
           bottom = getmaxy()/2+ 163;
           rectangle(left,top,right,bottom);
           line(left,top + 30,right,top + 30);
           line(left,bottom+29,right,bottom+29);
           rectangle(left,top – 110,right+523,top – 5);
           rectangle(right+4,bottom + 5, right + 523,bottom + 70);
```

Example 26. Graphic Drivers Routine (Continued)

```
            rectangle(left,botom +5,right,bottom+70);
            setcolor(15);
            settextstyle(0,0,1);
/*          outtextxy(left+25,top+12,"CONTROL");
            outtextxy(right+16,bottom+32,"MESSAGES:");
            outtextxy(left+13,bottom+13,"AIC STATUS");*/
            outtextxy(left+2,bottom+38,"Freq: 8kHz");
            outtextxy(left+2,bottom+53,"Gain: 2");
            settextstyle(0,0,3);
            outtextxy(left+10,top–72,"C54X EVM Spectrum Analyzer");
            setlinestyle(0,0,3);
            setcolor(15);
            left   = getmaxx() / 2 – 206;
            top    = getmaxy() / 2 – 123;
            right  = getmaxx() / 2 + 312;
            bottom = getmaxy() / 2 + 163;
            rectangle(left, top, right, bottom);
            floodfill(left+10,top+10,15);
            setcolor(15);
            setlinestyle(0,0,3);
            left   = getmaxx()/2 – 311;
            top    = getmaxy()/2 + 57 – 40;
            right  = left + 25;
            bottom = top +  25; rectangle(left,top,right,bottom);
            setfillstyle(1,1); floodfill(left+5,top+5,15);
            settextstyle(0,0,1);
            outtextxy(left+5,top+10,"F3");
//          outtextxy(left+30,top+10,"Quit");
            setcolor(15); setlinestyle(0,0,3);
            left   = getmaxx()/2 – 311;
            top    = getmaxy()/2 – 23 – 40;
            right  = left + 25;
            bottom = top + 25; rectangle(left,top,right,bottom)
            setfillstyle(1,1); floodfill(left+5,top+5,15);
            settextstyle(0,0,1); outtextxy(left+6,top+10,"F1");
            outtextxy(left+30,top+9,"Decrease");
            outtextxy(left+30,top+16,"amplitude");
            setcolor(15);
            setlinestyle(0,0,3);
            left   = getmaxx()/2 – 311;
            top    = getmaxy()/2 + 17 – 40;
            right  = left + 25;
            bottom = top + 25;
            rectangle(left,top,right,bottom);
            setfillstyle(1,1);
            floodfill(left+5,top+5,15);
            settextstyle(0,0,1); outtextxy(left+6,top+10,"F2");
            outtextxy(left+30,top+9,"Increase");
            outtextxy(left+30,top+16,"amplitude");
```

Example 27. Display the Data on the Screen

```
/*************************************************************************/
/*                            FILE NAME: VIEW2.C                         */
/*                       DISPLAYS DATA ON THE SCREEN                     */
/*                                                                       */
/*************************************************************************/
#include ”host.c”
void screen(void)
{
    int x,y;
    int i,n;
    /* setup window (viewport) to display the AIC data */
    x = getmaxx()/2 – 203;
    y = getmaxy()/2;
    setviewport(x,y–120,x+512,y+160,1);
    /* move CP to left side of viewport */
    x = getx();
    y = gety() + 143;
    moveto(x,y);
/* make waveform by drawing lines btwn 256 data points sent by EVM */
setlinestyle(0,0,1);
setcolor(14);
for(i=0;i<256;i++) lineto(x+1+2*i,y+dataa[i]/amplitude);
/* erase waveform just drawn by re–writing it in background color */
moveto(x,y);
setcolor(3);
for(i=0;i<256;i++) lineto(x+1+2*i,y+dataa[i]/amplitude);
}
```

Example 28. Linker Command File for the Application

```
/*************************************************************************
* This linker command file is assigns the memory allocation for the application
* based on the EVM54x specifically 541.
*************************************************************************/
    vectors.obj
    init_54x.obj
    init_ser.obj
    init_aic.obj
    aic_cfg.obj
    memory.obj
    main.obj
    prcs_int.obj
    rcv_int1.obj
    task.obj
    fir.obj
    iir.obj
    sym_fir.obj
    adapt.obj
    echo.obj
    rfft.obj
    bit_rev.obj
    fft.obj
    unpack.obj
    power.obj
    hand_shk.obj
    hst_int1.obj
    fifo.obj
    –o main.out
    –m main.map
```

Example 28. Linker Command File for the Application (Continued)

```
MEMORY
{
PAGE 0:
    PROG    : origin = 0x7000,    length = 0x1000
    VECS    : origin = 0xff80,    length = 0x7f
    COFF_SYM: origin = 0x1400,    length = 0x40
    COFF_FIR: origin = 0x1440,    length = 0x40
    AIC_TBLE: origin = 0x1480,    length = 0x10
    TASK_TBL: origin = 0x1490,    length = 0x10
    TASK_INT: origin = 0x14a0,    length = 0x10
    IIR_COFF: origin = 0x14b0,    length = 0x10
    COEFFH  : origin = 0x1500,    length = 0x100
    TWID_SIN: origin = 0x1000,    length = 0x80
    TWID_COS: origin = 0x1200,    length = 0x80
    PAGE 1:                     /* Data space */
    ALL_VARS: origin = 0x0080,    length = 0x0080
    DRAM    : origin = 0x0100,    length = 0x1300
    EXT_DAT : origin = 0x1400,    length = 0xE000
    REGS    : origin = 0x0000,    length = 0x0060
}
SECTIONS
  {
    .text           : {} >  PROG            PAGE 0          /* code */
    vectors         : {} >  VECS            PAGE 0         /* Vector table */
    main_prg        : {} >  PROG            PAGE 0
    zeropad         : {} >  PROG            PAGE 0
    aic_cnfg        : {} >  PROG            PAGE 0
    ser_cnfg        : {} >  PROG            PAGE 0
    fifo_fil        : {} >  PROG            PAGE 0
    task_hnd        : {} >  PROG            PAGE 0
    handshke        : {} >  PROG            PAGE 0
    fir_prog        : {} >  PROG            PAGE 0
    iir             : {} >  PROG            PAGE 0
    filter          : {} >  PROG            PAGE 0
    rfft_prg        : {} >  PROG            PAGE 0
    aic_reg         : {} >  AIC_TBLE        PAGE 0
    task_int        : {} >  TASK_INT        PAGE 0
    task_tbl        : {} >  TASK_TBL        PAGE 0
    coff_fir        : {} >  COFF_FIR        PAGE 0
    sym_fir         : {} >  COFF_SYM        PAGE 0
    iir_coff        : {} >  IIR_COFF        PAGE 0
    coeffh          : {} >  COEFFH          PAGE 0
    sin_tbl         : {} >  TWID_SIN        PAGE 0
    cos_tbl         : {} >  TWID_COS        PAGE 0
    inpt_buf        : {} >  DRAM,align(1024)PAGE 1
    outdata         : {} >  DRAM,align(1024)PAGE 1
UNION:                  >  DRAM align(1024)PAGE 1
{
    fft_bffr
    adpt_sct:
{
    *(bufferw)              /* This is needed for alignment of 128 words */
    .+=80h;
    *(bufferp)
}
}
```

Example 28. Linker Command File for the Application (Continued)

```
UNION:                    >  DRAM align(256) PAGE 1
{
    fir_bfr
    cir_bfr
    coff_iir
    bufferh
    twid_sin
}
UNION:                    >  DRAM align(256) PAGE 1
{
    fir_coff
    cir_bfr1
    bufferx
    twid_cos
}
GROUP:                    > ALL_VARS    PAGE 1
{
    aic_vars
    rcv_vars
    fifo_var
    tsk_vars
    fir_vars
    iir_vars
    adpt_var
    fft_vars
}
stack        : {} >  DRAM PAGE 1
}
```

Example 29. Memory Map of TMS320C541

```
;TMS320C541 MEMORY MAP
MR
;
MA 0x0000, 1, 0x002A, RAM  ; MMRs
MA 0x0030, 1, 0x0003, RAM  ;
MA 0x0060, 1, 0x0020, RAM  ; SCRATCH PAD
MA 0x0080, 1, 0x1380, RAM  ; INTERNAL DATA RAM
MA 0x0080, 0, 0x1380, RAM  ; INTERNAL PROGRAM RAM
MA 0x9000, 0, 0x7000, ROM  ; INTERNAL ROM
ma 0x1400, 0, 0xec00, ram  ; external ram
ma 0x1400, 1, 0xec00, ram  ; external ram
ma 0x0000, 2, 0x15, ioport ; i/o space
map on
;
;Define reset alias to set PMST for MC mode
;
;alias myreset, ”e pmst = 0xff80; reset ”
;e pmst = 0xffe0  ; MP mode, OVLY, DROM off CLKOUT on
;e hbpenbl = 0x0000
e *0x28 = 0x2000 ; two wait states on i/o, none for memory
e *0x29 = 0x0000 ; no bank switching necessary
dasm pc
echo Loaded TMS320C54x evminit.cmd
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.