EMBEDDED PROCESSING SOLUTIONS EDITION

Issue 64
Second Quarter 2008

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

Inside the New Virtex-5 FXT FPGA

COVER
Embedded Processing Innovations with Virtex-5 FXT Devices

INSIDE
Next-Generation Wireless Standards Using Virtex-5 FXT Devices

FPGA Coprocessors for Spacecraft Image Processing

Designing Multiprocessor SOCs

Building Linux Platforms on Xilinx Processors

XILINX®

www.xilinx.com/xcell/

Support **Across The Board**™

Bryan Fletcher
Avnet
Global Technical Marketing Manager
David Loftus
Xilinx
Vice President / General Manager
General Products Division (GPD)
XILINX
SPARTAN-3

Bringing Products to Life.

At Avnet Electronics Marketing, support across the board is much more than a tagline for us. From design to delivery – we are deeply committed to driving maximum efficiency throughout the product lifecycle.

The Challenge

When the world's largest FPGA maker Xilinx was mounting the launch of its latest Spartan™-3 Generation family of low-cost FPGAs, the company wanted to ensure designers would have everything they need to compete in the highly competitive high-volume market. Xilinx turned to longtime partner Avnet to lend its technical expertise in creating a complete solution.

The Solution

Avnet spent countless engineering hours alongside Xilinx – developing a comprehensive solution of starter kits, development boards, and Speedway™ educational workshops. With Xilinx® Spartan-3 Generation FPGAs and Avnet's support across the board, designers have access to all they need to save up to 50 percent in total system cost over competing FPGAs.

Visit **www.em.avnet.com/xilinxspartan3kits** to learn more about Spartan-3 solutions and to purchase a **Spartan-3A evaluation kit for only $39!**

XILINX®

AVNET® electronics marketing

Accelerating Your Success™

1 800 332 8638
www.em.avnet.com

Avnet Green Initiative

Xcell journal

PUBLISHER	Forrest Couch forrest.couch@xilinx.com 408-879-5270
EDITOR	Charmaine Cooper Hussain
ART DIRECTOR	Scott Blair
DESIGN/PRODUCTION	Teie, Gelwicks & Associates 1-800-493-5551
ADVERTISING SALES	Dan Teie 1-800-493-5551 xcelladsales@aol.com
TECHNICAL COORDINATORS	Larry Caputo Jay Gould
INTERNATIONAL	Piera Or, Asia Pacific piera.or@xilinx.com
	Christelle Moraga, Europe/Middle East/Africa christelle.moraga@xilinx.com
	Yumi Homura, Japan yumi.homura@xilinx.com
SUBSCRIPTIONS	All Inquiries www.xcellpublications.com
REPRINT ORDERS	1-800-493-5551

www.xilinx.com/xcell/

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124-3400
Phone: 408-559-7778
FAX: 408-879-4780
www.xilinx.com/xcell/

Integrated System Platforms: The Next Wave in FPGA-based Innovation for Embedded Processing Applications

FPGAs have come a long way since the days serving as prototypes to test design concepts before committing them to 'permanent' silicon. FPGAs have now become the foundation for many of the most innovative products that improve our lives everyday. Driven by diverse and demanding market requirements, FPGAs are valued above all else for their flexibility. Their inherent programmability is the insurance policy product developers depend on to quickly and efficiently meet market windows and keep products current in fast moving, evolving markets.

As many of the articles published in *Xcell Journal* illustrate, FPGAs are no longer a single-function device. These programmable devices have emerged as the preferred platform for the highly integrated world of systems-on-chip (SoC) and embedded system design. Therefore, it is no coincidence that many of you will see this issue of our magazine at the Embedded System Conference in San Jose, CA, one of the largest gatherings of hardware designers and software developers.

It's also appropriate that this theme is highlighted as Xilinx introduces its ultimate system integration platform – the Virtex™-5 FXT FPGA. Comprising the fourth platform in the 65-nanometer Virtex-5 family, Virtex-5 FXT devices combine flexibility with very high-performance embedded processing, digital signal processing and connectivity capabilities into a single chip to reduce total system cost, power, and board real estate.

In this Issue

The significance of the Virtex-5 FXT platform for embedded developers is underscored in the cover article, which describes in detail the industry's first FPGAs with embedded PowerPC® 440 processor blocks, high-speed RocketIO™ serial transceivers, and dedicated XtremeDSP™ processing capabilities. These devices deliver an optimal system integration platform that meets market and bandwidth requirements of transporting voice, video, and data for a wide range of applications in wired and wireless communications, audio/video broadcast equipment, military, aerospace and industrial systems, along with many others. This issue provides a "behind-the-scene" look into a variety of FPGA-based embedded system applications, ranging from outer space and aerospace to more earthly scientific and automotive innovations.

In addition, you'll learn how every aspect of Xilinx embedded processing solutions has been substantially upgraded and usability drastically simplified through intuitive hardware and software tools. Advancements include the v7 release of the MicroBlaze™ 32-bit soft processor with configurable memory management unit, the new high-performance processor interconnect architecture, and the ISE™ Design Suite 10.1 development environment with unified access to all logic, embedded, and DSP design tools. Of course, our expansive ecosystem of third-party embedded technology and service providers, several of whom are featured in this issue, also play a key role in helping developers maximize the benefits of Xilinx embedded solutions.

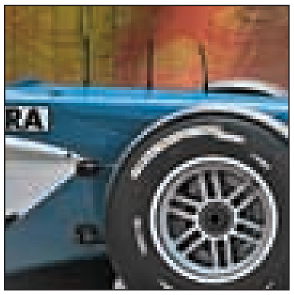
At the end of the day, the goal of these efforts is to enable rapid design of high-performance embedded systems with highly integrated, scalable hardware platforms, customizable embedded processors, and pre-verified IP cores. Designers can then focus their efforts on creating differentiated value, thereby getting products to market in time to extract maximum return and ultimately realizing the full potential of their own inspirations.

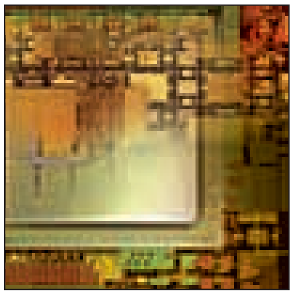
[illegible]

Forrest Couch
Publisher

SECOND QUARTER 2008, ISSUE 64

Xcell journal





EMBEDDED

Unlock your future

Enter the New Era of Configurable Embedded Processing

Adapt to changing algorithms, protocols and interfaces, by creating your next embedded design on the world's most flexible system platform. With the latest processing breakthroughs at your fingertips, you can readily meet the demands of applications in automotive, industrial, medical, communications, or defense markets.

Architect your embedded vision

- *Choose MicroBlaze™, the only 32-bit soft processor with a configurable MMU, or the industry-standard 32-bit PowerPC® architecture*
- *Select the exact mix of peripherals that meet your I/O needs, and stitch them together with the new optimized CoreConnect™ PLB bus*

Build, program, debug . . . *your* way

- *Port the OS of your choice including Linux 2.6 for PowerPC or MicroBlaze*
- *Reduce hardware/software debug time using Eclipse-based IDEs together with integrated ChipScope™ analyzer*

Eliminate risk & reduce cost

- *No worry of processor obsolescence with Xilinx Embedded Processing technology and a range of programmable devices*
- *Reconfigure your design even after deployment, reducing support cost and increasing product life*

Order your complete development kit today, and unlock the future of embedded design.

Get the Complete Embedded Solution

www.xilinx.com/processor

XILINX®

www.xilinx.com/processor

At the Heart of Innovation

Xilinx Welcomes New President & CEO

Wim Roelandts Passes the Torch to Industry Veteran Moshe Gavrielov

On January 7, 2008, Xilinx announced the appointment of a new president and CEO, Moshe Gavrielov. He assumes the role from Wim Roelandts, one of the most revered CEOs in the industry. Roelandts remains chairman of the board, while Gavrielov becomes only the third Xilinx® CEO in the company's 24-year history. In this issue's View from the Top, *Moshe provides a glimpse into his vision for Xilinx and how he plans to get us there.*

Let me start by saying how honored and excited I am to assume responsibility for leading one of the most highly respected semiconductor companies in the world. This is truly my dream job and I feel I'm joining the company at a very exciting time. We have a great opportunity to bring the benefits of Xilinx technology to a broader range of industries and applications in the coming years.

FPGAs are becoming more and more relevant to a wider range of designers. It's a well-known fact that the best way for designers to prepare for change is to arm themselves with options. The FPGA has always provided these options through the programmability of its hardware. We are the ultimate providers of faster time to market. But it's not just about silicon and flexibility – it's about the IP as well. In that way, Xilinx today is in a similar transition to one I have faced before: moving from a supplier of blank gates to being a supplier and supporter of the IP that goes into the gates. But supporting a body of IP in the field is a non-trivial undertaking.

I see a three-fold opportunity for Xilinx to better serve our customers while achieving higher revenues and increased market share. First, we must maintain our pace in expanding the underlying capabilities of our silicon. Second, we must continue to build our portfolio of IP across a growing breadth of applications. And third, we must continue to invest in our development tools so that we are able to serve the diverse needs of a growing, increasingly specialized community of traditional and new users.

Xilinx is already on the path to becoming a complete solutions supplier. But we have a ways to go before we get there. We need to learn from our customers across the globe, developing an intimate understanding of their pain points as if they were our own. Then we need to provide solutions to make their pain disappear.

On DSP and Embedded

Xilinx already had its eye on the DSP and embedded markets when I arrived. Three years ago, the company announced the formation of a new division to broaden the reach of Xilinx FPGAs into multi-billion dollar vertical markets previously the domain of of ASICs and ASSPs, including audio, video and broadcast, industrial automation, aerospace and defense, medical, automotive, and consumer electronics.

FPGAs offer a compelling value proposition and significant technology advantages for both high-performance DSP and embedded applications. Already, DSP and embedded processors have opened an entirely new set of opportunities for Xilinx. By integrating key functionalities into a single FPGA, designers can reduce total system cost, power, and board space by reducing component count. Simple, right? Not so simple actually, which leads me back to my point on becoming a total solutions provider. Every designer – particularly those coming from the DSP or embedded space – will need more than just FPGAs to make the proverbial leap from an ASIC or ASSP.

Xilinx must provide all of the necessary tools, including IP, software tools, and design methodologies that allow designers to leverage the high-performance, flexible capabilities of the FPGA. Oh yeah – and we need to make that leap as compelling and simple as possible. I, for one, am looking forward to doing just that.

For the latest in Xilinx solutions designed to meet the needs of the DSP and embedded designer, visit *www.xilinx.com.*

Embedded Processing Innovations with Virtex-5 FXT Devices

Maximizing performance and throughput utilizing the built-in PowerPC 440 processor block.

by Craig Abramson
Product Marketing Manager
Xilinx, Inc.
craig.abramson@xilinx.com

Dan Isaacs
Director of Embedded Processor Marketing
Xilinx, Inc.
dan.isaacs@xilinx.com

Ahmad Ansari
Principal Engineer
Xilinx, Inc.
ahmad.ansari@xilinx.com

With the advent of the Xilinx® Virtex™-5 FXT FPGA, you have an opportunity to get ahead of the embedded system design curve. The need to quickly develop and validate embedded systems has never been more apparent than in the realm of embedded system design.

Combining software and hardware to demonstrate this at a system level (as quickly as time permits) has become commonplace in the industry. By providing a more tightly coupled, flexible, scalable solution, you have a means to address many hardware and software SOC design challenges.

FPGAs provide a significantly faster path for designers to rapidly develop, prototype, and test their embedded designs. The Virtex-5 FXT device platform, the third-generation FPGA to feature a PowerPC processor, has added an embedded block that will help you meet more demanding design requirements while allowing you to finish your designs quickly and easily.

In this article, we'll provide a detailed description of the embedded processing innovations in the PowerPC 440 processor block and system interconnect. A key area of focus in the Virtex-5 FXT FPGA processor block is simplification through integration.

A corollary to this is ease of development and test. Quickly bringing up a system to allow software developers to get a head start on actual hardware is a major emphasis for the Virtex-5 FXT device's PowerPC 440 processor.

Simplification Through Integration

Integration is key. We have reduced the amount of FPGA logic needed to build a high-performance processing system while still allowing a wide variety of topologies. You still have the flexibility and advantages of an FPGA-based implementation, but you now also have the added benefit of a hardened, integrated interconnect architecture that (among other things) maximizes access to external memory.

As you will see, the result is an embedded block that allows you to develop a wider range of high-performance processing architectures in a shorter period of time.

PowerPC processors generically have three interfaces: instruction read, data read, and data write. In previous Virtex device architectures, which embedded the PowerPC 405, these processor buses would connect to FPGA fabric. The timing closure requirements of this circuitry would vary based on how many and what types of loads the design presented to the buses.

In the Virtex-5 FXT FPGA (where the processor is now the PowerPC 440), these buses are hardened and hooked directly to a new structure, an integrated 5 x 2 crossbar switch – generically referred to as the crossbar. This hardened interconnect provides significantly higher performance (with virtually no consumption of FPGA logic resources and fixed timing) when combined with the rest of the architectural enhancements in the Virtex-5 FXT device's embedded processor block. This results in an overall system cost reduction and invariably a more tightly integrated processor system.

The processor buses only take up three of the five "crossbar master" ports on the 5 x 2 crossbar (see Figure 1). The crossbar includes two additional master ports, because in many real-world applications it's not just the processor that needs access to memory or peripherals. Each of these "crossbar master" ports comprises a processor local bus (PLB) slave interface, as well as two channels of scatter-gather direct memory access (DMA).

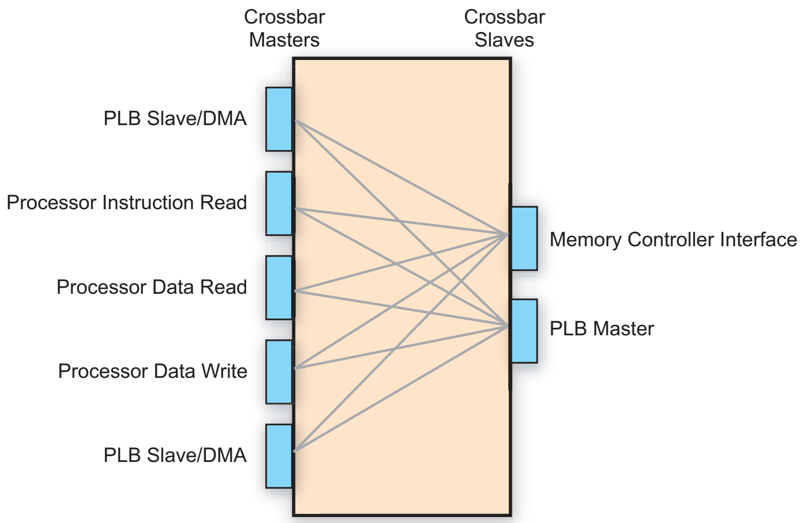


Figure 1 – The crossbar

The "slave" side of the crossbar comprises two ports. One port is a dedicated memory controller interface that provides a high-throughput generic interface to soft memory controllers. The other is a bus for attaching I/O devices and peripherals.

A Better Processor

Providing all of this extra functionality in the embedded processor block would be of little consequence if there were not a processor with the horsepower to take advantage of it. The Virtex-5 FXT FPGA represents the first time anyone has embedded a PowerPC 440-class processor in an FPGA.

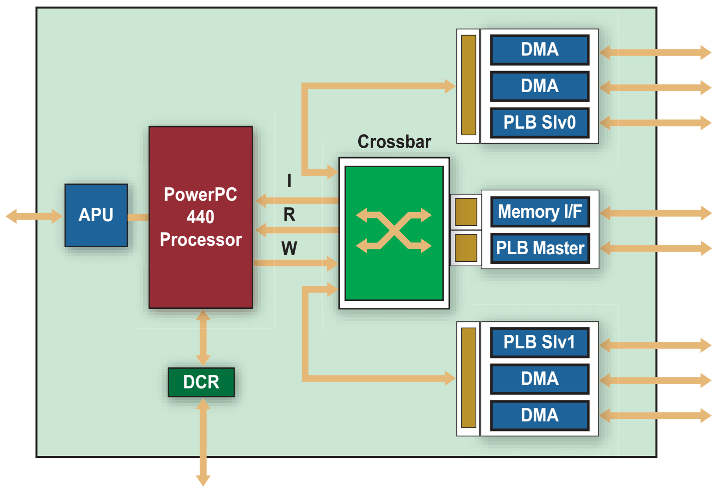


Figure 2 – The PowerPC 440 embedded processor block

The PowerPC 440 offers a significant performance improvement over the PowerPC 405 (which was embedded in previous Virtex families) in a number of areas.

First, the PowerPC 440, when in the fastest speed-grade FPGA, can be clocked at 550 MHz. The PowerPC 405 topped out at 450 MHz. This is almost a 20% performance improvement. But add to that the fact that the I and D cache sizes are doubled, the instruction pipeline is seven instead of five stages, and the execution unit can now execute two instructions out of order and in parallel.

The result? You've got a processor with performance sufficient to handle a great many of today's embedded processing challenges.

There are a number of other advantages to moving from the PowerPC 405 to the PowerPC 440, as shown in Table 1.

The PowerPC 440 embedded block is shown in Figure 2.

High-Throughput Switch Matrix

The 5 x 2 crossbar is more than just a big switch. It provides non-blocking pipelined access from the five crossbar masters to the two crossbar slaves (see Figure 1). It allows concurrent transfers between different agents on the crossbar at the same time.

As shown, we'll call the buses going into the crossbar "crossbar masters" and the buses coming out "crossbar slaves." These interfaces are highly pipelined, thus allowing a large number of transactions to be in progress at the same time.

In fact, up to four concurrent transactions can exist: two for each crossbar slave (such as the memory controller or PLB master). Additionally, each crossbar master (that is, the three processor PLBs and the two PLB slave interfaces) can pipeline four read and four write transactions to the same slave.

Another key feature of the crossbar is its highly programmable memory mapping. You can think of the entire system of having available memory space of 4 GB. Both the memory controller interface and the PLB master can have different memory windows mapped into the memory space of any of the crossbar masters. These memory spaces can be programmed through the FPGA bitstream, by the processor at run time, or even by external logic on the FPGA using the crossbar's sideband bus, called the device control register (DCR) bus.

Integrated PLB Interfaces

As we mentioned earlier, many of the buses connected to the crossbar are processor local buses, also called PLBs.

The PLB is one of the standard CoreConnect buses as defined by IBM. An earlier version of the PLB (version 3.4) was used as one of the standard buses on PowerPC 405 designs in Virtex-II Pro and Virtex-4 FX FPGAs and is also used in the new PowerPC 440 embedded processor block.

In the PowerPC 440 embedded processor block, the PLBs connect the processor's internal caches to the input side of the crosspoint. The buses are:

- ICURD: instruction cache unit read
- DCURD: data cache unit read
- DCUWR: data cache unit write

	PPC405 (Virtex-4 FX FPGA)	PPC440 (Virtex-5 FXT FPGA)	Benefit
Architecture	32-bit instruction, 32-bit address, 64-bit data	32-bit instruction, 36-bit address, 128-bit data, Book E compliant	Access more physical memory, higher speed data movement
Pipline	Single instruction/cycle, five-stage pipeline, in-order issue	Two instructions/cycle, seven-stage pipeline, out-of-order issue	More efficient instruction execution
Caches – I/D	16K/16K, two-way set associative, no locking	32K/32K, 64-way set associative, locking	Less memory access latency
MMU	Page size: 1 KB to 16 MB	Page size: 1 KB to 256 MB	Less page swapping
DMPS Estimate	700+ DMIPS	1000+ DMIPS	Better benchmarks equal higher performance

Table 1 – Virtex PowerPC integrated processor feature comparison

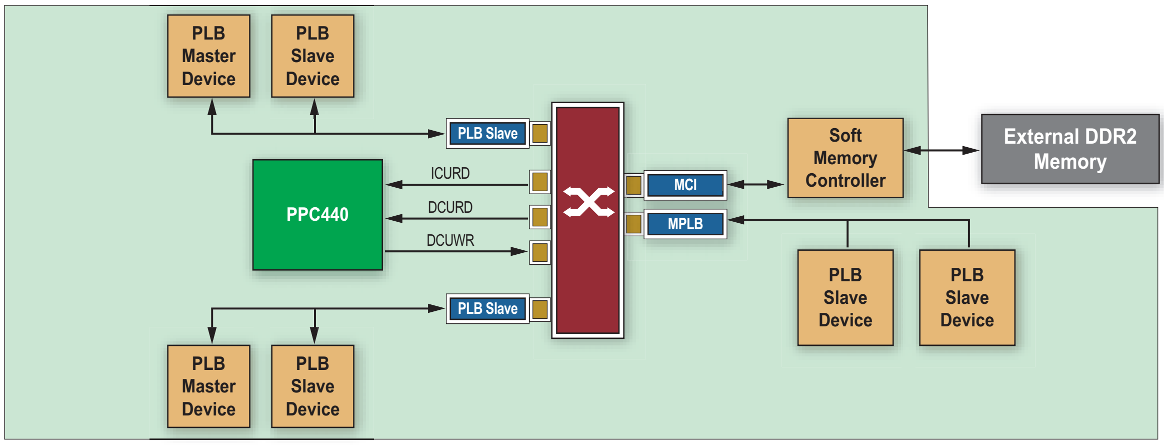


Figure 3 – PLB masters and slaves

The PLB used in the Virtex-5 FXT device is version 4.6 (PLB46). The PLB46 bus architecture brings with it a number of new capabilities that give it a nice performance boost over its predecessor. The most obvious is the fact that while PLB34 was 64 bits, PLB46 is 128 bits. But not to worry – if the IP connected to the bus is less than that, the bus will perform dynamic bus sizing as required to accommodate 32- and 64-bit transactions.

We should also point out that the PLB46 version is a Xilinx implementation of the IBM-defined PLB46, optimized to take advantage of FPGA resources.

PLB46 – and indeed all versions of PLB – have the concept of master and slave. This should not be confused with crossbar master and crossbar slave. (Again, refer to Figure 1.) As we stated earlier, there are two PLB slave port interfaces on the crossbar; they are crossbar masters. These slave ports are connected to the FPGA fabric.

In a processor system there is often the need to allow something besides the processor to access external memory or on-chip peripherals. The PLB slave interfaces allow just that. PLB masters, built from FPGA logic, connected to the PLB Slave ports (see Figure 3) can access either the MCI or the MPLB through the crossbar.

Similarly, the function of the PLB master (the one that is the crossbar slave) is to have a PLB to hook to I/O devices and soft peripherals. And because the PLB master is a crossbar slave, anything hooked to a crossbar master port can access it.

Note that there can be no more than four PLB masters connected to each PLB slave bus. Few systems are likely to need more than four masters, but if you did need more, you could always use the PLB/PLB bridge IP provided with the Embedded Development Kit (EDK) (see *www.xilinx.com/support/documentation/ipembedprocess_coreconnect_plbbusstruct.htm*).

Figure 3 is a simplified system diagram showing how PLB peripherals can be hooked to crossbar master and crossbar slave ports. Note that if you have multiple masters on any PLB, arbitration is handled by the IP for the bus. You do not need a separate arbiter.

Optimized DMA Engines

There are four additional crossbar masters; they are the four DMA channels. Each DMA channel has separate 32-bit transmit and 32-bit receive interfaces. They share crossbar arbitration with PLB slave interfaces, as shown in Figure 4.

All DMA ports can be operating at the same time. Each one has a dedicated FIFO, so as one DMA is accumulating data, the other DMA can be pumping data through the crossbar. Each DMA channel operates asynchronously to the processor clock.

The interface into the DMA channels is through an interface called LocalLink. Xilinx uses the LocalLink interface in a

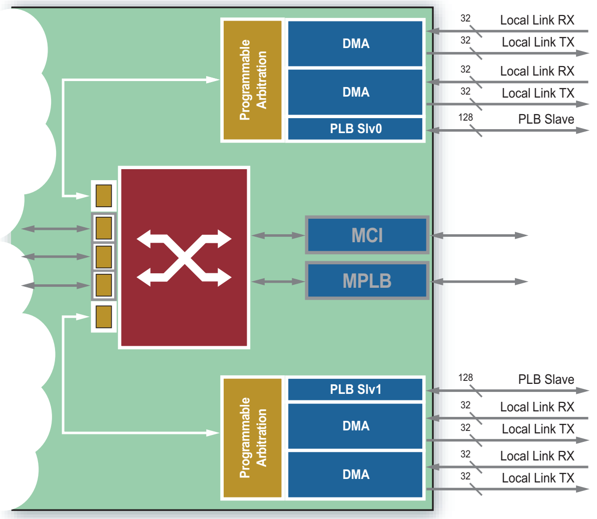


Figure 4 – PLB slave buses and DMA channels share crossbar arbitration.

To maximize throughput and performance, the PowerPC 440 embedded block employs scatter/gather DMA. To make using this capability as easy as possible, Xilinx provides wrappers for the various pieces of IP and embedded blocks it offers.

number of IP blocks. LocalLink is a point-to-point interface that sends packets to, or receives packets from, some external device.

The most notable type of processor IP that uses the LocalLink interface is the hard embedded tri-mode Ethernet media access controller (TEMAC) block. The TEMAC has a wrapper that allows it to communicate directly with the PowerPC 440 DMA.

Although all data paths through the crossbar are 128 bits, the LocalLink interface into and out of the DMA channels are all 32 bits. As such, there is built-in logic between the DMA controller and the crossbar that realigns data.

To maximize throughput and performance, the PowerPC 440 embedded block employs scatter/gather DMA. To make using this capability as easy as possible, Xilinx provides wrappers for the various pieces of IP and embedded blocks it offers.

The first one targeted specifically toward the PowerPC 440 is the soft wrapper for the embedded TEMAC blocks. This wrapper, combined with the functionality of the DMA engine in the PowerPC 440 embedded block, allows you to easily build a processing system with a high-performance TEMAC connected directly to the PowerPC 440 DMA channels. Figure 5 is a simplified system showing how both DMA and PLB peripherals can be hooked to crossbar master and crossbar slave ports.

The DMA channels are controlled by descriptors, small blocks of memory that are set up by the PowerPC 440 processor before commencing DMA operations. The descriptors control how much data is transferred and where data is located in system memory.

Descriptors can be chained together if need be, effectively creating a sequence of commands to control a DMA channel. The DMA controller is covered in its entirety in the reference guide, entitled "Embedded Processor Block in Virtex-5 FPGAs" (*http://www.xilinx.com/support/documentation/user_guides/ug200pdf*).

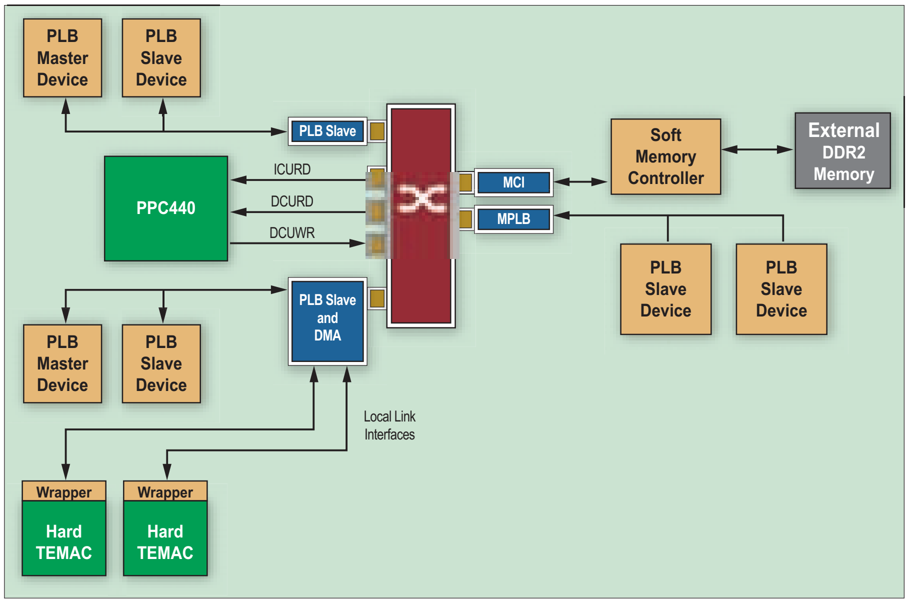


Figure 5 – Sample system with both PLB and DMA peripherals

High-Performance Dedicated Memory Interface

Rounding out the new processor block is the dedicated memory controller interface. The purpose of this interface is to provide a dedicated link out to external memory, but at the same time not be tied to any specific memory technology.

At this time, the memory controller interface supports a stand-alone DDR2 controller and MPMC4 controller, all available through Xilinx Platform Studio, EDK 10.1. This interface provides the flexibility to connect to virtually any memory technology now or in the future.

The memory controller interface is streamlined, comprising address/data/control. It can be programmed to support 128-, 64-, 32-, or even 16-bit memory. It does width and burst realignment, so while the DMA may be bursting one size packet, the memory controller can buffer and realign the packet data to maximize the bandwidth to the memory. Burst size is programmable and can be 1, 2, 4, or 8, and the memory controller interface will automatically adjust the address to accommodate the various burst widths.

The majority of soft memory controller handshaking signals are generated by the interface on behalf of the memory controller. They are provided ahead of time such that the soft memory controller can generate throttling signals back to the memory interface. The memory controller interface – on behalf of the soft memory controller – can also be programmed to detect bank and row misses ahead of time and will inform the soft memory controller to anticipate a bank or row miss. All of these features together provide a solution whose primary goal is to maximize memory throughput.

Tuning the System

In some situations, a PLB or DMA interface just may not be the right solution. For

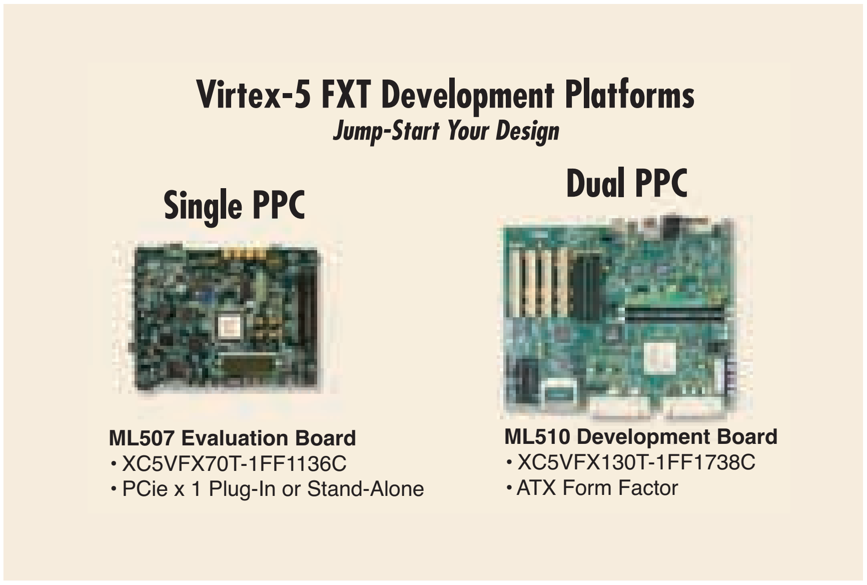


Figure 6 – Xilinx ML507 evaluation and ML510 development boards

instance, you might find that you have a software algorithm that takes too many cycles to execute and is affecting your system bandwidth. That algorithm is a great candidate for implementation in hardware, and the interface to which you may want that hardware connected would be the auxiliary processing unit, or APU interface.

The PowerPC 440 has a second-generation APU interface that is tightly coupled to the execution units of the processor. The interface is controlled by 16 user defined instructions (UDIs). The data path of the APU interface is 128 bits.

Perhaps the most common use of the APU interface is for connecting to a floating-point unit (FPU). The FPU is IEEE754-compatible and supports both single- and double-precision operations for the PowerPC 440.

The FPU is implemented in the FPGA soft logic fabric and utilizes the DSP48E blocks. The soft logic implementation operates up to half the frequency of the hard embedded processor.

Other uses of the APU interface include hardware algorithm acceleration, as well as an alternative high-bandwidth link to block RAM.

Configuring the Embedded Block

By integrating the PowerPC 440 block in the FPGA, the processor block can be configured in multiple ways. Virtually every interface is programmable.

For example, when you build your processing system in the Xilinx Platform Studio development environment and a bitstream is created, all of the specifications of the processing system are in the bitstream. Thus, when the FPGA starts up, your processor is up and running.

Now, let's say the processing system is up and running and you want to modify the operation of one of the DMA channels. You would do that through the DCR interface. There are DCR registers to control every aspect of DMA operation.

In fact, there is DCR access to virtually every other subsystem of the embedded block: the PLBs and crossbar, memory controller interface, and the APU controller. Refer to Figure 2 for more details.

Putting It All Together

This innovation would be for naught if Xilinx did not provide a comprehensive infrastructure to take advantage of all of the architectural enhancements. We should point out that the Virtex-5 FXT FPGA with the PowerPC 440 block represents our eighth year in embedded processing and our third-generation FPGA with a hardened processor.

Throughout that time we've been constantly updating EDK, our award-winning Embedded Development Kit. EDK includes Platform Studio, with its comprehensive library of IP for hardware design, and Platform Studio SDK, a software development environment familiar to many embedded software engineers.

With the introduction of the Virtex-5 FXT family of devices, we continue to further strengthen our third-party alliances with support from industry-leading operating system providers, including WindRiver Systems with VxWorks and Green Hills Integrity.

Linux support is provided through LynuxWorks, Monta Vista, and WindRiver Systems. In addition, Xilinx recognizes the importance of open-source Linux, and we're moving forward on that front.

Xilinx and its partner companies are also developing a wide variety of boards. Xilinx has multiple boards for the Virtex-5 FXT device: the ML507 with the XC5VFX70T and the ML510 with the XC5VFX130T, as shown in Figure 6. The ML507 evaluation platform enables your team to quickly begin developing hardware, software, or both. When multiple processors or a motherboard-type platform are required, the ML510 with the dual-processor XC5VFX130T is ideal.

Conclusion

A high-performance processing solution with optimized data throughput is high on the wish list of embedded designers everywhere. This is true whether you are running critical algorithms at the heart of the latest wireless base station, switching high-bandwidth data through a video switch, performing advanced signal processing for guidance systems using coprocessor acceleration, or handling complex control and system management tasks.

The Virtex-5 FXT embedded processor block, with a multi-ported non-blocking integrated processor interconnect and high-performance integrated DMA, offers a solution that allows you to focus on the key elements of your embedded design.

With a virtually unlimited number of ways to harness these embedded capabilities, the Virtex-5 FXT FPGA embedded processing solution provides a highly integrated platform for high-performance, high-throughput SOC designs.

Implementing Next-Generation Wireless Standards Using Virtex-5 FXT Device Parts

The Xilinx LTE baseband reference design shows how the Virtex-5 FXT device enables hardware, software, and high-speed off-chip comms for wireless baseband processing – implemented on a single device.

by Rob Payne
Staff Engineer
Xilinx, Inc.
robp@xilinx.com

The next generation of the 3GPP wireless standard is called long-term evolution (LTE). It provides a leap in performance and a complete move to packet-based processing. In the physical (PHY) level of the LTE specification, specific challenges exist when dealing with higher data throughput rates, as well as the move to OFDM (orthogonal frequency-division multiplexing) for transmission.

Xilinx has developed – or is in the process of developing – several new or revised DSP LogiCORE™ solutions to meet the demands of the new specification. With such blocks it is critical not only to verify them as stand-alone blocks, but also to validate them in real systems with real-world data. The Xilinx 3GPP downlink reference design provides this validation, as well as providing a reference to customers about how to use the blocks.

The higher data rate in LTE places increased processing demands on all parts of the system: increased DSP hardware processing in the baseband, increased software processing to implement the higher layers of the UMTS protocol stack, and increased I/O communication bandwidth to accept packets and pass data to remote radio-heads.

In this article, I'll review some of the new features of the LTE specification and how Xilinx® Virtex™-5 FXT devices address the increased processing demands of LTE through its tight integration of microprocessor subsystem, DSP-enhanced FPGA fabric, and high-speed communication links.

The 3GPP LTE Physical Layer

One of the key changes in the Layer 1 (PHY layer) of the 3GPP LTE is the change from CDMA (code-division multiple access) to OFDM (orthogonal frequency-division multiplexing). One of the main benefits of OFDM is that it reduces the problems associated with multiple paths in the radio channel. In CDMA, a significant amount of processing must be devoted to characterizing and tracking the radio channel to compensate for the effects of fading in the channel.

Figure 1 illustrates the structure of an example LTE subframe. The subframe comprises a number of OFDM symbols. Each OFDM symbol provides the data input for an inverse fast Fourier transform (IFFT). In LTE this may be as many as 2,048 input points for in-phase (I) and quadrature (Q) components.

A subframe can be represented as a resource grid, where each resource element in the grid comprises a single I/Q input point for the IFFT in an OFDM symbol. Resource grids can be layered to provide data to multiple antennas, supporting transmission schemes such as transmit diversity or MIMO (multiple input/multiple output) techniques.

The resource grid is allocated to different purposes. Resource elements are allocated to control channels, data channels, and synchronization signals. The diagram also shows the packetization of data on the channel – different areas of the resource grid are allocated to different users' data as resource blocks. The task of scheduling data transmission and allocating resource blocks to users is performed by the higher software layers in the LTE stack.

3GPP LTE Downlink Processing

Figure 2 shows the processing stages in the baseband section of the 3GPP LTE downlink. The processing for both transmit and receive can be split into two main sec-

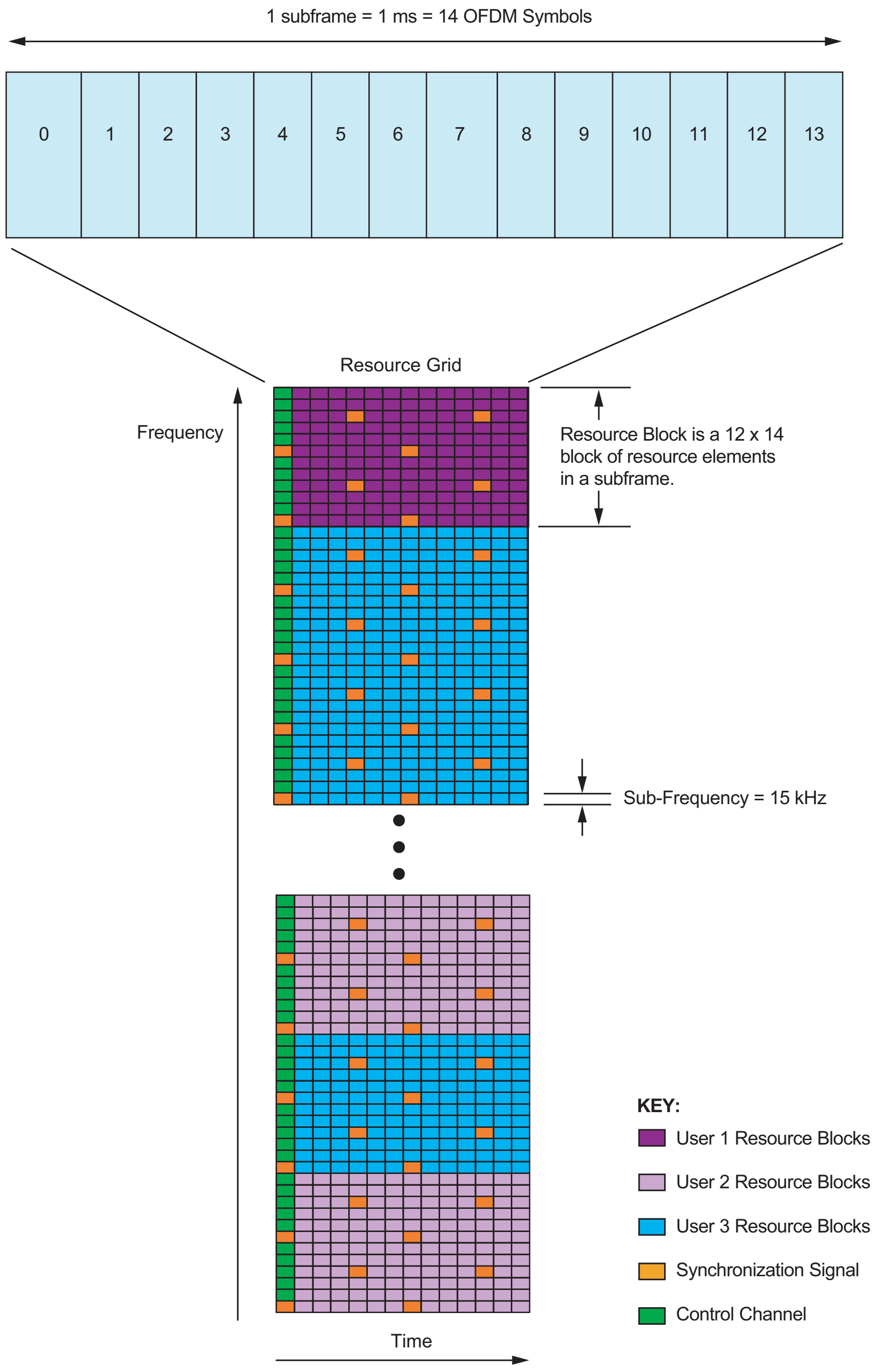


Figure 1 – LTE uses OFDM. A subframe comprises a resource grid, with areas allocated to control, synchronization, and user data. Each column of the grid forms an OFDM symbol that is converted to the time domain by an IFFT.

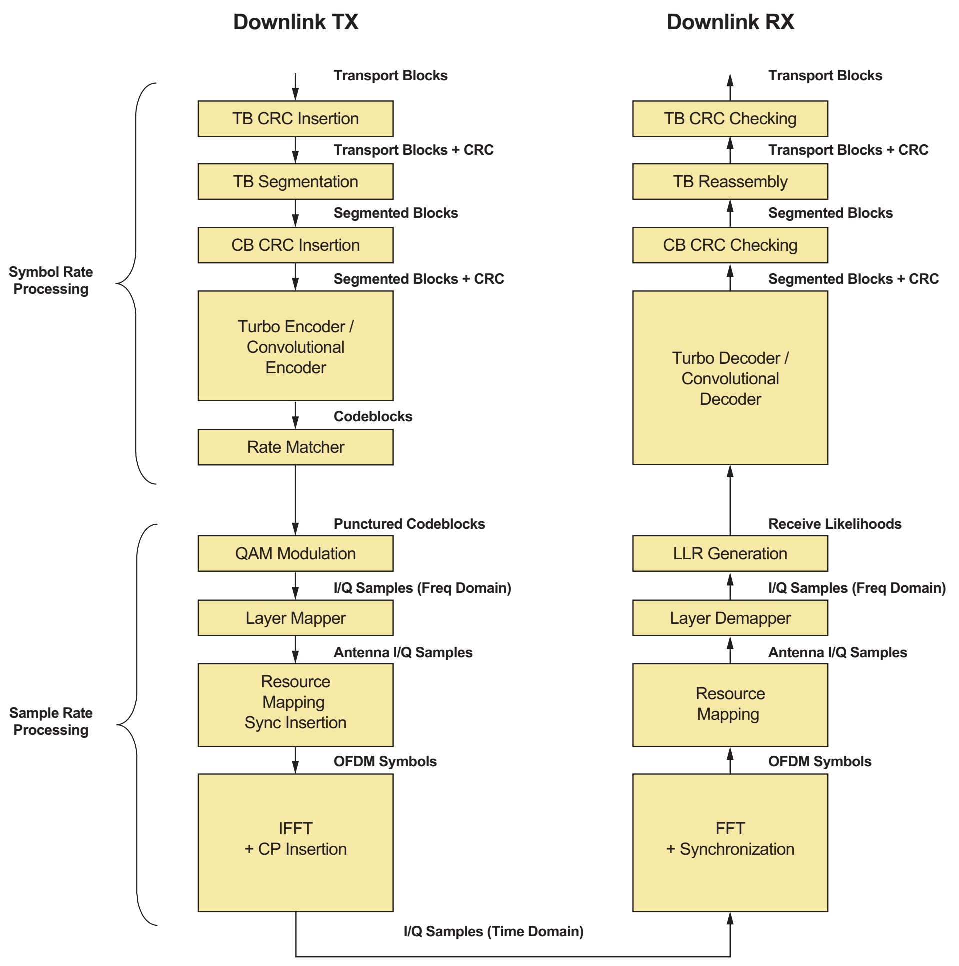


Figure 2 – 3GPP LTE downlink processing: transmit and receive chains

tions: symbol rate processing and sample rate processing.

Symbol-rate processing is centered around forward error correction, used to add redundancy to the data stream in a bandwidth-efficient manner and to recover data on the receiver. Sample-rate processing is centered around the IFFT/FFT that performs the OFDM part of the baseband operation.

Transmit Symbol Rate Processing

The first stage in the Layer 1 (PHY) processing of the LTE downlink takes transport blocks from the media access controller (MAC) layer. Transport blocks have cyclic redundancy checks (CRCs) added, while larger transport blocks may be segmented to ensure that blocks do not exceed a maximum size supported by the forward-error encoder.

Each segment then has a CRC added before it is supplied to the forward-error encoder: for data channels this is a turbo encoder and for control channels this is a convolutional encoder. Following the encoding, rate matching is applied to the output to puncture the data so that it will fill the available resource blocks in the OFDM resource grid. Finally, the data stream is modulated with a specified modulation (QPSK, QAM16, or QAM64) to give a sample value to go in the OFDM resource grid.

Transmit Sample Rate Processing

Samples are first mapped to different antenna layers. This mapping allows support for schemes such as transmit diversity (multiple transmit paths to reduce noise at receiver) or MIMO (MIMO techniques utilize multiple channels between transmitter and receiver to increase data rates).

The next step is to map samples to resource elements in the OFDM resource grid. This stage also adds synchronization signals to the resource grid, allowing the receiver to synchronize with the transmitter. The output of this stage is passed to an IFFT. The IFFT takes the samples from the frequency domain to IQ signals in the time domain, which are ready to be sent to the radio-head for transmission. OFDM requires a cyclic prefix (CP) added to the data, which repeats the end of the time-domain signal at the start of the output. The CP size is determined by the size of the mobile cell and reflections. A sufficiently larger CP is required to remove multi-path effects from the OFDM symbol.

Receive Sample Rate Processing

The receive processing generally follows the inverse of the transmit processing. The first step is to do an FFT on the incoming data to convert the time-domain signal back to the frequency domain. In the reference design, data is received across all OFDM sub-frequencies for all users, but a real mobile user would only decode data on the resource blocks that it had been allocated. Synchronization at this step is also performed to synchronize the system to the start of each sub-frame in the OFDM data. The output of the FFT is processed through a layer demapper that inverts the layer mapping in the transmission.

Receive Symbol Rate Processing

The first step in the receive processing is to take modulation symbols and convert them to individual bits. For turbo-encoded data, this is a set of probabilities in the form of logarithmic-likelihood-ratios for the turbo decoder. For convolutionally encoded data, it is a distance metric that is then fed to a Virterbi decoder. The output of the error correction is checked for CRC validity and reassembled into the original transport blocks.

The LTE Baseband Reference Design

LTE has required a number of new or revised Xilinx LogiCORE solutions to meet

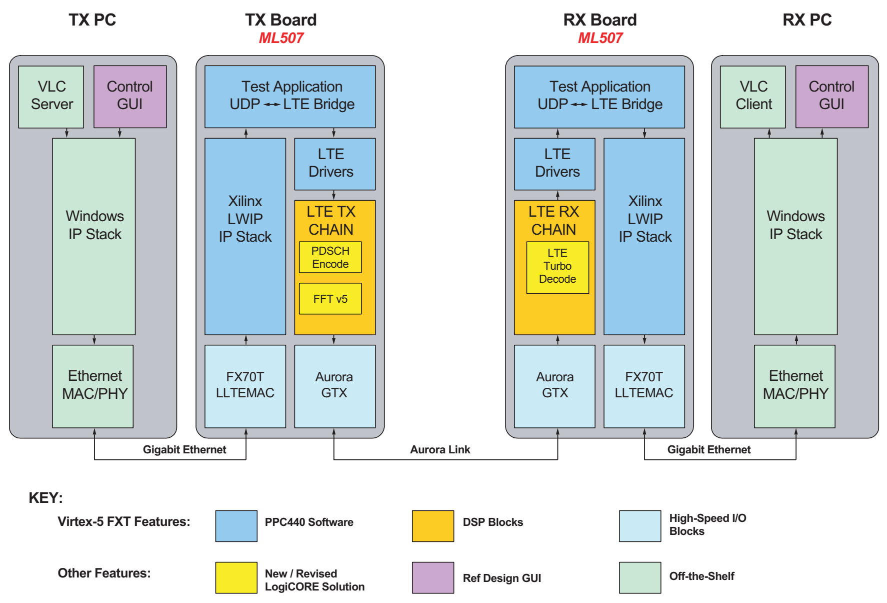


Figure 3 – Xilinx LTE baseband reference design showing system integration of hardware and software blocks. Blocks are stacked into respective protocols so that higher layers use services of lower-layer blocks. Peer-to-peer communication goes from left to right.

the new requirements of the specification. For example, the turbo encoder uses LTE-specific interleavers; the IFFT requires the addition of a variable-length cyclic prefix; and the turbo decoder requires a far higher throughput than previous 3GPP standards.

Verifying these individual blocks presents a number of challenges. First, the LTE standard is still changing and has not been ratified. Second, the size of OFDM symbol (potentially 2,048 points) and the length of the sub-frame and turbo-decoder iterations required mean that a large number of simulation cycles are required to verify the behavior of even a single transport block being decoded. This can limit the test coverage achievable in simulation. Finally, just running unit tests on blocks does not test the macroscopic behavior of systems such as transport-block error rates, or validate that blocks have compatible interfaces.

To address these issues, the design team decided to implement an LTE reference system design that would provide system-level validation of the new LTE LogiCORE IP using real-world data sources such as video streams.

Since the main aim of the reference system was to validate new IP LogiCORE solutions, we wanted to minimize the amount of additional design work for the reference design. We also wanted to minimize the system integration and tool issues and use off-the-shelf boards and IP blocks as much as possible.

Previous reference systems for WCDMA Release 6 of the 3GPP specifications had used a separate DSP microprocessor board coupled with an FPGA board. This meant using different tool flows for software and hardware design, plus the overhead of designing interposer boards to connect the main boards together.

For the LTE baseband reference design, we chose to avoid these issues by gaining early access to the Xilinx Virtex-5 FXT device silicon. The FXT parts integrate on a single chip a PowerPC 440 microprocessor, DSP-enhanced FPGA fabric, and high-speed GTX off-chip communication blocks. This decision minimized our system integration problems (as all the key components were on a single chip) and allowed us to use the standard customer design board, the ML507.

In addition, the design simplified our tool flow. The top level of the system was integrated in Xilinx Platform Studio (XPS), which allows a large number of pre-verified blocks to be pulled into the system from the Xilinx Embedded Development Kit (EDK). XPS was the framework to pull in the various LogiCORE systems that needed validating, plus additional blocks that were implemented in a mixture of VHDL and Xilinx System Generator for the DSP portions of the design.

Implementing on the Virtex-5 FXT FPGA

Figure 3 shows a top-level block diagram of the LTE baseband reference design. The system shown consists of two ML507s, which act as IP packet bridges between the Gigabit Ethernet and LTE protocol stacks.

The reference application is video streaming using the open-source VideoLan Server. The VideoLAN server runs on the transmitter PC, which communicates to the system over the Gigabit Ethernet link. Video is received by the VideoLAN client on the receive PC. A control GUI also runs on the PC, allowing you to set up the LTE blocks to be changed and monitored remotely.

Video packets are transmitted through the LTE software driver and into the LTE downlink transmission blocks previously described. The output I/Q data from the LTE downlink is then sent out over an Aurora link. Aurora was chosen initially over dedicated base station standards such as CPRI/OBSAI protocols because of its early availability on FXT parts.

The I/Q data is received on the receive ML507 board and passed into the LTE downlink receive chain. Processing and communication follows an inverse order to that for the transmission, and finally delivers video packets to the VideoLAN client on the receiving PC.

Figure 3 is color-coded to highlight how the different features of the FXT parts are used. The software elements of the system that run on the PowerPC 440 are highlighted in dark blue. DSP functions comprising the LTE downlink transmit and receive processing are in orange and high-speed I/O blocks are in light blue. New or revised LogiCORE solutions are shown in yellow.

We found we could implement both transmit and receive functionality on a single FX70T part, so by looping back the data on the Aurora link, we could implement the system on a single ML507.

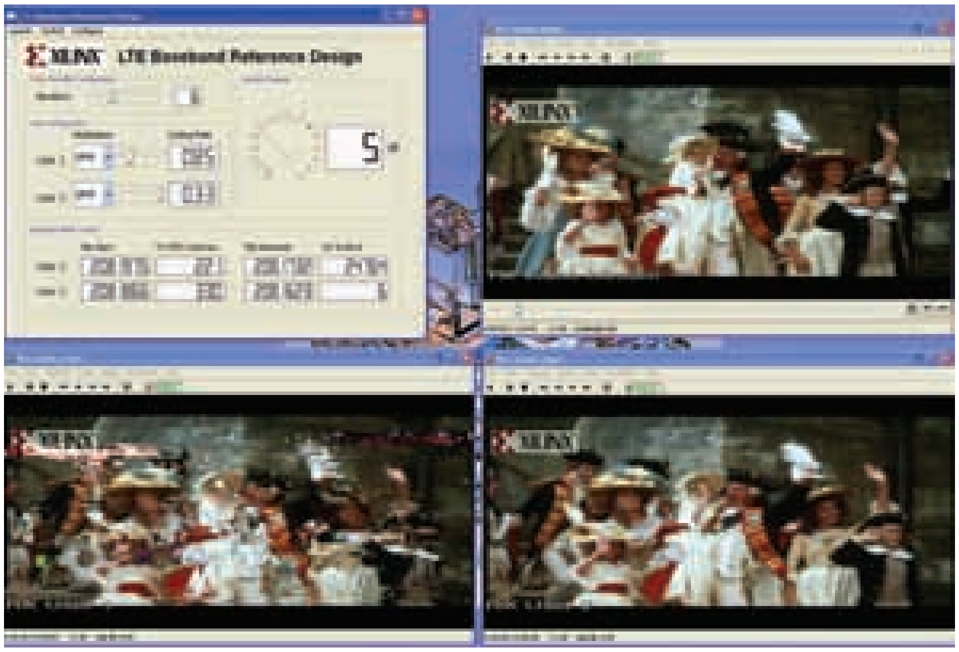


Figure 4 – Screenshot of demo running. The GUI allows variable modulation, encoding rate, and channel noise. Transmitted video is in the upper right-hand corner. Received video (with time lag for software video decode buffers) is shown for two users in the bottom half of the figure. Uncorrected errors appear as artifacts in the video decode.

Final Demonstration System

Figure 4 shows a screenshot of the final demonstration system in operation. The transmit video stream is shown along with the video stream received by two different LTE mobile users after the packets were transmitted over a noisy channel. The control GUI allows variable noise in the channel, along with modulation and encoding rate parameter settings for each user. The number of iterations used by the turbo decode is also variable.

The demonstration allows us to examine the behavior of the LogiCORE solutions when placed in a system with real-world data. In the case of the video streams shown in Figure 4, we used the same modulation scheme (QPSK) for both users. However, for user 1, we started increasing the data encoding rate, thus reducing redundancy in the data.

As we pass an encoding rate of 0.8, we cross a threshold and start to see a significant number of errors in the decoded video stream at this value of SNR. These errors appear as artifacts in the decoded video stream. In a real base station, higher layers in the LTE protocol stack would retransmit extra redundant data for the packet. To maximize efficient use of the channel, the base station has to balance the cost of retransmission against the benefits of lower overall data encoding rates, and also attempt to minimize latency in the system. These may feed into quality of service (QoS) parameters used by higher layers in the LTE protocol.

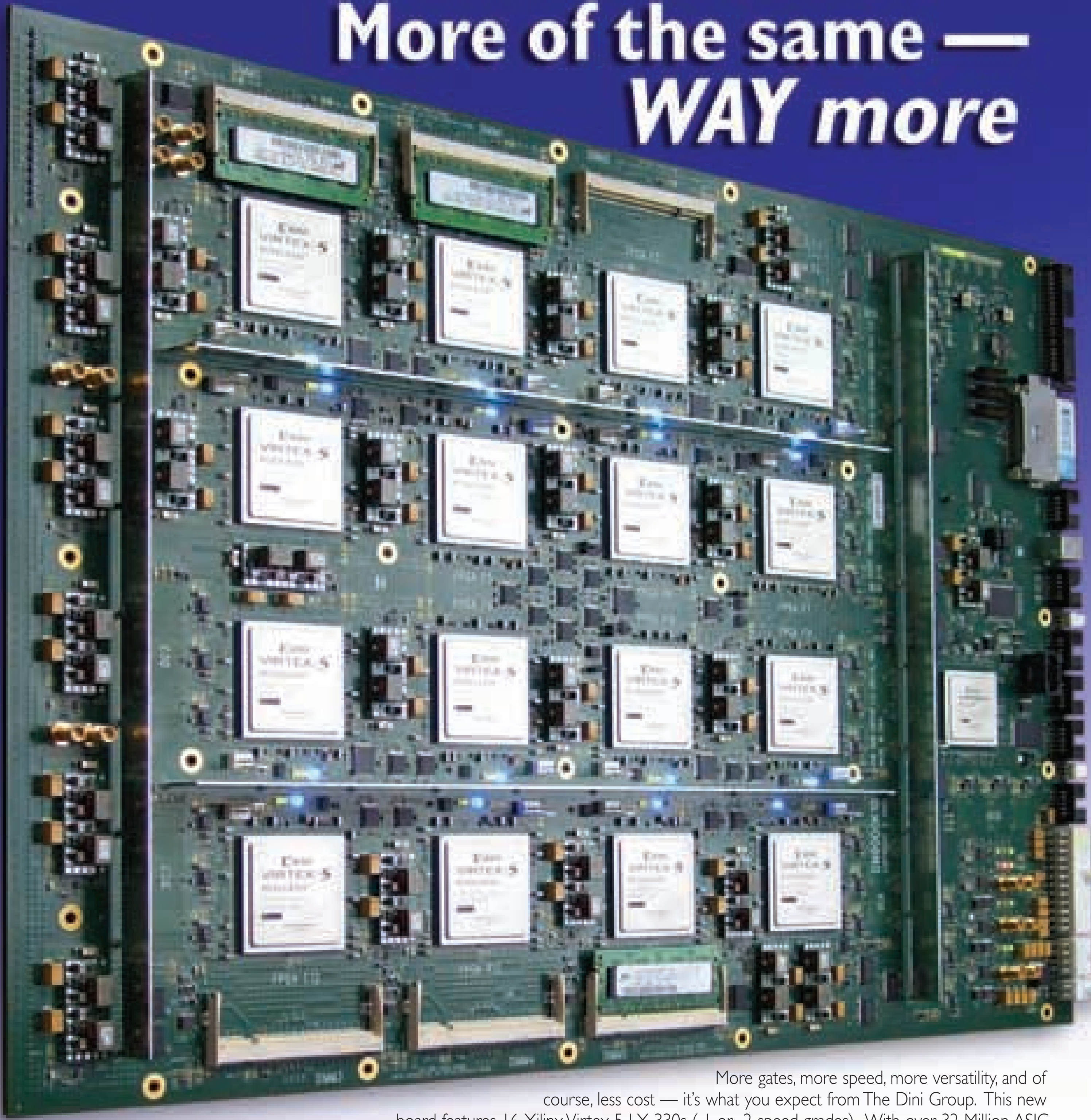
Conclusion

The Xilinx Virtex-5 FXT device provides a tightly coupled integration of processor subsystem, DSP-enabled FPGA fabric, and high-speed communication. Such high levels of integration have allowed both the hardware and software elements of the LTE baseband reference system to be integrated on a single Xilinx FX70T part using standard hardware boards.

By using blocks available in the EDK toolkit to maximize IP reuse, and using Xilinx Platform Studio as a single integration framework, design teams can concentrate on the novel parts of the LTE downlink design. This has allowed rapid development and tracking of changes in the LTE specification as it approaches ratification.

Many thanks to other members of the LTE baseband design team: Beth Cowie, Graham Johnston, Andrew Laney, Neil Lilliott, Jorge Seoane, and Bill Wilkie.

More of the same — *WAY more*



More gates, more speed, more versatility, and of course, less cost — it's what you expect from The Dini Group. This new board features 16 Xilinx Virtex-5 LX 330s (-1 or -2 speed grades). With over 32 Million ASIC gates (not counting memories or multipliers) the DN9000K10 is the biggest, fastest, ASIC prototyping platform in production.

The DiNI Group

User-friendly features include:

- 9 clock networks, balanced and distributed to all FPGAs
- 6 DDR2 SODIMM modules with options for FLASH, SSRAM, QDR SSRAM, Mictor(s), DDR3, RLDRAM, and other memories
- USB and multiple RS 232 ports for user interface
- 1500 I/O pins for the most demanding expansion requirements

Software for board operation includes reference designs to get you up and running quickly. The board is available "off-the-shelf" with lead times of 2-3 weeks. For more gates and more speed, call The Dini Group and get your product to market faster.

www.dinigroup.com • 1010 Pearl Street, Suite 6 • La Jolla, CA 92037 • (858) 454-3419 • e-mail: sales@dinigroup.com

Presenting Power Architecture Technology

The Power Architecture processing platform experiences rapid growth in unit shipments and software support.

by Mike Paczan
CTO
Power.org
paczan@us.ibm.com

Power Architecture processing technology is the common thread for a very broad range of computing devices, from 32-bit microcontrollers to 64-bit ASICs. It is also found in some of the most sophisticated FPGAs available today, including Xilinx® Virtex™-4 FX and Virtex-5 FXT FPGAs.

More than a billion Power Architecture-based chips have been built into electronic equipment since 1991, when the processing platform was first introduced as IBM's POWER (Performance Optimized With Enhanced RISC).

Every Power Architecture-based chip is rooted in the Power Instruction Set Architecture (ISA), a processing specification spanning server and embedded computing capabilities and incorporating the AltiVec vector (SIMD) processing extension. The Power ISA serves as the basis for future advancement of this highly successful processing platform.

Although the Power ISA name may be new to some, its features are familiar to thousands of software, hardware, and tool developers who have worked with PowerPC devices over the past 16 years. Power Architecture technology underlies many well-known chip families, including full-featured Virtex FPGAs from Xilinx; the Cell Broadband Engine from IBM, Sony, and Toshiba; the PowerQUICC line of SOCs from Freescale; the POWER5 and POWER6 server chips from IBM; and, of course, hundreds of products from companies all over the world.

Silicon is just a small part of the complete technology platform: hundreds of companies provide Power Architecture tools, software, systems, and services to speed up and simplify product development. In any given product category – from custom design services to real-time operating services – there are multiple providers from which to choose. The products and services these vendors offer are not only state-of-the-art but in many cases have also been refined by decades of experience in the marketplace.

Expanding Market Opportunities

The Power Architecture community's focus on improving customers' systems design experience, along with our product variety and vendor quality, has led to tremendous growth in the technology platform. Between 2005 and 2006, unit shipments for Power Architecture processors grew by 61%.

Today, Power Architecture processing technology is used in a huge variety of advanced electronics. For example, the world's two fastest supercomputers run Power Architecture processors, as do five of the world's 10 best-performing enterprise servers. Power Architecture controllers are in more than half the new cars on the road. Practically every phone call, e-mail, and Web page is delivered by equipment using Power Architecture microprocessors. Power Architecture devices are also pervasive in consumer game consoles, including the Microsoft Xbox 360, Nintendo Wii, and Sony PlayStation 3.

Looking forward, the communications infrastructure, automotive control, aerospace, and defense market segments will continue to be strongholds for Power Architecture technology. The consumer market – digital televisions, DVD players, set-top boxes, and particularly game consoles – will drive strong double-digit growth for Power Architecture devices through 2011. Xilinx Virtex FPGAs with PowerPC cores will remain a vital part of the Power Architecture portfolio for many of these major markets.

Enhancing Architecture Through Power.org

Dozens of influential companies providing chips, tools, software, services, and systems have joined together in Power.org to develop open specifications and standards for the Power Architecture platform. To simplify the development experience for system designers, Power.org's members will complete a number of projects and specifications this year:

- Searchable online product catalog covering almost all silicon, tools, software, and services offerings available for the Power Architecture technology platform
- Specifications establishing a common set of hardware debugging interfaces for Power Architecture implementations
- Publication of consolidated application binary interface specifications for 32- and 64-bit Power Architecture implementations with neutral licensing that permits wide use and reference by customers, third parties, and other interested parties to support the development of tools, operating systems, and other platform technologies
- Platform requirements specification for embedded systems built on Power Architecture technology
- An open-source Hypervisor optimized for embedded systems
- New models and simulation specifications that support the construction of "virtual prototypes" with Power Architecture processors
- Technical training for Power Architecture products offered online through webinars or in person at our regional Power Architecture conferences in Europe and Asia

Conclusion

Power.org's open-community approach to managing Power Architecture technology has already resulted in many tangible benefits. For instance, in 2007, Power.org's members completed the Server Power Architecture Platform Reference (sPAPR), an open specification for streamlining the development of Power Architecture products for Linux. This specification, built on more than 20 IBM patents, was made available to corporate members of Power.org royalty-free. Along with this specification, a compliant reference design was also released based on IBM's 970MP processor. This was made available with a Linux stack, Hypervisor, and firmware.

By working together within Power.org, our member companies are continuing to improve the product design experience for our many shared customers and to expand business opportunities for the entire Power Architecture community.

For more information about Power.org, visit *www.power.org.*

FPGA Powered – That's it!

FPGA-Module: TQM hydraXC –

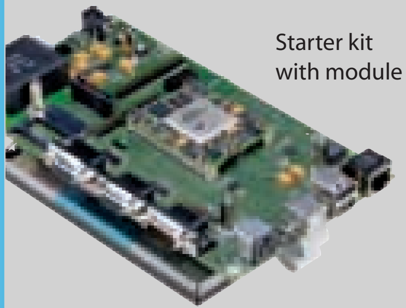
smallest, most universal Hardware Platform for Reconfigurable Computing

- Based on XILINX Spartan 3, Virtex 4 and Virtex 5 technology
- Ethernet 10/100, USB 2.0, RTC
- SPI-, NAND-Flash, DDR2 / SDRAM
- Size 2.13 Inch x 1.73 Inch (54 mm x 44 mm)
- Programmable VCC IOs

Embedded solution with TQM hydraXC for

- Fast time to market
- Economical series production
- Highest flexibility
- Hardware reduction

:: Starter kits available ::



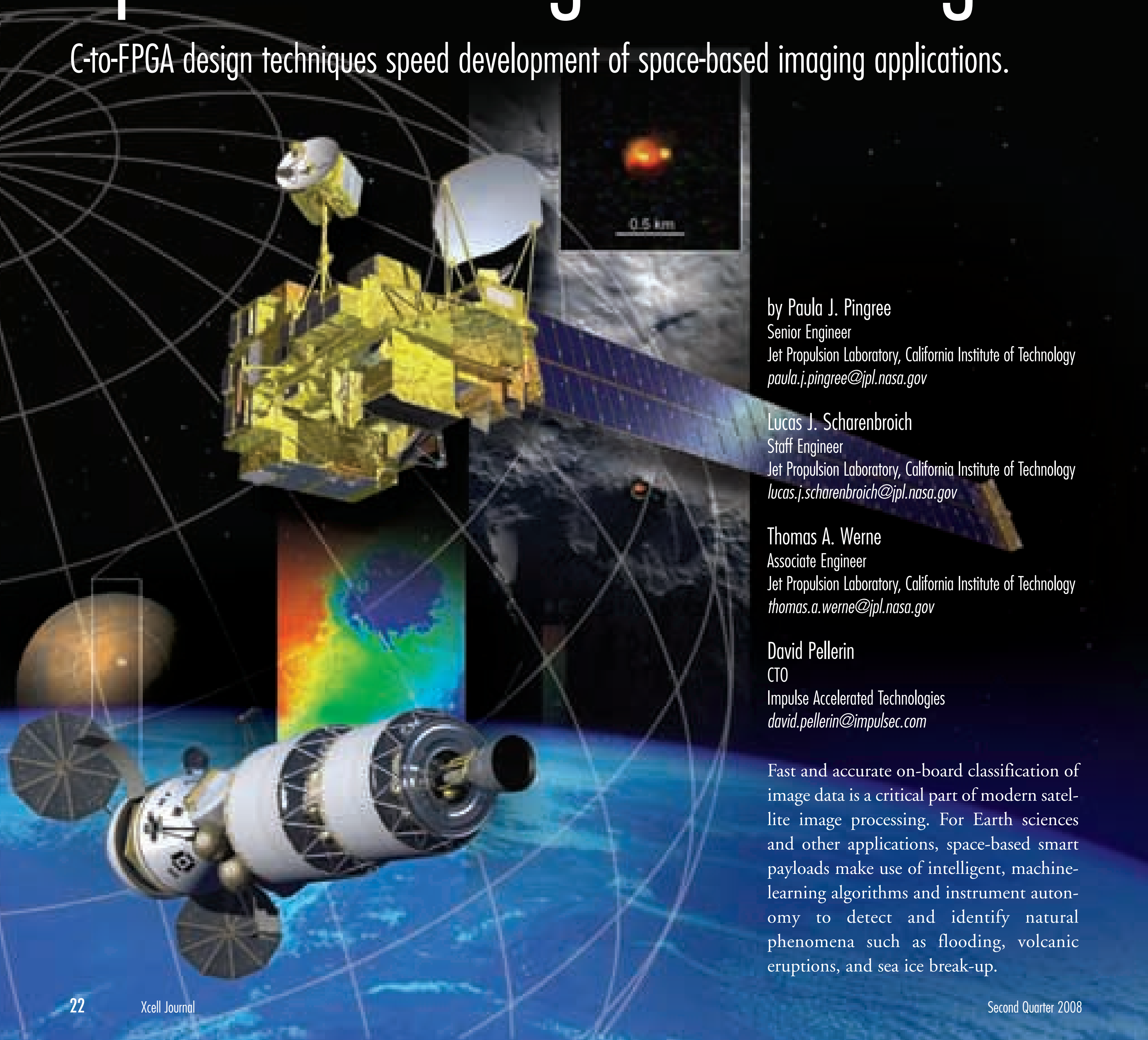
Starter kit with module

TQ components

Email: info@tqc.de
www.tq-group.com
Infoline: (+49) 81 53 / 93 08 - 333

Developing FPGA Coprocessors for Performance-Accelerated Spacecraft Image Processing

C-to-FPGA design techniques speed development of space-based imaging applications.



by Paula J. Pingree
Senior Engineer
Jet Propulsion Laboratory, California Institute of Technology
paula.j.pingree@jpl.nasa.gov

Lucas J. Scharenbroich
Staff Engineer
Jet Propulsion Laboratory, California Institute of Technology
lucas.j.scharenbroich@jpl.nasa.gov

Thomas A. Werne
Associate Engineer
Jet Propulsion Laboratory, California Institute of Technology
thomas.a.werne@jpl.nasa.gov

David Pellerin
CTO
Impulse Accelerated Technologies
david.pellerin@impulsec.com

Fast and accurate on-board classification of image data is a critical part of modern satellite image processing. For Earth sciences and other applications, space-based smart payloads make use of intelligent, machine-learning algorithms and instrument autonomy to detect and identify natural phenomena such as flooding, volcanic eruptions, and sea ice break-up.

The Jet Propulsion Laboratory (JPL), a National Aeronautics and Space Administration (NASA) laboratory, has developed support vector machine (SVM) classification algorithms used on board spacecrafts to identify high-priority image data for downlinking to Earth. These algorithms also provide onboard data analysis to enable rapid reaction to dynamic events (Figure 1). These onboard classifiers help reduce the amount of data downloaded to Earth, greatly increasing the science return of the instrument.

SVM classification algorithms are flying today, using computational platforms such as the RAD6000 and Mongoose V processors. These legacy processors have only limited computing power, extremely limited active storage capabilities, and are no longer considered state-of-the-art. For this reason, onboard classification has been limited to only the simplest functions running on only a subset of the full instrument data: for example, only 11 of 242 bands in the case of the Hyperion instrument on the Earth Observing-1 (EO-1) satellite.

FPGA coprocessors are an ideal candidate for these algorithms. FPGAs can provide significant improvement in onboard classification capability and accuracy when compared to the legacy processing platforms now flying.

To evaluate the effectiveness of FPGAs for SVM algorithms, we implemented a legacy snow-water-ice-land (SWIL) classifier, originally developed for the Hyperion instrument, on the Xilinx® Virtex™-4 FX60 FPGA. To develop the application more quickly, we took advantage of the Impulse C-to-FPGA compiler tools provided by Impulse Accelerated Technologies. These tools support the rapid development of highly parallel hardware algorithms and applications.

This article describes our approach to implementing the Hyperion linear SVM on the Virtex-4 FX60 FPGA, as well as additional experiments that we performed using an increased number of data bands and a more sophisticated SVM kernel. These experiments show the potential for more efficient, higher performance onboard classification using FPGA-embedded algorithms.

FPGAs for Onboard Computation

Onboard computation has become a significant bottleneck for advanced, space-based scientific and engineering applications. Currently available spacecraft processors have high power consumption, are expensive, require additional interface boards, and are limited in their computational capabilities.

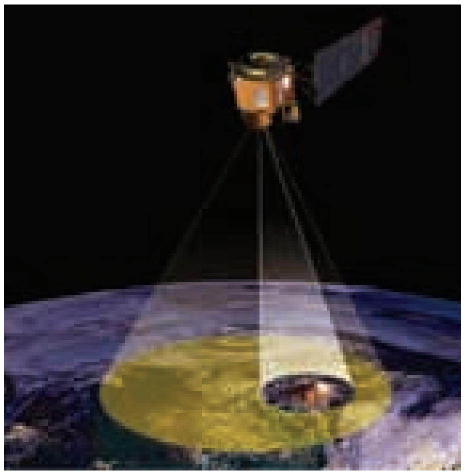


Figure 1 – NASA uses smart payloads to classify Earth image data and reduce the amount of data required for downloading (Image courtesy of NASA).

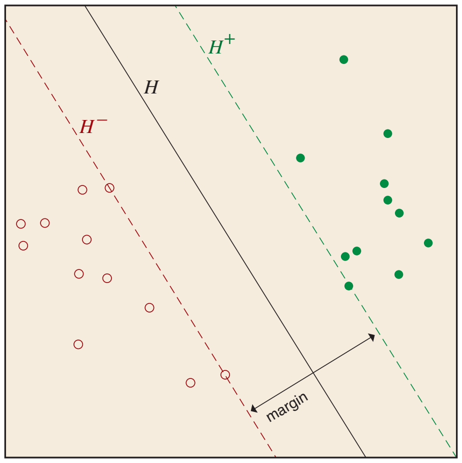


Figure 2 – Calculating a separating hyperplane using an SVM. The circled data points are the support vectors that lie on the margin.

Recently developed hybrid FPGAs, such as the Xilinx Virtex-4 FX device, offer the versatility of running diverse software applications on embedded processors while at the same time taking advantage of reconfigurable hardware resources, all on a single chip. These tightly coupled, single-chip hardware/software systems offer lower power and lower cost than general-purpose, single-board computers (SBCs). FPGA platforms offer breakthrough performance over radiation-hardened SBCs, leading to entirely new architectures for smart payloads.

To evaluate the potential for acceleration using FPGAs, we selected the Xilinx ML410 evaluation platform for development and demonstration of selected smart payload concepts. The Xilinx ML410 evaluation board comes equipped with a Virtex-4 FX60 FPGA that features two embedded PowerPC 405 processors and a large amount of available FPGA logic.

The specific algorithm we chose for our investigations is the SVM classification algorithm. Algorithms of this type have found broad application in general machine learning and classification tasks, as well as for onboard remote sensing. An SVM is a maximum margin classifier that calculates a separating hyperplane between two labeled classes such that the distance to the nearest data in each class is maximized (Figure 2). By selecting such a maximum margin hyperplane, the SVM classifier can exhibit better generalization to new data than other linear classification methods.

The goal of training a support vector machine is to learn a set of weights such that the sign of a weighted sum of dot products between the training data, x_i, and a test vector – t – will correctly predict the class of the new data vector.

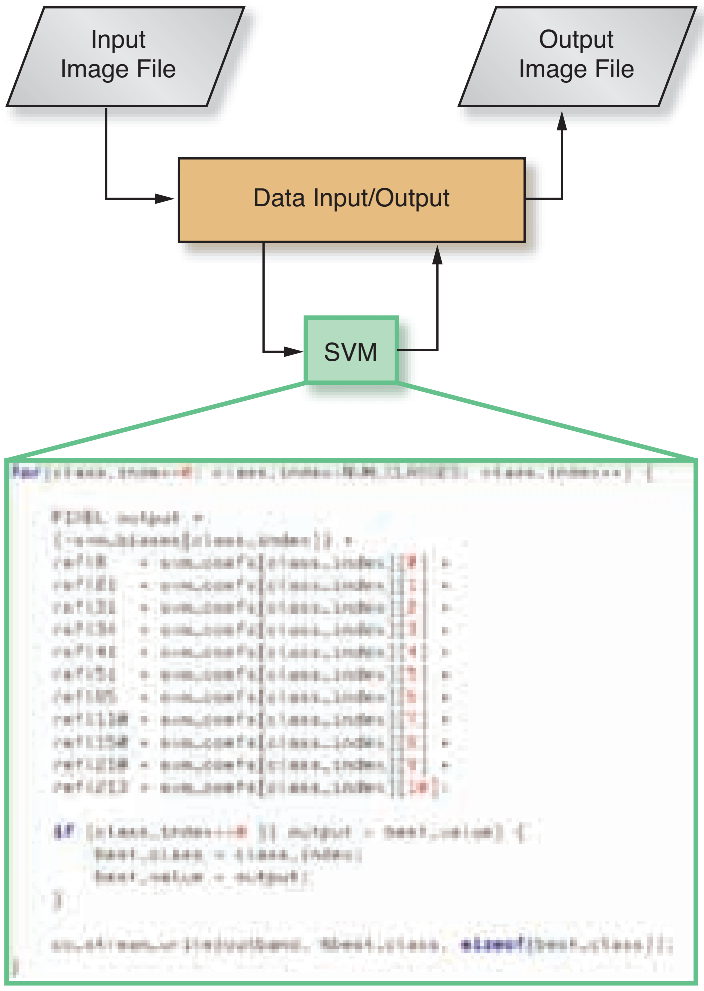
SVMs also incorporate what is known as the kernel trick, a method allowing them to be extended from purely linear to non-linear classifiers. This method involves formulating the training and testing algorithms in terms of dot products and then replacing the dot products with a kernel function that represents a dot product after passing the arguments through some non-linear function. The kernel function permits the high-, or even infinite-dimensional, dot products in the non-linear space to be computed using terms from the original, low-dimensional space.

SVMs are well suited to onboard autonomy applications. The property that makes SVMs particularly applicable is the asymmetry of computational effort in the training and testing stages of the algorithm. Classifying new data points requires orders-of-magnitude less computation than training because the process of training an SVM requires solving a quadratic optimization problem.

SVM training requires O(n^3) operations, where n is the number of training examples. In contrast, testing a new vector with a trained SVM requires only O(n) operations. Faster training algorithms that exploit the specific structure of the SVM optimization problem exist, but the training remains the primary computational bottleneck.

After training the SVM, many of the weights, w_i, will be equal to zero. This means that these terms can be ignored in the classification formula. Input vectors that have a corresponding non-zero weight are called support vectors. Reducing the number of support vectors is key to successfully deploying an SVM classifier on board a spacecraft, where there are severe constraints on the amount of CPU resources available.

Previously deployed classifiers have used such reduced-set methods, but were still constrained to operate on only a subset of the available classification features. Removing such bottlenecks is critical to realizing the full potential of SVMs as an onboard autonomy tool.



```
for (class_index=0; class_index<NUM_CLASSES; class_index++) {

    PIXEL output =
    (-svm_biases[class_index]) +
    ref10   * svm_coefs[class_index][0] +
    ref121  * svm_coefs[class_index][1] +
    ref131  * svm_coefs[class_index][2] +
    ref134  * svm_coefs[class_index][3] +
    ref141  * svm_coefs[class_index][4] +
    ref151  * svm_coefs[class_index][5] +
    ref185  * svm_coefs[class_index][6] +
    ref1110 * svm_coefs[class_index][7] +
    ref1150 * svm_coefs[class_index][8] +
    ref1210 * svm_coefs[class_index][9] +
    ref1213 * svm_coefs[class_index][10];

    if (class_index==0 || output > best_value) {
        best_class = class_index;
        best_value = output;
    }

    co_stream_write(outband, &best_class, sizeof(best_class));
}
```

Figure 3 – Software/hardware partitioning for the SVM algorithm

Partitioning the Problem

When using FPGAs with embedded processors, efficient partitioning of algorithms between software and hardware is important to achieve high performance. For the FPGA-based development of the SVM, we implemented a previously software-only legacy algorithm in the FPGA hardware fabric to take advantage of the FPGA's high-speed parallel processing capabilities. The image file input and classification file output are managed within the embedded PowerPC processor using the CompactFlash card provided on the ML410 board.

In this implementation, the software side of the application is coded in C and compiled to the embedded PowerPC 405 processor using the Xilinx EDK tools. The embedded software application reads an input image file consisting of 857,856 pixels. The image file is read from the CompactFlash card installed in the ML410 board.

The software-side application streams the image data to the SVM, which is also written in C but has been compiled (using the Impulse C-to-FPGA compiler) to FPGA hardware. The SVM hardware process performs the required SVM operation on the image and streams the results back to the PowerPC 405 processor. The processor then writes the pixel classifications (e.g., snow, water, ice, land, cloud, or unclassified) to an output file on the CompactFlash card (Figure 3).

The PowerPC ran the software portion of the task, which sends data to and collects data from the SVM hardware module. We chose to use the PowerPC instead of a MicroBlaze™ processor because the PowerPC can operate at triple the clock frequency of the MicroBlaze processor. Also, the MicroBlaze processor would be instantiated in valuable FPGA fabric, whereas the PowerPC exists external to the fabric.

Because the 256-MB DIMM is the largest source of memory on the board, we used it as main memory for the program. The processor local bus (PLB) is a high-speed bus (compared to the on-chip peripheral bus [OPB]) that allows for fast data transfer to/from the memory and SVM core peripherals. The 16-GB CompactFlash card holds the input and output data files, which are too large to fit on the DIMM. The UART was used for debugging output. The OPB is a low-speed bus that serves as the default interface between the processor and the SystemACE™ interface controller and UART peripherals.

The Impulse C compiler performs these optimizations and generates hardware in the form of either VHDL or Verilog. This hardware can then be synthesized using FPGA tools such as Xilinx ISE™ software and Platform Studio.

In support of partitioned software/hardware applications such as this, the Impulse tools include a library of C-compatible functions that implement a number of process-to-process communication methods. These methods include streaming, shared memory, and message passing. For this application, the Impulse C streaming programming model was the obvious choice.

In Impulse C streaming applications, hardware and software processes communicate primarily through buffered data streams implemented directly in hardware. This buffering of data makes it possible to write parallel applications at a relatively high level of abstraction without the cycle-by-cycle synchronization that would otherwise be required.

Figure 4 illustrates the design flow for C-to-hardware compilation using the Xilinx FX60 FPGA as a target.

On the software side of the application (in this case on the PowerPC 405 processor used for hardware-level testing), Impulse C functions are used to open and close data streams, read or write data on the streams, and, if desired, send status messages or poll for results. In the case of the Virtex-4 FX, stream reads and writes can be specified as operations that take advantage of either the PLB or the auxiliary peripheral unit (APU) interface.

Generating Parallel FPGA Hardware

To create the hardware portion of our application, we used the Impulse C compiler to generate synthesizable HDL files ready to use with the Xilinx EDK tools. In addition to generating HDL files, the Impulse compiler also generates additional files required by the EDK tools, including the needed PLB and APU bus interfaces. The Impulse C compiler performs a variety of low-level optimizations, including C statement scheduling and loop pipelining, saving application developers a great deal of time that would otherwise be spent performing tedious, low-level hardware optimization.

The Impulse C compiler performs these optimizations and generates hardware in the form of either VHDL or Verilog. This hardware can then be synthesized using FPGA tools such as Xilinx ISE™ software and Platform Studio. On the processor side, the compiler generates run-time libraries ready for use on the embedded PowerPC processor.

Validating the Algorithm

The output of the combined software/hardware application is a file comprising a column of integers indicating the resulting class of each pixel in the image. To validate the results of the experiment and see the results, we used MATLAB to reformat the column of integer values to the original pixel-wise dimensions of the image. Each class was assigned an arbitrary color, and the number of pixels belonging to each class was tabulated. We could then easily calculate the percentage of pixels belonging to each class as well as visualize the resulting file of classified pixels as a colored image.

This validation was required to meet two goals of this project. First, it was necessary to validate both the pixel classification results from the legacy (software-only) version of the SVM and the generated hardware implementation of the SVM. This was necessary to verify that the integer and floating-point calculations performed in the FPGA (in hardware) returned the same results (within acceptable margins) as those observed in software currently flying.

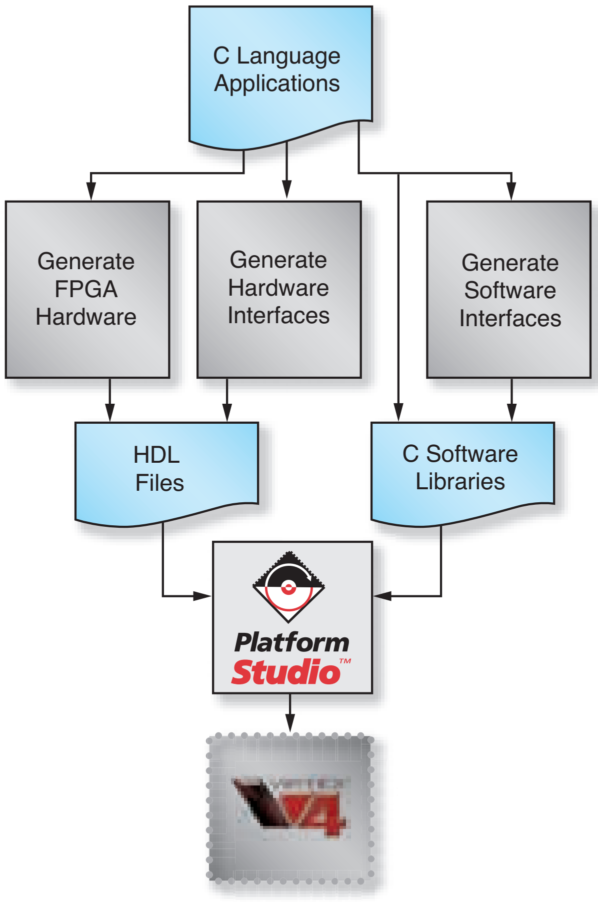
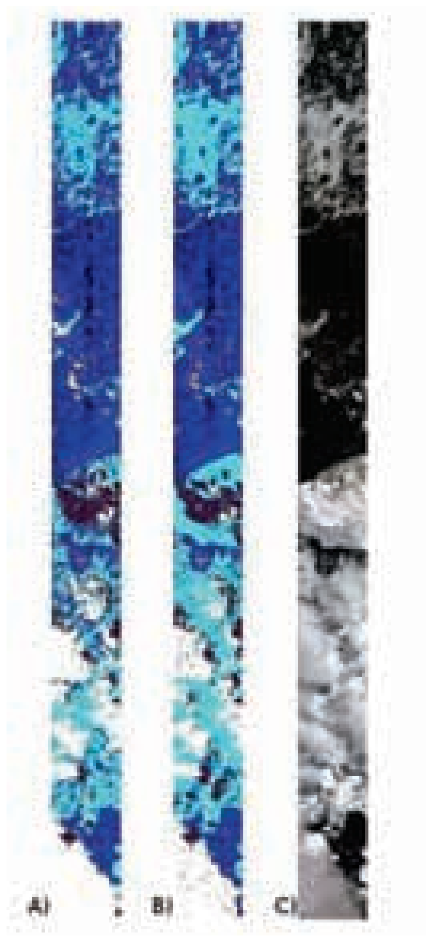


Figure 4 – Design flow from C-code to FPGA-embedded application



[The color key is blue = water, cyan = ice, dark purple = snow, lavender = unclassified]

Figure 5 – A comparison of the results from a) the legacy 11-band SVM implementation, b) the FPGA-accelerated implementation, and c) the original hyperspectral image.

We began the validation process by comparing the pixel classification percentage results to those reported by the SVM used in the currently flying EO-1 satellite. The classification percentages show good agreement, particularly for the snow and water classes.

Image visualizations were also important in this effort. Our resulting visualizations show excellent agreement with the results from the EO-1 (Figure 5).

In addition to the qualitative comparison of the images, we also conducted a pixel-by-pixel comparison of the legacy algorithm results and our classifications. The pixel-by-pixel classification comparison showed that 76.8% of the pixel classifications in our results matched those of the Autonomous Sciencecraft Experiment (ASE). We believed that the remaining discrepancies were caused by the differences in the training data sets of the SVMs, but nonetheless we decided to verify the hardware implementation on a pixel-by-pixel basis.

To completely validate the algorithm, we compared the outputs of a software implementation, using the same input data, to the version implemented in hardware by the Impulse C compiler. The two implementations produce identical classifications on a pixel-by-pixel basis. The combination of the good agreement of our results with the legacy ASE results, as well as the independent results from the software platform, led us to believe that our implementation was valid. All of these validations were performed using a combination of C language and MATLAB programming methods, with no need to use hardware design methods or hardware description languages.

Extending the Algorithm

Having successfully implemented the legacy SVM designed for Hyperion, we then considered two extensions to the C-language algorithm: using a larger number of bands with the same linear kernel SVM and creating a new SVM with a nonlinear kernel. For the expanded linear kernel SVM, we arbitrarily selected 30 of the available 242 bands in the image. For the non-linear kernel SVM, we used the same 11 bands as the legacy SVM, but with a modified kernel. Because training data was not available for the original legacy SVM, we could not generate new SVMs that would be comparable to it, so we used new training data to generate the two new SVMs. We also generated a new, 11-band linear-kernel SVM for comparison to the legacy SVM.

Using the C-to-hardware compiler, we were able to quickly experiment with these alternative implementations and compare results, both in terms of accuracy and in terms of performance. The hardware implementation of these SVMs produced results that agree very well with the software simulations of the algorithms and demonstrate significant increases in overall system performance.

We were also able to determine (using Xilinx synthesis tools) the FPGA fabric utilization percentages for each of these SVMs, as shown in Table 1.

Conclusion

FPGAs with embedded processors are demonstrating levels of performance and efficiency that were previously impossible using traditional processors. Hardware acceleration of SVM algorithms promises to dramatically improve onboard data processing in future science missions.

By using embedded FPGAs such as the Virtex-4 FX60 device, we can implement increasingly advanced SVMs with “room to grow” in onboard resources. Our results demonstrate that we can achieve even our most advanced SVM extension, a polynomial kernel, using just one-third of the DSP blocks and slices available in the Virtex-4 FX-60 FPGA.

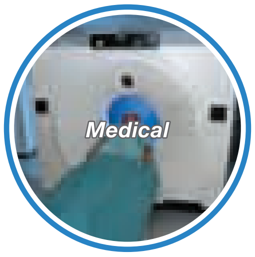
Software-to-hardware design tools played an important role in the fast prototyping and development of these SVM algorithms. The Impulse C tools allowed us to more easily manage the complexity of the application and to experiment with alternative implementations. For testing and algorithm validation, the Xilinx ML410 board provided an excellent and cost-effective development target.

FPGA Resources	Total on V4FX60	Linear (11 bands)	Linear (30 bands)	(2,1) Polynomial
Slices	25,280	1,151 (4%)	2,253 (8%)	2,082 (8%)
Slice Flip-Flops	50,560	1,290 (2%)	1,337 (2%)	2,519 (4%)
Four-Input LUTs	50,560	1,838 (3%)	3,110 (6%)	3,287 (6%)
FIFO16/RAMB16s	232	2 (1%)	2 (1%)	5 (2%)
DSP48s	128	4 (3%)	4 (3%)	12 (9%)

Table 1 – Device utilization for an FPGA-based SVM algorithm

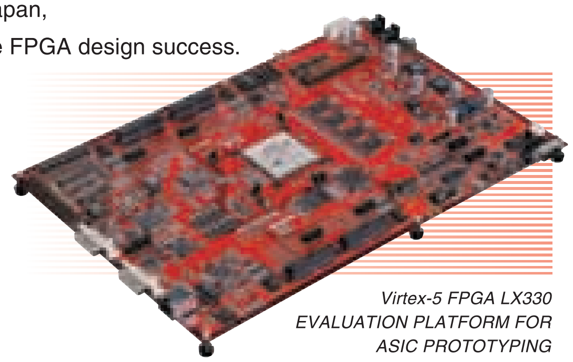
inrevium

by Tokyo Electron Device Ltd.



More FPGA Needs in More Applications

As a leading distributor of Xilinx FPGA solutions in Japan, Tokyo Electron Device has helped customers achieve FPGA design success. To accelerate FPGA designs in various applications, TED has developed application-specific development platforms under the "inrevium" brandname. The inrevium platform has adopted the leading-edge Xilinx FPGA and delivered with reference designs, optional I/O boards and technical supports. Available now throughout North America, Europe and Asia.


Virtex-5 FPGA LX330 EVALUATION PLATFORM FOR ASIC PROTOTYPING

Jump-start your next FPGA design with the inrevium platform, visit us at

http://www.inrevium.jp/eng/x-fpga-board/

™ **TOKYO ELECTRON DEVICE LIMITED**

World Headquarters
1 Higashikata-cho, Tsuzuki-ku, Yokohama, Kanagawa 224-0045, Japan
Tel.+8-45-474-7028 E-mail.psd-sales@teldevice.co.jp

US office
2953 Bunker Hill Lane, Suite 300Santa Clara, CA 95054, USA
Tel.+1-408-919-4772

Using the MicroBlaze Processor to Develop Speed Sensors for F1 Racing Cars

CEA Leti Minatec, Michelin, and EASii IC developed an accurate and real-time optical sensor prototype that calculates both speed and slip angle.

by Viviane Cattin
Research Engineer
CEA Grenoble Leti Minatec
viviane.cattin@cea.fr

Bernard Guilhamat
Research Engineer
CEA Grenoble Leti Minatec
bernard.guilhamat@cea.fr

Angélo Guiga
Research Engineer
CEA Grenoble Leti Minatec
angelo.guiga@cea.fr

Sébastien Riccardi
Hardware Activity Manager
EASii IC
sebastien.riccardi@easii-ic.com

Understanding the dynamic behavior of cars, in particular two-dimensional horizontal speed parameters (also known as the car's slip angle), is critical to optimize performance during races and improve essential equipment like tires.

You can estimate horizontal speed parameters through indirect modeling of wheel speed and yaw-angle measurements; however, this method suffers from modelling errors and other uncertainties. Some optical sensors give measurements that are, in practice, not reliable in wet conditions.

We set out to develop a compact and real-time sensor that could deliver a direct slip-angle measurement with high accuracy and flexibility in various conditions (dry, wet, humid, or snowy). In this article, we'll describe how the Xilinx® Virtex™ FPGA family, with its embedded MicroBlaze™ processors, helped us achieve our goal.

Sensor Measurement Principles

A laser velocimeter is commonly used for taking speed measurements in the printing and textile industries. The velocimeter takes laser images of the surface and uses an autocorrelation function to compute accurate speed estimates.

We extended this approach to measure both longitudinal and transversal speed components. For our sensor, two laser diodes emit elliptical beam shapes to illuminate the road. The first front laser beam runs parallel to the longitudinal axis of the car, while the second back laser beam measures orthogonally. Two linear photodiode arrays collect the reflected light beams on the ground.

A pixel from the front photodiode array is compared to all pixels on the back photodiode array. The back pixels provide information about slip angle or transversal speed, while a corresponding time delay gives longitudinal speed estimation. Figure 1 illustrates the general optical configuration of the sensor.

We then developed an additional functionality to our sensor. Because of the laser beam shape, we inserted a standard triangulation process to measure ground height variation beneath the car. This parameter is useful for understanding the behavior of the vehicle and also improves overall system accuracy.

Finally, we developed an innovative process to avoid optical disturbances. In wet conditions, random spurious reflections disturb systems locally and corrupt the calculation of correlations. Our solution is two-fold: inclining the laser beams at a specific angle and using preprocessing to split the optical signals. We successfully tested our sensor in real time in various conditions.

Embedded Intelligence

During the development of our sensor, we designed and validated the signal processing algorithms using a reconfigurable FPGA. Unlike other targets such as DSPs or ASICs, an FPGA allowed us to adapt routines and software architecture depending on test results and evolving specifications.

Software Architecture

Figure 2 shows the main architecture of the sensor as proposed by Leti, the research laboratory in charge of this project. Three processes run concurrently:

- The height process calculates ground height variations.
- The speed process computes longitudinal speed and slip angle from correlations and height estimations of the car.
- The output process sends sequential outputs on a high-speed, CAN-standard automotive bus system.

These processes share variables through memory access.

Figure 2 includes some key points marked with circles. As in the example previously mentioned, signal chopping improved our ability to take measurements in wet conditions. Still other developments are essential to provide accurate measurements:

- Sampling frequency is the constant parameter that ensures accurate measurements. We controlled the acquisition frequency of diodes so that the ground sampling remained constant during vehicle displacement. A control loop adjusted the sampling frequency according to the speed estimation and prediction, part of the speed process previously described.

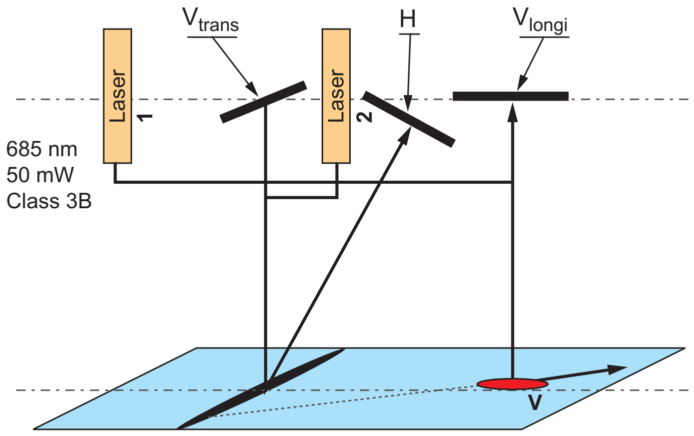


Figure 1 – General optical configuration of a laser speed sensor. Three photodiodes arrays measure longitudinal speed, height measurement, and transversal speed

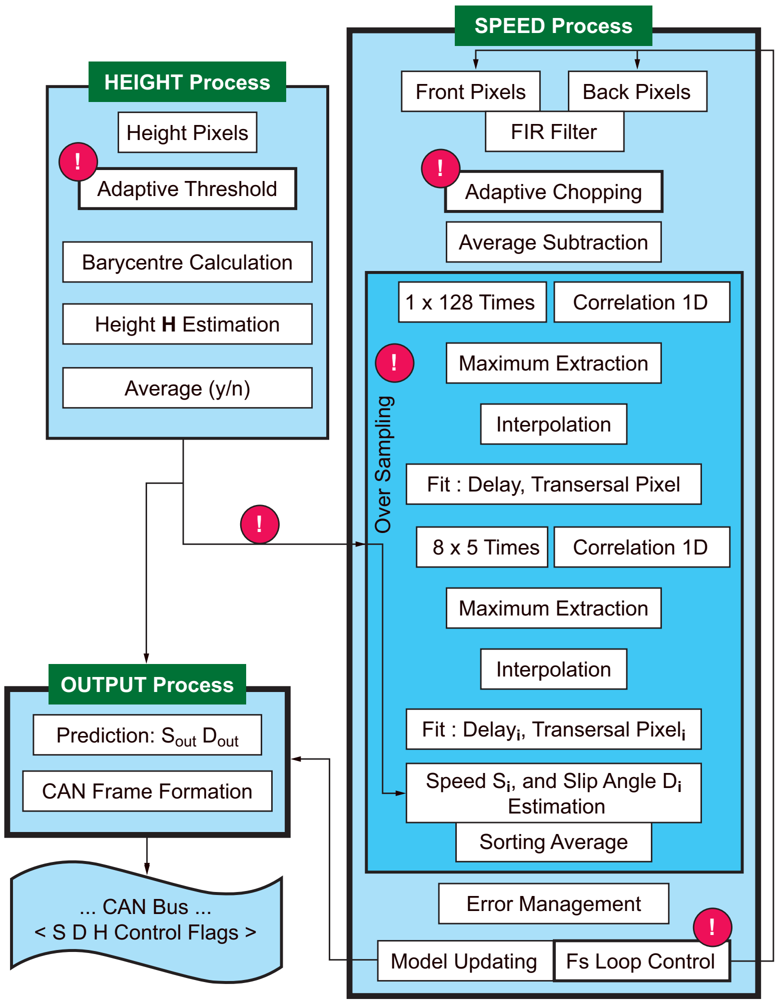


Figure 2 – Main software architecture with key points

We wanted a programmable embedded target that offered the possibility to accelerate software run times, used reconfigurable parameters, and facilitated evolutions of the algorithm.

- Oversampling of ground signals improved measurement accuracy, particularly for the high speeds (above 190 kilometers per hour) of F1 racing cars. At these frequencies, we encountered a technological ceiling of photodiode array integration time. Therefore, we averaged several slightly shifted measurements to improve accuracy (similar to when using very fast oscilloscopes). This routine additionally offered a critical way to evaluate the quality of sensor measurements. If all calculations matched, we could confidently believe in the accuracy of the measurement.
- Taking into account height estimation when calculating speed improves measurement quality; we used the car's ground height variation to refine optical parameters and added a suitable threshold to the signal to avoid sun perturbations on height calculation.

Other smaller processes run in parallel. For example, a specific algorithm evaluates whether speed is equal to zero and communicates that the car is stopped.

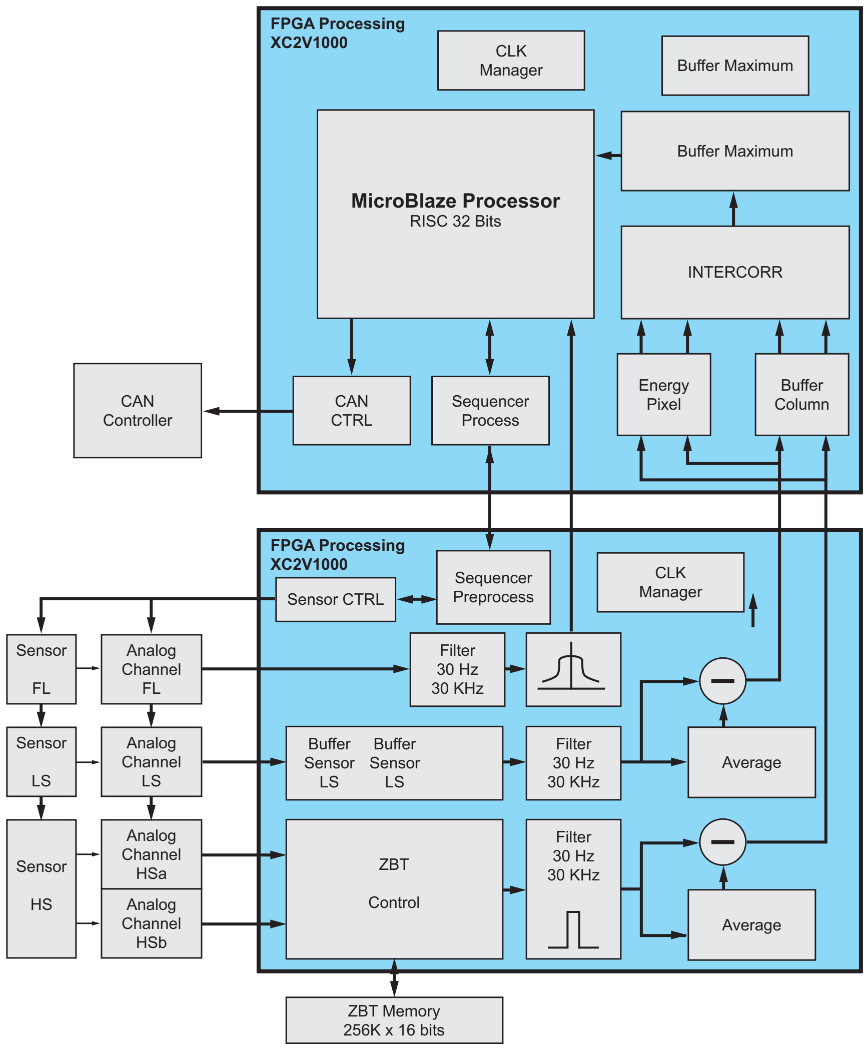


Figure 3 – Main software architecture with four analog inputs, digital-to-analog converter timing, memory storage, calculations, timing, and output control

Hardware Architecture

We wanted a programmable embedded target that offered the possibility to accelerate software run times, used reconfigurable parameters, and facilitated evolutions of the algorithm. An FPGA had all of these features, as well as offering the potential to parallelize correlation calculations.

We used two FPGAs from the Virtex device family: one devoted to preprocessing (sensor control, digital-to-analog converter, memory, filters, averages, height processes) and one focused on the main processing.

We added a MicroBlaze embedded processor inside the second FPGA to implement our high-level algorithm. It includes mainly conditional instructions, with few mathematical calculations. Using this processor allowed us to apply mathematical functions (like trigonometric functions, for example). The soft-core processor is also more user-friendly (based on C language) and facilitates debugging.

Figure 3 shows the architecture as proposed by EASii IC, the company in charge of hardware realization.

Component Choice

The sensor must operate through the severe conditions of F1 racing cars, which include low weights, reduced size, high vibrations, and high accelerations. We used both DC/DC converters and low dropout regulators to avoid major perturbations from the car's power supply.

The output is transmitted over a high-speed automotive CAN bus, thanks to an SJA1000 chip manufactured by NXP

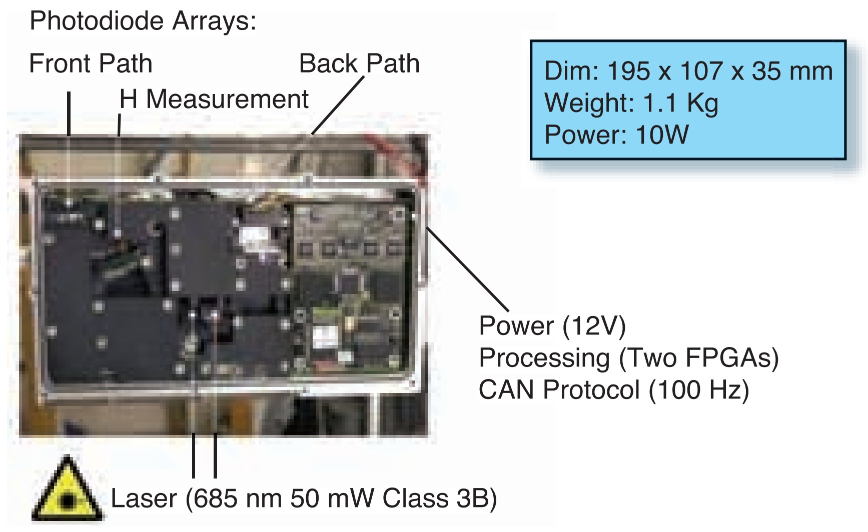


Figure 4 – Sensor photography and main parameters

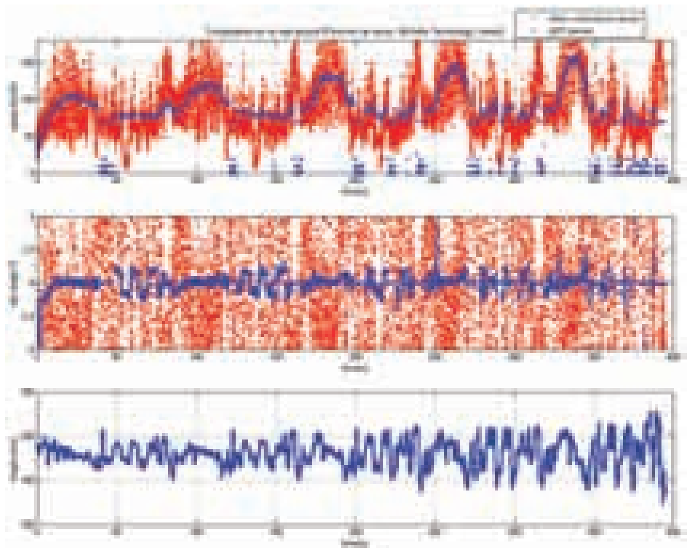


Figure 5 – Comparative result on wet surfaces (LETI sensor in blue, other commercial sensor in red) showing speed (top), slip angle (middle), and height output (bottom)

Semiconductors. To improve the heat transfer of the most consuming components (such as the analog-to-digital converter), we added a thermal paste, which facilitated thermal exchanges with the metal packaging. Additionally, a thermal sensor gives the temperature of the sensor to prevent overheating.

To avoid connection failure and stress breakdowns, we added glue to the largest electronic modules. Most of the basic components of our system are off-the-shelf, which means that further optimization is possible if we applied our system to a specific design. The total power consumption of the sensor is ~10W. The total size (195 mm x 107 mm x 35 mm, including both electronic and optical elements) facilitates rapid mounting on the car, either on the keel for racing cars or anywhere on the chassis.

Performance

We set up and calibrated our prototype at the Michelin Technology Centre, our partner in this project and the eventual end user of the sensor. They have a dedicated tool for simulating high-speed and slip-angle variations. The sensor is designed to work with velocities from 2 to 400 kilometers per hour and can calculate slip angles from -10° to +10°. We reported relative errors of 0.5% on speed, 0.3 mm on ground height variation, and less than 0.1° on slip angle.

We have conducted more than 50 trials since 2003 at the Michelin Technology Centre and with F1 partner teams. These measurements allowed us to significantly improve sensor performance and validate its use under a wide variety of conditions.

Figure 4 presents experimental results in wet conditions. Outputs of the other commercial speed sensor are irrelevant because it was very disturbed by the behavior of water on asphalt. Our prototype presents more robustness; F1 car experts consider these results fully functional.

Conclusion

Various experiments have shown that our prototype addresses the accuracy and robustness necessary for F1 race cars. We assume that these results show the engineering maturity of our sensor, especially with its innovative ability to operate on wet surfaces – a key challenge for non-contact speed sensors.

The simplicity of the global architecture (which includes electronic devices as well as optical parts) makes us confident that our optical speed sensor prototype could be implemented at a low cost. We have already studied a new hardware design that fully exploits the available space between optical elements, reducing package volume by a third.

Future works may include an image sensor to simplify the optical part and fusion with other data (such as accelerometer measurements) to enhance sensor functionalities.

For more information, visit Leti (*www-leti.cea.fr*) or EASii IC (*www.easii-ic.com*) or e-mail *viviane.cattin@cea.fr.*

We wish to acknowledge Sébastien Dauvé, Bruno Flament, William Foucault, Jean-Michel Ittel, Philipe Peltié, and Jean-Charles Vidal from Leti research laboratories and Jean-Baptiste Monnard from EASii IC for their contributions to this project.

FPGA-Based Controller Enables Precise Chemical Analysis

An embedded controller using a Spartan-3 device and MicroBlaze processor provides electronic control of gas flows and pressures.

by Robert Henderson
Development Engineer
Agilent Technologies
bob_henderson@agilent.com

A gas chromatograph (GC) is a chemical analysis instrument that separates volatile substances in a complex chemical sample. GCs help answer such basic questions as:

- "What contaminants are in this ground water"?
- "Is this gasoline formulated correctly"?
- "Are there any residual solvents in this pill"?

The basic mechanism for gas chromatographic separation is the distribution of a sample between two phases (a stationary phase and a gas mobile phase). In modern chromatography, the stationary phase is coated on the inside of a long narrow tube known as a column.

As the sample passes down the column, the different chemical components travel at different rates and are detected at the exit of the column by a detector. In addition to the gas supporting the flow of the sample down the column, each detector also uses between one and three supporting gases (see Figure 1).

Agilent Technologies (spun off from Hewlett Packard in 1999) has been a part of the GC business since 1965. In 1973, Hewlett Packard introduced the world's first microprocessor-controlled analytical instrument, the 5830 GC, and over the years introduced a series of next-generation instruments, including the Agilent 7890A high-performance GC in 2007.

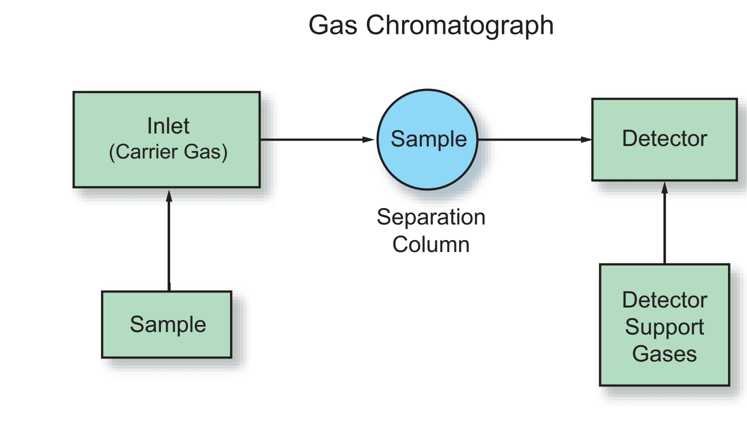


Figure 1 – Block diagram of gas flows in a gas chromatograph

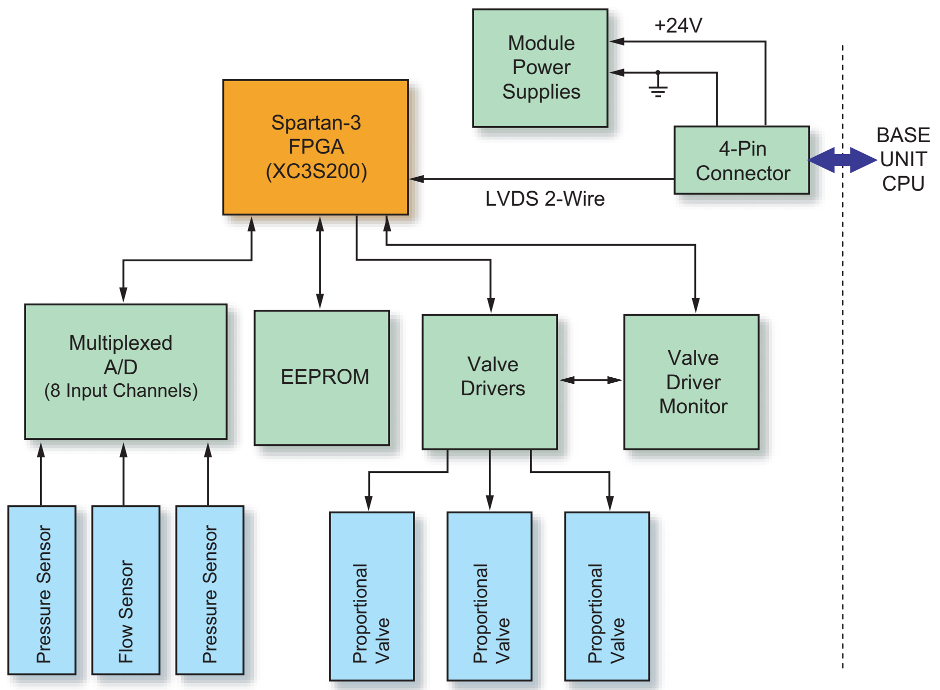


Figure 2 – Block diagram of Spartan-3 FPGA-based pneumatic module

GCs have evolved measurably over the last 20 years, from simple mechanical pressure and flow regulators to sophisticated electronic closed-loop pressure and flow controllers that allow for the storage and retrieval of the entire instrument setup. Additionally, this capability also allows for programmable setpoints and for the logging of any deviation from the setpoint(s).

Spartan-3 FPGA-Based Pneumatic Modules

Because of the disparate different application needs of our customers, we designed the 7890A GC with a modular structure, allowing as many as six pneumatic modules to communicate with a common "base unit" that provides the central CPU and memory, communication ports, keyboard and display interfaces, and power supplies.

Pneumatic modules can regulate up to three channels of gas for each inlet or detector. Each module is essentially an embedded controller that receives commands from the CPU in the base unit, but otherwise operates independently. The module controls the three independent pneumatic channels in a closed-loop manner, offloading the CPU in the base unit from this real-time task (Figure 2).

Four-Wire Interface (Module to Base Unit)

From the base unit to the module, the entire interface comprises an unshielded four-wire cable: two wires are for power and two are for communications. The power supplied is an unregulated, full-wave-rectified +24V, and the communications path is a bus LVDS system.

The LVDS communications path is a differential signal path that minimizes noise susceptibility issues and allows for cable lengths long enough to place modules anywhere inside the instrument, maximizing instrument configurability options. The bus LVDS communications operate in a half-duplex mode, allowing the same pair of wires that sends commands to the module to provide command responses from the module.

Additionally, because the communications interface in the base unit is also implemented with an FPGA, the LVDS communications link between the two FPGAs required essentially no hardware, saving cost, reducing complexity, and improving reliability.

MicroBlaze Embedded Processor Core

In order to implement the embedded controller, we configured the Xilinx® MicroBlaze™ embedded processor in the FPGA using the Platform Studio tool provided in the EDK development system. This tool not only defines, synthesizes, and routes user logic, utilizing MicroBlaze processor IP and associated bus interface designs, but also develops and compiles the program code for the MicroBlaze processor. All FPGA hardware description is in VHDL, and all MicroBlaze processor coding is in C.

Specifying the optional barrel shifter and hardware divider in the microprocessor hardware specification file (system.mhs) enabled the processor system to function as a real-time controller. Performing operations such as bit shifting or division in C code instead of VHDL hardware takes too long for some of the time-critical PID controller processes.

Using the Xilinx-supplied template VHDL file, the user logic interfaces to the MicroBlaze processor OPB bus with an OPB to IPIF (IP interface) block. This block takes care of the OPB bus protocol and interface signals and presents a simplified interface to the user logic called the IP InterConnect (IPIC).

The major user logic VHDL blocks complement the external hardware controlled by the MicroBlaze processor, resulting in the overall FPGA system design (Figure 3). Let's review the major components.

Multiplexed A/D Converter Control

A common delta-sigma analog-to-digital converter (ADC) is multiplexed between eight inputs. With this type of ADC, when the input changes the output filter must be "flushed" of the previous channel's data. A VHDL block manages the ADC and its serial data output.

When the MicroBlaze processor selects a specific channel, the VHDL subsystem reads and throws away the old channel's readings, then reads and averages data automatically from the ADC. The MicroBlaze processor specifies to the VHDL block how many readings to average, and thus can perform other operations (such as executing a new command from the base unit CPU) until an averaged reading is ready.

Serial EEPROM Control

The VHDL hardware manages the commands, addresses, and 8-bit data to and from the serial EEPROM and presents a memory-mapped, 32-bit-word-wide interface to the MicroBlaze processor. The EEPROM stores calibration coefficients for the sensors in the pneumatic module and PID coefficients, as well as other configuration and identification information.

Valve PWM Drive and Monitor

After each PID calculation, the MicroBlaze embedded processor outputs another valve drive setpoint to the memory-mapped addresses for the valve drive. The valve drive VHDL state machine uses these values to generate a pulse-width modulated (PWM) drive to the proportional valves at 65 kHz. This high a frequency is used so that in addition to being above the audible range, it is filtered out well by the inductive time constant of the valve, resulting in a smooth current profile.

Signals from the valve drive are read back into the FPGA for diagnostic purposes. This allows the MicroBlaze processor to examine the voltages and determine if a valve is installed and if the PWM drive is able to drive both high and low.

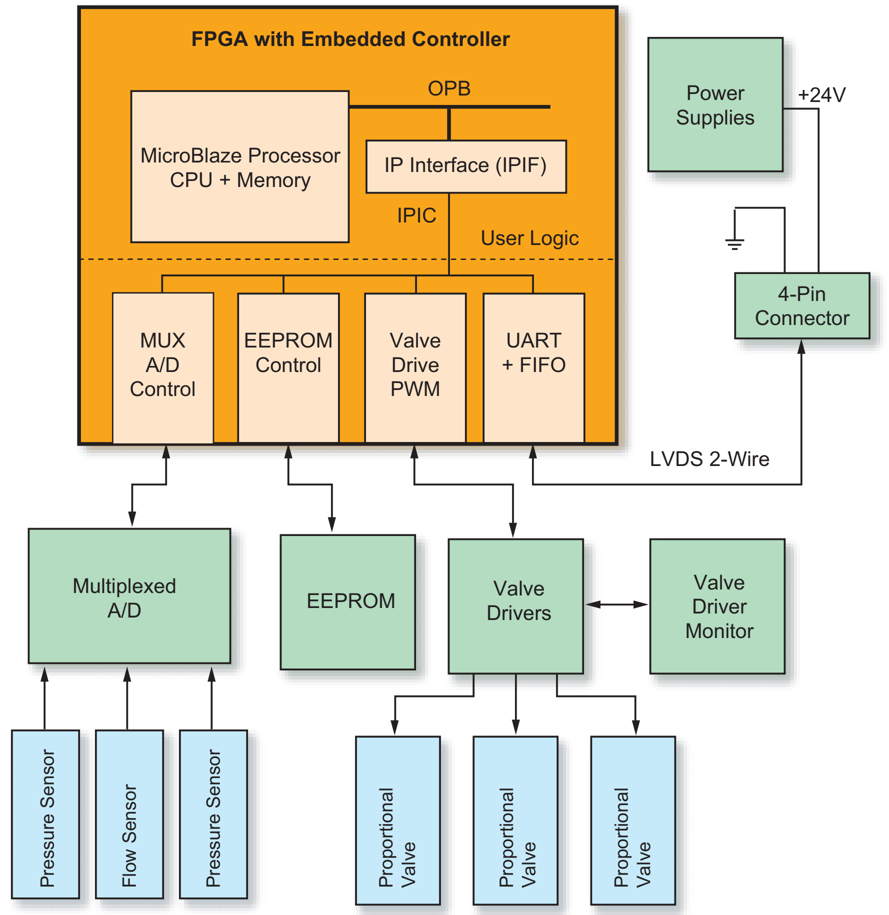


Figure 3 – Overall pneumatic controller block diagram

UART/FIFO

We implemented a UART and FIFO in VHDL hardware to handle the commands and responses with the base unit CPU. Because of the half-duplex protocol, the pneumatic module must only be in the transmit mode during the response to commands.

VHDL hardware automatically returns to receive mode a fixed time after the command response and presents to the MicroBlaze processor a transmit and receive FIFO along with the necessary flags (FIFO empty, FIFO full).

Basic Pneumatic Module Operation

As I mentioned earlier, the physical setpoints supplied by the customer may be static or programmed. They are expressed in physical units such as psig (gauge pressure) or ml/minute (flow rate) and can be set with a resolution of 0.001 psig and 0.01 ml/min. These setpoints are sent at a rate of 50 Hz to each pneumatic control module.

The basic job of the pneumatic controller module is to regulate the pressure or flow to the desired setpoint. This is done with a modified proportional integral derivative (PID) controller.

Command Interpreter

Because there is no clock shared between the base unit CPU and the LVDS pneumatic modules, commands to the pneumatic control module are not synchronized with the closed-loop operation of the embedded controller in the module.

For example, you can send commands while the MicroBlaze processor is in the middle of PID calculations. Because the control loop takes precedence, the command parsing and execution will be temporarily delayed until the MicroBlaze embedded processor is idle. This delay is short enough that it allows the command response to fall within the acceptable response-time limits set by the base unit CPU.

The simple command interface defines commands that set setpoints, read actuals, read and write the EEPROM, specify the gas type (H2, Helium, N2, ArCh4), as well as a number of diagnostic tests on the system, A/D converter, and valve drives.

Filtered Sensor Readings

The sensors read data hundreds of times a second. Two IIR filters in the MicroBlaze processor process the data, which makes use of the barrel shifter hardware configured as part of the processor configuration file.

The first low-pass filter helps reduce raw sensor noise above the bandwidth of the control loop that would result in a wider control band. The second low-pass filter is used on data returned to the base unit CPU. It limits the bandwidth of the data to below 25 Hz so that there is no aliasing of the data in the 50-Hz rate of data to the base unit CPU.

Control Loop

The PID controller function implements a standard PID control with a couple of necessary twists. Figure 4 shows the overall operation of the PID controller.

The first modification of a standard PID is due to the +24V power supply. A fully loaded instrument could have 18 proportional valves to power. Rather than implement a 20W-regulated +24V supply, we decided to use a simple and reliable full-wave-rectified and unregulated supply. A supply like this will track the AC voltage and ripple at twice the line frequency.

The magnitude of the +24V supply can range from 22V to 28V, depending on the AC voltage to the instrument. This results in a variation in the open-loop gain of the system, which can affect stability.

Additionally, the AC ripple on the +24V supply modulates the drive to the valve and can result in pressure or flow variations that are at too high a frequency for the control loop to attenuate away (the disturbances are above the bandwidth of the controller).

We solved these problems by reading the value of the +24V supply with the multiplexed A/D converter and having the MicroBlaze processor calculate a feed-forward compensation term. If the +24V supply increases, the valve drive is automatically reduced to compensate. This requires a divide operation. Because this happens at over 100 Hz and takes away from PID calculation time, the hardware divider was implemented in the MicroBlaze processor system specification.

The second modification from a standard PID controller is due to the pneumatic system itself. The range of setpoints you can define covers a broad dynamic range, with pressure setpoints from 0.2 psi up to 150 psi and flows from 3 ml/min up to 1,250 ml/min.

Not surprisingly, the dynamics of the pneumatic system over this range of operation change a lot. It is possible, for example, to have more than a 30-dB gain change in the transfer function of the pneumatic system (ratio of sensor out to valve drive in) over these ranges. In addition, the bandwidth (poles) of the pneumatic system varies greatly over this operating range.

To obtain good control and step response performance, you can alter the values of the PID coefficients based on the setpoints specified. This alters both the gain and frequency response of the PID controller to match the characteristics of the pneumatic system at that setpoint.

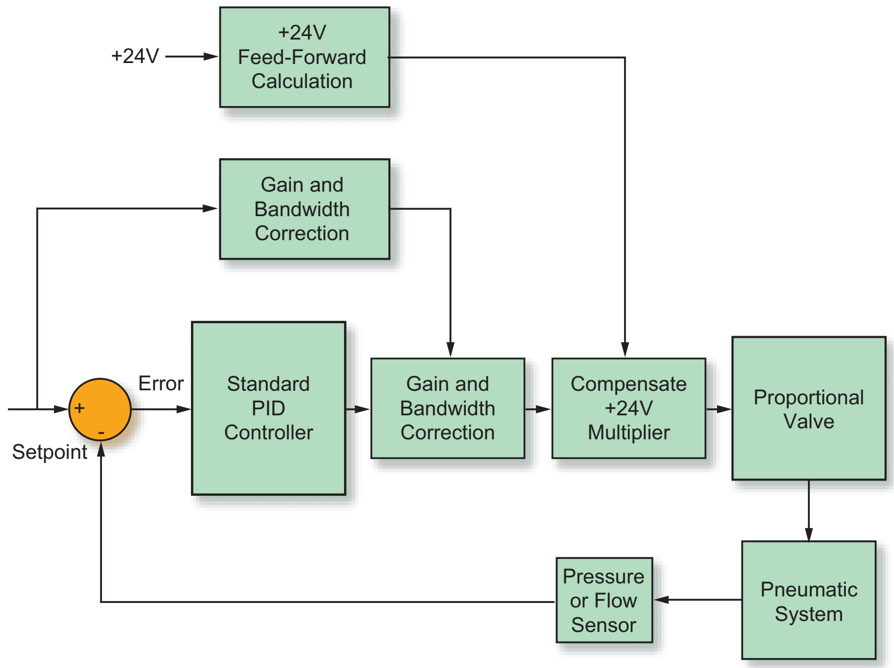

+24V
+24V Feed-Forward Calculation
Gain and Bandwidth Correction
Error
+
-
Setpoint
Standard PID Controller
Gain and Bandwidth Correction
Compensate +24V Multiplier
Proportional Valve
Pneumatic System
Pressure or Flow Sensor

Figure 4 – Modified PID controller

Conclusion

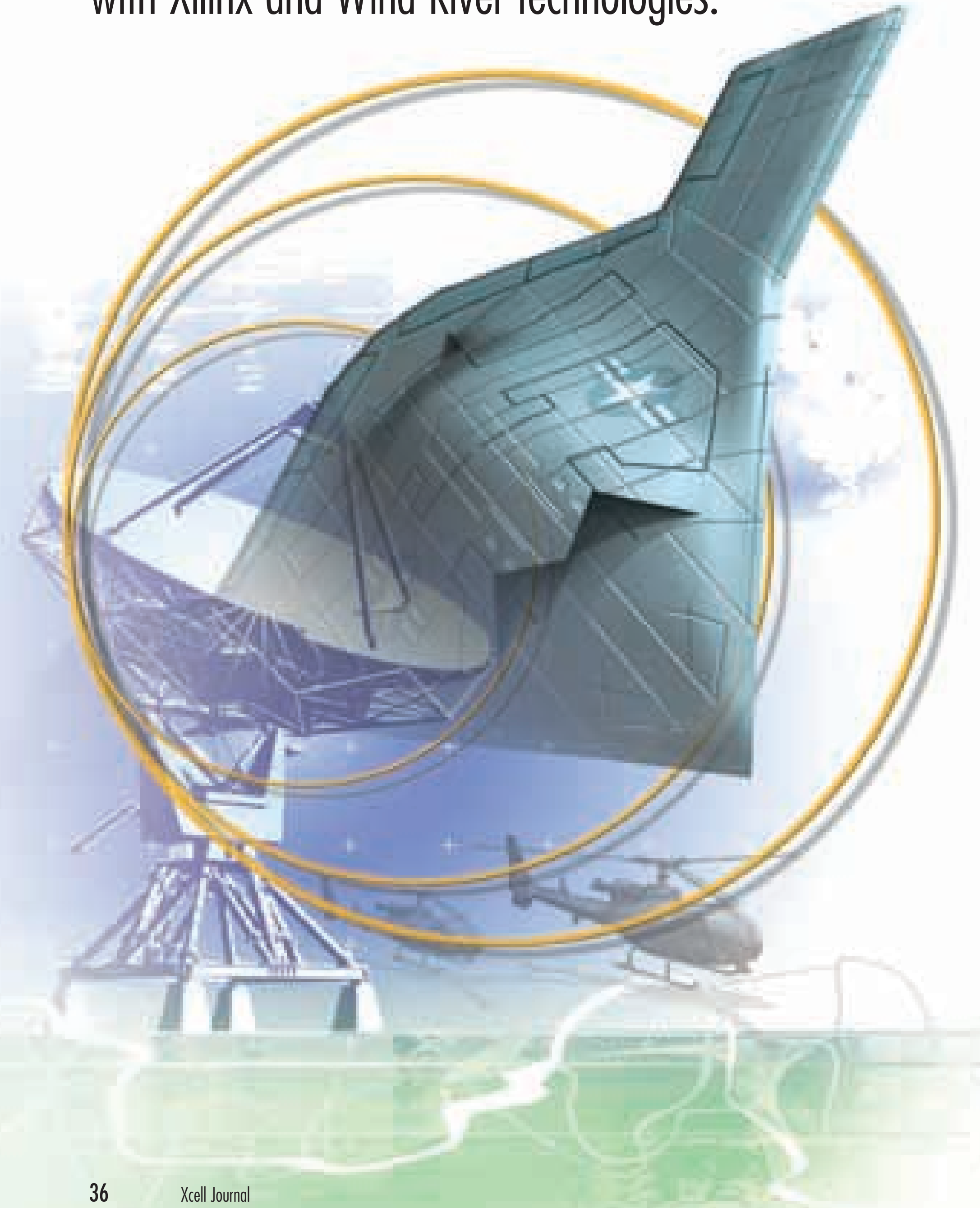
New techniques and improved precision in chemical analysis techniques have enabled our customers to meet increasingly stringent requirements for chemical identification.

Part of that improvement comes from the increased accuracy and precision of gas supplies within the GC. The new Agilent 7890A network gas chromatograph continues this tradition, while the Xilinx Spartan-3 XC3S200 FPGA with MicroBlaze embedded processor helped the product meet its functionality, precision, cost, and modularity requirements.

Reconfiguring the Battlespace

You can develop aerospace and defense applications with Xilinx and Wind River technologies.

by Paul Parkinson
Senior Systems Architect
Wind River
paul.parkinson@windriver.com



The development of modern defense systems presents a familiar technological challenge in that the requirements for increased application functionality conflict with the requirements for minimal footprint in terms of space, weight, and power (often referred to as "SWaP"). This is a particular concern for airborne platforms, especially unmanned air vehicles (UAVs) that have physical constraints.

Some defense systems are physically vulnerable to interception by hostile forces because of their operational role, including intelligence, surveillance, and reconnaissance (ISTAR) assets such as reconnaissance aircraft, UAVs, and land-based sensor systems. If these platforms are to use leading-edge technology, it must be technology that cannot be compromised or reverse-engineered.

In addition, field-based configuration and upgradability are essential to achieve interoperability between new and rapidly deployed coalition forces.

In this article, I'll explain how Xilinx® Virtex™-II Pro and Virtex-4 platform FPGAs, along with Wind River software platforms, are ideal for the development of ISTAR systems, and present design techniques that will enable secure deployment, with the use of partial reconfiguration in the field to help enable interoperability between coalition forces.

Technology Challenges

In previous decades, defense systems used military-grade components from semiconductor manufacturers. These components had long life cycles, which were essential to support the in-service use of deployed hardware for 20, 30, or even 40 years.

In 1994, when U.S. Secretary of Defense William Perry first advocated the use of commercial off-the-shelf (COTS) components on U.S. military programs where appropriate, other governments around the world subsequently adopted this philosophy (to varying degrees). The migration to

Figure 1 – Workbench kernel configuration for Xilinx ML410 VxWorks 6 BSP

COTS has led to a move away from specific military-grade components and toward a greater reliance on industrial-grade components. The latter's shorter supported life cycles could impact in-service lifetimes.

In addition, defense companies have developed custom ASICs at great expense for specialized functions in defense systems, particularly ISTAR systems. Replacing these devices in deployed systems can be prohibitively expensive. There is also an ever-growing requirement to extend the in-service life of ASICs as the operational lifetimes of front-line defense systems are extended.

These challenges can now be addressed through the use of Virtex-II Pro and Virtex-4 FX FPGAs, given their increased gate counts and incorporation of CPU and DSP functionality.

The platform FPGA also has the potential to be used for algorithmic operations, but until recent years this has not been exploited because it is difficult to express software algorithmic operations in million-gate applications in VHDL. However, the ability to program reconfigurable logic from high-level software languages such as C as well as VHDL opens up the potential of these devices to further exploitation.

Platform Development

You can readily exploit the potential of the platform FPGA through the Xilinx Platform Studio (XPS), which generates hardware peripheral and IP definitions in VHDL, closely coupled with the automatic generation of a VxWorks board support package (BSP) in C source code for the PowerPC 405 processor core(s) contained within the Virtex-II Pro and Virtex-4 FX device fabric.

(The details of this approach were previously discussed by Rick Moleres and Milan Saini in their article, "Generating Efficient Board Support Packages" [Xilinx *Embedded* magazine, March 2006] and included integration with Wind River's Tornado 2.2 IDE and VxWorks 5.5 RTOS.)

Since the publication of that article, it has become possible to undertake similar developments using the state-of-the-art Wind River Workbench development suite and VxWorks 6 RTOS. VxWorks 6 provides a number of enhanced capabilities when compared to VxWorks 5.5, including technologies that are critical in defense applications; advanced memory protection for application isolation using real-time processes; and a dual-mode IPv4 and IPv6 network stack, which the U.S. Department of Defense mandated for new programs in 2003.

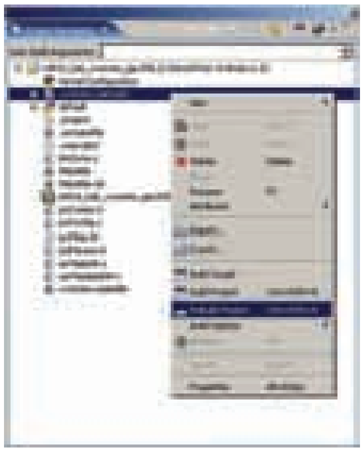


Figure 2 – Workbench build of Xilinx board VxWorks 6 kernel image

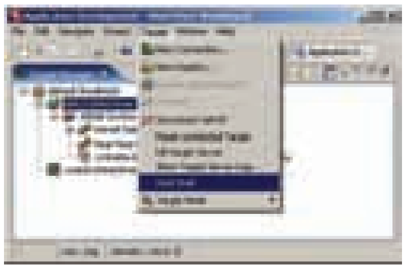


Figure 3 – Workbench target connection to PowerPC 405 core in Virtex FPGA

After XPS generates the VxWorks 6 BSP, it can be configured in the Workbench project configurator (Figure 1). The configurator enables components to be configured in a hierarchical manner and parameter values to be specified. When this has been completed, you can build the VxWorks 6 kernel for your target device from Workbench through automated processes (Figure 2). Once VxWorks is running on the PowerPC 405 core(s), you can use Workbench for source-level application development (Figure 3).

The integration between XPS and Workbench allows you to use a VxWorks 6-based common software platform on Virtex FPGAs for application or algorithm control or gateway network interfaces. This provides a very flexible approach that you can exploit for functions such as image compression and data link encryption to relevant NATO standardization agreements.

System and application software in these devices (known as device software) are often implemented in an architecture-specific manner, using low-level programming languages such as assembly language and custom software schedulers. Although this has enabled exploitation of the hardware's performance potential, it has also made the task of technology insertion and program upgrades more difficult.

You can overcome this problem by using software platforms based on open standards. For example, the Wind River General Purpose Platform – VxWorks Edition incorporates the Wind River Workbench development suite, which uses the Eclipse open-source framework to provide seamless integration between tools for different parts of the device software development life cycle, as well as a consistent GUI for developers. This approach has tangible benefits in terms of developer efficiency, transferable skills, and knowledge retention.

On the runtime side, VxWorks 6.4 introduced 100% conformance to the POSIX PSE52 real-time controller profile, enabling software reuse from legacy systems and the development of new portable applications.

Communications Security in Defense Systems Design

Communications security (ComSec) is an important requirement in many defense systems, especially in UAVs, which often need to maintain continuous communications links for remote piloting and real-time image streaming. These communication links must be secure from eavesdropping and interference by hostile forces, so encryption is required for flight control and sensor data transmissions. You could implement this encryption in software (which places an additional load on the processor) or in dedicated hardware logic. You could also use reconfigurable technology, which provides benefits in terms of performance and secure design (which I'll soon discuss).

Let's consider a hypothetical scenario involving the use of Triple DES-encryption on a communications link. Triple DES encryption provides a relatively high level of security by encrypting data three times using three 64-bit private encryption keys. This is inherently more secure than single DES encryption but will take a processor three times longer to compute, because the processor performs the encryption as three sequential steps.

Given the parallelism inherent in a Xilinx platform FPGA, you can implement three encryption stages operating in parallel, with the output of one stage pipelined into the next stage in 64-bit words. This acceleration enables the passing of data at higher rates over secure downlinks.

For example, a 350-MHz PowerPC can Triple DES-encrypt a data stream at the rate of 1.2 Mbps, whereas a lower power 20-MHz Xilinx Virtex-II Pro FPGA can Triple DES-encrypt a data stream at the rate of 22 Mbps, with each 64-bit word processed in 57 clock cycles.

You could apply this encryption acceleration technique to provide a secure TCP/IP-based communications framework (using IPSec in conjunction with Triple DES or potentially 256-bit AES encryption) and data link protocols. This would involve performing the network packet processing on the PowerPC processor, with computationally intensive encryption offloaded to the FPGA, acting as a coprocessor.

Information Security in Defense Systems Design

Defense systems now contain an increasing number of subsystems. In an airborne platform, for example, there are avionic systems (for flight control), mission systems, and sensor systems for payloads such as electro-optic/infrared sensors (EO/IR) and synthetic aperture radar (SAR) (Figure 3). ComSec in this case refers to the use of firewalls and encryption for the transmission and reception of information securely between networks without transforming the information during transport.

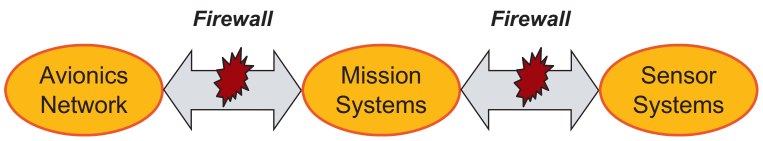


Figure 3 – Military airborne data networks

Data communication between the avionics systems and mission systems will include the passing of global positioning information, bearing, and altitude. However, information with differing security classifications may also need to be transformed securely between applications, subsystems, or networks – this is known as InfoSec.

Preventing Reverse Engineering by Secure Design

The implementation of a system destruct sequence capability is possible for systems that are vulnerable to compromise or capture by foreign forces. However, previous implementations of destruct sequences may not provide a rapid, complete, and irreversible destruction of subsystems; although they may achieve software destruction, they may not sufficiently destroy all of the hardware architecture, especially fixed hardware components such as ASICs.

Thus, it could still be possible to perform reverse engineering on some aspects of the hardware design. To prevent this, reconfigurable technology offers the ability to achieve rapid and complete destruct sequences and prevent reverse engineering.

FPGA devices can be erased completely, leaving no trace of their original application. By asserting a specific signal, the device could be cleared in hardware in a few hundred microseconds. You could also perform the software destruct sequence through programmatic software control issued from a remote system, a sensible scenario in the case of a UAV capture.

You can implement the sequence from a VxWorks-based common software platform connected to the command and control center through a secure IP-based network, permanently and irreversibly erasing the content of a Xilinx FPGA using the PLD API for VxWorks Embedded Systems (PAVE), as shown in Figure 4.

I referred to Triple DES encryption in the context of secure communication links earlier, but you can also employ this method within the content of the Xilinx FPGA (also known as the payload, not to be confused with a UAV payload). When the bitstream is read out from the FPGA, the Triple DES encryption must be decrypted before the bitstream can be decoded. This not only protects the FPGA content from being copied blindly and reused; it also prevents it from being reverse-engineered should the UAV be intercepted.

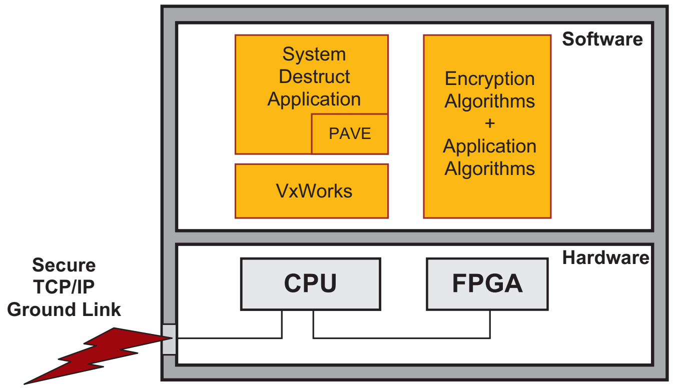

Software
System Destruct Application
PAVE
VxWorks
Encryption Algorithms + Application Algorithms
Hardware
CPU
FPGA
Secure TCP/IP Ground Link

Figure 4 – CPU-controlled reconfiguration of a Xilinx FPGA

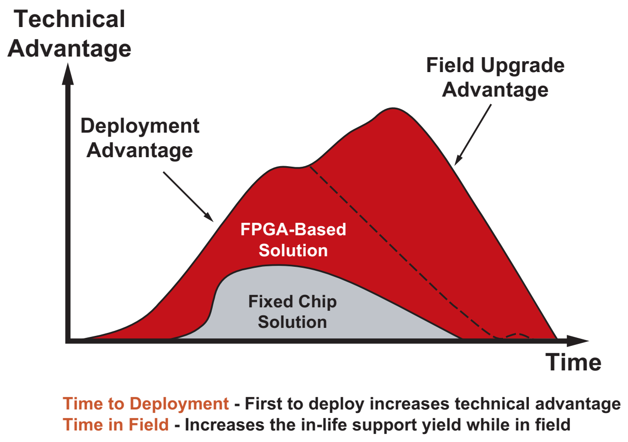


Figure 5 – Reconfigurable technology life cycle

Conclusion

ISTAR systems will spearhead the deployment of new coalitions in response to potential or emerging conflicts, providing vital imagery intelligence and situational awareness from reconnaissance missions. A common software platform and reconfigurable technology could be used to implement codecs that support reconfiguration in the field, assisting in the rapid deployment and interoperability with NATO and/or other coalition forces by sharing encryption keys. Platforms can also be reconfigured on the fly to communicate with legacy or incompatible systems belonging to other coalition members (if a suitable codec exists).

Reconfigurable technology not only provides the ability to rapidly deploy new technology, but also the means to extend the in-service lifetimes of deployed systems. A network-enabled software platform acts as a secure gateway to deliver new content and perform partial or full reconfiguration of FPGAs while deployed in the field. This approach would harvest the architecture shown in Figure 4, but employ an upgrade application instead of a system destruct application.

When considered together, these capabilities provide a technical advantage over traditional fixed-chip solutions (Figure 5) and present a powerful argument for their adoption on defense programs.

For more information, I recommend a paper by Saar Drimer of the University of Cambridge on the use of FPGAs in the design of secure defense systems. To read "Volatile FPGA Design Security – A Survey," visit *www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf.*

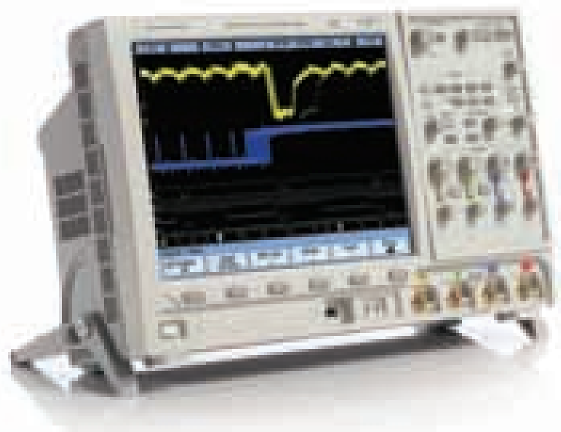
For more information about the Wind River General Purpose Platform – VxWorks Edition, visit *www.windriver.com.*

Save days to weeks on your next FPGA debug cycle

Agilent Logic Analyzers

Up to 612 channels, up to 256 M deep memory

OR



Agilent Mixed Signal Oscilloscopes

4 scope channels + 16 timing channels

+

Agilent FPGA Dynamic Probe

Application software to increase visibility inside your FPGA

=

Fastest FPGA Debug Available

- Perform real-time functional and parametric debug that time-correlates internal FPGA activity with the surrounding system
- Change internal FPGA probe points in seconds without design changes
- Get fast serial bus decode for I^{2}C, SPI, CAN/LIN and RS-232/UART

See what these tools can do for you.
www.agilent.com/find/fpgatools

Agilent Technologies

Hardware/Software Optimization of Embedded Systems

Xilinx FPGAs provide a excellent opportunity to improve system performance during the embedded system design process.

by Peter Thorwartl
CEO/Sr. Systems Architect
So-Logic Electronic Consulting
thor@so-logic.co.at

Franz Summerauer
Hardware Engineer
AVL List GmbH
franz.summerauer@avl.com

Alfred Pöelzl
Software Engineer
AVL List GmbH
alfred.poelzl@avl.com

At the beginning of a new embedded system design, there are so many different solution options that it is hard to predict their impact on system cost, development time, and performance.

Some example of possible choices are:

- Memory devices (DDR, SRAM, flash, SPI-Flash)
- Bus architecture and topologies (PLB, OPB, OCM)
- Number of bus masters and arbitration schemes
- Processor architecture (PowerPC 405, PowerPC 440, Xilinx® MicroBlaze™ processor, ARM)
- Operating system (stand-alone, Xilinx microkernel, Embedded Linux, commercial RTOS)
- Single or multiple processors
- Interprocessor communication (shared memory, shared bus, interrupts)
- Interfaces to the outside world (Ethernet, PCI Express)

In this article, we'll show how you can evaluate these different choices during the whole design process to build an optimized system.

]

Figure 1 -Typical environment for the data acquisition system

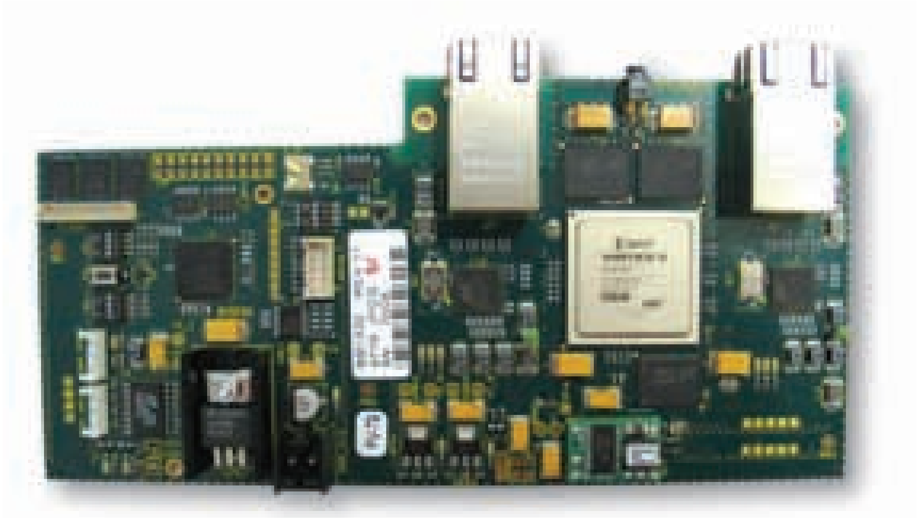


Figure 2 – Board of data acquisition system

System Architecture

AVL offers modular test bed components, highly reliable instrumentation, and advanced tools supporting the development and calibration process for all kinds and sizes of combustion engines (Figure 1). One part of this system is the embedded Gigabit Ethernet card discussed in this article, which collects data from as many as four data acquisition boards through parallel burst interfaces, performs basic data preprocessing, and streams the pre-processed data to a host system through a Gigabit Ethernet interface. Multiples of such acquisitions should be cascaded; therefore, we planned to use two Ethernet interfaces, as shown in Figure 2.

The main components of this card are a Virtex™-4 FX60 device, three DDR RAMs, two Gigabit Ethernet interfaces, and one XC9500XL family CPLD. Two PowerPC processors determine the functionality of the interface board.

Because of high speed requirements, the multi-port memory controller executes checksum generation and the data management of four burst data interfaces. The measurement data is placed into an external high-speed DDR RAM. Both PowerPC processors are responsible for the data flow management and Ethernet communication. Five high-speed interfaces control the external data acquisition boards.

Exploring with a Demo Board

A good way to explore the capabilities of a new FPGA family is to use a demo board like the ML403. First, we implemented the simplest possible embedded system running on one PowerPC (see Figure 3). The CPU executes the software from internal block RAMs. With this small system, we could evaluate the throughput of the Gigabit Ethernet interface. We could also determine the memory bandwidth and speed of the PowerPC.

We estimated the required logic resources like the number of flip-flops, block RAMs, PowerPCs, DCMs from the data sheet to select the device, and the number of I/O blocks, I/O standards, and I/O banks to select the package.

After this first experience, we decided to use the Virtex-4 FX family and FF676 package, which allowed us to use different devices (FX20 with one PowerPC, FX40 and FX60 with two PowerPCs). We had to use different TCP/IP stacks to use the full performance of the Gigabit Ethernet interface, so we selected the Treck stack.

A single PowerPC is too slow to move the data from the burst interfaces to the memory and then to the Ethernet interfaces, so we had to connect the burst data

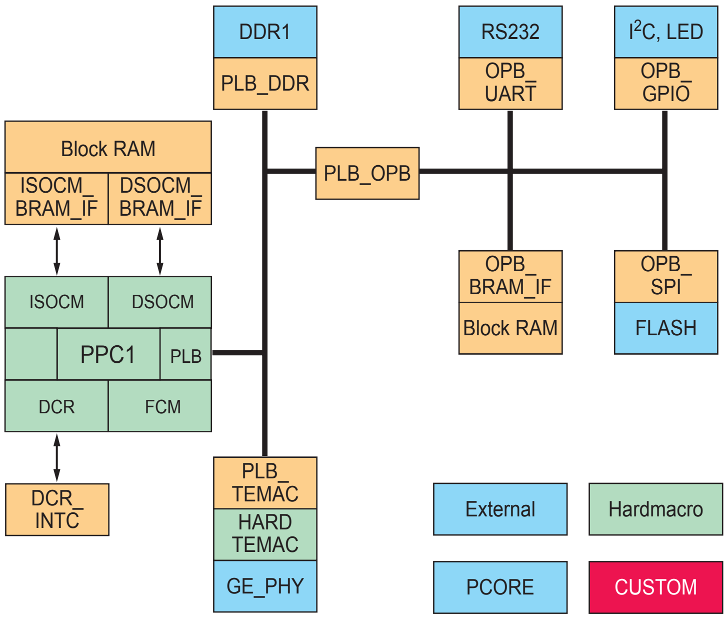


Figure 3 – Block diagram of simple system with ML403 demo board

and Ethernet interfaces directly to the memory. We decided to connect one network interface to each PowerPC.

We are sure that a platform without a real-time operating system will make the development of this application easier. This has to be decided on a case-by-case basis.

Hardware Design

The system uses three DDR RAMs: one for each PowerPC and one for the measurement data storage. We had two design possibilities: a special peripheral that is capable of bus mastering DMA transfer or a multiport memory controller MPMC2 to enable simultaneous access to the data acquisition RAM. We selected the second option.

We were then ready to start developing our new platform. We mainly used Mentor Graphics tools: DxDesigner for schematic capture, Boardstation for PCB layout, Hyperlynx for signal simulation, and IODesigner for the interface between hardware and ISE™ software. The PCB design was awarded the first prize in the Transportation & Automotive category in the Mentor Graphics' 2007 PCB Technology Leadership Awards.

We also developed the peripheral cores for the custom peripherals. We needed two custom cores: one for transmitting data to the transmission link (SO_TLINK_FSL) and one for capturing the block data (SO_NPI_FSL_BURST).

SO_TLINK_FSL

We already had VHDL code for the transmission link with a register interface, so we decided to use a block RAM interface to connect the register memory mapped to the PowerPC. One advantage of this solution is that for every bus (OPB, PLB, OCM, and even LMB for the MicroBlaze processor), a block RAM interface controller exists, so we could easily try the performance on different bus configurations. If connected to the OPB bus, we could share this resource between the two PowerPCs. With this universal approach, we had the freedom to move the control software modules from one PowerPC to the other.

Unfortunately, this solution was too slow for two reasons. First, we saw that bus arbitration takes too many clock cycles; even worse, because we used the OPB bus, we needed an additional PLB2OPB bridge. For the OCM bus, time for bus arbitration would be lower, but we would need a second DSOCM_block RAM interface, which slows down the whole bus frequency.

The other problem was that we could not send further data until we had acknowledgment that the packet was received. One solution would have been to integrate a FIFO in our core, but we could

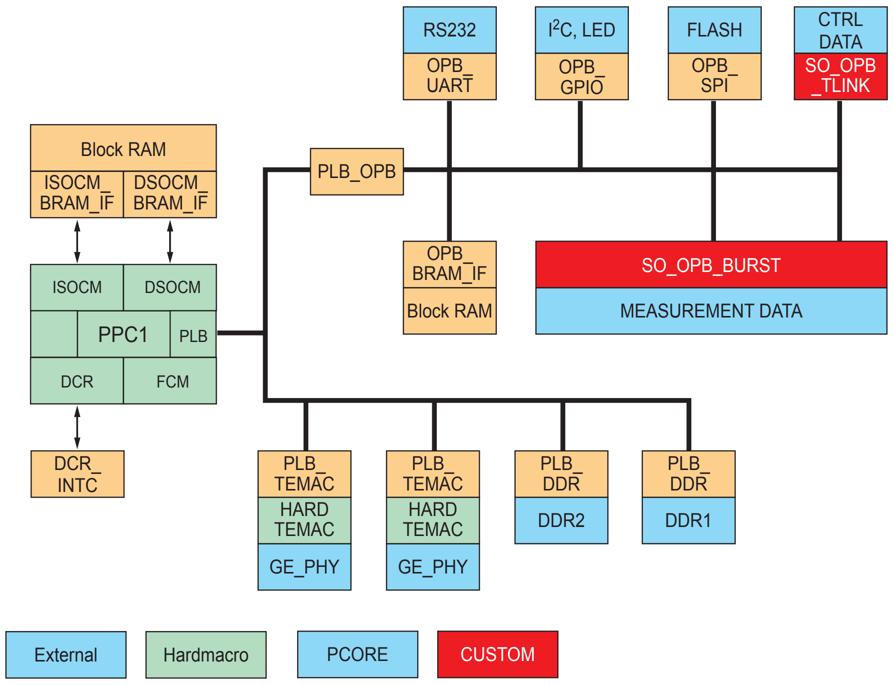


Figure 4 – Block diagram collecting performance data with PLB

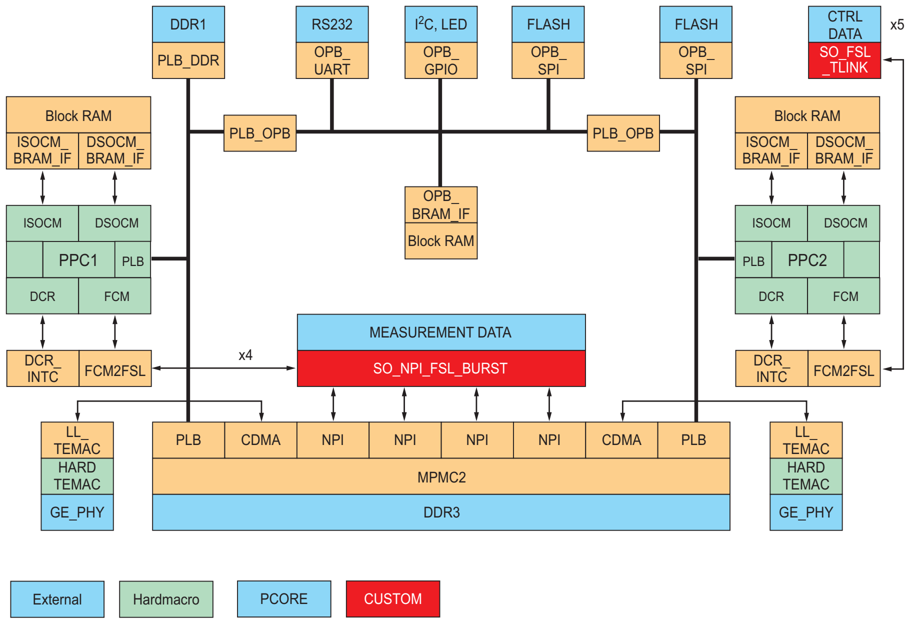


Figure 5 – Block diagram of optimized system with MPMC2

also use the FSL links instead and eliminate both disadvantages. We estimated the best FIFO size, because we could optimize it later under different load conditions. The FSL interface automatically uses the SRL16 for small FIFOs and the FIFO16B hard block for larger depth.

SO_NPI_FSL_BURST

We wanted the SO_NPI_FSL_BURST core to move the data directly to the MPMC2, so we needed to develop an interface between the test card interface and NPI (native port interface similar to the local link interface) of the MPMC2. The PowerPC must control this interface to specify the address location and burst length. We decided to use the FSL interface instead of the GPIO because the FSL is really simple to handle and saves a few clock cycles.

For the simulation of the peripheral cores, we used ISIM to verify the fundamental functionality of our cores. For a system simulation with the PowerPCs, we used ModelSim PE with the SWIFT interface for the processor models.

In the implementation process, we generally used the generated make file from the Xilinx Platform Studio software and added a lot of features like checkout and tagging for source code repository, tar ball, and .zip file generation, different programming file format generation, debugging with the ChipScope™ analyzer core inserter, and compilation of different application software for both PowerPCs.

Our scripts ran the development tools on Windows XP and Linux (CentOS 5.1) and supported different boards and devices.

Real Life

We first migrated our design from ML403 to our new target board to check all external peripherals (DDR, Gigabit Ethernet, SPI, I^2C) (see Figure 4). We can reuse this design later for hardware test for manufacturing.

We then used small internal block RAMs on the data and instruction side OCM buses, with boot loaders for each PowerPC and fast interrupt routines. The boot loader moves the data from the SPI flash to the dedicated DDR RAM. No parallel flash is needed. We use the same SPI flash for booting the FPGA. The CPLD converts the signals from SPI to the usual serial interface for booting.

We implemented the I^2C with the OPB_GPIO core and a software driver without the special core OPB_IIC core, because it was free of charge and we needed only to read an ID tag once after boot.

For the interprocessor communication between the two PowerPCs, we used a shared memory implemented in a single ported block RAM connected to the OPB bus with the OPB_BRAM controller.

We implemented two separated interrupt controllers, one for each PowerPC connected to the DCR bus.

The central core in our design is the multiport memory controller with eight ports: four for data acquisition (SO_NPI_FSL_BURST), two PLB ports, one for each PowerPC, and two CDMA ports to connect the Gigabit Ethernet interfaces (Figure 5).

An interesting design phase was finding performance bottlenecks and fine-tuning the system performance. We connected our board to real test equipment to get real-life data.

One very useful tool for qualifying the results is the combined XMD and ChipScope analyzer for hardware/software co-debugging. You can trigger with the ChipScope analyzer and stop the software debugger, and vice versa.

We also experimented with interrupt priorities, FIFO sizes on the MPMC2, and arbitration schemes inside the MPMC2. We also moved some software modules from PowerPC1 to PowerPC2 and vice versa to optimize system performance.

Conclusion

You can start with a very simple design and upgrade step by step to a higher performance design by boosting Ethernet throughput, boosting DDR throughput with a multiport memory controller, adding data filtering, FFTs, and so on.

In the future, it will be necessary more often than not to postpone important system architecture decisions to later stages during the design process. With an FPGA embedded system, you have many opportunities to remove bottlenecks from your system.

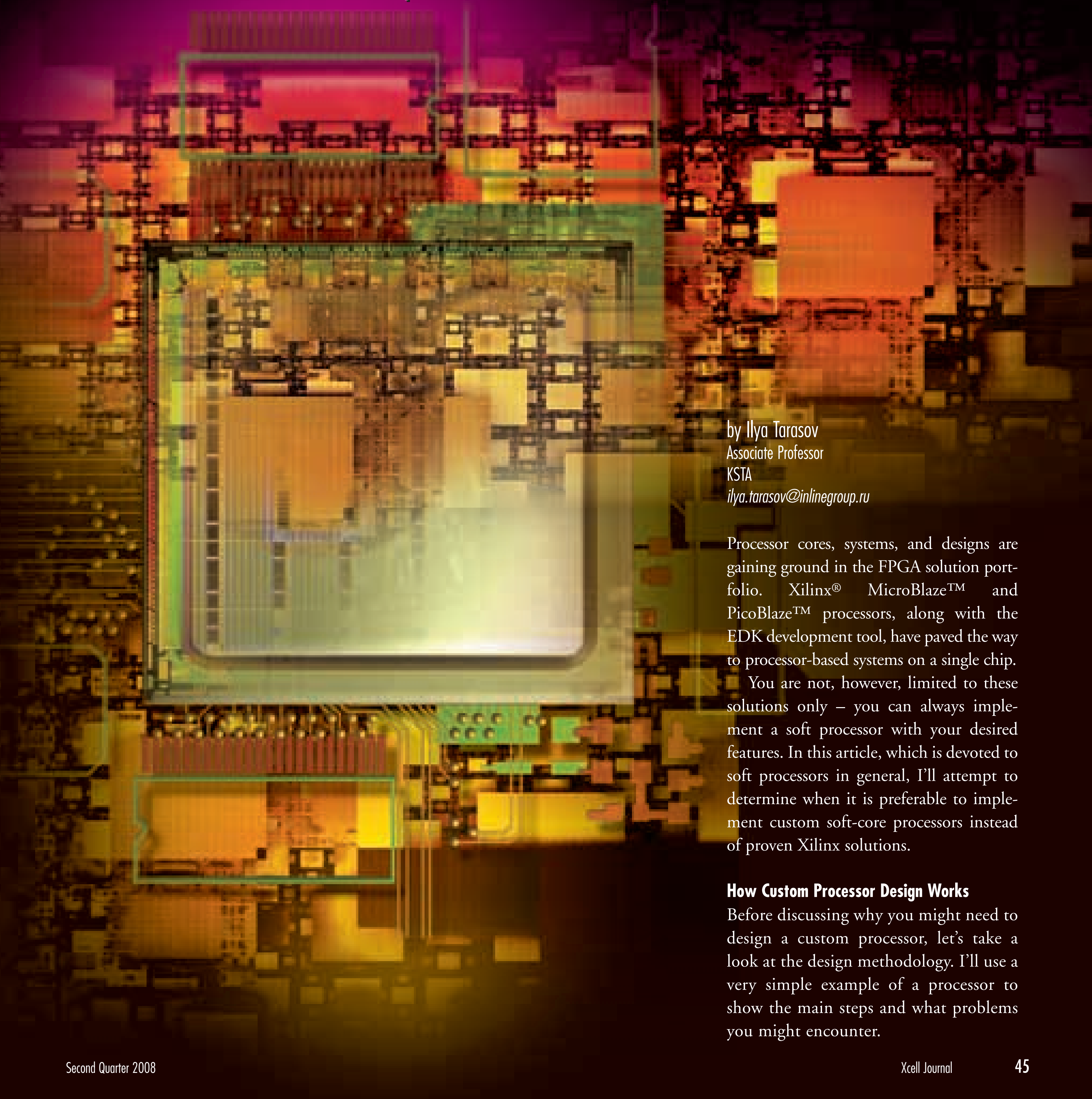
For more information, visit *www.so-logic.net* or *www.avl.com*.

Further Improvements

- Upgrading the actual design environment to the newest versions of ISE software and EDK 9.2.
- Moving all OPB cores to the PLB bus, because all OPB cores are now also available for PLB. In this case, we had to decide which shared peripheral is connected to which PowerPC. As a positive consequence, we could remove the whole OPB bus and two relative large PLB2OPB bridges, each saving 500 slices.
- Running embedded Linux on one or both PowerPCs if you needed to run one of the many applications that are available for Linux.
- Adding cores for double-precision floating-point to offload the host PC application or integrate more DSP functionality into the FPGA.
- Downgrading to a lower performance system with only one PowerPC or, as an even more low-cost solution, replacing the PowerPC with the MicroBlaze processor v7.0, which also has a PLB bus interface.
- Upgrading to a higher performance Virtex-5 FXT FPGA with PowerPC 440 and removing the CPLD, because of the more user I/O pins available on the FPGA in the same package and native boot support from SPI flash.
- Upgrading the FIFO links to serial point-to-point, high-speed links with RocketIO™ transceivers to improve data transfer speed.

Custom Processors in FPGAs

You can build a custom processor with the newest Xilinx FPGAs.



by Ilya Tarasov
Associate Professor
KSTA
ilya.tarasov@inlinegroup.ru

Processor cores, systems, and designs are gaining ground in the FPGA solution portfolio. Xilinx® MicroBlaze™ and PicoBlaze™ processors, along with the EDK development tool, have paved the way to processor-based systems on a single chip.

You are not, however, limited to these solutions only – you can always implement a soft processor with your desired features. In this article, which is devoted to soft processors in general, I'll attempt to determine when it is preferable to implement custom soft-core processors instead of proven Xilinx solutions.

How Custom Processor Design Works

Before discussing why you might need to design a custom processor, let's take a look at the design methodology. I'll use a very simple example of a processor to show the main steps and what problems you might encounter.

The heart of the processor is a finite state machine (FSM). An FSM is quite effective because the latest FPGAs are synchronous by nature; therefore, an FSM is a good base from which to describe processor behavior. Being an FSM, the processor must change its state at the rising edge of the clock signal.

Let's take a straightforward example where we have three 8-bit registers: PC (program counter); RA and RB (general-purpose registers); and an 18-bit command bus named "cmd." This scheme is very efficient for an FPGA with 1,024 x 18-bit organization. Figure 1 shows the behavioral code for this example.

This example illustrates the implementation of a fully synchronous digital design. If you can provide an appropriate sequence of commands for each new program counter value, you'll get a programmable state machine; in other words, a processor.

Of course, you may want to use on-chip FPGA resources for a complete system-on-chip solution without any external memory components. But looking at block RAM waveforms, you might find it necessary to spend at least one clock cycle to read command from memory. To correct this, let's implement a two-stage FSM where the first stage is "read" and the second stage is "execute," shown in Figure 2.

Why (and When) to Opt for a Custom Processor

Designers familiar with the MicroBlaze processor and EDK software may ask why we need yet another core, or how it could replace proven, high-performance products. Indeed, you don't need another processor in general if you can perform a common task using well-known approaches.

But because the MicroBlaze processor is a 32-bit RISC core, it is possible to achieve a significantly different solution. Let's analyze when these steps might lead to valuable results.

DSP Algorithms with High Parallelism

In this case, FPGA performance is determined by the structure of hardware multiplier blocks, which realize certain algorithms of data processing. There may be many such data paths with embedded multipliers and XtremeDSP™ solution slices, and no processor core can achieve comparable benchmarks given the highly paralleled nature of the DSP block array.

You may want to control this array with standard interfaces and protocols, collect statistics, and perform additional tasks in each DSP block. This is a job for processor cores. But you can't just place the same number of MicroBlaze processor cores as embedded multipliers on the FPGA. For these purposes, simple processor cores with little register sets, short commands, and small command memories (even based on distributed RAM) may prove very useful.

Non-Symmetrical Register Architecture

A MicroBlaze embedded processor realizes a symmetrical three-address register architecture. This means that any registers may perform the same operations (with few exceptions for MicroBlaze processors), while general commands may choose operands and destinations separately.

A three-address architecture with many registers is good for compilers, because many variables can be loaded into the processor without needing to be stored back into memory. If you have fewer registers, some variables may be forced to unload when the number of active variables

```
-- register declaration omitted for saving article space
process(clk)
begin
  if rising_edge(clk) then
    case conv_integer(cmd) is
      when 0 => pc <= pc + 1; -- no operation, but we must remember to increment program counter
      when 0x100 to 0x1FF => RA <= cmd(7 downto 0); pc <= pc + 1;  -- load imm value to RA
      when 0x200 to 0x2FF => if conv_integer(RA) = 0
                             then pc <= cmd(7 downto 0);
                             else pc <= pc + 1;
                            end if;  -- jump if RA = 0
-- we can add any 'simple' commands, which will affect any desired registers
   when 1 => RA <= RA + RB; pc <= pc + 1;
--- etc, with format: when <value> => RA <op> RB; pc <= pc + 1;

-- also, let's look on the parallel data processing and combined commands
    when 100 => RA <= RA + 1; RB <= RB – 1; pc <= pc + 1; -- different operations with two registers

-- we may also add other complex commands, which have no direct analogs in common instruction sets

    end case;
  end if;
end process;
```

Figure 1 – Simple decoder and ALU for custom processor

```
-- in architecture section
signal st : bit; -- state of core: 0 – fetch, 1 – execute;
-- after 'begin' keyword
process(clk)
begin
  if rising_edge(clk) then
    case st is
      when '0' => st <= '1'; -- simple wait for BRAM, go to EXECUTE state
      when '1' => st <= '0'; -- return to FETCH state
           case conv_integer(cmd) is
           when 0 => pc <= pc + 1;
            …. – all of the 'case' code from the previous example
           end case;
    end case; -- st
  end if;
end process;
```

Figure 2 – Simple processor engine for use with block RAM

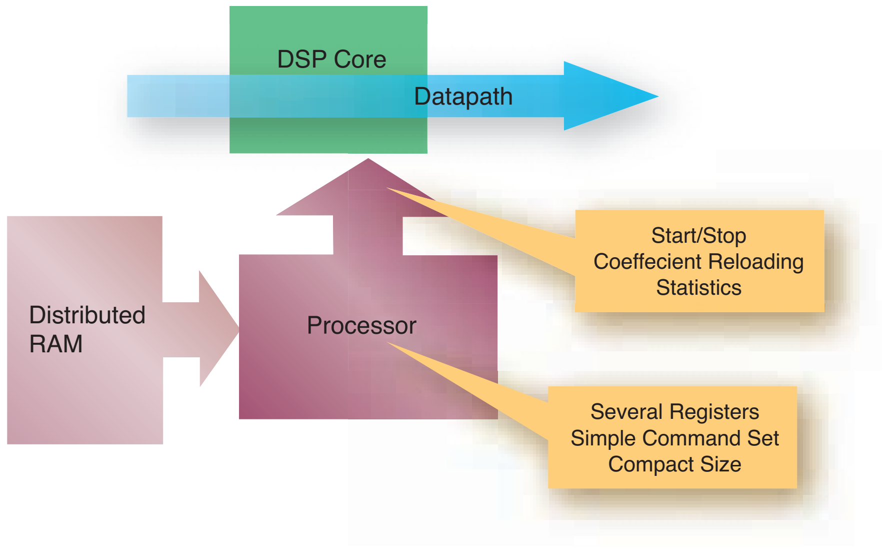


Figure 3 – A DSP coprocessor with general-purpose core

becomes greater than the number of registers. (This is not a problem for less complex algorithms, when calculations are simple enough to impose some limitations on processor architecture.)

For example, you can use a dedicated accumulator to act as a destination register for all ALU activity. This allows command width to shrink because you can exclude information about destination registers (it is always known). Looking further, it is possible to realize a zero-operand processor (with a stack architecture like Forth processors or Java machines) with the smallest possible command width. Code compactness is pretty useful in the FPGA world because of the limited amount of on-chip block RAM.

I/O-Oriented Project

Here's another uncommon approach to I/O module integration. In generic processor cores, it is necessary to assign some addresses to each I/O port and then write low-level drivers that will exchange with these I/O cores through in and out commands.

But if you implement any high-performance-critical I/O cores, you may want to boost and simplify data exchange with them. You can also turn to dedicated buses for modules with dedicated data exchange commands, too. This takes additional resources and adds special commands, of course, so the result will not be universal (indeed, you can't add hundreds of I/O modules this way as you can for an on-chip peripheral bus). However, it may be an "ad hoc" solution.

Advantages of FPGA-Based Processors

The unlimited reprogrammability of FPGAs allows you to realize very uncommon instructions. This is important, because it is virtually impossible to add any non-standard, uncommon instruction to ASIC processors without ensuring its further usage. If you don't meet the market requirements for ASICs, you'll lose time and money, but an experimental instruction set costs almost nothing with FPGAs.

If an experimental architecture with additional commands is not required, you can simply reset the FPGA configuration and download the MicroBlaze processor as a proven, supported solution. Experimental architectures may be very effective for particular purposes, and we can always try them at the beginning of a new project. Uncommon algorithms, parallel calculations, or operations with ultra-wide numbers are quite appropriate reasons to use custom processor architectures.

Xilinx FPGAs have a complex set of features that make all processor implementations effective. For example:

- The balanced set of resources and register-rich architecture allow for implementation of deeply pipelined, register-rich processor cores.
- On-chip, dual-port block RAM stores program and data on the same chip to make FPGAs a system-on-chip. Two independent ports also extend memory bandwidth and provide easy downloading and debugging.
- Distributed RAM in SliceM allows you to create small memory sections, such as register files, buffers, stacks, and FIFOs. All of these are important parts of a processor system and can significantly improve performance and functionality.
- The newest Xilinx Spartan™-3AN FPGAs with internal flash memory allow the implementation of a single-chip system with as much as 11 Mb of flash memory and 72 kb of dual-port SRAM. This is enough to complete a serious embedded system-on-chip.
- High-performance Virtex™-5 FPGAs provide better results for processor soft cores, thanks to their six-input look-up tables (LUTs). In real projects (with ALU based on combinatorial logic), this makes it possible to implement the processor with fewer logic levels than with previous Virtex platform FPGA generations (and other FPGAs with four-input LUTs). In general, when compared to Virtex-4 devices, processor core designs become 30% faster.

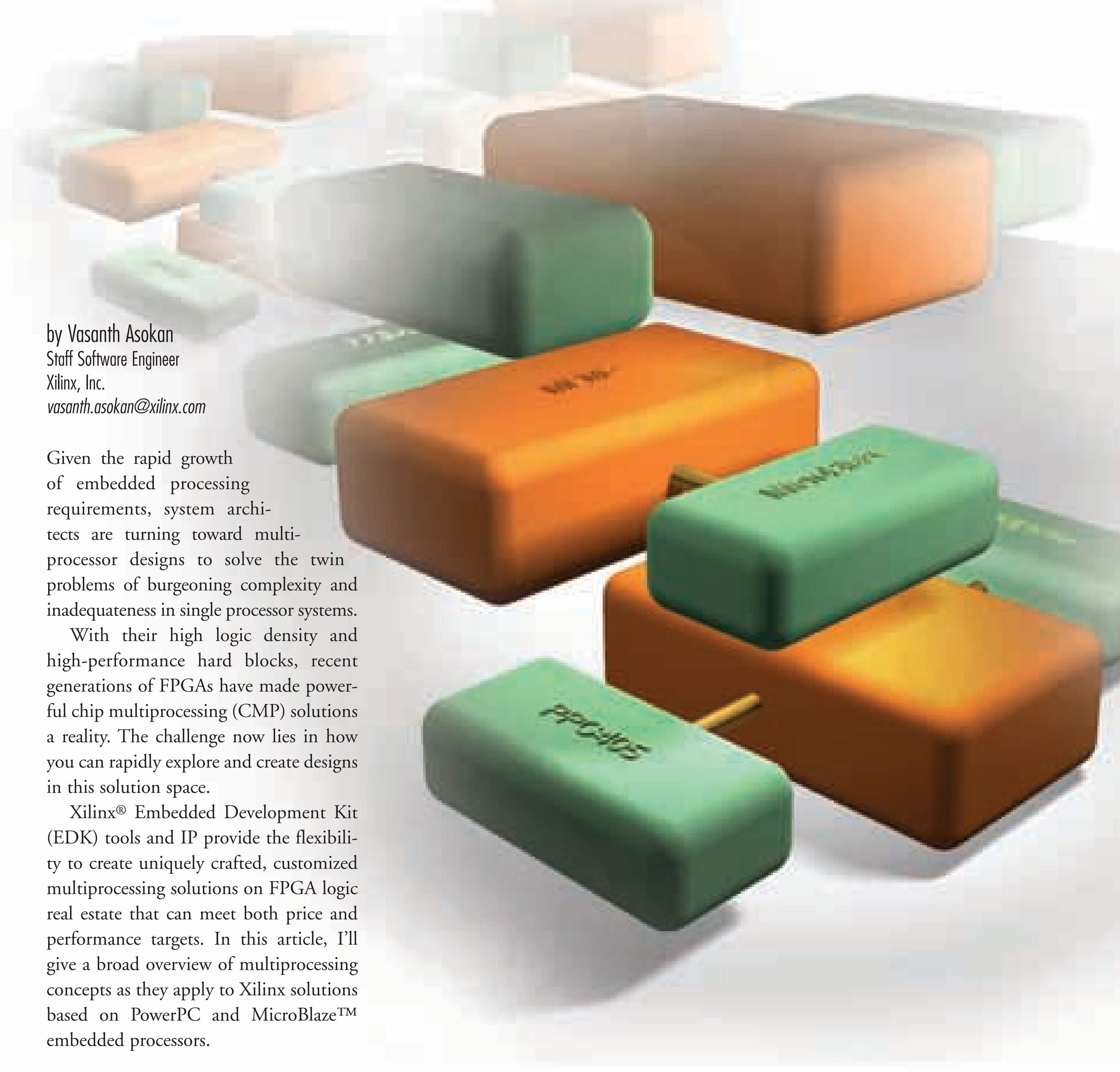
Conclusion

Now that designers have access to real hardware prototyping devices, it is high time to take a fresh look at the embedded processor world. With powerful and easy-to-use development tools like ISE™ software, you can easily try processor designing. This is a promising field for research teams, qualified designers, those who need a specific computing platform, and IT enthusiasts.

CTC Inline Group in Russia, a training center, is now holding training courses on this topic; additionally, you will find a non-commercial community of those creating experimental processor implementations with FPGAs at *www.fforum.winglion.ru* (with text in both Russian and English).

Designing Multiprocessor SOCs

Using Xilinx EDK tools and IP, you can scale your applications by designing SOCs with multiple processors.



by Vasanth Asokan
Staff Software Engineer
Xilinx, Inc.
vasanth.asokan@xilinx.com

Given the rapid growth of embedded processing requirements, system architects are turning toward multiprocessor designs to solve the twin problems of burgeoning complexity and inadequateness in single processor systems.

With their high logic density and high-performance hard blocks, recent generations of FPGAs have made powerful chip multiprocessing (CMP) solutions a reality. The challenge now lies in how you can rapidly explore and create designs in this solution space.

Xilinx® Embedded Development Kit (EDK) tools and IP provide the flexibility to create uniquely crafted, customized multiprocessing solutions on FPGA logic real estate that can meet both price and performance targets. In this article, I'll give a broad overview of multiprocessing concepts as they apply to Xilinx solutions based on PowerPC and MicroBlaze™ embedded processors.

Scenarios

Performance and functional partitioning are compelling reasons to design systems with multiple processors. In general, there are some commonly encountered scenarios where multiprocessing can help:

- Multiple independent functions. The design may have multiple, independent sets of processing tasks. An attractive way to address this challenge is to create independent processing modules dedicated to each processing task, assigning each processing module a unique processor and peripheral set.
- Control or data-plane offload. One common scenario is the presence of a distinct set of real-time (compute- or data-intensive) and non-real-time tasks that might cause a non-response in solutions based on a single processor to be non-responsive. In these cases, you can dedicate a slave processor to perform the real-time task in a timely manner. This leaves the master processor to perform other regular tasks and serve as an interface to the host system. The master processor also monitors and controls the slave processor. The slave processor may contain specialized functions or interfaces, allowing it to meet computation performance requirements. Some examples of this scenario include network offload, media processing, and security algorithms.
- Interface processing. On systems that act as a bridge for or switch between multiple interfaces, you can dedicate a slave processor to the processing of data at each interface, while one or more master processors perform higher level bridging and switching tasks.
- Stream processing. For handling stream-oriented computation, you can arrange processors to act on the data stream in a pipeline fashion. Each peer stage in the multiprocessor pipeline acts on one portion of the computation before passing it to the next processor. This is an effective way to increase system throughput.
- Reliability and redundancy. You can replicate processing systems multiple times to provide reliability and redundancy.
- Symmetric processing. Traditional symmetric processing (SMP) is a useful solution with which you can scale up (by adding more processors) the performance of applications that do not possess clean partition boundaries. An SMP-capable OS layer manages parallel tasks and automatically schedules them across multiple processors. The SMP use model cannot be applied to Xilinx processors, however, because they lack cache coherency, a requirement for implementing SMP.

Apart from the SMP scenario, all other scenarios are feasible on Xilinx FPGAs with EDK tools. The unique capability of Xilinx processing solutions is the flexibility to customize each of the processing subsystems to the application requirements. For example, not all processors may need a cache or a floating-point unit. By assigning specific functions to specific processors, you can create a tailored solution that meets all design goals.

A Simple and Scalable System Architecture

As you can see, a number of use models are possible for multiple processors. There are also a large number of possibilities for the system architecture. Reconciling a use case to a clean, scalable topology and architecture can be daunting, so it helps to define a baseline architecture that fits most needs.

Figure 1 illustrates a dual-core architecture. This architecture presents a simple and scalable multiprocessor system definition. You can generate derivative topolo-

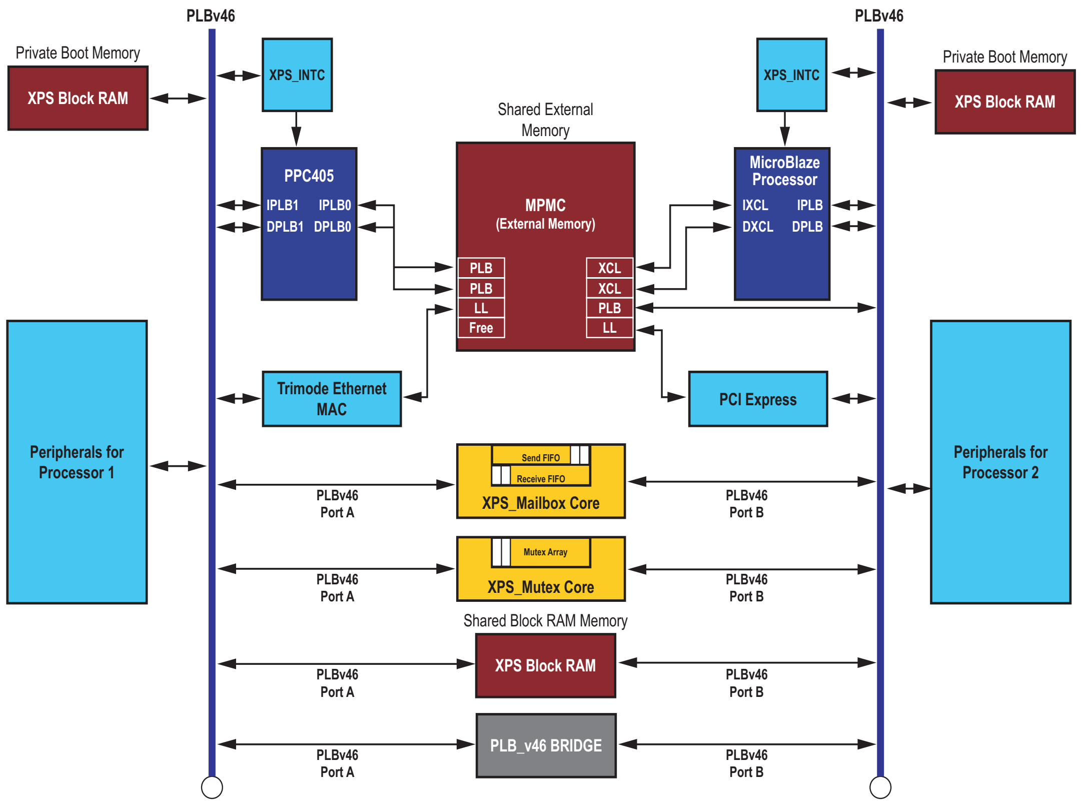


Figure 1 – Dual processor architecture

gies, starting from this definition, to meet design constraints or challenges. The key concepts of this architecture are as follows:

- The architecture is a simple extension of two completely independent single processing systems, formed by linking the systems together with communication components.

- The shared components are all multi- (or dual-) ported in nature. The multiported nature of these components allows each processor's system bus to be independent of the other, both in terms of static as well as dynamic load. By isolating each processing subsystem, you ensure that the system bus is not locked out for a processor or peripheral because of a transaction executed for another processor. All multiported peripherals arbitrate accesses on various ports internally.

- The key shared peripheral is the multiported memory controller (MPMC). MPMC provides access to external memory through different port interfaces. Multiple processors can connect to MPMC through independent ports. This topology allows both the PowerPC and MicroBlaze processors to simultaneously access external memory with minimal latency and high bandwidth. MPMC currently provides a maximum of eight ports, thus allowing as many as three to four processors to connect to a single external memory.

- The architecture also shows sharing of internal block RAM memory between processors. Sharing on-chip block RAM can be an extremely fast way to pass kilobyte-sized data between the processors. Accesses to block RAMs can also be deterministic – an important requirement in some applications.

- Apart from shared memory, there are two other cores – the XPS Mailbox and XPS Mutex – that provide simple forms of interprocessor communication. The XPS Mailbox core provides a low-latency, FIFO-style message-passing interface between the two processors synchronously or asynchronously. It can be used for either directly passing messages or for passing pointers to messages stored in shared memory. You can use the XPS Mutex core for arbitrating accesses to shared resources (whether they are on-chip or off-chip) between software on the two processors. Together, these cores allow you to build cooperating software programs on each processor.

- Some systems might wish to share peripherals that are not multiported (like a UART or SPI or I^{2}C). Such a situation requires providing a system bus bridge from the bus that does not connect to the peripheral to the bus that does connect to the peripheral. Figure 1 shows the use of a bus bridge to share a UART between the two processors.

- Figure 1 intentionally shows a PowerPC 405 processor as the first and a MicroBlaze processor as the second to illustrate certain specific characteristics of each processor. However, any one processor can be equivalently replaced by the other with very minimal adaptation. Thus, this architecture can be seamlessly transitioned between processors.

Although Figure 1 illustrates the recommended overall multiprocessing architecture, various constraints may require you to further refine the architecture. For instance, in a system in which logic area and resource usage are a key concern, all of the processors could be connected to the same system bus. Although this makes the system less deterministic and increases run-time load on the bus, it offers area savings by eliminating a new system bus as well as removing the need for multiple ports on IPs.

Other derivative architectures are also possible, such as having the high-performance processor on a separate system bus and multiple low-performance processors on a shared system bus. You can also create hierarchical topologies by connecting processing subsystems to each other through multiple levels of bridges. The various tools and IP provided in EDK allow you to further refine this base topology down to something that exactly fits your needs.

Other Considerations

You will typically apply a few other considerations to your multiprocessing architecture. For instance, you will need to define memory maps in a non-conflicting manner between the two processors. The automatic address generation tools in EDK simplify this task down to a push of a button.

You will also need to give some thought to your clocking and reset networks. You will have an option to clock all of the processors at the same rate or use different clocking domains for each processor. Similarly, you can define reset domains at various levels such as processor-only reset, processor subsystem reset, and system reset. The processors must also be independently connected to a debugger port, thus allowing separate debugging sessions for each processor.

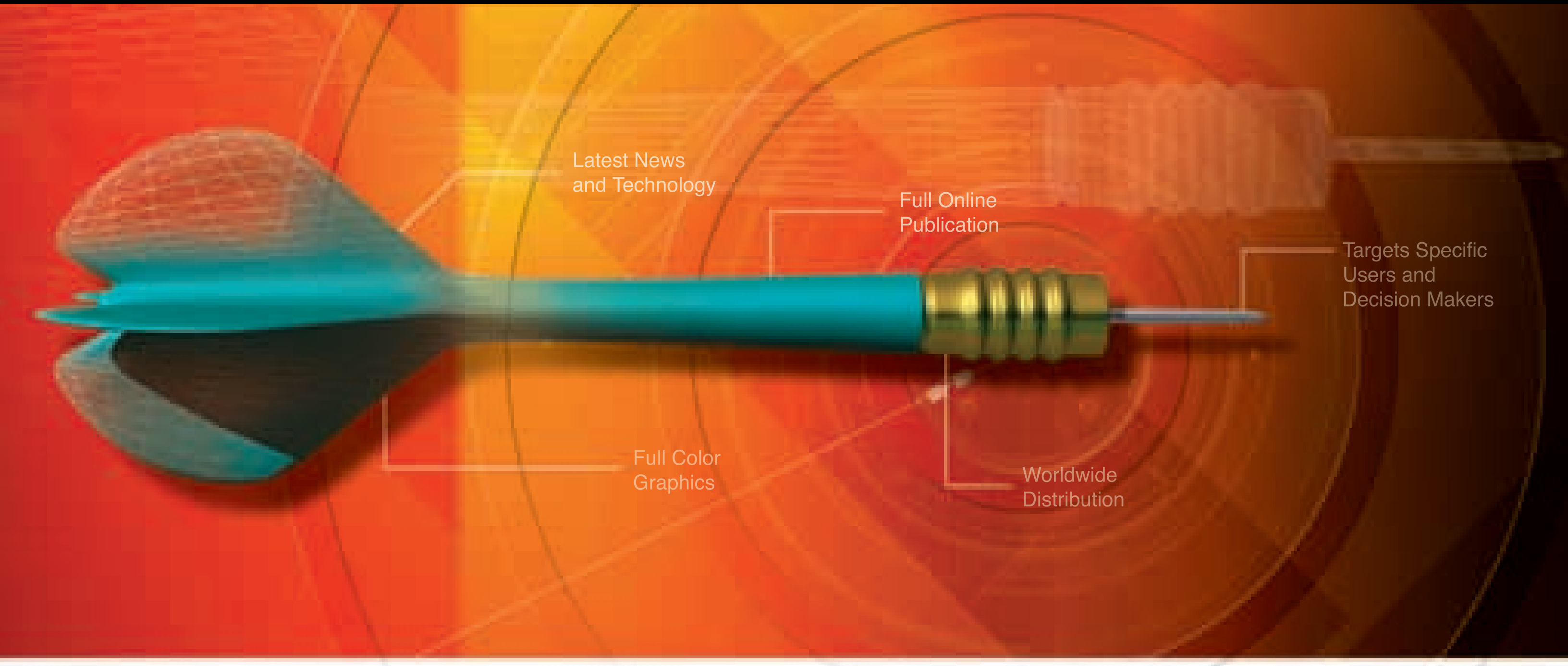
Beyond the hardware considerations, you will also be designing your software systems such that they can work cooperatively. This includes using shared memory, message passing, and common synchronization concepts such as barriers and rendezvous so that the system behaves in a predictable and synchronized fashion. Commercial software stacks are also available that can provide higher level communication paradigms.

Conclusion

For a more details about the possibilities of multiprocessor systems, a longer white paper version of this article is available at: *www.xilinx.com/support/documentation/white_papers/wp262.pdf*. Also consider the reference designs described and provided in Xilinx application note XAPP996, "Dual Processor Reference Design Suite," at *www.xilinx.com/support/documentation/application_notes/xapp996.pdf*.

Stay tuned for new features in future EDK tools, such as support for automated multiprocessor design creation and cooperative debugging. The Xilinx Virtex™-5 FXT platform, with the powerful PowerPC 440 processor, also opens up endless possibilities for creating ultra-high performance multiprocessor systems.

GET ON TARGET


Latest News and Technology
Full Online Publication
Targets Specific Users and Decision Makers
Full Color Graphics
Worldwide Distribution

LET XCELL PUBLICATIONS HELP YOU GET YOUR MESSAGE OUT TO THOUSANDS OF PROGRAMMABLE LOGIC USERS.

Hit your target by advertising your product or service in the Xilinx *Xcell Journal*, you'll reach thousands of qualified engineers, designers, and engineering managers worldwide.

The Xilinx *Xcell Journal* is an award-winning publication, dedicated specifically to helping programmable logic users – and it works.

We offer affordable advertising rates and a variety of advertisement sizes to meet any budget!

Call today :
(800) 493-5551
or e-mail us at
xcelladsales@aol.com

Join the other leaders in our industry and advertise in the *Xcell Journal*!

Xcell PUBLICATIONS

www.xilinx.com/xcell/

XILINX®

Accelerating Video Development on FPGAs Using the Xilinx XtremeDSP Video Starter Kit

Try a video development methodology that is processor-friendly, generates highly optimized results, and does not require VHDL or Verilog knowledge.

by Tom Hill
Sr. Marketing Manager, DSP
Xilinx, Inc.
tom.hill@xilinx.com

Along with next-generation video compression standards, the industry shift from basic video processing to more complex and integrated processing solutions are driving system requirements for video performance beyond what standalone DSPs can deliver. FPGAs such as the Xilinx® Spartan™-3A DSP fill this gap for cost-sensitive military, automotive, medical, consumer, industrial, and security applications by providing more than 20 GMACs of DSP performance for less than $30. FPGAs uniquely provide logic, embedded processing, OS support, and drivers to offer a complete end-to-end solution for video.

It is not that developers lack understanding about the performance benefits of FPGAs that prevent their use in video applications; rather, it's a lack of experience with the design flow. This is especially true for traditional DSP program developers accustomed to programming in C.

You can achieve FPGA performance gains by exploiting the flexibility of the device to configure a hardware architecture optimized for a particular application. This flexibility adds a degree of freedom to the development process that also contributes to its complexity.

The XtremeDSP™ Video Starter Kit (VSK) provides a complete and easy-to-use design environment. Example applications and full support for standard Xilinx tool flows help accelerate the design process yet still allow for end-product differentiation.

Introducing the XtremeDSP VSK – Spartan-3A Edition

The XtremeDSP Video Starter Kit – Spartan-3A Edition is a video development platform comprising the Spartan-3A DSP 3400A development platform, the FMC-video daughtercard, and a VGA camera.

The Spartan-3A DSP 3400A development platform, which you can purchase separately, is built around the Spartan-3A DSP XC3SD3400A device. This device provides 126 embedded DSP blocks for implementing coprocessing and high-performance video processing systems.

The FMC-video daughtercard extends the video I/O capabilities of the Spartan-3A DSP 3400A development platform by including the following additional interfaces:

- DVI-I input, both digital and analog
- Composite input and output
- S-video input and output
- Two camera inputs

Video Development Tools

You can create video applications for the VSK without RTL knowledge or experience using the Xilinx Embedded Development Kit (EDK) and System Generator for DSP. EDK is a comprehensive solution for designing embedded programmable systems and includes the Platform Studio tool suite, embedded IP cores, and the MicroBlaze™ embedded processor.

System Generator for DSP enables the use of The MathWorks Simulink/MATLAB modeling environment for FPGA design by providing a Simulink blockset of more than 100 Xilinx-optimized DSP building blocks.

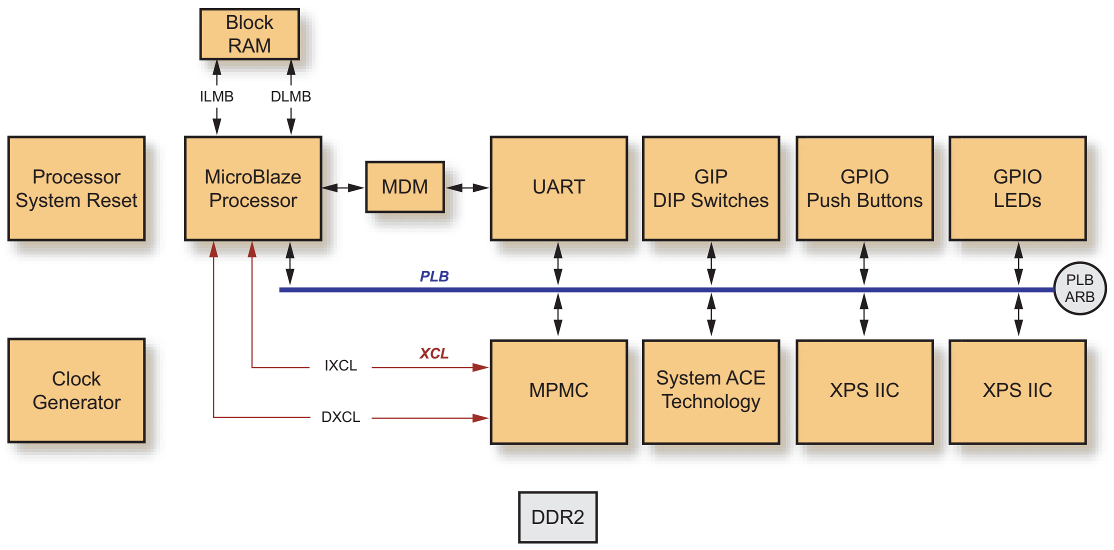


Figure 1 – Base platform block diagram

Developing Video Applications Using the Base Platform

An embedded system called the base platform provides the framework from which you can develop video applications using the VSK. The base platform is an embedded system created using the Xilinx Platform Studio base system builder (BSB) and includes a MicroBlaze embedded processor.

This framework provides a starting point for new designs or serves as an easy migration path for existing applications developed on processor-based systems. You can recompile any C code created for external processors on the MicroBlaze processor with minimal effort; once ported, the high-performance video chains can migrate from software to the FPGA fabric.

To assist in this migration, the VSK includes an IP library of custom peripherals that you can easily add to the base system using Platform Studio. You can connect to the video interfaces, manage data frames, and perform memory access and basic video processing. These custom peripherals include:

- DVI in
- DVI out
- Camera
- Video frame buffer controller (VFBC)
- Video processing pipeline

The Xilinx VFBC is ideal for video applications where the hardware control of two-dimensional data is required to achieve real-time operation. This is typical of motion estimation, video scaling, on-screen displays, and video capture used in video surveillance, video conferencing, and video broadcasts.

Jump-Start Development Using VSK Reference Designs

The VSK provides three reference designs for jump-starting the development of video applications running on Xilinx FPGAs. Each reference design is built on the base platform and uses custom peripherals from the VSK IP library. Table 1 lists each reference design and the video processing and connectivity capabilities that it illustrates. These reference designs are intended to serve as a starting point from which further development may occur. Figure 2 shows how the DVI pass-through reference design interfaces into the base system.

Reference Design	Functionality Description
DVI Pass-Through	• Capturing a video stream from the input port • Performing real-time image processing on the video stream • Displaying the processes video
DVI Frame Buffer	• Capturing a video stream from a DVI source • Buffering the video stream in external memory • Displaying the buffered video • Reporting memory bandwidth utilization data
Camera Frame Buffer	• Capturing a video stream from a camera • Performing processing on the video stream • Buffering the video stream in external memory • Displaying the processed video at a different rate • Using a microprocessor to configure various aspects of the video pipeline

Table 1 – VSK reference design summary

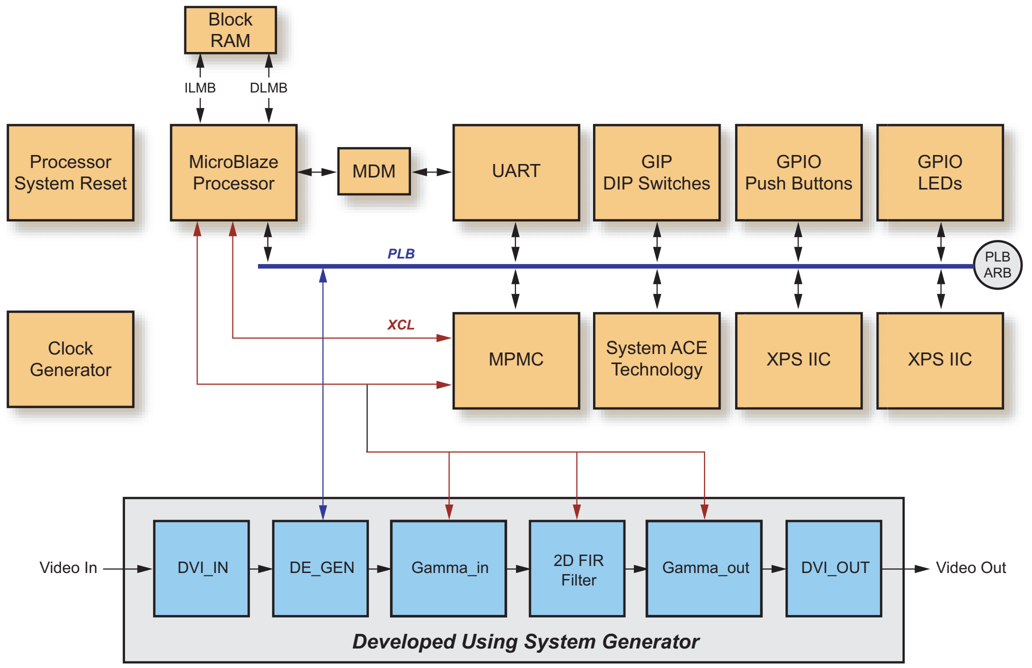


Figure 2 – Base system with video pipeline

Using Model-Based Design to Create Video Applications

Accelerating video applications on FPGAs requires that performance-critical operations migrate from software running on processors to hardware. The VSK supports a variety of hardware design flows. This includes flows that leverage strong hardware design backgrounds using VHDL/Verilog and flows that accommodate little or no hardware design experience by leveraging more abstract modeling environments including C, MATLAB, and Simulink.

Simulink from The MathWorks is a model-based design environment that you can use to develop algorithmic models of video systems. The MathWorks provides an optional video and imaging blockset for Simulink that provides a rich set of video building blocks for easily processing streaming video and visualizing the results at each step in the model.

You can initially model the video processing algorithm itself abstractly using floating-point data types and high-level video and imaging blocks, refining the algorithm as you consider the trade-offs associated with complexity, system cost, and performance.

System Generator for DSP enables the use of Simulink for Xilinx FPGA designs by providing a rich set of DSP building blocks, optimized for Xilinx devices. Tight integration allows DSP designs captured in System Generator to be converted into custom peripherals for Platform Studio and connected to the base system using the processor local bus or Fast Simplex Link bus.

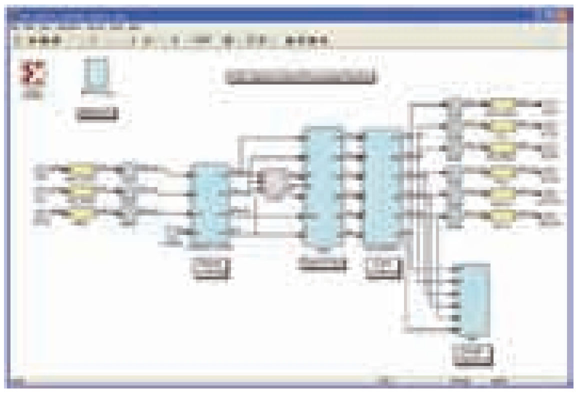


Figure 3 – System Generator diagram for a camera video processing pipeline

Figure 3 shows an example of a camera video processing pipeline created using System Generator, which is included in the camera frame buffer reference designs shipped with the VSK.

System Generator supports hardware-in-the-loop co-simulation using the Spartan-3A DSP 3400A development platform. You can use this platform to accelerate the performance of Simulink simulations up to 1,000x. This acceleration enables video algorithm development and debugging using real-time video streams read into Simulink through The MathWorks data acquisition toolbox.

Conclusion

The XtremeDSP Video Starter Kit – Spartan-3A DSP Edition keeps development costs low by providing a complete video development solution for under $1,600. The DSP and embedded design tools included enable rapid FPGA development of the video system without requiring RTL design experience.

DSP-optimized FPGA platforms such as the Virtex-5 SXT device, Virtex-4 SX device, and Spartan-3A DSP are particularly well suited to meet both the cost and performance requirements of high-performance video and image processing applications in security, broadcast, industrial, consumer, medical, and automotive applications – while insulating products against early obsolescence.

For more information, visit *www.xilinx.com/s3adsp_vsk.*

MicroBlaze v7 Gets an MMU

Memory Manager Brings Full-Fledged Linux to Xilinx Processor Core

By Tom R. Halfhill
Senior Analyst
In-Stat/Microprocessor Report
thalfhill@reedbusiness.com

Xilinx is upgrading its MicroBlaze embedded-processor core again, this time adding an optional memory-management unit (MMU) that allows the 32-bit processor to run sophisticated operating systems supporting virtual memory. Developers can also substitute a simpler memory-protection unit (MPU) or omit supervised memory management altogether.

The first full-fledged operating system announced for the new MicroBlaze v7 is Lynuxworks BlueCat Linux. Until now, MicroBlaze processors were limited to simpler embedded operating systems that don't support virtual memory or memory protection. With its optional MMU or MPU, MicroBlaze v7 is suitable for a wider range of embedded applications requiring greater security and reliability.

MicroBlaze v7 has other improvements as well. New instructions provide faster floating-point performance and better I/O with coprocessors and custom logic. In addition, Xilinx has upgraded the CoreConnect interface to the latest CoreConnect Processor Local Bus (PLB) v4.6 specification, which provides faster links to on-chip peripherals.

All these improvements strengthen MicroBlaze's position against competing processor cores intended for synthesis in FPGAs – primarily Altera's Nios II and ARM's new Cortex-M1. Those rival processors don't have MMUs or MPUs. MicroBlaze v7 is available now as part of a $495 development kit. (An extra $100 buys a development board with a Xilinx Spartan-3E FPGA.) That price includes a MicroBlaze v7 license, and developers owe no royalties when deploying the processor in Xilinx chips.

Robust Memory Management

Xilinx has steadily improved MicroBlaze since introducing the soft processor in 2001. Two years ago, Xilinx began offering an optional FPU. (See *MPR 5/17/05-02*, "MicroBlaze Can Float.") Last year, Xilinx lengthened the instruction pipeline to permit higher clock speeds. (See *MPR 11/13/06-01*, "Xilinx Revs Up MicroBlaze.") And earlier this year, Xilinx released MicroBlaze v6, which added a few other minor enhancements. Now, with MicroBlaze v7, Xilinx is introducing big-league memory management, which significantly expands the range of embedded applications for which MicroBlaze is suited.

With an MMU, MicroBlaze can run full-fledged operating systems based on the Linux 2.6 kernel. Windows CE is another possibility, although Xilinx hasn't announced a deal with Microsoft yet. Processors that support virtual memory can run powerful operating systems, larger application programs, and multiple programs while avoiding memory collisions that would compromise security and reliability. In addition, virtual memory allows a system to work with less physical memory, which can reduce costs and power consumption. As small embedded systems increasingly tackle heavy-duty applications once limited to bigger systems, sophisticated memory management is becoming almost a necessity.

Of course, many embedded systems don't need this level of memory management, so the MicroBlaze MMU is optional. It's just another configuration option available when MicroBlaze developers synthesize the processor using the Xilinx development tools. Another option is to implement an MPU, which provides memory protection without virtual memory and address translation. An MPU is appropriate for embedded systems that must shield a program's memory region against accidental or malicious intrusions by other programs. Yet another option is to implement privileged-mode execution without memory protection or virtual memory. In privileged mode, only the operating system or a privileged application program can execute certain instructions critical to system security.

Table 1 shows how each option (MMU, MPU, or privileged execution) affects the size of the synthesized processor, as measured by the number of additional lookup tables (LUT) required to implement the

Memory Manager Type	LUTs (Virtex-5)	LUTs (Spartan-3)
Memory Management Unit (MMU)	910	1,100
Memory Protection Unit (MPU)	560	670
Privlieged Mode Only	34	38

Table 1 – Sizes of three MicroBlaze v7 memory-management options. Each option occupies additional lookup tables (LUT) in the FPGA, enlarging the processor core. (A fully configured core requires about 3,000 LUTs.) Note that the sizes for these options are slightly different in Xilinx Virtex-5 and Spartan-3 FPGAs, because the LUTs are different: Virtex-5 LUTs have six inputs, whereas Spartan-3 LUTs have four inputs.

feature in the FPGA's programmable-logic fabric. Implementing privileged mode alone requires so few LUTs that it's practically a no-brainer. The other options demand more forethought. In particular, the MMU – which requires about 1,000 LUTs – will account for approximately one-third of a fully configured MicroBlaze v7 core. (To put that size in perspective, the Spartan-3E 1600E chip on a MicroBlaze v7 development board has about 33,000 LUTs and costs $10 to $15, depending on volume.)

The full-blown MMU is the largest memory-management option, partly because it needs a translation lookaside buffer (TLB) to cache a subset of the table that translates virtual and physical memory addresses. MicroBlaze v7 has a 64-entry unified TLB. To supplement this software-managed buffer, there are shadow entries for instruction-memory pages and data-memory pages. The number of shadows is user configurable: one, two, four, or eight entries for instructions and the same for data. (The default configuration is two shadows for instructions and four shadows for data.) The processor automatically manages the shadows, which prevent the TLB from thrashing. Memory pages can range in size from 1KB to 16MB, and mixed sizes are allowed. With 32-bit effective addressing, MicroBlaze v7 can address up to 4GB of flat memory.

The MicroBlaze MMU is patterned after the one in an IBM Power 405 processor. That's no coincidence. Some Virtex family FPGAs integrate a hardened Power 405 core, which is much faster than a MicroBlaze processor synthesized in the fabric. Having a similar MMU brings a few advantages to MicroBlaze v7. First, programmers will have an easier time porting virtual-memory operating systems from the Power Architecture to MicroBlaze. Second, developers should find it easier to create a multicore design that mates one or more MicroBlaze cores with a Power 405 in a shared-memory configuration. And third, the Power-style MMU prepares MicroBlaze v7 for possible future arrangements with IBM to integrate newer Power cores into Xilinx FPGAs.

Faster CoreConnect Bus

CoreConnect is IBM's on-chip bus for SoCs, introduced in 1999. (See *MPR 7/12/99-03*, "PowerPC 405GP Has CoreConnect Bus.") Although IBM created CoreConnect mainly for its own Power Architecture processors, anyone can freely license CoreConnect as synthesizable intellectual property (IP), and it's not specific to a particular CPU architecture. Over the past eight years, soft-IP vendors have made many of their licensable peripheral cores compatible with CoreConnect. The only on-chip bus supported more widely is ARM's AMBA.

For greater efficiency, CoreConnect separates low-speed and high-speed peripherals on separate buses joined by a bridge. Until now, MicroBlaze supported only the slower On-chip Peripheral Bus (OPB), which has a 32-bit-wide datapath. MicroBlaze v7 still supports the OPB for backward compatibility but adds support for the faster Processor Local Bus (PLB). The PLB's datapaths are configurable during synthesis for 32-, 64-, or 128-bit widths. Bus bandwidth depends on the width of these datapaths and the FPGA's clock frequency, which can reach 550MHz in the fastest Virtex-5 devices. At 550MHz, the maximum theoretical bandwidth of a 128-bit PLB would be 8.8MB/s.

The PLB connects directly to the CPU and provides a multidrop bus shared by several on-chip peripherals. It supplements a proprietary Xilinx interface called the Fast Simplex Link (FSL). An FSL is a direct point-to-point interface, not a multidrop bus. FSLs are faster than shared buses but require more logic gates per I/O interface. An SoC design can use one or more FSLs in combination with a CoreConnect PLB for different purposes, giving developers a wealth of options.

Figure 1 shows an example of an SoC implemented in a Xilinx FPGA with a MicroBlaze or Power 405 processor core. In this example – a TCP/IP packet processor – the most critical datapaths link the Ethernet controller to the external-memory controller and the CPU. Those datapaths are FSLs, which can be 32 to 128 bits wide. Less critical components share a CoreConnect PLB. The Xilinx development tools can automatically configure an FSL for a particular purpose or allow developers to configure the interface manually.

With IBM's blessings, Xilinx has slightly modified the standard CoreConnect IP. These modifications were necessary because the programmable-logic gates of FPGAs aren't as efficient as the standard-cell gates of ASICs, especially when routing signals across a large chip. Datapaths and clock trees tend to stretch much longer in FPGAs, making timing closure more difficult. The problem gets worse in complex designs that distribute numerous peripherals throughout the chip on a shared bus. Consequently, Xilinx has modified the PLB to be more synchronous and to eliminate indeterminate data bursts. Xilinx says these changes, though relatively minor, allow developers to attach 10 or 20 peripherals to the CoreConnect PLB without timing problems.

In addition, Xilinx has modified the PLB to work more efficiently with hardened transceivers built into some Virtex-5 FPGAs. These transceivers are a PCI Express endpoint and a trimode Ethernet media-access controller (TEMAC), which deliver much better performance than would equivalent soft-IP controllers synthe-

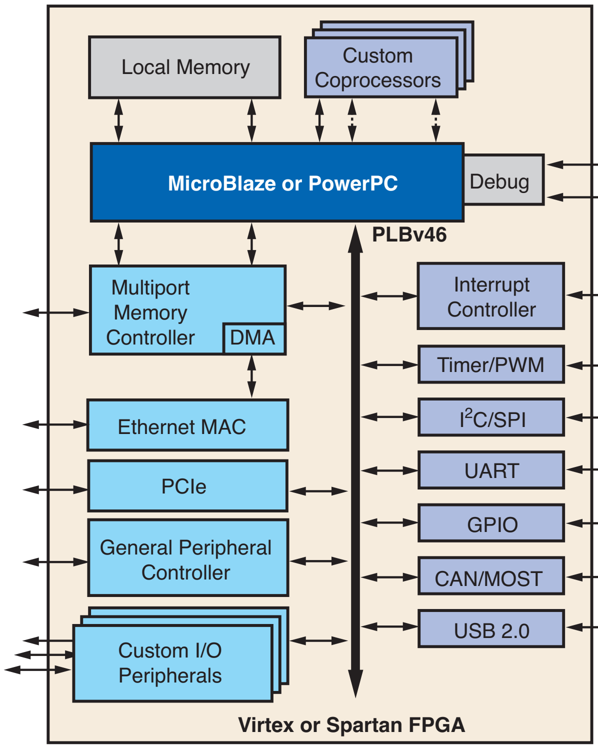


Figure 1 – Example SoC block diagram. This packet processor uses Xilinx Fast Simplex Links (FSL) for critical datapaths and a shared IBM CoreConnect Processor Local Bus (PLB) for other on-chip peripherals. MicroBlaze v7 is the first version of the core to support the CoreConnect PLB; previous MicroBlaze cores supported only the slower CoreConnect On-chip Peripheral Bus (OPB). The multiport memory controller is special Xilinx IP with built-in DMA; it's compatible with DDR1 and DDR2 SDRAM.

sized in the fabric. The PCI Express endpoint is fully buffered and supports 1, 2, 4, or 8 lanes. The TEMAC transceiver supports 10Mb/s, 100Mb/s, and Gigabit Ethernet rates. In a Xilinx benchmark test, a packet processor based on MicroBlaze v7 and the TEMAC achieved 750Mb/s raw throughput – an impressive 75% utilization of the transceiver's maximum theoretical bandwidth.

Although AMBA enjoys wider support than CoreConnect does, the latter standard is better for MicroBlaze. The Power 405 processor built into some Virtex family FPGAs uses CoreConnect, so it's easier for developers to create asymmetric multiprocessors around the Power 405 and MicroBlaze cores. Many peripheral-IP cores from third-party vendors work with either AMBA or CoreConnect, usually by adding a simple gasket adapter.

New Instructions Boost Performance

Xilinx has added 11 new instructions to MicroBlaze v7: three for floating-point operations, and eight for use with FSLs. The new floating-point instructions are straightforward. One instruction, FSQRT, calculates the square root of a 32bit floating-point value in either 27 or 29 clock cycles, depending on whether the MicroBlaze processor is configured with a three- or five-stage pipeline. (The deeper pipeline is faster.) Without using FSQRT, the same operation performed by a function call to a software library would take about 500 cycles.

The other two new floating-point instructions convert integers to floats or vice versa. The FLT instruction converts a 32-bit integer into a 32-bit float in four or six clock cycles, depending on the pipeline depth. Calling the same function in software would take 330 cycles. Conversely, the FINT instruction converts a 32-bit float into a 32-bit integer in five or seven clock cycles, depending on the pipeline depth. A software function call would take 88 cycles.

Eight new instructions improve I/O when a coprocessor is connected to a MicroBlaze core via an FSL. These instructions take the form of PUT and GET operations, and they allow programs to manage the coprocessor I/O as blocking or non-blocking operations. In a blocking relationship, the CPU stops processing other operations until it handles a coprocessor's request for attention. In a non-blocking relationship, the CPU continues processing other operations while the coprocessor's requests enter a FIFO buffer. The CPU isn't blocked unless the buffer fills. Developers can configure the size of the buffer according to the coprocessor's needs.

In addition, MicroBlaze v7 can accommodate twice as many FSL interfaces as before (16 vs. 8), and programs can dynamically assign coprocessors to individual FSL interfaces at run time. Previously, coprocessor assignments to FSLs were hard-coded into the user's software. Any changes required developers to recompile the software. Dynamic assignments allow developers to write software that adapts to changing conditions and workloads. For example, a developer could create precompiled software libraries that select their appropriate coprocessors at run time, according to the custom hardware in the coprocessors and the tasks to be performed. Multimedia-acceleration libraries could run on coprocessors specializing in fast Fourier

Price & Availability

MicroBlaze v7 is available now as part of Xilinx Embedded Development Kit 9.2, for $495. The kit includes MicroBlaze configuration tools, Eclipse-based software-development tools, and other software, plus documentation. For $595, the kit comes with a development board, a Spartan-3E 1600E FPGA, and a JTAG probe. Xilinx doesn't charge royalties for deploying MicroBlaze designs in Xilinx FPGAs.

For more information, see:

www.xilinx.com/microblaze

Feature	MicroBlaze v6 Embedded Dev Kit 9.1	MicroBlaze v7 Embedded Dev Kit 9.2
TCP/IP Stack	LwIP	Treck
On-Chip Interconnect	CoreConnect On-chip Peripheral Bus (OPB)	CoreConnect Processor Local Bus (PLB v4.6)
External Memory Controller	Xilinx MCH_OPB_DDR	Xilinx Multiport Memory Controller
Ethernet MAC	Fast Ethernet (10–100 Mb/s)	Gigabit Ethernet Trimode MAC (10–100–1,000 Mb/s
Xilinx FPGA	Virtex-5 LX	Virtex-5 LXT
Packet Throughput	~70 Mb/s	>250 Mb/s

Table 2 – MicroBlaze v6 versus MicroBlaze v7 performance. This comparison is based on a packet-processor design pitting the new Micro-Blaze core against the older version. The upgraded design is more than three times faster. However, Xilinx also changed other variables, including the TCP/IP stack, memory controller, and Ethernet controller, which clouds the comparison somewhat. In particular, the network interface leaps from a soft-IP Fast Ethernet MAC (100Mb/s) to a hard-wired Gigabit Ethernet Trimode MAC (10–100–1,000Mb/s).

transforms (FFT) or finite impulse-response (FIR) filters.

Table 2 shows the results of porting a packet-processor design from MicroBlaze v6 to v7 – throughput improved more than 3x, from about 70Mb/s to more than 250Mb/s. However, notice that this comparison (conducted by Xilinx) doesn't isolate the effect of each variable changed in the design. In particular, the Ethernet controller in the upgraded design is much faster. Nevertheless, the comparison demonstrates what is possible. Increasing the maximum theoretical bandwidth of a system doesn't guarantee higher throughput, and one feature of MicroBlaze v7 is better CoreConnect support for the TEMAC hard-wired into Virtex-5 LXT chips.

New Kid on the Block: ARM

MicroBlaze v7 is the second new version of the processor that Xilinx has introduced this year, and the third since 2006. Although these steps have been incremental, they add up to a significantly better processor. The quickening pace of improvement may be due to the arrival of fresh competition: ARM's Cortex-M1.

The Cortex-M1 is the first ARM processor core sanctioned for deployment in FPGAs and optimized for their programmable-logic fabrics. Previously, ARM allowed licensees to test their designs in FPGAs but not to deliver finished designs in the chips. ARM's course change was prompted partly by the rising costs of designing and manufacturing ASICs, and partly by the popularity of the Xilinx MicroBlaze and Altera Nios II cores. (Xilinx and Altera have sold tens of thousands of licenses for their processors.) The Cortex-M1 is a major new development that alters the competitive landscape. (See *MPR 3/19/07-01*, “ARM Blesses FPGAs.”)

The first FPGA vendor to announce support for the Cortex-M1 was Actel, a much smaller company than Altera or Xilinx. Actel has a special arrangement with ARM to sell Cortex-M1 FPGAs without requiring customers to purchase an ARM license or pay chip royalties to ARM. This deal dramatically lowers the cost of deploying an ARM-based design. Xilinx hasn't announced a similar arrangement yet, but MPR suspects it's a possibility in the future. Although the CortexM1 and MicroBlaze processors would seem to make rivals of ARM and Xilinx, their relationship remains more cooperative than competitive. ARM understands that MicroBlaze is primarily a loss-leader product that Xilinx created to sell more FPGAs. A MicroBlaze v7 license costs only $495, so the chips – not the licenses – are the real moneymakers. Xilinx is happy to see customers buying its FPGAs to use with the Cortex-M1, too.

Even so, while ARM and Xilinx cordially shake hands, MicroBlaze v7 is slapping the Cortex-M1 upside the head. The brand-new ARM processor suffers in comparison with the Xilinx loss leader. Although MicroBlaze v7 is priced at a pittance, it's embarrassingly rich in features missing from the Cortex-M1, such as an optional FPU, MMU/MPU, 32-bit divider, and instruction/data caches. On top of that, MicroBlaze can reach higher clock frequencies than the Cortex-M1 can. ARM's biggest selling point is that the Cortex-M1 is from ARM. The ARM architecture is almost an industry standard, and it's supported by tons of peripheral IP, development tools, and software.

As Table 3 shows, Altera's Nios II is a closer match for MicroBlaze, even though it hasn't been significantly upgraded since 2004. (See *MPR 6/28/04-02*, “Altera's New CPU for FPGAs.”) The addition of an optional MMU/MPU gives MicroBlaze v7 its first big advantage over Nios II. However, Altera retains one advantage: a user-configurable instruction-set architecture. Nios II developers can create custom instructions to accelerate specific applications, which can dramatically improve performance. To achieve similar results, MicroBlaze developers can implement coprocessors in the programmable-logic fabric. (Coinciding with the online publication of this article on November 13, Altera is announcing an arrangement with Synopsys that will make it easier for developers to move Nios II-based designs from FPGAs to standard-cell ASICs. MPR plans to cover this development in the future.)

Note that the price gaps among these processors are shrinking. Before the Cortex-M1, the difference between licensing a processor core from an FPGA vendor or licensing one from ARM was four orders of magnitude: about $500 for a MicroBlaze or Nios II versus millions of dollars for an ARM. With the Cortex-M1, ARM has

Feature	Xilinx MicroBlaze v7.0	Xilinx MicroBlaze v6.0	Altera Nios II/f	Altera Nios II/s	Altera Nios II/e	ARM Cortex-M1
Architecture	MicroBlaze	MicroBlaze	Nios II	Nios II	Nios II	ARMv6-M
Primary FPGA Targets	Virtex-5 Spartan-3	Virtex-5 Spartan-3	Stratix, Cyclone, HardCopy	Stratix, Cyclone, HardCopy	Stratix, Cyclone, HardCopy	Fusion, ProASIC-3, Stratix, Virtex-4/5 Cyclone, Spartan
Config. ISA	–	–	Yes	Yes	Yes	–
Pipeline Depth	3 or 5 stages	3 or 5 stages	6 stages	5 stages	1 stage*	3 stages
I-Cache	0–64K	0–64K	0–64K	0–64K	–	–
D-Cache	0–64K	0–64K	0–64K	0–64K	–	–
Local Memory	0 or 2 256K each	0 or 2 256K each	0–8 Configurable	0–4 Configurable	–	0 or 2 1K–1,024 K each
32-Bit Multiplier	Optional	Optional	Optional	Optional	–	Yes, 2 options
32-Bit Divider	Optional	Optional	Optional	Optional	–	–
Barrel Shifter	Optional	Optional	Optional	Optional	–	Yes
FPU	Optional 32 bits (New instructions)	Optional 32 bits	Optional 32 bits	Optional 32 bits	Optional 32 bits	–
Branch Predict	–	–	Dynamic	Static	–	1
Priv. Levels	1 or 2	1	2	2	2	AMBA-Lite
Coprocessor Interface	FSL (New instructions)	FSL	Avalon-MM	Avalon-MM	Avalon-MM	AMBA-Lite
On-Chip Interconnect	CoreConnect PLB v4.6	CoreConnect OPB	Avalon-MM	Avalon-MM	Avalon-MM	–
Memory Management	Optional MPU or MMU	–	–	–	–	–
Translation Lookaside Buffer (TLB)	Optional 8-entry I + D 64-entry unified	–	–	–	–	–
Core Freq (Max)	220 MHz †	220 MHz †	205 MHz	165 MHz	200 MHz	Up to 72MHz >170MHz**
Int. Perf (Max)	240 Dmips	240 Dmips	225 Dmips	127 Dmips	31 Dmips	0.8 Dmips/MHz
FP Perf (Max)	50 MFLOPS	50 MFLOPS	n/a	n/a	n/a	n/a
FPGA Logic Cells	980–3,000	960–1,700	1,800	1,150	600	4,300+ LUT3 tiles ~1,900 LUT4 cells
Introduction	Nov 2007	Mar 2007	2004	2004	2004	4Q07
Price	$495	$495	$495	$495	$495	<$100,000 (ARM) Free (Actel)

*Table 3 – Feature comparison of the Xilinx MicroBlaze v7, MicroBlaze v6, Altera Nios II, and ARM Cortex-M1 processor cores. All are 32-bit embedded processors designed and optimized for synthesis in FPGAs. Altera's Nios II processor is available in three basic configurations that developers can customize. Key differences between the MicroBlaze v7 and v6 are highlighted in purple text. With its new optional MMU/MPU, MicroBlaze v7 gains an important advantage over Nios II; the Cortex-M1 fares poorly in this comparison but is redeemed by a more familiar CPU architecture. (FSL is the Xilinx Fast Simplex Link. Avalon-MM is Altera's memory-mapped interface.) *Nios II/e has a six-stage pipeline, but it works like a one-stage pipe. †Maximum clock speed assumes synthesis in the fastest Xilinx Virtex-5 FPGAs. ‡Estimate for synthesis in an Actel ProASIC3 or Fusion FPGA. **Estimate for synthesis in a Xilinx Virtex-5 FPGA. (n/a: not applicable)*

departed from its long-standing practice of not publicly disclosing licensing fees. Although the exact price for a Cortex-M1 license is still under wraps, ARM says it will cut deals for less than $100,000 – a huge price break. And, as described above, Actel sells preconfigured Cortex-M1 FPGAs without requiring an ARM license at all. With Altera and Xilinx practically giving away their FPGA-ready processors, ARM had to modify its own licensing model to make the Cortex-M1 competitive.

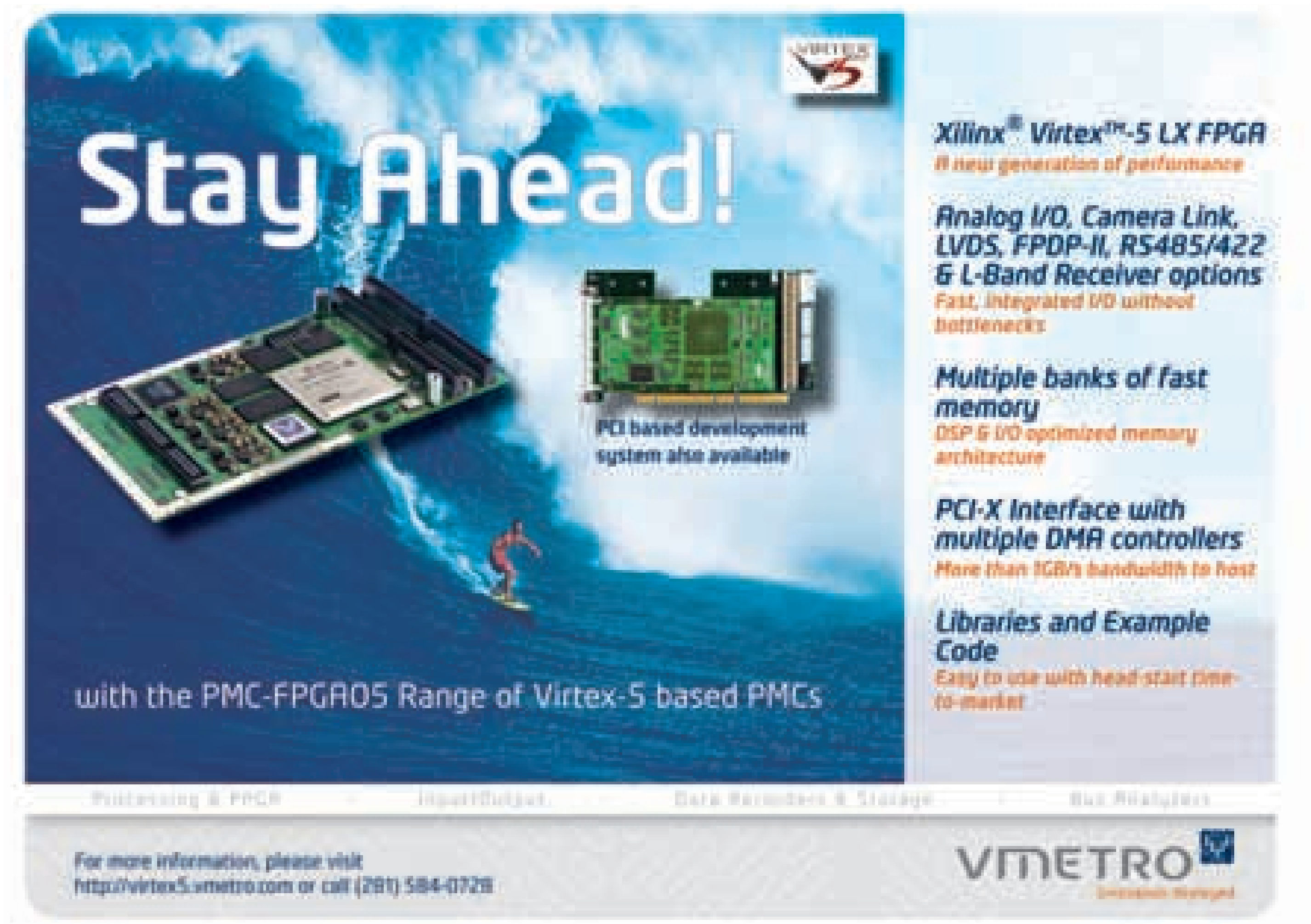
The Future of CPUs for FPGAs

As FPGA prices fall, and the costs of ASICs rise, deploying an SoC in programmable logic becomes increasingly attractive. As MPR has noted in the past, the production volume at which an FPGA implementation becomes more economical than an ASIC implementation keeps tilting in favor of FPGAs, and we perceive nothing on the horizon that will alter that trend. For that reason, the future of MicroBlaze (and Nios II) looks bright.

However, one thing that could change is the CPU architecture that developers choose to implement in an FPGA. For now, MicroBlaze and Nios II are by far the most popular choices, because they are promoted by their respective FPGA vendors and are practically free. ARM's CortexM1 changes the equation by introducing the option of using the industry's most popular 32-bit embedded-processor architecture. In time, other embedded-processor architectures may join the fray. At present, ARC, MIPS Technologies, and Tensilica don't necessarily forbid licensees from deploying designs in FPGAs, but they don't encourage it, either – and they have not optimized their processors for programmable logic.

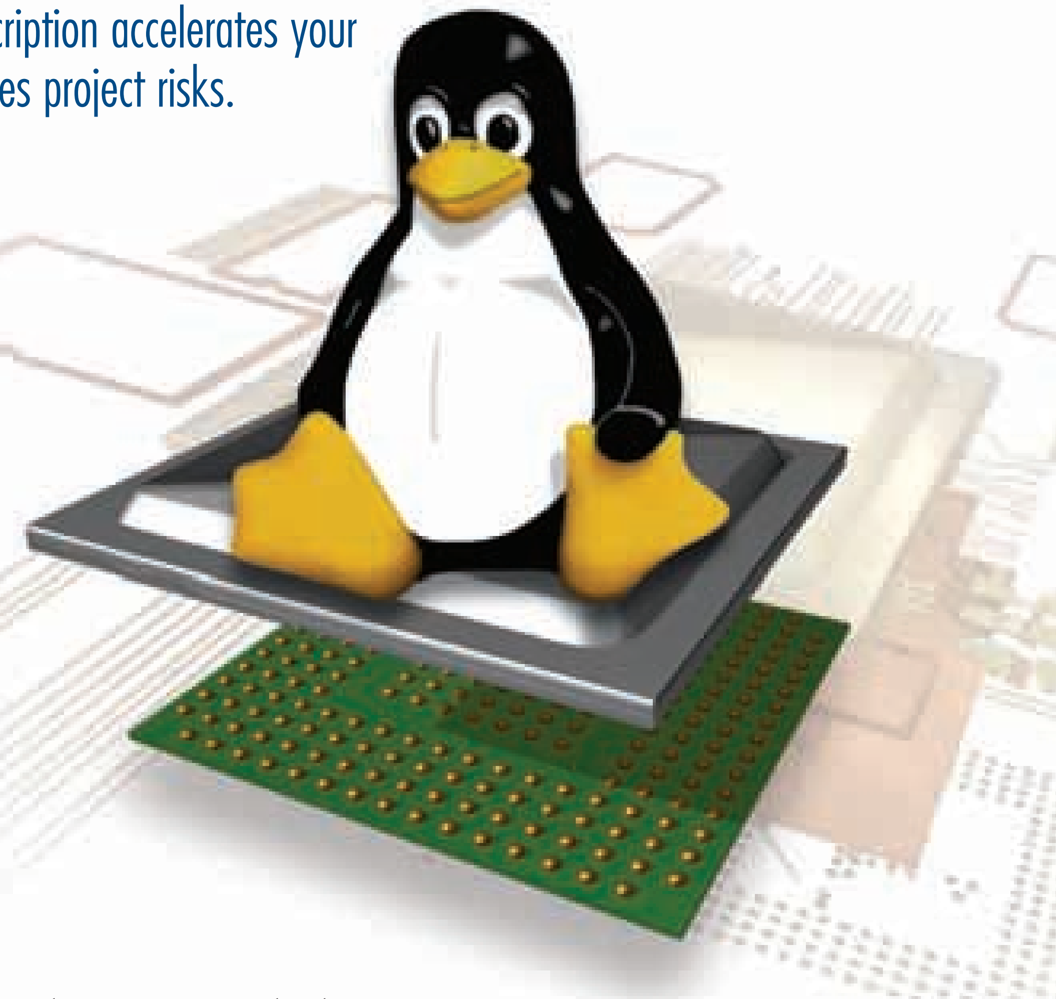
If other CPU architectures do become available for FPGAs, at affordable prices, the time may come when the FPGA vendors' own CPU architectures look less attractive. Although Altera and Xilinx have sold many more CPU licenses than even ARM ever will, a great number of those licenses are purchased by students and tinkerers who never intend to mass-produce their designs. Companies seriously intending to produce FPGA-based SoCs in volume may prefer to use a more widely supported CPU architecture. MicroBlaze and Nios II could become interesting footnotes in the history of microprocessors.

Even if that happens, the Altera and Xilinx processors will have served their purposes. They are selling more FPGAs, they are helping to seed the market for FPGA-based SoCs, and they are defining the features and optimizations that FPGA-specific processors should have. Whether or not MicroBlaze and Nios II live long and prosper, they are wise investments for their vendors.


Stay Ahead!
PCI based development system also available
with the PMC-FPGA05 Range of Virtex-5 based PMCs
Xilinx® Virtex™-5 LX FPGA
A new generation of performance
Analog I/O, Camera Link, LVDS, FPDP-II, RS485/422 & L-Band Receiver options
Fast, integrated I/O without bottlenecks
Multiple banks of fast memory
DSP & I/O optimized memory architecture
PCI-X Interface with multiple DMA controllers
More than 1GB/s bandwidth to host
Libraries and Example Code
Easy to use with head-start time-to-market
For more information, please visit http://virtex5.vmetro.com or call (281) 584-0728
VMETRO

Building Linux Platforms on Xilinx Processors with LinuxLink

Find out how a LinuxLink subscription accelerates your Linux development and mitigates project risks.



by Maciej Halasz
Senior Product Manager
TimeSys
maciej.halasz@timesys.com

Many of you have built a Linux platform on top of Xilinx® processors using PowerPC-based Virtex™ FPGAs. The decision to use Linux is happening more often, and for a variety of reasons. The most appealing aspect of Linux is that it is free to end users and does not require any royalties or run-time license payments to an OS vendor.

There are, however, a number of challenges that drive up the cost and risk of adopting Linux. In this article, I'll outline those challenges and discuss how you can leverage LinuxLink by TimeSys when developing on Xilinx platforms, lowering your cost, risk of failure, and shortening the project timeframe.

The Challenge

Many of you choose to build a Linux platform on your own, spending substantial amounts of time "hunting and gathering" the right Linux components for your project, irrespective of your experience level.

Assuming that you are successful in gathering the right components, the next challenge lies in the aggregation and cross-compilation of all of these pieces. Unfortunately, this aggregation process can be complicated, particularly in an embedded environment where Linux components need to work with an embedded processor. This process is unpredictable and very error-prone, sharply increasing project risks. All of this work to assemble a Linux platform for an application does not add real value to the project. The value comes from applications built on top of the Linux platform.

What is My Task?

There are probably several answers to this question. At a high level, most of you want to build a Linux platform that works on custom hardware, with a value-added application on top (see Figure 1).

The hardware design is central to the project; once such a design is in place, the next step is to get a Linux kernel running on it. Depending on your specific requirements, you must change the Linux kernel to match the hardware design. Sometimes you have to develop new device drivers. This task is supported today to some degree by Xilinx EDK tools.

The next task is to assemble a root filesystem that includes the functionality required by the end application running on your custom hardware design. Assembling a root filesystem can be a very involved and time-consuming process that grows in time and risk level with the number of packages that you need to include.

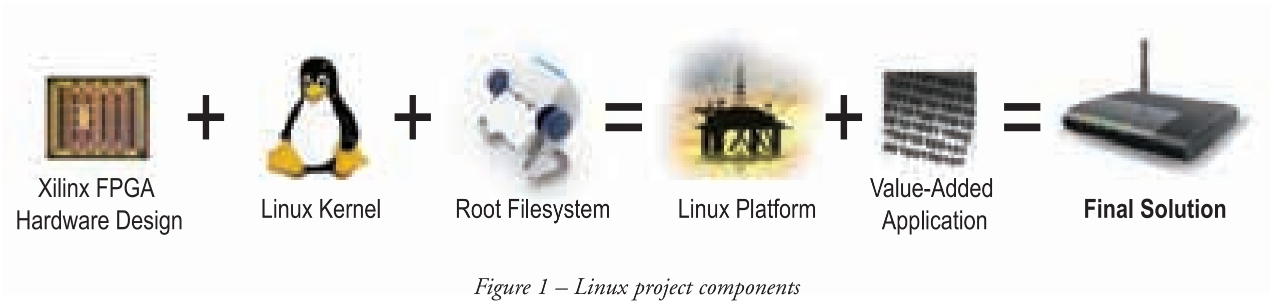


Figure 1 – Linux project components

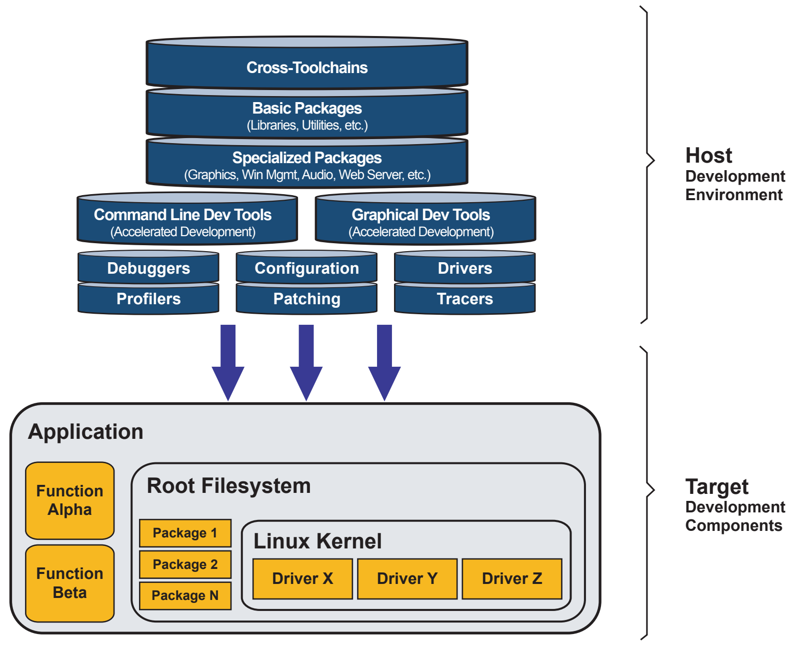

Cross-Toolchains
Basic Packages (Libraries, Utilities, etc.)
Specialized Packages (Graphics, Win Mgmt, Audio, Web Server, etc.)
Command Line Dev Tools (Accelerated Development)
Graphical Dev Tools (Accelerated Development)
Debuggers
Configuration
Drivers
Profilers
Patching
Tracers
Host Development Environment
Application
Function Alpha
Function Beta
Root Filesystem
Package 1
Package 2
Package N
Linux Kernel
Driver X
Driver Y
Driver Z
Target Development Components

Figure 2 – Host and target components needed in a Linux project

Once a Linux kernel and a root filesystem are running on the custom hardware, you can focus on developing value-added applications. If the application requirements are well understood before the project begins, this stage of the project ends here, with aggregation of all Linux pieces into deployable images.

However, application requirements often change during the project cycle. But for Linux, this means not only application changes but also root filesystem changes. Sometimes even the Linux kernel has to be adjusted.

What Do I Need?

You will need several components to develop an embedded software platform (see Figure 2).

For any Linux-based project, Linux components are developed and/or adopted from already existing sources (open-source projects). Depending on the specific application and how complex the hardware design is, the Linux platform development process can take anywhere from weeks to months, even though there are tools and information online that you can leverage in the process. Tools, libraries, and various utilities are definitely needed to develop, debug, and test the Linux solution throughout entire project cycle.

To accelerate development processes while minimizing the risk of failure, you can use highly integrated development tools and prebuilt Linux components from a trusted source.

Development tools that run on the host development platform, combined with your experience and knowledge, will greatly drive how much time and effort you spend on a project.

Linux Development Process

The Linux project development cycle, as shown in Figure 3, can be divided into several stages. Some of the tasks in the cycle (stages 3 and 4) are executed in parallel; others are sequential (stages 5 and 6).

The process of developing Linux-based systems on a Virtex platform starts at the hardware level, when you select a set of hardware blocks (IPs) needed to support your application. Part of the reason why FPGAs are so popular is the flexibility they offer in assembling the hardware platform. That flexibility translates directly into how LinuxLink supports you in your efforts in getting a Linux solution in place – in a timely fashion and with minimal risk.

Linux Kernel Choice

The Linux kernel code is just like other software: it evolves to meet the needs of new designs and applications. Each kernel release, occurring on average three to four times a year, introduces new updates, fixes, and support for ever-growing sets of drivers. For this reason alone, the latest version of the Linux kernel is the preferred choice for many Linux projects.

Root Filesystem Challenges

The number of open-source projects that provide filesystem functionality grows every day, providing you with access to hundreds of Linux components that you can adopt in your project. This is both good and bad for the embedded Linux application. The wide selection of Linux components available from the open-source community provides platform support for all kinds of applications.

On the negative side, that same breadth causes headaches, as it is very difficult to

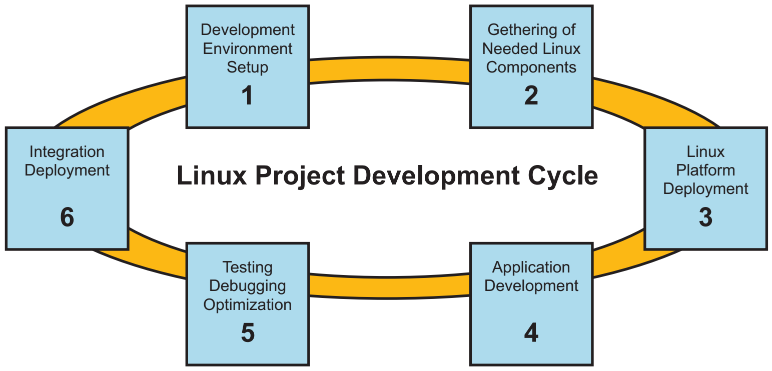

Development Environment Setup 1
Gethering of Needed Linux Components 2
Linux Platform Deployment 3
Application Development 4
Testing Debugging Optimization 5
Integration Deployment 6
Linux Project Development Cycle

Figure 3 – Tasks in the Linux project development cycle

find the packages that work as expected or packages that work well together.

It can take a lot of time and effort to find a good set of packages, libraries, and utilities that properly support application-level development. Certainly there are some starting points available in the open-source community, but they all bear the same challenge – they are not easy to customize for speed, functionality, and the footprint required by the final deployment image.

The process of building a Linux platform based on open-source information and components is not a problem for a very knowledgeable engineer, although it still bears a high risk of failure.

How LinuxLink Accelerates Development

LinuxLink is a subscription-based system that provides embedded engineers with online access to hundreds of Linux components optimized for the Virtex platform, all of which have been tested to work with each other.

The subscription provides you with access to preassembled reference distributions optimized for different applications that include the following:

- EDK studio project files for specific hardware configurations
- Binary Linux kernel
- Cross-toolchain with platform libraries
- Development-ready root filesystem
- Sources

Linux components available through LinuxLink are regularly updated, providing you with the latest code available for both Linux kernel and platform packages.

The reference distribution is optimized to achieve two distinctive goals:

1. Setup the platform and application development environment for the target Virtex platform on your host machine
2. Get the Linux kernel up and running in matter of minutes on a Xilinx reference board (such as the ML405)

After installation of the reference distribution, you can develop, in parallel, both value-added applications and the software platform on which that application will execute (blocks 3 and 4 in Figure 3).

You can easily assemble the Linux platform with pre-built, ready-to-use packages like “alsa” sound libraries or video codecs. You can immediately download these ready-to-use packages to the host development system with the command line or Eclipse-based TimeStorm tools, where they can be aggregated into a complete and functional root filesystem. These filesystems are ready for deployment in a variety of media, including on-board NAND flash or Compact Flash cards.

Application development is well supported by the same command line and graphical development tools used for platform development. The set of architecture-specific cross-toolchains provided with a LinuxLink subscription come preinstalled with the libraries frequently needed for application development. This makes your task of building value-adding applications seamless. Development environment setup by LinuxLink is always kept in sync with target binaries. Libraries that are available on the host for application development are also available in the form of ready-to-deploy, on-target packages for the platform design.

Finally, a LinuxLink subscription provides access to tools that are indispensible in the development of an embedded Linux solution. You can download tools from the LinuxLink site such as debuggers for both system and application-level debugging, profilers for code optimizations, and tracing tools. The subscription comes with ticket-based engineering support staffed with embedded engineers with years of experience.

With a LinuxLink subscription, you get help throughout the entire development cycle (Figure 3). This means that you can get your job done much faster and in a more controlled environment.

Conclusion

LinuxLink is designed to support you as you assemble a custom Linux platform. With prebuilt, ready-to-run Linux components and support, you need not spend valuable time building out a commoditized open-source platform. You can instead focus on the development of value-added applications, along with integration, testing, and optimization. You can then introduce products to the market much faster, at higher quality and lower cost.

LinuxLink supports Virtex-4 and Virtex-5 FPGAs with the latest Linux kernels. With a LinuxLink subscription, you can choose from the latest continuously updated Linux components for your Xilinx FPGA, including the newest Linux kernel and other Linux component versions.

To find out more about how you can use the software, tools, and support provided with a LinuxLink subscription to accelerate your next embedded Linux project, visit *www.timesys.com*. You can browse the content of the LinuxLink repository to see everything available to subscribers.

Xilinx Productivity Advantage.

THE ADVANTAGE IS ALL YOURS!

The Xilinx Productivity Advantage (XPA) program provides all your FPGA design needs up-front where and when you need them. This one-stop, single-vendor solution allows you to create customized bundles of software and services to meet your specific needs. The Xilinx Productivity Advantage program gives you the competitive advantage by:

• Delivering cost savings in reduced speed grades

• Decreasing your time to market

• Decreasing the gap between consumable and consumed technology

Design Faster and More Cost-Effectively

Each XPA solution bundle includes your user licenses for ISE™, the industry's leading design tools and development system for FPGAs. Pre-verified and pre-optimized IP cores and reference designs. Customized development and evaluation boards. And training credits toward the most comprehensive portfolio of expert Education Services. The Xilinx Productivity Advantage program: The Advantage is All Yours!

To learn more about Xilinx Services and the Xilinx Productivity Advantage, we invite you to visit: **www.xilinx.com/xpa**

XILINX®

Quickly Build Distributed Systems

Mobius generates both hardware and software for XPS-based projects.

by Per Ljung
President
Codetronix LLC
ljung@codetronix.com

Many applications use a distributed architecture, including multi-FPGA systems and FPGA-accelerators for microprocessors. Creating master-slave or peer-to-peer distributed systems is complex and requires a design methodology that supports efficient partitioning, as well as an efficient implementation for communication and synchronization.

In this article, I'll present the Mobius tools and work flow for distributed systems. Mobius is a high-level language that can generate both efficient hardware (Verilog, VHDL) and software (C with scheduler) from a single source. Mobius uses handshaking for multi-threaded synchronization and communication, resulting in a target-independent, latency-insensitive handshaking processes. Mobius is suitable for both control- and data-dominated applications. Figure 1 shows how Mobius is a front-end to the Xilinx® Platform Studio tool.

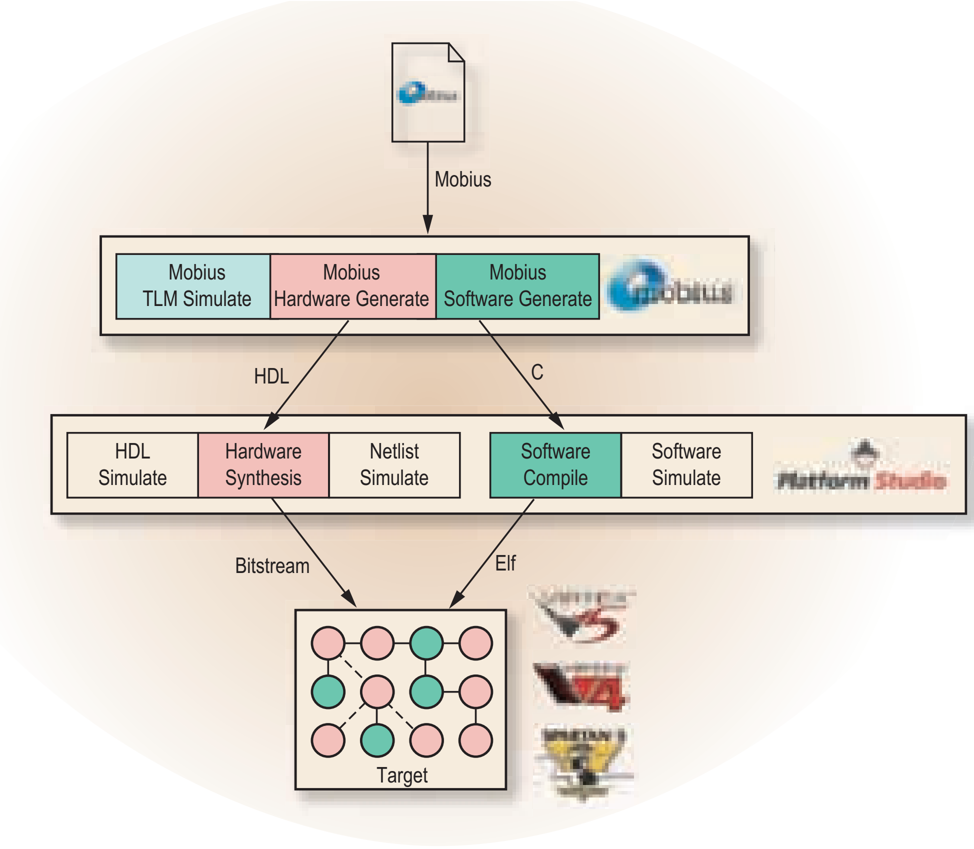


Figure 1 – Mobius is a front-end to Xilinx Platform Studio tools.

The micro-architecture is user-specified using a WYSIWYG syntax; for example, which tasks are run sequentially or in parallel. Parallel tasks synchronize and communicate using message passing over unidirectional point-to-point blocking channels. Because both the reader and writer wait for the other to be ready, the channels provide a robust bridge between hardware and software subsystems. You can compile a Mobius source to hardware, software or any arbitrary combination, allowing for exceptional flexibility when partitioning.

An ESL tool, Mobius generates software for the MicroBlaze embedded processor, circuits for FPGA logic, and bridges between the software and hardware domains. A Xilinx Platform Studio project is automatically generated with Mobius-generated software and hardware, which further increases productivity. Using Mobius, you can design and implement a complex co-design within an hour.

Mobius Language and Tools

Codetronix designed the Mobius language, a tiny high-level multi-threaded language, to be simple to learn and to enable efficient implementations. A uniform programming model allows for target-independent specification and subsequent compilation to target-specific implementations.

The programming model uses the communicating sequential processes (CSP) methodology with blocking message passing. The Mobius language supports sequential or parallel statements, channels, timers, events, and arbitration, in addition to parameterized signed/unsigned integers and fixed- and floating-point (arithmetic, relational, transcendental) operators. As a result, the Mobius language offers many of the built-in capabilities of an RTOS.

The CSP methodology allows compile-time semantic checking of parallel constructs (such as the illegal simultaneous read and write of a variable) for faster development. The Mobius high-speed transaction-level simulator provides quick bit-accurate simulations, enabling functional verification with a test bench written in Mobius. The test bench is also compiled by Mobius, enabling quick verification of the generated hardware or software.

Benchmarks show that quality of results for Mobius-generated software and hardware are excellent. For example, signed fixed-point FIR filters easily achieve 500 MHz on Virtex™-4 and Virtex™-5 device targets, and the commstime benchmark running on a MicroBlaze processor shows that Mobius is 66 times faster than a Posix pthread implementation.

Handshaking Primitives and Channels

The Mobius compiler decomposes the high-level Mobius source into a handshaking graph of handshaking primitives (nodes) connected by handshaking channels (arcs). Some of the handshaking primitives are shown in Figure 2, and can be implemented in either hardware or software as a tiny finite state machine (FSM) reacting to the signals on any connected handshake channels. Because all control and dataflow is local, there are no global FSMs or wires.

Consider the Mobius-generated handshaking graph for the Euclid algorithm to compute the greatest common denominator (GCD) in Figure 3. A control channel is depicted as an arc terminated in a solid

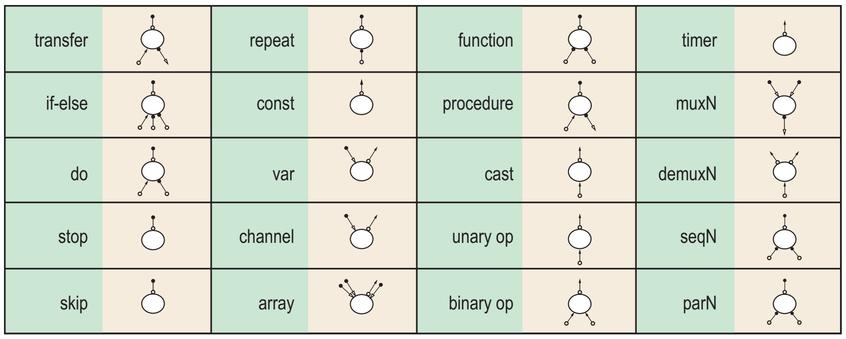


Figure 2 – Target-independent handshaking primitives

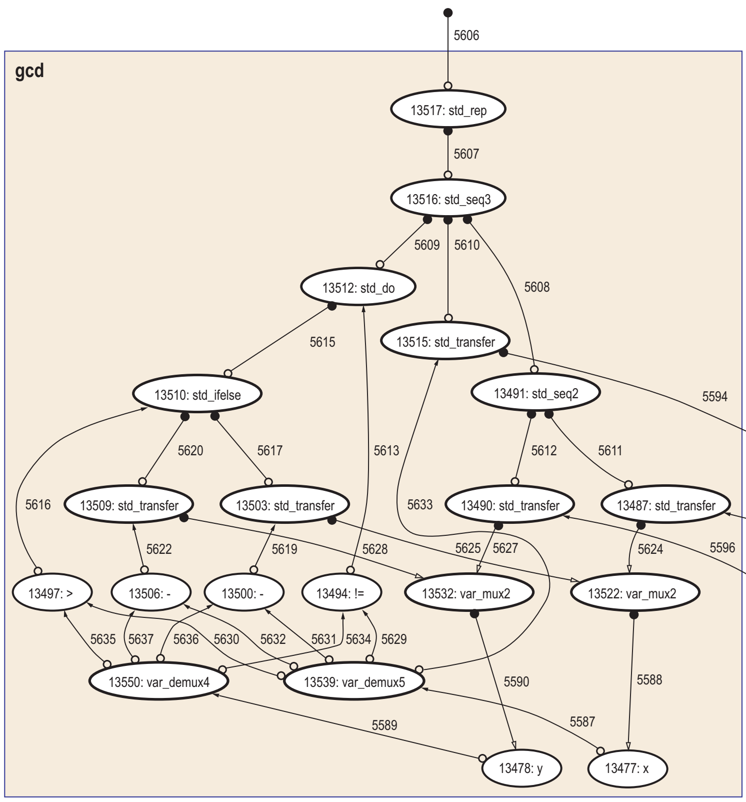


Figure 3 – Handshake graph of GCD()

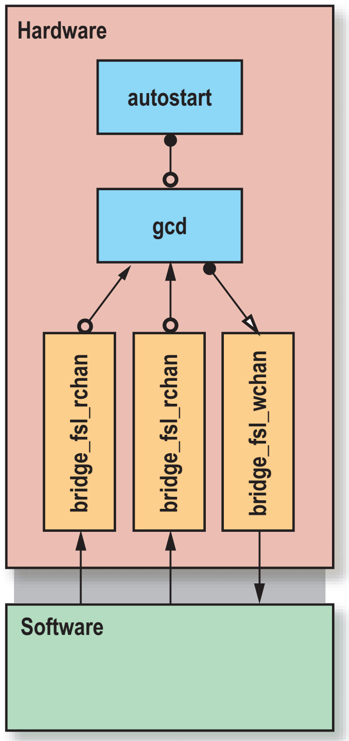


Figure 4 – Mobius-generated hardware and software connected by bridges

dot and a hollow dot. The solid dot denotes an active port that initiates the handshake, and the hollow dot denotes a passive port that waits for the handshake. A push or pull channel has arrow heads signifying the direction of data flow, where the arrow head may be hollow or solid, signifying an active or passive protocol.

Bridges between Hardware and Software

Channels are transport layer-independent and can use wires, SERDES, or packets as appropriate. You can insert bridges on any channel and allow full-featured support of legacy interconnects, including OCP-IP, Wishbone, PLB, FSL, PCI, Ethernet, or RocketIO™ transceivers. Similarly, channels allow domain crossing – between software and hardware, for instance – between different clock domains or between different voltage domains. As a result, implementing distributed and hardware/software systems using handshaking is a straightforward process.

Consider a simple example where the GCD is implemented as a hardware circuit, while the test bench runs in C on a MicroBlaze processor. The Mobius compiler identifies that the reader and writer of each channel are in different domains and emits C for the software domain, HDL for the hardware domain, and FSL bridges to connect the hardware and software.

The Xilinx fast simplex link (FSL) is a peer-to-peer, unidirectional, point-to-point synchronization and communication protocol using queues, which maps easily to Mobius channels. Figure 4 shows how the software writes two values to two FSL bridges and then reads a value from a third FSL bridge that is blocking until the output is available. Simultaneously, the hardware reads two values from its two input channels and computes a result that is written to its single output channel. An autostart() module generates a req pulse to start the Mobius-generated hardware on a system reset.

Automatic XPS Projects

Assembling a complex hardware and software design in Xilinx Platform Studio can be challenging. To enable users to rapidly generate correct XPS projects, Codetronix provides a utility called xpscustom that adds Mobius-generated software applications and hardware IP to an existing Xilinx Platform Studio project.

As shown in Figure 5, xpscustom adds Mobius-generated hardware IP, bridges, and wire connections. On the software side, xpscustom automatically adds simple IP drivers, a software-only application, and a co-design application. This results in a push-button flow where you only have to compile the software and generate the bitstream for a distributed implementation.

Conclusion

Mobius allows you to rapidly develop high-quality hardware and software solutions. You can map Mobius-generated HDL and C to arbitrary voltage, clock, physical, or co-design domains and then connect these domains using synchronous channels, asynchronous channels, or bridges to build distributed embedded systems on any Xilinx FPGA. Mobius also compiles your test bench to hardware and software, allowing functional verification in both software and hardware implementations.

Users benefit from significantly higher design productivity, letting them generate and explore alternative microarchitectures to optimize their system. For more information about using Mobius in your next design, visit *www.codetronix.com.*

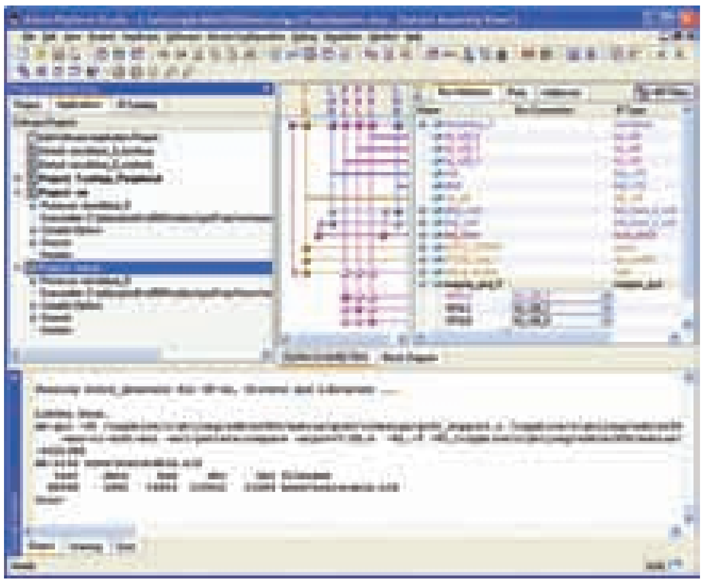
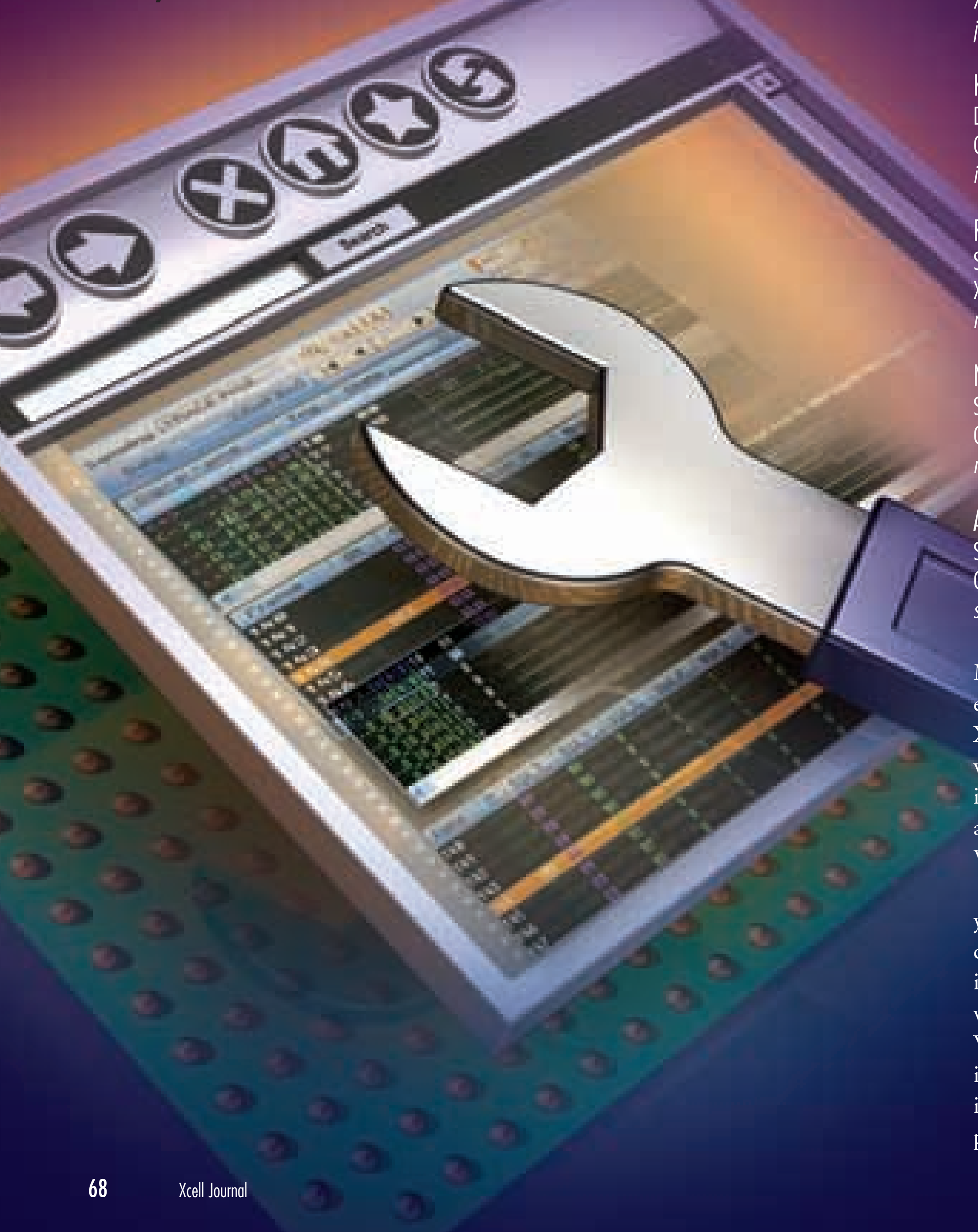


Figure 5 – Hardware and software integrated into a Xilinx Platform Studio project by xpscustom

Debugging Systems with FPGA Embedded Processors

F-Sight improves debugging efficiency through easy-to-use, state-of-the-art features.



by Jorge Carrillo
Staff Software Engineer
Xilinx, Inc.
jorge.carrillo@xilinx.com

Koji Imanishi
Department Manager
Computex Co., Ltd.
ima@computex.co.jp

Raj Nagarajan
Senior Software Engineer
Xilinx, Inc.
raj.nagarajan@xilinx.com

Nobuhiro Nishiguchi
Superintending Engineer
Computex Co., Ltd.
nishi@computex.co.jp

Ayako Suzuki
Senior Engineer
Computex Co., Ltd.
suzuki@computex.co.jp

More and more FPGA designs now include embedded processors (like PowerPC and Xilinx® MicroBlaze™ processors) to deal with control tasks that are easier to describe in a software language like C rather than in a hardware description language like VHDL or Verilog.

When designing embedded systems, you probably spend the vast majority of design time in the debugging phase. It is important to reduce the time associated with finding problems and solving them. When you do face design problems, it is important to use the right tools for the job, including a tool that allows you to identify problems quickly.

Computex F-Sight is an integrated debugger that provides both hardware and software debugging capabilities. On one side, it allows full software debugging of processors inside the FGPA. On the other side, it allows monitoring of FPGA hardware signals. In this article, we will describe how F-Sight can help improve your debugging efficiency.

Enabling the Debugger

Xilinx has enabled the features available in the Computex debugger for use on FPGA internal embedded processors. For MicroBlaze processors, you can use the MicroBlaze debugging module (MDM) to control and debug the processor execution. You can also use the Xilinx MicroBlaze trace core (XMTC) to monitor the processor program execution in a non-intrusive manner.

Because of pin limitations in FGPAs, it is important to reduce the amount of signals that are brought out to pins. The XMTC provides encoded instruction and data traces with pin requirements only 10% of what unencoded signals would require.

To enable the debugger to collect a trace, all you need to do is connect both MDM and XMTC cores to the MicroBlaze processor's debugging and trace interfaces, respectively, and bring out the encoded trace signals to FPGA pins so that F-Sight can collect the data. After FPGA implementation, connect the F-Sight debugger to your board's Mictor connector. If you are using a Xilinx-made ML400 series, ML500 series, or Spartan™-3E/3A/3AN FPGA board that does not include a Mictor connector, you can still employ the processor trace feature in F-Sight by using the corresponding Computex F-Sight adapter. Figure 1 shows F-Sight connected to a Spartan-3 board using the F-Sight adapter.

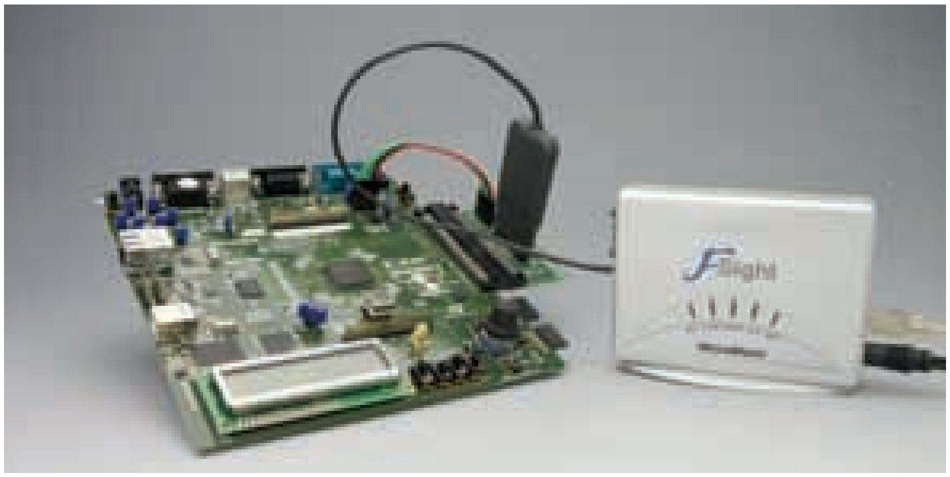


Figure 1 – F-Sight connected to a Spartan-3 board

Using Processor Trace

A processor trace monitors program execution without stopping the processor. This allows you to analyze program flow over long periods to identify problems in the code, all without changing the processor execution state. Computex F-Sight provides processor trace capabilities that can prove very useful in different situations.

Imagine a program that is constantly generating exceptions. Exceptions may happen anywhere in the program; the challenge is to find out why and where the exceptions are being generated. For this, you can set a breakpoint either at the start of the exception being thrown or at the exception vector so that the program stops when it reaches the breakpoint. When the program stops, you can look at the execution history collected by F-Sight. It will show which instructions were executed before going into the exception handler.

Stack overflow is also a common problem in embedded systems. All of a sudden a program starts executing in a location that just does not look right. The stack is probably getting corrupted because of the overflow. If you suspect this problem is occurring, you can set a trigger to start or stop trace data collection. By setting the trigger condition to allow a comparison between the stack pointer and the upper stack limit, the program will break right when the condition occurs. You can then easily identify the stack overflow and the location where it happened.

In certain cases with real-time systems, stopping a processor for debugging may not be an option, as stopping might change the program behavior. Other times the problem might occur very rarely, which might require you to monitor the program execution over a long time. Using F-Sight, you can set complex trigger conditions and collect trace data, which can be post-analyzed to debug the problem.

Probing Internal Signals

When debugging FPGAs, it is common to begin by simulating the design. The simulator is not capable of finding faults with the specifications, although it is able to find errors in the design. Also, it often happens that designs that pass all tests when simulated do not eventually work once implemented on the FPGA. When this happens, you are forced to debug the problem on the actual target system by using a logic analyzer.

The problem arises when you try to bring the signals out of the FPGA so that the logic analyzer can monitor their waveform patterns. As in the majority of cases when designing large-scale embedded systems, it will take a considerable amount of time to go through the FPGA synthesis and implementation tools even for minor changes (such as the changes needed to route signals out to pins). Plus, you risk running into timing problems caused by the different placement and routing. The actual time to run the implementation tools depends on the scale of circuitry and the host computer performance, but it's possible that you will only be able to debug a few times in a day.

Fortunately, Computex F-sight has a very useful feature that allows you to modify the design to bring internal FPGA signals out to pins without going through the synthesis and implementation tools. The

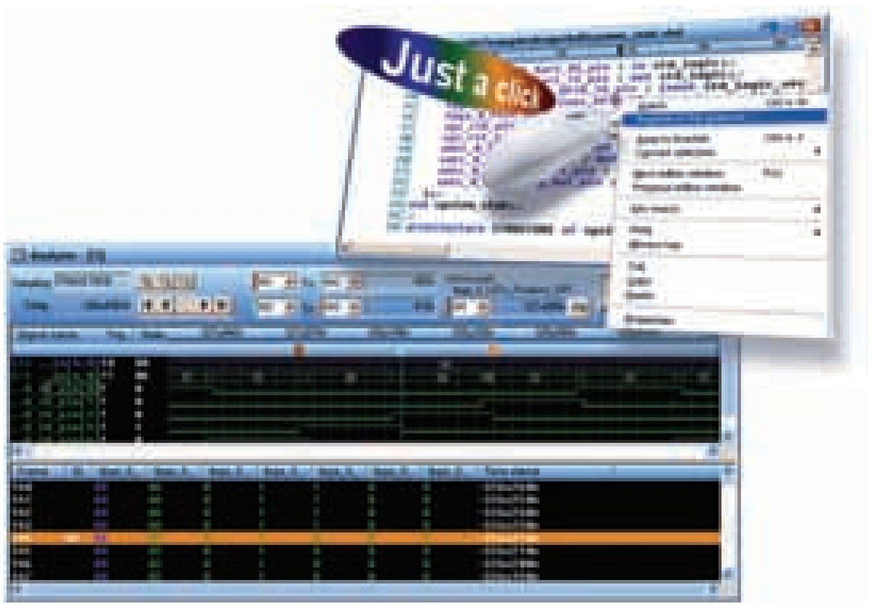


Figure 2 – F-Sight probing

feature is called "probing." Simply select the internal FPGA signals in the display showing HDL source (Figure 2). F-Sight will automatically take care of the rest, adding the appropriate routing to the test pins on your behalf. It does this by utilizing the FPGA Editor included in Xilinx ISE™ software tools. With this feature, the time required for logic synthesis and place and route – time that you could have used for debugging – is minimized, allowing you to spend more time monitoring waveform patterns.

Cooperative Debugging

When the system is not functioning properly, the only thing you can do is investigate the cause of the problem based on actually occurring events. In some cases, event tracking is easier if you do it from the hardware; in other cases, it is easier if you do it from the software. For example, in the case of hardware, if the signal indicating abnormality is asserted you can set the signal as a trigger. In the case of software, if the exception handler is called out, you can set a breakpoint to the exception handling routine and run the user program. The process of event occurrence will be captured into the tracing buffer in F-Sight.

However, the issue here is that even if the history was captured, indentifying causes will take time unless the correlation between the hardware and software is known. For this reason, Computex has implemented a cooperative debugging feature that allows you to synchronize the hardware (analyzer) and software (trace) histories in F-Sight. With this feature, you can check the wave pattern and program behavior at the time the event occurred on the same time axis. The program execution history and the source code view will follow as you scroll through the wave pattern display in the analyzer window (Figure 3). The cooperative debugging feature is powerful in that it quickly identifies causes by allowing cooperative debugging from both the hardware and software sides.

Debugging Flash Memory

The FPGA's internal memory is commonly used to store the embedded processor program. However, this memory's capacity is often not sufficient if the size of the program is too big. One alternative is to use external flash memory and store the user program in it.

Although some debuggers do not have the capability to write to flash memory, F-Sight allows you to fully debug programs that are located in flash memory just as you debug programs located in internal memory. For example, you can download the user program, apply patches to a part of the memory, or even set software breakpoints in the flash memory.

In F-Sight, you can select from among more than 1,000 available types of flash memory options. Even if the flash memory you are using is not available from the options in the list, you can manually and easily add the entry using the graphical user interface.

Conclusion

Debugging systems with FPGA embedded processors should not be a time-consuming task. When you encounter problems, you need to have the right tools that will allow you to efficiently deal with the problem, saving you time to focus on the development part. F-Sight certainly proves to be a tool that will improve your debugging efficiency.

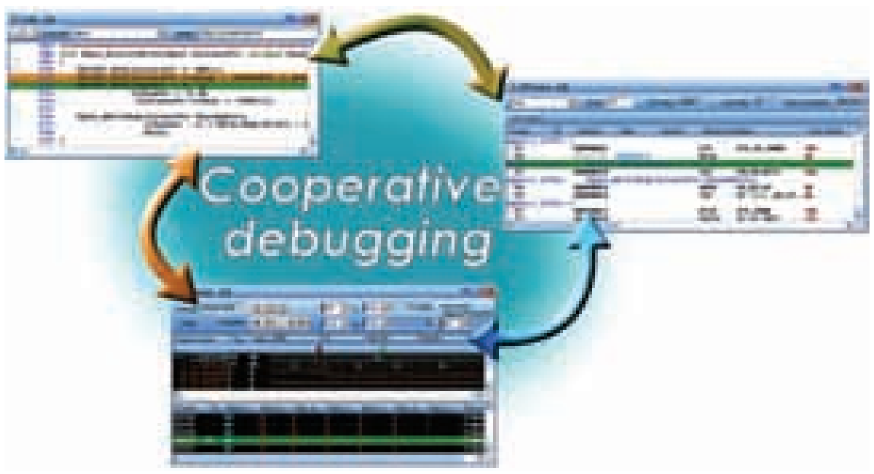


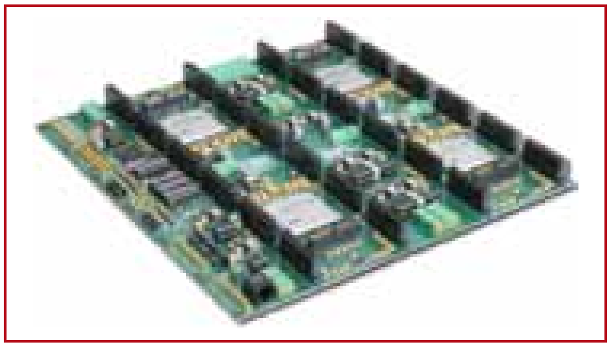
Figure 3 – F-Sight cooperative debugging

ACCELERATE VERIFICATION

With The Confirma™ ASIC/ASSP Verification Platform From Synplicity®

Reduce verification time from months to days

The Confirma platform is a tightly-integrated, easy to use, and comprehensive at-speed ASIC/ASSP verification solution that dramatically accelerates functional verification of ASICs, ASSPs, and SoC designs.



HAPS-54 Virtex-5 Board Offers Prototypers 8 Million ASIC Gates

The three major components of the Confirma Platform are already considered best-in-class tools: the Certify® ASIC verification software implements an existing ASIC design in multiple FPGAs; the HAPS™ High-performance ASIC Prototyping System™ is the execution engine; and the Identify® Pro software, with its revolutionary TotalRecall™ technology, provides full visibility and a seamless interface into industry-standard software simulators for easy debugging. When used together, the tools provide the highest-performance ASIC and ASSP verification platform available today.

Visit http://www.synplicity.com/confirma/xcell to learn more about reducing verification time.

Synplicity®

Simply Better Results

Synplicity
600 West California Avenue
Sunnyvale, CA 94086
Phone: (408) 215-6000
Email: info@synplicity.com

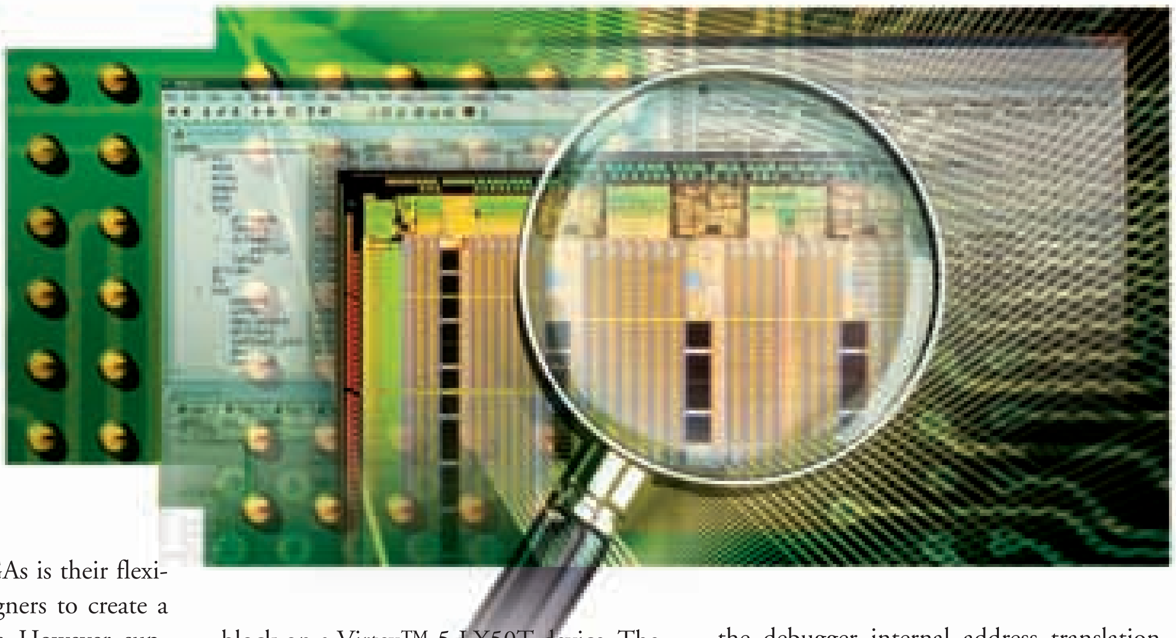
A Complete Debugging Solution for Xilinx Embedded Processors

Employ the industry-proven TRACE32 debugger to cope with the challenges of debugging advanced embedded processor systems.

by Jorge Carrillo
Staff Software Engineer
Xilinx, Inc.
jorge.carrillo@xilinx.com

Raj Nagarajan
Senior Software Engineer
Xilinx, Inc.
raj.nagarajan@xilinx.com

Oliver Oppitz
System Design Engineer
Lauterbach
oliver.oppitz@lauterbach.com



The best thing about FPGAs is their flexibility, which inspires designers to create a myriad of different designs. However, support for debugging the design is often considered last – if at all – so the debugger often has to adapt to the system.

The good news is that a debugger exists from a company that has been around for nearly 30 years in the embedded arena, gaining experience with all of the problems you can imagine and some that you do not even want to hear about. In this article, we'll show you a few examples of big and small features that Lauterbach's TRACE32 debugger offers – features that might save your day or even your project.

Flexibility in Debugging for a Flexible Platform

Speaking of flexible designs, a really interesting customer system comes to mind. It used two Xilinx® MicroBlaze™ processor cores with an internal block RAM memory block on a Virtex™-5 LX50T device. The peculiar thing about this design was that each MicroBlaze processor had only an I-side interface to the block RAM block for instruction fetch. The second memory port of the block RAM was connected to a PCI Express interface, used to remotely change the application code and boot the processor at run time. The challenge was how to debug in a memory area that the debugger could neither read nor write, because the MicroBlaze processor itself could not perform load/store operations in this area.

We solved the problem using TRACE32's internal "virtual memory," a simulated memory inside the debugger that has an indefinite address space (64 bits) and for which memory is allocated on demand. We loaded the target program into this virtual memory and configured the debugger internal address translation mechanism to remap access from unreadable target memory to virtual memory. This made it possible to diagnose the program even at the assembly level.

But there was another challenge: for stepping through the program, especially over high-level language lines and conditional branches, one normally uses software breakpoints. With no access to the block RAM from the MicroBlaze processor, this was out of the question. The solution came in the form of a "map.break" command, used to force the debugger to exclusively use hardware breakpoints for the given address range. Other flavors of the map command allow you to specify the data width of memories, change the endianness, or completely deny the debugger from accessing an address range with critical peripheral registers.

Small but Useful Features

Lauterbach's experience with JTAG debuggers and emulators shows in both the features of the TRACE32 debugger as well as in many small and unexpected details. Given that Lauterbach develops all software with its own tools, this is not surprising. And daily problems often inspire the most useful features.

Did you ever want to disable the annoying timer interrupt handler during a debugging session or invert a conditional branch without restarting? Ever wanted to patch a loop to see if the core executes correctly from external memory? The built-in assembler does just that.

Another typical scenario during debugging: How often did you just step one line too far and had to start all over? The register-restore feature will undo the last steps.

TRACE32 displays memory by permanently rereading the memory 10 times per second, even while the processor is stopped. Why does it do this? Perhaps a second MicroBlaze processor in your system, which you kept running, is causing some data corruption and you would like to also monitor this. Perhaps your sometimes unstable memory now causes telltale flickering on the screen. Or perhaps your JTAG interface is really not so stable at 20 MHz. These are things you would certainly like to know. On the other hand, TRACE32 guarantees that it will only access memories when required.

Having touched on the topic of peripherals, we should also mention the peripheral register files. These specify the location, width, and even bit-wise encoding of memory-mapped registers, grouping them into a tree of related registers. In this way the peripheral registers are easily accessible for inspection and modification: you can disable your DMA controller with a click instead of digging in the target manuals for the correct bit. The debugger comes with specifications for standard peripherals, but you can modify the files for your purposes using a simple text editor. For the Xilinx tool chain, an add-on is in the works to generate these files on the fly as part of the Xilinx build process.

Flexible IDE

TRACE32 offers a powerful graphical user interface (GUI), but its command-line origins still show in two highly useful features: the debugger command line at the bottom of the screen and the fact that practically every function of the GUI is also accessible through commands – and thus from scripts. This automates all of your routine tasks, including target configuration, laying out windows, and arranging them on multiple virtual screens. Best of all, the windows do not have a docking behavior like many IDEs but can be freely positioned and resized, even overlapping each other. There are also couplings with various IDEs, so you can invoke TRACE32 directly from your Eclipse environment, for example.

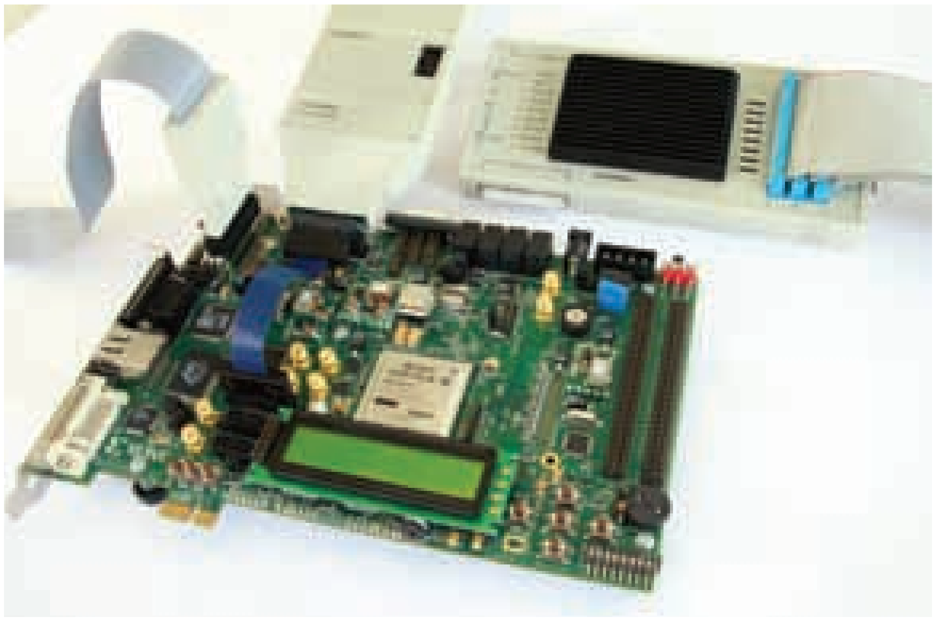


Figure 1 – Lauterbach TRACE32 debugging and trace cable connected to Xilinx ML507 board

Attaching to Multi-Core Targets

Another interesting feature is Lauterbach's intuitive approach to debugging multiple cores in the target. Just start an instance of the GUI for each core and have them share a debugging cable; this also works for heterogeneous systems with PowerPC and MicroBlaze cores or for another system from the 50-plus processor architectures supported by TRACE32 (Figure 1).

TRACE32 attaches to the same JTAG connector used by the Xilinx platform cable and will work with any design created with the Xilinx Embedded Development Kit (EDK). For PowerPCs, dedicated debugging connectors are also supported.

In the context of multi-core systems, the synchronized starting and stopping of cores becomes an issue. For targets supporting this in hardware, like in multi-MicroBlaze processor configurations, the debugger uses hardware features to achieve cycle-accurate synchronization; otherwise the synchronization is done in software. The integrated scripting language is multicore-aware, allowing control of all GUIs from a master script for connecting debuggers to their respective cores, resetting them, and downloading and starting application programs.

Real-Time Program Flow and Data Trace

The main feature of the real-time trace is recording the program flow, defined as each and every instruction the processor executed as well as data transactions. For MicroBlaze processors, this is done through the Xilinx MicroBlaze trace core (XMTC) that comes with Xilinx Platform Studio. XMTC includes a trace encoder, which contains an input interface that attaches to a MicroBlaze processor trace port comprising approximately 200 unencoded signals.

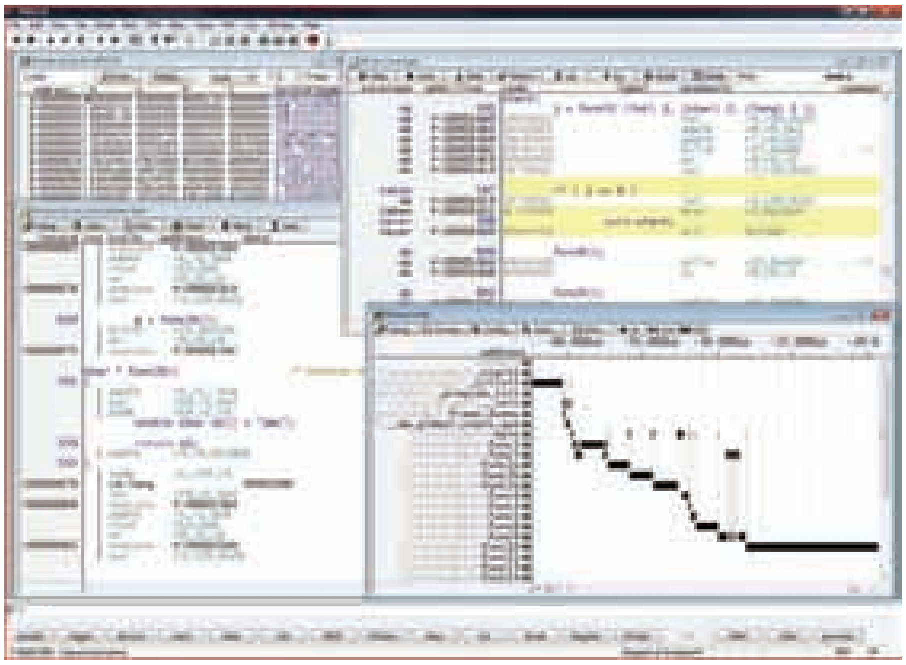


Figure 2 – Lauterbach TRACE32 IDE showing trace, code coverage, and function call chart views

Figure 3A – Lauterbach Mictor MicroBlaze trace adapter for Xilinx Spartan-3E FPGA board

Figure 3B – Lauterbach Mictor MicroBlaze trace adapter for Xilinx MLx boards

It also includes an output interface, which provides an encoded trace in 21 signals.

The trace hardware provides up to 512 MB of external high-speed trace memory, which is used instead of the scarce on-chip memory resources for storing trace information. Trace functionality is also supported on PowerPC architectures. More advanced features include statistical analysis of function and task run times, variable access, and code coverage analysis (Figure 2, Figure 3A and 3B).

Operating System Support

On the MicroBlaze processor, TRACE32 comes with a so-called kernel awareness module for µClinux and Linux. For PowerPCs, many more operating systems are supported, including QNX, VxWorks, and Nucleus PLUS. The extensions make the debugger aware of the kernel-related data structures in the target. They allow debugging on the process level using process-specific breakpoints and program control. Other features include full MMU support; real-time, non-intrusive display of Linux system resources like loaded kernel modules or mounted file systems; statistical evaluation and graphical display of task run times; and task-related evaluation of function run times.

Conclusion

The Lauterbach TRACE32 provides a comprehensive debugging solution for PowerPC and MicroBlaze processors on all Xilinx device families. Future Xilinx-related additions will be to enhance the debugging cable so that the Xilinx ChipScope™ analyzer can use it for target accesses jointly with the debugger and downloading FPGA bitstreams through the debugger itself.

For more information, please visit *www.lauterbach.com.*

Xilinx Global Trade Show Calendar

Xilinx participates in numerous trade shows and events throughout the year. This is a perfect opportunity to meet our silicon and software experts, ask questions, see demonstrations of new products, and hear other customer success stories.

For more information and the most current schedule, visit ***www.xilinx.com/events/***.

April 14-17, 2008
NAB 2008
Las Vegas, NV

April 15-17, 2008
Embedded Systems Conference
Silicon Valley
San Jose, CA

April 15-17, 2008
Special Event Effects
Symposium
Long Beach, CA

April 21-22, 2008
China International Medical
Electronics Technology
Shenzhen, China

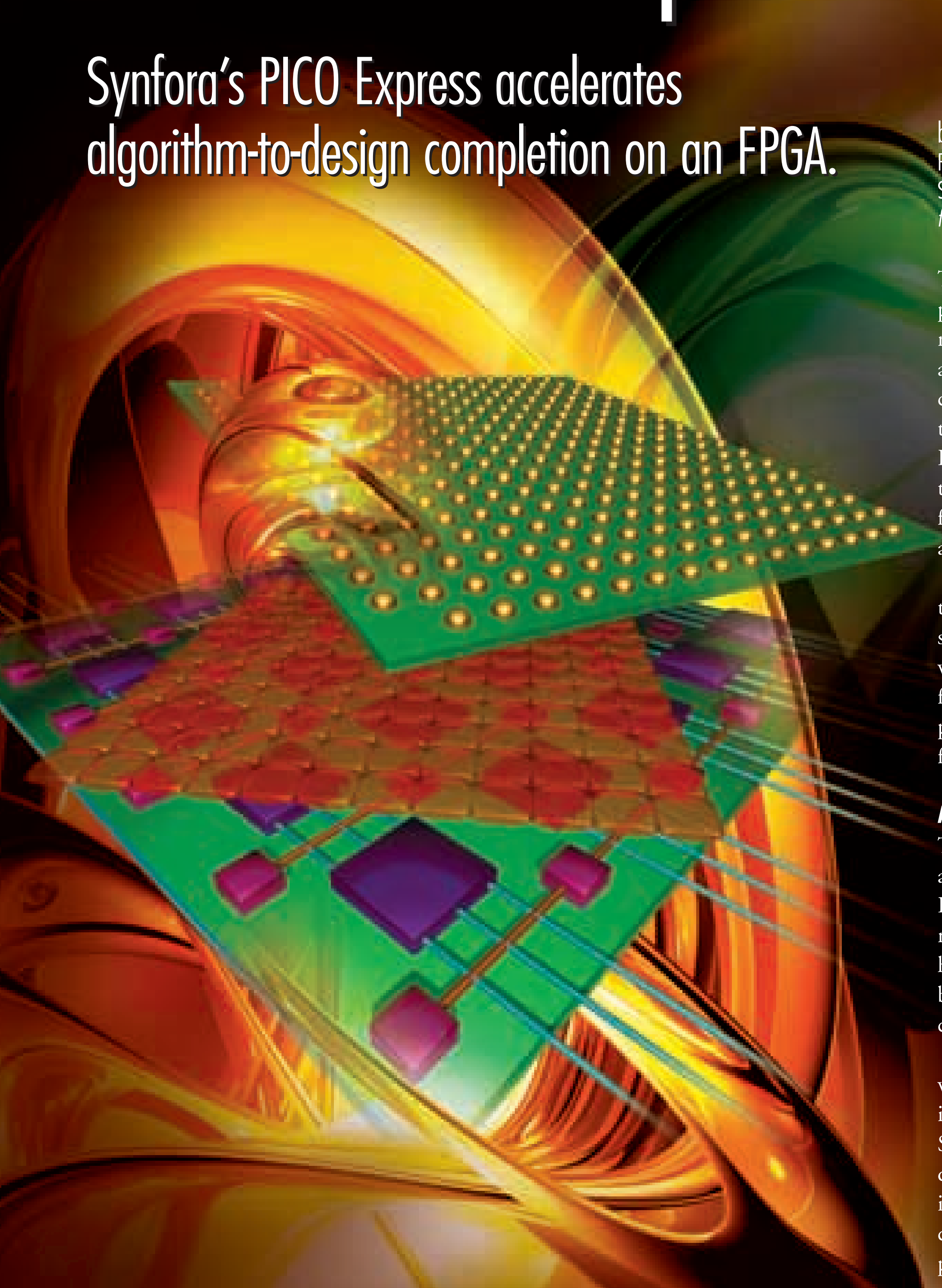
May 14, 2008
Linley Tech Seminar:
High-Speed Interconnects
San Jose, CA

May 14-16, 2008
Embedded Systems Expo 2008
Tokyo, Japan

XILINX®

Reduce FPGA Design Time with PICO Express FPGA

Synfora's PICO Express accelerates algorithm-to-design completion on an FPGA.



by Fernando Martinez
Field Application Engineer
Synfora
fernando@synfora.com

The Xilinx® Spartan™ and Virtex™ product families offer designers a wide range of choices for embedded design. This array of products allows designers to create custom accelerators for embedded applications as well as entire systems on a single FPGA. The most vexing problem facing today's FPGA designers is shifting away from part capacity and toward design turnaround time.

Depending on the complexity of the target application, going from a written specification or reference C model to a working FPGA design can take anywhere from weeks to months. This increases the pressure on designers and makes the goal of first to market harder to reach.

Advanced Algorithmic Synthesis

The only way to accelerate design time on an FPGA is to automate the creation of the RTL. Although several attempts have been made to go from C to RTL, these solutions have fallen short of customer expectations because of language transformations, application complexity, and size limitations.

These concerns, which have limited the widespread adoption of high-level synthesis in the FPGA market, are addressed by Synfora's PICO platform, a complete development environment capable of taking sequential algorithmic ANSI-C and creating high-quality parallel hardware on par with RTL created by hand.

PICO is also fully integrated with FPGA back-end flows from both Xilinx and Synplicity. You can either use your own synthesis scripts or scripts provided by PICO to generate an FPGA programming bitstream.

PICO Express FPGA, which is a key component in the PICO platform, provides a complete development environment capable of taking sequential algorithmic ANSI-C and creating high-quality parallel hardware that is on par with RTL created by hand.

The design methodology in PICO Express FPGAs allows design teams to move to a higher level of abstraction without sacrificing design performance. Another aspect of the PICO design flow is the ability to retarget RTL generation across Xilinx product families in terms of part, speed grade, package, and clock frequency without having to modify the C code.

PICO is also fully integrated with FPGA back-end flows from both Xilinx and Synplicity. You can either use your own synthesis scripts or scripts provided by PICO to generate an FPGA programming bitstream.

Besides integration into standard FPGA back-end flows and the ability to target Spartan and Virtex parts, PICO gives you the ability to carry out what-if analysis. Under design space exploration, PICO allows side-by-side comparisons of multiple designs targeted at different throughputs, clock frequencies, part families, package options, and speed grades.

In this article, I'll show how PICO Express can quickly take an ANSI-C functional model and create a fully functional FPGA design. The application chosen for this demonstration combines real-time color space conversion from RGB to YUV and a Sobel edge detector. Although the C model for this application is part independent, Synfora has implemented a working system on the Spartan-3E video display kit.

Application Development on PICO Express

Application development on PICO Express is divided into three stages. The first two stages are the driver and synthesizable C model creation. As the example will show, the synthesizable C model is completely sequential without any extra need for a user declaration of parallelism. The advanced compiler built into PICO automatically analyzes user applications to extract data dependencies and derive parallelism from sequential code.

The driver code comprises a test bench used to verify the correct operation of the function targeted for hardware synthesis. In the case of this example, the driver code reads input image files, passes the data to the hardware procedure, and gathers the results. The results of the hardware function are compared against the expected output created during test vector selection.

During the several steps needed to go from C to RTL, PICO simulates each transformation to ensure correct hardware by design. The simulations during hardware design are also useful in estimating design performance and resource utilization before RTL simulation and back-end place and route.

The second stage is the creation of the synthesizable C model. The synthesizable C model is the software procedure that will be implemented as a hardware block. This procedure code can encompass functionality as simple as a single accelerator block to functionality as complex as a complete H.264 encoder/decoder. You only need to let PICO know the name of the top-level procedure in the synthesizable model. PICO Express automatically transforms all sub-procedures into hardware.

Color Conversion and Edge Detection

The color conversion and Sobel edge detection application is divided into two main components. Color conversion takes the RGB from the video source and creates an on-the-fly YUV representation of this video. The conversion to YUV has the benefit of acting as a filtering operation, which reduces noise in the original RGB video stream.

Figure 1 shows a summarized pseudo code for the color converter; note that the type of C code accepted by PICO Express is no different from the C code accepted by any ANSI-C compatible software compiler. This allows for code development using open-source tools such as gcc.

```
unsigned char rgb_to_yuv(unsigned int rgb) {
...
  resulty = ((66* Rf)> 8) + ((129* Gf)>8) + ((25* Bf)>8) + 4096;
  resulty = (result + 128) > 8;
  if(resulty>255) resulty=255;
...
}
```

Figure 1 – RGB to YUV converter

After color conversion, the new video stream is passed through a Sobel edge detector. Sobel edge detection is a classic filter used in image processing to reduce the information in both still images and video to a series of edges. The fundamentals of Sobel edge detection are based on:

$$N(x,y) = \sum_{k=-1}^{1} \sum_{j=-1}^{1} K(j,k)\, p(x-j, y-k)$$

This equation shows the use of a 3 x 3 window to calculate the strength of an edge at the center of the window. Depending on the chosen threshold, pixels will either have a value of 1 or 0 after the windowing function. Figure 2 shows the code used for the edge detector.

Like the code in Figure 1, the code in Figure 2 is completely compliant with ANSI-C and can be compiled on any C compiler. Figure 2 shows the use of arrays for storing both horizontal and vertical windows.

```
unsigned char sobel(
unsigned int window[3][3])
{
...
 y00 = rgb_to_y(window[0][0]);
  y01 = rgb_to_y(window[0][1]);
  y02 = rgb_to_y(window[0][2]);
  y10 = rgb_to_y(window[1][0]);
  y12 = rgb_to_y(window[1][2]);
  y20 = rgb_to_y(window[2][0]);
  y21 = rgb_to_y(window[2][1]);
  y22 = rgb_to_y(window[2][2]);

  weight_horizontal = weight_calc(
y00, y01, y02, y20, y21, y22);

 weight_vertical = weight_calc(
y00, y10, y20, y02, y12, y22);

 total_weight = weight_horizontal + weight_vertical;
 return total_weight >= THRESHOLD;
}
```

Figure 2 – Sobel edge detector

PICO will analyze arrays in your code to determine the optimal mapping to minimize resource utilization and maximize performance. Depending on the size of an array and how it is used in the design application, PICO will determine the optimal storage medium for the array as LUTs, block RAMs, or external memories. It is the advanced memory analysis capabilities of PICO that allow for a simple coding model for memories. Arrays are a common storage structure with an easily understood usage model in the C domain. Therefore, designers can specify both simple and complex memory architectures with ease.

Figure 3 summarizes the data and control flow in this example. For the board implementation, the bitstream is mapped onto a Spartan-3E device. The base C code used to create the hardware is independent of the source video resolution. Like in most video processing systems, the PICO-designed hardware will use the first image frame to calculate the image resolution from the video source.

In this case, the video is of HD quality at 720p resolution. On the FPGA, the system runs at 133 MHz, which is sufficient to comply with the DVI standard for 720p HD video. In terms of synthesis time, this application takes less than five minutes in PICO to generate RTL and less than 10 minutes to generate a bitstream from Xilinx Synthesis Technology (XST).

Conclusion

PICO Express for FPGA provides the necessary tools and methodology for you to quickly go from algorithmic code to fully functional designs on an FPGA. The ANSI-C based input accepted by PICO presents you with a familiar coding language and helps reduce the learning curve associated with high-level synthesis tools. Also, PICO provides the capability to quickly compare price/performance trade-offs that allow you to choose the right part without having to modify your application code.

For additional information on how PICO Express can accelerate the development of FPGA-based embedded systems, visit *www.synfora.com.*

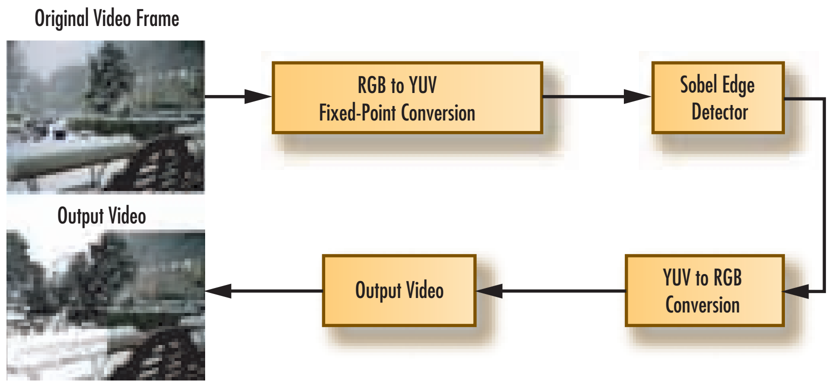


Figure 3 – Color conversion and Sobel edge detector

Design Embedded Systems with Xilinx and Synplicity

You can use the integrated Xilinx EDK and Synplicity's Synplify Pro or Synplify Premier software development flow to synthesize and debug.

by Angela Sutton
Sr. Product Marketing Manager
Synplicity, Inc.
sutton@synplicity.com

The Xilinx® Embedded Development Kit (EDK) is available to designers wishing to implement embedded designs using IBM PowerPC hard processor cores, Xilinx MicroBlaze™ embedded processor cores, and Xilinx-supplied implementations of buses, block RAMs, and peripherals such as USB and PCIe.

If you generated an embedded processor subsystem with EDK and wish to synthesize and debug your complete system for implementing on a Xilinx FPGA, you can do so using the Synplify Pro and/or Synplify Premier (referred to in this article as Synplify Pro/Premier) integrated embedded design flows from Synplicity, Inc. These Synplify Pro/Premier/EDK flows allow you to perform optimization and debug operations on an entire embedded FPGA system, including Xilinx embedded cores.

An integrated Synplify Pro/Premier/EDK flow allows designers to synthesize an entire design top-down as one design entity, handled as a single Synplify Pro/Premier design project.

Flow	When to Use	Details
ISE software/EDK hardware development flow (.ise project file)	Most processor-based embedded subsystems and designs with non-peripheral custom logic	• Allows the addition of non-peripheral custom logic to the system • Allows the addition of HDL files that contain non-peripheral custom logic directly to the project • You can integrate processor subsystems as either top-level modules or sub-modules
Standalone EDK hardware development flow (.xmp project file)	PowerPC or MicroBlaze processor-based embedded system or sub-modules	• The flow includes system top-level hardware synthesis, mapping, and place and route processes in the background so that it is not necessary to work with Xilinx ISE software to generate the configuration bitstream file. • Compiles and executes multiple software applications, generates libraries for the code, and merges software with hardware files for downloading to a target board. • Each hardware component is treated as an independent core (a separate XST project file is generated for each core). • During synthesis, XST automatically runs on each core separately to generate netlists (NGC files) through the Xilinx Xflow. • If the core is encrypted (MicroBlaze processor cores, for example), XST is always invoked to provide an NGC netlist.

Table 1 – Choosing which Synplify Pro/Premier/EDK flow to use

In addition to automating the flow, this integrated Synplify Pro/Premier/EDK flow offers the following advantages:

- Better quality of results (QoR): Synplify Pro/Premier considers the design in its entirety when performing timing-driven and area optimizations. Moreover, the tools can time through the core and may potentially optimize across EDK core boundaries, which improves design performance QoR.
- Visibility and debugging capabilities during implementation. You only need to run synthesis once to generate the final design netlist. Debugging and optional floorplanning can be performed on the entire design. You can also use Synplify Pro/Premier's HDL analyst, timing analyst, physical analyst, island timing analyst, and add-on Identify RTL debugging tool to debug and analyze the entire design.

Let's describe this flow in more detail.

Overview of Xilinx EDK Flow

The Synplify Pro or Synplify Premier tool from Synplicity is required to perform synthesis, as a part of the overall embedded FPGA hardware creation process, after the EDK cores have been included in the design. You can use the two available Xilinx EDK embedded hardware development flows that exist today to create your embedded design and to generate the embedded hardware in HDL format.

The two flows are:

- Xilinx stand-alone EDK hardware development flow for those using Xilinx Platform Studio (XPS)
- Xilinx ISE™ software/EDK hardware development flow for those using Xilinx ISE software

Table 1 includes additional details about which of these two flows to use. Figure 1 illustrates the overall design flow steps for both flows.

Integrated Synplify Pro/Premier/EDK Flow

The Synplify Pro/Premier/EDK flow comprises four steps, as shown in Figure 2.

Here are some details about each step.

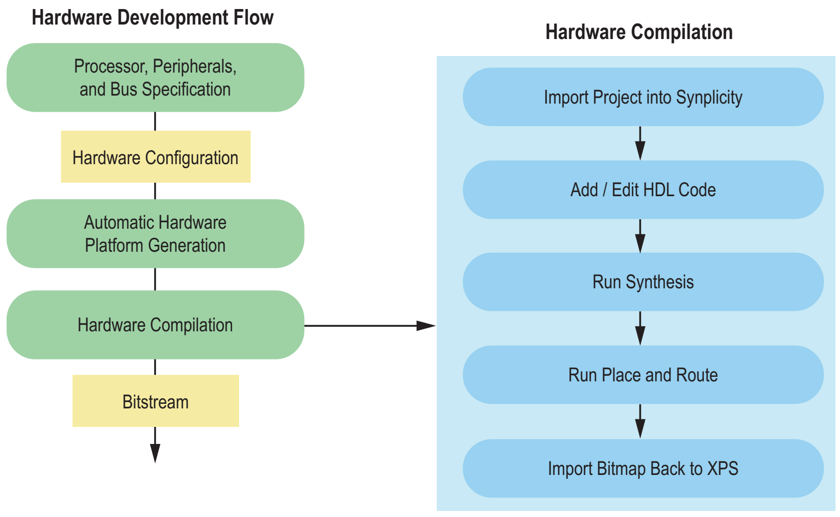

Hardware Development Flow
Processor, Peripherals, and Bus Specification
Hardware Configuration
Automatic Hardware Platform Generation
Hardware Compilation
Bitstream
Hardware Compilation
Import Project into Synplicity
Add / Edit HDL Code
Run Synthesis
Run Place and Route
Import Bitmap Back to XPS

Figure 1 – Synplify Pro/Premier/EDK flow. Embedded cores are read into the Synplify Pro/Premier tool, where the design is fully synthesized. The post-place and route-generated bitmap is returned to Xilinx Platform Studio.

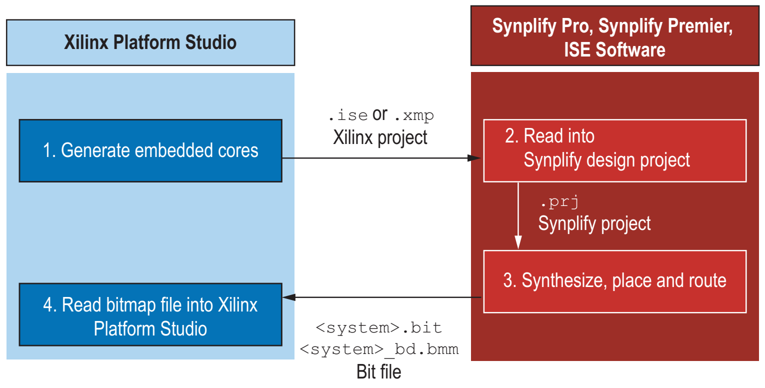

Xilinx Platform Studio
Synplify Pro, Synplify Premier, ISE Software
1. Generate embedded cores
.ise or .xmp Xilinx project
2. Read into Synplify design project
.prj Synplify project
3. Synthesize, place and route
4. Read bitmap file into Xilinx Platform Studio
<system>.bit
<system>_bd.bmm
Bit file

Figure 2 – The four steps involved in the EDK Synplify Pro/Premier flow.

1. Generate embedded cores using XPS or ISE software. Create an EDK project that Synplify Pro/Premier can read to generate a core netlist using one of the two flows for XPS users:
   - The Xilinx stand-alone EDK hardware (generates an .xmp file)
   - Xilinx ISE software/EDK hardware development flow (generates an .ise file) for ISE software users

In the XPS GUI:

- Select Project > Project Options > Hierarchy and Flow
- Select Implement Design in ISE software (Export to Project Navigator Flow: Depreciated)
- Select Hardware > Generate Netlist to generate the netlist

The .ise or .xmp file generated will refer to a variety of design files that represent the core. These files are shown in Table 2.

2. Read the generated files into Synplify Pro/Premier design project. The files that Synplify Pro and Synplify Premier automatically read are listed in Table 3. The Synplify Pro/Premier tools read the NGC netlist and EDK-generated platform (as specified by the .ise/.xmp project files); they are automatically included in the design project. The Synplify Pro/Premier software reads many different files that are part of the Xilinx project file, including RTL, netlists, and constraints (V/VHD, EDN, NGC, NGO, SDC, UCF, OPT). It adds these definition and constraints files to the synthesis project it creates before running synthesis.

Files and Directories Created by EDK	
.xmp	Xilinx microprocessor project file
.mhs	Microprocessor hardware specification file
.mss	Microprocessor software specification file
pcores/	User IP cores
hdl	HDL wrappers (encrypted cores)
etc/	UCF Xilinx constraint files
data/	Place and route option files
synthesis/	Synthesis files
implementation/	Top-level place and route results, block .ngc netlist files

Table 2 – The embedded core files generated by EDK software

Note that any custom HDL logic will be included in the .ise project file and read automatically by the Synplify Pro/Premier tool if you used the Xilinx EDK/ISE hardware development flow. However, you must add these files manually to the Synplify Pro/Premier project if you are using the Xilinx stand-alone EDK hardware development flow.

3. Invoke and run Synplify Pro/Premier, then place and route.
   - In the Synplify Pro/Premier tool, open the Synplify Pro/Premier design (.prj) project, which now includes the .ise/.xmp embedded cores. The cores will now be part of the Synplify Pro/Premier design.
   - If the EDK cores are a subsystem in a larger design, simply instantiate the subsystem into the top-level HDL design.
   - Run synthesis and then complete place and route using the normal Synplify Pro/Premier ISE software flow. You can run Xilinx place and route from the Synplify Pro/Premier interface to generate a top-level hardware bitmap file.

Table 4 shows the output files created after synthesis, including a bitmap file.

4. Read bitmap back into XPS.
   - Import the bitmap file back into the XPS environment. The <system>.bit and <system>_bd.bmm files originally found in the synplify/rev_1/par_1 directory should be placed in the "implementation" directory within the EDK project directory. In the XPS GUI, select Project > Import from ProjNav.
   - Select the BIT and BMM files from the PAR directory
     - BIT file synplify/rev_1/par_1/<system>.bit
     - BMM file synplify/rev_1/par_1/edkBmmFile_bd.bmm

Input Files Read by Synplify Pro/Premier	
.ise	Project file in ISE software project directory, if created
.xpm and .mhs	Project file in EDK project directory, if created
.mpd, .pao, HDL (.v, .vhd), .edn, .ngo, .ngc	Design specification files in EDK project directory, if available, or in EDK hardware library in a location specified through the EDK installation path

Table 3 – These EDK files are read by Synplify Pro and Synplify Premier

Output Files Created by Synplify Pro/Premier	
.prj	Project file
.sdc	Constraint file
.ucf	Xilinx user constraint file
OPT file and core wrapper file	Xilinx Xflow OPT file and core wrapper file generated, if the core is a black box to the Synplify directory (for example, if the core was encrypted)
.bmm	Bitmap file – in Synplify Pro/Premier place and route directory

Table 4 – Output files generated by Synplify Pro and Synplify Premier, ready for Xilinx Platform Studio to read

   - Click OK
   - Run Device Configuration > Update Bitstream. You can now download the bitstream into the FPGA.

Encrypted Cores

In some cases, the cores are encrypted by Xilinx (for example, Xilinx MicroBlaze processors). You can merge these cores into your design with Synplify Pro/Premier. The core may appear as a black box in the tool.

The PAO file generated by XPS will point to an encrypted IP core definition file. Synplify Pro/Premier software allows you to specify the appearance of the core as a black box (in which case the core wrapper file will be present and copied into the Synplify Pro/Premier folder). If you specify that the core should not be treated as a black box – in other words, a predefined netlist exists for the core – the tool will seek to locate a definition file named implementation/<core>.ngc. If it finds the file, it adds it to the Synplify Project .prj file. If it does not find the implementation/<core>.ngc file, it issues an error message.

In Synplify Pro/Premier software version 9.0.2 and above, a new secure .ngc flow exists that allows you to easily include encrypted cores in the synthesis flow. It also allows the Synplify Pro/Premier tool to additionally optimize inside cores, resulting in better overall QoR.

Conclusion

An increasing percentage of FPGA designs include cores. For designers generating microprocessor and peripheral design cores using the Xilinx Embedded Development Kit (EDK), the Synplify Pro and Synplify Premier software tools offer a fully integrated flow that allows you to synthesize and implement an FPGA system.

These tools offer better QoR and a more convenient flow, as well as full design debugging and analysis during the synthesis and placement phases. For more information, contact Synplicity customer support at *support@synplicity.com.*

Embedded Processing Application Notes and Reference Systems

Popular application notes and reference systems available from Xilinx.

Xilinx maintains and updates a large database of application notes on numerous topics as well as reference systems to assist you in your next design.

PowerPC 440 System Simulation

http://www.xilinx.com/support/documentation/application_notes/xapp1003.pdf

This reference system demonstrates the functionality of the PowerPC 440 processor block on the Virtex™-5 FXT FPGA. This system includes common peripherals like Ethernet, DDR2, DMA, interrupt controller, and RS232.

Several stand-alone software applications can verify functionality of the peripherals and the PowerPC 440 processor block.

You can learn how to set up the simulation environment for the system, execute the simulation, and add PLB v4.6 peripherals. The Xilinx ML507 Rev A board verifies the system in hardware on the Virtex-5 FXT FX70TFF1136-1FPGA.

PLBv46 PCI Using the ML555 Embedded Development Platform / ML410 Embedded Development Platform / Avnet Spartan-3 FPGA Evaluation Board

http://www.xilinx.com/support/documentation/application_notes/xapp999.pdf

http://www.xilinx.com/support/documentation/application_notes/xapp1001.pdf

http://www.xilinx.com/support/documentation/application_notes/xapp1038.pdf

These application notes describe how to build reference systems for the processor local bus peripheral component interconnect (PLBv46 PCI) core using either a PowerPC 405 or MicroBlaze™ processor-based embedded system.

A set of files containing Xilinx Microprocessor Debugger (XMD) commands are provided for writing to the configuration space headers and to verify that the PLBv46 PCI core is operating correctly. Several software projects illustrate how to configure the PLBv46 PCI core(s), set up interrupts, scan configuration registers, and set up and use DMA operations.

PCI Bus Performance Measurements Using the Vmetro Bus Analyzer

http://www.xilinx.com/support/documentation/application_notes/xapp998.pdf

This application note illustrates how to measure performance using the Vmetro Vanguard PCI bus analyzer. These measurements use a system in which the ML410 evaluation platform, the ML555 evaluation platform, and Vmetro Vanguard-PCI (VG-PCI) boards communicate over the PCI bus. The test method is provided so that similar tests can be done using different PCI bus transactions, different boards, or other Xilinx PCI cores.

Accessing Spartan-3AN In-System Flash Using XPS SPI

http://www.xilinx.com/support/documentation/application_notes/xapp1034.pdf

The application note demonstrates how to access the in-system flash in a Spartan™-3AN FPGA after the FPGA is configured. The software applications use the serial peripheral interface (XPS SPI) core in a MicroBlaze processor-based reference system.

Introduction to Software Debugging on Xilinx PowerPC 405 Embedded Platforms / MicroBlaze Processor Embedded Platforms

http://www.xilinx.com/support/documentation/application_notes/xapp1036.pdf

http://www.xilinx.com/support/documentation/application_notes/xapp1037.pdf

These application notes offer an introduction to software debugging of Xilinx PowerPC 405 or MicroBlaze embedded processor platforms by discussing the use of the Xilinx Microprocessor Debugger (XMD) and the GNU software debugger (GDB) to debug software defects.

Ethernet PHY Register Access with GPIO

http://www.xilinx.com/support/documentation/application_notes/xapp1042.pdf

The XPS Ethernet MAC lite peripheral does not provide any mechanism to access Ethernet PHY registers. These registers are used to configure auto negotiation parameters and to obtain PHY status. This application note provides a PowerPC 405 reference system and a MicroBlaze processor system where the PHY serial management interface signals (MDC, MDIO) are connected to an XPS GPIO peripheral.

Setup of a MicroBlaze Processor Design for Off-Chip Trace

http://www.xilinx.com/support/documentation/application_notes/xapp1029.pdf

This application note describes how to modify an existing MicroBlaze processor design to support the trace features in MicroBlaze processor version 7 and above. It is a detailed tutorial on how to add and configure the trace core in a MicroBlaze processor design. The application note does not target any specific development board, but I/O constraints for some boards are provided to work with Lauterbach and Computex trace tools.

VxWorks 6.x on the ML403 Embedded Development Platform

http://www.xilinx.com/support/documentation/application_notes/xapp947.pdf

This guide shows the steps required to build and configure a ML403 embedded development platform to boot and run the VxWorks RTOS. A VxWorks bootloader is created, programmed into flash, and used to boot the design. The concepts presented here can be scaled to any PowerPC-enabled development platform. This popular application note is now available for EDK / ISE™ software 10.1.

Embedded Processing QuickStart!

Get your design off to the right start.

by Jannis McReynolds
Sr. Manager, Services Marketing
Xilinx, Inc.
jannis.mcreynolds@xilinx.com

More than ever, designers are being asked to reduce system costs, reduce power consumption, and meet aggressive design schedules. With QuickStart!, Xilinx helps you meet these challenges with an unprecedented level of on-site support and training.

QuickStart! is available for a wide range of technologies, including embedded systems, PlanAhead™ software floorplanning, timing optimization, multi-gigabit transceivers, and PCIe integration.

QuickStart! provides you with the accelerated ramp-up time you need to go to market faster. QuickStart! ensures that you will have a more knowledgeable team – one that is capable of executing better designs, lowering overall costs, and enhancing your competitive edge.

What is QuickStart!?

QuickStart! supplies you with an engineer at your site for one week. The Xilinx expert will train and empower your team to complete your project on time and on budget, while ensuring you make the best use of your Xilinx device.

QuickStart deliverables include:

- A customized, two-day, on-site, technology-specific course
- Three days of consulting from a senior applications engineer
- Expert advice on design architecture and implementation optimization

Two-Day Customized Course: Embedded Systems Development

Customers engaged in an embedded design will receive the Xilinx Embedded Systems Development course.

This course gives you a better understanding of how to develop a PowerPC and MicroBlaze™ processor embedded system by using the Embedded Development Kit (EDK). This course provides hands-on labs regarding the development, debugging, and simulation of the embedded system. Labs provide you with a choice of targeting either PowerPC or MicroBlaze processor systems.

Note that QuickStart! is available for projects beyond embedded processing. Please see Table 1 for details about our additional offerings.

Three Days of Consulting: Deliverables

- Configuration of the Xilinx design environment
- Set up and customization of EDK software
- Design architecture and implementation consultation and guidance
- System partitioning consultation
- Advice on using of advanced features and capabilities of the Xilinx MicroBlaze processor and PowerPC processor

How You Benefit

- Shortened ramp-up times by learning essential design and debugging techniques
- Empowered hardware teams through coaching on software-based optimized systems
- Maximum performance gained from understanding advanced and proven FPGA design techniques
- Minimized design risks acquired by using the finest expert advice

Titanium Dedicated Engineering

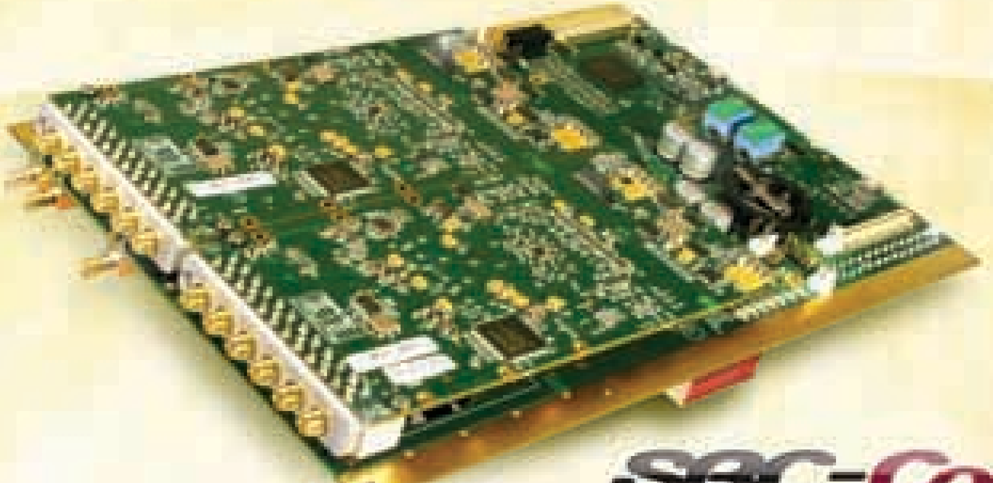
Titanium Dedicated Engineering provides a senior engineer on-site similar to QuickStart! However, Titanium allows for longer engagements (more than one week) and does not include a training class. For more information, visit *www.xilinx.com/titanium.*

Take the Next Step

Get your next Xilinx design off to the right start with QuickStart! To learn more, visit *www.xilinx.com/quickstart.*

Type	Course	Specific Features
Embedded	Embedded Systems Development	Design environment configuration EDK software customization Design architecture consultation System partitioning guidance MicroBlaze / Power PC processor optimization
PlanAhead Software	Designing with PlanAhead Software	PlanAhead software design environment configuration Floorplanning implementation consultation Synthesis and project tips Design and performance-improving analysis
Timing Optimization	Designing for Performance	Proper constraining techniques Best-known HDL coding methods Timing optimization reviews Debugging techniques demonstrations
MGT	Designing with MGTs	MGT ports and attributes education Advanced features utilization Configuration methods Simulation and implementation flows
PCIe Integration	Designing with MGTs	Migrating Virtex-II/Virtex-4 software designs Advanced timing closure Evaluating design requirements

Table 1 – QuickStart engagement types



Industry-Standard COM-Express CPU plus Dual XMC I/O Modules in a Compact Embedded PC

Use SBC-ComEx to create powerful Embedded Instruments or address difficult Distributed Data Acquisition Applications. Add XMC (PCI Express) Modules, available for any application from RF Receivers to Servo Control, to create the perfect Embedded Instrument!

COM Express CPU
- Low-Power Celeron or Core 2 Duo Performance
- Up to 4 GB System Memory
- Windows XP Embedded and Linux Ready

Rugged & Compact
- Boots from SATA HDD or USB FLASH
- AC or 12V DC Operation
- Small 250 x170mm Form Factor

GPS Support
- Integrated Timing and Triggering via GPS-disciplined Clock

Vast I/O
- PCI Express I/O Sites Deliver >800 MB/s to System Memory
- On-board USB 2.0 (x6), Gigabit Ethernet, SATA (x4), VGA, AC '97 Audio
- Supports X5/X3 XMC I/O Module Features (Private Data Channels, Triggering, Timing)

Compatible XMC Modules from Innovative Integration

X5-400M - 400 MSPS 14-bit A/D (x2) • 500 MSPS 16-bit D/A (x2) 1GB DDR2 DRAM
X5-210M - 210 MSPS 14-bit A/D (x4) • 512MB DDR2 DRAM
X3-10M - 25 MSPS 16-bit A/D (simultaneously sampling x8)
X3-25M - 130 MSPS 16-bit A/D (x2) • 50 MSPS 16-bit D/A (x2)
X3-A4D4 - 4 MSPS 16-bit A/D (x4) • 50 MSPS 16-bit D/A (x4)
X3-DIO - 64-bit 66MHz LVDS
X3-SD - 216 kHz 16-bit A/D (x16)
X3-SDF - 20 MHz 24-bit A/D (x4)
X3-Servo - 250 kHz 16-bit A/D (x6) • 1 MSPS 16-bit D/A (x6)

FrameWork Logic

Innovative Integration
... real time solutions!

www.innovative-dsp.com • sales@innovative-dsp.com • 805-578-4260

GET UP TO 50% LOWER COST

Xilinx Spartan™-3 Generation devices are the only low-cost FPGA family to cut your total cost by half… and we can prove it! We offer the industry's widest range of devices, most comprehensive IP library (8x the nearest competitor!), a huge selection of development boards and we've minimized the need for external components.

Visit us at **www.xilinx.com/spartan** to download our free ISE™ WebPACK™ design tools and start your Spartan-3 design today. You too can experience the lowest total cost. Period.

XILINX®

www.xilinx.com/spartan

The World's Most Widely Adopted Low-Cost FPGAs

PN 2074