

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

Issue 73
Fourth Quarter 2010

FPGAs Enable Greener Future for Industrial Motor Control

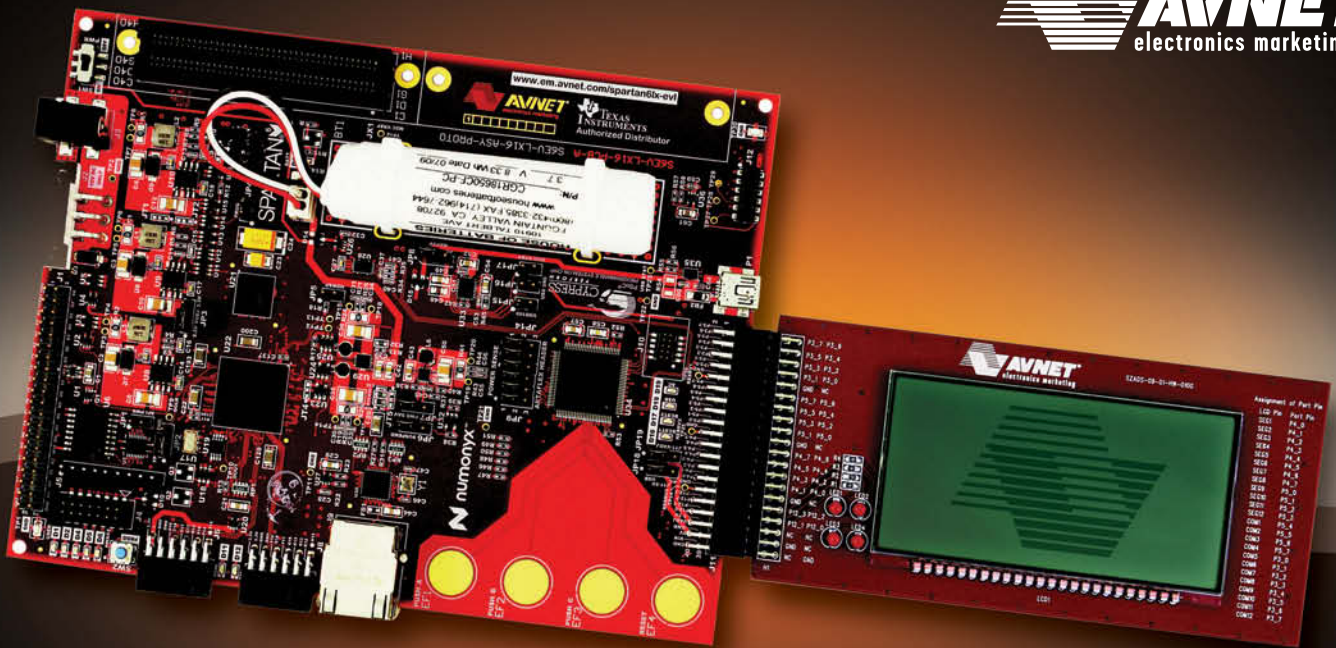
INSIDE

**FPGA Partial Reconfiguration
Goes Mainstream**

**Resurrecting the Mighty
Cray on a Spartan-3 FPGA**

**How Hardware Accelerators
Can Speed Your Processor's
Sine Calculations**

**Xilinx Releases 2009
Quality Report**



DESIGNED BY AVNET

Introducing the First-ever Battery-powered Xilinx FPGA Development Board

The low-cost Xilinx® Spartan®-6 FPGA LX16 Evaluation Kit, designed by Avnet, combines a Spartan-6 LX16 FPGA with a Cypress PSoC® 3 controller and LPDRAM memory. This kit demonstrates a versatile battery-powered solution for low-power applications and provides a sound development environment for the most demanding applications. The baseboard can also be expanded with new FPGA Mezzanine Card (FMC) modules, making design porting and FMC swapping nearly seamless between Avnet and Xilinx platforms. FMC modules for ISM Networking, Dual Image Sensing and DVI I/O are currently available from Avnet.

Spartan®-6 FPGA LX16 Evaluation Kit Includes:

- Spartan-6 LX16 DSP FPGA
- LCD add-on panel
- USB A-mini B cable
- ISE® WebPACK™ DVD
- AvProg configuration & programming utility
- Downloadable documentation & reference design

To purchase this kit, visit
www.em.avnet.com/spartan6lx-evl
 or call 800.332.8638.

Quickly see inside your FPGA



Get the most out of your test equipment and shave time out of your next debug cycle with Agilent oscilloscopes and logic analyzers. Our innovative FPGA dynamic probe application allows you to quickly connect to multiple banks of internal signals for rapid validation with real time measurements. Reduce errors with automatic transfer of signal names from the .cdc file from your Xilinx Core Inserter.

Toggle among banks of internal signals for incremental real-time internal measurements without:

- Stopping the FPGA
- Changing the design
- Modifying design timing

Shown with: 9000 Series oscilloscope and 16900 logic analyzer with FPGA dynamic probes



*Also works with all InfiniiVision
and Infiniium MSO models.*

**See how you can save time by downloading our
free application note.**

www.agilent.com/find/fpga_app_note



Xcell journal

PUBLISHER	Mike Santarini mike.santarini@xilinx.com 408-626-5981
EDITOR	Jacqueline Damian
ART DIRECTOR	Scott Blair
DESIGN/PRODUCTION	Teie, Gelwicks & Associates 1-800-493-5551
ADVERTISING SALES	Dan Teie 1-800-493-5551 xcelladsales@aol.com
INTERNATIONAL	Melissa Zhang, Asia Pacific melissa.zhang@xilinx.com Christelle Moraga, Europe/ Middle East/Africa christelle.moraga@xilinx.com Miyuki Takegoshi, Japan miyuki.takegoshi@xilinx.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/

© 2010 Xilinx, Inc. All rights reserved. XILINX, the Xilinx Logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

The articles, information, and other materials included in this issue are provided solely for the convenience of our readers. Xilinx makes no warranties, express, implied, statutory, or otherwise, and accepts no liability with respect to any such articles, information, or other materials or their use, and any use thereof is solely at the risk of the user. Any person or entity using such information in any way releases and waives any claim it might have against Xilinx for any loss, damage, or expense caused thereby.

Xilinx Demonstrates Quality Commitment to Customers

A few years ago, I wrote a cover story for *EDN* magazine on the subject of reliability in electronic design. In the process of researching the topic, I learned many interesting things—first of which is that most companies don't want to speak publicly about the quality of their products. Most, not all.

That original article was inspired by a shopping experience. While ringing up an Xbox 360 that I was purchasing for the kids, the salesperson strongly suggested I also buy an aftermarket cooling unit that plugged into the console to keep it from getting too hot. At the time it seemed absurd that after forking over hundreds of dollars for a videogame system I'd need to pony up extra money to make up for the possibility that there were deficiencies in the Xbox 360's design.

Needless to say, I didn't buy the cooling gizmo, and a year later (and a year after I wrote the reliability article) my Xbox 360 experienced what many Xbox 360 owners collectively called "the red ring of death." Microsoft subsequently fixed the defect (rumored to be a thermal issue with an ASIC) for free for all its customers. The company must have forked over untold amounts of cash to respin the ASIC, set up the infrastructure to refurbish units, cover postage and organize the training for customer service folks to say "I've never heard of the term 'red ring of death'" with some attempt at sincerity. Given the quasi recall, one has to wonder how much more successful the Xbox would have been had Microsoft not encountered the quality issue.

Quality is really something everyone in electronics should be more mindful of, especially as IC silicon processes continue to shrink and can accommodate ever-more-elaborate designs that power a broader range of applications, spanning from consumer electronics to medical and mission-critical systems.

So given this background, I was very proud to see our quality team establish one of the few corporate quality reports available for public consumption. For many years, our customers have recognized Xilinx for its outstanding quality. The 2009 quality report—downloadable in PDF form at https://docs.google.com/viewer?url=http://www.xilinx.com/publications/prod_mktg/2009-quality-annual-report.pdf—highlights Xilinx's continued improvements in quality and customer satisfaction.

"Quality is no longer just about silicon," said Vincent Tong, senior vice president of worldwide quality and new product introductions. "The whole Xilinx design experience and IP ecosystem are now being transformed into the types of programs we've run for decades to drive relentless quality improvements and superb customer experiences."

In keeping with this thrust, Tong says this year's report will focus on Xilinx's Targeted Design Platforms, with an emphasis on how quality is important in each element of what Xilinx delivers to its customers. "We offer outstanding FPGAs, but our commitment to quality does not end with silicon," said Tong. "We are committed to improving the overall customer experience. Our customers have done amazing things with our products and we are committed to helping them become even more successful in creating wonderful new innovations."

I encourage you to give the 2009 quality report a read.



Mike Santarini
Publisher



SPARTAN⁶

- Easy-to-use, low power, serial transceivers support up to 3.125Gbps to enable industry standards such as PCIe[®]
- Low voltage option reduces total power consumption by 65% over previous generations
- Integrated DSP, memory controllers, and clocking technology simplifies designs

VIRTEX⁶

- High bandwidth serial connectivity with up to 72 low-power transceivers supporting up to 11.18Gbps
- Ultra high-performance DSP using up to 2016 low-power, performance-optimized DSP slices
- Integrated high-performance ExpressFabric[™] technology running at 600 MHz clocking and performance-tuned IP blocks
- Proven cost-reduction with EasyPath[™]-6 FPGAs

Potential. Realized.

Unleash the full potential of your product design with Xilinx[®] Virtex[®]-6 and Spartan[®]-6 FPGA families — the programmable foundation for Targeted Design Platforms.

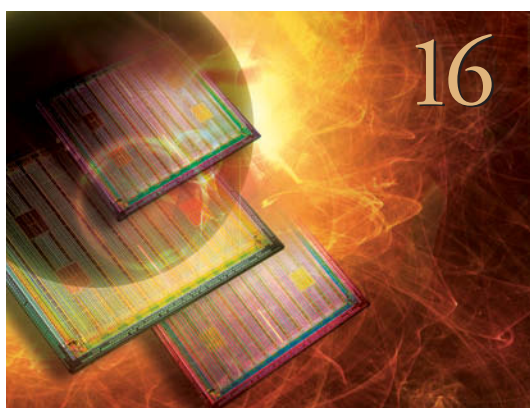
- Reduce system costs by up to 60%
- Lower power by 65%
- Shrink development time by 50%

Realize your potential. Visit www.xilinx.com/6.

VIEWPOINTS

Letter From the Publisher

At Xilinx, Quality Extends Beyond Silicon...4



Cover Story

FPGAs Create a Greener Future
for Industrial Motor Control

8

XCELLENCE BY DESIGN APPLICATION FEATURES

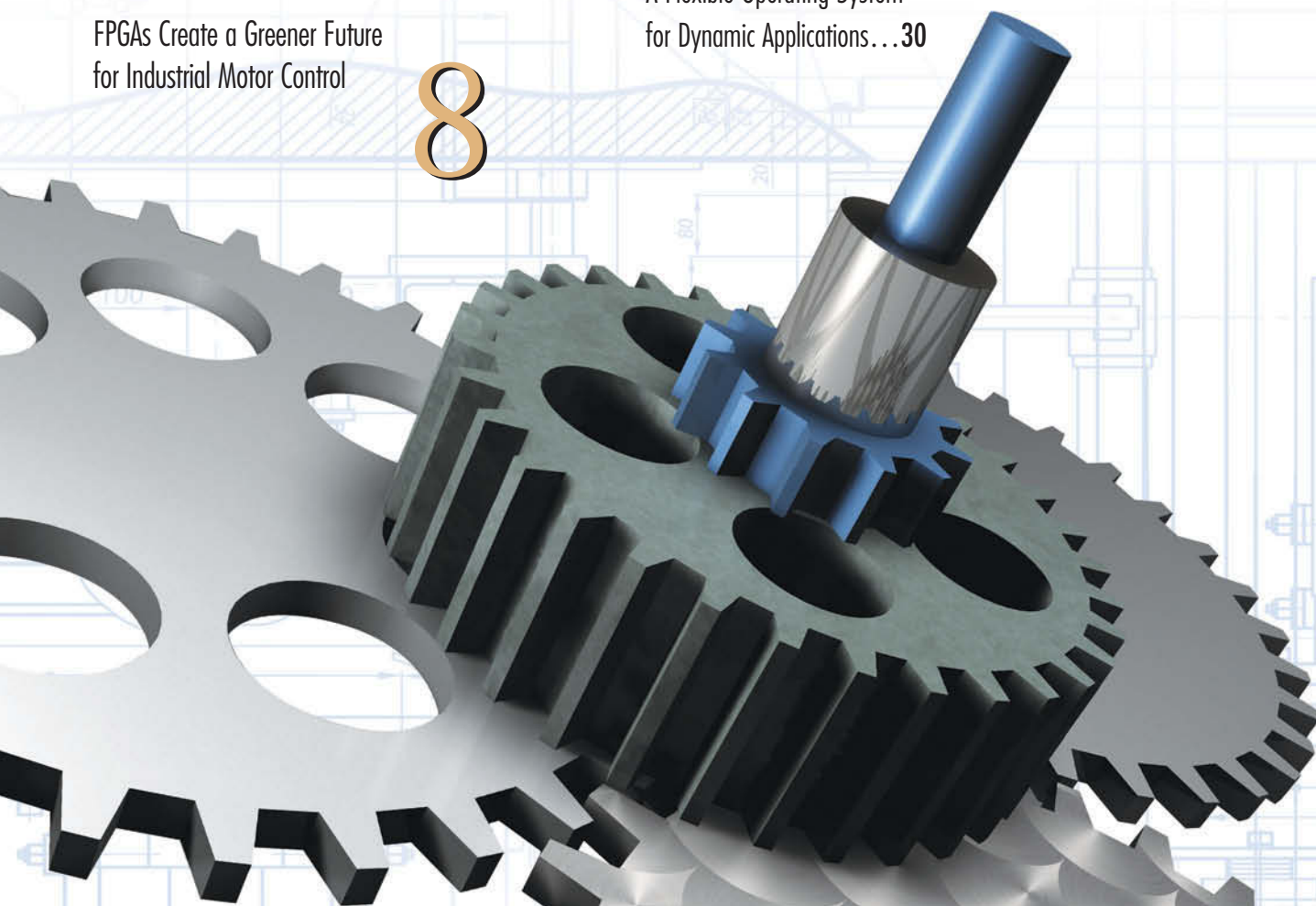
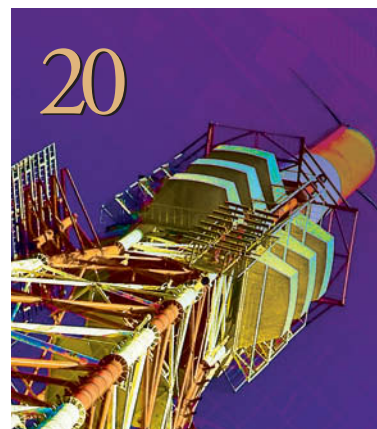
Xcellence in Industrial
FPGAs Fuel a Revolution
in Intelligent Drive Design...12

Xcellence in A&D
Using FPGAs in
Mission-Critical Systems...16

Xcellence in Wireless Comms
Virtex-4 Forms Foundation
for GSM Security...20

Xcellence in Networking
Research Design Platform Ignites
Real-World Network Advances...24

Xcellence in New Applications
A Flexible Operating System
for Dynamic Applications...30



THE XILINX XPERIENCE FEATURES

Xperiment Resurrecting the
Cray-1 in a Xilinx FPGA... **36**

Xperts Corner FPGA Partial Reconfiguration
Goes Mainstream... **42**

Xplanation: FPGA 101 Multirate Signal
Processing for High-Speed Data Converters... **50**

Ask FAE-X How to Speed Sine Calculations
for Your Processor... **54**



XTRA READING

Are You Xperienced? Take advantage
of a simplified training experience... **60**

Xtra, Xtra The latest Xilinx tool
updates and patches, as of September 2010... **62**

Xclamations! Share your wit and wisdom by
supplying a caption for our cartoon, and win an
SP601 Evaluation Kit... **66**



Xcell Journal recently received
2010 APEX Awards of Excellence in the
categories "Magazine & Journal Writing" and
"Magazine & Journal Design and Layout."

Xilinx FPGAs: Creating a Greener Future for Industrial Motor Control

FPGAs are advancing the sophistication of industrial motor control products. An upcoming Targeted Design Platform aims to push it further.

by Mike Santarini
Publisher, Xcell Journal
Xilinx, Inc.
mike.santarini@xilinx.com

Electric motors are everywhere—running toys and appliances in your home, cooling your office and powering cars, trains and factory assembly lines worldwide. They are even found in robots on Mars. Given this wide range of applications, it isn't surprising that there are dozens of types of electric motors, each with advantages and disadvantages. Function, size, power consumption/efficiency, performance, reliability, product life and, of course, cost are all attributes companies take into account when designing new electric motors and deciding which ones to incorporate in their products.

Today, however, the electric motor is at the dawning of a new era of maximum efficiency and automation built upon highly sophisticated FPGA-powered motor control systems. These new systems are just starting to make their way into the factories that create the world's products and consume a huge percentage of the world's power. In the first quarter of 2011, Xilinx will deliver a Targeted Design Platform that will allow companies to quickly develop the most advanced motor control systems to date and help factories reach new levels of automation and efficiency.

'Creating algorithms to control the efficiency of a motor is very complex, and adding these diagnostic capabilities on top of efficiency controls requires a far more sophisticated algorithm.'

The Growing Complexity of Factory Motor Control

Joe Mallett, senior product line manager in Xilinx's industrial, scientific and medical group, breaks industrial motor control into five elements: communications, control, drive, feedback and diagnostics.

At a minimum, said Mallett, all industrial motor control systems employ some type of communications to the outside world to allow a motor to work with other motors via master controllers or enterprise software. "Traditionally, the communications part of motor control for the factory has been based on serial communications, but it is rapidly moving to Ethernet due to the real-time requirements of communications, especially in safety applications, and to more easily link with the enterprise, which has been Ethernet based for years," said Mallett.

Then comes an element called "control," which is the term the industry generally uses to describe the algorithm at the heart of the motor control system. This algorithm controls another element called drive—essentially, the silicon power devices that switch the current on and off to drive the motor. The control and drive work together in what's called "the control and drive loop" to run motors at optimum performance and efficiency given their application.

The next element in a motor control system is feedback. "These are sensors that monitor various aspects of a motor in real time, including position, speed and torque, among others," said Mallett. "To ensure reliability, many companies are trying to minimize the number of sensors they use in their motor control systems. As such, many are moving to sensorless control, which requires high-performance, advanced algorithms to retrieve the feedback information."

The final piece of the system puzzle is diagnostics. "Being able to predict when a motor is likely to fail is becoming a common requirement," said Mallett. Here

too, "prediction and diagnostics require fast decisions and sophisticated signal-processing capabilities."

In recent years, most of the R&D and work in the motor control segment—whether it be for factory equipment, automobiles or even toys—has focused on improving motor efficiency.

Factories depend heavily on electric motors to power robots, tools, assembly lines and HVAC (heating, cooling and ventilation) systems. Many run all this machinery 24 hours a day, seven days a week, to fill orders and keep revenue coming in. As such, reliability and efficiency are paramount, as an unexpected failure in an electric motor can bring an entire assembly line or factory to a halt, and may even harm workers.

Advanced Motor Control Combines Efficiency, Reliability and Safety

Further, with 24/7 operation, energy consumption has a drastic impact on the total cost of ownership and bottom line. In the factory space, efficiency is increasingly mandatory, said Mallett.

"It's always great to drive down energy bills, but the fact is, many factories have no choice in the matter," said Mallett. "They have to comply with government mandates to reduce the pollution they create and the energy they consume. However, most factories can't afford to shut down for a long period of time and replace their entire manufacturing lines with newer, more efficient ones. What they would prefer to do is run their existing machinery more efficiently and swap in newer, more efficient motors over time, as needed—keeping factory downtime to a minimum." To that end, equipment manufacturers are racing to develop the most efficient motor and control combination possible to help factories meet their energy-savings goals, Mallett said.

However, Giulio Corradi, ISM systems architect for the industrial, scientific and

medical markets at Xilinx, pointed out that it's quite complex to create a motor control unit that makes even one motor more efficient because for maximum efficiency, the control algorithms must be tailored to a given motor's targeted task in the factory. The complexity of the algorithm grows exponentially if it needs to control multiple motors and even more so if the motor needs to incorporate feedback or diagnostics.

Corradi explained that even the most basic motor control systems employ complex algorithms to regulate the amount of power going into a motor or series of motors. Incorporating feedback and diagnostics adds another level of complexity, but delivers systems that will instantly warn operators if there is a potential problem with a motor, and perhaps identify what the problem is and gauge its severity. The most advanced systems even estimate how long the motor will last given the problem.

"All motors will eventually wear out, but it's extremely valuable if you can estimate if and when it will happen," said Corradi. "If a motor in an assembly line fails unexpectedly, it can bring an entire factory to a standstill. Creating algorithms to control the efficiency of a motor is very complex, but adding these diagnostic capabilities on top of efficiency controls requires a far more sophisticated algorithm, and a sophisticated device to execute that algorithm."

The most advanced control systems today include active, real-time safety features such as sensors that spot cracks in machinery and stop the machines in milliseconds to avoid catastrophic failure or conditions that could injure factory workers. Mallett notes that in many cases these advanced safety features are also becoming standardized and mandated by government. The IEC 61508 standards effort, for example, stipulates that control systems include functionality to detect potentially dangerous conditions and prevent hazardous events. In Europe, the

Machinery Directive 2006/42/EC went into effect in December 2009, mandating that machinery manufacturers provide products that comply with IEC 61508.

But motor control in the factory is not restricted to monitoring efficiency, safety and motor longevity, Corradi explained. Companies also leverage motor controls to make robots perform ever-more-sophisticated and exacting functions. “A single robot has several motors and they must be able to perform several functions,” said Corradi. “They must be programmable to change tasks and they must work in concert with other robots. The algorithms to run the motors in a robot are extremely advanced.”

Even an HVAC system motor control is vitally important. Many factories use chemicals, solvents and contaminants and must ventilate them properly. Clean rooms must also minimize contaminants. These machines must operate correctly all the time.

Mallett points out that in modern factories all these systems—assembly lines, robots

and HVAC—are interrelated and interconnected. They must be orchestrated in concert with one another. In many cases, newer systems use high-speed wired interconnect standards such as Industrial Ethernet, while some are starting to even use wireless communications. Since factories tend to be noisy environments, the latter trend adds further layers of complexity to motor control designs.

FPGAs Displacing Off-the-Shelf Processors in Advanced Motor Control

Because of all this complexity on the factory floor, the demand for processing performance is growing exponentially and programmable logic control is increasingly displacing traditional processor-based motor control.

Historically, engineers have used microcontrollers for factory-class motor control, said Corradi. But that design approach is hitting a wall. “The new factory systems are so sophisticated now and integrate so many functions that you would have to use several microcontrollers, and often the biggest

and most expensive ones,” he said. “Then, after you’ve done that, you are fairly restricted in what functionality you can add to your motor control. With an FPGA, you can add all the advanced features required to a single chip and even add new functions after you’ve deployed the system in a factory.”

A prime example of this elevated complexity is a new trend called “networked control systems,” in which motor control feedback is routed via networking to enterprise software systems, not just to a local operator’s station. In this application, “not only are the control algorithms more complex, but also, the network is essentially ‘in-the-loop,’ demanding real-time performance far beyond what standard off-the-shelf processors can handle,” Corradi said. “By contrast, FPGAs can perform networking and control support simultaneously” (see Figure 2).

“A single FPGA can really do the work of dozens of microcontrollers,” said Corradi. “It’s much easier to have this com-

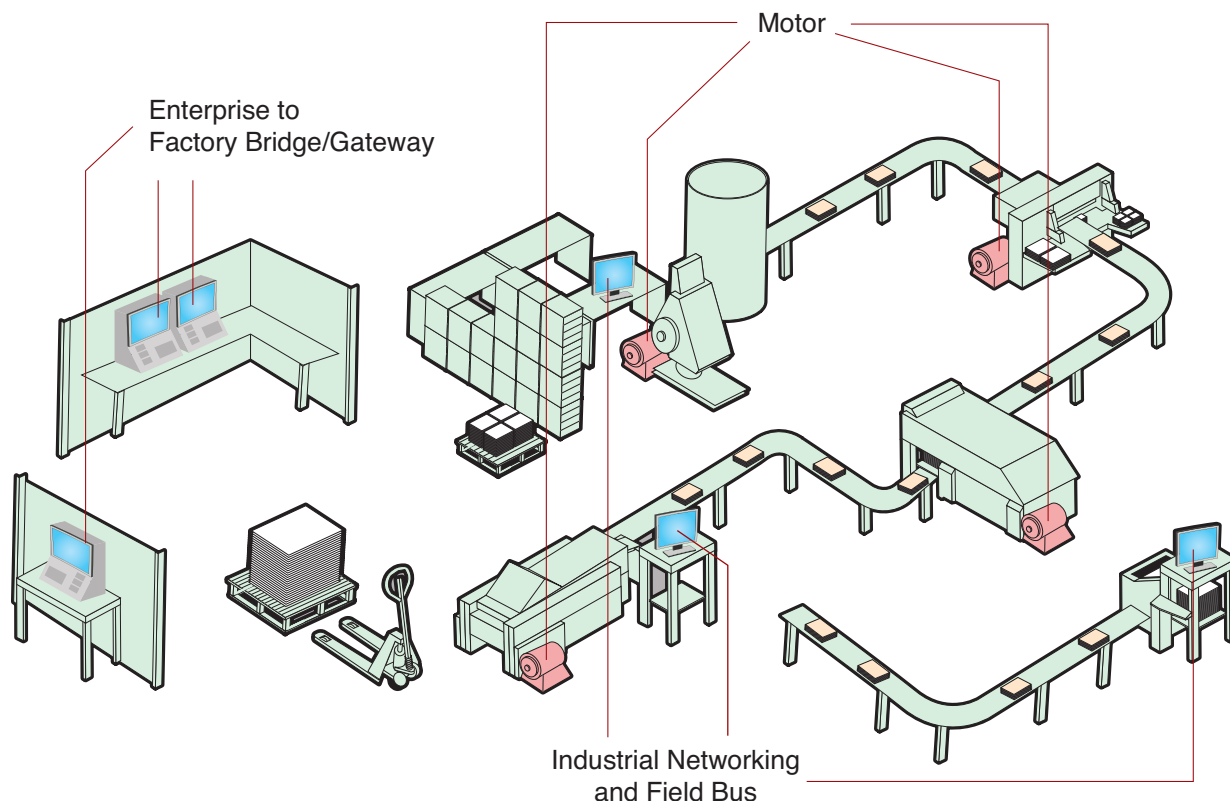


Figure 1 – Xilinx FPGAs play an increasing role in advanced motor control at multiple points in the factory and even the connection between the factory and the enterprise.

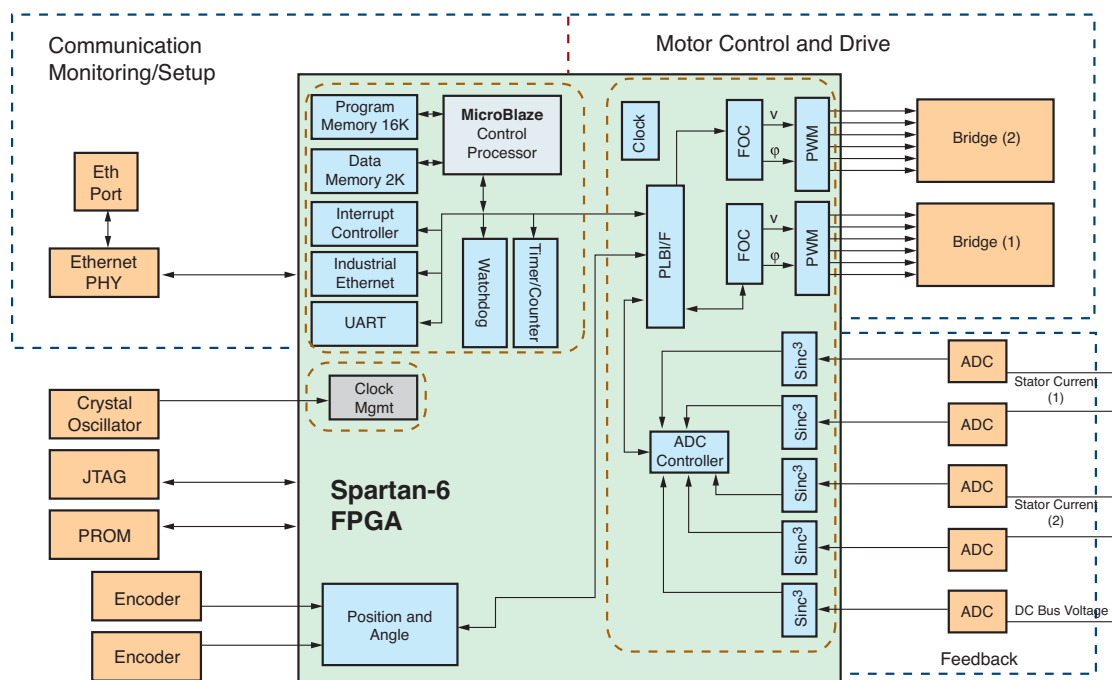


Figure 2 – Engineers can implement many motor control features on a single FPGA.

plexity in one device rather than distributed across a board with many processors.”

What’s more, FPGAs excel at parallel as well as serial operations. This means designers can program a single FPGA to coordinate multiple motor controller functions for a variety of motors simultaneously. The real-time requirements don’t stop there, but also include functional safety features, with the concomitant need to shut down a line in milliseconds if attached sensors detect unsafe conditions. “Functional safety is enabled within FPGAs by using silicon-specific functions, dedicated hardware IP and a software stack that, combined, make the complete system,” said Corradi.

Corradi noted that customers can leverage an FPGA’s reliability and the design techniques Xilinx has refined over 20 years for aerospace applications to implement safety features in their motor control systems. “Companies that add safety features to their motor control units typically have to design specific chips dedicated to safety,” said Corradi. “In an FPGA, you can partition a segment of the chip just for this purpose and have a clear separation between the safety and nonsafety segments of the chip. This makes it easier if they need to create derivative systems or upgrade systems, because they can simply leave the safety part of the chip alone.”

Another option would be to add duplicate or triplicate functions in the design to ensure space-quality safety controls in these systems.

In fact, the companies that specialize in the safety aspect of motor control have been the early adopters of FPGA technology. “They see the advantage in how the FPGA could clearly bring a reduction in development times and additional reliability, and a clear separation of safety and non-safety functions,” Corradi said.

Xilinx Readies Industrial Ethernet Motor Control Targeted Design Platform

It is for these reasons that companies are turning to FPGAs over processor-based systems for industrial motor control. Mallett predicts that the pace of adoption will advance rapidly over the next few years, especially once engineers get their hands on the upcoming Xilinx Targeted Design Platform tailored for factory automation and motor control.

This market-specific platform will pull all the pieces together for the end customer developing advanced motor controllers. The development environment will include the hardware, tools and IP needed to integrate Industrial Ethernet and motor control into a single platform. Corradi said

the Targeted Design Platform will also include a targeted reference design to help customers shorten their development cycles and differentiate their products.

Xilinx will base the first-generation system on a Spartan®-6 FPGA running a MicroBlaze® processor. Looking forward, one can envision a platform leveraging Xilinx’s ARM® MPU-based Extensible Processing Platform, further accelerating the role of programmable logic in motor control design—potentially saving factories millions of dollars and ensuring the safety of industrial workers worldwide.

A key part of the factory automation Targeted Design Platform is a library of general-purpose and market-specific IP that Xilinx and its alliance partners will offer to help customers develop Spartan-6-based integrated motor controllers quickly (see related story, page 12).

Mallett predicts that as companies become more familiar with programming FPGAs for motor control systems that end up in factories, they will start to use FPGA-based motor control across other product lines and for other control applications. “Companies are always seeking ways to build a better motor or at least a better way to control them, and FPGAs are ideal for the job,” said Mallett. ●●●

Revolutionizing Intelligent Drive Design with the Spartan-6

Xilinx Design Services uses the SP605 Evaluation Kit and state-of-the-art motor control IP to prototype an interface-independent intelligent drive control system.

by Kasper Feurer
Embedded Systems Engineer
Xilinx Ireland
kasper.feurer@xilinx.com

Richard Tobin
Embedded Systems Engineer
Xilinx Ireland
richard.tobin@xilinx.com

Manufacturers of intelligent drives, as well as many other players in the automotive and ISM sectors, are facing numerous challenges to satisfy new market demands and meet constantly evolving standards. In modern industrial and automotive applications, motors must provide maximum efficiency, low acoustic noise, a wide speed range and reliability, all at an acceptable cost. On today's factory floor, motor-driven equipment consumes two-thirds of total electricity energy, leading to the urgent need to develop more energy-efficient systems. Interoperability is also a critical design requirement, as in many cases a drive will serve as a component in a large-scale process. A key factor that influences this requirement is the breadth of industrial networking protocols (the field buses) and associated device profiles, which serve to standardize the representation of a drive within a network. The field buses themselves (for example, CAN and Profibus) are diverse, and despite sharing the same generic name they are not readily interchangeable. In an attempt to reduce cost and improve communications between industrial controllers, field bus

providers have developed Ethernet-based industrial networking solutions, and several new protocols, such as EtherCAT, Profinet and EtherNet I/P among others, have emerged in recent years. However, these are also divergent technologies and drive manufacturers struggle to support all the major players.

Xilinx Design Services (XDS) has addressed all of these issues in developing an FPGA-based prototype motor control platform supporting the CANopen and EtherCAT interfaces for a key player in the ISM space. Our role was to design and implement a fully functional and modular system fit for reuse in the customer's next-generation intelligent drives. The Xilinx® Spartan®-6 FPGA SP605 Evaluation Kit Base Targeted Design Platform, along with third-party IP, pro-

vided state-of-the art motor control algorithms and industrial networking support in a modular system architecture, yielding an efficient and scalable design.

Choosing the FPGA Path

The customer's existing, microcontroller-based solutions did not deliver what the customer wanted most: a scalable platform. A Spartan-6 FPGA-based intelligent drive control system provides all the necessary scalability, logic and compute power in a single chip, reducing costs while also avoiding obsolescence. Such a platform can be upgraded for years to come with the latest standards of industrial networking and the most efficient motor control algorithms. In addition, the reprogrammable nature of FPGAs facilitates customization of a single base motor control system to meet specific



customer requirements, allowing easy integration into the existing industrial network. In short, the Spartan-6 FPGA fulfills all the challenging requirements of the industrial space.

For newcomers to the world of FPGA-based system design, like our customer, Xilinx's Targeted Design Platforms offer the ideal starting point by providing a robust set of well-integrated and tested elements right out of the box. You can automate an even greater portion of the final design by adding domain-specific and market-specific platform offerings to the Base platform. These targeted reference designs reduce the learning curve by demonstrating FPGA implementations of real-world concepts, allowing customers to put their muscle into the design and development of the differentiating features of the final product.

Our solution combined the Spartan-6 SP605 Evaluation Kit with third-party offerings—namely the NetMot FMC board and IP from QDeSys plus industrial networking IP from Bosch and Beckhoff. Not only were all the basic building blocks of the desired system in place from the start, but we could proceed with prototype development without the need for a custom FPGA board, thus allowing the customer to verify the viability of this new platform at minimum cost. To further enhance the time-to-market and reduce the risks involved with a first-time FPGA system design, the customer asked us to not only deliver this prototype but also to support the adoption of FPGAs in its next-generation intelligent drives.

Ultimately, both engineers and their managers benefited by this approach. The former learned FPGA-based design faster, armed with best practices gleaned from XDS, while the management slashed the time it took to deliver product along with the business risks.

Intelligent Drive Control System Prototype

The XDS portfolio covers the entire FPGA design development cycle, from specification creation through coding, verification, timing closure and system integration. Drawing on years of experience in embedded-processor system and

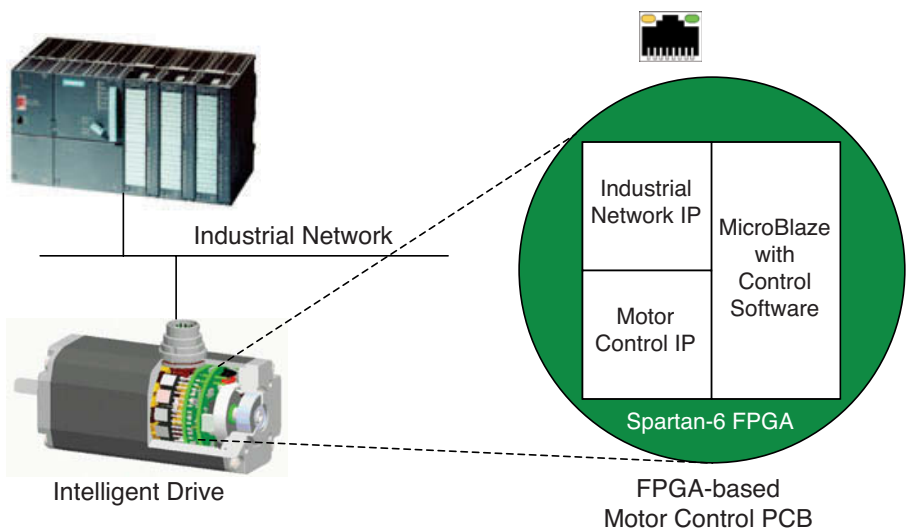


Figure 1 – The drive control system prototype

software application design, along with the ability to integrate third-party IP, good project management practices and a fully certified ISO9001 development process, XDS delivered the intelligent drive control system prototype very early in the customer's product development cycle. The resulting custom Targeted Design Platform enabled the customer's engineers to familiarize themselves with FPGA design processes and hence optimize the power of this new technology in their next-generation products.

Let's look more closely at the main components of this intelligent drive control system prototype as illustrated in Figure 1.

The programmable logic controller (PLC) operates the intelligent drive, which is attached to the industrial network in real time. For the purpose of this prototype, we used two PC-based PLCs to handle the two industrial networking standards the system supports: miControl mPLC for the controller-area network (CAN) and TwinCAT for the EtherCAT industrial Ethernet field bus system. The PLC generates predefined command messages (for example, start and stop) and verifies the correct behavior of the motor by analyzing the responses received (current speed, temperature, voltage and so on).

Depending on the combination of PLC and the type of intelligent drive (CAN or EtherCAT), the industrial network is either a serial bus or a standard 100-Mbit Ethernet interface. For both solutions the prototype uses a direct line link between the PLC and the motor—either a two-wire serial interface for CAN, or a standard RJ45 100Base-TX Ethernet connection for EtherCAT.

The motor control printed-circuit board, typically one of a number of PCBs in an intelligent drive, is dedicated to controlling the motor via commands from the PLC. This motor control board is where the flexibility of an FPGA comes into play. Rather than a single interface and single motor control algorithm solution, as in the conventional ASIC/microprocessor approach, the Spartan-6 FPGA can be reprogrammed with dedicated networking and motor control IP blocks, as well as the control software to suit the customer's specific needs. In this manner, a single FPGA-based PCB can fulfill the functions of many ASIC-based boards. At the same time, it future-proofs the intelligent drive by providing a mechanism to update the IP to the latest standards.

Rather than designing this motor control board from scratch, XDS exploited the

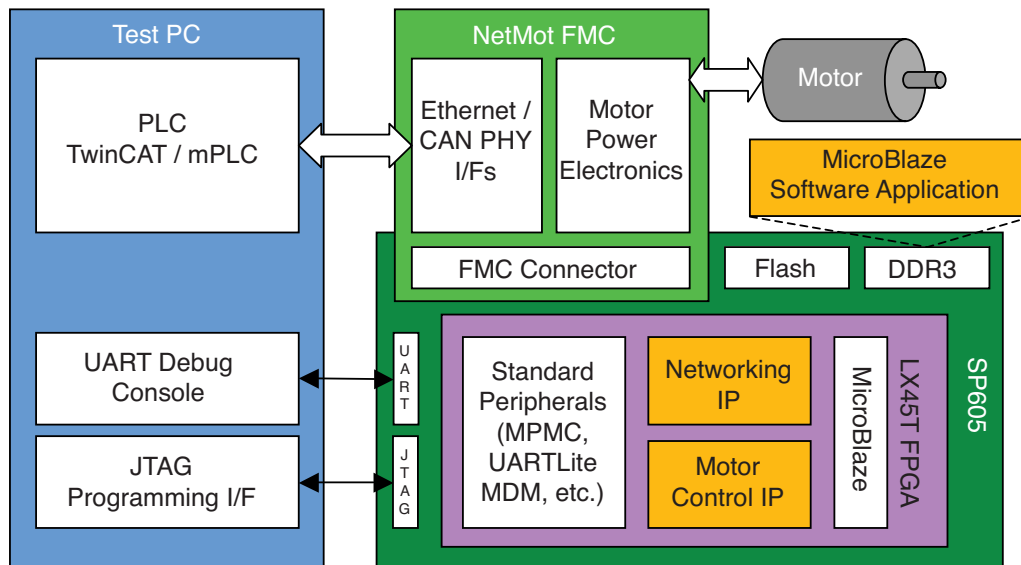


Figure 2 – Prototype Spartan-6 FPGA-based motor control board

Targeted Design Platform concept, integrating all the customer's desired elements by combining the Xilinx Spartan-6 SP605 Evaluation Kit, the NetMot FMC board, and industrial networking and motor control IP, thus delivering this proof-of-concept prototype before the customer had completed its new PCB. Figure 2 shows how we combined the various components to yield the prototype development platform. As a result, the customer greatly reduced the integration effort and was able to explore the best design options without re-engineering the final design.

The SP605 Base Targeted Design Platform is a general-purpose FPGA platform that hosts a Spartan-6 LX45T in a proven implementation, alongside many commonly needed peripherals such as DDR3 RAM, flash memories for program/bitstream storage, UART for debug and JTAG for FPGA programming. Another key element of the SP605 as well as all recent Xilinx boards is the FPGA Mezzanine Card (FMC) connector, which enables designers to expand the base board with custom functions and interfaces.

This feature of the SP605 enabled us to build on this base by using features provided by the QDeSys NetMot FMC (www.qdesys.com), which implements the

power electronics required for motor control, such as voltage inverters and ADCs for obtaining sensor data. You can connect the motor directly to these inputs/outputs as shown in Figure 2. The NetMot FMC also expands the industrial network connectivity features of the SP605 by adding two CAN and two Ethernet physical-layer interfaces. They are accessible to the FPGA via the FMC connector and a PLC over standard interfaces.

The test PC functions as both a host for the PLC software and an FPGA programming/debug platform using the UART and JTAG interfaces. In addition, we used this test PC to develop the MicroBlaze™ embedded-processor system for the SP605's LX45T FPGA with Xilinx's ISE® 12.1 Design Suite. This embedded system is responsible for processing the commands received from the PLC and controlling the motor accordingly.

The MicroBlaze software application, networking and motor control IP blocks seen in Figure 2 represent the modules of the design that change depending on the interface (EtherCAT or CANopen) and motor type chosen. One of the main challenges for XDS was to ensure that the process of switching between these options be as simple as possible, thus ensuring that

the customer would be able to reuse the same methods in the future for further industrial network types like Profinet and for new motors.

Implementation Details

Let's examine these parts of the Spartan-6 embedded system in greater detail. As shown in Figure 3, the motor control IP block that we used—the Xilinx Motor Control Library (XMCLIB)—is identical for both versions of the design. This custom IP core, which permits the FPGA to control the NetMot FMC's motor power electronics, plugs directly into Xilinx's Embedded Development Kit (EDK). This allowed us to add the IP to our embedded design from the Xilinx Platform Studio (XPS) project and configure it to suit the motor that was connected to the FPGA via the FMC connector. The XMCLIB software driver is a set of low-level functions giving the motor control application access to the XMCLIB register interface.

The networking IP, on the other hand, is where the system differs for the two variants. For the CAN version of the design we chose the standard LogiCORE™ IP XPS Controller Area Network, delivered with the ISE 12.1 design suite and licensed by Bosch GmbH. For the EtherCAT version, we used Beckhoff's EtherCAT Slave Controller IP

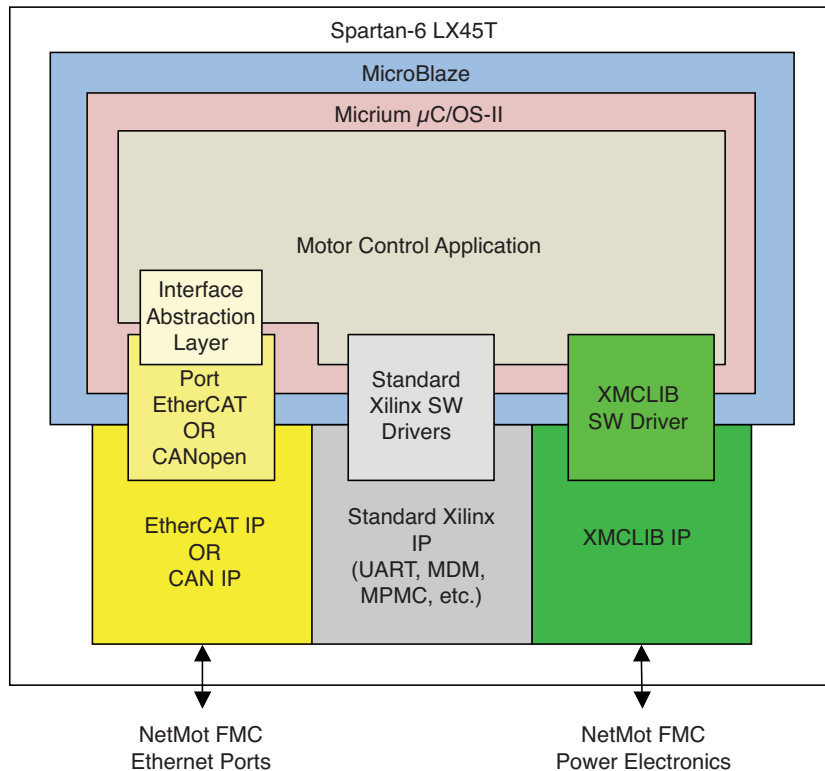


Figure 3 – CAN/EtherCAT embedded system

Core (www.beckhoff.com) for Xilinx FPGAs. Both of these IP cores are also available from the XPS tools' IP Catalog tab, making integration and configuration in the design very straightforward. In this case, instead of using a simple driver to provide access to the networking IP, we used Port's (www.port.de) CANopen and EtherCAT Stack solutions, which offer fully featured protocol implementations right out of the box.

Finally, we designed a custom embedded-software application to run on Micrium's (www.micrium.com) µC/OS-II on the MicroBlaze processor system. This embedded operating system enhances the prototype system's real-time capabilities and provides multitasking, message queues and semaphores, among other features.

We also recognized that it was important to structure the application in a way that would allow us to retarget it to different network interfaces. To achieve this, we designed an interface abstraction layer that lets us encapsulate the communications and motor control elements of the software.

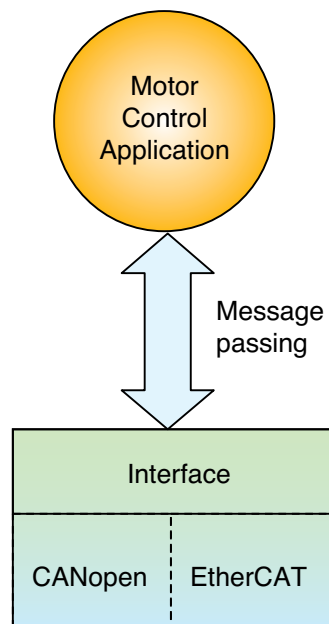


Figure 4 – Interface abstraction layer

On one side of this interface (Figure 4), we implemented a networking module (Port's CANopen or EtherCAT) to manage the communications for the networking IP available in the system. These modules plug in seamlessly to our interface abstraction layer. At the top level of these stacks, we pass communications and control data (such as PDOs, SDOs and NMT state transitions) into the abstraction layer, which interprets the data and presents it to the motor control application as commands such as start/stop or rotate at a specified velocity or to a specified position.

To determine a common set of messages and commands for the interface abstraction, we researched existing publications in the area of industrial networking and encountered the IEC 61800-7 standard. For the existing field bus technologies, several schemes are used to standardize the communications with a drive device (such as CiA-402 for CANopen or PROFIdrive for Profinet). IEC 61800-7 presents a common representation of a drive and proceeds to provide a set of mappings between this representation and the existing drive profiles.

The concepts presented in this standard allowed us to develop our interface abstraction, which in turn allowed us to encapsulate the networking component of the system. We can therefore change the networking interface present in the system, and we only need to tailor a small portion of the software to make it compatible with the existing motor control application.

Going Forward

The successful delivery of the intelligent drive control system prototype has clearly illustrated the potential power of FPGAs in industrial Ethernet networking, field buses and motor control. Although some work remains to develop a fully featured product, XDS tailored a Targeted Design Platform and enhanced it for the customer, creating a custom solution that will greatly reduce the risks and effort required to come to a final, engineered product. As a next step, XDS is investigating expanding this Targeted Design Platform to support the Profinet IP core and stack, demonstrating that the modular approach and the design practices adopted are very effective for the customer. 

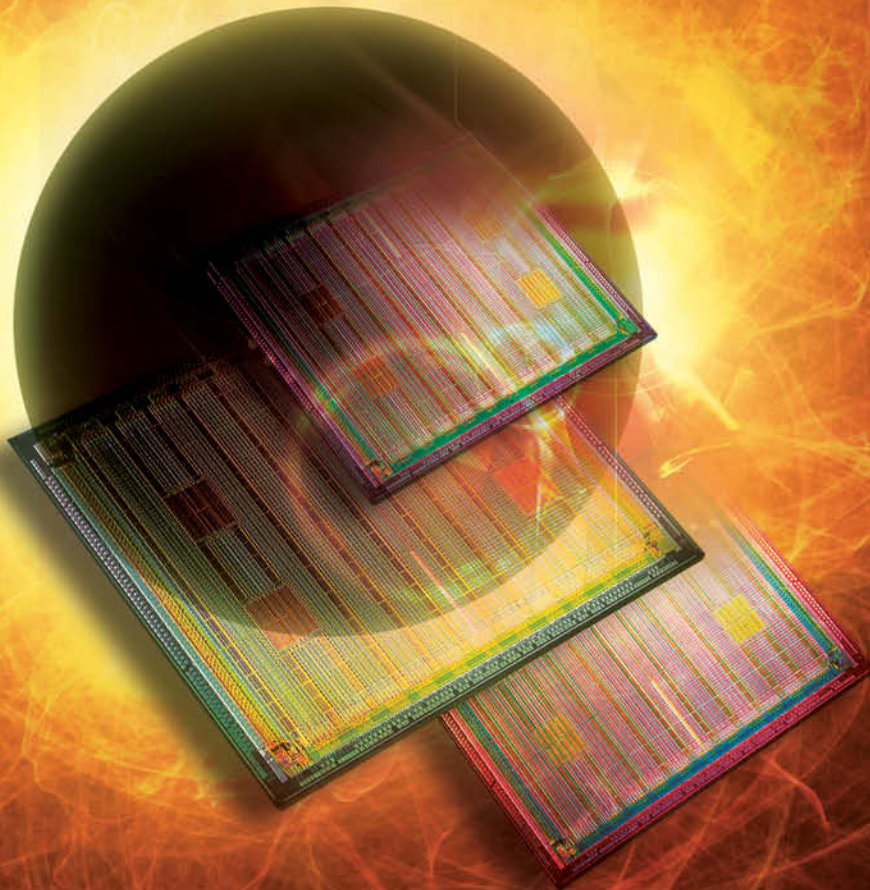
Using FPGAs in Mission-Critical Systems

SEU-resistant state machines hold the key to adapting programmable logic devices for high-reliability applications.

by Adam Peter Taylor
Principal Engineer
EADS Astrium
aptaylor@theiet.org

Dramatic surges in FPGA technology, device size and capabilities have over the last few years increased the number of potential applications that FPGAs can implement. Increasingly, these applications are in areas that demand high reliability, such as aerospace, automotive or medical. Such applications must function within a harsh operating environment, which can also affect the system performance. This demand for high reliability coupled with use in rugged environments often means you as the engineer must take additional care in the design and implementation of the state machines (as well as all accompanying logic) inside your FPGA to ensure they can function within the requirements.

One of the major causes of errors within state machines is single-event upsets caused by either a high-energy neutron or an alpha particle striking sensitive sections of the device silicon. SEUs can cause a bit to flip its state (0 -> 1 or 1 -> 0), resulting in an error in device functionality that could potentially lead to the loss of the system or even endanger life if incorrectly handled. Because these SEUs do not result in any permanent damage to the device itself, they are called soft errors.



The backbone of most FPGA design is the finite state machine, a design methodology that engineers use to implement control, data flow and algorithmic functions. When implementing state machines within FPGAs, designers will choose one of two styles, binary or “one hot,” although in many cases most engineers allow the synthesis tool to determine the final encoding scheme. Each implementation scheme presents its own challenges when designing reliable state machines for mission-critical

state machine does not enter when it is functioning normally. Designers must address these unused states to ensure that the state machine will gracefully recover in the event that it should accidentally enter an illegal state. There are two main methods of achieving this recovery. The first is to declare all 2^N number of states when defining the state machine signal and cover the unused states with the “others clause” at the end of the case statement. The others clause will typically set the outputs to a safe

startup following reset release. Typically, these states also keep the outputs in a safe state; should they be accidentally entered, the machine will cycle around to its idle state again.

One-hot state machines have one flip-flop for each state, but only the current state is set high at any one time. Corruption of the machine by having more than one flip-flop set high can result in unexpected outcomes. You can protect a one-hot machine from errors by

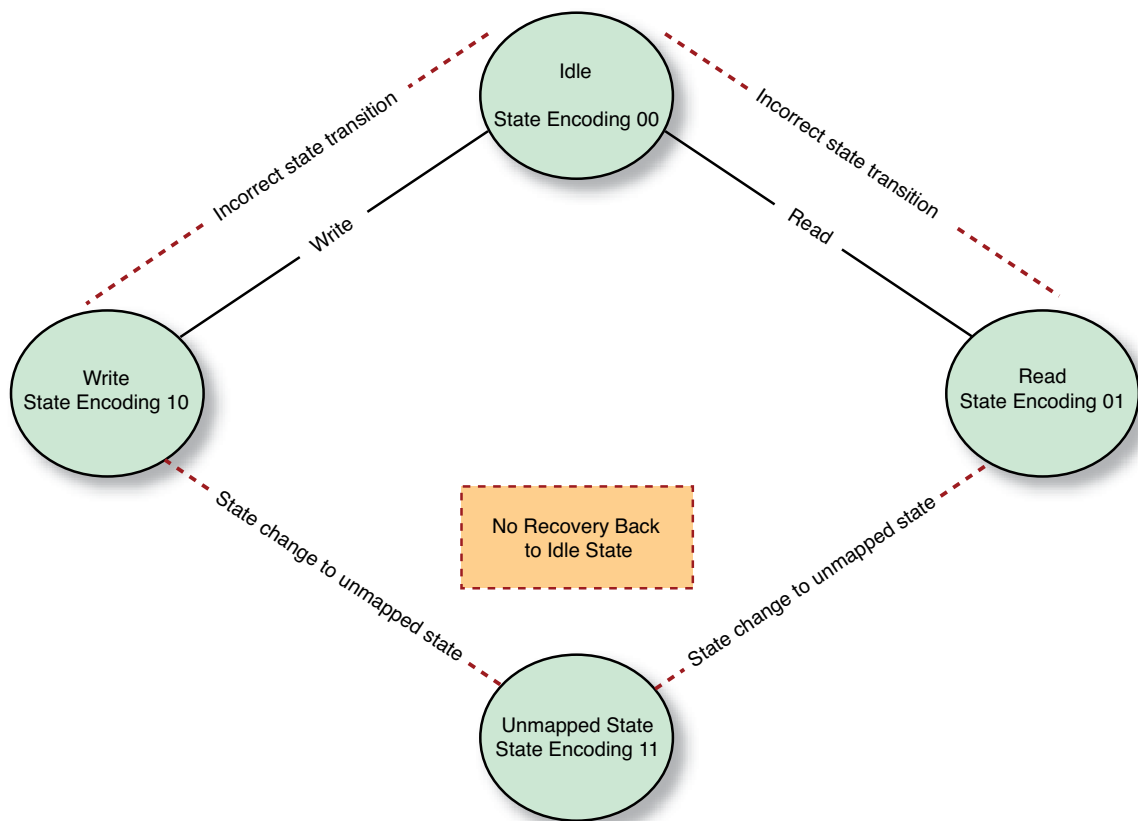


Figure 1 – Even a simple state machine can encounter several types of errors.

systems. Indeed, even a simple state machine can encounter several problems (Figure 1). You must pay close attention to the encoding scheme and in many cases take the decision about the final implementation encoding away from the synthesis tool.

Detection Schemes

Let's first look at binary implementations (sequential or “gray” encoding), which often have leftover, unused states that the

state and send the state machine back to its idle state or another state, as identified by the design engineer. This approach will require the use of synthesis constraints to prevent the synthesis tool from optimizing these unused states from the design, as there are no valid entry points. This typically means synthesis constraints within the body of the RTL code (“syn_keep”).

The second method of handling the unused states is to cycle through them at

monitoring the parity of the state registers. Should you detect a parity error, you can reset the machine to its idle state or to another predetermined state.

With both of these methods, the state machine's outputs go to safe states and the state machine restarts from its idle position. State machines that use these methods can be said to be “SEU detecting,” as they are capable of detecting and recovering from an SEU, although the state machines' oper-

ation will be interrupted. You must take care during synthesis to ensure that register replication does not result in registers with a high fanout being reproduced and hence left unaddressed by the detection scheme. Take care also to ensure that the error does not affect other state machines that this machine interacts with.

Many synthesis tools offer the option of implementing a “safe state machine” option. This option often includes more

logic to detect the state machine entering an illegal state and send it back to a legal one—normally the reset state. For a high-reliability application, design engineers can detect and verify these illegal state entries more easily by implementing any of the previously described methods. Using these approaches, the designers must also take into account what would happen should the detection logic suffer from an SEU. What effect would this have upon the reliability of the

design? Figure 2 is a flow chart that attempts to map out the decision process for creating reliable state machines.

Correction Schemes

The techniques presented so far detect or prevent an incorrect change from one legal state to another legal state. Depending upon the end application, this could result in anything from a momentary system error to the complete loss of the mission.

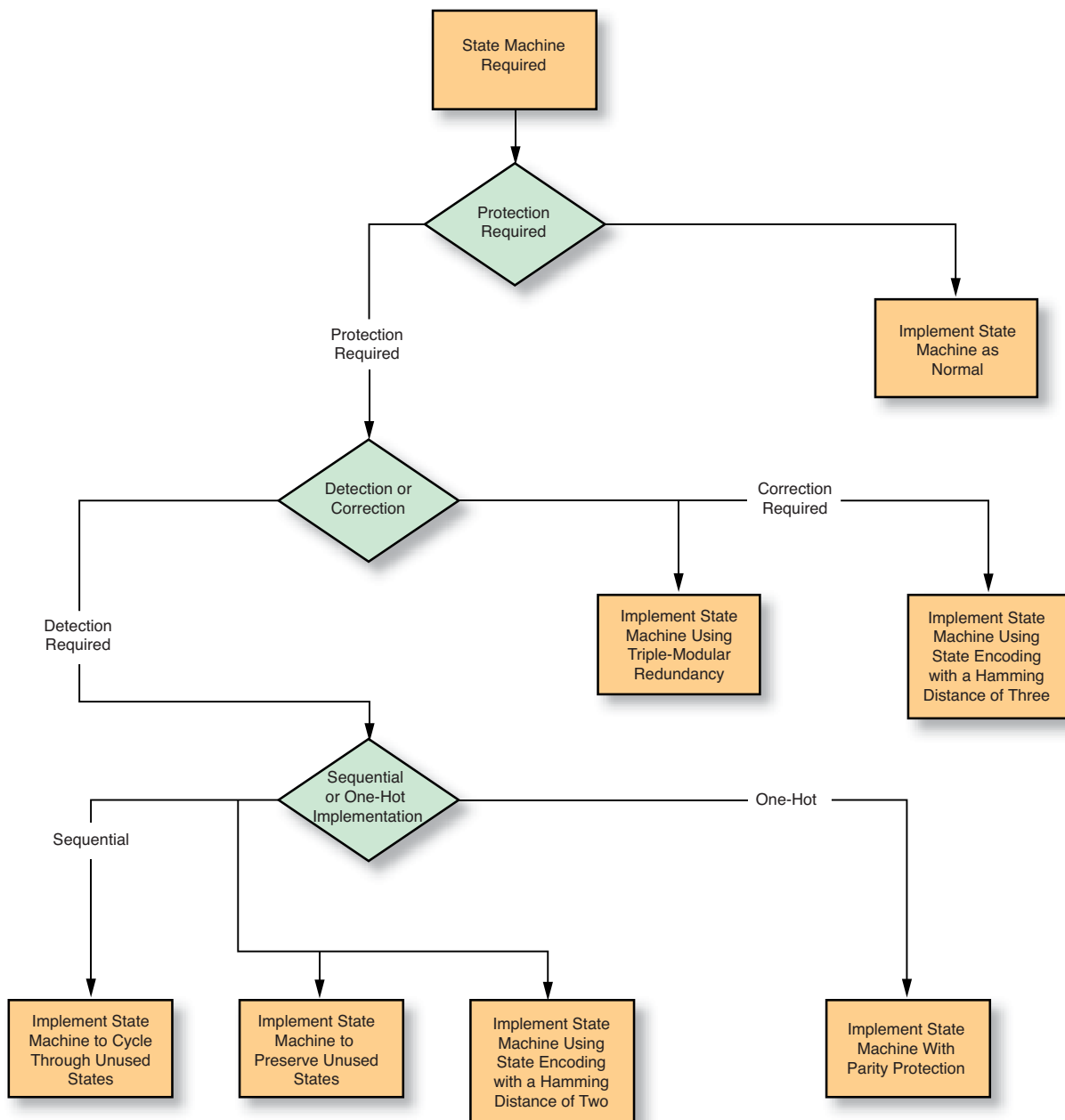


Figure 2 – This flow chart maps out the decision process for creating reliable state machines.

Techniques for detecting and fixing incorrect legal transitions are triple-modular redundancy and Hamming encoding. The latter provides a Hamming distance of three and covers all possible adjacent states. A simpler technique for preventing legal transitions is the use of Hamming encoding with a Hamming distance of two (rather than three) between the states, and not covering the adjacent states. This, however, will increase the number of registers your design will require.

Triple-modular redundancy, for its part, involves implementing three instantiations of the state machine with majority voting upon the outputs and the next state. This is the simpler of the two approaches and many engineers have used it in a number of applications over the years. Typically, a TMR implementation will require spatial separation of the logic within the FPGA to ensure that an SEU does not corrupt more than one of the three instantiations. It is also important to remove any registers from the voter logic, since they can create a single-point failure in which an SEU could affect all three machines.

The use of a state machine with encoding that provides a Hamming distance of three between states will ensure both SEU detection and correction. This guarantees that more than a single bit will change between states, meaning an SEU cannot make the state transition from one legal state to another erroneously. The use of a Hamming distance of two between states is similar to the implementation for the sequential machine, where the unused states are addressed by the “when others” clause or reset cycling. However, as the states are explicitly declared to be separate from each other by a Hamming distance of two within the RTL, the state machine cannot move from one legal state to another erroneously and will instead go to its idle state should the machine enter an illegal state. This provides a more robust implementation than the binary one mentioned above.

If you wish to implement a state machine that will continue to function correctly should an SEU corrupt the current state register, you can do so by using a Hamming-code implementation with a Hamming

distance of three and ensuring its adjacent states are also addressed within the state machine. Adjacent states are those which are one bit different from the state register and hence achievable should an SEU occur. This use of states adjacent to the valid state to correct for the error will result in $N*(M+1)$ states, where N is the number of states and M is the number of bits within the state register. It's possible to make a small high-reliability state machine using this technique, but crafting a large one can be so complicated as to be prohibitive. The extra logic footprint associated with this approach could also result in lower timing performance.

Deadlock and Other Issues

There are other issues to consider when designing a high-reliability state machine beyond the state encoding schemes. Deadlock can occur when the state machine enters a state from which it is never able to leave. An example would be one state machine awaiting an input from a second state machine that has entered an illegal state and hence been reset to idle without generating the needed signal. To avoid deadlock, it is therefore good practice to provide timeout counters on critical state machines. Wherever possible, these counters should not be included inside the state machine but placed within a separate process that outputs a pulse when it reaches its terminal count. Be sure to write these counters in such a way as to make them reliable.

When checking that a counter has reached its terminal count, it is preferable to use the greater-than-or-equal-to operator, as opposed to just the equal-to operator. This is to prevent an SEU from occurring near the terminal count and hence no output being generated. You should declare integer counters to a power of two and, if you are using VHDL, they should also be modulo the power of two to ensure in simulation that they will wrap around as they will in the final implementation [count <= (count + 1) Mod 16; for a 0- to 15-bit integer counter]. Unsigned counters do not require this, since there is no simulation mismatch between RTL and post-route simulation regarding wraparound.

You can replicate data paths within the design and compare outputs on a cycle-by-cycle basis to detect whether one of them has been subjected to an SEU event. Wherever possible, edge-detect signals to enable the design to cope with inputs that are stuck high or stuck low. You should analyze each module within the design at design time to determine how it will react to stuck-high or stuck-low inputs. This will ensure it is possible to detect these errors and that they cannot have an adverse effect upon the function of the module.

Simulation and Verification

Once you have designed the state machine, you will of course wish to simulate it to ensure it meets requirements. It is also possible during simulation to try to test how the machine will behave when subjected to an SEU (it is worth noting that this can take considerable time and effort to achieve in full). You can corrupt the state machine using the force option within the ModelSim utilities library from a test bench. When attempting to simulate the effects of SEUs, it is important to remember that they are asynchronous events that can occur at any point in time. There are also third-party tools available that can test designs for SEU performance. One free offering, which injects SEUs into the design, is the ESA Single Event Upsets Simulation Tool, available at <http://www.nebrija.es/~jmaestro/esa/>.

Designers can undertake a similar approach upon the post-route simulation. However, it's a better idea to verify the correct performance of the RTL code with regard to functionality and SEU tolerance and then conduct a formal equivalence check between the RTL and the post-route simulation netlist to ensure that they both behave in the same manner.

FPGAs will continue to be used in an increasing number of mission-critical applications as the growth in capability and performance demanded of these systems increases. The techniques presented here will provide you with the methodology required to make such a design a success. ●●●

Virtex-4 FPGA Forms Foundation for Secure GSM Standards

Here are some design mechanisms and tips you can try when implementing current and future A5/x algorithms using Xilinx FPGAs.

by Mansoor Naseer
Assistant Professor
Bahria University, E-8, Islamabad, Pakistan
mansoor.naseer@gmail.com

Waiser Mehmood
Student
Bahria University, E-8, Islamabad, Pakistan
waisermehmood@gmail.com

Dr. Abid Ali Minhas
Associate Professor
Bahria University, E-8, Islamabad, Pakistan
abid.research@gmail.com

Mobile telephony, the Internet and streaming applications enable people to communicate over long distances and find information and entertainment online. But with the explosion in mobile communications usage, the tremendous volume of information transmitted in the air is vulnerable to security risks. In order to maintain security, engineers use cryptographic techniques to disguise the data. For the Global System for Mobile communication (GSM) standard, their algorithm of choice is the A5/1. Using A5/1, the encryption and decryption are performed over the network, both at the handset and at the base station.

This article will walk you through an implementation of the A5/1 stream cipher, taking note of the code snippets used to implement different blocks of the cipher. We will also take a look at a couple of new and different twists on the original A5/1 algorithm for enhanced security. In particular, I am changing how the feedback function behaves as well as reworking the combiner output. For this work, I used a Xilinx® Virtex® 4 LX series FPGA for hardware implementation and the Xilinx ISE® for synthesis. I used Mentor Graphics' ModelSim 6.0 for simulation.

Before plunging into implementation details, let us begin with a brief description of the A5/1 algorithm.

Ins and Outs of A5/1 Algorithm

In the GSM standard, information travels over the airwaves as sequences of frames. Frame sequences contain a digitized version of the user's voice as well as streaming audio and video data used in mobile Internet browsing. The system transmits each frame sequentially, 4.6 milliseconds apart. A frame consists of 228 bits; the first 114 bits are called "uplink data" and the next 114 bits are known as "downlink data."

We initialize A5/1 using a 64-bit secret key together with a known frame number that is 22 bits wide. The next step is to feed the input bits of the frame and secret key into three linear feedback shift registers (LFSRs) that are 19, 22 and 23 bits in size respectively. The feedback is provided after XOR'ing of selected taps, which helps increase randomness in the final output. I will call the three LFSRs R0, R1 and R2, such that each LFSR follows a primitive function. R0, R1 and R2 respectively use $x^{19}+x^5+x^2+x+1$, $x^{22}+x+1$ and $x^{23}+x^{15}+x^2+x+1$ as their primitive functions.

The clocking sequences of LFSRs use a particular mechanism; otherwise, it would be very easy to decipher the output of LFSRs. Therefore, clocking is based on a majority rule. This rule uses one tap each from R0, R1 and R2, and determines whether the taps are mainly 0 or 1. That is, two or more taps with a value of 0 imply a majority value 0; otherwise, it is 1. Once the majority is established, clock the LFSRs if their input bit matches the majority bit. If the input bit does not match the majority value, no input bit will be inserted in the LFSR and hence, no shifting will take place. Additionally, we get the output by XOR'ing each output bit of R0, R1 and R2 in the original A5/1. In this work, I have replaced the standard XOR'ing with a more sophisticated output-generating function that we call a "combiner."

The key setup routine of the A5/1 algorithm is as follows:

1. Initialize R0, R1 and R2 with 0.
2. Load the value of session key KC and frame Fn into the LFSRs, one after the

other. The session key KC takes 64 cycles in loading whereas Fn takes another 22 cycles. In every cycle, the input bit is XOR'ed with a linear feedback value coming from selective taps.

3. Next, allow the LFSRs to clock for 100 cycles, a process called mixing. Linear feedback is active and so are the chip enables, allowing a shift/no-shift that is also active.
4. After the mixing step is complete, the next 228 cycles result in the final output key stream. The uplink and downlink streams (each consisting of 114 bits) are used for decryption and encryption operations. This step employs nonlinear feedback and a modified combiner as against the traditional A5/1 algorithm defined above.
5. You can repeat the previous steps for every new frame Fn. The frame numbers are updated where the session key remains unchanged until authentication protocols provoke it.

Hardware Implementation

I did the hardware implementation of the proposed algorithm on Xilinx Virtex-4 series FPGAs. Because we designed the implementation logic for compactness, I targeted the smallest device in the Virtex-4 family, namely the XC4VLX15. Figure 1 shows the overall architecture of the top-level module.

The top level comprises two building blocks and glue logic. I designed the glue logic to provide chip connectivity, start logic and data-loading features. There are two other building blocks, Majority/Tap rule and LFSR bank, each with its own unique functions.

The Majority/Tap rule block receives the selective tap positions from the LFSR bank and performs two separate functions independently. As the selection criteria for Majority or Tap rule are based on the last tap of R0, R1 and R2, the logic for Majority rule and Tap rule is implemented in parallel. I also performed one other optimization in the implementation of Majority rule. The pseudo algorithm for selecting majorities is described as:

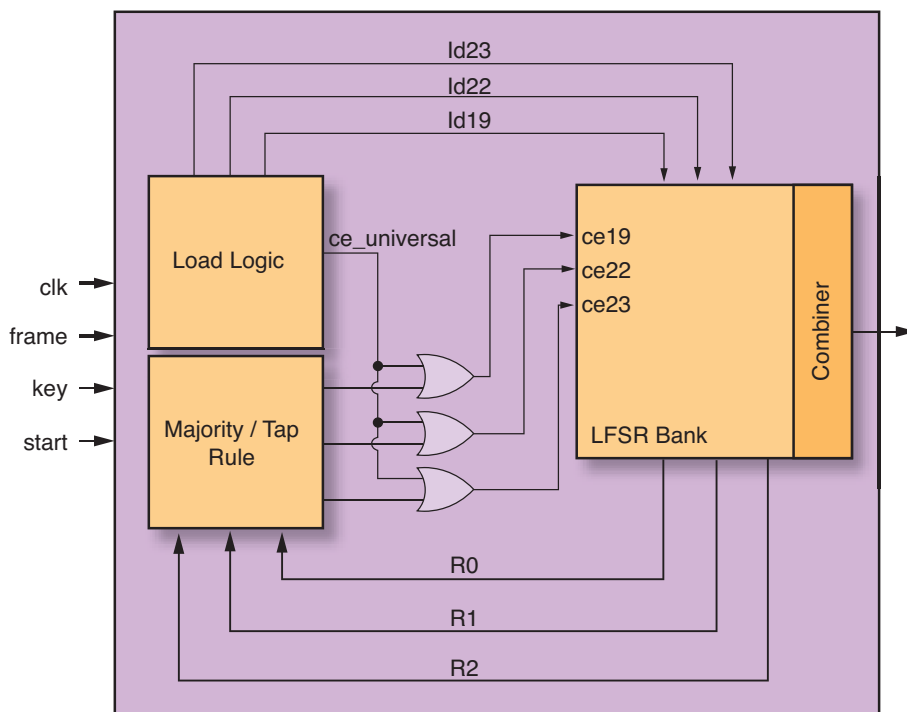


Figure 1 – Architectural diagram of the top module

```

If (R0[13] + R1[12] + R2[16])
>= 2 then maj1 = 1
else then maj1 = 0

```

Similarly, we find the value of maj2 based on tap numbers 7, 5 and 8. Based on these majority functions, we define the chip enable for R0 as:

```

If (R0[13]== maj1) AND (R0[7] ==
maj2) then chip enable for Majority
rule is 1 else 0.

```

I implemented the pseudo code for selecting maj1 and maj2 in the form of simple and/or gates. Similarly, the pseudo code for chip enables maps onto a simple XNOR function, combining all the logic and reducing area.

The technique implements the Tap rule algorithm using case statements. The original A5/1 algorithm determines the shift operation or chip enable for LFSRs based on the majority rule only. As I have introduced a modified Majority rule and an additional Tap rule for shifting LFSRs, I needed another mechanism. Thus, make the final selection of chip enables for R0,

R1 and R2 based on the PoliSel signal, which determines whether to output the chip enable signal established by Majority or Tap rule. The PoliSel signal implements this equation: PoliSel = R0[18] XOR R1[21] XOR R2[22].

The LFSR Bank

The LFSR bank module houses multiple functions and implementations. This module instantiates a core LFSR with varying parameters for establishing R0, R1 and R2 LFSRs. The width of these LFSRs is 19, 22 and 23 respectively. This module also instantiates an improved linear feedback function from A5/1. Additionally, the LFSR bank implements our novel nonlinear feedback logic. I also implemented in this block the combiner that produces the final output. Figure 2 shows the block diagram for the LFSR bank. Here, the LFSRs are serial-in, parallel-out shift registers. Xilinx Virtex-4 devices allow convenient mapping of these shift registers onto hardware. Using the standard

template for LFSRs as shown below, we can infer generic serial registers.

```

module lsr (clk, ce, si, po);
parameter    width = 18;
input        clk, ce, si;
output [width-1:0] po;
reg [width-1:0] shftreg = 0;
always @ (posedge clk)
begin
    if (ce) shftreg <=
        {shftreg[width-2:0], si};
end
assign po = shftreg;
endmodule

```

With the use of parameters, it becomes easy to instantiate LFSRs at the higher level. By passing the parameters using the “defparam” feature of Verilog, we can infer the required LFSRs of width 19, 22 and 23 bits at the top level. The inference is conveniently done by using the following lines of code:

```

lsr lfsr19bit (clk, ce19, si19,
lfsr19);
defparam lfsr19bit.width = 19;

```

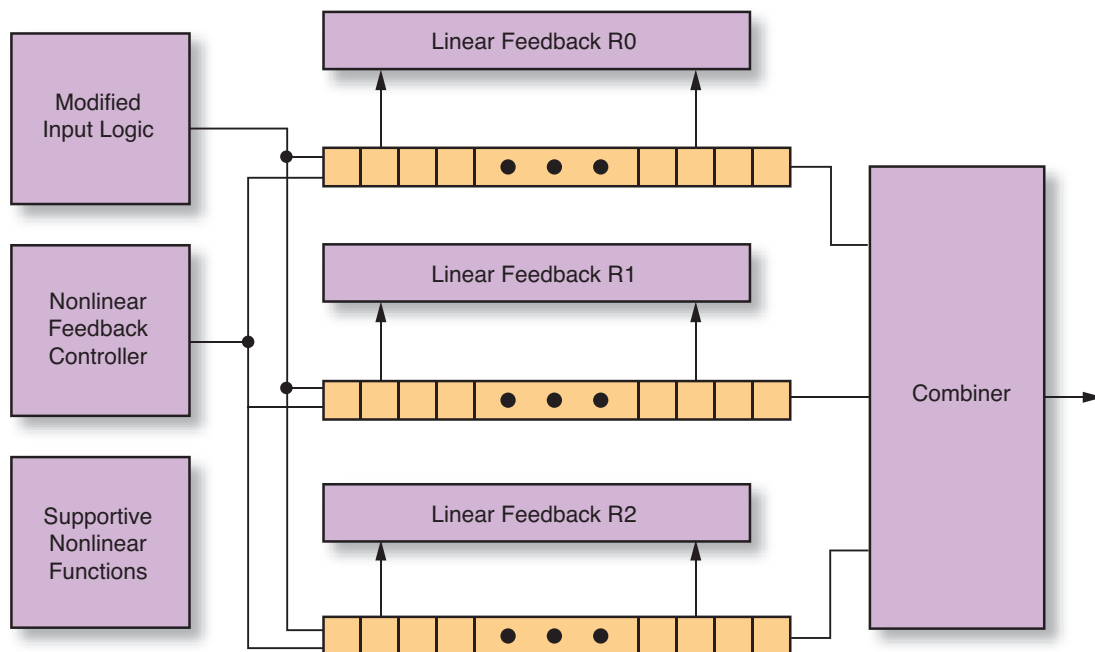


Figure 2 – Logic blocks at the input and output of R0, R1 and R2 in LFSR bank module

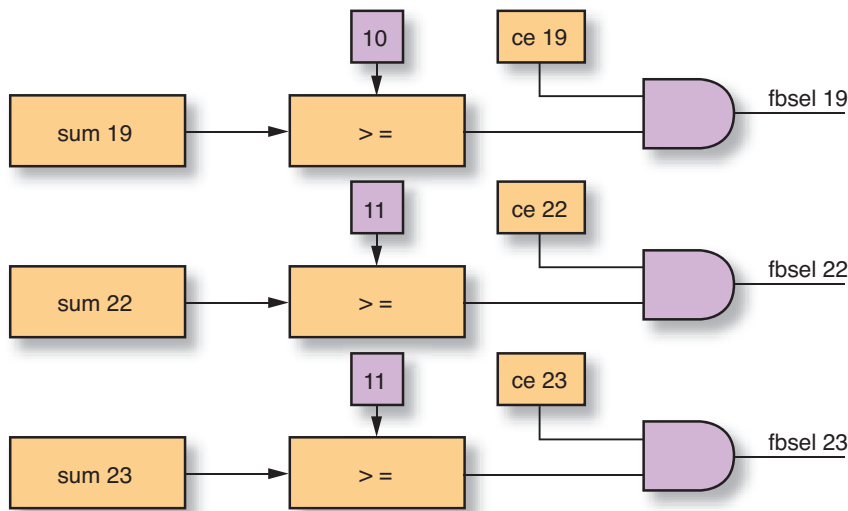


Figure 3 – Nonlinear feedback controller and input selection values

(The latter controls the parameter width inside instance lsr.)

As seen from the following code snippet, the input loading path of the LFSRs is well-defined in terms of simple XOR operations.

```
wire frameXor19, keyXor19, si19lfb;
assign frameXor19 = frame ^ si19lfb;
assign keyXor19 = key ^ si19lfb;
assign si19input = (frameXor19 ^
keyXor19) ? frameXor19:keyXor19;
```

The existence of asynchronous gates increases the length of the critical path. However, here it is well within tolerable limits, as the critical path in the current case consists of only three XORs and one multiplexer.

After loading is completed, the feedback path chooses between either linear or nonlinear feedback. The nonlinear feedback controller makes this selection decision. A choice of nonlinear feedback suddenly changes the design specifications and parameters. For instance, the nonlinear feedback function is activated periodically every 11 clock cycles and remains active for 10 clock cycles. To keep the implementation area small, we have implemented counters rather than adders to enforce the periodic behavior of the nonlinear feedback function. The implementation requires separate counters for

R0, R1 and R2 for counting the number of 1's. The pseudo algorithm for this implementation is as follows:

1. Detect falling edge on ldlfsr signal.
2. Start counting for 100 cycles.
3. After 100 cycles, for every 10 cycles activate the stream breaker, then deactivate for 10 cycles until 114 clocks elapse.

Stream breaker pseudo code:

4. Sum the contents of lfsr 19, 22, 23.
5. If count19 > 10, count22 > 11 or count23 > 11
6. pass 1 into the LFSR
7. else
8. pass 0 into the LFSR.

Since the decision of inserting 1's or 0's in the input of R0, R1 and R2 needs to be taken every clock cycle, once the nonlinear feedback is activated, the number of counters and adders significantly increases in area and critical path.

The combination of chip enable and input causes the nonlinear feedback path to assume 0 or 1 based on the algorithm of the stream breaker. Figure 3 shows a hardware building block implementing this scheme.

Using counters with defined thresholds assists the implementation of the

nonlinear feedback controller. Do not allow the counter to decrement after approaching 0 or to increment after reaching the high value corresponding to R0, R1 and R2, namely 19, 22 and 23 respectively. The following code snippet provides one example implementation of the counter:

```
reg [4:0]    sum19=5'd0;
wire        inc_sum19, dec_sum19;
assign      inc_sum19 = si19 & ce19;
assign dec_sum19 = lfsr19[18] & ce19;
always @ (posedge clk)
begin
    case ({inc_sum19, dec_sum19})
        2'b01:
begin
    if (sum19 == 5'd0) sum19<=5'd0;
    else
        sum19<=sum19-1;
end
2'b10:
begin
    if (sum19 == 5'd18) sum19<=5'd18;
    else
        sum19<=sum19+1;
end
default: sum19 <= sum19;
endcase
end
```

This code also shows how the increment and decrement functions are dependent on chip enable signals and not just the value at the input of LFSR on every rising edge of the clock.

It is extremely important to implement tight and secure encryption algorithms for future mobile communication standards. As the bandwidth requirements are rapidly evolving and streaming-data volume is on an exponential rise, it is all the more necessary to be able to implement vary fast and parallel ciphers for data security. A Xilinx Virtex-4 device offers the perfect solution for parallelism, high speed and bandwidth requirements in this application. With low-voltage operations and ultralow power consumption, this platform can easily adapt to the upcoming secure mobile applications. With the small tricks and snippets of code presented here, you can easily map the A5/1 and its future variants onto Xilinx FPGAs. 🌈

FPGA Research Design Platform Fuels Network Advances

Xilinx and Stanford University are teaming up to create an FPGA-based reference board and open-source IP repository to seed innovation in the networking space.

by Michaela Blott
Senior Research Engineer
Xilinx, Inc.
mblott@xilinx.com

Jonathan Ellithorpe
PhD Candidate
Stanford University

Nick McKeown
Professor of Electrical Engineering and Computer Science
Stanford University

Kees Vissers
Distinguished Engineer
Xilinx, Inc.

Hongyi Zeng
PhD Candidate
Stanford University

Stanford University, together with Xilinx Research Labs, is building a second-generation high-speed networking design platform called the NetFPGA-10G specifically for the research community. The new platform, which is poised for completion this year, uses state-of-the-art technology to help researchers quickly build fast, complex prototypes that will solve the next crop of technological problems in the networking domain. As with the first-generation platform, which universities worldwide have eagerly adopted, we hope the new platform will spawn an open-source community that contributes and shares intellectual property, thus accelerating innovation.

An open-source hardware repository facilitates the sharing of software, IP and design experiences, promoting technological solutions to the next generation of networking problems.

This basic platform will provide everything necessary to get end users off the ground faster, while the open-source community will allow researchers to leverage each other's work. The combination effectively reduces the time spent on the actual implementation of ideas to a minimum, allowing designers to focus their efforts on the creative aspects.

In 2007, we designed the first-generation board, dubbed NetFPGA-1G, around a Xilinx® Virtex®-II Pro 50, primarily to teach engineering and computer science students about networking hardware. Many EE and CS graduates go on to develop networking products, and we wanted to give them hands-on experience building hardware that runs at line rate, uses an industry-standard design flow and can be placed into an operational network. For these purposes, the original board had to be low in cost. And with generous donations from several semiconductor suppliers, we were able to bring the design in at an end price of under \$500. As a result, universities were quick to adopt the board and today about 2,000 NetFPGA-1Gs are in use at 150 schools worldwide.

But the NetFPGA very quickly became more than a teaching vehicle. Increasingly, the research community began to use it too, for experiments and prototypes. For this purpose, the NetFPGA team provided free, open-source reference designs and maintains a repository of about 50 contributed designs. We support new users, run online forums and offer tutorials, summer camps and developers' workshops.

Trend Toward Programmability

For more than a decade, networking technology has trended toward more-programmable forwarding paths in switches, routers and other products. This is largely because

networking hardware has become more complicated, thanks to the advent of more tunneling formats, quality-of-service schemes, firewall filters, encryption techniques and so on. Coupled with quickly changing standards, those factors have made it desirable to build in programmability, for example using NPUs or FPGAs.

Researchers often want to change some or all of the forwarding pipeline. Recently, there has been lots of interest in entirely new forwarding models, such as OpenFlow. Researchers can try out new network architectures at the national scale at national testbeds like GENI in the United States (<http://geni.net>) and FIRE in the EU (http://cordis.europa.eu/fp7/ict/fire/calls_en.html).

Increasingly, researchers are embracing the NetFPGA board as a way to prototype new ideas—new forwarding paradigms, scheduling and lookup algorithms, and new deep-packet inspectors—in hardware. One of the most popular reference designs in the NetFPGA canon is a fully functional open-source OpenFlow switch, allowing researchers to play with variations on the standard. Another popular reference design accelerates the built-in routing functions of the host computer by mirroring the kernel routing table in hardware and forwarding all packets at line rate.

NetFPGA, Take Two

For the second-generation platform, the so-called NetFPGA-10G, we have expanded our original design goals to also include ease of use, with the aim of supplying end customers with a basic infrastructure that will simplify their design experience. This goal is closely aligned with the objective of Xilinx's mainstream Targeted Design Platforms, which provide users with tools, IP and reference

designs in addition to FPGA silicon so as to speed up the design process.

To realize this vision, we will deliver a board with matching FPGA infrastructure design in the form of both basic and domain-specific IP building blocks to increase ease of use and accelerate development time. We further will develop reference designs, such as a network interface card and an IPv4 reference router, as well as basic infrastructure that assists with building, simulating and debugging designs. The idea is to allow users to truly focus their development time on their particular area of expertise or interest without having to worry about low-level hardware details.

Unlike the mainstream Targeted Design Platforms, our networking platform targets a different end-user group, namely, the larger research community, both academic and commercial. Semiconductor partners are heavily subsidizing this project to assist in the effort to keep the board cost to an absolute minimum so as to encourage broad uptake. Not only Xilinx but other key component manufacturers such as Micron, Cypress Semiconductor and NetLogic Microsystems are donating parts for the academic end user. (Commercial researchers will pay a higher price.)

Part of the project's strength is the fact that this platform is accompanied by a community as well as an open-source hardware repository that facilitates the sharing of software, IP and experiences beyond the initial base package. The result is an ever-growing IP library that we hope will eventually encompass a wide range of reference components, networking IP, software and sophisticated infrastructure thanks to contributions from many well-known universities, research groups and companies. We hope that by providing a carefully designed framework, some lightweight coordination to share expertise and IP

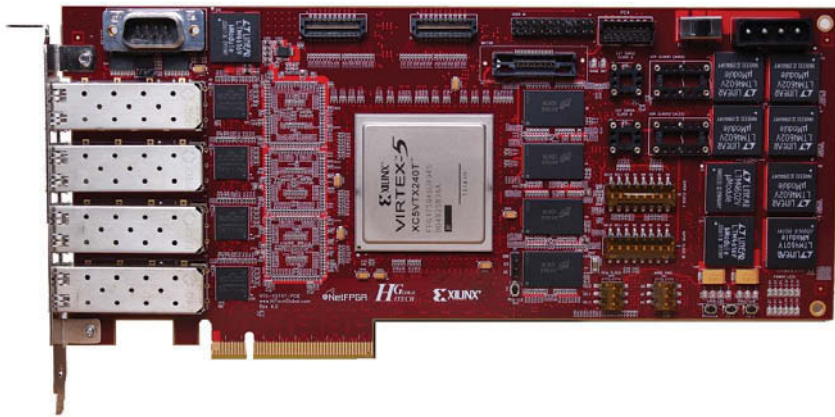


Figure 1 – The NetFPGA-10G board is built around a Virtex-5 FPGA.

in a systematic way, and a well-designed plug-and-play architecture with standardized interfaces, the open-source hardware repository will grow, promoting technological solutions to the next generation of network problems.

The NetFPGA-10G is a 40-Gbit/second PCI Express® adapter card with a large FPGA fabric that could support as many

through which the network traffic enters the FPGA. The memory subsystem, for its part, consists of several QDRII and RLLDRAMII components. The majority of the I/Os are used for this interface, to maximize the available off-chip bandwidth for functionality such as routing tables or packet buffering. The FPGA also interfaces to the PCIe subsystem.

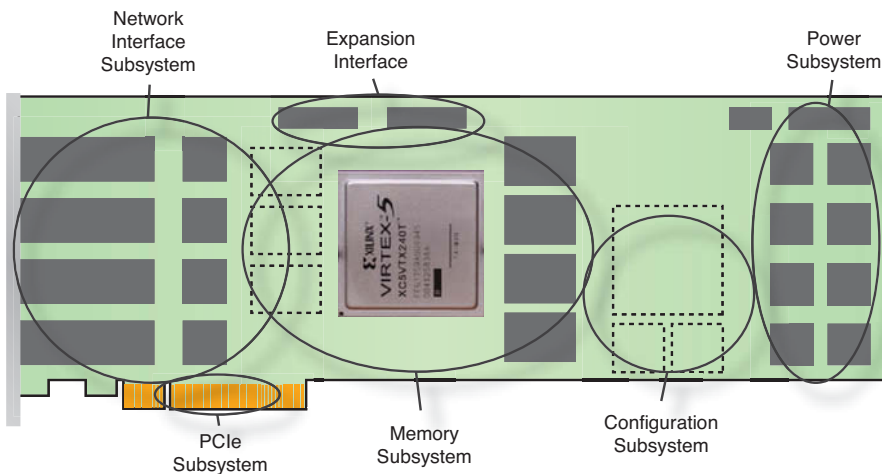


Figure 2 – The board connects to five subsystems: network interface, PCIe, expansion, memory, configuration and power.

applications as possible. As shown in Figure 1, the board design revolves around a large-scale Xilinx FPGA, namely a Virtex-5 XC5VTX240T-2 device [1].

The FPGA interfaces to five subsystems (see Figure 2). The first of these, the network interface subsystem, hosts four 10-Gbps Ethernet interfaces with PHY devices

The fourth subsystem is an expansion interface designed to host a daughter card or to communicate with another board. For that purpose we brought all remaining high-speed serial I/O connections out to two high-speed connectors. Finally, the fifth subsystem is concerned with the configuration of the FPGA.

Overall, the board is implemented as a dual-slot, full-size PCIe® adapter. Two slots are required for reasons of heat/power and height. Like high-end graphics cards, this board needs to be fed with an additional ATX power supply, since power consumption—given absolute maximum load on the FPGA—could exceed the PCIe single-slot allowance of 50 watts. However, the board can also operate standalone outside a server environment.

Memory Subsystem

A central focus in our design process was the interface to SRAM and DRAM components. Since the total number of I/Os on the FPGA device limits the overall available off-chip bandwidth, we had to strike a carefully considered compromise to facilitate as many applications as possible. Trying to support applications ranging from network monitoring through security and routing to traffic management imposes greatly varying constraints.

In regard to external memory access, for example, a network monitor would require a large, flow-based statistics table and most likely a flow classification lookup. Both accesses would require short latencies, as the flow classification needs more than one lookup with intradependencies, and the update in a flow statistics table would typically encompass a read-modify-write cycle. Hence, SRAM would be an appropriate device selection. However, a traffic manager would mainly need a large amount of memory for packet buffering, typically implemented through DRAM components due to density requirements. As a final example, consider an IPv4 router that needed a routing-table lookup as well as packet buffering in respect to external memory.

Summing up the requirements from a range of applications, we realized that certain functionality would consume external memory bandwidth, whether it be SRAM or DRAM. Where packet buffering (requiring a large storage space) would point to DRAM, SRAM would be the choice for flow classification search access, routing-table lookup, flow-based data table for statistics or rule-based firewall, memory

management tables for packet buffer implementations and header queues.

All of this functionality needs to be carried out on a per-packet basis. Therefore, given the worst case of minimum-size packets of 64 bytes with 20 bytes of overhead, the system needs to service a packet rate of roughly 60 Megapackets per second. Second, we need to differentiate the accesses further. To begin with, many memory components such as QDR SRAM and RLD RAM SIO devices have separate read and write data buses. Since the access patterns cannot be assumed to be symmetri-

refined our requirements further as seen in Table 1.

Assuming a clock speed of 300 MHz on the interface, a QDR II interface can service $2 \times 300 = 600$ million accesses/second for read and for write operations. Hence, three QDR II x36-bit interfaces could fulfill all of our requirements.

In regard to DRAM access, we considered mainly the case of packet storage, where each packet is written and read once from memory. This translates into a data access bandwidth of roughly 62 Gbps once you have removed the packet overhead

Compared with other 10G transceiver standards such as XENPAK and XFP, SFP+ has significant advantages in terms of power consumption and size. With SFP+ cages, we can support a number of interface standards, including 10GBase-LR, 10GBase-SR, 10GBase-LRM and low-cost direct-attach SFP+ copper cable (Twinax). Furthermore, SFP modules for 1-Gbps operation can be utilized, thereby supporting the 1000Base-T or 1000Base-X physical standards as well.

Each of the four SFP+ cages connects to a NetLogic AEL2005 device via the SFI

	Data width (bits)	#Reads per packet	#Writes per packet	#Reads per x36 interface	#Writes per x36 interface
Flow classification (5tuple + return key)	144	2	0	8	0
Routing-table lookup (dip + next hop)	96	2	0	6	0
Flow-based information	128	2	0	8	0
Packet buffer memory management	32	2	2	2	2
Header queues	32	1	1	1	1
Total number of accesses				25	3
Total number of accesses per second (MAps)				1,500	180

Table 1 – SRAM bandwidth requirements

cally distributed, we cannot pool the overall access bandwidth, but must consider the operations individually.

What's more, there is a third type of access, namely "searches." Searches can be ideally implemented through TCAM-based devices, which give guaranteed answers with fixed latencies. However, we ruled out this type of device for our board for price and power reasons, along with the fact that TCAMs further constrain the use of the I/O. Searches can also be implemented in many other ways such as decision trees, hash algorithms and decomposition approaches, with or without caching techniques, to name a few [2]. For the purpose of this discussion, we assumed that a search can be approximated through two read accesses on average. Given these facts and making some common assumptions on data widths, we

from the originally incoming 2×40 Gbps. In terms of physical resources, an RLD RAM II access can probably achieve an efficiency of around 97 percent, whereas DDR2 or DDR3 devices would more likely come in at around 40 percent [3], hence requiring significantly more I/O. We therefore chose RLD RAM II CIO components. Two 64-bit RLD RAM II interfaces at 300 MHz deliver a combined bandwidth that is roughly enough to service the requirement.

Network Interface

The network interface of the NetFPGA-10G consists of four subsystems that can be operated as 10-Gbps or 1-Gbps Ethernet links. To maximize usage of the platform as well as minimize power consumption, we chose four Small Form-Factor Pluggable Plus (SFP+) cages as the physical interface.

The AEL2005 is a 10-Gigabit Ethernet physical-layer transceiver with an embedded IEEE 802.3aq-compliant electrical-dispersion compensation engine. Besides regular 10G mode, the PHY device can support Gigabit Ethernet (1G) mode. On the system side, these PHY devices interface to the FPGA via a 10-Gigabit Attachment Unit Interface (XAUI). When operating in 1G mode, one of the XAUI lanes works as a Serial Gigabit Media Independent Interface (SGMII).

Suitable interface IP cores are available inside the Virtex-5 FPGA. For 10-Gbps operation, Xilinx markets the XAUI LogiCORE™ IP as well as a 10-Gigabit Ethernet media-access controller (10GEMAC) LogiCORE IP [4, 5]. For 1G operation, the interfaces can be directly connected to the embedded Xilinx Tri-Mode Ethernet MAC core [6].

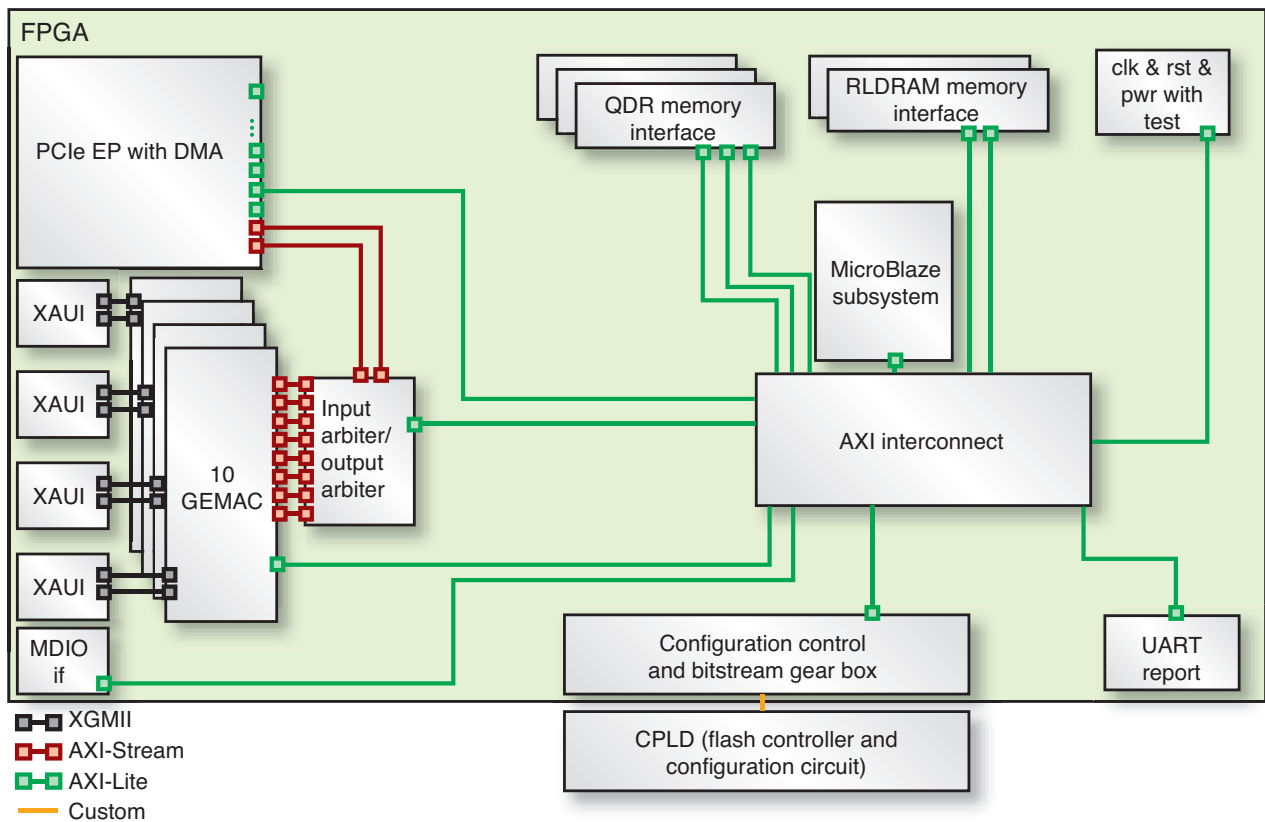


Figure 3 – The FPGA design architecture for 10G operation relies on the AXI protocol.

PCIe Subsystem

The NetFPGA-10G platform utilizes the Open Component Portability Infrastructure (OpenCPI) as its main interconnect implementation between the board and the host computer over PCIe [7]. Presently we support the x8 first generation, but might potentially upgrade to second generation in the future. OpenCPI is a generalized, open-source framework for interconnecting blocks of IP that may each use varied types of communication interfaces (streaming, byte-addressable and so on), with varied bandwidth and latency requirements. In its essence, OpenCPI is a highly configurable framework that provides the critical communication “glue” between IP that is needed to realize a new design quickly.

OpenCPI delivers a few key features for the NetFPGA-10G platform. On the software side, we are able to provide a clean DMA interface for transferring data (primarily including packets, although not

limited in any way to this type of information) and for controlling the device via programmed input/output. For networking applications, we provide a Linux network driver that exports a network interface for each of the physical Ethernet ports on the NetFPGA device. This allows user-space software to transfer network packets to the device as well as read any host-visible register in the design.

On the hardware side, OpenCPI provides us with a clean, or even multiple, data-streaming interfaces, each configurable through the OpenCPI framework. In addition, OpenCPI handles all of the host-side and hardware-side buffering and PCIe transactions, so that users can focus on their particular application rather than the details of device communication.

Expansion Interface, Configuration Subsystem

The purpose behind the expansion subsystem is to allow users to increase port

density by connecting to a second NetFPGA-10G board—to add different flavors of network interfaces through an optical daughter card, for example—or to connect additional search components such as knowledge-based processors with high-speed serial interfaces. We bring out 20 GTX transceivers from the FPGA and connect them through AC-coupled transmission lines to two high-speed connectors. Designed for transmission interfaces such as XAUI, PCIe, SATA and Infiniband, these connectors can link to another board either directly, with mating connector, or via cable assemblies. Each transmission line is tested to operate at 6.5 Gbps in each direction, thereby providing an additional 130-Gbps data path in and out of the FPGA.

Configuring FPGAs with ever-increasing bitstream size can potentially be a problem when the configuration time of the device exceeds the PCIe limits. This is

the case with the V5TX240T. The access speed of the platform flash devices, which is significantly below what the V5TX240T can theoretically handle, imposes a bottleneck. As countermeasures, designers might consider configuration of partial bitstreams, as well as accessing platform flash devices in parallel and configuring the FPGA at maximum speed. To facilitate the latter possibility, we equipped the board with two platform flash devices that connect through a CPLD to the FPGA's configuration interface. In addition, the board also supports the standard JTAG programming interface.

The FPGA Design

A key aspect behind a successful open-source hardware repository must be a clean architectural specification with standardized, abstracted and well-defined interfaces. In fact, we believe these interfaces to be crucial to providing a building-block system that enables easy assembly of components contributed from a large global community. Standardization ensures physical compatibility, and a rigorous definition that goes beyond physical connectivity reduces the potential for misunderstanding on the interface protocols. Ideally, users can then deploy abstracted components without further knowledge of their intrinsic implementation details.

As part of the full platform release, we will provide the complete architectural specification as well as a reference design with key components. These key components include:

- PCIe endpoint from OpenCPI [4]
- Four XAUI LogiCORE IP blocks and four 10GEMAC LogiCORE cores (the latter under licensing agreement) in 10G mode [7,8]
- Two RLLDRAMII memory controllers based on the XAPP852 [8]
- Three QDRII memory controllers based on MIG3.1 [9]
- Clocking and reset block that checks clock frequencies and power-OK signals, and generates all required system clocks

- Tri-Mode Ethernet MAC (TEMAC) for 1G operation [9]
- MDIO interface for configuration of the PHY devices
- Control processor in the form of a MicroBlaze®, with a support system that handles all administrative tasks
- UART interface
- Configuration control and bitstream gear box, which provides a programming path to the platform flash devices
- Input arbiter that aggregates all incoming traffic into one data stream
- Output arbiter that distributes the traffic to the requested output port

Figure 3 illustrates how these components are interconnected for 10G operation. For data transport we chose the AMBA®4 AXI streaming protocol, and for control traffic the interface is based on AMBA4 AXI-Lite. You can find further details on these interface specifications on www.arm.com (and on the Xilinx webpage in the near future).

Status and Outlook

Design verification of the board is complete. Once production test development is finished, the board should shortly be released for manufacturing. Initial FPGA designs are running and we are working now toward a first code release.

You can order through HiTech Global, which is manufacturing and distributing the board (http://www.hitechglobal.com/Boards/PCIEExpress_SFP+.htm). Different pricing models apply to academic and commercial customers. For the most up-to-date information on the project, please visit our website, www.netfpga.org, or subscribe to the netfpga-announce mailing list: <https://mailman.stanford.edu/mailman/listinfo/netfpga-announce>.

The first code releases, planned for the coming months, will open up the source code to the wider community. It basically contains the design detailed in the FPGA design section and implements the introduced architecture. In addition, we will put

significant effort into providing the right type of infrastructure for building, simulating and debugging the design. A last focus point in the coming months will be the repositories and framework that enable the efficient sharing of experience, expertise and IP within the NetFPGA community. 🌈

Acknowledgements

Many thanks to the following people for their help with this project: Shep Siegel, CTO of Atomic Rules and driving force behind OpenCPI; John Lockwood, CEO of Algo-Logic Systems, driving force behind the first generation of NetFPGA and an active contributor to the current platform; Paul Hartke from Xilinx University Program (XUP) for his endless energy and support; Rick Ballantyne, senior staff engineer at Xilinx, for his invaluable advice on board design; Adam Covington and Glen Gibb at Stanford for their support; Fred Cohen from HiTech Global for his cooperation.

Furthermore, we would like to thank the National Science Foundation and the following sponsors for donating their components, namely Micron, Cypress Semiconductor and NetLogic Microsystems.

References

- [1] XC5VTX240T data sheet: http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf
- [2] David E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *ACM Computing Surveys (CSUR)*, volume 37, No. 3, September 2005
- [3] Bob Wheeler and Jag Bolaria, *Linley Report: "A Guide to Network Processors," 11th Edition, April 2010*
- [4] XAUI data sheet: <http://www.xilinx.com/products/ipcenter/XAUI.htm>
- [5] 10GEMAC data sheet: <http://www.xilinx.com/products/ipcenter/DO-DI-10GEMAC.htm>
- [6] Tri-Mode Ethernet MAC http://www.xilinx.com/support/documentation/ip_documentation/hard_temac.pdf
- [7] www.opencpi.org
- [8] XAPP852 (v2.3) May 14, 2008: http://www.xilinx.com/support/documentation/application_notes/xapp852.pdf
- [9] MIG data sheet: <http://www.xilinx.com/products/ipcenter/MIG.htm>

A Flexible Operating System for Dynamic Applications

By using the dynamic reconfiguration capabilities of Xilinx FPGAs, software flexibility and hardware performance meet in a homogeneous multithread execution model.

by Fabrice Muller

Associate Professor

University of Nice Sophia, Antipolis, France — LEAT - CNRS

fabrice.muller@unice.fr

Jimmy Le Rhun

Research Engineer

Thales

jimmy.lerhun@thalesgroup.com

Fabrice Lemonnier

Project Manager

Thales

fabrice.lemonnier@thalesgroup.com

Benoît Miramond

Associate Professor

ETIS, UMR 8051 CNRS - ENSEA - UCP

benoit.miramond@ensea.fr

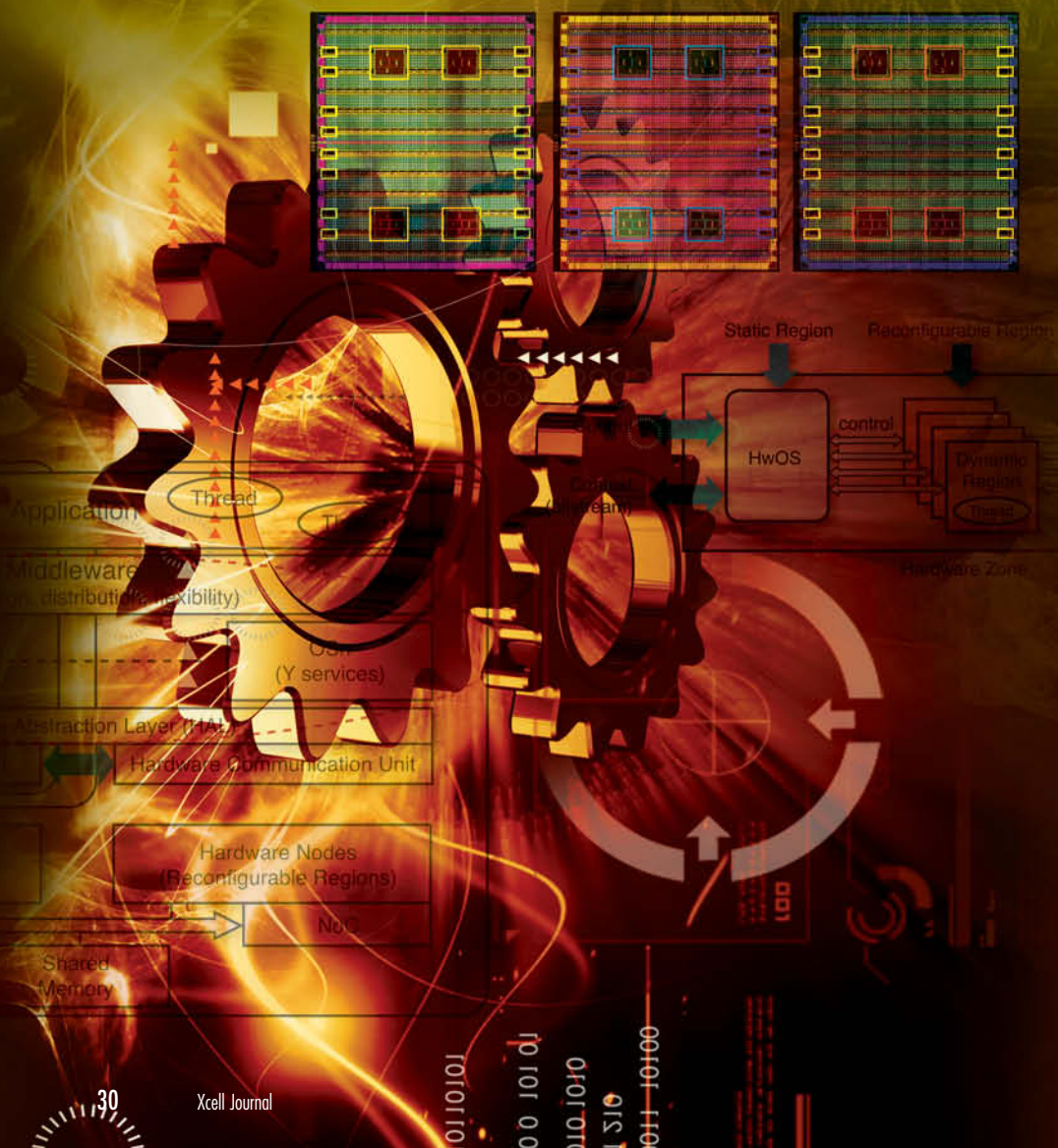
Ludovic Devaux

PhD

University of Rennes 1 / Cairn Inria team

ludovic.devaux@irisa.fr

An autonomous robot finding its way in unknown terrain; a video decoder changing its decompression format according to signal strength; a broadband electronic countermeasure system; an adaptive image-tracking algorithm for automotive applications. These are among the many emerging embedded or mission-critical applications that show dynamic behavior with strong dependency on an unpredictable environment. Where statically resolved worst-case allocation was once an answer to strong real-time constraints, flexibility is now also a requirement. The solution proposed in a French research project is an operating system distributed onto the FPGA resources that manages both hardware and software threads.



Our goal is to define an architecture that supports a new kind of system partitioning, where hardware and software components follow the same execution model. This requires a very flexible and scalable operating system.

In recent years, denser systems-on-chip (SoCs) have made it possible to meet the design constraints by parallelizing tasks and data, thereby increasing the number of computation units, in particular in embedded systems. This trend continues today with the addition of heterogeneity of computing cores. But this technique becomes a wall of complexity which dictates a higher level of abstraction in the programming model.

To surmount these challenges, we propose to define a unified execution model that is used whether the threads are mapped onto hardware or software. The hardware implementation of this execution model relies strongly on the use of dynamically reconfigurable logic. Coupled with a traditional multicore software subsystem, a fully distributed architecture provides the best of both worlds. The software part is well-suited to intelligent event-based control and decision making, while the hardware part excels in power efficiency, high throughput and number crunching. In between, we get the ability to balance performance and resource utilization, not just for each specific application, but also for a specific state of an application.

The high integration capabilities of modern, platform FPGAs make it possible to put a full, heterogeneous and dynamic computing system into one or two chips, with a high level of flexibility and scalability.

The adaptive hardware is very useful in applications such as missile electronics and software-defined radio, where power consumption and system size are limited, and which must be highly reactive to the environment. Thanks to dynamic reconfiguration, you can implement dedicated architectures for the different application modes without increasing the power consumption of your system or the size of the board. The classical solutions centralize the

control and do not, today, seem able to address effectively both the number of execution units and their heterogeneity. Only a distributed approach that is both flexible and scalable can take up this challenge of creating a future-proof architecture.

Despite the potential of this technology, the use of dynamic reconfiguration is still a challenge for the industry. Engineers need a clear design methodology to get the full benefit of dynamic reconfiguration, without impacting on application description, and, above all, with no increase in development costs. In an attempt to combine dynamicity and performance, we propose to abstract the heterogeneity by means of an execution model based on multithreading. The developer will be able to program their application as a set of threads, irrespective of whether the threads are executed on a standard processor or on dedicated hardware. In this context, dynamic reconfiguration becomes a method for thread preemption and context switching. The FOSFOR project (Flexible Operating System for Reconfigurable platform), sponsored by the French National Research Agency (ANR), is dedicated to developing this new generation of embedded, distributed real-time operating systems.

FOSFOR Architecture Basics

Our goal is to define an architecture that supports a new kind of system partitioning, where hardware and software components follow the same execution model. This requires a very flexible and scalable operating system, one that provides similar interfaces to the software and hardware domains. Unlike classical approaches, this OS can be completely distributed, with the whole platform being homogeneous from the application's point of view. This means

you can deploy the application threads in software (processors) or in hardware (reconfigurable units), either statically or dynamically, with indifferent access to the distributed services.

We have implemented operating-system services in hardware, next to the reconfigurable zones, to reach a high level of efficiency. We have implemented a communication layer between heterogeneous OS kernels to ensure the homogeneity of the services from the application's point of view. So, deploying the OS on the architecture as a large number of modules and execution units takes advantage of the virtualization mechanisms that allow the application threads to run and communicate without a priori knowledge of assignment.

From the programmer's point of view, the application is just a set of threads. We use the dynamic reconfiguration capabilities of Xilinx® FPGAs to propose the new concept of hardware threads, along with an implementation of this concept with the same behavior as software threads. Our implementation takes advantage of the performance of dedicated computing IP blocks.

Along with the execution units present in the multiprocessor SoC, the memory organization must meet several requirements: the storage of the data needed by the application threads, the storage of the execution context of each thread and the exchange of data between threads. Regarding the storage of execution contexts, we envisage several possibilities. One of these is to centralize the execution contexts and thus provide a medium for their distribution to different execution units. We identify three types of communication streams within the platform: application data, control signals and configuration/execution contexts. The high-bandwidth data

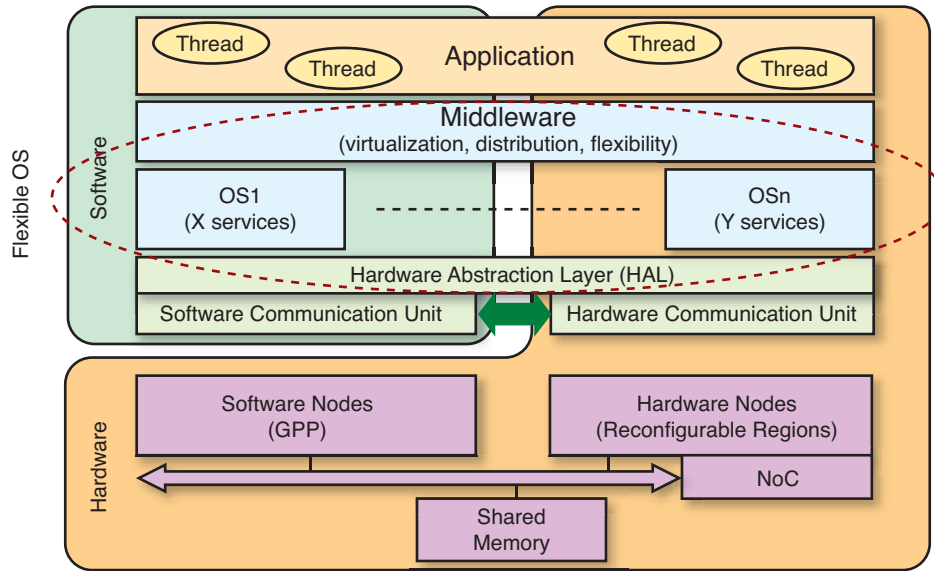


Figure 1 – Generic FOSFOR architecture

paths between hardware threads use a dedicated network-on-chip (NoC).

The Global Architecture

The global architecture is depicted in Figure 1 and consists of:

- A set of nonspecialized, or general-purpose, processors (GPPs). A GPP supports the execution of software threads. It also supports a set of OS services, one of which schedules the threads. The GPPs are not necessarily homogeneous in terms of instruction-set architecture and number of offered services.
- A set of dynamically reconfigurable zones (called the reconfigurable region, RR), which are in charge of the simultaneous or sequential execution of a set of hardware threads. Like a GPP, an RR supports the execution of OS services thanks to a hardware OS (HwOS). These regions correspond to fine-grained (FPGA) or coarse-grained (reconfigurable processor) architectures.
- Virtual channels of communication—control, data and configuration—that can share one or more physical communication channels. The control channels allow communication

between OS services distributed onto the execution units (GPP and RR). The data channels convey information related to the environment (devices, sensors) and the exchange between the threads. The configuration channels allow the transfer of the configurations of the software threads (binary code) and hardware threads (partial bit-streams) between the configuration memory and the execution units.

Each processor has its own local memory. This memory stores the local data and, where applicable, the software code. A shared memory, connected to the data channel, allows data sharing between threads on different processors. Each execution unit has access to shared memory, containing the data and the programs of software execution resources. Each resource also has access to a configuration memory, to save and restore its execution context. This structure makes it possible to implement any thread or service on any execution resource.

Within the RR, only the hardware tasks need to be dynamically reconfigured. The dynamic region (DR), the area hosting the tasks, is surrounded by the static region (SR), which contains hardware implemen-

tation of OS services, along with communication media both internal and external to the RR. Internal data-stream communications rely on a dedicated network-on-chip. The interfaces between DR and SR use a bus macro, and have a fixed location. To abstract this constraint, and the heterogeneity of the communication medium, we propose a middleware approach that provides virtualized access to the reconfigurable zone. An RR is organized according to the model defined in Figure 2. The FOSFOR prototype platform consists of dynamically reconfigurable FPGA devices that can already support this architecture model. We selected Virtex®-5 devices for their ability to reconfigure rectangular regions.

We have defined a scheduling/placement algorithm that ensures efficient use of the FPGA elements (LUTs, registers, distributed memory, I/O) inside each RR according to the precomputed resource needs of the application threads.

OS, NoC and Middleware

To be flexible, the FOSFOR architecture uses at least two instances of operating systems: a software OS that runs on each processor and handles software threads, and a hardware OS that is able to manage hardware threads. To correctly balance the

Thanks to partial dynamic reconfiguration provided by Xilinx FPGAs, the hardware OS can schedule hardware threads with as much flexibility as a classical operating system would with software threads.

trade-offs between performance, development time and standardization, we used an existing operating system for the software part, but we designed the hardware operating system from scratch.

The requirements for the software OS are real-time behavior, the ability to handle multiple processors and the availability of basic interprocess communication services. We chose a free, open-source OS, namely RTEMS (Real-Time Executive for

machine to synchronize the service call sequence with the HwOS. The static part contains a control interface with the HwOS, and a network interface for data exchange with the other tasks, both hardware and software.

To support diverse interthread data transfer needs, we have developed a flexible network-on-chip called DRAFT. Communication services of a classical operating system are sufficient to support

for DRAFT is an extension of the Fat-Tree topology. The main objective of our design is to limit resource overhead while allowing high-performance interthread communication.

The heterogeneity of the hardware platform is a major factor of complexity when designers deploy their applications. In the FOSFOR project, this heterogeneity comes not only from the different embedded processors in the software domain, but

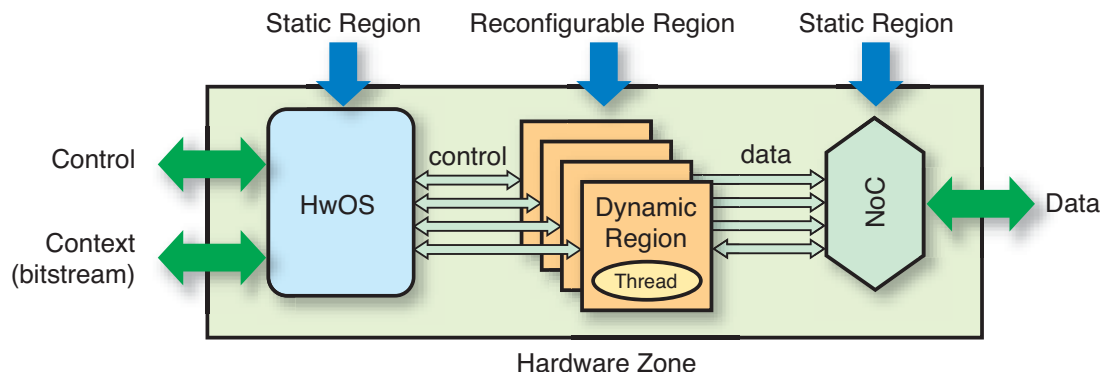


Figure 2 – Reconfigurable region structure

Multiprocessor Systems; see <http://www.rtems.org/>). For compatibility reasons, we selected the LEON Sparc soft-core processor, which is also free and open-source, as the software node.

Thanks to partial dynamic reconfiguration provided by Xilinx FPGAs, the HwOS can schedule hardware threads with as much flexibility as a classical operating system would with software threads. A hardware thread consists of two parts, one dynamic and one static. The dynamic part includes an IP block that implements the thread functionality and a finite state

communications between software threads. However, in our case, the OS also needs to support communications between hardware threads. We specifically designed the DRAFT network for this purpose. We synthesize each hardware thread for one or more DRs, and statically define each DR interface.

The static definition of the communication interface allows us to define a static network-on-chip. In general, the hardware threads need large bandwidth and low latency, so the NoC must provide high performance. The topology we chose

above all from the presence of both software and hardware computation models in a single platform.

The middleware solves this problem by proposing an abstraction layer between hardware and software and providing a homogeneous programming model. It implements a set of virtual channels that allow communication between threads, independently of their implementation domain. These services are distributed across the platform and thus offer a scalable and flexible abstraction layer that completes the FOSFOR concept.

Performance Speedup

The main reasons for building a hardware OS are performance and flexibility. The OS could have been fully software or fully hardware. Since each call to an OS primitive

munication path is always available between two connected elements. Data travels through DRAFT with an average latency near 45 clock cycles (450 nanoseconds), which is compliant with many applications.

Number of implemented structures	8	16	32
CLB slices	2,408 (15%)	3,151 (20%)	4,327 (27%)
D flip-flops	5,498 (8.5%)	6,650 (10.4%)	8,918 (13.9%)
BRAMs	8 (3.5%)	16 (7%)	32 (14%)

Table 1 – Resource usage of the HwOS (Virtex-5 FX100)

involves overhead and means a waiting time for the thread, the faster the OS is, the less time wasted. To evaluate this overhead, we must compare the HwOS timings with the original software OS, RTEMS.

Hardware local operations need only a few tens of cycles, while hardware global operations require a few hundred cycles due to shared memory access. We have evaluated a speedup of about 60 times on local creation-and-deletion operations, and about 50 times for other operations, compared with results from the software operating system.

The resource usage of the HwOS (Table 1) varies greatly, depending upon the number and the capabilities of activated services. For example, we select the number of objects (semaphores, threads and so on) for each service. We use a Xilinx Virtex-5 FX100T to implement the system. The table lists the resources that the HwOS uses. The remaining space can be used to implement other system components and the hardware threads themselves.

Concerning network performance for a configuration where DRAFT connects eight elements with a word width of 32 bits, a buffer depth of four words and a 100-MHz frequency, the network-on-chip supports a data rate of up to 1,040 Mbits/second per connected element. The topology and the routing protocol of the network guarantee that no contention and no congestion can occur. At least one com-

Looking Ahead

We have proposed an innovative OS that provides a homogeneous execution model based on multithreading, on a heterogeneous multicore architecture composed of

processors and dynamically reconfigurable hardware IP blocks. A hardware OS manages the hardware threads, typically for thread creation and suppression but also for semaphore and message queue services. In terms of communications, we have proposed improvements to the Fat-Tree NoC for data exchanges, a dedicated bus for hardware thread management and a communication layer for inter-OS synchronization.

The next step, from an industrial point of view, is to demonstrate that the hardware functions, which were added to ensure homogeneity of the execution model, lead to a real improvement in programming efficiency while keeping a low performance overhead on the dedicated IP blocks.

We will demonstrate our approach on a representative Thales application, based on a search-and-track algorithm. The tracking threads are mapped onto reconfigurable zones and are created dynamically, depending on the target detection. ●●●

Imagine. Combine. Implement. Today.
IP Cores for Your Low-Volume FPGA Products

you have an amazing **idea?** | choose & configure IPs | implement your FPGA w/o coding | logicBRICKS™ make your ideas come true
looking for the perfect graphics controller?

Low Cost IP Cores:

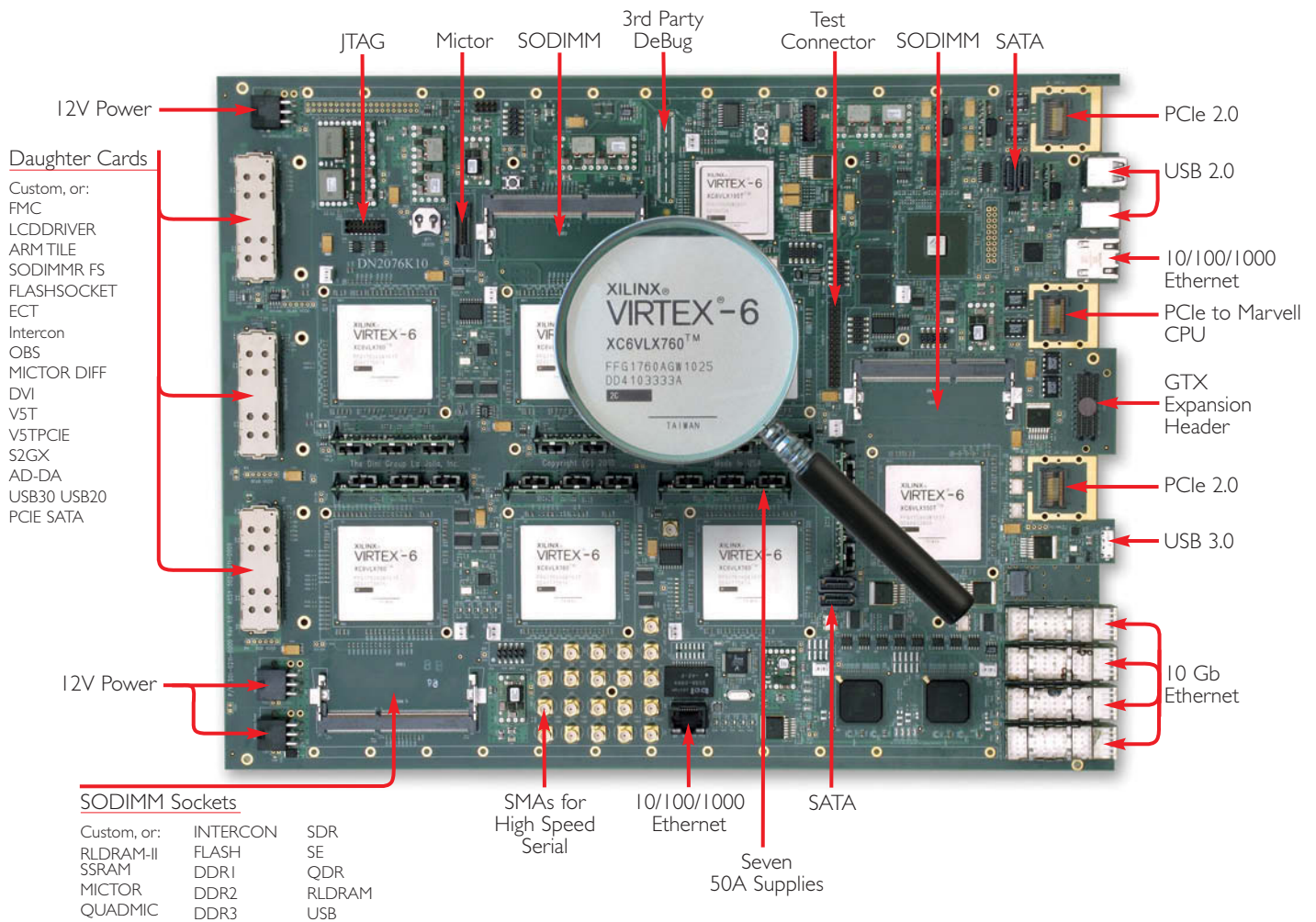
- ★ Graphics and video solutions
- ★ Fully compatible with Xilinx® tools
- ★ Simple GUI parameterization
- ★ Evaluation prior to purchase
- ★ Support from skilled FPGA Engineers
- ★ Available reference hardware platforms
- ★ Support all the latest Xilinx FPGA families

Get Online, Plug and Play!
Prices start at €500
Visit our web shop today:
www.logicbricks.com

logicBRICKS™
Designed by XYLON

logicBRICKS is a registered trademark of Xylon d.o.o.
All other trademarks are the property of their respective owners

Why build your own ASIC prototyping hardware?



DN2076K10 ASIC Prototyping Platform

All the gates and features you need are — off the shelf.

Time to market, engineering expense, complex fabrication, and board trouble-shooting all point to a 'buy' instead of 'build' decision. Proven FPGA boards from the Dini Group will provide you with a hardware solution that works — on time, and under budget. For eight generations of FPGAs we have created the biggest, fastest, and most versatile prototyping boards. You can see the benefits of this experience in our latest Virtex-6 Prototyping Platform.

We started with seven of the newest, most powerful FPGAs for 37 Million ASIC Gates on a single board. We hooked them up with FPGA to FPGA busses that run at 650 MHz (1.3 Gb/s in DDR mode) and made sure that 100% of the board resources are dedicated to your application. A Marvell MV78200 with Dual ARM CPUs provides any high speed interface you might want, and after FPGA configuration, these 1 GHz floating point processors are available for your use.

Stuffing options for this board are extensive. Useful configurations start below \$25,000. You can spend six months plus building a board to your exact specifications, or start now with the board you need to get your design running at speed. Best of all, you can troubleshoot your design, not the board. Buy your prototyping hardware, and we will save you time and money.



Resurrecting the Cray-1 in a Xilinx FPGA

Want a classic, mammoth, number-crunching supercomputer of your own? Build one.

by Christopher Fenton

The year was 1976. Disco was still popular, the Cold War was in full swing and I wouldn't even be born for another nine years when the Cray-1 burst onto the computing scene. Personal computing was barely in its infancy (the MITS Altair had been introduced a year earlier) at the time, and companies like Control Data Corp. and IBM dominated the high end. The Cray-1 was one of those legendary machines that helped define the term "supercomputer" in the public imagination. Its iconic C-shape structure housed a fire-breathing machine running at 80 MHz—something desktops wouldn't reach until almost two decades later. The Cray had speed. It had style.

Now let's fast-forward 33 years, to the morning in early 2009 when I woke up and just decided I wanted to own one.

I first got into FPGA-based retro-computing, something I lovingly refer to as "computational necromancy," shortly after graduating from the University of Southern California with a BSEE in December 2007. As a newly minted electrical engineer and all-around fan of arcane computer architectures, I saw this pursuit as the perfect excuse to improve my Verilog skills. Starting with a Digilent Spartan®-3E 1200 board that I bought myself as a graduation present, my first machine was another abandoned relic of the 1980s, the NonVon-1. This was one of the first "massively parallel" machines, similar to the more successful Connection Machine series of the same vintage, although geared more toward databases. It was a wonderfully odd machine, composed of a binary tree of 8-bit processors (with 1-bit ALUs).

The Cray-1 was one of those legendary machines that helped define the term 'supercomputer' in the public imagination. Its iconic C-shape structure housed a fire-breathing machine running at 80 MHz—something desktops wouldn't reach until almost two decades later.

After a few months of tinkering, I eventually found myself the proud owner of a 31-node supercomputer dwarfed in computing power by any modern wrist-watch. As useless as it was, however, the machine made me realize just how far Moore's Law has brought us. And it whetted my appetite for more.

After my success with the NonVon-1, I was casting around for a new project (and my Verilog skills were still a bit lacking). I realized that low-end FPGAs had grown to the point that they could handle some pretty serious hardware—even 32-bit "soft" processors are fairly common these days. Searching about for a new target to try to revive, I considered a few—the UNIVAC is an interesting machine, but it's a bit too old for me. Digital Equipment Corp.'s PDP series has been emulated before. Simulators for Z80 machines are commonplace. That's where the Cray comes in.

What is the Cray-1?

The Cray-1 was Seymour Cray's first machine after splitting off from Control Data and founding his own company, Cray Research, in the early 1970s. It was a ruthless number cruncher that required a room full of computers and disks to keep it fed with data. It also had a full-time staff of engineers just to keep it running, and nearly required its own power plant just to boot up. This is a machine that redefined the term "supercomputer" (I mean, it's a Cray)—and, fortunately, it's also beautifully simple in its design. Thankfully, it's incredibly well-documented too (Figure 1). The Cray-1 Hardware Reference Manuals (readily available on the Internet) go into a level of detail that's almost shocking to modern-day readers used to being handed black boxes. Nearly every op-code, register and timing diagram is documented in exquisite detail.

The computer itself is a 64-bit, pipelined processor with in-order instruction issue and a mere 128 unique instructions. It has a very RISC-like instruction set, with all instructions being either between memory and registers (load or

store instructions) or between two operand registers and a destination register (all arithmetic/logic instructions). Instructions are either 16 or 32 bits long. The machine uses three different types of registers: address, scalar and vector registers. The

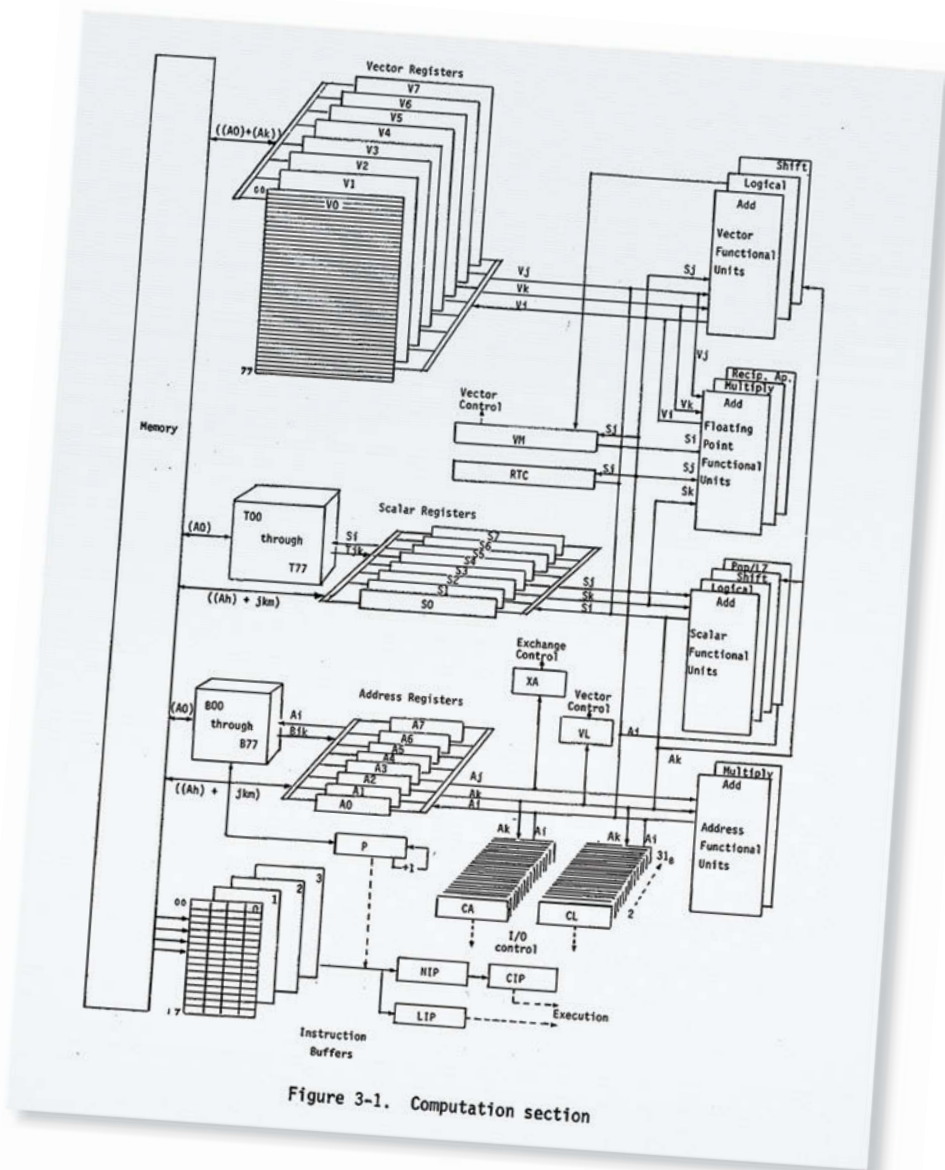


Figure 1 – Fortunately for hobbyists, the Cray architecture is beautifully simple in its design and very well-documented.

address registers are 24 bits wide and let the machine address up to 4 Megawords (32 Mbytes) of main memory. The scalar registers, which are 64 bits wide, are used for computation. Each vector register contains sixty-four 64-bit registers, giving the machine great performance when doing scientific calculations on large matrices.

Inside the CPU, instructions can be issued to 13 independent, fully pipelined “functional units.” Heavy pipelining was

yes, I was.) Was my FPGA big enough to actually fit it? (As it turned out, no, it was not.) Even if the design is fairly straightforward, it’s still a large design (currently ~5,600 lines of Verilog and counting). I just had to get myself into the right mindset. Building your own supercomputer is a marathon, not a sprint. I could only hope to accomplish it one step at a time.

I started, one by one, with creating the functional units. Like building your own

working on the Cray-1 in early 2009 and probably spent 19 to 20 months total on it.

I started to get my second wind toward the end of the floating-point units’ design, and regained steam as I moved on to the vector units. As I mentioned earlier, the Cray-1 was designed as a number-crunching behemoth. It has eight vector registers, each of which holds sixty-four 64-bit registers. When a vector instruction executes, say an addition operation, one entry from each operand will be added and stored in a third (result) vector on every cycle.

An awesome feature that the Cray-1 supports is called “vector chaining.” The vector add unit, for instance, only takes three cycles to generate the first result. If we’re adding two 64-entry vectors together, however, we don’t want to wait for all 64 entries to finish adding before we do something with the result. Vector chaining allows us to “chain” the result coming out of the adder unit straight into the input of another unit, without waiting for the operation to finish. We can start multiplying the result with a third vector two cycles after the first result is available. For some large matrix calculations, you could almost sustain two floating-point operations per clock cycle—at 80 MHz, that’s a peak rate of 160 MFLOPS! Common desktop computers didn’t catch up to the Cray-1 until the mid-1990s.

With the functional units in place, I could almost see the light at the end of the tunnel. Surely it was just a matter of adding in a bit of glue logic and being done with it, right? Well, close. It turns out there’s a lot of glue logic. Even though the Cray-1 is well-documented, it’s not *that* well-documented. I knew exactly what every instruction was supposed to do, but I got stuck reverse-engineering minor (and not-so-minor) details like instruction issuing, hazard detection and vector chaining. Some things, like big 64-bit data buses, are probably easier to build with discrete logic chips than with FPGAs designed for narrower datapaths. The vector registers gave me a routing headache.

A few features I also had to fudge. The Cray-1 had a 16-bank all-SRAM memory system the size of my refrigerator that could

Building your own supercomputer is a marathon, not a sprint. I could only hope to accomplish it one step at a time. I started, one by one, with the functional units, creating the easiest blocks first.

crucial to achieving the Cray’s insanely high (for the time) 80-MHz clock frequency. Separate functional units handle logical operations, shifting, multiplication and so on. A floating-point multiply instruction, for instance, takes seven cycles to complete, but the computer can issue a new multiply instruction on every cycle (assuming no register conflicts exist). An interesting consequence of this design is that there is no “divide” instruction. Instead, the machine uses “division by reciprocal approximation.” Rather than computing X / Y , you compute $(1 / Y) * X$. A separate floating-point “reciprocal approximation” functional unit can calculate a reciprocal in 14 clock periods.

The Marathon

When I first began working on this project, I still hadn’t convinced myself it would be possible to re-create such a sophisticated computing machine by myself. The original Cray-1 took a whole team of people years to design and build. Was I motivated enough to stick with it? (As it turned out,

hot rod, building a complete computer gets you acquainted with every aspect of a design in a way you would rarely experience otherwise. I explored multiplier and adder design. I reopened textbooks on floating-point arithmetic. I learned how to use three iterations of the Newton-Raphson method to compute a reciprocal approximation to 30 bits of accuracy (did I mention how detailed the hardware reference manual is?).

One by one, the functional units took shape. This was a strictly “free-time” project, so progress came in fits and starts. I started with the easiest blocks first, and finished the two address functional units (a simple adder and a multiplier) without much difficulty. My momentum started to falter as I tackled the scalar functional units (an adder, a logical unit, a shifter and a population/leading zero count). I hit a low point in my motivation as I fiddled with the three floating-point functional units (an adder, a multiplier and the infamous reciprocal approximation unit). As I said, this was a marathon, not a sprint. I started

sustain 640 Mbytes/second of bandwidth (one 64-bit word per cycle at 80 MHz) to its 4 Megawords of memory, something the measly DDR memory chip on my development kit could never approach. I wound up using nearly all of my FPGA's on-die Block RAM to scrounge together a mere 4 kilowords of memory space, by far my Cray's biggest limitation at the moment. And I had to leave a few features out altogether: DMA-style I/O channels designed to communicate with disk drives and "host" minicomputers, and rapid context-switching support. These might make it back in once I get a useful amount of memory and a bit of software for the machine.

Hardware Obstacles

I should take a few words to make note of some of the FPGA-related challenges I ran into along the way. First of all, my original Spartan-3E 1200 chip didn't prove to be up to the challenge. As soon as I added the Cray's mammoth vector registers to my design, I overflowed the chip's meager logic resources. I started to sweat a bit at this point—I was already more than a year into this project, and the price tag of bigger FPGAs seems to go up exponentially with size. The larger Virtex® chips could handle my design with ease, but development boards typically exceed a few years' worth of my hobby budget. Fortunately, Digilent also sells a slightly larger Spartan-3E 1600 board for a still-hobbyist-friendly price. It proved to be just large enough to fit the whole system (and significantly boosted the amount of Block RAM I had to play with).

The other problem I ran into was simply one of speed. Despite 30 odd years of Moore's Law, the Cray-1's original 80-MHz design proved to be too much for my poor Spartan-3E. My initial design topped out at about 33 MHz, with the critical path running through cascaded adders I used in my rather naive implementation of a floating-point multiplier. Fortunately, the Spartan-3E is equipped with a number of 18-bit hardware multipliers that were able to boost the speed up to nearly 50 MHz (and shave off 5 percent of the area), but at that point the rest of the design starts to run into a wall. The

Cray-1's spaghetti bowl of 64-bit datapaths and complicated instruction-issue logic limit the cycle time to 20 nanoseconds or so on the Spartan-3E. For now I'm satisfied, but maybe one of the newer Spartan-6 chips will be up to the challenge of hitting the magical 8-0 mark.

Case Construction

With the hardware in mostly working order, I moved on to the fun part—the case. Sure, the circuit design is the interesting part for any engineer, but what fun is owning your own Cray-1 if it doesn't look like a Cray-1? It also gave my friend Pat the perfect project to try out his new CNC milling machine. A few trips to Home Depot and a busy

built-in "pleather" couch. Finally, Mattel's new Computer Engineer Barbie had somewhere to rest her weary feet!

The Software

With the CPU in mostly working shape and a matching case all ready to go, I was prepared to declare victory. I was flawlessly executing simple loops and programs of a few instructions. But what about real software? One of the unfortunate things about my shiny new Cray-1 was its total lack of software, and a computer without software is only marginally more useful than the sand it's built from. The Cray-1 unfortunately suffers from the double-whammy of existing in a pre-Internet world and being



Figure 2 – Putting together the iconic C-shaped case took some finessing and the help of a handy friend.

Saturday in Pat's garage (see Figure 2) left me with something that was definitely starting to look like a miniature Cray. The square shape of the development board meant I had to get a bit creative with the base (squares and C-shapes don't play nicely together), but the board's compact dimensions conveniently meant that my design worked out to be almost exactly 1/10 scale.

Another few weeks of sanding, painting and polishing followed. Finally, a trip to the local fabric store let me finish off the design with a tiny replica of the Cray-1's distinctive

sold primarily to government agencies with scary-sounding acronyms. I spent months trawling the depths of the Internet looking for software, but came up dry. I e-mailed people at some of our national labs. I even filed a Freedom of Information Act request with the National Nuclear Security Agency (government watch-list, here I come), but struck out there too.

Seeing that I needed a little more help, I finally decided to enlist the citizens of the Internet. I added a page on my micro-Cray (and adorable 1/10-scale case) to my web-

site, told a few of my friends with Twitter accounts about the project and let the Internet do its thing. Within a few days the story had been posted all over a number of news aggregator sites, flooding my in-box with e-mails from all over the world. Many were nostalgic messages from former “Crayons,” sympathetic with my plight. Gradually, however, a few people surfaced with decades-old source code on stacks of paper, nine-track tapes and even a reel of microfiche. Someone even e-mailed me a 20-year old PhD thesis, including their source code for a Cray-compatible compiler for an obscure programming language (written, of course, in that same obscure programming language). It was the ulti-

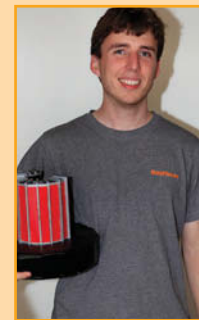
mate moment of pack-rat vindication for people storing obsolete software and documentation “just in case.”

The Future

What does the future hold for my mini mainframe (Figure 3)? I’d still like to iron out some more of the machine’s bugs and finish up the missing features. I haven’t completely solved my software problem, but things are starting to improve. One helpful netizen dug up a copy of a DOS-based Cray simulator from the late 1980s that also doubles as a simple assembler (and still runs perfectly under Windows 7). Programming in Cray Assembly Language is a dream compared with using straight

octal machine code. If I can overcome the challenges of obsolete media, it looks like I might also be able to get my hands on the source code for at least one operating system as well as a real Fortran compiler.

My obscure hobby also caught the eye of someone at the Computer History Museum (<http://www.computerhistory.org/>), so maybe my tiny Cray will retire to Mountain View, Calif., someday. In the meantime, though, I’m already starting to think about what my next project could be. 🌈



Christopher Fenton, an electrical engineer living in New York City, is an active member of his local Hackerspace, NYCResistor.

He is an avid historical-computing enthusiast and enjoys building impractical and unnecessarily complicated projects in his free time. Details on the Cray-1 and other projects are available on his website, www.chrisfenton.com.



Figure 3 – The final micro mainframe, built on a Spartan-3E.

A Great Listener

Digitize, down-convert and decode all on a single card!



ip wireless
cores

X6 rx

Features

- Four 160 MSPS, 16-bit A/D channels
- Down-Converter ASIC supporting up to 24 Narrowband or 8 Wideband Channels
- +/-1V, AC-Coupled, 50 ohm, SMA inputs
- Xilinx Virtex6 SX315T/SX475T or LX240T
- 4 Banks of 128MB DRAM
- Ultra-low jitter programmable clock
- x8 PCI Express Gen2, providing 2 GB/s sustained transfer rates
- PCI 32-bit, 66 MHz with P4 to Host card
- PMC/XMC Module (75x150 mm)
- < 15W typical
- Conduction Cooling per VITA 20
- Ruggedization Levels for Wide Temperature Operation
- Adapters for VPX, Compact PCI, desktop PCI and cabled PCI Express systems

Ideal for

- Wireless Receiver
- WLAN, WCDMA, WiMAX front end
- RADAR
- Medical Imaging
- High Speed Data Recording and Playback
- IP development



FrameWork Logic



**Innovative
Integration**
... real time solutions!

805.520.4260 phone • www.innovative-dsp.com

FPGA Partial Reconfiguration Goes Mainstream

This PR primer shows that techniques to reconfigure portions of a device on the fly are well within the reach of any design group.

by John McCaskill
President
Faster Technology
jhmccaskill@fastertechnology.com

David Lautzenheiser
Business Development
Faster Technology
dlautzenheiser@fastertechnology.com

Recognizing that it is no longer sufficient to just build an ever-larger FPGA, Xilinx has put the spotlight on the partial-reconfiguration capabilities of its devices as a powerful weapon in the competitive landscape, and more importantly has dedicated the resources and support to make partial reconfiguration an equally powerful capability for users. The tools and methodology are now in place to enable a much broader set of design teams to take advantage of partial reconfiguration than the avid early adopters of the past. This design technique is truly ready for prime time.

Xilinx's main-line FPGAs, by the very nature of the static-memory cells used to hold configuration information, have always been reconfigurable. In the earliest days, designers viewed this shape-shifting as a big headache because of the novelty of logic that "forgot" its function when the power was turned off. The initial negative perception was soon overcome as designers and their management realized the ultimate power of a logic device that could be reconfigured, enabling endless engineering experimentation, last-minute design changes or fixes and—to a few early visionaries—the idea that a system could be conceived of with multiple personalities. Thus was born the notion of partial reconfiguration.

In early devices, there was such a large transistor overhead burden of static memory to hold configuration data that the programming mechanisms had to be as simple and "lean" as possible. To keep

things simple, early FPGA families could only be configured or reconfigured as a whole. Even in this restricted environment, some clever users decoded bitstreams and figured out how to change LUT functions and a few other things in attempts to allow some on-the-fly changes to devices in their systems—a very early form and use of reconfiguration on the fly.

As Moore's Law progressed, an ever-increasing number of transistors became available for FPGA architects to consider using in new and novel ways. Early architectural experiments with completely random access to configuration data, such as the Xilinx® 6200 family, or with multiple configuration memory planes, such as the Prism project, tested what might be possible if some of this flood of transistors was applied to configuration schemes beyond just full-chip programming.

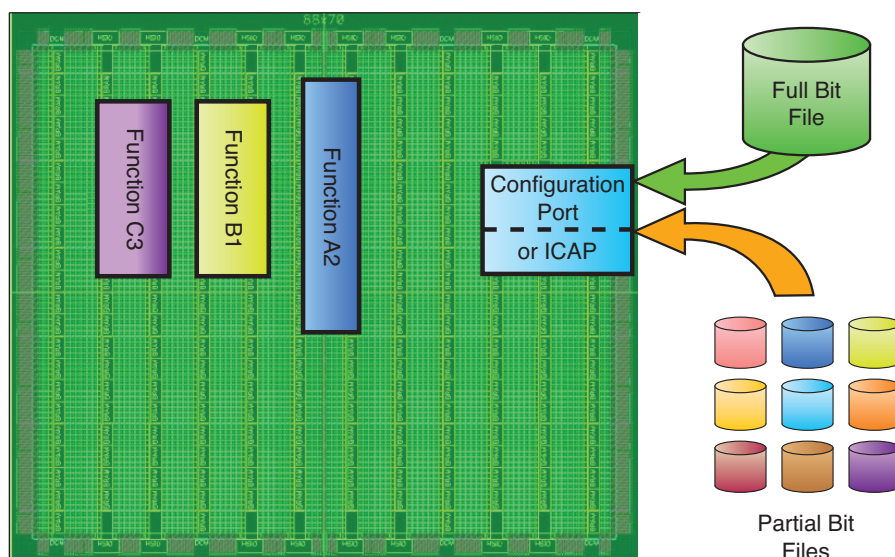


Figure 1 – Partial reconfiguration is the ability to dynamically modify blocks of logic by downloading partial bit files while the remaining logic continues to operate without interruption.

At the same time, design methods shifted from logic captured in schematics to functionality detailed in hardware description languages. The move to HDL design enabled a design team to conceive and implement more complex designs, but also removed them from many of the intricate low-level details of early FPGA design. This shift enabled teams to consider higher-level system functionality and alternative solutions more rapidly than previously possible.

All of the experiments and best thinking to that date went into the first Virtex® family. At this point, it became possible to put hardware capabilities into the device architecture that the tools flow could not immediately support. The hope, or perhaps dream at the time, was that tools would rapidly improve to take advantage of new and novel architectural capabilities. While it was a noble goal, designers never really got access to the vast majority of the really clever things in the first Virtex family—including the first commercially available true partial-reconfiguration capability. Figure 1 illustrates this concept of partial reconfiguration.

With the never-ending pace of improvements in FPGA devices and tools, the first Virtex family soon gave way to Virtex-2 and then II-Pro, 4, 5, 6 and, now, Virtex-7.

Throughout this evolution, a small contingent of dedicated users took whatever tools they could find, or in some cases created their own, and figured out very clever system uses for partial reconfiguration. A key

stumbling block to broader acceptance has been the lack of a well-defined methodology and supporting tools that fit with the design style of a significant and substantial fraction of the FPGA design community.

But today, the tools and support are finally in place and this powerful design approach is poised for broad uptake. The balance of this article will provide a primer on how an average designer can approach partial reconfiguration (PR) as a unique system capability and be successful at it, without diving as deeply into FPGA anatomy as was previously required. Those unfamiliar with PR terminology will find a cheat sheet on the lingo in Table 1.

Some Definitions and Requirements

While implementing PR in a design is largely automated with the latest releases, ISE® 12.1 or newer, of Xilinx design tools, the innovation involved in determining where to use partial reconfiguration is still very much a manual process. Only the ingenuity of a designer is able (at this point in tools evolution) to see where to apply PR and how to manage the overall system to

Reconfigurable partition (RP)	Design hierarchy instance marked by the user for reconfiguration; a physical area of the FPGA that will be configured
Reconfigurable module (RM)	A specific instance of the logical design that occupies the RP
Static logic	All of the logic in the design that is not reconfigurable
Configuration	A full design consisting of static logic and one RM for each RP
Partition pins	Pins at ports on a partition that interface between static and reconfigurable logic
Proxy logic	LUTs inserted automatically on each partition pin to anchor connections between static logic and the RMs
Configuration frame	Smallest addressable segment of configuration memory space
Internal Configuration Access Port (ICAP)	Internal connections to the SelectMAP interface for FPGA configuration
Bottom-up synthesis	Synthesis of individual partitions with no optimizations across boundaries, resulting in a separate netlist for each partition
Partition	A logical section of a design at a hierarchy boundary
Reconfigurable logic	Any logic that is part of a reconfigurable module

Table 1 – Talking the talk: a PR glossary

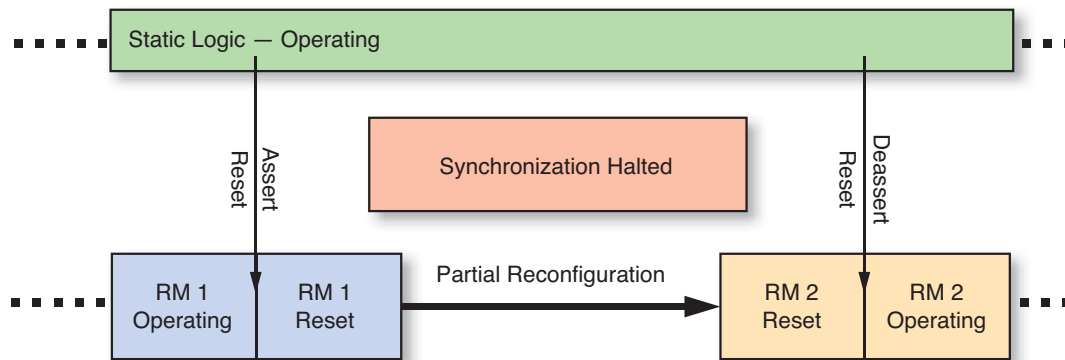


Figure 2 – Synchronization best practice

use it successfully. Some considerations and guidelines include:

- What modules can or should be “swapped” using PR?
- What is the impact of swapping on the state of the balance of the design and what steps are needed to make the swapped logic function properly (isolation/decoupling requirements, and how do I manage those in my system)?
- What is the impact on performance (timing relationships, etc.) and on verification?
- How am I going to manage the PR process, including storing and delivering the configuration files? How concerned am I about the reliability of the PR process?
- What constraints does PR impose on clocks and clock resources?

Let’s examine each of these questions briefly as we prepare to use PR in an actual design.

What Modules Can Be Swapped Using PR?

The key for using PR is to identify portions of the design that do not need to be active all the time in all uses of the FPGA design. For the most part, you can identify these sections simply by looking at a

high-level block diagram or function list and asking a few basic questions:

- Is this piece of functionality dependent on some external event, option, user command, etc. that initiates its use separate from some other, similar function?
- Would the implementation of this function be the result of a decision such as “If X then Y” or “If X then Y else Z,” or perhaps a case statement?
- Are there large sequential processing functions in the design that do not overlap? (Even in the case of some overlapping functions, alternating functions vs. purely sequential ones might be candidates for swapping.)
- Are there parts of the design that are never going to be operating at the same time? This is only obvious to someone familiar with the overall system architecture, but is one of the most basic uses of PR dating back to the beginning of FPGAs.
- Conversely, is there a portion of the logic that should be locked in place and never changed or reconfigured? This might serve as a base static-logic set, with all else potentially considered for PR.

- Some end equipment constraints might force a startup-time constraint on the FPGA. For large FPGAs it might be possible to configure only the minimal portion required for startup and have the remainder as one or more reconfigurable modules (RMs) that get loaded after startup is complete. PCI Express® implemented in a large Virtex device is a candidate for PR in this mode of operation. (See Xilinx Answer Record 35380 for details.)

Using PR in a design should be considered as akin to “hot-swapping” a device or board in a system. As the reconfigurable partition (RP) is reconfigured, routing resources, logic functions, RAM blocks and so forth will be changing state. The designer must consider this and use implementation techniques that will prevent any spurious data within the RM from affecting operation of the static logic while the RP is being configured. Best practices include forcing the RP into a reset state prior to starting reconfiguration, and holding it in reset until after reconfiguration has been completed and sufficient time or clock cycles have occurred so that it will emerge in a known-stable state. Figure 2 illustrates this practice. In addition, we strongly suggest that you place isolation registers in both the static logic and the RM at the boundaries for all inputs and outputs.

Impact on Verification

Clearly, the addition of isolation registers at the boundaries between static logic and RMs may add cycles of latency to the overall operation of the combination of the static logic and each RM that gets swapped into the RP. However, these registers will also make timing closure easier. Also, the additional constraints of putting boundaries on where the RM logic can be located may cause some decrease in the maximum operating speed and achievable density.

Only the designer can make the decision about the impact of these trade-offs to the overall design, or the propriety of eliminating isolation registers on a case-by-case basis, for example. Because it is just another FPGA design, you must manage the logic inside the RM using the same techniques to achieve the desired system performance.

You must verify the combination of the static logic and each RM as a unit, just as if it were all static logic. Where verification becomes more complex and the current

state of tools does not help is in the transition from one RM to another, and the overall system operation during that process. At this time, there is no means to simulate the FPGA during the reconfiguration process. Therefore, you can only do verification up to the point where a reconfiguration is initiated, and can only start from the point where reconfiguration has been completed going forward.

You can emulate the process of partial reconfiguration by using a code wrapper in your simulation to swap out which RM is being used, with the appropriate time pause for the reconfiguration to complete and the new RM to become active. This will be much easier and yield more realistic results if you follow the reset and isolation register guidelines.

Reconfiguration is a subset of complete device configuration and uses essentially the same hardware and techniques. While the tools flow will properly produce the appropriate configuration files, the designer must figure out what method of programming is appropriate for the system

being designed, and what precautions or safety measures are required based on the nature of the end equipment. The methodology section later in this article discusses this issue further, but the designer must consider it when contemplating use of PR in the overall system design. Biometrics, software-defined radio, cryptography, high-performance computing and networking are among the system applications beginning to use PR today.

Methodology and Tools

Standard tools for Xilinx FPGA designs have advanced to the point of being completely applicable to PR designs. You will need a separate license from Xilinx to enable the various aspects of the tools that are specific to PR, so talk to your Xilinx representative if you want to try them out. The design flow is built around the PlanAhead™ tool. If you are not familiar with PlanAhead, now is the time to sign up for a class to become expert with it.

You implement a design that will take advantage of PR just like a regular design,

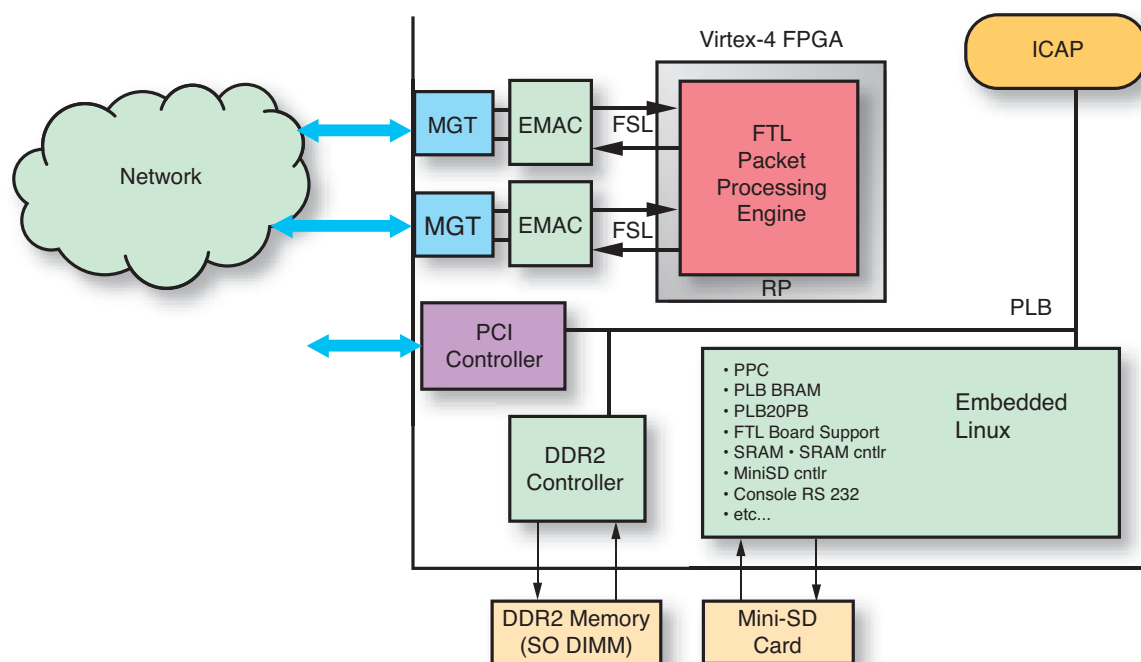


Figure 3 – Design example block diagram

but you must use a “bottom-up” synthesis flow. This simply means that no optimizations will be done across the boundaries between the static logic and the RMs that will be swapped. To accomplish this, each of the partitions will be a separate synthesis project with its own separate netlist.

The general steps for a design using PR are:

1. Set up the design structure, deciding on static vs. reconfigurable logic. Synthesize all netlists, define RMs and create partitions. Then, assign RM netlists to the appropriate RPs.
2. Provide the proper constraints for each RP based on the assigned RMs.
3. Run the PR-specific design rule checks (DRCs) that are in PlanAhead.
4. Place and route, remembering that each configuration is a complete design (static logic and one netlist for each RP). Create a “golden reference” for the static logic and iterate to close timing on all combinations of static logic and RMs.
5. Create the bit files.
6. Test the design.

Design Example

We will present an actual design to show the steps involved in implementing PR in the real world. The design uses a Xilinx XCV4FX60 FPGA to process network packets that are streaming into it from a Gigabit Ethernet interface. Two sets of operations are to be performed on the packets based on input from the user. In case 1, packets are inspected for Ping packets. When a Ping is received, the system immediately returns it to the sender with the proper MAC and IP address reversal and CRC. We did this design in VHDL.

In case 2, the packets are similarly received and inspected, but here the system is looking for UDP packets. When it receives one, it performs MAC and IP address reversal and proper port assignment before returning it to the sender with the corrected CRC. We did this design in Impulse C to show that the RM

logic can come from different sources; only the synthesized netlist is important. In both cases, turnaround time from receipt to response is critical, so all of the packet manipulation is done immediately after receipt by the MAC. Figure 3 shows a block diagram of this system.

The hardware for this example uses a Virtex-4-based FPGA accelerator card from Faster Technology (FTL): the P-6 card, with a XCV4FX60 FPGA. SFP modules provide direct access to the FPGA from Gigabit Ethernet connections of either copper or fiber-optic media. The P-6 is equipped with a robust embedded Linux system, which greatly simplifies the implementation of this example, as well as providing a means to demonstrate use of the Internal Configuration Access Port (ICAP) block for partial reconfiguration from an embedded processor.

The first step, of course, is the creative one of deciding what logic will be in the static area and what logic will be in the one or more modules that will be swapped into or out of the static design. Once you have settled on that basic structure, then you can use the design flow to implement those decisions.

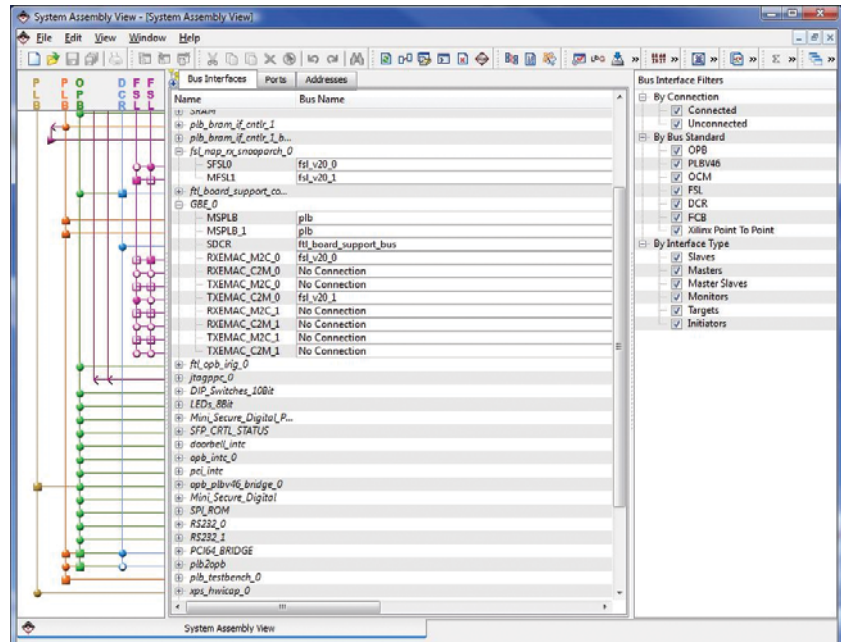


Figure 4 – EDK view of the design example

For our example design, the embedded Linux system with the PCI interface, the basic Gigabit Ethernet MACs and the ICAP interface will be in the static logic. Each of the packet-processing engine implementations will be in a separate RM, since they are never used at the same time. While this is a simple example of mutually time-exclusive functions, it is readily extensible to a wide variety of system uses.

The netlist structure and the physical partitions on the device mirror each other. Because the netlist will determine the size of RP needed, it is generally best to synthesize all of the modules that are RM candidates to determine the logic resources needed before deciding on the partition sizes and shapes.

In the example design, the P-6 card supplies the elements of the embedded Linux. The design uses a number of special IP blocks supplied by FTL, including a PCI interface, a dual Gigabit Ethernet MAC wrapper around the Xilinx hard macros and some special IP blocks to simplify the integration of user IP into the overall Linux system. Because the PR flow deals with the netlists, the

blocks in the system can come from various sources and flows.

We created the simple Ping-return RM in VHDL as a separate ISE project. We developed the more complex packet-processing block as a module in C, using the Impulse C tools from Impulse Accelerated Technology. Figure 4 shows the EDK view of the “golden” implementation with the UDP response design in place. The FSL bus connects the RM to the static logic in this design.

Once you have synthesized all the netlists, create a project within PlanAhead and add the netlists to it; PR designs are based on the netlist, not the HDL code. Create the partitions that will contain each RM or group of RMs by selecting the netlist from the project tree (right click on it, and select “Set Partition”). This will bring up a wizard to walk you through the process. One of the options will be to make the partition reconfigurable. Partition boundaries are not required to align with reconfigurable frame boundaries, but you will achieve the most efficient design results when they are aligned. Frame boundaries and the resources in each frame are different for different FPGA families, so be sure to check the documentation for the family you are using.

Since clocking resources cannot be partially reconfigured, all global clock nets that an RM might use must be routed to the RP that the RM will occupy. Routing congestion can result if many clocks are used in multiple clock regions. You can minimize this effect by reducing the number of clocks that an RP requires and the number of clock regions it spans. In general, short and wide regions are better than tall and narrow ones.

You may also use regional clocks. They may cross the partition boundary in some generations of FPGAs, but at the cost of using general-purpose routing instead of the dedicated clock resources. You may use BUFIOs and I/O clocks in PR partitions, but they may not cross partition boundaries. Figure 5 shows the RP (the white-bounded area) that will contain either of the two RMs in our example design.

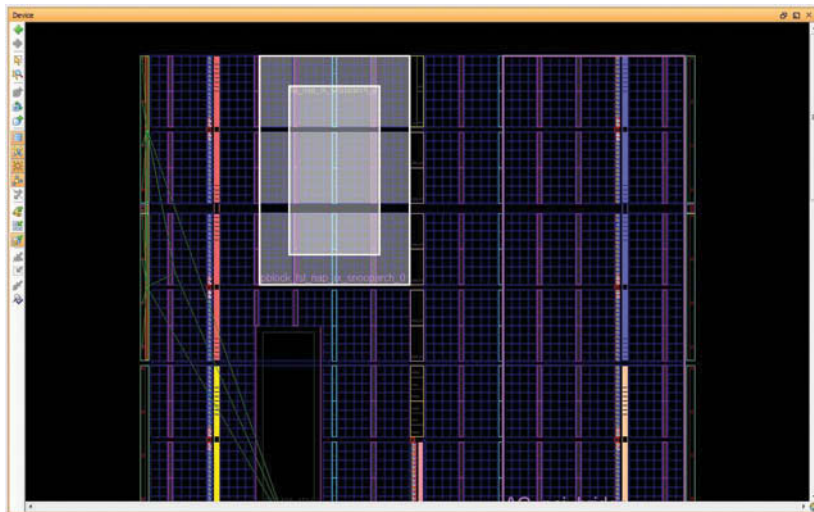


Figure 5 – Reconfigurable partition defined

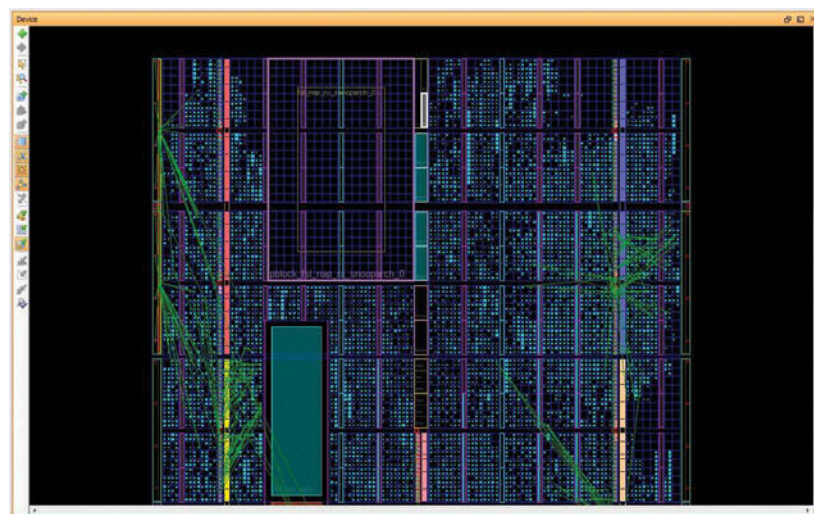


Figure 6 – Design implementation with black-box RP

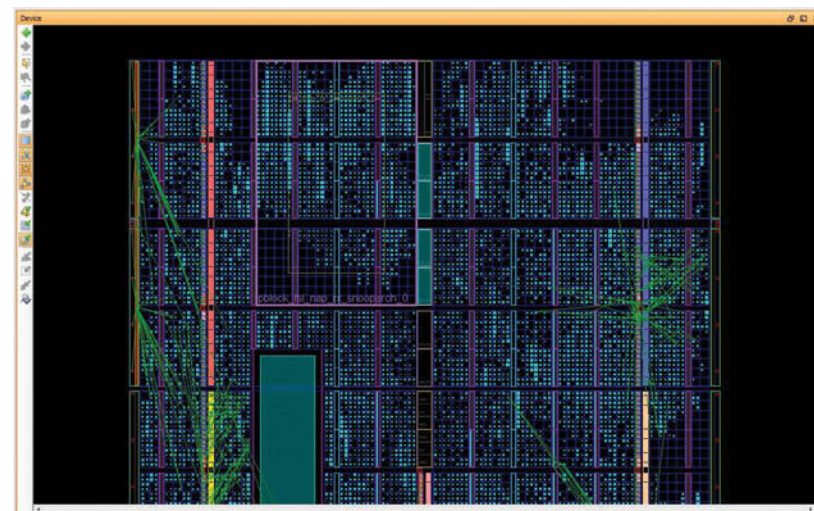


Figure 7 – Design implementation with UDP response

Assign each RM to the RP that it will occupy by selecting the partition in the netlist tree, right clicking on it and selecting “Assign Reconfigurable Module.” This will allow you to choose the netlists of the RMs that will occupy that RP.

Constraints, Placing and Routing

Constraints for the static-logic portion of the design are propagated down to each of the RMs. In most instances the top-level UCF will be sufficient. If one or more RMs require constraints beyond that of the static-logic constraints, you can add them in this step. For example, it may be necessary to assign some specific component placement constraints to meet critical timing requirements. In general, the constraints inherited from the top level will be sufficient. Our example design required no additional constraints within the two RMs. If necessary, use the *tpsync* constraint to help constrain the paths that cross the reconfiguration boundary.

For designs that utilize PR, use additional DRCs beyond those required for a conventional design to check various aspects of the PR implementation. You can initiate these DRCs from the tools menu.

Perform an initial implementation run to get a “golden reference” for the static part of the design. This will include the static routes that cross reconfigurable partitions, as well as the proxy pins that are inside the reconfigurable partitions. Since all of the reconfigurable modules that may be loaded into an RP must contain these static elements, if you reimplement the static part of the design, you must also reimplement all of the RMs. Perform multiple implementation runs to get the rest of the RMs implemented.

To provide a test case against which to compare the two “real” designs, we did the first implementation of our example design with a “black box” for the packet-processing engine. Note that a black box for an RM will create a reset region of the FPGA; that’s a handy way to reduce the power of a device when a function is not needed. Figure 6 shows the black-box configuration, while Figure 7 shows the same static logic with the UDP packet response logic in place.

As each configuration is implemented, the place-and-route tools strive to meet the timing constraints. After each place-and-route run, timing analysis is automatically run on that configuration. Unless there are combinatorial paths connecting two RPs, nothing special should be required. If there are such combinatorial paths, use the *tpsync* constraint to break the path into two independent paths.

Create the Bit Files

Bit files are created for each combination of static logic and RMs from the implementation step just completed. Once you have a successful implementation run for enough configurations to cover all of the RMs, and have promoted the results, you can highlight one or more configurations in the design runs window, right click and select “Generate Bitstream.” This will generate a full bitstream for each selected configuration and partial bitstreams for each RM in those configurations. No special options are required.

Once the design has been completed, you need to load it into the target FPGA. For the power-on configuration, load the full bit file including all of the static logic and the first RM in each RP. In some designs, the static logic may be loaded

without any logic in any or all of the RPs. This is the “trick” used in the PCI Express case to reduce configuration load time. The overall configuration process is shortened when subsequent PR configuration load cycles are initiated.

In a partial-reconfiguration process, the FPGA already has an active configuration loaded. The DONE pin is not deasserted and no configuration memory is cleared, as this could cause glitches in logic or routes that are not being reconfigured. The FPGA is reconfigured one configuration frame at a time. Each configuration frame covers an area that is one clock region tall by one design element wide. The design elements are CLBs, BRAMs, DSP blocks and the like. As each configuration frame is loaded, it becomes active.

Delivery of the partial bit files uses standard interfaces: SelectMap, serial or JTAG configuration ports or the ICAP port. All FPGA bitstreams have CRC to check their validity. In a full configuration, CRC is checked before the device is allowed to become active to prevent the possibility of damage from internal contention. In a partial configuration, the bitstream is being loaded into an already active FPGA. Also, the logic becomes active immediately when a frame is

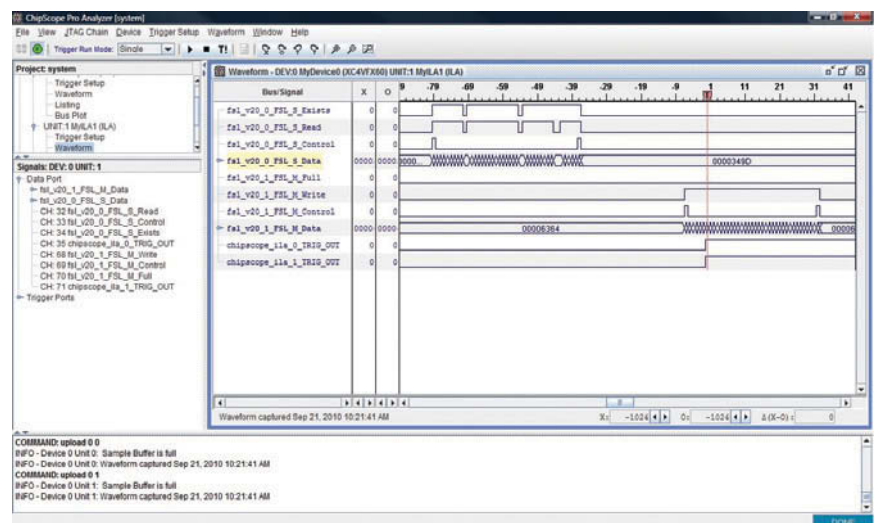


Figure 8 – ChipScope view of packet processing

loaded, but the CRC check is not done until the full reconfiguration has been completed. Therefore, if it fails the CRC check after being loaded, there is a small chance of damaging the FPGA if you fail to take corrective action quickly.

There are several options. You can reload the same bitstream in case it was just a temporary error or load a different known-good bitstream, such as an empty black box. Alternatively, reconfigure the entire FPGA or check the integrity of the bitstream inside the FPGA before it is loaded via the ICAP interface.

For that fourth option, Xilinx has a reference design that will add extra CRCs to the bitstream for each configuration frame. It then checks the CRC of each

frame before it is loaded into the ICAP. In this way, any CRC error is caught before the frame is used. (See Xilinx Answer Record 35381 for more information.)

For the example design, the P-6 card contains a Spartan® FPGA that reads the initial configuration from a MiniSD card and programs the main V4FX60 FPGA. Once the FPGA is operating, the embedded Linux system in the Virtex-4 FPGA swaps in the selected RM for the packet-processing function the user desires.

We used the ChipScope™ tool to capture operations within the logic of the FPGA to show the operation of the different packet-processing engines. Figure 8 shows the Virtex-4 receiving a UDP packet, processing it and sending it back to the source.

Straightforward Flow

Partial reconfiguration, or the notion that a system can modify itself during operation, is no longer just for explorers on the extreme edge of design. The PR design flow is straightforward enough so that mainstream design teams can use it, reaping the attendant advantages in system flexibility and adaptability, power savings and potential system cost savings.

The design example discussed here should encourage everyone to consider PR for their next design. Going forward, Faster Technology will offer a series of Web demonstrations and live examples that you can access over the Internet. Look for these at www.fastertechnology.com. 



Whether you're building complex FPGA-based systems or ASIC/ASSP prototypes, you're probably experiencing one or more of these issues:



Weeks of painful debugging in the system lab



Hundreds of long synthesis + place & route iterations



Slow simulation/multi-day regression tests



IP or RTL block mismatches between simulation and system

The GateRocket debug and verification solution cuts debug time in half by making simulation fast, accurate and eliminating many of the synthesis-place-route cycles in the lab. Problems that would otherwise slip through to the system lab are exposed in simulation, on the desktop, where they are easily and rapidly fixed. Learn why industry leading companies like Qualcomm have adopted this solution by visiting GateRocket.



Phone: +1 (781) 908-0082, Option 3

Email: RocketDrive@GateRocket.com

Visit: <http://www.GateRocket.com>

Multirate Digital Signal Processing for High-Speed Data Converters

By combining certain features of the FPGA architecture with parallel-processing techniques, your next design can attain performance beyond that of the FPGA fabric.

by Luc Langlois
Global Technical Marketing, DSP
Avnet
luc.langlois@avnet.com

Recent advances in high-speed data converters are expanding the boundaries of performance in a vast range of systems, including communications, medical and aerospace. As these trends intensify, the art of combining the high-speed analog signal chain with high-performance digital signal processing grows increasingly challenging. FPGAs have emerged as the solution of choice to meet these challenges, providing high-speed interfaces to the latest-generation data converters and digital signal processing at high sampling rates.

Selected concepts of multirate digital signal processing form the basis for efficient high-speed data converter interfaces in Xilinx® Virtex®-6 and Spartan®-6 FPGAs. By combining certain features of the FPGA architecture with parallel-processing techniques, your next design can attain performance beyond that of the FPGA fabric.

Multirate Digital Signal Processing

In many systems, it is desirable to use a fast analog-to-digital converter (ADC) to oversample at a rate f_s (Megasamples per second, or MSPS) that is beyond the minimum sampling rate defined by Nyquist, or twice the signal bandwidth f_B . For example, a digital receiver for LTE (Long Term Evolution) wireless communications may sample the analog signal at 122.88 MSPS, while the baseband sampling rate may be as low as 7.68 MSPS. Oversampling spreads the quantization noise inherent in the sampling process evenly across a wider bandwidth, leaving less noise power in the signal bandwidth of interest (Figure 1). Subsequent attenuation of the out-of-band noise through a digital filter yields an improved signal-to-noise ratio compared with a critically sampled signal.

Once the signal has been oversampled by the ADC and captured in the FPGA, it enters the digital domain. At this point there is every motivation to reduce the sampling rate prior to extracting the information content of the signal. Reducing the sampling rate of a digital signal is called decimation, and its benefits include lower FPGA resource usage, lower computation rate, lower power consumption and easier timing closure of the design. However, decimation alone will simply undo the benefits of oversampling by aliasing the quantization noise back into the signal of interest, thereby degrading the SNR anew. To find an efficient way to preserve the processing gain from oversampling, let us first review the basic theory of decimation.

Decimation Theory

The relationship between the input and output spectra of a discrete-time decimated signal without prior filtering is expressed in Equation 1 and shown in Figures 2 and 3.

Notice how decimation has the effect of overlaying D spectral replicas of the original spectrum, each stretched by a factor D and shifted at intervals of 2π rad/sample. This can cause irreparable distortion due to aliasing unless the decimator is preceded by a low-pass filter to attenuate any spectral content beyond π/D (rad/sample).

To preserve the SNR benefits of oversampling, you must combine decimation with digital filtering to attenuate out-of-band noise, a combination known as a decimation filter.

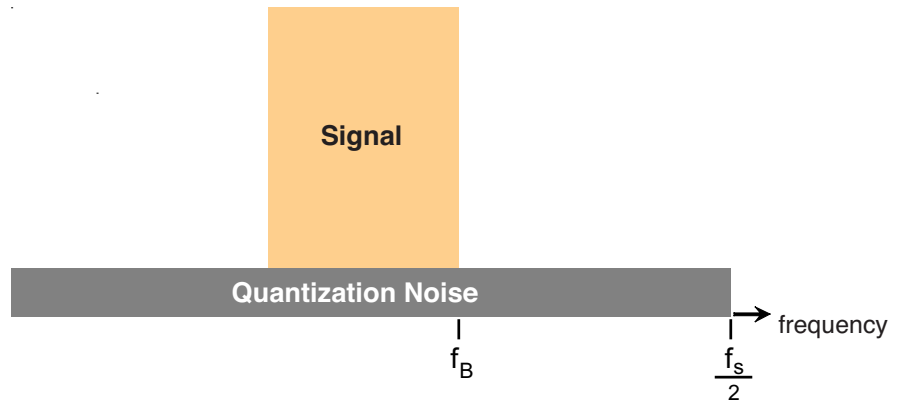


Figure 1 – Band-limited signal, 4x oversampled

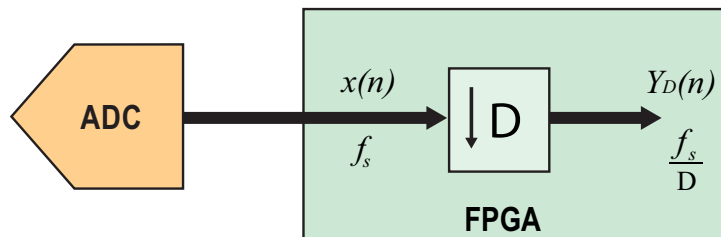


Figure 2 – Decimation without prior filtering

$$Y_D(e^{j\omega}) = \frac{1}{D} \sum_{k=0}^{D-1} X(e^{j[\frac{\omega-2\pi k}{D}]})$$

Equation 1

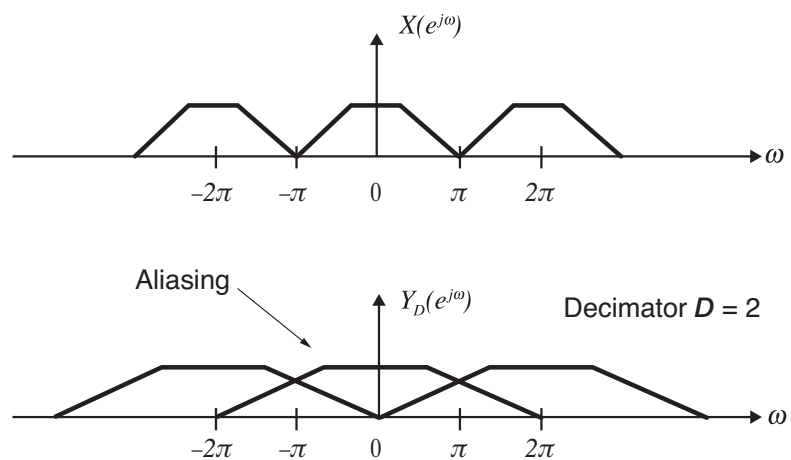


Figure 3 – Aliasing due to decimation without prior filtering

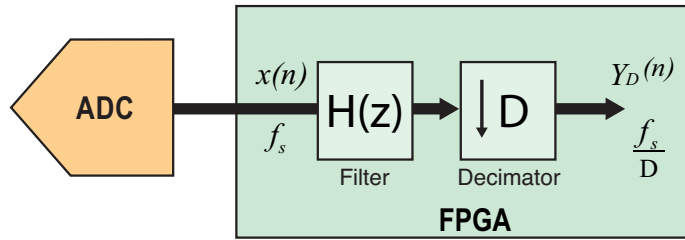


Figure 4 – Series-cascaded filter and decimator

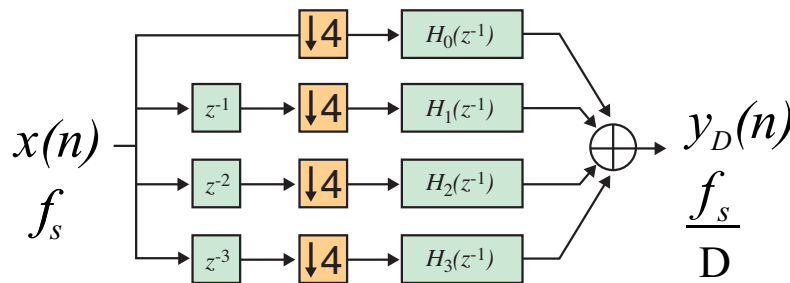


Figure 5 – 4x polyphase decimator

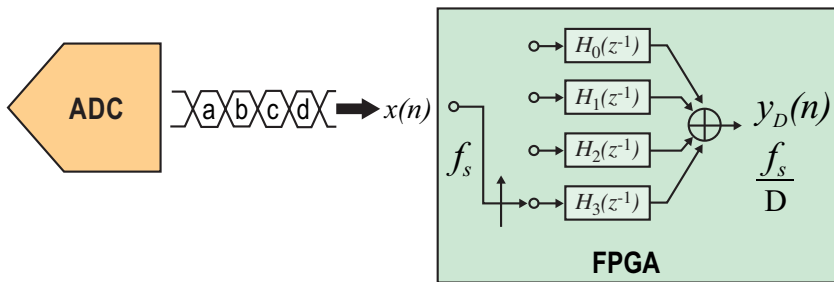


Figure 6 – Traditional 4x polyphase decimator

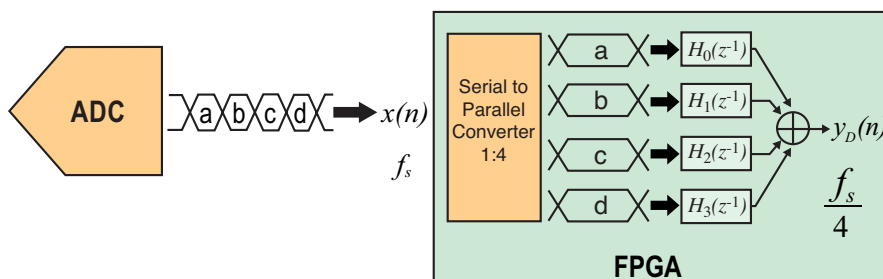


Figure 7 – 4x polyphase decimator with serial-to-parallel converter

Polyphase Filters

While the simple series cascade of filter and decimator of Figure 4 is functionally correct, there exists a more efficient decimation filter topology based on polyphase decomposition (as P.P. Vaidyanathan points out in *Multirate Systems and Filter Banks*, Prentice Hall, 1993).

A polyphase decimator comprises D subfilters $H_d(z^{-1})$, each of which contains a subset of the total set of coefficients (Figure 5). Compared with a simple series-cascaded filter and decimator, the polyphase decimator is a parallel processor with the advantage that each subfilter operates at the slow sample rate, easing timing closure of the design.

The combination of delay of increasing order followed by decimation that distributes fast incoming samples sequentially to each subfilter is often depicted as a commutator, as shown in Figure 6, for a traditional 4x polyphase decimator. It assumes that incoming samples from the ADC arrive sequentially at the input of the polyphase decimator.

Alternatively, you may concatenate incoming samples in groups of length D by a serial-to-parallel converter and present them to the input of the polyphase structure at a slower rate in block fashion (Figure 7). Such a mechanism, while functionally equivalent to the amalgam of delays and decimators, can greatly enhance the performance of polyphase filters in Virtex-6 and Spartan-6 FPGAs due to their dedicated serial-to-parallel converters.

Help from ISERDES, DSP48 Slices

Xilinx Virtex-6 and Spartan-6 FPGAs contain dedicated ISERDES serial-to-parallel converters in each I/O block. The ISERDES avoids the additional timing complexities encountered when constructing deserializers in the FPGA fabric. Designed for high-speed source-synchronous applications, the ISERDES can implement an efficient polyphase decimator by concatenating incoming samples from high-speed ADCs to support sampling rates beyond 1 GSPS. Such high data rates would otherwise overwhelm a decimation filter implemented with fabric-based deserializers.

In addition, the Virtex-6 and Spartan-6 FPGAs contain dedicated hardware known as DSP48E1/A1 slices that are optimized

for high-performance digital signal processing. Not only do the DSP48 slices avoid the additional timing complexities encountered when implementing digital signal-processing functions in FPGA fabric, they are also very power-efficient. The DSP48 slices are the ideal building blocks for the polyphase subfilters.

The optimal design employs the DSP48 slices near their maximum performance rating (DSP48E1 f_{max} = 600 MHz, 1 speed

grade; DSP48A1 f_{max} = 390 MHz, 4 speed grade). For example, the first decimation stage in Virtex-6 for a 1-GSPS ADC might use a 2x polyphase decimator in which each subfilter operates at 500 MSPS (Figure 8).

The same 1-GSPS ADC could be processed through a 4x polyphase decimator in Spartan-6, with each subfilter operating at 250 MSPS (Figure 9).

Faster ADCs can accommodate higher decimation rates, employing increased par-

allelism and slower throughput in each subfilter. Xilinx FIR Compiler IP can assist designers in making the optimal trade-offs for the polyphase subfilters, providing pushbutton implementation of differing hardware architectures such as overclocked DSP48 slices to process several coefficients in time-multiplexed fashion.

Polyphase Interpolator

There are situations that require an increase in the sampling rate, known as interpolation. For example, a cable modem typically employs an FPGA-based digital upconverter to interpolate the baseband data up to a faster sampling rate suitable for driving a high-speed digital-to-analog converter (DAC). The motivation here stems from the fact that the higher the DAC sampling rate, the greater the separation in the frequency domain between spectral images at the output of the DAC. This eases the task of post-DAC analog filtering, yielding an improvement in signal-to-noise ratio.

As was the case for decimation, an interpolation filter based on a polyphase structure has advantages over a simple series-cascaded interpolator followed by a filter. That's because each subfilter operates at the slower computation rate, easing timing closure. When combined with the dedicated parallel-to-serial converter (OSERDES) in each Virtex-6 or Spartan-6 I/O block, you can construct high-performance interpolation filters with minimal FPGA fabric. Figure 10 shows a 4x polyphase interpolator in a Spartan-6 driving a DAC at 1 GSPS.

Ripe for Customization

We have shown an efficient design methodology for high-performance polyphase digital filters. Now you can customize the critical processing stages of your high-speed data converter designs by harnessing the unique features of Virtex-6 and Spartan-6 FPGAs for multirate digital signal processing.

The techniques described in this article are part of an Avnet Speedway design workshop titled "FPGA-Based System Design with High-Speed Data Converters," to be held across the globe this fall. For more information, please go to www.em.avnet.com/adcspeedway.

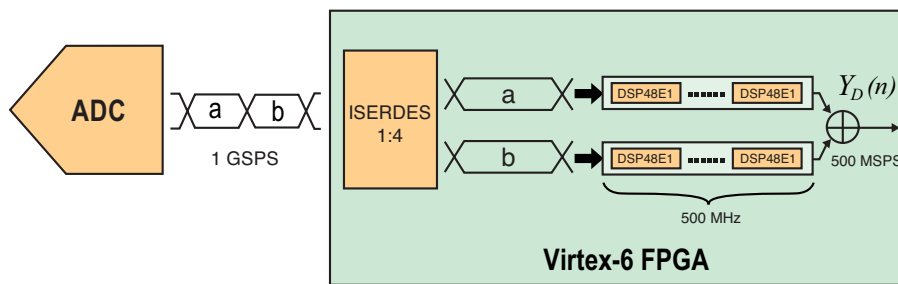


Figure 8 – 2x polyphase decimator with serial-to-parallel converter in Virtex-6

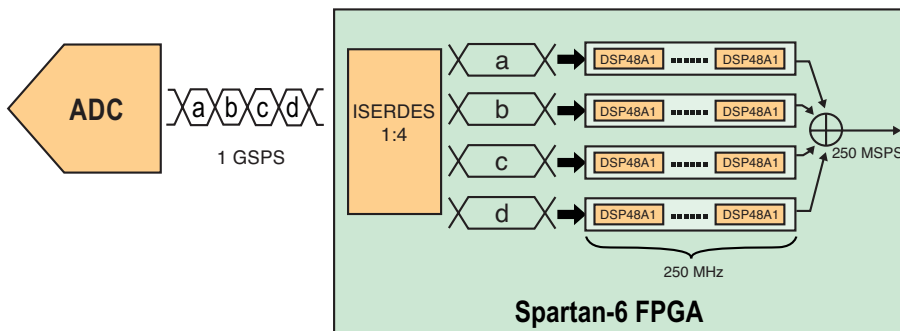


Figure 9 – 4x polyphase decimator with serial-to-parallel converter in Spartan-6

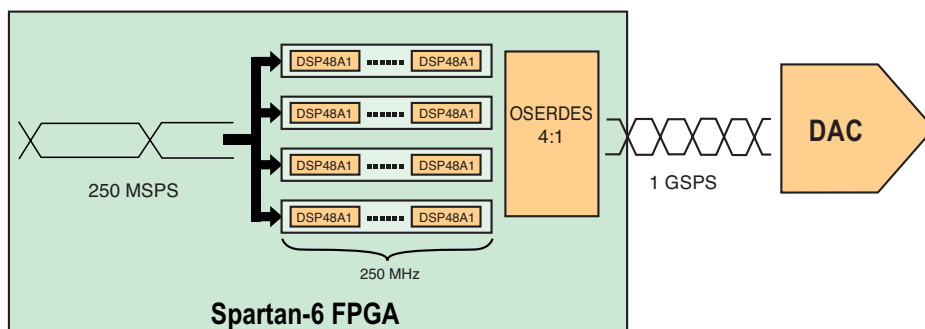


Figure 10 – 4x polyphase interpolator with parallel-to-serial converter in Spartan-6

How to Speed Sine Calculations for Your Processor

If software changes don't deliver the speed you need, it's easier than you might think to add hardware accelerators to your design.



by Karsten Trott
Field Application Engineer
Xilinx GmbH, Munich, Germany

There are many ways to calculate a sine value for single-precision floating-point numbers, but one of the most powerful is to add hardware accelerators. That was our conclusion after exploring a number of approaches to providing a good, fast and efficient solution when a customer application required this type of sine calculation.

To determine which implementation works best for your application, first start by profiling your code base to find out which function needs some improvement. Then, because software is easier and quicker to change than hardware, check to see if soft-

ware changes will deliver the higher speed you need (sometimes they do). But if you still require more performance, then think about implementing parts of the algorithm in hardware. With hardware acceleration, you can easily outperform any microcontroller or DSP on the market.

To get a feel for the process, let's walk through a real-world case of how we handled a military application that needed a sine calculation for single-precision float numbers. For cost/performance reasons, the customer had already chosen a Spartan®-6 FPGA with an embedded MicroBlaze® that functioned as the main system controller. The software algorithm that handles the sine calculation should run on that MicroBlaze.

The customer's algorithm heavily utilized floating-point operations. Due to the complexity of the algorithm, changing over to fixed-point arithmetic was not an option. Also, the customer tried to avoid over- and underrun situations that can happen when fixed-point arithmetic is being used.

The customer was aware that the MicroBlaze IP provides two types of floating-point units (FPUs), and had already chosen the extended (as opposed to the basic) version to accelerate the algorithm. However, this choice makes it impossible to use the math emulation libraries that are delivered together with the EDK as part of the GNU tool chain. The software emulation routines from the math library are

pretty slow; you should avoid them under all circumstances for performance-critical parts of an algorithm.

The customer knows also that both versions of the MicroBlaze FPUs can process only single-precision data, but not double-precision data. The customer's algorithm explicitly used float precision data only. But sometimes implicit conversions are done when you start using math functions. These conversions can force your algorithm to use double-precision data without being noticed.

Step 1: Analyze the Problem

Our customer had already run his algorithm, but found out that it was running slowly on the MicroBlaze processor. Upon profiling the code base, the customer found that it was the sine calculation that was causing the slowdown. The next step was to find out why this was the case and figure out what to do to speed up the processing.

The first approach involved using the standard sine function provided by the math library and to run the algorithm completely as the customer wrote it, without any modifications. The major drawback is that the math library functions are created for double-precision data only. That means the prototype of the sine function looks like this:

```
double sin(double angle);
```

But the customer would like to use it the following way:

```
float sin_val;
float angle;
...
sin_val = sin(angle);
```

This is, of course, possible and the C compiler automatically adds the required conversions for the parameter `angle` to double and to convert the result of the function call back to float value. Again, the math library functions usually execute these two additional conversion functions and even the sine calculation.

Remembering that the FPU of the MicroBlaze is a single-precision version only leads to the following execution:

```
sin_val = (float)sin((double)angle);
```

Because of the double-precision declaration of the sine function by the math library, the FPU cannot do the sine calculation. Instead, a pure-software solution is required. The drawback of this implementation is that it was quite slow and did not fit the customer's requirements.

We verified that the calculation of the sine value using double-precision data was the reason for the slow execution. First we created the assembler code directly from our executable file by using:

```
mb-objdump.exe -D executable.elf
>dump.txt
```

Checking the assembler code, we found the following line:

```
brlid r15,-15832 // 4400d300 <sin>
```

This is exactly the call to the math library for double-precision sine calculation. Then we measured a single sine calculation utilizing the math library functions. It takes around 38,700 CPU cycles.

Special single-precision functions are available for certain jobs, such as calculating the square root:

```
float sqrt_f( float h);
```

The usage of that special function avoids the conversion from single- to double-precision data and can make use of the MicroBlaze FPU.

Unfortunately, there is no such function to process the sine calculation on the FPU. At this point, we started developing several versions to speed up the calculation of the sine value so as to gain more performance.

Step 2: Create a Better Software Algorithm

Creating hardware accelerators usually takes some time and debugging effort, so we tried to avoid that approach in the first run. We discussed the performance issue with our customer to get the key parameters for the sine calculation.

The customer algorithm requires a precision of 1/100 degree resolution for the parameter `angle` of the sine calculation. And the calculated sine-value precision should be better than 0.1 percent compared with the result of the math library

function call. These are the key parameters, and the customer told us that he sometimes has to calculate multiple sine values in a row (when, for example, filling small tables before processing them).

A lookup table filled with all the values is obviously not possible due to the required size of that table. The minimum number of entries is 360,000 float values (4 bytes per value). The customer was looking for a fast solution, but it had to be size-efficient too.

Our proposed solution made use of the following equation:

```
sin(xi) with xi = x + d
results in:
sin(x+d) = sin(x)*cos(d) +
cos(x)*sin(d)
```

where `d` is a value that is always less than the smallest possible `x` value (that is, greater than zero).

What's the benefit of this solution? We need less space for the tables, but some additional calculations. The tables are split into four tables at the beginning:

```
cos(x)
sin(x)
cos(d)
sin(d)
```

Figures 1 and 2 demonstrate the resolution required for all four tables and how these values typically look. The tables show entries for only 16 values to demonstrate what we had to place into our lookup tables. We use much more in our final solution.

In fact, we used 1,024 values in each table. The smallest value for `x` results in: $360/1024 = 0.3515625$ degree. All values for `d` will be less than or equal to this number. This strategy reduces the memory consumption regarding a full-blown lookup table to 4,096 entries (4 bytes each).

The overall precision for the argument that we will reach using this approach is:

```
360/(1024*1024) = 0,000343 degree
```

And the precision is pretty good. The calculation fully utilizes the MicroBlaze FPU.

The real calculation takes some clock cycles—specifically, two *fmul* operations

and one *fadd* operation. But we needed some additional calculations. First, we have to split the argument *xi* into the two values for *x* and *d*. Then we have to read out the values from our tables, and finally we have to calculate the result using our new algorithm.

When we implemented the algorithm in software and tested it, we got an overall clock cycle count of 6,520 clocks.

To further increase the resolution we could use the following quadrant relations:

First quadrant:

$$\sin(x) = \sin(x)$$

Second quadrant:

$$\sin(x) = \sin(\pi - x)$$

Third quadrant:

$$\sin(x) = -\sin(\pi + x)$$

Fourth quadrant:

$$\sin(x) = -\sin(2\pi - x)$$

This would increase the overall resolution fourfold while keeping the tables at the same size. On the other hand, we need some additional computations to find out which quadrant we have to calculate for. There is still some room to improve the algorithm or to decrease the size of the tables (by a factor of four). We have not yet implemented that additional step.

Step 3: Optimize the Algorithm

Because the solution we have reached so far was still not fast enough for our customer, we tried to optimize the algorithm a little bit more, still keeping everything on the software side running on the MicroBlaze processor. There is one simple optimization possible, but it costs some precision. That's why we created a software model (running on the PC to get more speed) that runs over all possible values and compares the original double value from *sin()* with the sine values created by our software algorithm. We decided to run that algorithm on a standard PC because doing the comparison and calculation on the MicroBlaze takes quite a long time (remember that our MicroBlaze is running at a much lower speed than the PC).

Now we started to optimize the calculation to get the sine value:

$$\sin(x+d) = \sin(x)\cos(d) + \cos(x)\sin(d)$$

Since we used 1,024 values in each table, that means that *d* is always less than 360 degrees/1,024 steps. Or:

$$\cos(2\pi / 1024) = 0.99998$$

and this is nearly equal to 1.0.

For small values of *d*, the following equation is valid:

$$\cos(d) \approx 1.0$$

That simplifies our formula to the following equation:

$$\sin(x+d) = \sin(x) + \cos(x)\sin(d)$$

Using our PC model, we checked the precision of that new equation before we implemented it on the MicroBlaze and found out that the maximum error was

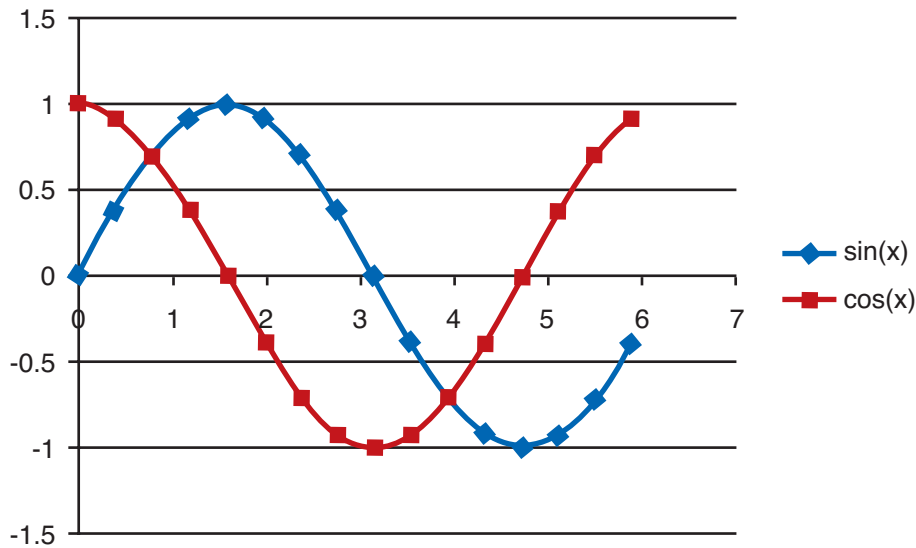


Figure 1 – Sine and cosine tables for the *x*-values, which have the range 0 to 360 degrees

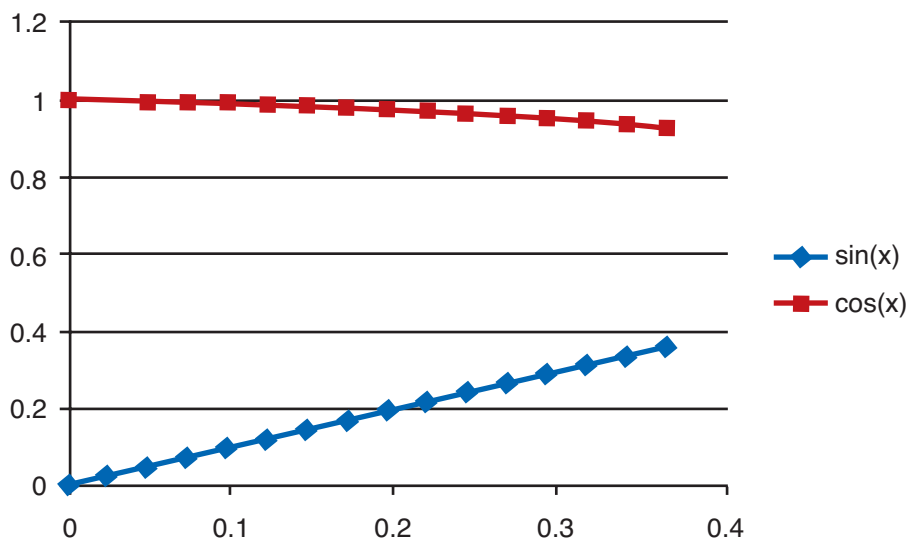


Figure 2 – Sine and cosine tables for the *d*-values, which have the range 0 to 360/16 degrees

still below the target of our customer.

Now we implemented this algorithm on the MicroBlaze as a software algorithm, still using tables with 1,024 entries each. The new algorithm requires only three tables, one less than in the previous implementation. This saves memory space—and makes time for the additional calculations.

We measured the algorithm on our hardware. It required 6,180 cycles for a single calculation.

Step 4: Further Optimizations

Another optimization that seemed possible was to convert the floating-point value of the sine calculation and use an integer argument here. The algorithm we were using allowed us to create $\sim 1E6$ different values ($1,024 \times 1,024$). An integer argument was sufficient to hold that number.

This optimization allowed us to use a much simpler calculation for the splitting of the xi value into x and d. The splitting is just a simple AND operation plus some shifting by 10 bits. The upper 10 bits of our parameter angle are xi while the lower 10 bits are d.

We again created a software model on the PC, checked it and then implemented it on the MicroBlaze processor system. This took 5,460 cycles for a single sine calculation.

Step 5: Thinking about Hardware Implementation

Even though the speed of the algorithm was much improved compared with the original calculation from the math library, the customer required a much faster implementation. But the last step above showed us a version that can be easily converted to hardware.

It requires some operations for the splitting of the xi value. In hardware, however, you can accomplish it by just a wiring of the required bits. Then we need three tables; we used inferred ROMs with predefined values calculated by our PC model and then transferred into the VHDL code of the IP. The IP reads all three tables at the same time, again saving some computation time. And finally, we require one float-MUL and one float-ADD operation. For this job, we found that the CORE Generator™ module for floating-point

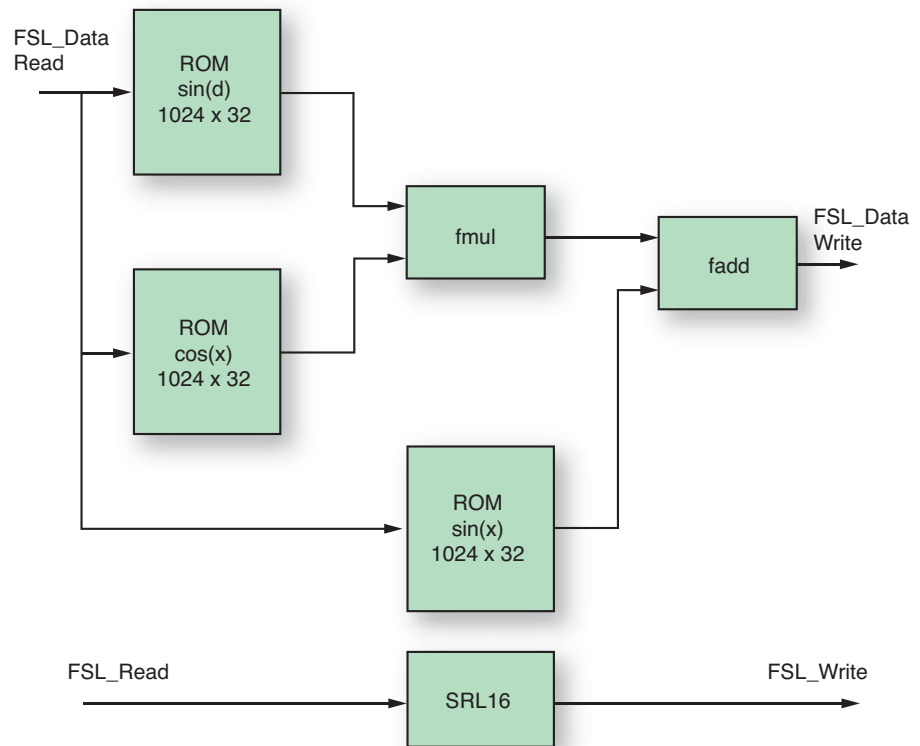


Figure 3 – Accelerator IP without pipelining

operations works pretty well.

We instantiated two of these hardware modules, a job that required some slices and some multipliers. Both cores required a latency of four to five cycles to match our timing of the design. The latency was not an issue yet and will be discussed in the next steps.

We implemented the final IP as a Fast Simplex Link (FSL) IP for the MicroBlaze. The first estimation on timing showed:

- one clock cycle to transfer the data from the MicroBlaze to the FSL bus
- one clock cycle to transfer the data from the FSL bus into FSL IP (data will be immediately read from the BRAMs when the argument of the sinus calculation is read from FSL bus, so this doesn't require any clock cycles)
- four clock cycles to process the MUL operation ($\cos(x) \cdot \sin(d)$)
- one clock cycle to store the result of that equation in a register

- four clock cycles to process the ADD operation
- one clock cycle to send the data back to the FSL bus
- one clock cycle for the MicroBlaze to read the data from the FSL IP

Please note that the data for the argument has to be stable over the whole time, assuming you do not use any additional pipelining (we will discuss this in the next step). That means that the MicroBlaze can only request a single sine calculation and has to read the value after at least 13 clock cycles before it can ask for another calculation.

Thus we estimated that it would take 13 clock cycles to run that implementation. And of course, a few more clock cycles are required for handling the function calls on the software plus some additional operations.

We implemented this IP in less than a day simply by putting some standard

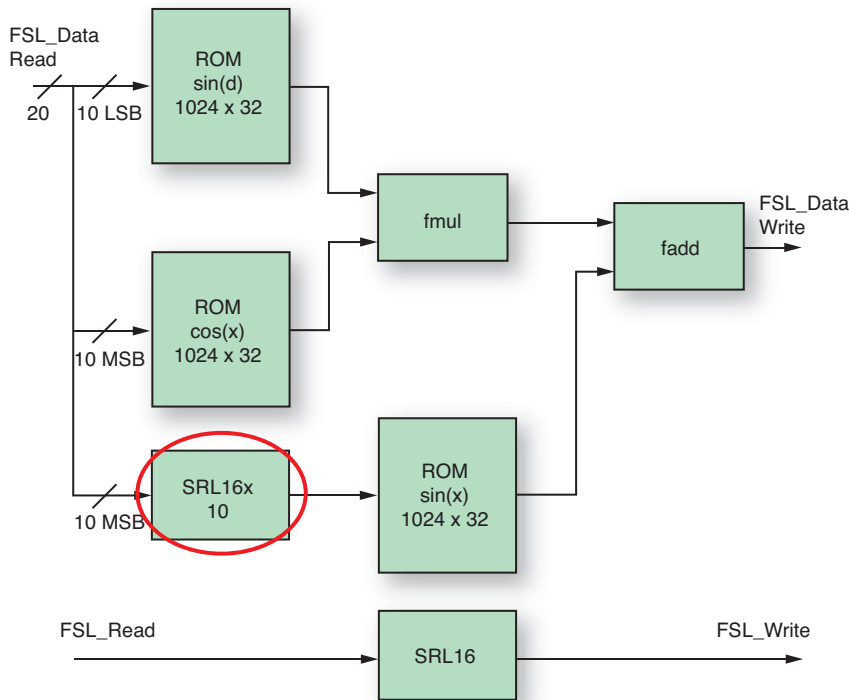


Figure 4 – Accelerator core with pipelining

clocks together, and measured the algorithm in hardware. The whole algorithm (mixed software/hardware) took 360 clock cycles (including all function calls). This was a big step forward, but still not enough to meet the customer's needs.

We used one SRL16 to delay the writing of the signal until our accelerator IP was processing all data.

The algorithm was now running in parallel to our MicroBlaze—but it can only calculate one value at a time.

Step 6: Adding Pipelining and Adapting Customer Code

At this point in the design, we started to add pipelining to our core. The CORE Generator modules for float-ADD and float-MUL are already pipelined, so there was nothing for us to do here. The first version required that the argument stay constant over time until the calculation is done.

After a new calculation starts (argument data arrived inside the FSL IP), two BRAMs are immediately read and the float-MUL is executed. The result of that operation is valid after few clock cycles.

Our argument x_i of $\sin(x_i)$ is a 20-bit-wide integer number, split into x and d . Therefore, we had to delay the MSB part x of our argument x_i a few clock cycles to read the content of the BRAM, storing x_i , to match to the result of the MUL operation. We used a few SRL16 elements, 10 in all, for our 10-bit-wide number, eating up to 10 LUTs (but due to LUT combining in the Spartan-6, it will require only five LUTs thanks to that device's wider LUT6 structure). The final effort was quite minimal for our implementation. The additional SRL16x10 bit is marked with the red circle in Figure 4.

Then we changed our FSL bus FIFOs using the EDK wizards to allow storing of multiple values (we decided that storing eight values was sufficient for our purposes, but you can easily add more if needed).

That means that our customer can request up to eight values before he even asks for the first result. This was sufficient for our customer's current needs, but in case he wants to request more sine values, he can easily expand the FIFO buffers to larger values.

We discussed this new approach with the customer and found out that it is fur-

ther possible to split the sine calculation into two parts:

1. Requesting of the sine calculation (*fslput* operation)
2. Requesting of the result of the sine calculation (*fslget* operation)

Because we have a fixed latency in the calculation, if both of these operations execute immediately one after the other, the MicroBlaze will stall and wait until the FSL IP has processed the request.

If we can split the two operations—which was possible in the customer algorithm—then we can further improve the overall speed of the calculation.

With the additional pipelining, the final code for execution on the MicroBlaze looks like this:

```
putfsl(arg1,fsll_id);
putfsl(arg2,fsll_id);
putfsl(arg3,fsll_id);
putfsl(arg4,fsll_id);
putfsl(arg5,fsll_id);
putfsl(arg6,fsll_id);
putfsl(arg7,fsll_id);
putfsl(arg8,fsll_id);
...
getfsl(result1,fsll_id);
getfsl(result2,fsll_id);
getfsl(result3,fsll_id);
getfsl(result4,fsll_id);
getfsl(result5,fsll_id);
getfsl(result6,fsll_id);
getfsl(result7,fsll_id);
getfsl(result8,fsll_id);
```

This put us into a very good position. The core is fully pipelined and allows the splitting of the two calls of the sine operation. The latency of the IP core is still there, but not visible anymore. The MicroBlaze will no longer stall and wait for a pending IP calculation, and this improves the overall performance.

The customer agreed to change his code accordingly, which was a minor effort for him. By using C macros instead of function calls, we were being able to in-line all the required calls into the code base.

The final implementation of that algorithm took only four clock cycles for a

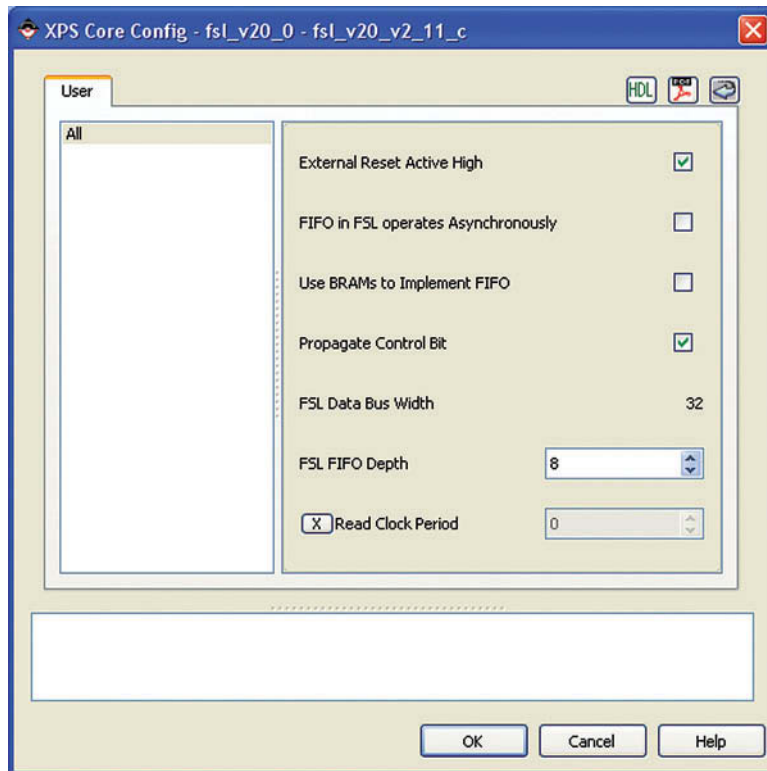


Figure 5 – The EDK enables a FIFO with a depth of eight for the FSL buses, to improve pipelining.

single calculation. The whole latency of the processing is no longer visible, but is hidden by the splitting of calling and requesting of the result. And the overall IP required a few additional BRAMs (six for our three tables) and a few additional multipliers/DSP slices plus some additional slices.

But the results are quite astonishing. Our MicroBlaze now behaves like a super-high-end processor core, but is still running at a pretty low frequency (it is now ~9,600 times faster than the original sine calculation).

Step 7: Further Optimization?

When we reached this level of implementation, our customer was satisfied with the result and we finished our work on the accelerator IP. The speed and the precision were good enough.

Of course, there is one final optimization that could be done. The algorithm can be further refined if we evaluate the $\sin(d)$ values for very small values of d :

$$\sin(d) = -d$$

where d is less than $2\pi/1024$ – means less than 0,0061359, and the overall error is less than $1E-8$ (for tables containing 1,024 values).

The final step for our algorithm could be:

$$\sin(x+d) = \sin(x) + \cos(x) * d$$

This would introduce only a very small additional error. But we could get rid of the third table.

Of course, we would have to keep the $fadd$ and $fnul$ operators.

There might be other ways to calculate a sine value for float values, but this approach shows the power of additional hardware accelerators. Our experience shows that you need not be afraid if the algorithm to be placed into hardware contains floating-point operations. 🌈

Karsten Trott is a Xilinx FAE in Munich, Germany. He holds a PhD in analog chip design and has a strong background in chip design and synthesis. You can reach him at Karsten.Trott@xilinx.com.

SPARTAN-6 OEM FPGA MODULES

THE CAN-DO EVERYTHING SoPC HARDWARE

DUAL FAST ETHERNET

MARS MX1

- Industry-standard form factor
- Quick & simple integration
- Low cost, low power FPGA
- Saves up to 120 components
- Allows for simple 4-layer carrier board hardware designs
- Microblaze reference design available for download

SPARTAN-6 LXT

THE PCIe/GiGE SIGNAL PROCESSING FPGA HARDWARE

68x30 MM SO-DIMM

MARS MX2

- Dual Multi-Gigabit Transceivers
- PCIe 1X Endpoint
- Gigabit Ethernet PHY
- Single 3.3V Supply Voltage

ALSO AVAILABLE:

- CARRIER BOARD DESIGN SERVICES
- CUSTOM MODULE CONFIGURATIONS

FPGA DESIGN SERVICES

- Algorithms
- HDL Design
- Hardware
- Software
- Software Defined Radio
- Digital Signal Processing
- Connectivity/Networking
- Embedded Processing

CHALLENGE US TODAY!

WWW.ENCLUSTRA.COM

FPGA Design Center

Take Advantage of a Simplified Design Experience

by Tom Thomas
Training Program Manager

Rhett Whatcott
Customer Training Development Manager
Xilinx, Inc.

Over the last year, Xilinx has invested heavily in making the design experience easier for our customers. Xilinx® has developed a unified silicon platform upon which Artix™-7, Kintex™-7 and Virtex®-7 devices are built, allowing for the greatest degree of flexibility and scalability for customers. Xilinx is also working toward a unified design methodology, integrating the FPGA design flow into one, seamless design flow.

With the recent ISE® Design Suite 12.3 release, Xilinx takes a bold leap in this journey. In this version, users will see two significant changes. First, ISE Design Suite 12.3 will begin supporting the AMBA® AXI4 Stream Protocol Specification. This will allow designers to leverage a wide variety of existing IP in their FPGA designs, thereby reducing time-to-market. Second, Xilinx has invested heavily in the PlanAhead™ design cockpit to provide a simpler, more consolidated view into your design. This cockpit lets you manage your entire design from within PlanAhead.

Enabled by a new and improved graphical user interface, PlanAhead now offers an RTL-to-bitstream pushbutton task-based flow with three main steps: synthesis, implementation, and program and debugging. The tool has all the necessary features to efficiently manage projects by adding HDL file sources, IP cores, constraints and other resources. You can visualize the design at any flow stage and any time for analysis, including a clear view of design resources and timing closure.

The AXI4 interconnect, meanwhile, is a point-to-point interface developed to address system-on-chip performance challenges. It supports pipelining, multiple clock domains and data upsizing and downsizing. AXI4 also includes features such as address pipelining, out-of-order

completion and multithreaded transactions. All of these features, when taken together, allow much higher-performance systems than those built over other bus architectures. When converted to AXI4, Xilinx's embedded-platform targeted reference design provides twice the bandwidth of the previous targeted reference design.

Taking full advantage of these changes requires a bit of a shift in the way users think about their design. Xilinx has invested heavily in developing training to help designers do just that.

Aligning Training to the ISE Design Suite 12.3 Release

In step with the focus of the ISE 12.3 release, Xilinx customer training is updating the following courses to align with the further progress of the PlanAhead design cockpit and the AXI4 IP preproduction release.

- **Essential Design with the PlanAhead Analysis and Design Tool:** Learn to manage design performance, plan an I/O pin layout and implement using the PlanAhead software tool.
- **Advanced Design with the PlanAhead Analysis and Design Tool:** Advanced capabilities of PlanAhead software help you to increase design performance and achieve repeatable performance.
- **Embedded-Systems Design:** We have updated the embedded design courses to include the new AXI interface, while retaining the option to also utilize the PLB bus. This course teaches experienced FPGA designers to develop embedded systems using the Embedded Development Kit (EDK). The lectures and labs also cover the features and capabilities of the Xilinx MicroBlaze® soft processor and the PowerPC® 440.
- **Advanced Features and Techniques of Embedded-Systems Design:** This course provides developers with the necessary training to develop complex

embedded systems and enables them to improve their designs by using the tools available in the EDK.

- **Embedded-System Software Design:** This course introduces you to software design and development for Xilinx embedded-processor systems. You will learn the basic tool use and concepts required for the software phase of the design cycle, after the hardware design is completed.

Additionally, we have updated the following classes.

- **Xilinx Partial Reconfiguration Tools and Techniques:** This course demonstrates how to use the ISE, PlanAhead and EDK software tools to construct, implement and download a partially reconfigurable (PR) FPGA design.
- **Designing a LogiCORE® PCI Express® System:** Updated to utilize the connectivity targeted reference design as the primary lab design, this class will give you a working knowledge of how to implement a Xilinx PCI Express core in your applications.
- **Labs targeting Virtex-6 demonstration boards (ML605)** have updated content.
- **Free recorded E-learning content** now includes the latest devices and software (available at <http://www.xilinx.com/training/>).

A Global Push

Each of these classes is available worldwide today. There are 26 Xilinx Authorized Training Providers operating in nearly 50 countries (see chart). Space is limited, so sign up today. To view the new, interactive worldwide training schedule, go to <http://www.xilinx.com/training/worldwide-schedule.htm>. Otherwise, contact your local ATP for more details: <http://www.xilinx.com/training/atp.htm>. 

AUTHORIZED TRAINING PARTNER	CONTACT	COUNTRY/REGION(S) SUPPORTED
Xilinx Training Worldwide	www.xilinx.com/training	Worldwide
AMERICAS	registrar@xilinx.com	
Anacom Eletrônica	www.anacom.com.br	Brazil
Bottom Line Technologies	www.bltn.com	Delaware, District of Columbia, Maryland, New Jersey, New York, Eastern Pennsylvania, Virginia
Doulos	www.doulos.com/xilinxNC	Northern California
Faster Technology	www.fastertechology.com	Arkansas, Colorado, Louisiana, Montana, Oklahoma, Texas, Utah, Wyoming
Hardent	www.hardent.com	Alabama, Connecticut, Eastern Canada, Florida, Georgia, Maine, Massachusetts, Mississippi, New Hampshire, North Carolina, Rhode Island, South Carolina, Tennessee, Vermont
Multi Videia Designs (MVD)	www.mvd-fpga.com	Argentina, Brazil, Mexico
North Pole Engineering	www.npe-inc.com	Illinois, Iowa, Kansas, Minnesota, Missouri, Nebraska, North Dakota, South Dakota, Wisconsin
Technically Speaking	www.technically-speaking.com	Arizona, British Columbia, Southern California, Idaho, New Mexico, Nevada, Oregon, Washington
Vai Logic	www.vaillogic.com	Indiana, Kentucky, Michigan, Ohio, Western Pennsylvania, West Virginia
EUROPE, MIDDLE EAST & AFRICA (EMEA)	eurotraining@xilinx.com	
Arcobel Embedded Solutions	www.arcobel.nl	The Netherlands, Belgium, Luxembourg
Bitsim AB	www.bitsim.com/education	Sweden, Norway, Denmark, Finland, Lithuania, Latvia, Estonia
Doulos	www.doulos.com/xilinx	United Kingdom, Ireland
Inline Group	www.plis.ru	Moscow Region
Logtel Computer Communications	www.logtel.com	Israel, Turkey
Magnetic Digital Systems	www.magneticgroup.ru	Urals Region
Mindway	www.mindway-design.com	Italy
Multi Video Designs (MVD)	www.mvd-fpga.com	France, Spain, Portugal, Switzerland
Pulsar Ltd.	pulsar.co.ua/en/index	Ukraine
Programmable Logic Competence Center (PLC2)	www.plc2.de	Germany, Switzerland, Poland, Hungary, Czech Republic, Slovakia, Slovenia, Greece, Cyprus, Turkey, Russia
SO-Logic Consulting	www.so-logic.co.at	Austria, Brazil, Czech Republic, Hungary, Slovakia, Slovenia
ASIA PACIFIC	education_ap@xilinx.com	
Active Media Innovation	www.activemedia.com.sg	Malaysia, Singapore, Thailand
Black Box Consulting	www.blackboxconsulting.com.au	Australia, New Zealand
E-elements	www.e-elements.com	China, Hong Kong, Taiwan
Libertron	www.libertron.com	Korea
OE-Galaxy Co., Ltd.	edu-electronic@oegalaxy.com.vn	Vietnam
Sandeepani Programmable Solutions Pvt. Ltd.	www.sandeepani-vlsi.com	India
SymmId	www.symmId.com	Malaysia
WeDu Solution	www.wedusolution.com	Korea
JAPAN	education_kk@xilinx.com	
Avnet Japan	www.jp.avnet.com	Japan
Paltek	www.paltek.co.jp	Japan
Shinko Shoji Co., Ltd.	xilinx.shinko-sj.co.jp	Japan
Tokyo Electron Device Ltd.	ppg.teldevice.co.jp	Japan

Xilinx Tool & IP Updates

In early October, Xilinx made available the ISE® Design Suite 12.3. This new version features the rollout of several intellectual-property (IP) cores that meet the AMBA® AXI4 specification for interconnecting functional blocks in system-on-chip (SoC) design. It also introduces several productivity enhancements to the PlanAhead™ tool, most notably a new RTL-to-bitstream design flow that has a new and improved user interface and project management capabilities. ISE Design Suite 12.3 also includes intelligent clock-gating support for reducing dynamic power consumption in Spartan®-6 FPGA designs. The download is available at <http://www.xilinx.com/support/download/index.htm>.

New Device Support

ISE Design Suite 12.3 adds support for the Spartan-6 Lower Power (-1L) and XA Spartan-6 production devices (via the 12.3 speed files patch); XA Spartan-6 -3 speed grade; and defense-grade Virtex®-6Q Devices (I-Grade only).

ISE Design Suite: Logic Edition

Ultimate productivity for FPGA logic design

Latest version number: 12.3

Date of latest release: October 2010

Previous release: 12.2

URL to download the latest release:

www.xilinx.com/download

Revision highlights:

The ISE Design Suite Logic Edition includes several updates, most notably the final release of ModelSim Xilinx Edition-III and Xilinx's emphasis on its ISIM simulator as its flagship simulation environment going forward. For more information, please see change notice XCN 10028 at http://www.xilinx.com/support/documentation/customer_notices/xcn10028.pdf.

Project Navigator: Integration is improved thanks to support for custom SmartXplorer strategy files from within Project Navigator, the ability to save only "N" best results in order to efficiently manage disk space, additional resource-utilization data displayed in the SmartXplorer Results window and the ability to abort individual SmartXplorer runs.

In addition, the project archive now has the ability to archive sources only, along with a new option to exclude generated files when creating a project archive. The tool includes a new option to create VHDL libraries based on file directory name and will automatically add all VHDL files from a selected directory.

PlanAhead: PlanAhead boasts a number of new features, the biggest of which are an updated AXI IP core in the IP Catalog and integration with CORE Generator™; pin-planning usability improvements, including table-based constraint editing, package pin location swaps and improved visibility of configuration settings; and improved SSN prediction for Spartan-6 devices.

The tool also includes a number of project infrastructure improvements, such as the ability to launch runs without

unnecessary copying of sources, improvements to functionality for when directory trees are added as sources, manual file-ordering control for synthesis, backups to journal and log files, passing PCF files to bitgen and copy run features.

Among several GUI improvements and changes, the 12.3 version has removed on-the-fly creation of user-defined attributes and offers additional options for find-in files. Other GUI changes include the ability to open a project directly from the Getting Started page; UCF file order display; the ability to open a file browser in a run directory; improved identification of read-only files; tool tips for missing files; and an option to clear all output from the Tcl console view.

This version of PlanAhead includes floorplanning improvements for swapping macro location constraints for Block RAMs and DSPs, improvements to ChipScope™ integration for CDC file export and a more consistent debug core-naming convention. It also includes a new tutorial for Tcl and SDC commands.

ChipScope Pro: ISE Design Suite: Logic Edition includes IBERT Support for IBERT CORE Generator and an analyzer IBERT console, both for Virtex-6 GTH rev2.0. An example file for IBERT CORE Generator GUI for the Virtex-6 GTX shows you how to include IBERT core logic in your designs. Also, an AXI monitor in XPS supports AXI4-based designs for Spartan-6 and Virtex-6 families, along with AXI4-Memory Map and AXI4-Lite interfaces.

Implementation (place and route): The ISE Design Suite implementation tools now support intelligent clock gating for Spartan-6 FPGAs (including BRAM optimizations).

ISE Design Suite: DSP Edition

Flows and IP tailored to the needs of algorithm, system and hardware developers

Latest version number: 12.3
 Date of latest release: October 2010
 Previous release: 12.2
 URL to download the latest release:
www.xilinx.com/download

Revision highlights:

All ISE Design Suite Editions include the enhancements listed above for the ISE Design Suite: Logic Edition. Specific to the DSP Edition, ISE 12.3 introduces several enhancements to both System Generator for DSP and DSP IP.

AXI4 support: System Generator and CORE Generator 12.3 contain new AXI4 IP, specifically the complex multiplier 4.0, DDS Compiler 5.0, FFT 8.0 and FIR Compiler 6.0. The DSP Edition also includes beta support for AXI Pcore generation. That is, bus abstraction will make it possible to port existing designs over to AXI by just selecting a radio button on the EDK block in System Generator. You can import and co-simulate AXI-based EDK designs using hardware co-simulation. The DSP Edition also includes SDK co-debug of AXI systems. In particular, it offers software debug with SDK of AXI systems, Pcore design and debug in System Generator and an updated tutorial for the AXI FIR and AXI Pcore.

Run-time improvements: The tool provides a 2x to 3x decrease in run-time for the entire flow, from design loading to netlisting, in System Generator 12.3. It also includes improvements in the Compile (Ctrl-D), Initialization, Simulation and Netlisting steps.

ISE Design Suite: Embedded Edition

An integrated software solution for designing embedded processing systems

Latest version number: 12.3
 Date of latest release: October 2010
 Previous release: 12.2
 URL to download the latest patch:
www.xilinx.com/download

Revision highlights:

All ISE Design Suite Editions include the enhancements listed above for the ISE Design Suite: Logic Edition. The following is the list of enhancements specific to the Embedded Edition.

XPS enhancements: With ISE 12.3, XPS includes support for AXI-based designs. To accomplish this, Xilinx has added Base System Builder support for single-MicroBlaze® AXI-based designs along with the ability to stitch together a system with AXI-based IP interfaces. Each AXI master or slave can run at a different frequency; XPS automatically handles any clock conversion logic. The XPS System Assembly View connectivity panel provides a mechanism to capture sparse connectivity between various AXI masters and slaves for AXI interconnect IP. The AXI Interconnect IP Configuration GUI includes a dialog to change master- or slave-specific settings. In the System Assembly View, it will automatically add a slave module at the time of instantiation.

Support for embedded-software development is not available in XPS for AXI-based designs. The Xilinx SDK remains the primary embedded-software development tool. XPS continues to support software development for PLB designs in a deprecated mode.

XPS includes streamlined integration of MIG flows for AXI-based DDR interfaces, along with AXI ChipScope Monitor support and the ability to cascade multiple AXI interconnect IP blocks in a system, as well as the ability to easily connect to an AXI module that is outside the EDK subsystem.

XPS Ports View can now group ports as part of a bus or an I/O interface. It now includes the ability to easily connect ports of an interface to a module outside the EDK subsystem. Users can rearrange the order in which IP blocks are shown in the System Assembly View and customize the view to suit their design.

SDK enhancements: The 12.3 version of SDK supports AXI-based designs. MicroBlaze v8.00a includes little-endian support (AXI only) and a GNU tool chain update, including a new `-m` little-endian switch. MicroBlaze also includes make-file generation and software tool updates, including SDK, XMD, Libgen and FlashWriter.

System Generator co-debug is now enabled for AXI IP. The SDK also supports drivers and BSP for AXI IP, along with sample application generation and Xilkernel support. An AXI interface is available for MDM v2.00a.

SDK 12.3 has several interface updates. In this version, relative paths for repositories are allowed if the repository is at the same directory level as the BSP directory or one level above. Repository paths are refreshed when importing BSP projects. Manual modification of the BSP settings file (MSS) now triggers a rebuild. Restoring defaults in C/C++ build settings no longer removes inferred options. Default suggested Windows workspace location has been changed.

A Hello World SDK sample application supports MDM UART, while a screencast demonstrating how to use SDK is available from the SDK Welcome page. Several bug patches are provided with 12.3 Answer Record 36777 at <http://www.xilinx.com/support/answers/36777.htm>.

Xilinx CORE Generator & IP Updates

Name of IP: ISE IP Update 12.3
 Type of IP: All

AXI support:

With the release of ISE Design Suite 12.3, Xilinx is introducing CORE Generator IP with

the AXI4 interface. Xilinx has updated the latest versions of the CORE Generator IP listed below with various forms of AXI4 interface support.

In general, the latest version of a given piece of IP for Virtex-6 and Spartan-6 device families will support the AXI4 interface. Older “production” versions of IP will continue to support the legacy interface for the respective core on Virtex-6, Spartan-6, Virtex-5, Virtex-4 and Spartan-3 device families.

For general information on Xilinx AXI4 support, see www.xilinx.com/axi4.htm. For more information on Xilinx AXI4 IP support, see www.xilinx.com/ipcenter/axi4_ip.htm.

Connectivity IP: Aurora 8b/10b v6.1 is now available with the AXI4-Stream interface. Performance capabilities are retained from the LocalLink version with minimal core size increases. CAN v4.1 and DisplayPort v2.1 now come with an AXI4-Lite interface. The PCI Express® core retains the performance capabilities of the TRN version with a minimal increase in core size. The Spartan-6 and Virtex-6 integrated block for PCI Express v2.1 is now available with the AXI4-Stream interface.

DSP IP: FIR Compiler v6.0, DDS Compiler v5.0, Complex Multiplier v4.0 and Fast Fourier Transform (FFT) v8.0 are now available with the AXI4-Stream interface. The release provides a demonstration VHDL testbench for the selected CORE Generator configuration and supports the upgrade from the previous version with a legacy interface. (Please note that this upgrade will only affect core parameters. You must adapt the core instantiation in the design to use the AXI4-Stream interface. For further information, please refer to the section on migrating from earlier versions in the respective IP data sheet.

Memory and storage elements: FIFO Generator v7.2 is now available with AXI4, AXI4-Lite and AXI4-Stream and native interfaces. The FIFO Generator automati-

cally uses write-first mode for Spartan-6 SDP BRAM-based FIFO configurations when different clocks are used for reduced BRAM utilization. MIG v3.6 is now available with AXI4 and native interfaces.

Embedded processor IP: Platform Studio now includes more than 30 new AXI4-compliant cores, such as MicroBlaze version 8.00a supporting both PLBv46 and AXI interfaces, AXI Ethernet and AXI Ethernet Lite, AXI USB 2.0 Device, AXI-based DMAs and AXI Serial Peripheral Interface.

Wireless IP: The new 3GPP LTE Channel Estimator v1.0 is available with AXI4-Stream interface. This latest component of Xilinx’s LTE Baseband Targeted Design Platform provides an optimized channel-estimation function for the physical-uplink shared channel (PUSCH) in LTE base stations. Antenna configurations up to and including 2 x 2 multiuser MIMO are supported. You can tailor this configurable function to meet the unique needs of your application.

Additional IP Highlights

- Digital predistortion (DPD) v3.0 gets its first release in CORE Generator. This market-leading DPD solution for common wireless standards reduces base station equipment capital and operating expenditures by increasing power amplifier efficiency. Configurable to support different algorithmic performance and area combinations, it allows customers to uniquely tune the core implementation to their needs. Supporting up to eight antennas, it offers the smallest, lowest-power and lowest-cost FPGA-based DPD solution available in the market today.
- Soft-error mitigation (SEM) v1.1 automatically detects and corrects soft errors caused by ionizing radiation to enable the highest system reliability and availability and reduce system costs. You can config-

ure this feature to support multiple error-correction methods. Optional error classification and error injection are included. The IP, which is delivered in CORE Generator, supports the Xilinx Virtex-6 FPGA family.

- Block Memory Generator v4.3 has new write-first mode support for single dual-port (SDP) memory type for Spartan-6 devices. Use the write-first mode instead of read first for SDP BRAM when the read and write ports are clocked by different clocks to avoid address space overlap issues. The core also supports soft Hamming error correction in SDP BRAM configurations for data widths < 64 bits (Virtex-6, Virtex-5 and Spartan-6 devices only).

You will find a comprehensive listing of cores that have been updated in this release at www.xilinx.com/ipcenter/coregen/12_3_data sheets.htm. For more information, see www.xilinx.com/ipcenter/coregen/updates_12_3.htm.

Additional CORE Generator Enhancements

The CORE Generator catalog now includes a new “AXI4” column highlighting IP cores with AXI4 support. The IP information panel displays information on supported AXI4 and native interfaces.

You can expand IP symbols to display the components and their bit allocation on a single Tdata channel. For example, you can expand a complex multiplier’s s_axis_a_tdata [31:0] pin symbol to show the packing of real[15:0] and imaginary[31:16] components.

You can now access supplementary IP documents such as user guides from the pull-down menu in the IP GUI as well as from the IP information panel.

Additional PlanAhead IP Design Flow Enhancements

A new “AXI4” column in the IP catalog highlights IP cores with AXI4 support, while the IP information panel displays information on supported AXI4 and native interfaces. 

Meeting the Power Challenge: Transistors Only Take You So Far

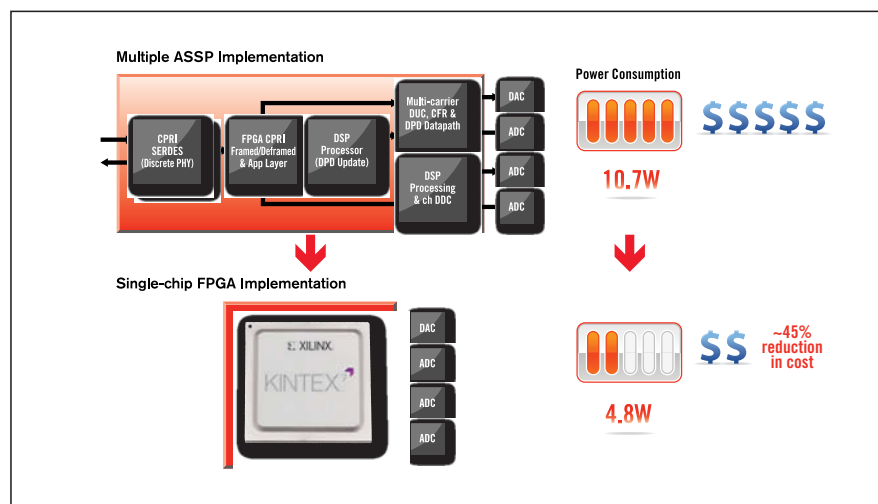
By Manuel Uhm

WITH ELECTRICITY BILLS

accounting for a large part of operating expenses (OPEX), and OPEX itself accounting for ~70% of total costs, operators have had to pay attention to power consumption. Traditionally, silicon suppliers have looked at transistor and process technology to find ways to lower power consumption. While a major contributor, transistors are not the only factor and can only get you so far.

Power reduction is better achieved with a more holistic system-level approach. The best results are achieved by taking into consideration silicon process technology, leveraging power-aware tools, designing code for low power, adjusting system-level architectures, and applying algorithms that can significantly reduce system-level power (e.g., digital pre-distortion [DPD] in a remote radio head application).

Choosing the right silicon partner can help. Xilinx approaches power management as just described — holistically — rather than just focusing myopically on transistors and process node technology. Xilinx® FPGA platform solutions help designers adopt power-optimizing design approaches and system-level design and integration techniques that broadly address the power problem. At the design level, Xilinx power-aware tools, as well as an extensive library of power-efficient reference designs and application notes help engineers optimize overall power consumption. Dedicated teams of Xilinx application engineers can also help designers meet



Designs based on Xilinx FPGAs can take advantage of industry-leading functional density and advanced radio algorithms, such as DPD, to minimize external circuitry and reduce the power consumption of the power amplifier, thereby minimizing power for the complete system.

stringent power goals. Xilinx engineers can walk customers through design optimization techniques such as folding a DSP-intensive design to reduce the size of the design, therefore lowering static power consumption and cost by using a smaller device.

At the system level, Xilinx's attention to integration has yielded great results. For example, integrating multiple discrete components into a single FPGA greatly reduces the total amount of system I/O, which can account for a significant amount of the power draw. Furthermore, using advanced algorithms like DPD in a remote radio head allows telecom equipment manufacturers (TEM) to use a lower-power, lower-cost power amplifier — having the most significant impact on system-level power consumption.

Certainly Xilinx recognizes that transistor and process node technology

cannot be completely ignored when reducing power. The 28nm Xilinx 7-Series FPGAs enable a 50% overall power reduction compared to the previous 40nm generation. In terms of transistor technology, Xilinx's low-power process and use of multiple transistor sizes minimize static power. Xilinx FPGAs use hard blocks for DSP, memory, and SERDES, which greatly minimizes dynamic power consumption over comparable DSP and other FPGA designs.

Addressing the power challenge at the transistor level will get you started in the quest to lower power and lower OPEX. However, only by fine-tuning all levels holistically will you see the greatest results.

To learn more about Xilinx's affordable, power-optimized wireless solutions, please visit www.xilinx.com/esp/wireless.



About the Author: Manuel Uhm is the Director of Wireless Communications at Xilinx Inc. (San Jose, Calif.) and Chair of the User Requirements Committee, Wireless Innovation Forum (SDR Forum v2.0). Contact him at more_info@xilinx.com

© Copyright 2010 Xilinx, Inc. Xilinx, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Xpress Yourself in Our Caption Contest



Let's talk turkey. If you have a yen to Xercise your funny bone, here's your opportunity. In the run-up to the holidays, we invite you to cook up an engineering- or technology-related caption for this cartoon depicting some unusual goings-on in a corporate conference room. The image might inspire a caption like "Does anyone know the mathematical formula for gravy?"

Send your entries to xcell@xilinx.com. Include your name, job title, company affiliation and location, and indicate that you have read the contest rules at www.xilinx.com/xcellcontest and agree to them. After due deliberation, we will print the submissions we like the best in the next issue of *Xcell Journal* and award the winner the Xilinx® SP601 Evaluation Kit, our entry-level development environment for evaluating the Spartan®-6 family of FPGAs (approximate retail value, \$295; see <http://www.xilinx.com/sp601>). Runners-up will gain notoriety, fame and a cool, Xilinx-branded gift from our SWAG closet.

The deadline for submitting entries is 5:00 pm Pacific Time (PT) on December 31, 2010. So, get writing!

NO PURCHASE NECESSARY. You must be 18 or older and a resident of the fifty United States, the District of Columbia, or Canada (excluding Quebec) to enter. Entries must be entirely original and must be received by 5:00 pm Pacific Time (PT) on December 31, 2010. Official rules available online at www.xilinx.com/xcellcontest. Sponsored by Xilinx, Inc. 2100 Logic Drive, San Jose, CA 95124.

The Best Prototyping System Just Got Better

HAPS™ High-performance ASIC Prototyping Systems™

- ✓ Highest Performance
- ✓ Integrated Software Flow
- ✓ Pre-tested DesignWare® IP
- ✓ Advanced Verification Functionality
- ✓ High-speed TDM



HALF THE POWER



TWICE THE PERFORMANCE

A WHOLE NEW WAY OF THINKING.

*Lowest power
and cost*

ARTIX⁷

*Best price
and performance*

KINTEX⁷

*Highest system
performance
and capacity*

VIRTEX⁷

Introducing the 7 Series. Highest performance, lowest power family of FPGAs.

Powerful, flexible, and built on the only unified architecture to span low-cost to ultra high-end FPGA families. Leveraging next-generation ISE Design Suite, development times speed up, while protecting your IP investment. Innovate without compromise.

LEARN MORE AT WWW.XILINX.COM/7

