Xcell journal

ISSUE 90, FIRST QUARTER 2015

SOLUTIONS FOR A PROGRAMMABLE WORLD

16nm UltraScale+ Devices Yield 2-5X Performance/Watt Advantage

60G Millimeter-Wave Backhaul Link Poised to Boost Cellular Capacity

A Double-Barreled Way to Get the Most from Your Zynq SoC

How to Port PetaLinux Onto Your Xilinx FPGA

Solar Orbiter Will Process Data Onboard Using Xilinx FPGAs 16

XILINX ALL PROGRAMMABLE™

www.xilinx.com/xcell

How can Embedded Vision Make Your System Smarter?

Explore the endless possibilities that embedded vision brings to your next design with the MicroZed Embedded Vision Development Kit

Order your kit today at:
www.microzed.org/product/microzed-embedded-vision-kits

Accelerating Your Success™

Integrated Hardware and Software Prototyping Solution

HAPS and ProtoCompiler accelerate software development, HW/SW integration and system validation from individual IP blocks to processor subsystems to complete SoCs.

- Integrated ProtoCompiler design automation software speeds prototype bring-up by 3X
- Enhanced HapsTrak I/O connector technology and high-speed time-domain multiplexing deliver the highest system performance
- Automated debug captures seconds of trace data for superior debug visibility
- Scalable architecture supports up to 288 million ASIC gates to match your design size

To learn more visit: **www.synopsys.com/HAPS**

SYNOPSYS®
Accelerating Innovation

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	Judy Gelwicks 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
	Tomoko Suto, Japan tomoko@xilinx.com
REPRINT ORDERS	1-800-493-5551

Xcell PUBLICATIONS

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/

Get Ready for More Innovations from Xilinx

Ever since Xilinx® began shipping the industry's first All Programmable SoC back in 2011, users have been creating a vast array of innovative products across a growing number of end markets. Automotive, industrial, scientific, wired and wireless communications, aerospace, test and measurement, broadcast and consumer electronics—all these markets have launched or will launch innovations driven with the Zynq® SoC. If you have been reading *Xcell Journal* over the last few years or visiting the new Xcell Daily blog, you've probably noticed the growing percentage of content related to the use of the device.

Certainly a common theme throughout all the Zynq SoC-related articles is the amazing system performance the device achieves simply by integrating and interconnecting the ARM® dual-core Cortex™-A9 MPCore processors and 7 series FPGA logic—all on a single device. With more than 3,000 interconnects linking the processing system with the programmable logic, the Zynq SoC achieves performance that a two-chip, ASSP/ASIC + FPGA simply can't manage. There are not enough external I/Os on the outside of any discrete FPGA and ASSP to do the job. An added benefit of this integration is a reduction in power requirements (as well as the BOM), since the two-chip-for-one-chip swap also means the system needs less power circuitry. Over the last four years, the Zynq SoC has certainly proven worthy of all the Innovation awards it has received from tier-one trade publications worldwide.

So it is with great excitement that here at *Xcell Journal*, we finally get to reveal the on-chip details of the next-generation All Programmable SoC from Xilinx: the Zynq UltraScale+™ MPSoC, which is scheduled for shipment early next year. I encourage you to read the cover story for details about the device and the rest of Xilinx's newly unveiled 16nm UltraScale+ portfolio. Leveraging lessons learned from the original Zynq SoC, feedback from users and insights into their product road maps, Xilinx has created an All Programmable MPSoC that achieves exponentially higher levels of system integration and system performance/watt than the remarkable first-generation Zynq SoC.

In fact, the cover story explains how all the devices—FPGAs, 3D ICs and MPSoCs—in the new UltraScale+ portfolio attain at a minimum twice the performance per watt of preceding-generation systems, thanks to implementation in TSMC's 16nm FFT+ process. Additional performance/watt benefits accrue from a new, larger memory called UltraRAM that Xilinx is implementing in most of these devices and from a new system-level interconnect technology called SmartConnect.

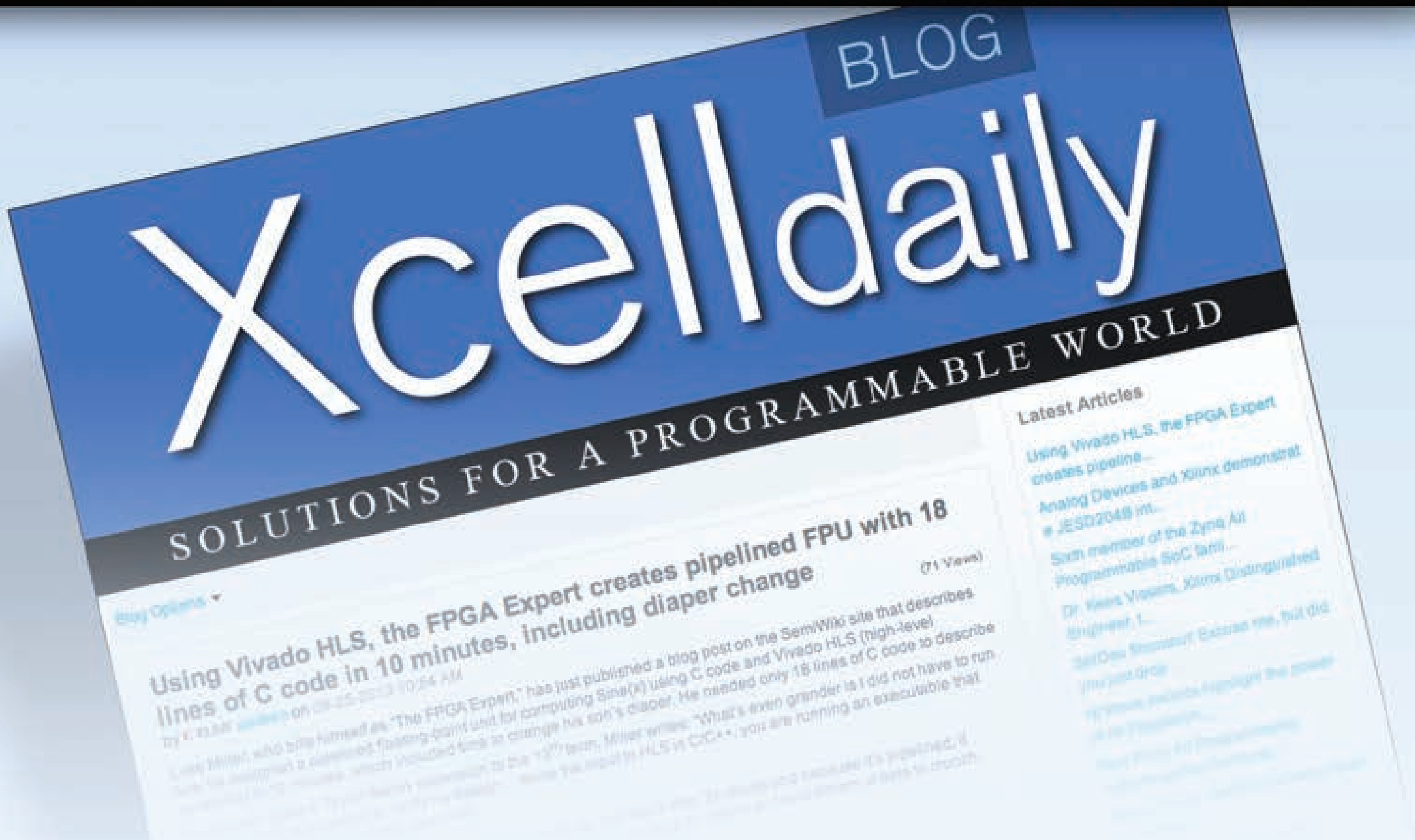
By far the greatest performance/watt benefit can be achieved with the Zynq UltraScale+ MPSoC, which is a kitchen sink All Programmable SoC. It has a quad-core 64-bit APU, a dual-core RPU, a graphics processor plus a host of peripherals, security features and power management, all on one chip. Zynq MPSoC systems will be able to achieve 5x the performance/watt of 28nm Zynq SoC systems.

You have created some amazing systems with the Zynq SoC. I can't wait to see what you do with the Zynq UltraScale+ MPSoC. I hope that as Xilinx begins rolling out these remarkable devices, you will continue to share your design experiences with your colleagues by publishing articles in *Xcell Journal*.

Mike Santarini
Publisher

We dedicate this issue to ***Dan Teie***, Xcell Journal's *longtime advertising and creative director, who passed away in January. Dan was a sportsman, an adventurer, a fighter, a gentleman and a great human being who had a deep love for his family and friends and great passion for living life to its fullest. We miss you, Dan.*

Xcell Journal Adds New Daily Blog



BLOG

Xcelldaily

SOLUTIONS FOR A PROGRAMMABLE WORLD

Blog Options

Using Vivado HLS, the FPGA Expert creates pipelined FPU with 18 lines of C code in 10 minutes, including diaper change

(71 Views)

Latest Articles

Using Vivado HLS, the FPGA Expert creates pipeline...

Analog Devices and Xilinx demonstrate JESD204B int...

Sixth member of the Zynq All Programmable SoC fami...

Dr. Kees Vissers, Xilinx Distinguished Engineer, t...

Xilinx has extended the Award Winning Journal and added an exciting new *Xcell Daily Blog*. The new site provides dedicated readers with a frequent flow of content to help engineers leverage the flexibility and extensive capabilities of Xilinx products, ecosystem, and customers to create All Programmable and Smarter Systems.

Recent

- *Five I-MOVIX X10 UHD (4K) cameras recorded Super Bowl XLIX in glorious ultra slow motion—FPGAs tackled the video*
- *14-bit, 1.6Gsamples/sec PCIe/PXIe/USB/Micro-TCA Digitizer employs FPGA for custom signal processing at speed*
- *An Open Source Kickstarter SDN project based on Zynq? For $649? Wowzers!*
- *Zynq-based open-source programmable RF transceiver project*
- *Dave Jones' EEVBlog reviews $149 Digilent Basys 3 Eval Board based on Xilinx Artix-7 FPGANational Instruments enhances LabVIEW FPGA*

Visit Blog: www.forums.xilinx.com/t5/Xcell-Daily/bg-p/Xcell

XILINX ALL PROGRAMMABLE.

VIRTEX UltraSCALE
KINTEX UltraSCALE+

FIRST QUARTER 2015, ISSUE 90

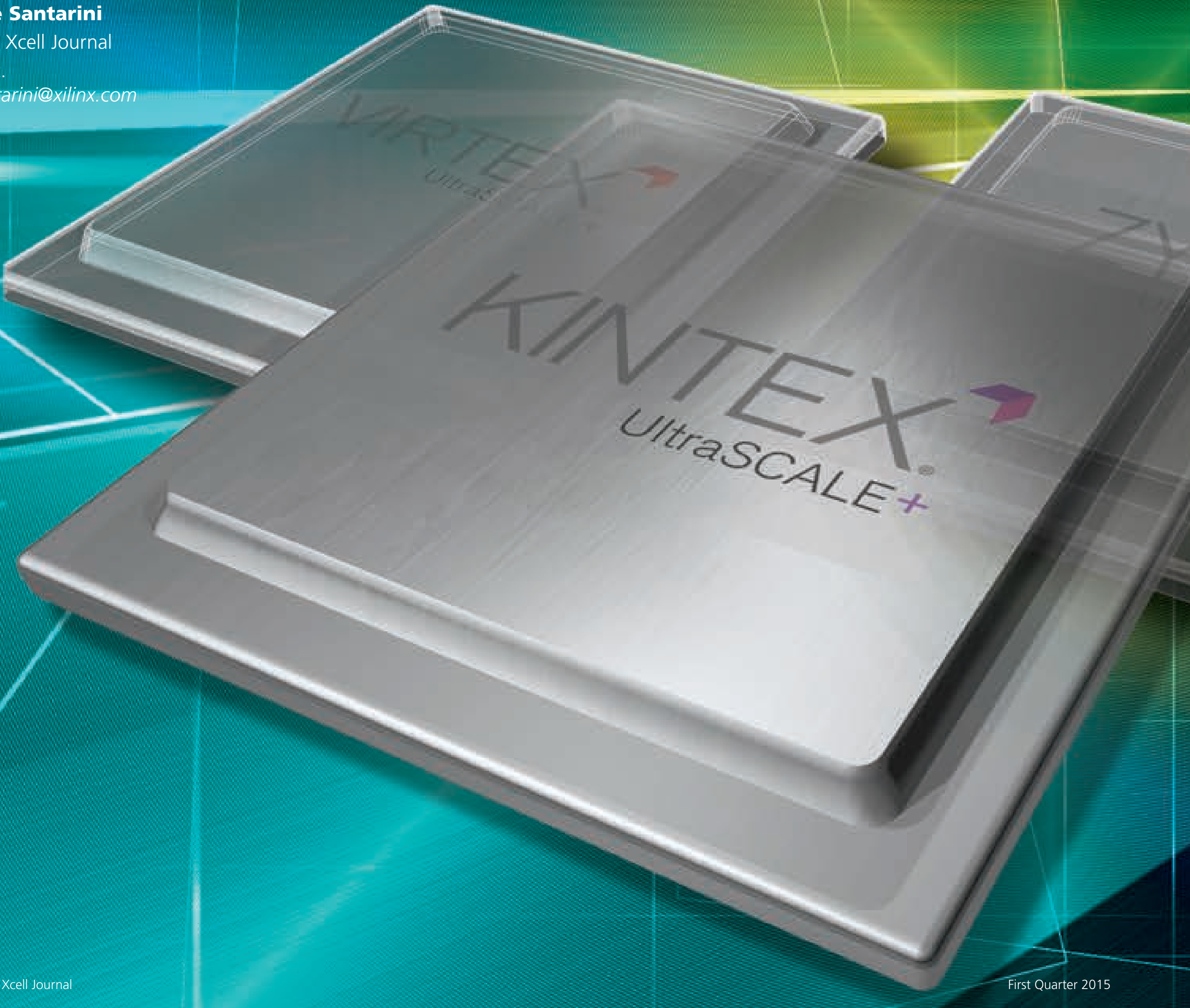
THE XILINX XPERIENCE FEATURES

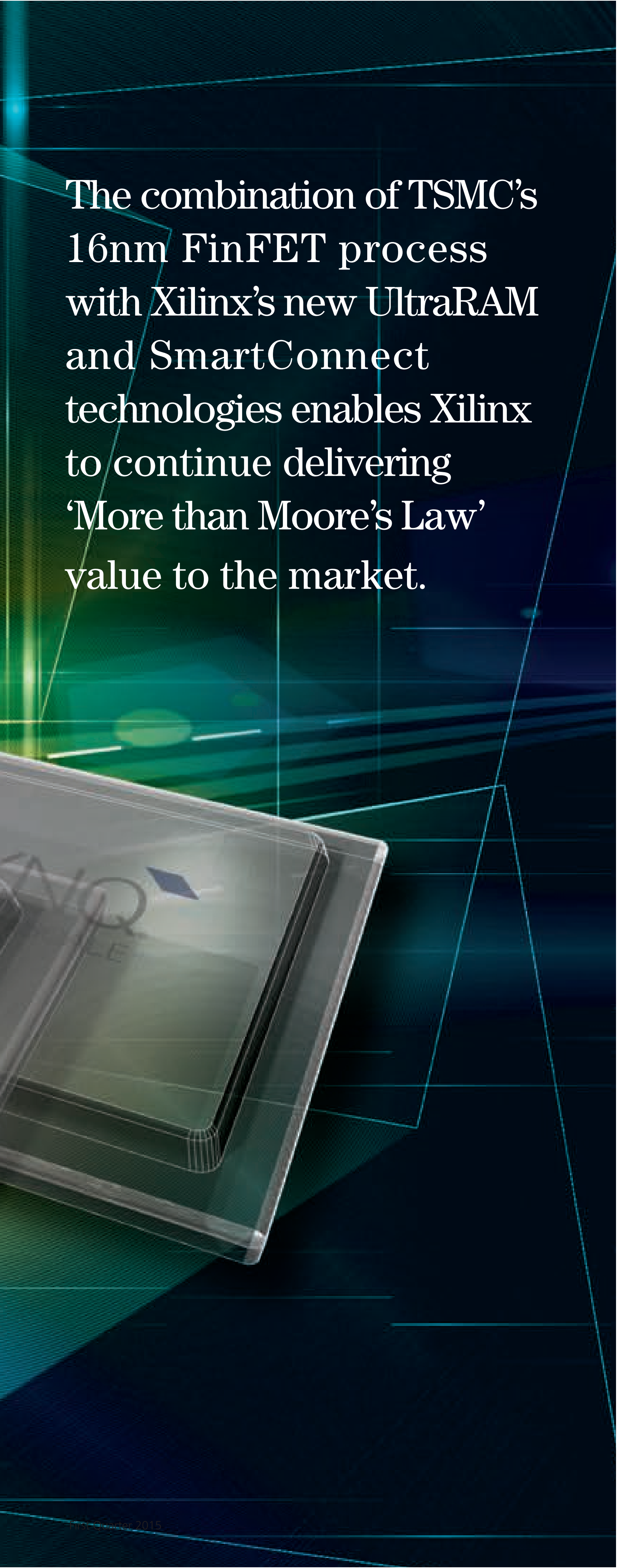
XTRA READING

COVER STORY

Xilinx 16nm UltraScale+ Devices Yield 2-5X Performance/Watt Advantage

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





The combination of TSMC's 16nm FinFET process with Xilinx's new UltraRAM and SmartConnect technologies enables Xilinx to continue delivering 'More than Moore's Law' value to the market.

Building on the Generation Ahead lead gained with its 28nm, 7 series All Programmable device family and its first-to-market 20nm UltraScale™ portfolio, Xilinx® has just unveiled its 16nm UltraScale+™ lineup. The device portfolio will enable customers to build systems with a 2X to 5X performance-per-watt advantage over comparable systems designed with Xilinx's 28nm devices. These performance/watt advantages rest on three main pillars: device implementation in TSMC's 16FF+ (16nm FinFET Plus) process, Xilinx's on-chip UltraRAM memory and an innovative system-level interconnect-optimization technology called SmartConnect.

In addition, Xilinx has also unwrapped its second-generation Zynq® All Programmable SoC. The Zynq UltraScale Multiprocessing SoC (MPSoC) features on a single device a quad-core 64-bit ARM® Cortex™-A53 application processor, a 32-bit ARM Cortex-R5 real-time processer and an ARM Mali-400MP graphics processor, along with 16nm FPGA logic (with UltraRAM), a host of peripherals, security and reliability features, and an innovative power control technology. The new Zynq UltraScale+ MPSoC gives users what they need to create systems with a 5X performance/watt advantage over systems designed with the 28nm Zynq SoC.

FINFET EXPANDS ULTRASCALE PORTFOLIO WITH EXTRA NODE OF VALUE

"With the 16nm UltraScale+ portfolio, we are creating an extra node of value ahead of what Moore's Law would traditionally afford users," said Dave Myron, senior director of silicon product management and marketing at Xilinx. "We are addressing a broad range of next-generation applications, including LTE Advanced and early 5G wireless, terabit wired communications, automotive advanced driver-assistance systems and industrial Internet-of-Things applications. The UltraScale+ portfolio will enable customers to create greater innovations while staying ahead of the competition in their respective markets."

With its UltraScale generation of products, Xilinx is concurrently offering devices from two process nodes: TSMC's 20nm planar process (already shipping) and now TSMC's 16FF+ process (which Xilinx plans to ship in the fourth calendar quarter of 2015). Xilinx will be offering 16nm UltraScale+ versions of its Virtex® FPGA and 3D IC families, its Kintex® FPGA family as well as the new Zynq UltraScale+ MPSoCs.

Mark Moran, director of new product introduction and solution marketing, said that Xilinx decided to begin its UltraScale rollout with 20nm in 2013, instead of waiting for TSMC's 16FF+ process. That's because in some application spaces, it was imperative to have 20nm devices—which are inherently higher in performance and capacity than 28nm—a year and a half sooner.

"Our entire portfolio is designed with market needs in mind," said Moran. "The capabilities of the devices in the 20nm UltraScale architecture are better suited to next-generation products of certain markets and end applications that don't require that extra node of performance/watt UltraScale+ offers. We built 20nm FinFET knowing 16nm was close behind. And so we implemented a lot of architectural changes in 20nm that we knew we could build on for 16nm to add an extra level of performance and value for markets that need it. We have customers who are getting a head start and developing on the 20nm devices we have available today so that when 16nm UltraScale+ devices become available, they can quickly port their designs and get those designs to market sooner."

Myron added that many of the Virtex UltraScale+ devices will be pin-compatible with the 20nm Virtex UltraScale devices, making it easy to trade up for designs that require the extra performance/watt benefits.

"From a tools perspective, the 20nm UltraScale and 16nm UltraScale+ devices look almost identical," said Myron. "So there is an additional benefit of using the 16nm UltraScale+ devices, because the improvements in performance/watt make it easier to achieve performance and power goals."

Myron said that UltraScale+ FPGAs and 3D ICs will afford more than a 2X performance/watt advantage over 28nm, 7 series FPGAs. Meanwhile, Zynq UltraScale+ MPSoCs, with their additional integrated heterogeneous processing capabilities, will have more than a 5X system performance/watt advantage over comparable systems built with the 28nm Zynq SoCs (Figure 1).

PERFORMANCE/WATT EDGE FROM TSMC'S 16FF+ PROCESS

Based purely on the process migration to 16nm FinFET, Xilinx has produced devices that boast a 2X performance/watt advantage over 28nm, 7 series devices. "TSMC's 16FF+ is an extremely efficient process technology in that it virtually eliminates transistor power leakage associated with the preceding silicon processes implemented with planar transistors,"

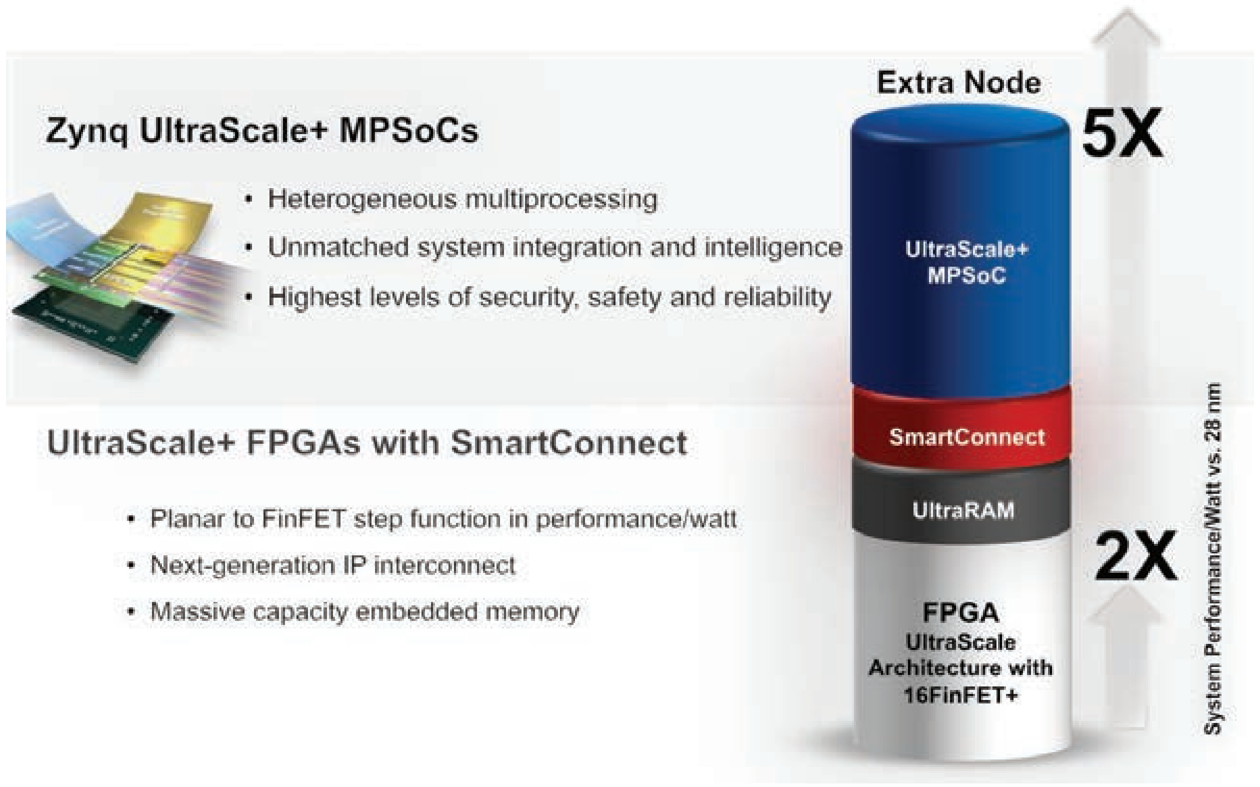

Zynq UltraScale+ MPSoCs
• Heterogeneous multiprocessing
• Unmatched system integration and intelligence
• Highest levels of security, safety and reliability
UltraScale+ FPGAs with SmartConnect
• Planar to FinFET step function in performance/watt
• Next-generation IP interconnect
• Massive capacity embedded memory
Extra Node
5X
UltraScale+ MPSoC
SmartConnect
UltraRAM
2X
FPGA UltraScale Architecture with 16FinFET+
System Performance/Watt vs. 28 nm

Figure 1 – Xilinx 16nm UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs offer design teams an extra node of value.

said Myron. "In addition, we have worked diligently with TSMC to refine UltraScale+ devices to take full advantage of the new process technology. At a minimum (just from the new process technology innovations), UltraScale+ designs will see more than twice the performance/watt improvement over designs implemented in 28nm 7 series devices."

For a detailed description of Xilinx's 20nm UltraScale architecture and the advantages of FinFET over planar transistor processes, see the cover story in *Xcell Journal*, issue 84.

In the UltraScale+ family, Xilinx is also offering the industry's first 3D-on-3D devices—its third-generation stacked-silicon interconnect 3D ICs implemented on TSMC's 16FF+ 3D transistor technology.

The award-winning 7 series 3D ICs surpassed the performance and capacity limits of Moore's Law by offering multiple dice on a single integrated circuit, Myron said.

"With our homogeneous 3D IC, we were able to smash the capacity limits of Moore's Law, offering a device that was twice the capacity of what the largest monolithic FPGA could produce at 28nm," said Myron. "Then, with our first heterogeneous device, we were able to mix FPGA dice with high-speed transceiver dice and offer system performance and bandwidth not possible with a 28nm monolithic device. With UltraScale+ 3D ICs, we'll continue to offer capacity and performance exceeding the Moore's Law trajectory."

PERFORMANCE/WATT ADVANTAGE FROM ULTRARAM

Myron said that many UltraScale+ designs will gain an additional performance/watt improvement vs. 28nm from a new, large on-chip memory called UltraRAM. Xilinx is adding the UltraRAM to most of the UltraScale+ devices.

"Fundamentally, what we are seeing is a growing chasm between the on-chip memory you have, such as LUT RAM or distributed RAM and Block RAM, and the memory you have off-chip, such as DDR or off-chip SRAM," said Myron. "There are so many processor-intensive applications that need different kinds of memory. Especially as you design larger, more complex designs, there is a growing need to have faster memory on-chip. Block RAMs are too granular and there are too few of them. And if you put memory off the chip, it adds to power consumption, complicates I/O and adds to BOM cost."

These are the reasons Xilinx created UltraRAM. "What we've done is add another level of memory hierarchy on-chip, along with the ability to easily implement large blocks of memory into the design," Myron said. "We have made it easy for designers to place the right size memory on-chip and the timing is guaranteed."

LUT or distributed RAM allows designers to add RAM in bit and kilobit sizes, and BRAM lets them add memory blocks in tens of megabits. UltraRAM will allow those using UltraScale+ devices to implement on-chip SRAM in blocks counted in hundreds of megabits (Figure 2). By doing so, designers will be able to create higher-performance and more power-efficient systems that require less off-chip RAM (SRAM, RLDRAM and TCAM). The result will be a reduction in BOM costs. The largest UltraScale+ device, the VU13P, will have 432 Mbits of UltraRAM.

PERFORMANCE/WATT ADVANTAGE FROM SMARTCONNECT

Another new technology, called SmartConnect, brings additional performance/watt improvements to UltraScale+ designs.

"SmartConnect is a co-optimization of the tools and hardware and an intelligent way to enable designs to be more

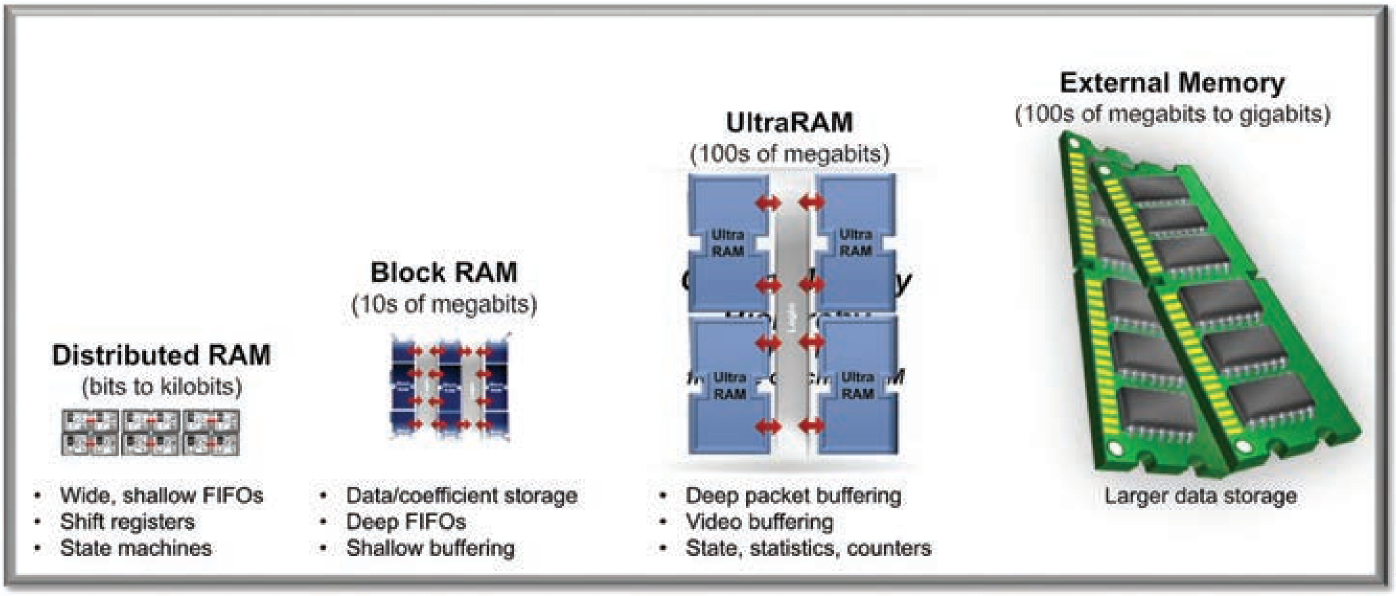


Figure 2 – UltraRAM closes the memory gap between on-chip and off-chip memory, allowing designers to create higher-performance and lower-power systems with larger local memory blocks.

easily implemented even as they are becoming more complex," said Myron.

Traditionally as engineers cram more IP blocks into a design, the overhead—in terms of power and area requirements—increases. With SmartConnect, Xilinx has added optimizations to the Vivado® Design Suite that will look at the entire design from a system level, Myron said. SmartConnect will come up with the most efficient interconnect topologies to get the lowest area and highest performance, leveraging certain new enhancements to the AXI interconnect along with the 16nm UltraScale+ silicon.

"The 16nm UltraScale+ devices will have greater efficiency at this higher protocol level, not just the routing level," said Myron. "That means there is an additional net performance/watt benefit on top of the 16nm FinFET advantage."

Figure 3 illustrates a real design that has eight video-processing engines, all interfacing with a processor and memory. "It may be surprising that in a real- world design like this, the interconnect logic actually can consume almost half the design's total area. This not only impacts power but limits frequency," said Myron. "SmartConnect can automatically restructure the interconnect blocks and decrease power by 20 percent at the same performance.

16NM ULTRASCALE FPGA BENCHMARK

To illustrate the performance/watt advantage in an FPGA design scenario, a 48-port wireless CPRI compression and baseband hardware accelerator implemented in a 28nm Virtex-7 FPGA consumes 56 watts (Figure 4). The same design running at the same performance but implemented in a 16nm Virtex UltraScale+ FPGA consumes 27 W, or 55 percent less, giving it a 2.1X performance/watt advantage. With the additional performance/watt advantage from UltraRAM and SmartConnect, the performance/watt advantage of the Virtex UltraScale+ version of the design jumps to better than 2.7X that of the 28nm Virtex-7 FPGA implementation, with 63 percent less power.

Similarly, in an image-processing PCI module with a 15-W FPGA power budget, a 28nm Virtex-7 yields performance of 525 operations per second. In comparison, the same design implemented in 16nm UltraScale yields 1,255 operations per second, a 2.4X performance/watt increase. Adding the gains from UltraRAM and SmartConnect, the performance/watt advantage of the Virtex UltraScale+ version jumps to over 3.6X that of the 28nm Virtex-7 FPGA implementation.

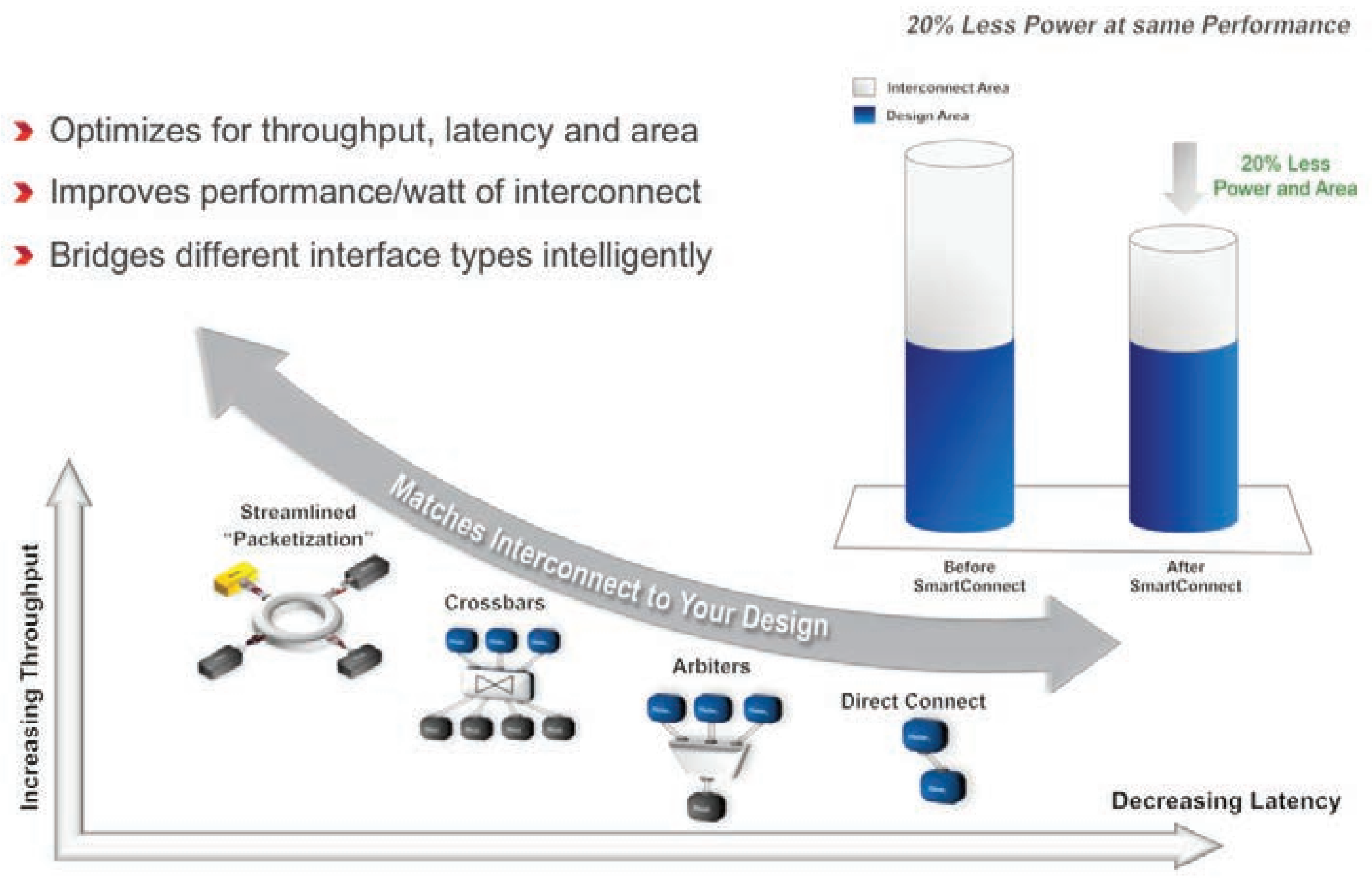

20% Less Power at same Performance
Interconnect Area
Design Area
Optimizes for throughput, latency and area
Improves performance/watt of interconnect
Bridges different interface types intelligently
20% Less Power and Area
Before SmartConnect
After SmartConnect
Matches Interconnect to Your Design
Streamlined "Packetization"
Crossbars
Arbiters
Direct Connect
Increasing Throughput
Decreasing Latency

Figure 3 – SmartConnect technology cuts the area of interconnect required by up to 20 percent, which in turn reduces power consumption by 20 percent at the same performance level.

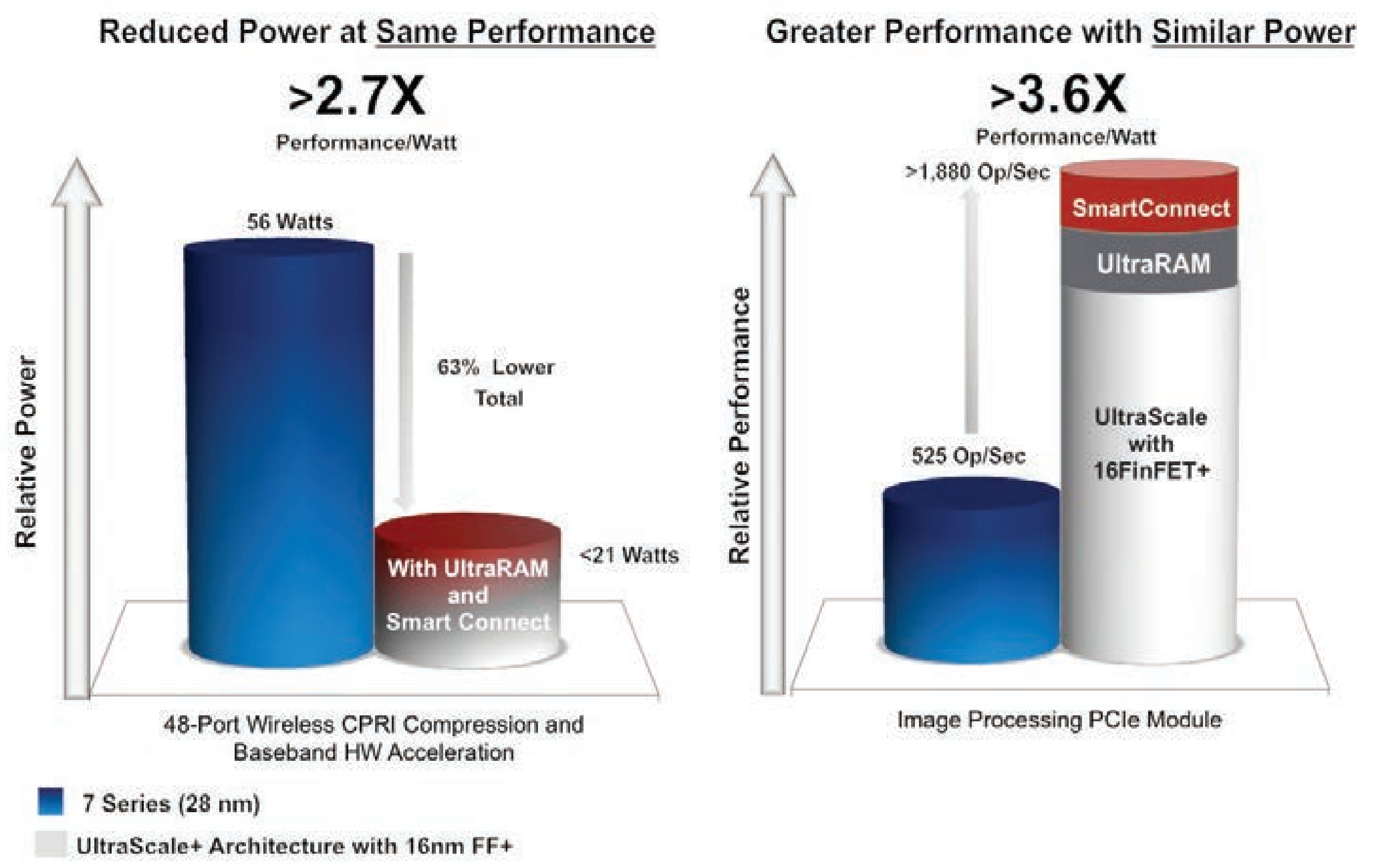


Figure 4 – The 16nm UltraScale+ retains its impressive performance/watt advantage for those seeking to implement faster designs on the same power budget or those seeking drastic power reductions with the same performance.

ZYNQ ULTRASCALE MPSOC OFFERS PERFORMANCE/WATT ADVANTAGE OF OVER 5X

While Xilinx could have implemented its second-generation All Programmable SoC in TSMC's 20nm process, the company chose to wait to implement the device in TSMC's 16nm FinFET process. The heterogeneous multiprocessing feature set of the device, paired with the performance/watt advantages of the 16nm UltraScale architecture, make the 16nm Zynq UltraScale+ MPSoC an even more efficient central processing system controller. The device delivers more than 5X the performance of the 28nm Zynq SoC.

Last year, Xilinx unveiled its "Right Engines for the Right Tasks" use model for the UltraScale MPSoC architecture but withheld details regarding which particular cores the Zynq UltraScale+ MPSoC devices would have. The company is now unveiling the full feature set of the Zynq UltraScale+ MPSoC (Figure 5).

Certainly the biggest value-add of the original 28nm Zynq SoC was in integrating an ARM processing system and programmable logic on a single device. More than 3,000 interconnects (running at a peak bandwidth of ~84 Gbps) link the Zynq SoC's processing system and programmable logic blocks. This tight connection between the PS and PL yields throughput and performance simply not possible with a two-chip system architecture consisting of an FPGA and a separate ASSP.

Now, with the 16nm UltraScale+ MPSoC, Xilinx has dramatically improved the performance between the processing system and programmable logic, giving the device more than 6,000 interconnects running at 500-Gbps peak bandwidth. "This makes the connection between the Zynq UltraScale+ MPSoC's processing and logic systems 6X faster than what is possible with the 28nm Zynq SoC," said Barrie Mullins, Xilinx's director of All Programmable SoC product marketing and management. "It leaves two-chip ASSP-plus-FPGA architectures that much further behind in terms of system performance."

Mullins said that at the center of the Zynq UltraScale+ MPSoC is the 64-bit, quad-core ARM Cortex-A53 processor, which delivers better than double the performance of the 28nm Zynq SoC's dual-Cortex-A9 processing system. The application processing system is capable of hardware virtualization and asymmetric processing, and fully supports ARM's TrustZone®

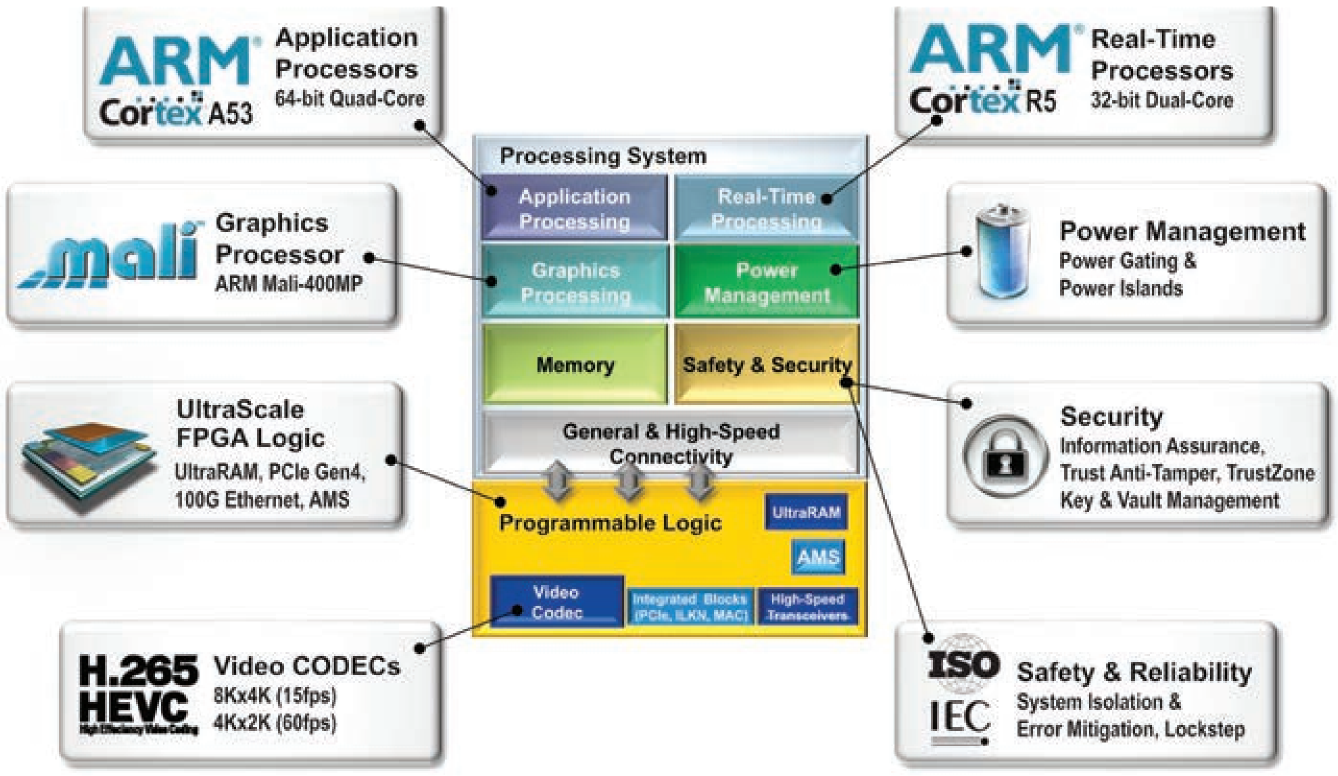


Figure 5 – The 16nm Zynq UltraScale+ MPSoC features a rich set of processing engines that design teams can tailor for unmatched system performance, drastically increasing the value of their systems.

suite of security features.

Xilinx also gave the Zynq UltraScale+ MPSoC a dual-core, ARM Cortex-R5 real-time processing subsystem that allows users to add deterministic operation to their systems. The real-time processor ensures instantaneous system responsiveness for applications requiring the highest levels of throughput, safety and reliability.

The Zynq UltraScale+ MPSoC also includes a number of dedicated graphics engines for further gains in processing performance. An ARM Mali™-400MP dedicated graphics acceleration core offloads graphics-intensive tasks from the main CPU. To complement the GPU, Xilinx added a hardened video codec core to the programmable logic block for video compression/decompression supporting the H.265 video standard for 8Kx4K (15 frames per second) and 4Kx2K (60 fps). A DisplayPort source core allows users to speed video data packetization while eliminating the need for an external DisplayPort TX transmitter chip in their systems.

The Zynq UltraScale+ MPSoC also features a number of on-chip memory enhancements. The largest devices in the product family will include UltraRAM in addition to Block RAM in the programmable logic. Meanwhile, the Zynq UltraScale+ MPSoC's processing cores share L1 and L2 caches.

The Zynq UltraScale+ MPSoC also features a wider, 72-bit DDR interface core with ECC (64 bits plus 8 bits for ECC). The interface boasts speeds of up to 2,400 Mbps for DDR4, with support for larger-memory-depth DRAM capacity of 32 Gbytes.

A dedicated security unit on the Zynq UltraScale+ MPSoC enables military-class security such as secure boot, key and vault management, and anti-tamper capabilities—all standard requirements for machine-to-machine communication and connected control applications. In addition, the Zynq UltraScale+ MPSoC's programmable logic system also includes integrated connectivity blocks for 150G Interlaken, 100G Ethernet MAC and PCIe® Gen4. An onboard Analog Mixed-Signal (AMS) core helps design teams test their systems with System Monitor.

With all these features, it is unlikely that any application would use every engine available in the MPSoC. Therefore, Xilinx gave the Zynq UltraScale+ MPSoC an extremely flexible dedicated power-management unit (PMU). The core enables users to control power domains and islands (coarse and fine-grained) to power only those processing units the system is using. What's more, design teams can program the core for dynamic operation, ensuring the system runs only the features needed to perform a given task and then powers down. The PMU also drives a multitude of safety and reliability capabilities such as signal and error detection and mitigation, safe-state mode, and system isolation and protection.

Myron said that thanks to all of these processing features added to the 16nm performance/watt features discussed above, designs built with the Zynq UltraScale+ MPSoC will enjoy on average a 5X performance/watt advantage over designs implemented with the 28nm Zynq SoC.

16NM ZYNQ ULTRASCALE MPSOC BENCHMARK

To illustrate the Zynq UltraScale+ MPSoC's performance/watt advantage, let's look at benchmarks for three of the many applications the device serves, color-coded to demonstrate the diversity of processing engines (Figure 6).

To create a videoconferencing system that runs full 1080p video, designers used a Zynq SoC paired with a separate H.264 ASSP. With the advantages of the Zynq UltraScale+ MPSoC, designers can now implement a 4Kx2K UHD system in one Zynq UltraScale+ MPSoC with the same power budget and achieve 5X the performance/watt savings of the two-chip system.

"In a public-safety radio application that required a Zynq SoC along with two ASSPs, you can now implement the entire design in one Zynq UltraScale+ MPSoC with 47 percent less system power and 2.5X the performance of the previous configuration, yielding a 4.8X performance/watt advantage," said Sumit Shah, senior SoC product line manager.

Likewise, Shah said an automotive multicamera driver assist system previously implemented in two 28nm Zynq SoCs can shrink to one Zynq UltraScale+ MPSoC. The one-chip system delivers 2.5X the performance of the two-chip design and consumes 50 percent less power. This yields a net 5X performance/watt advantage over the previous implementation.

Early customer engagements are in process for all of the UltraScale+ families. Xilinx has scheduled first tapeouts and early-access release of the design tools for the second calendar quarter of 2015. The company expects to begin shipping UltraScale+ devices to customers in the fourth calendar quarter of 2015.

For more information on the 16nm UltraScale portfolio's performance/watt advantage, visit www.xilinx.com/ultrascale. For further information on the Zynq UltraScale+ MPSoC, visit www.xilinx.com/products/technology/ultrascale-mpsoc.html.

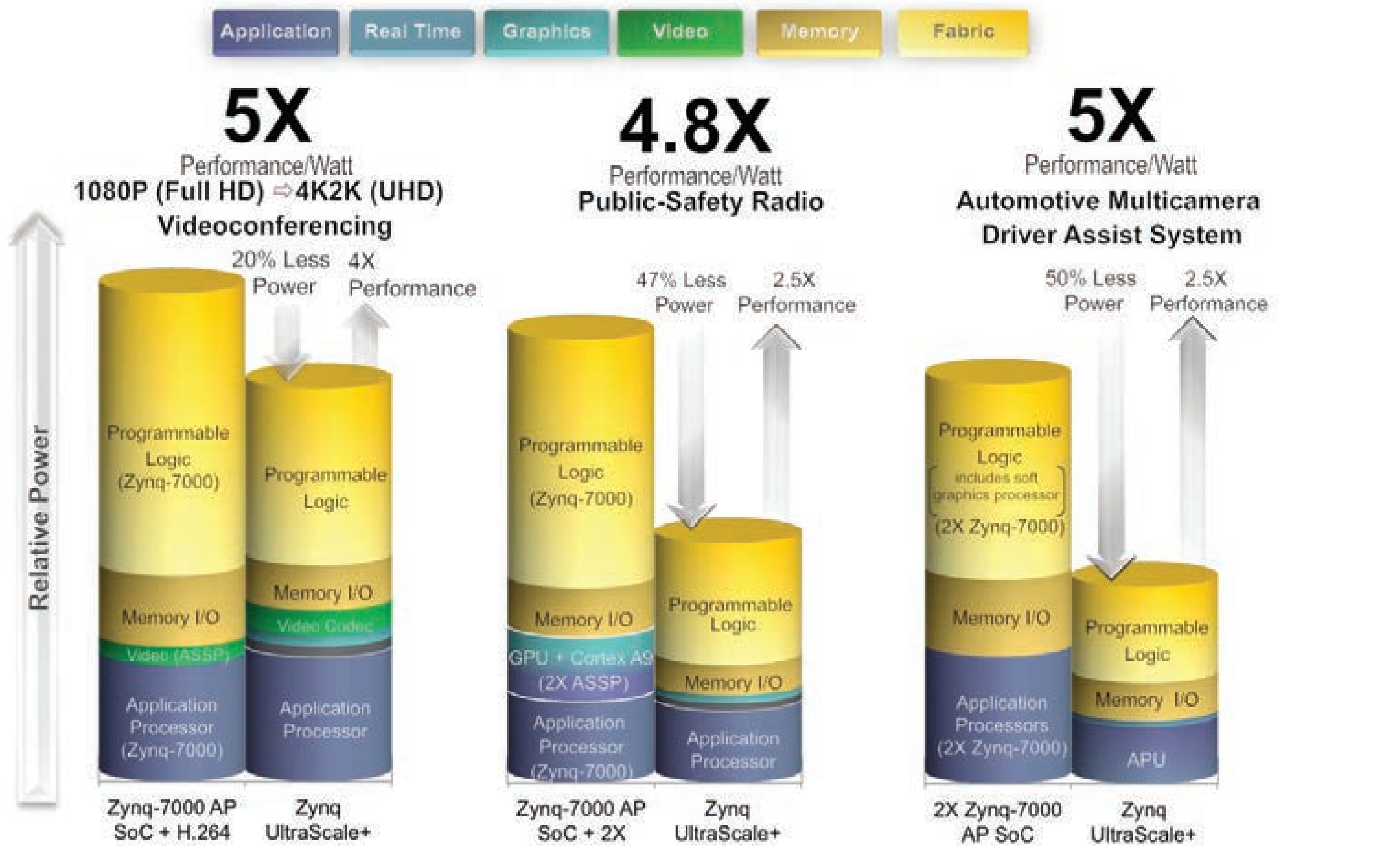
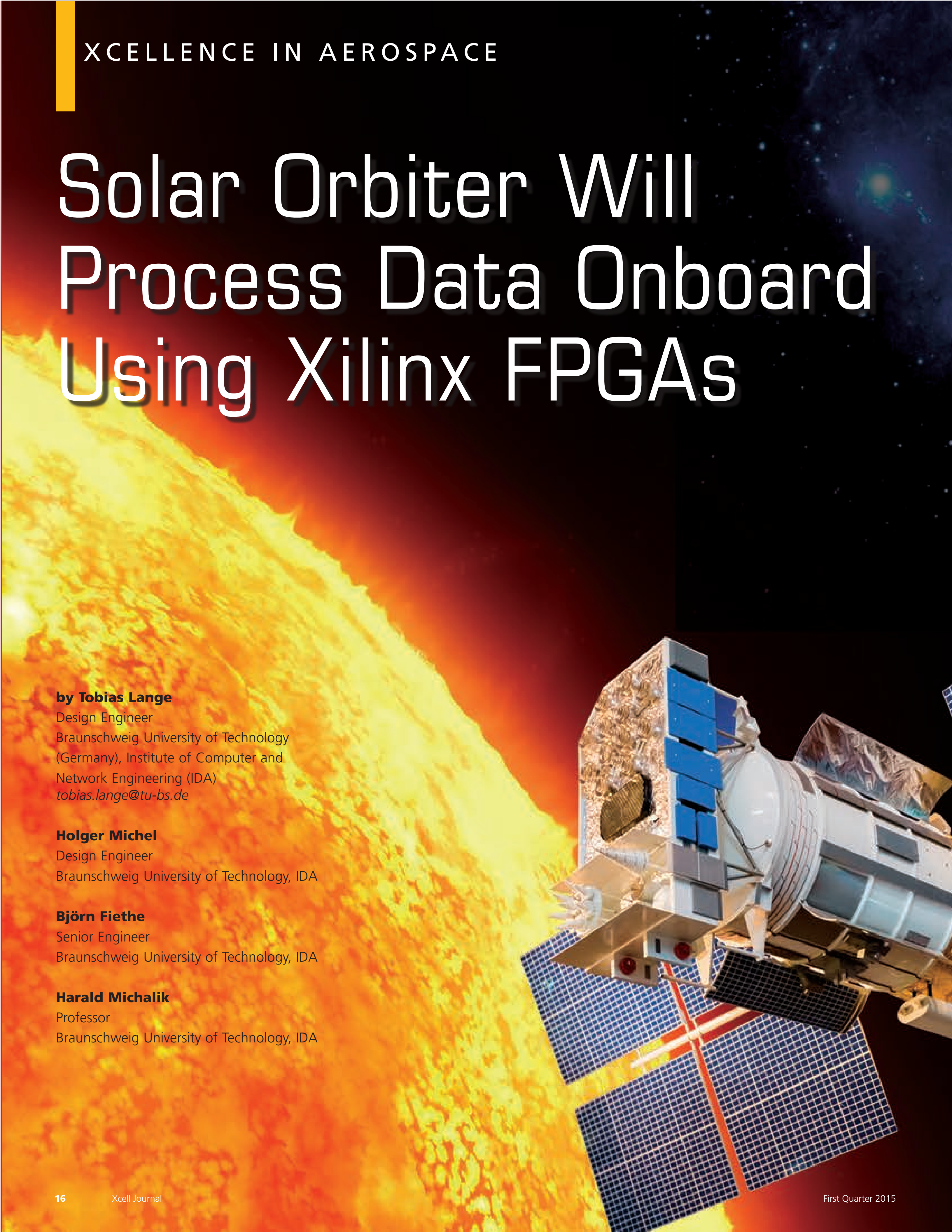


Figure 6 – The Zynq UltraScale+ MPSoC's extensive processing blocks, rich peripherals set and 16nm logic blocks enable design teams to create innovative systems with a 5X performance/watt advantage over designs using 28nm Zynq SoCs.

Solar Orbiter Will Process Data Onboard Using Xilinx FPGAs



by Tobias Lange
Design Engineer
Braunschweig University of Technology (Germany), Institute of Computer and Network Engineering (IDA)
tobias.lange@tu-bs.de

Holger Michel
Design Engineer
Braunschweig University of Technology, IDA

Björn Fiethe
Senior Engineer
Braunschweig University of Technology, IDA

Harald Michalik
Professor
Braunschweig University of Technology, IDA

Space-grade Virtex FPGAs will accelerate the acquisition of image data and enable in-flight processing on scientific space instruments.

State-of-the-art remote sensing instruments on spacecrafts deliver vast amounts of high-resolution image data. For classical Earth observation missions, scientists typically evaluate the collected data after reception on the ground. While deep-space missions also have to cope with high imaging data rates, the telemetry rate, on the other hand, is very limited.

One demanding example is the Polarimetric and Helioseismic Imager (PHI) instrument, which has been selected as part of the scientific payload for the European Space Agency's Solar Orbiter mission, due to launch in 2017. The PHI instrument, developed mainly at the Max Planck Institute for Solar System Research (MPS) in Germany, will provide maps of the continuum intensity, magnetic-field vector and line-of-sight velocity in the solar photosphere.

Because of the high amount of captured data and the limited downlink capability, extracting scientific parameters onboard the spacecraft will reduce the data volume dramatically. As a result, scientists will be able to take a closer look into the solar photosphere.

To cope with these onboard processing demands, Xilinx® SRAM-based FPGAs with high gate counts offer an attractive solution. Our team at the Braunschweig University of Technology in Germany already has a history with Xilinx FPGAs in active space missions. We have used these devices for classical image-data compression in the processing units of the Venus Express Monitoring Camera (VMC) and the Dawn Framing Camera (DawnFC), both now in successful operation for several years. For the Solar Orbiter PHI processing unit, we decided to use two space-grade Virtex®-4 FPGAs, which will be reconfigured during flight.

Before going into the details of how the FPGAs will streamline data capture on this important mission, let's take a closer look at the PHI itself and examine how it operates.

THE SOLAR ORBITER PHI

The PHI instrument acquires sets of images from an active-pixel sensor. A fil-

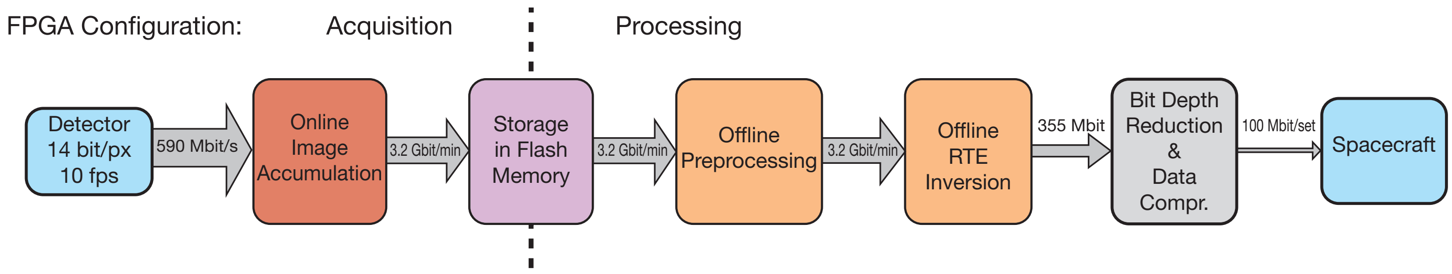


Figure 1 – Polarimetric and Helioseismic Imager (PHI) data-processing pipeline

ter wheel in the optical path applies different wavelength and polarization settings to these images. By preprocessing the captured image data (for example, dark- and flat-field correction) and performing a compute-intensive inversion of the radiative transform equation (RTE), it's possible to calculate magnetic-field vectors from pixel data. Together with standard data compression, this approach will reduce the amount of data from 3.2 Gbits to 100 Mbits per data set. This is a factor of 64 compared with the raw-data input

The processing flow of the PHI can be divided into two modes of operation (Figure 1). Changing between these two modes is perfectly applicable for in-flight reconfiguration of the Virtex FPGAs. Here's how the process works.

During the acquisition phase (the reddish box at left in Figure 1), the detector provides images with a resolution of 2,048 x 2,048 pixels. The acquisition FPGA will accumulate a set of multiple images at different filter settings and directly store them in a large array of NAND flash memory. To reduce residual spacecraft jitter, a dedicated controller for an image-stabilization system will run simultaneously.

After the data acquisition, we reconfigure the two Virtex-4 FPGAs with a preprocessing core and an RTE core (the orange boxes in Figure 1). The preprocessing core retrieves the previously stored data from the flash memory and performs dark- and flat-field correction, addition, multiplication and convolution of frames. Subsequently, the RTE core computes the inversion of the radiative transfer equation.

The FPGA design of the RTE inversion

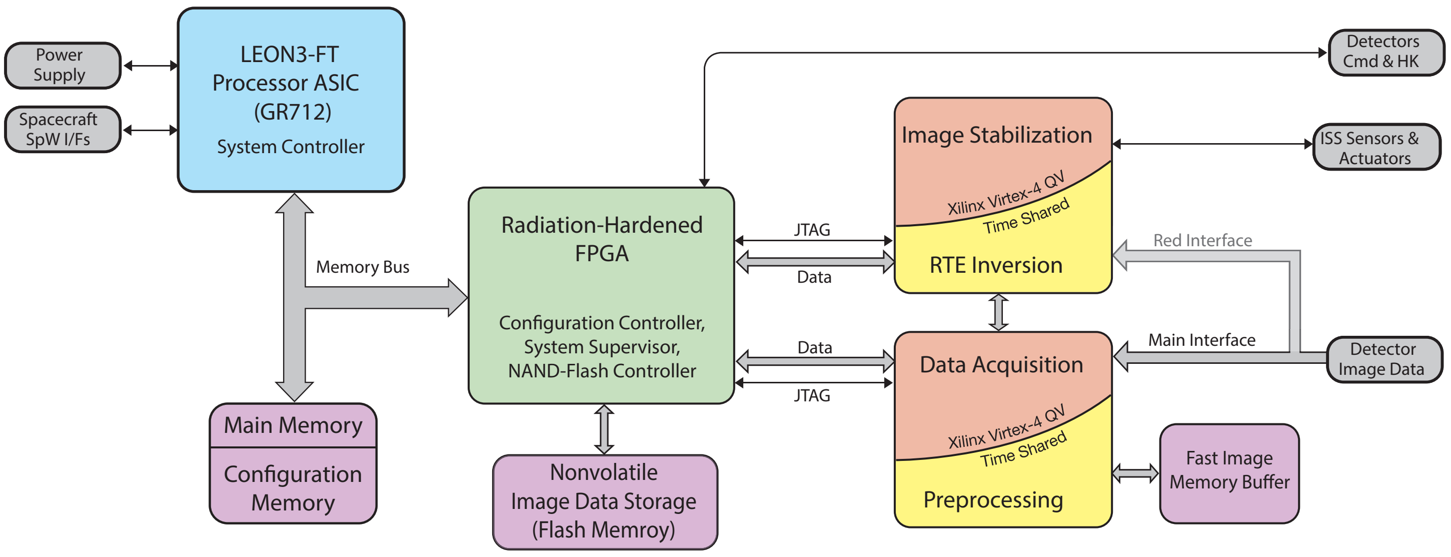


Figure 2 – Architecture of the PHI data-processing unit

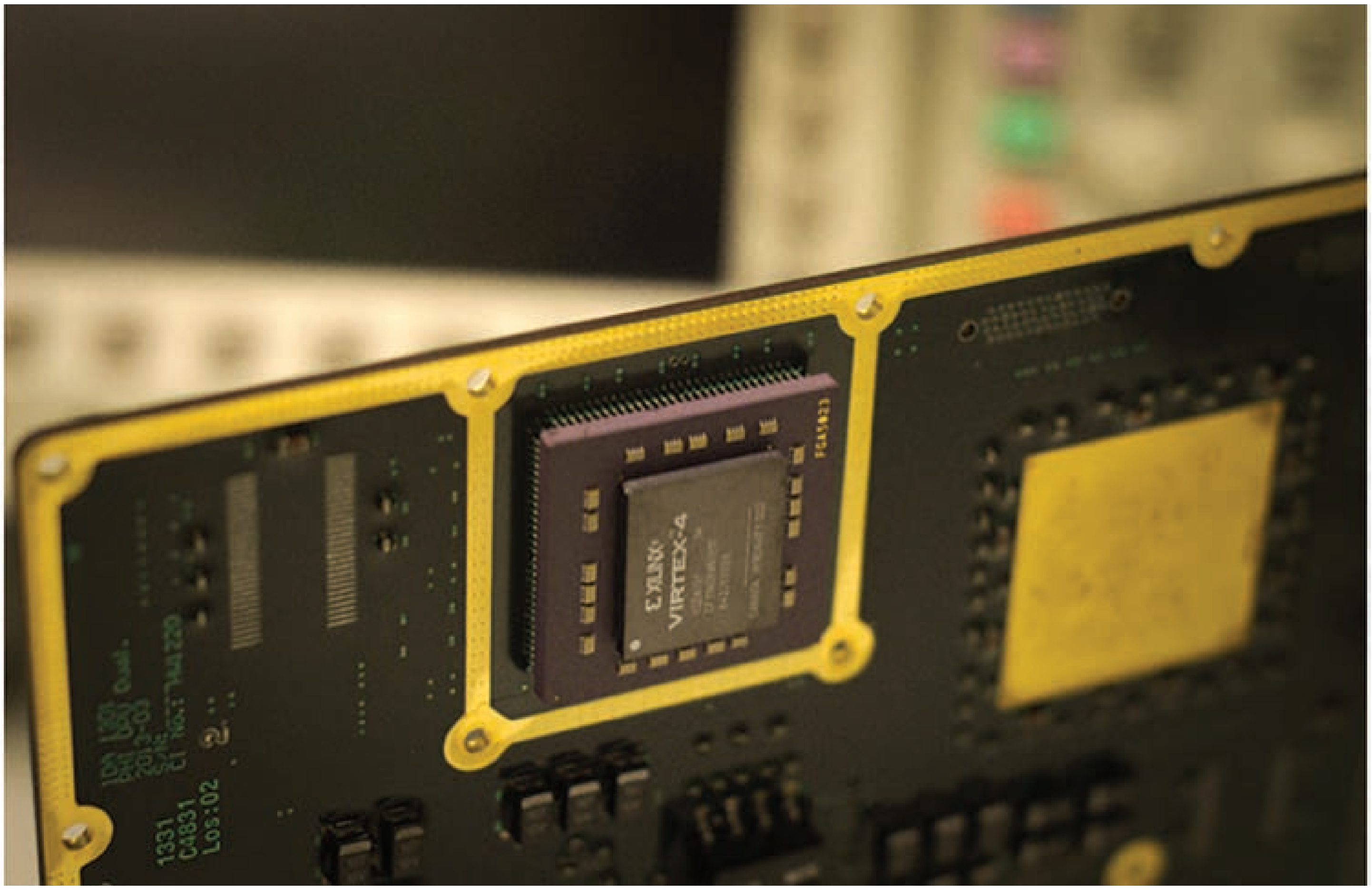


Figure 3 – Qualification board with soldered daisychain device

is a contribution of the Instituto de Astrofísica de Andalucía in Granada, Spain. The design for the controller of the image-stabilization system was developed at the University of Barcelona.

ARCHITECTURE OF THE PHI PROCESSING UNIT

Figure 2 shows the architecture of the data-processing unit. For communication to the spacecraft and the system controller, we use a dedicated and highly reliable GR712 LEON3-FT processor ASIC running at a clock frequency of 50 MHz. The CPU has its own 2-Gbit SDRAM memory and is also connected to 1 Gbit of nonvolatile NOR flash memory that stores software and FPGA bitstream configuration files. For the image acquisition, image stabilization, preprocessing and RTE inversion, we use two Virtex-4QV FPGAs. The possibility of in-flight reconfiguration allows us to effectively utilize these two devices in a time-shared manner. This scheme reduces mass, volume and the power consumption of the platform, which are very important factors for deep-space missions.

A one-time-programmable radiation-hardened FPGA connects the LEON3 system controller and the two Virtex-4 devices. Furthermore, this same FPGA functions as a system supervisor. It provides I/O signals and interfaces to control parts of the hardware and the external power supply, and to communicate with sensors and actuators. Two JTAG interfaces allow this FPGA to write and read back the configuration bitstreams of the two Virtex-4 devices.

To store the large amount of image data, we designed a memory board based on an array of NAND flash devices with a total capacity of 4 Tbits. To address this set of memories, located on a separate board, we developed a NAND flash controller that is also placed in the system-supervisor FPGA. To cope with the relatively slow data rate between the NAND flash array and the processing FPGAs, the data acquisition and preprocessing rely on a fast, external buffer memory. A dedicated network connects the system-controller FPGA with the two Virtex-4 FPGAs off-chip and the NAND-flash memory controller with the processing cores on-chip.

DEALING WITH RADIATION EFFECTS

The Xilinx Virtex-4QV is a radiation-tolerant FPGA, which means that the device will not suffer physical damage through radiation effects. Nevertheless, bit upsets can occur and the design has to mitigate them. Radiation can affect SRAM-based

FPGAs in two layers: the configuration layer and the application layer.

Bit upsets in the configuration layer will mainly alter routing logic and combinational functions. One way to repair errors introduced into this layer is to overwrite the configuration SRAM in certain time intervals, a technique known as scrubbing. We optimized the scrubbing process by doing a read back on the bitstream and we reconfigure a certain configuration frame only when an upset is detected.

Radiation effects induced into the application layer will result in faults of the control logic, for example stuck state machines, or simply wrong values in the data path. We will mitigate the upsets on this layer by using triple-modular redundancy (TMR) and error detection and correction (EDAC) techniques.

For a successful mitigation of upsets in the FPGA design, it's crucial to create mechanisms to protect both

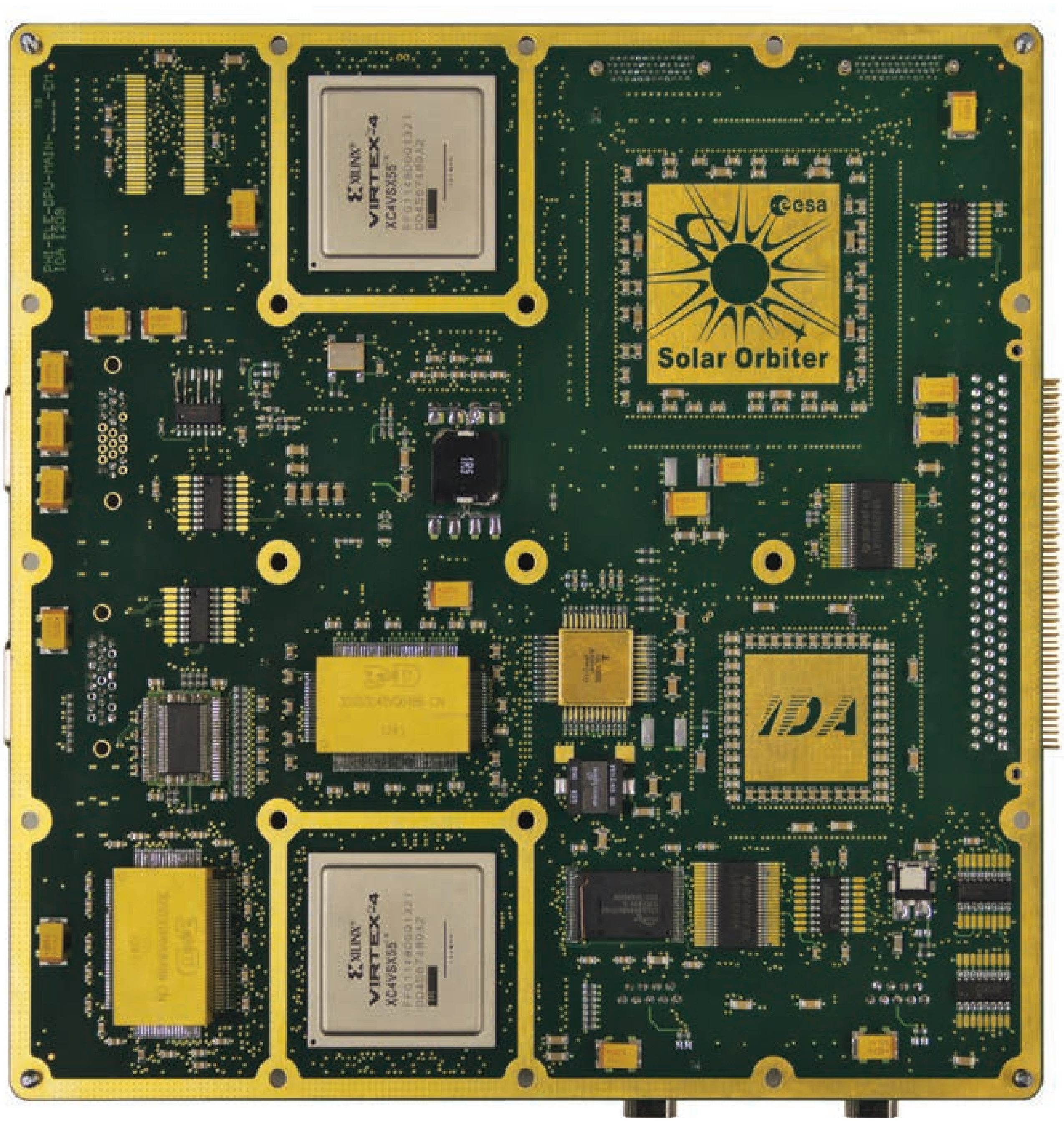


Figure 4 – Bottom side of the PHI data-processing unit engineering model

A high gate count and the ability for in-flight reconfiguration of the Virtex-4 devices made it possible for us to develop a compact and high-performance data-processing platform with reduced size, mass and power consumption.

the configuration layer and the application layer. Integrating only one mitigation scheme alone, scrubbing or TMR, would not be sufficient.

ASSEMBLY QUALIFICATION FOR THE VIRTEX-4QV

Unlike commercial versions of Virtex-4 FPGAs, which come in a flip-chip BGA package, the space-grade Virtex-4QV devices are delivered in a ceramic package. Maintaining a compatible footprint, these devices are equipped with solder columns. When we decided to use Virtex-4QV parts in 2012, no qualified process manufacturer was available in Europe to assemble these CF1140 packages. For this reason we had to start a mission-specific package assembly qualification.

For this purpose, we assembled three representative qualification boards with overall six CF1140 daisy-chain devices (Figure 3). After dedicated shock and vibration tests, we started a thermal-cycling test with close monitoring of resistances of the daisy-chain packages. Before and after each test step, optical and X-ray inspection of the devices proved that no critical physical damage had occurred. We are just finishing the qualification process by means of a destructive micro-sectioning of one PCB.

CURRENT STATUS AND OUTLOOK

After defining a basic architecture for our design including error mitigation and the qualification of the Virtex-4 assembly, we started to work on a prototype of the data-processing unit based on commercial parts. This engineering model fits the final 20 x 20-cm shape of the electronic housing and is already in operation without major problems. It has a mass of 550 grams (without the NAND-flash memory board) and consumes less than 15 watts. The bottom side of this model, equipped with the two Virtex FPGAs, is shown in Figure 4. Currently, we are focusing on finishing the qualification model of our board, equipped with qualified parts.

In summary, the high gate count and the ability for in-flight reconfiguration of the Virtex-4 devices made it possible for us to develop a compact and high-performance data-processing platform with reduced size, mass and power consumption. The data flow of the acquisition and processing perfectly suits a system with two reconfigurable FPGAs.

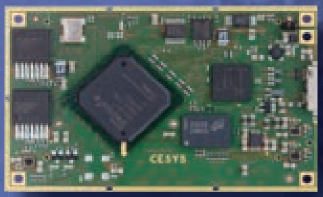
This system is a first step in bringing in-flight reconfiguration technology to deep-space missions. Electronics for the space industry are usually a couple of years behind commercial technology. Today, the Xilinx Zynq®-7000 All Programmable SoC family integrates SRAM-based FPGA technology with multiple processor cores into a single system-on-chip. In coming years it will be of interest for us to see if these types of SoC solutions will also adapt to the space industry's needs.

FPGA

Boards & Modules

EFM-02

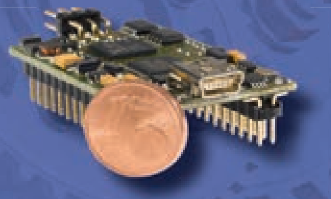
FPGA module with USB 3.0 interface. Ideal for Custom Cameras & ImageProcessing.



- Xilinx™ Spartan-6 FPGA XC6SLX45(150)-3FGG484I
- USB 3.0 Superspeed interface Cypress™ FX-3 controller
- On-board memory 2 Gb DDR2 SDRAM 64 Mb Dual SPI flash
- Samtec™ Q-strip connectors 191 (95 differential) user IO

EFM-01

Low-cost FPGA module for general applications.



- Xilinx™ Spartan-3E FPGA XC3S500E-4CPG132C
- USB 2.0 Highspeed interface Cypress™ FX-2 controller
- On-board memory 4 Mb SPI flash
- Standard 0.1" pin header 50 user IO

CESYS

Hardware • Software • HDL-Design

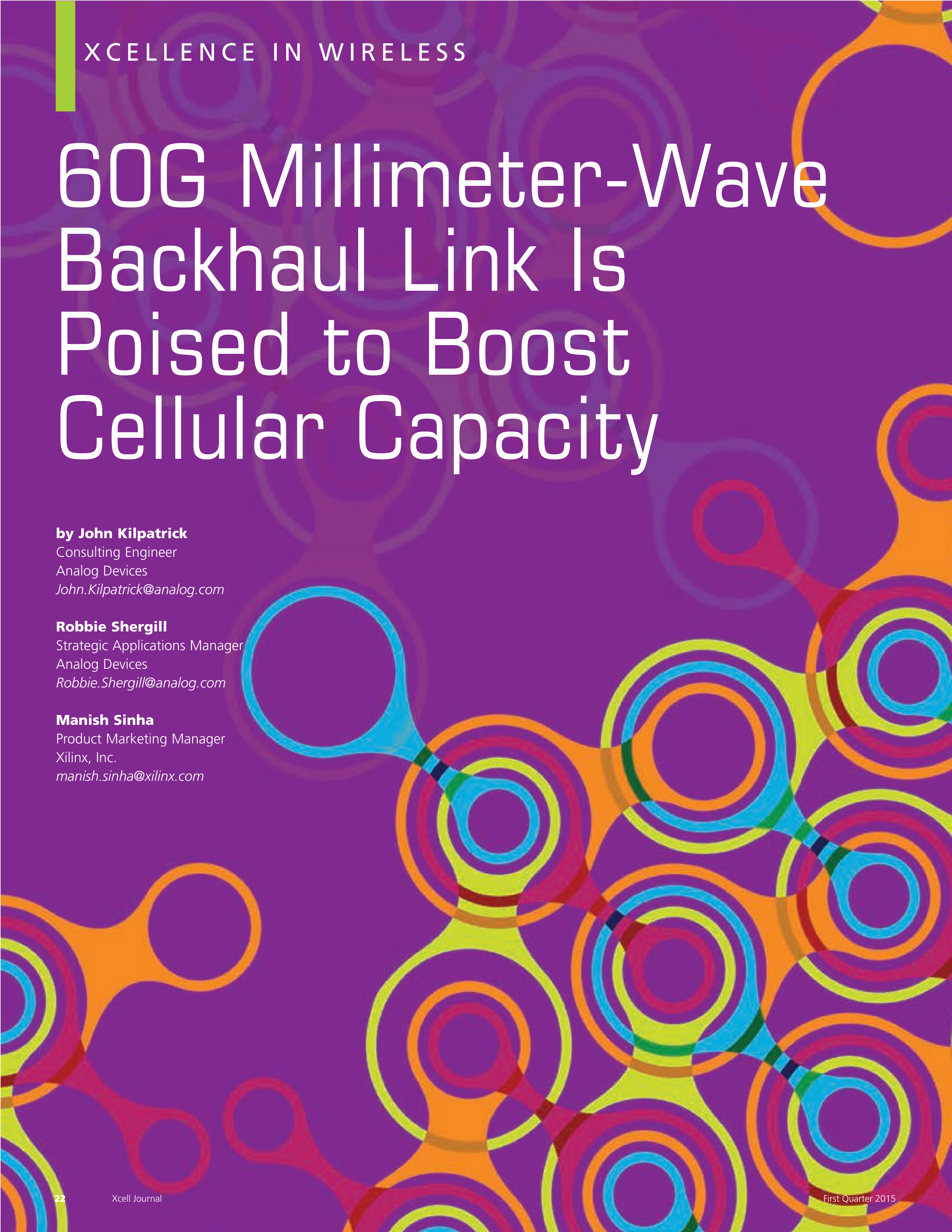
www.cesys.com

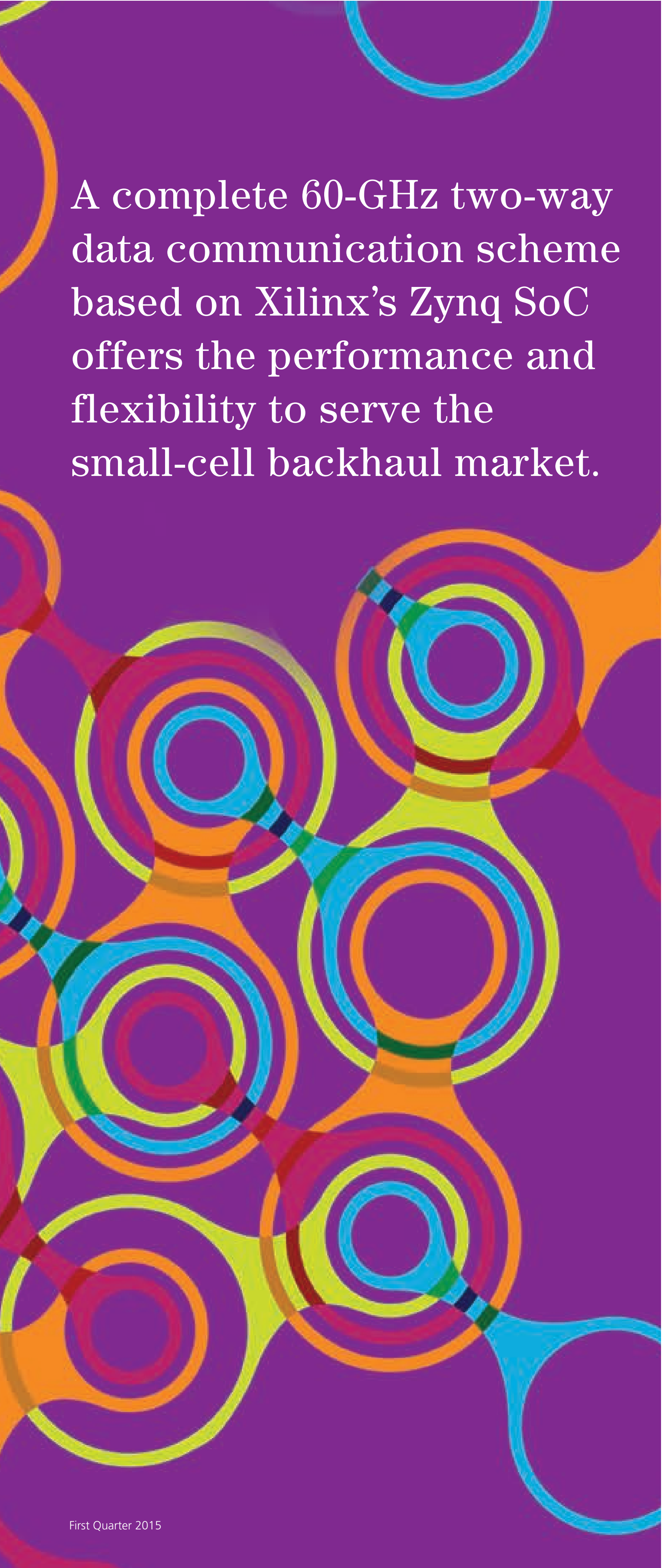
60G Millimeter-Wave Backhaul Link Is Poised to Boost Cellular Capacity

by John Kilpatrick
Consulting Engineer
Analog Devices
John.Kilpatrick@analog.com

Robbie Shergill
Strategic Applications Manager
Analog Devices
Robbie.Shergill@analog.com

Manish Sinha
Product Marketing Manager
Xilinx, Inc.
manish.sinha@xilinx.com





A complete 60-GHz two-way data communication scheme based on Xilinx's Zynq SoC offers the performance and flexibility to serve the small-cell backhaul market.

The ever-increasing demand for data on the world's cellular networks has operators searching for ways to increase the capacity 5,000-fold by 2030 [1]. Getting there will require a 5x increase in channel performance, a 20x increase in allocated spectrum and a 50x increase in the number of cell sites.

Many of these new cells will be placed indoors, where the majority of traffic originates, and fiber is the top choice to funnel the traffic back into the networks. But there are many outdoor locations where fiber is not available or is too expensive to connect, and for these situations wireless backhaul is the most viable alternative.

Unlicensed spectrum at 5 GHz is available and does not require a line-of-sight path. However, the bandwidth is limited and interference from other users of this spectrum is almost guaranteed due to heavy traffic and wide antenna patterns.

Communication links of 60 GHz are emerging as a leading contender to provide these backhaul links for the many thousands of outdoor cells that will be required to meet the capacity demands. This spectrum is also unlicensed, but unlike frequencies below 6 GHz, it contains up to 9 GHz of available bandwidth. Moreover, the high frequency allows for very narrow and focused antenna patterns that are somewhat immune to interference.

A complete 60-GHz two-way data communication link developed by Xilinx and Hittite Microwave (now part of Analog Devices) demonstrates superior performance and the flexibility to meet the requirements of the small-cell backhaul market (Figure 1). Xilinx developed the digital modem portion of the platform and Analog Devices, the millimeter-wave radio portion.

As depicted in Figure 1, two nodes are required to create this link. Each node contains a transmitter (with a modulator) with its associated analog Tx chain and a receiver (with a demodulator) with its associated analog Rx chain.

The modem card is integrated with analog and discrete devices. It contains oscillators (DPLL module) to ensure the accuracy of frequency synthesis, and all the digital functions are executed in an FPGA or SoC. This single-carrier modem core supports modulations from QPSK to 256QAM in channel bandwidths up to 500 MHz, and achieves date rates as high as 3.5 Gbps. The modem also supports both frequency-division duplex (FDD) and time-division duplex (TDD) transmission

Robust modem design techniques reduce the phase noise implications of the local oscillators. Powerful LDPC coding is included for improved performance and link budget.

schemes. Robust modem design techniques reduce the phase noise implications of the local oscillators and powerful LDPC coding is included for improved performance and link budget.

MILLIMETER-WAVE MODEM

The Xilinx millimeter-wave modem solution enables infrastructure vendors to develop flexible, cost-optimized and customizable links for their wireless backhaul networks. This solution is targeted at the Xilinx® Zynq®-7000 All Programmable SoC or Kintex®-7 FPGA devices, which are part of Xilinx's "generation-ahead" 28-nanometer product family.

Xilinx's solution is fully adaptive, is low in power and small in footprint, and can be used to deploy indoor and full outdoor point-to-point links as well as point-to-multipoint microwave links. Just as with its silicon, Xilinx's road map for its millimeter-wave modem solution is very aggressive, and presents operators with the unique ability to deploy scalable and field-upgradable systems.

Figure 2 further details the digital modem as implemented on the Zynq SoC platform. Alongside the programmable logic (PL), the platform's scalable processing system (PS) contains dual ARM® Cortex™-A9 cores with integrated memory controllers and multistandard I/Os for peripherals.

This system-on-chip (SoC) platform is highly flexible. Here, it is used to perform various data and control functions and to enable hardware acceleration. An integrated millimeter-wave modem solution complete with PHY, controller, system interfaces and packet processor is shown in Figure 2. However, based on the required architecture, you could insert, update or remove different modules. For instance, you might choose to implement an XPIC combiner so that you could use the modem in cross-polarization mode with another modem. The solution is implemented in the PL, where serdes and I/Os are used for various data path interfaces such as those between the modem and packet processor, the packet processor and memory, inter-modem or DAC/ADC.

Some of the other important features of the Xilinx modem IP include automatic hitless and errorless state switching through adaptive coding and modulation (ACM) to keep the link operational;

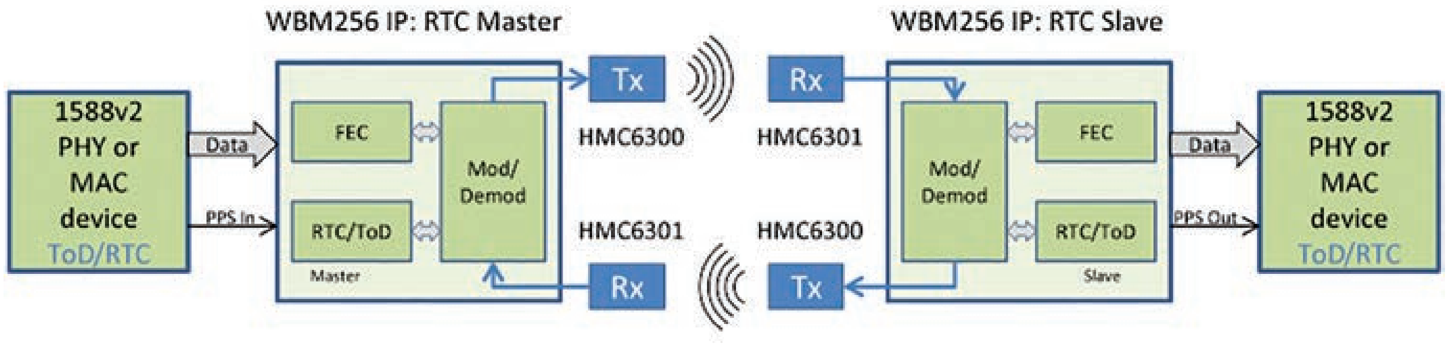


Figure 1 – High-level block diagram of the complete two-way communication link

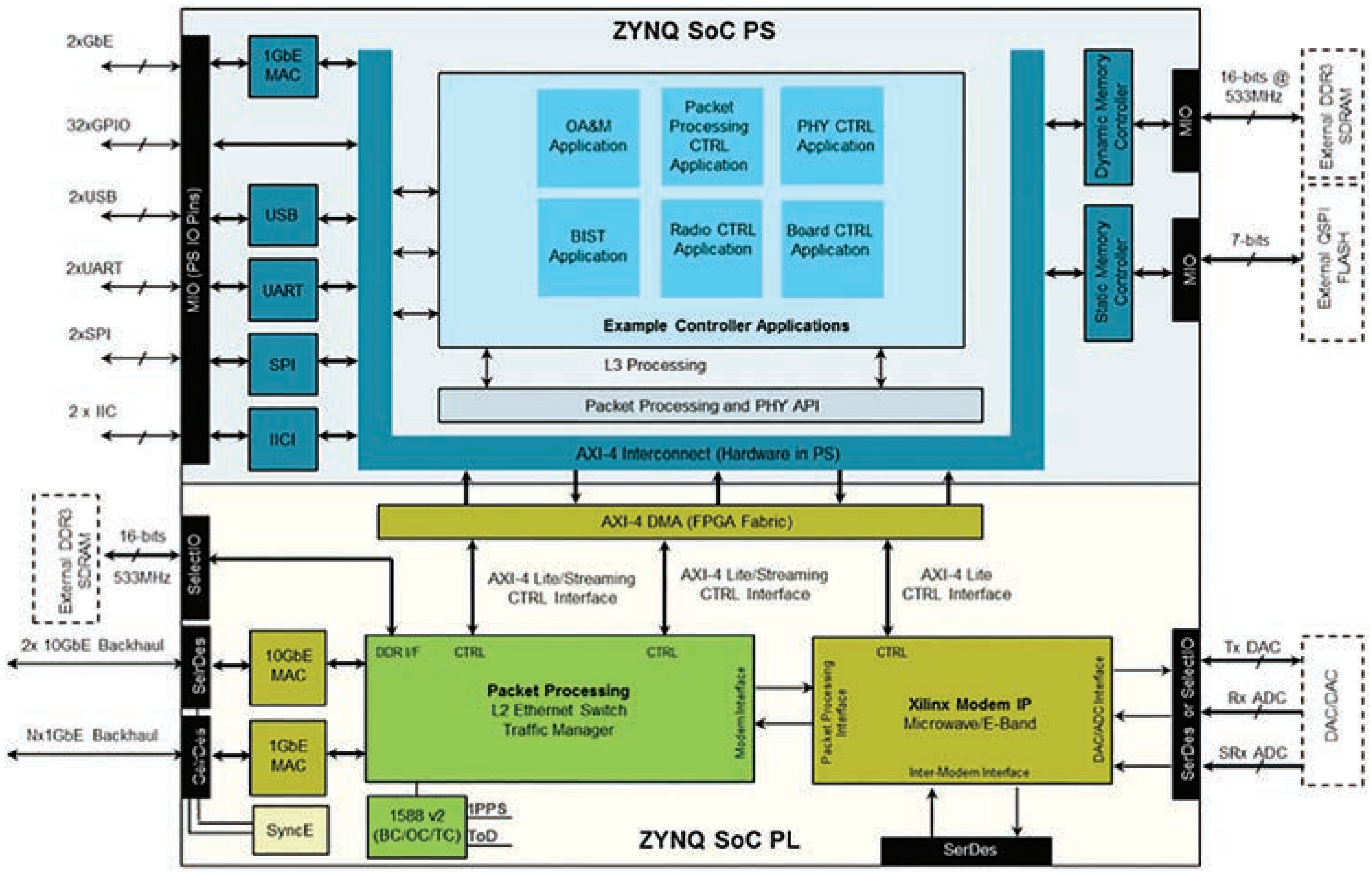


Figure 2 – All Programmable SoC for wireless modem applications

adaptive digital closed-loop predistortion (DPD) to improve RF power amplifier efficiency and linearity; synchronous Ethernet (SyncE) to maintain clock synchronization; and Reed-Solomon or low-density parity check (LDPC) forward error correction (FEC). The FEC choice is based on the design requirements. LPDC FEC is the default choice for wireless backhaul applications, whereas Reed-Solomon FEC is preferred for low-latency applications such as front-haul.

LDPC implementation is highly optimized and exploits FPGA parallelism for the computations done by the encoders and decoders. The result is noticeable SNR gains. You can apply different levels of parallelism by varying the number of iterations of the LDPC core, thereby optimizing the size and power of the decoder. You can also model the solution based on channel bandwidth and throughput constraints.

The Xilinx modem solution also comes with a powerful graphical user interface (GUI) for both display and debug, and is capable of high-level functions such as channel bandwidth or modulation selection as well as low-level ones such as setting of hardware registers. To achieve 3.5-Gbps throughput for the solution shown in Figure 1, the modem IP runs at a 440-MHz clock rate. It uses five gigabit transceivers (GTs) for connectivity interfaces to support ADCs and DACs, and a few more GTs for 10GbE payloads or CPRI interfaces.

MILLIMETER-WAVE TRANSCEIVER CHIP SET

In late 2014, Analog Devices released its second-generation silicon germanium (SiGe) 60-GHz chip set, significantly enhanced and optimized for the small-cell backhaul application. The HMC6300 transmitter chip is a complete analog baseband-to-millimeter-wave upconverter. An improved frequency synthesizer covers 57 to 66 GHz in 250-MHz steps with low phase noise and can support modulations up to at least 64QAM. Output power has increased to roughly 16-dBm linear power, while an integrated power detector monitors the output power so as not to exceed the regulatory limits.

The transmitter chip offers either analog or digital control of the IF and RF gains. Analog gain control is sometimes needed when using higher-order modulations, since discrete gain changes can be mistaken for amplitude modulation, leading to bit errors. Digital gain control is supported using the built-in SPI interface.

For applications requiring even higher-order modulation in narrow channels, an external PLL/VCO with even

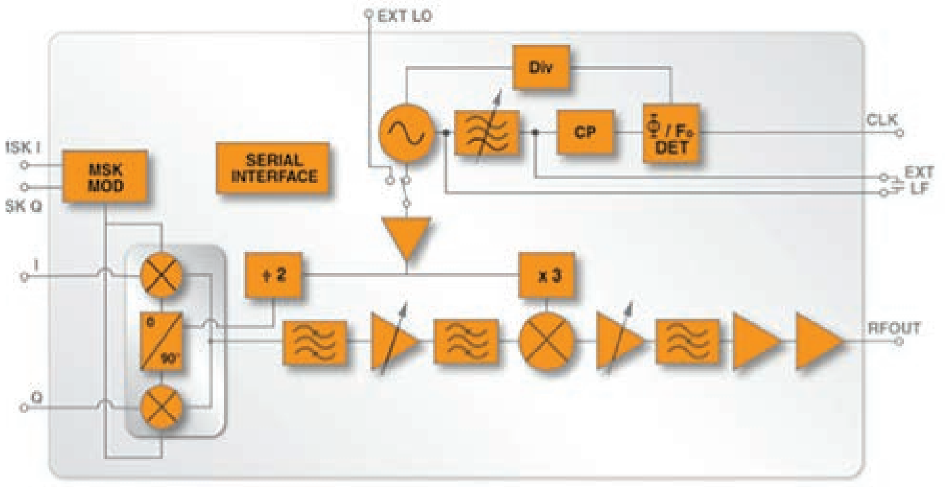


Figure 3 – HMC6300 60-GHz transmitter IC block diagram

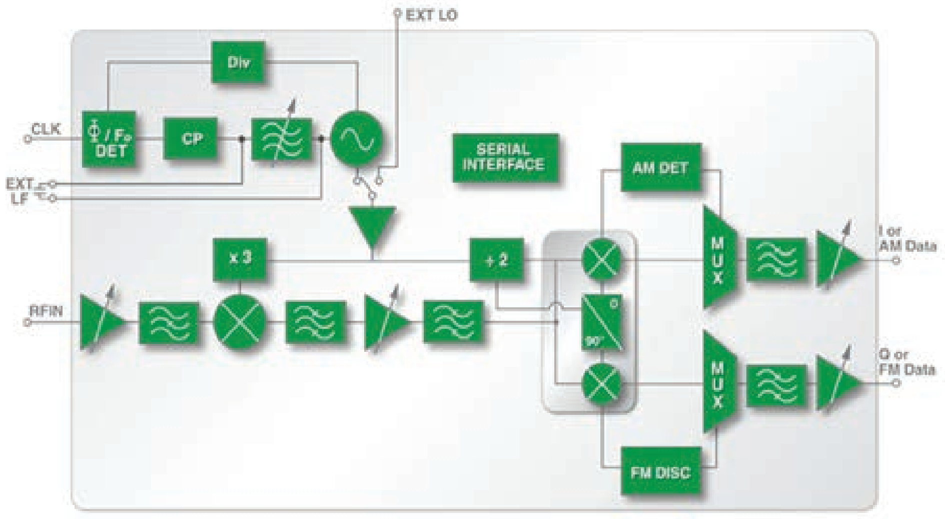


Figure 4 – HMC6301 60-GHz receiver IC block diagram

lower phase noise can be injected into the transmitter, bypassing the internal synthesizer. Figure 3 shows a block diagram of the HMC6300.

The transmitter supports up to 1.8 GHz of bandwidth. An MSK modulator option enables low-cost data transmissions up to 1.8 Gbps without the need for expensive and power-hungry DACs.

Complementing this device is the HMC6301 receiver chip, likewise optimized to meet the demanding requirements of small-cell backhaul. The receiver features a significant increase in the input P1dB to -20 dBm and IIP3 to -9 dBm to handle short-range links where the high gain of the dish antennas lead to high signal levels at the receiver input.

Other features include a low, 6-dB noise figure at the maximum gain settings; adjustable low-pass and high-pass baseband filters; the same new synthesizers as found in the transmitter chip to support 64QAM modulation over the 57- to 66-GHz band; and either analog or digital control of the IF and RF gains.

A block diagram of the HMC6301 receiver chip is shown in Figure 4. Note that the receiver also contains an AM detector to demodulate amplitude modulations such as on/off keying (OOK). Also, an FM discriminator demodulates simple FM or MSK modulations. This is in addition to the IQ demodulator that is used to recover the quadrature baseband outputs for QPSK and more-complex QAM modulations.

Both the HMC6300 transmitter and HMC6301 receiver will be available in a 4 x 6-mm BGA-style wafer-level package. They will be designated the HMC6300BG46 and HMC6301BG46 and are scheduled for sampling in early 2015. These surface-mount parts will enable the low-cost manufacturing of the radio boards.

A block diagram of an example millimeter-wave modem and radio system is shown in Figure 5. In addition to the FPGA, modem software and millimeter-wave chip set, the design also contains a number of other components. They include the AD9234 dual-channel 12-bit, 1-Gsample/second ADC; the AD9144 quad-channel 16-bit, up to 2.8-GSPS Tx-DAC; and the HMC7044 ultralow-jitter clock synthesizer with support for the JESD204B serial data interface that is employed on both the ADC and the DAC ICs.

DEMONSTRATION PLATFORM

Xilinx and Analog Devices have jointly created a demonstration platform implementation featuring the FPGA- based modem on the Xilinx KC705 development board, an industry-standard FMC board containing ADCs, DACs and clock chip, and two radio module evaluation boards (Figure 6). The demo platform includes a laptop for modem control and visual display, and a variable RF attenuator to replicate the path loss of a typical millimeter-wave link. The Xilinx KC705 development board features the Kintex-7 XC7K325T-2FFG900C FPGA executing the WBM256 modem firmware IP. An industry-standard FMC mezzanine connector on the development board is used to connect to the baseband and millimeter-wave radio boards.

The millimeter-wave modules snap onto the baseband board. The modules have MMPX connectors for the 60-GHz interfaces as well as SMA connectors for optional use of an external local oscillator.

This platform contains all the hardware and software needed to demonstrate point-to-point backhaul connections of up to 1.1 Gbps in 250-MHz

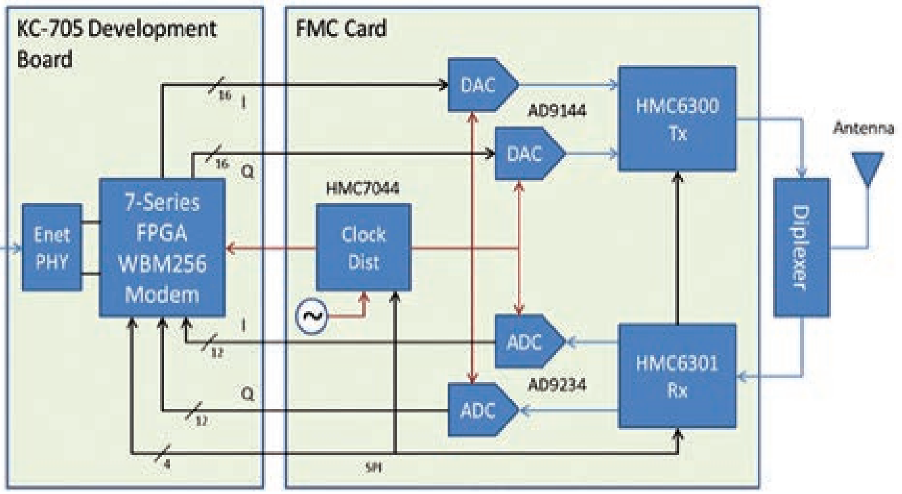


Figure 5 – Example reference design using Xilinx and Analog Devices ICs

channels for each direction of a frequency-division duplex connection.

MODULAR AND CUSTOMIZABLE

FPGAs are increasingly being used in various wireless backhaul solutions, since the platforms based on them can be highly modular and customizable, thereby reducing the total cost of ownership for the OEMs. Owing to significant power improvements in its 7 series FPGA/SoC families and high-performing wideband IP cores, Xilinx expects its millimeter-wave modem solution to be a front-runner for the small-cell backhaul application. Xilinx FPGAs and SoCs are suitable for high-speed and power-efficient designs, and its high-speed GTs can be used effectively for wideband communications and switching functions. Xilinx's solution can be scaled to support multiple product variations, from lower-end small-cell backhaul products operating at a few hundred megabits per second to 3.5 Gbps on the same hardware platform.

For the radio portion, the transceivers have now been integrated into silicon-based ICs and packaged into surface-mount parts, allowing for low-cost manufacturing. Analog Devices' millimeter-wave chip set meets the wireless backhaul needs of the small-cell deployments and provides market-leading performance in power, size, flexibility and functionality. Analog Devices also provides industry-best data converters and clock-management ICs that are critical components of this complete solution. Together, the two companies intend to drive the industry adoption of this exciting technology.

Reference

1. "Evolutionary and Disruptive Visions Towards Ultra High Capacity Networks," IWPC, April 2014

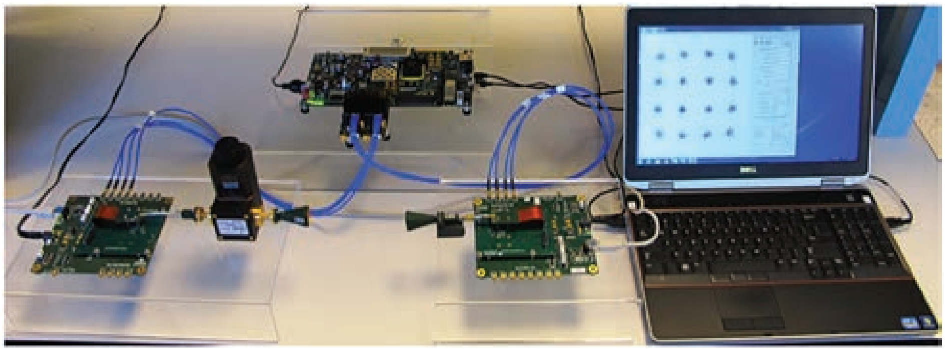
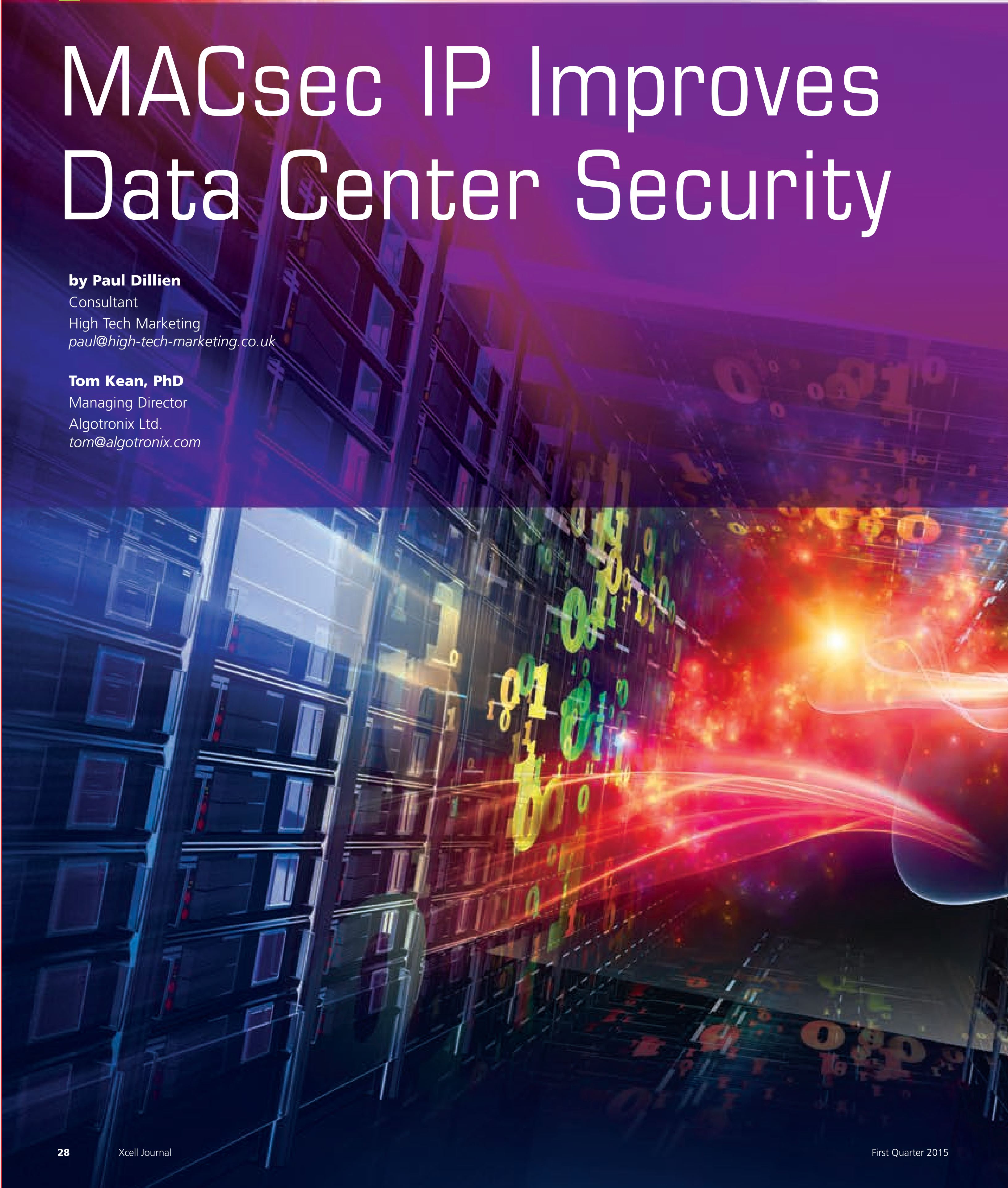


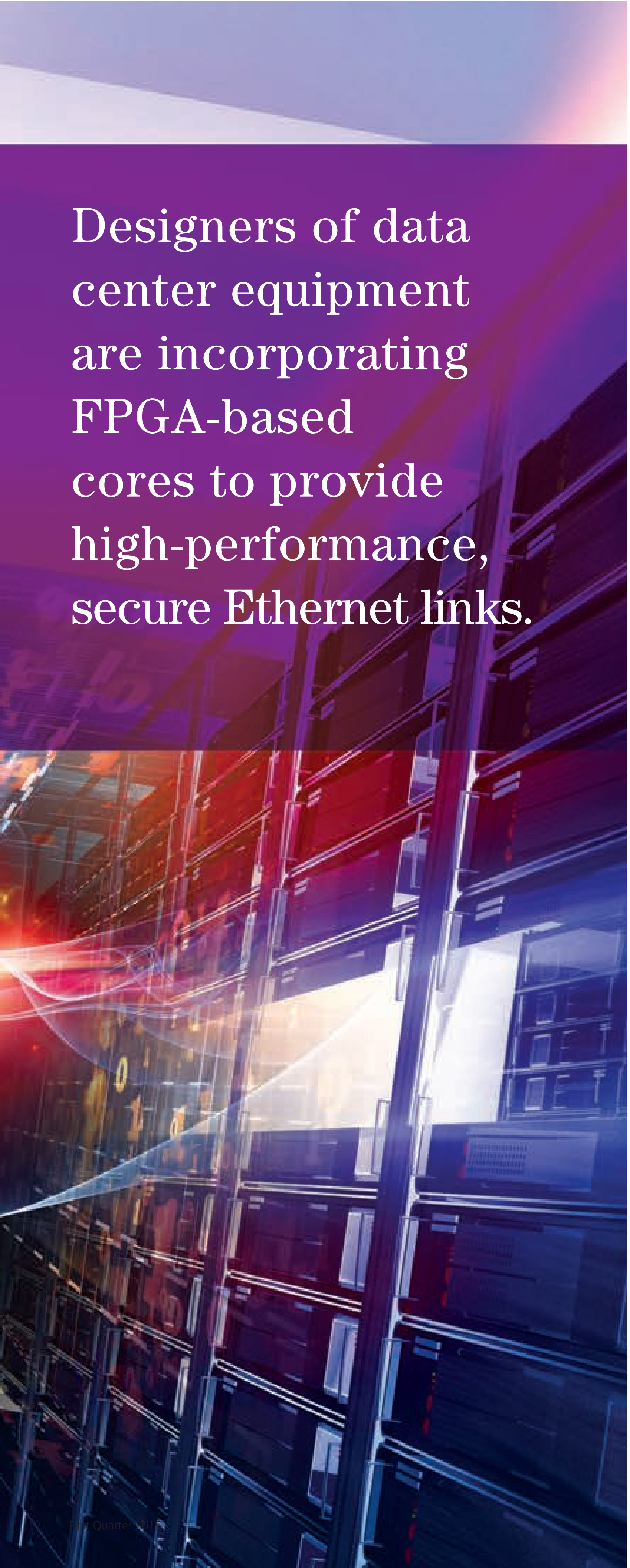
Figure 6 – The demonstration platform in action

MACsec IP Improves Data Center Security

by Paul Dillien
Consultant
High Tech Marketing
paul@high-tech-marketing.co.uk

Tom Kean, PhD
Managing Director
Algotronix Ltd.
tom@algotronix.com





Designers of data center equipment are incorporating FPGA-based cores to provide high-performance, secure Ethernet links.

Cloud storage and the outsourcing of IT services hold a number of attractions for IT managers, because these options can save costs and reduce the support burden. But one big disincentive about allowing sensitive data outside a company's firewall is the concern about security. The hesitation is understandable, as information is one of the most valuable assets for many companies, whether it is accountancy, customer or manufacturing-related data.

But now equipment manufacturers can add performance and raise the bar on security with a Xilinx® FPGA-based solution. A comprehensive security subsystem from Algotronix, which meets the new Ethernet standard known as MACsec, uses a high-performance, low-latency and power-efficient intellectual-property (IP) core inside a Xilinx FPGA.

An FPGA-based solution is much faster than one based in software. In addition, the dedicated hardware offloads the system processor and frees it for other tasks, such as deep packet inspection. Alternatively, the designer could use a lower-cost processor.

ENCRYPTION AND AUTHENTICATION

An obvious tactic for protecting information is to encrypt data as it transits the network and moves around the data center. Encryption ensures that, should the data be intercepted by an unauthorized party sniffing the link, it cannot be read. Ideally, too, the data should be authenticated to ensure its integrity. Message authentication is designed to detect where the original encrypted data has been altered, either by means of a transmission error or from being maliciously tampered with by an attacker seeking to gain an advantage.

Ethernet transmission has grown to dominate communications because it is both efficient and extendable to high-speed transmissions. The popularity of the Ethernet standard has driven down costs, making it even more attractive, and this virtuous circle ensures the continuance of Ethernet as the Layer 2 technology of choice. However, up until a few years ago, the specification did not include any encryption, leaving the job to technologies such as IPsec that operate in the upper layers of the communications protocol stack.

Now, a new extension to Ethernet adds a raft of security measures, under the specification IEEE 802.1AE. Specified a few years ago, this technology features an integrated security system that encrypts and authenticates messages while also detecting and defeating a range of attacks on the network. The specification is known as the Media Access Control Security standard, or more commonly as MACsec, and Algotronix set out several years ago to produce

IP cores that provide hardware-accelerated encryption over a range of data rates. (Algotronix also supplies an intellectual-property core for IPsec that has a very similar interface to the MACsec product and would be a good choice in systems that need to support both standards.)

A brief overview of the MACsec system will serve to illustrate the comprehensiveness of the specification, as well as give an insight into the complexity of implementing it.

TRUSTED ENTITIES

The concept of MACsec is that nodes on a network form a set of trusted entities. Each node can receive both encrypted and plaintext messages, and the system policy can dictate how each is handled. The core includes a bypass option for plaintext messages, which are not authenticated or verified. Unlike protocols such as IPsec, which operates at Layer 3/4 and is an end-to-end technology, MACsec decrypts and verifies each packet whenever a packet enters or leaves an Ethernet LAN. MACsec is suitable for Ethernet topologies such as star-connected or bus-based LANs, as well as point-to-point systems.

The MACsec specification uses what are called Security Entities (SecY), an approach in which each node or entity has a unique key linked with its Ethernet source address. We designed the 1G variant of the core to support multiple virtual SecYs. As a result, a single Ethernet MAC can have multiple MACsec SecYs associated with it for applications like multiple-access LANs. MACsec typically works in conjunction with IEEE 802.1X-2010 or the Internet Key Exchange (IKE), which provides the secure key distribution around the network.

The reason that data centers might choose to use Layer 2 connectivity for moving packets inside the center is to achieve high speed with a minimum of latency and overhead data in the packet. By contrast, in communications using secure Layer 3 technologies such as IPsec, the message has to be passed up the stack for processing, with added latency.

A Layer 2 solution also eliminates the complexities of creating Layer 3 security policies. Data centers can adopt MACsec to provide protection behind the firewall or use it on direct links between data centers. The system administrator can authorize equipment to communicate in a secure fashion. The equipment can detect errors or misuse, such as attempted denial of service (DOS).

PRIME FOR PROGRAMMABILITY

A customizable FPGA solution is ideal for MACsec, as the market is fragmented by differing requirements. Originally, MACsec was conceived as a technology to be applied to metropolitan-area networks, but it is now also finding use in data centers, which increases the overall demand for an FPGA-based solution.

It was a natural evolution for Algotronix to develop a MACsec core, because we had already created a range of crypto engines called AES-GCM. These cores operate at 1G, 10G and 40G. We achieved that speed by pipelining, increasing the clock speed and moving progressively from, say, Xilinx Artix® to Kintex® devices and then on to Virtex® FPGAs. We are adopting these techniques to push the throughput to 100G on Virtex UltraScale™ devices.

The performance we can achieve using an IP core in an FPGA is selectable to support anywhere from Gigabit Ethernet to 10 GbE (that is, the actual throughput through the core under worst-case conditions), with 40G and 100G versions planned. This is much faster than a software-based system could achieve. The cores are normally connected directly to the hardware MAC, as shown in Figure 1, because software on the embedded processor on the FPGA chip can struggle to transfer data fast enough to handle their throughput. If the security functions are implemented in hardware and additionally, unencrypted keys are never available to software, then the system is less vulnerable to common software-based attacks such as Trojan horses and viruses. This makes it easier to analyze for

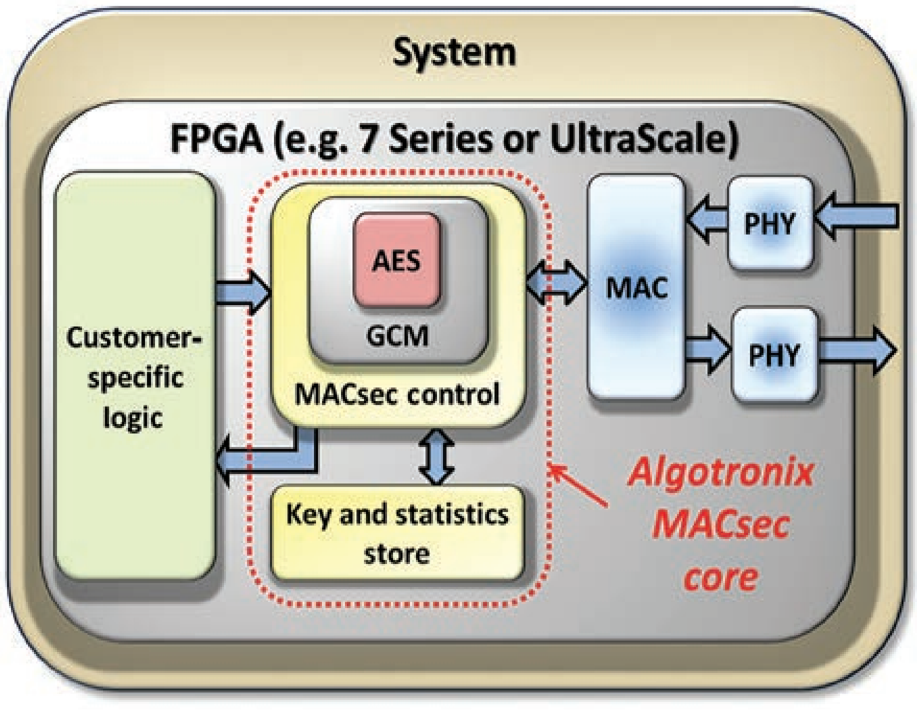


Figure 1 – The MACsec IP core sits entirely within the FPGA for maximum security.

vulnerabilities than in a case where IT professionals must consider the entire software side of the system.

Another important consideration is the dramatic power saving in systems where FPGAs accelerate algorithms such as cryptographic functions that would otherwise be implemented in software. FPGAs are dramatically more power efficient than a software solution.

One useful attribute built into all Algotronix encryption cores is the ability to implement crucial blocks, called S-Boxes, in either Block RAM or in the lookup tables (LUTs) of the FPGA fabric. This option allows customers to squeeze the design into the available resources by trading off the two resource types, for example using Block RAM to implement the S-Boxes if the design outside of the MACsec core did not use BRAM heavily or LUTs if it did.

INS AND OUTS OF MACSEC

The MACsec system features the concept of each source of data using different encryption keys. When a message is received, the receiver looks it up in a list held in on-chip CAMs to determine the correct key to use to decrypt the packet. Each packet is also numbered to ensure that repeated or replayed packets can be detected and rejected, a technique that defends against “man-in-the-middle” attacks.

MACsec also collects statistics about the number of packets that are rejected and the reasons for rejection. Providing statistics to support detection of attacks is a further layer of security beyond the basic cryptographic privacy, authentication and replay prevention, allowing a system manager to proactively respond to an attack in progress.

We took the approach of “wrapping” the MACsec logic around the proven AES-GCM core. That said, designing an efficient and fast encryption core is only part of the design challenge. The MACsec specification is extensive and includes many variables. For example, the standard originally specified only 128-bit encryption keys. With 128-bit keys, the data undergoes 10 transformations (known as rounds) to complete the encryption process within the core. The standard was later revised to include an option for 256-bit keys, which have 14 rounds of processing through the encryption. This is achieved by adding pipeline stages and increasing the width of the memory used for storing the keys.

MACsec is agnostic as to the Ethernet traffic type, and it is transparent to higher-layer protocols. With the introduction of these cores, it’s easy to add MACsec to systems to provide an additional layer of protection in a network. Sites equipped with MACsec can still communicate with other sites, but without the extra security of MACsec.

Ethernet packets are fed to the MACsec core from the media-access controller (MAC). You can build a compact and efficient solution using, say, the 1G MACsec core in conjunction with on-chip transceivers and a trimode Ethernet MAC (TEMAC). Each of the packets contains the destination and address of the source that initiated its transmission. This standard is retained in a MACsec system, but an important aspect is that in a multihop transmission, the “source” will be the address of the last equipment to forward the packet. So, unlike IPsec—which can be considered an end-to-end scheme—MACsec works on a hop-by-hop basis. For each hop, MACsec requires that all encrypted data on the ingress is decrypted and then re-encrypted with the unique key assigned to that equipment for forward transmission. The decrypted plaintext allows the option for packet inspection at each stage, as illustrated in Figure 2, and can be used by traffic managers to regulate the flow of data.

In the MACsec standard, the header shown in Figure 3 includes an additional field known as the MAC Security TAG (SecTAG), which defines the EtherType and flags whether the packet is encrypted or not. Authentication is achieved by appending data to the end of the message in a field called ICV. The ICV works with the encryption key to authenticate the frame, including the header and MACsec tag, to ensure that not even the source or destination address of the frame could be manipulated. We implemented this logic in the FPGA fabric to ensure that it would have fast and predictable timing to minimize any latency.

The MACsec core includes a lookup table that is linked to each source address. The table includes the key that needs to be used to successfully decrypt the message, and we designed this feature to be implemented efficiently in the LUTs and Block RAM on the devices. We exploited the flexibility of the FPGA solution by designing the core with implementation options such as a choice between 128- and 256-bit keys and the ability to vary the number of virtual SecYs that the core supports.

Another useful feature of the new standard is the collation of statistics by MACsec at the packet level. The system administrator can, for example, see how many packets were rejected because they were delayed, failed integrity checks due to an invalid decryption key or used the wrong key, and compare those statistics with the number of correct packets transmitted.

The MACsec standard has a simplified option for point-to-point applications. This eliminates the need for a CAM to determine the key from an explicit Secure Channel Identifier in the packet and an option for point-to-multipoint operation. Our core also supports multiple virtual SecYs associated with a single Ethernet so that different keys can be used to encrypt data transmitted from that MAC to different destinations. The MACsec standard defines this kind of configuration as a multi-access local-area network, since it is as if the destinations were on different Ethernet LANs. This feature allows the system to partition the receiving equipment by encrypting the output with different keys.

A data center might use multiple SecYs to create a virtual division so that data from Customer A is partitioned

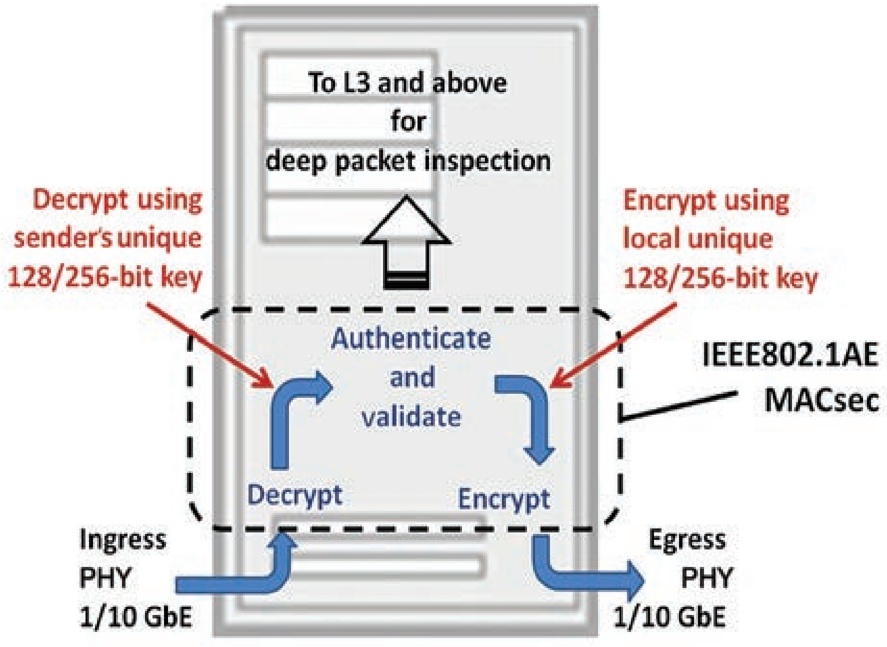


Figure 2 – The message is decrypted on the ingress port and encrypted on the egress port.

from that of Customer B by virtue of a unique encryption key. Communications internally in a data center could, if required, be organized to segregate selected racks to provide virtual isolation areas. This capability can address data integrity and separation concerns in data center and cloud applications. Whether from an accidental wrong connection or a malicious act (see Figure 4), the MACsec system will detect packets that are unauthenticated and the system administrator can set the policy to quarantine or delete them.

All data encryption and decryption are performed at the port level. Apart from the additional MACsec header and small added latency, there is no overhead or performance impact when turning on port-level encryption.

Equipment vendors can use these cores today to differentiate their systems by incorporating an encrypted Ethernet Level 2 scheme compliant with IEEE 802.1AE. Cloud-based users, who may be mutually suspicious of other customers, can benefit from the confidentiality and data source authentication MACsec offers for added reassurance that their data is protected. Equipment manufacturers have a choice of IP cores that are available to cover the needs of 1- and 10-Gbit Ethernet throughputs. The architectural design makes it easy to achieve 10 Gbps in Kintex or Virtex FPGA devices. The design supports both jumbo frames and minimum-size packets with a key change on every packet. This scenario represents the worst-case situation for the system. The cores comply with the full specification, and each MACsec core can support a range of popular FPGA families.

COMES WITH SOURCE CODE

Algotronix takes the unusual step of supplying the HDL source code for every core licensed. The main motivation is to allow customer inspection so as to prove that the code has no virus or Trojan code incorporated, and that it cannot be forced into unauthorized states or operations. Having the source code therefore reduces the cost and complexity of a security audit for customers. In addition, the source code speeds up the design process, because engineers can easily experiment with configuration variables such as encrypt, decrypt or encrypt/decrypt and with key length, and can see the state of signals within the

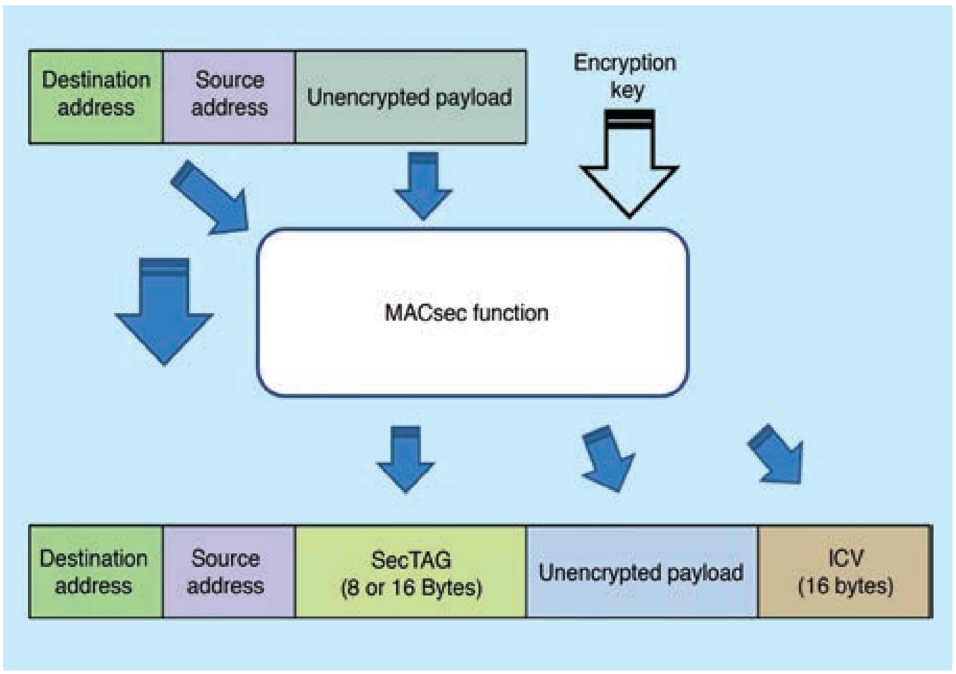


Figure 3 – The MACsec frame structure includes a field known as the MAC Security TAG (SecTAG), which defines the EtherType and flags whether the packet is encrypted.

core in their own simulations. You can configure the cores for high throughput by implementing a wide data path or for minimum FPGA footprint by selecting a narrow data width. Further benefits of having source code are that it is easier to understand the operation of the core; that makes documenting and archiving easier and quicker.

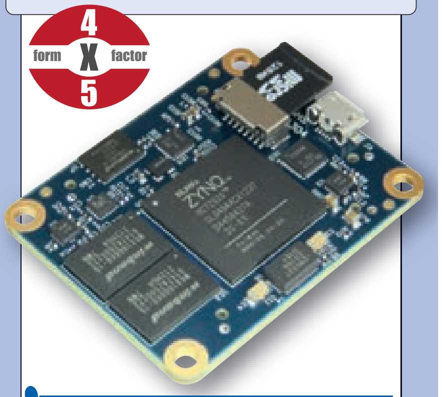
An extensive verification testbench is also included, allowing customers to confirm the correct operation in tools such as ModelSim. The testbench includes a behavioral model of MACsec and a self-checking version of the MACsec IP core where the outputs of the synthesizable hardware are checked against the behavioral model. This self-checking design can be instantiated in user simulations, making it easy to test the core in the context of the actual user design and providing useful diagnostic messages if it is driven incorrectly.

As there are so many options available in the core, the precise resource count will depend on your choice of parameters such as data rate, key length and number of SecYs selected, among others. However, the 10G MACsec core listed on the Intellectual Property section of the Xilinx website uses 6,638 slices, 20,916 LUTs and 53 BRAM blocks. Contact Algotronix for licensing options.

The combination of low-power Xilinx FPGAs and the Algotronix MACsec core offers a high-performance and low-latency solution for equipment manufacturers to differentiate their products. The security features allow data centers to assure their customers of confidentiality, while also enabling security administrators the ability to detect and defeat malicious acts.

Figure 4 – MACsec will reject packets that arrive via wrong connections, either accidentally or maliciously.

All Programmable
FPGA and SoC modules



rugged for harsh environments
extended device life cycle

Available SoMs:

ZYNQ KINTEX 7
ARTIX 7 SPARTAN 6

Platform Features
- 4×5 cm compatible footprint
- up to 8 Gbit DDR3 SDRAM
- 256 Mbit SPI Flash
- Gigabit Ethernet
- USB option

XILINX®
ALLIANCE PROGRAM
CERTIFIED MEMBER — BASE

Design Services
- Module customization
- Carrier board customization
- Custom project development

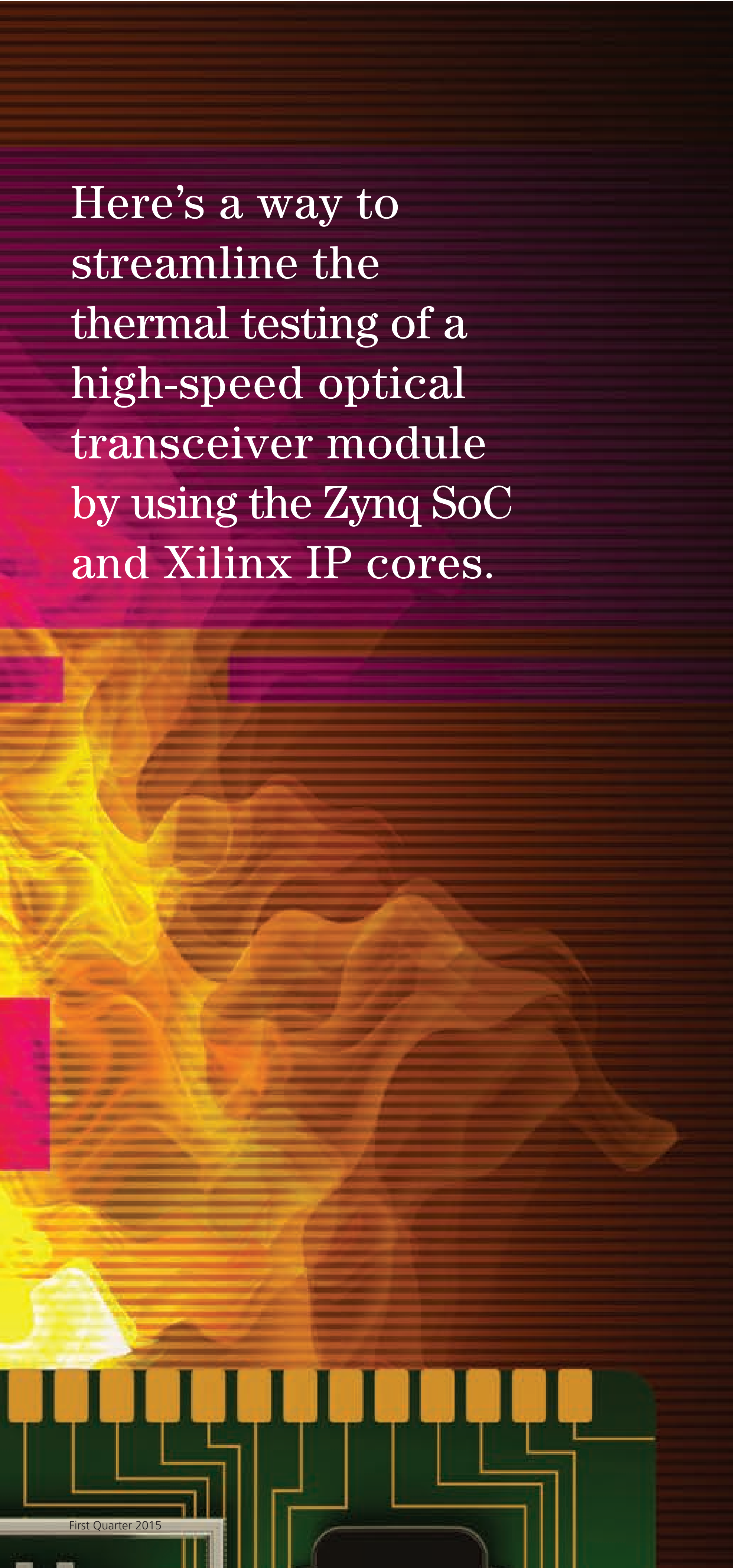
trenz electronic

difference by design

www.trenz-electronic.de

Simplify Your 'Hot' Testing with Xilinx's Zynq SoC

by Lei Guan
Member of Technical Staff
Bell Laboratories, Alcatel Lucent Ireland
lei.guan@alcatel-lucent.com



Here's a way to streamline the thermal testing of a high-speed optical transceiver module by using the Zynq SoC and Xilinx IP cores.

A

As the transmission speed of optical transceiver modules in data centers rises ever higher, the temperature of each chassis in a data center is also rising dramatically. The increase in temperatures becomes compounded when these modules are stacked on top of one another in racks that are flanked by even more racks of speedy but hot modules. This compounded rise in temperature can cause chips to exceed their thermal limits, creating catastrophic chip failures that in turn could adversely affect entire data center systems. Thus, it's imperative that engineers designing optical transceiver modules take thermal properties into account. Designers must zero in on the heat sources and attempt to keep them in check with effective cooling methods at the module and even rack level.

To test the thermal properties of optical modules, engineers traditionally had two choices. They could use a complicated network data generator to create high-speed (10-Gbps) links and then test the thermal properties of the optical modules; or they could utilize a so-called "thermal-equivalent" module with preset tunable voltage and current to mimic the thermal situation and evaluate the thermal properties without using any real high-speed data.

Neither of these approaches is optimal. The first approach is a costly operation due to the need for a professional high-speed network data generator, while the second method is too abstract. A thermal-equivalent module cannot fully reflect the temperature variation driven by the physical switching behavior.

But recently, my team at Bell Laboratories, Alcatel Lucent Ireland, radically simplified this process by using a Xilinx®

I picked the Xilinx ZC706 evaluation board because the GTX transceivers on the main device can easily achieve single-line 10-Gbps data transmission.

Zynq®-7000 All Programmable SoC platform and Xilinx intellectual-property (IP) cores to do the same job. Let's take a closer look at how we accomplished this simplification of testing.

PREDESIGN ANALYSIS

The fundamental requirement of this type of thermal testing is to stimulate the XFP optical transceiver continuously with 10-Gbps data while using an IR camera to track and characterize the temperature variation.

I picked the Xilinx ZC706 evaluation board as the development host, because the GTX transceivers on the main device, the Zynq-7000 SoC XC7Z045 (speed grade -2), can easily achieve single-line 10-Gbps data transmission. The Zynq SoC device contains a processing system (PS) built around an ARM® core and a Kintex®-7 FPGA programmable logic (PL) fabric. Initially, resources at the PL die are enough for handling the 10-Gbps duplex data transmission. Then we can use the PS to generate particular user data patterns if they are required in the future.

Our thermal group provided a Finisar XFP evaluation board as the optical transceiver housing. This FDB-1022 evaluation board is a powerful host for evaluating the state-of-the-art 10-Gbps XFP optical transceivers. SMA connectors are provided for differential data inputs and outputs. The board can be configured to allow a direct connection of a 1/64 clock (that is, 156.25 MHz = 10 GHz/64) via SMA connectors for clocking the module.

SYSTEM DESIGN

I've found over the course of my seven years of doing FPGA development that you can significantly reduce your design cycle by using as many Xilinx cores as possible. In this design, I kept the same strategy and started from the Integrated Bit Error Ratio (IBERT) core, which you can utilize to perform pattern generation and verification to evaluate the GTX transceivers on the Zynq SoC. Then, in order to properly route the design, I created a phase-aligned clock-distribution unit based on the Mixed-Mode Clock Manager (MMCM) core for simultaneously clocking both of the GTX transceivers on the FPGA fabric and the optical transceiver on the XFP evaluation board. Figure 1 shows the system diagram.

For this design project, I used Xilinx's older ISE® Design Suite tools and did the work in three steps.

Step one involved creating an IBERT core with the CORE Generator™ tool.

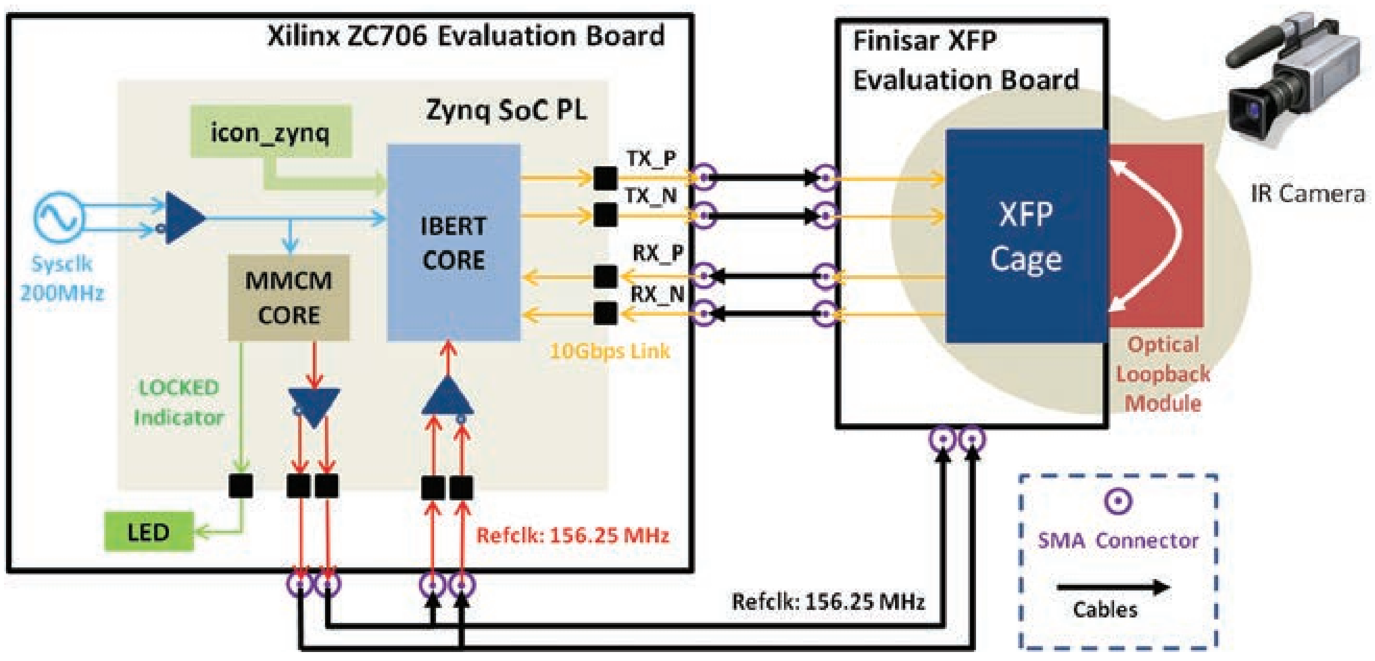
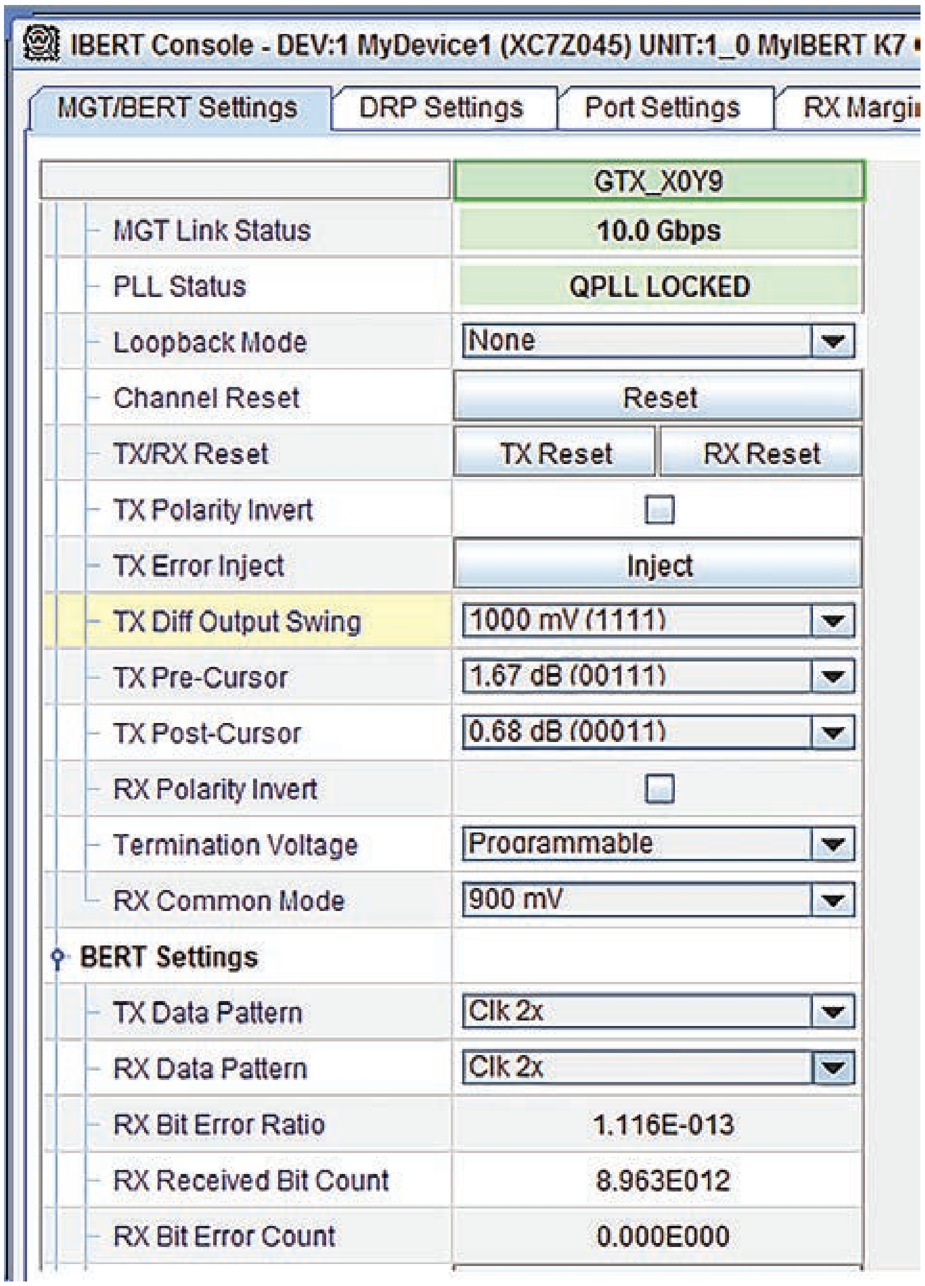


Figure 1 – Block diagram of the proposed system with a connection example


IBERT Console - DEV:1 MyDevice1 (XC7Z045) UNIT:1_0 MyIBERT K7

MGT/BERT Settings	GTX_X0Y9
MGT Link Status	10.0 Gbps
PLL Status	QPLL LOCKED
Loopback Mode	None
Channel Reset	Reset
TX/RX Reset	TX Reset / RX Reset
TX Polarity Invert	☐
TX Error Inject	Inject
TX Diff Output Swing	1000 mV (1111)
TX Pre-Cursor	1.67 dB (00111)
TX Post-Cursor	0.68 dB (00011)
RX Polarity Invert	☐
Termination Voltage	Programmable
RX Common Mode	900 mV
BERT Settings	
TX Data Pattern	Clk 2x
RX Data Pattern	Clk 2x
RX Bit Error Ratio	1.116E-013
RX Received Bit Count	8.963E012
RX Bit Error Count	0.000E000

Figure 2 – Snapshot of the ChipScope Pro screen

Here are some of the key settings for this IBERT 7 series GTX (ChipScope™ Pro) IBERT core. In my design, the IBERT system clocking comes from an external clock source on the board—a 200-MHz differential clock with `P pin location = H9` and `N pin location = G9`. The GTX clocking mode is independent for `QUAD 111`, and I set the line rate to `Max Rate = 10Gbps`. I set the reference clock for the GTX to `Refclk = 156.25 MHz` and the `Refclk source = MGTREFCLK1 111`.

In step two, I created an MMCM core with the CORE Generator. It was imperative to get the tool's Clocking Wizard settings correct. To do this, I set the clock features as `frequency synthesis` and `phase alignment`. The input clock has to be the same system clock on the board (200 MHz). And I set the targeting derivative clock to 156.25 MHz with 50 percent duty cycle. I used two extra signals, `RESET` and `LOCKED`, for controlling and indicating the MMCM core.

The third step was to assemble everything with Xilinx's tools. For this project, I used the ISE Design Suite 14.4. At some point later on, I am planning to switch to the Vivado® Design Suite in order to maximize the performance of the chip.

I first created a new project in ISE, then moved the IBERT core folders (`example_ibert_gtx.vhd`, `ibert_gtx_top.ucf`, `ibert_core.ngc` and `icon_zynq.ngc`) to the ISE project. Next, I added `mmcm_core.vhd` from the MMCM core folder (step 2) to the ISE project. I then used `example_ibert_gtx.vhd` as the top module, instantiated the `mmcm_core` and added three new signals (`CLK_OUTPUT_P`, `CLK_OUTPUT_N` and `LED_REFCLK`) to the design and made corresponding pin assignments in the `ibert_gtx_top.ucf`.

SYSTEM TEST

After generating the .bit file, the FPGA design was ready for stimulating the XFP optical transceiver with a 10-Gbps link. I connected the two boards (as shown in Figure 1), then opened a ChipScope Pro analyzer and configured the device with the newly built .bit file. Next, I double-clicked the IBERT console, causing a new graphical user interface to pop up (as shown in Figure 2). With this screen, we can thoroughly evaluate the thermal performance of the optical transceiver by tuning the predefined data patterns, such as Clk 2x (1010….), and pseudorandom binary sequences (PRBS).

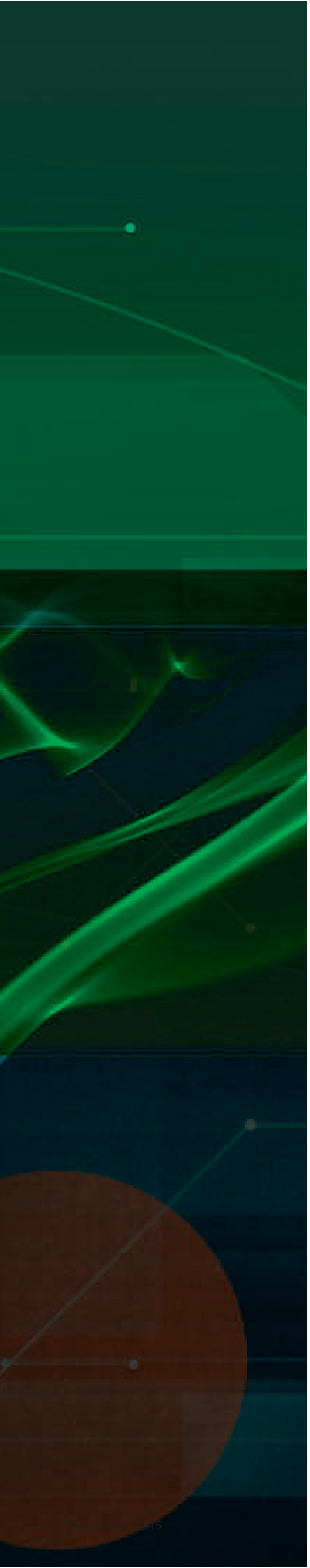
By using Xilinx cores, together with the ZC706 evaluation board, it's easy to build a test platform for evaluating high-speed optical transceivers. In this design, we illustrated the evaluation of a single XFP module. However, you can straightforwardly apply the design methodology to quickly build a logic core for testing multiple optical transceiver modules.

For more information, please contact the author at *lei.guan@alcatel-lucent.com.*

A Double-Barreled Way to Get the Most from Your Zynq SoC

Using both of the ARM A9 cores on Xilinx's Zynq SoC can significantly increase the performance of your system.

by Adam P. Taylor
Chief Engineer, Electrical Systems
e2v
aptaylor@theiet.org



One of the many benefits of Xilinx®'s Zynq®-7000 All Programmable SoC is that it is has two ARM® Cortex™-A9 processors onboard. However, many bare-metal applications and simpler operating systems use only one of the two ARM cores in the Zynq SoC's processing system (PS), a design choice that can potentially limit system performance.

Depending upon the application in development, there could, however, be a need to have both processors running bare-metal applications, or to run different operating systems on each of the processors. For instance, one side could be performing critical calculations and hence running a bare-metal/RTOS application while the second processor is providing HMI and communications using Linux.

WHAT IS MULTIPROCESSING?

Either of those scenarios is an example of multiprocessing. Briefly defined, multiprocessing is the use of more than one processor within a system. A multiprocessing architecture can allow the execution of multiple instructions at a time, though it does not necessarily have to.

There are two kinds of multicore processing: symmetric and asymmetric.

Symmetric multiprocessing makes it possible to run a number of software tasks concurrently by distributing the load across a number of cores. Asymmetric multiprocessing (AMP) uses specialized processors or applications execution on identical processors for specific applications or tasks.

Using both of the cores on the Zynq SoC with bare metal or different operating systems is, by definition, an example of asymmetric multiprocessing. AMP on the Zynq SoC can involve any of the following combinations:

- Different operating systems on Core 0 and Core 1
- Operating system on Core 0, bare metal on Core 1 (or vice versa)
- Bare metal on both cores executing different programs

When you decide upon the need to create an AMP system on the Zynq SoC, you must consider the fact that the ARM processor cores contain a mixture of both private and shared resources that must be correctly addressed. Both processors have private L1 instruction and data caches, timers, watchdogs and interrupt controllers (with both shared and private interrupts). A number of shared resources also exist, of which common examples include I/O peripherals, on-chip memory, the interrupt controller distributor, L2 cache and system memory located within the DDR memory (see Figure 1). These private and shared resources require careful management.

Each PS core has its own interrupt controller and is capable of interrupting itself, with one or both cores using software interrupts. These interrupts are distributed by means of ARM's Distributed Interrupt Controller technology.

As the program being executed for each core will be located within the DDR memory, you must take great care to ensure that you have correctly segmented these applications.

GETTING AMPED UP

The key aspect required to get AMP up and running on the Zynq SoC is a boot loader that will look for a second executable file after loading the first application into memory. Xilinx helpfully provides an application note and source code in XAPP1079. This document comes with a modified first-stage boot loader (FSBL) and modified standalone OS, which you can use to create an AMP system.

The first thing to do is to download the ZIP file that comes with this application note before extracting the two elements—FSBL and OS—to your desired working directory. Then, you must rename the folder called SRC "design." Now, it's important to make sure the software development kit (SDK) is aware of the existence of these new files containing both a modified FSBL and a

Using software interrupts is not too different from using hardware interrupts except, of course, in how you trigger them.

modified standalone OS. Therefore, the next step is to update your SDK repository such that it is aware of their existence.

This is straightforward to achieve. Within SDK under the Xilinx tools menu, select “repositories” and then “new,” navigating to the directory location <your working directory>\app1079\design\work\sdk_repo as shown in Figure 2.

COMMUNICATING BETWEEN PROCESSORS

Before creating the applications for your AMP design, you will need to consider how the applications will communicate (if they need to). The simplest method is to use the on-chip memory. The Zynq SoC has 256 kbytes of on-chip SRAM that can be accessed from one of four sources:

- From either core via the snoop control unit (SCU)
- From the programmable logic using the AXI Accelerator Coherency Port (ACP) via the SCU
- From the programmable logic using the High-Performance AXI port via the on-chip memory (OCM) interconnect
- From the central interconnect, again via the OCM

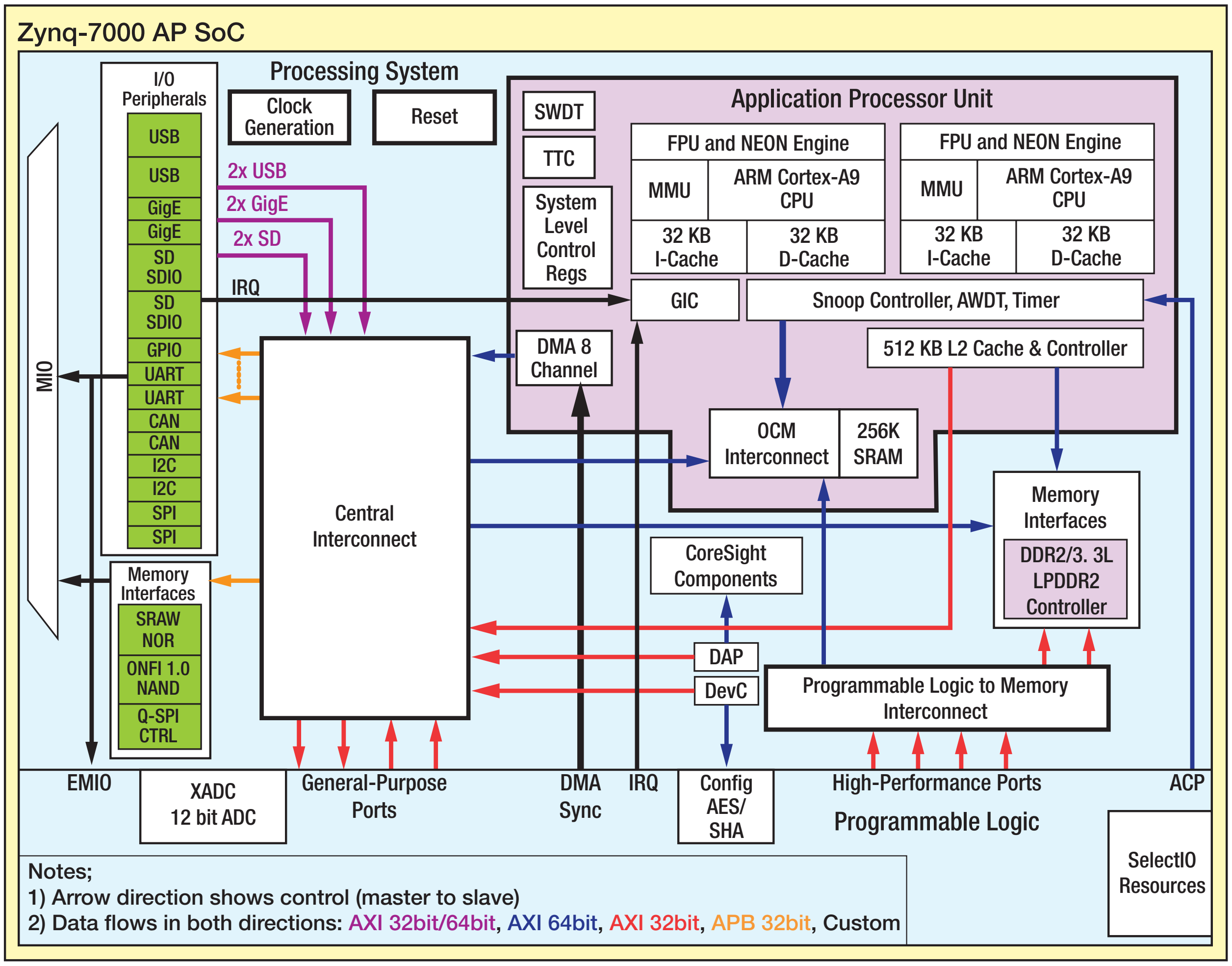
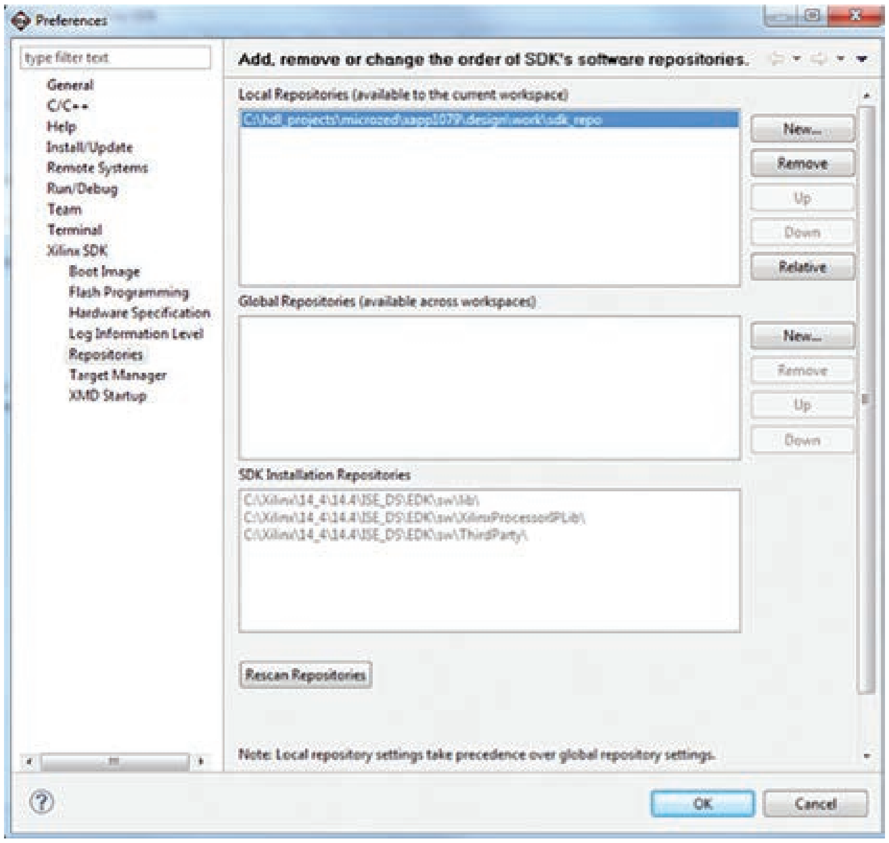


Figure 1 – The Zynq SoC processing system, showing private and shared resources



Add, remove or change the order of SDK's software repositories.

Local Repositories (available to the current workspace)

C:\hdl_projects\microzed\xapp1079\design\work\sdk_repo

Global Repositories (available across workspaces)

SDK Installation Repositories

C:\Xilinx\14_4\14.4\ISE_DS\EDK\sw\lib\
C:\Xilinx\14_4\14.4\ISE_DS\EDK\sw\XilinxProcessorIPLib\
C:\Xilinx\14_4\14.4\ISE_DS\EDK\sw\ThirdParty\

Note: Local repository settings take precedence over global repository settings.

Figure 2 — Adding your new files to the repository

With these different sources that can read and write the on-chip memory, it is especially important to understand the operation of the OCM in detail before using it.

Since there are multiple sources accessing the OCM, it is only sensible that you define a form of arbitration and priority. As the lowest latency is required by the snoop control unit, which is either a processor core or an AXI ACP interface, an SCU read from these sources has the highest priority followed by the SCU write and then the OCM interconnect read and write. The user can invert the priority between the SCU write and the OCM interconnect access by setting the SCU write priority low in the on-chip memory control register.

The OCM itself is organized as 128-bit words, split into four 64-kbyte regions at different locations within the PS address space. The initial configuration has the first three 64-kbyte blocks arranged at the start of the address space and the last 64-kbyte block toward the end of the address space (Figure 5).

SIMPLE ON-CHIP MEMORY EXAMPLE

You can access the OCM using Xilinx I/O functions to read and write to and from the selected memory address. These functions, which are contained within Xil_IO.h, allow for storing and accessing 8-, 16- or 32-bit char, short or int within the CPU address space. Using these functions just requires the address you wish to access and the value you wish to store there. If it is a write, for example,

```
Xil_Out8(0xFFFF0000,0x55);
read_char = Xil_In8(0xFFFF0000);
```

A better way to use this technique to ensure the addresses are both targeting the same location within the on-chip memory, especially if different people are working on the different core programs, is to have a common header file. This file will contain macro definitions of the address of interest for that particular transfer, for instance:

```
#define LED_PAT 0xFFFF0000
```

An alternative approach is for both programs to access the memory location using a pointer. You can do this by defining the pointer, which points to a constant address, normally in C, using a macro:

```
#define LED_OP (*(volatile
  unsigned int *)(0xFFFF0000))
```

Again, you could also use another macro definition for the address to ensure that the address is common to both application programs. This approach does not then require the use of the Xilinx I/O libraries and instead allows simple access via the pointer.

INTERPROCESSOR INTERRUPTS

The Zynq SoC has 16 software-generated interrupts for each core. As noted above, each can interrupt itself, the other core or both cores. Using software interrupts is not too different from using hardware interrupts except, of course, in how you trigger them. The use of software interrupts frees the receiving application from having to poll an expected memory location for an updated value.

Within both cores, you need to configure the Generic Interrupt Controller just as you would for any hardware interrupt. See Xcell Journal issue 87, "How to Use Interrupts on the Zynq SoC," for further information.

You can then trigger a software interrupt in the updating core using the XScuGic_SoftwareIntr function provided within xscugic.h. This command will issue a software interrupt to the identified core, which can then take the appropriate action:

```
XScuGic_SoftwareIntr(<GIC
Instance Ptr>, <SW Interrupt
ID>, <CPU Mask>)
```

You must correctly segment the DDR memory for Core 0 and Core 1 applications or run the risk of one corrupting the other.

CREATING THE APPLICATIONS

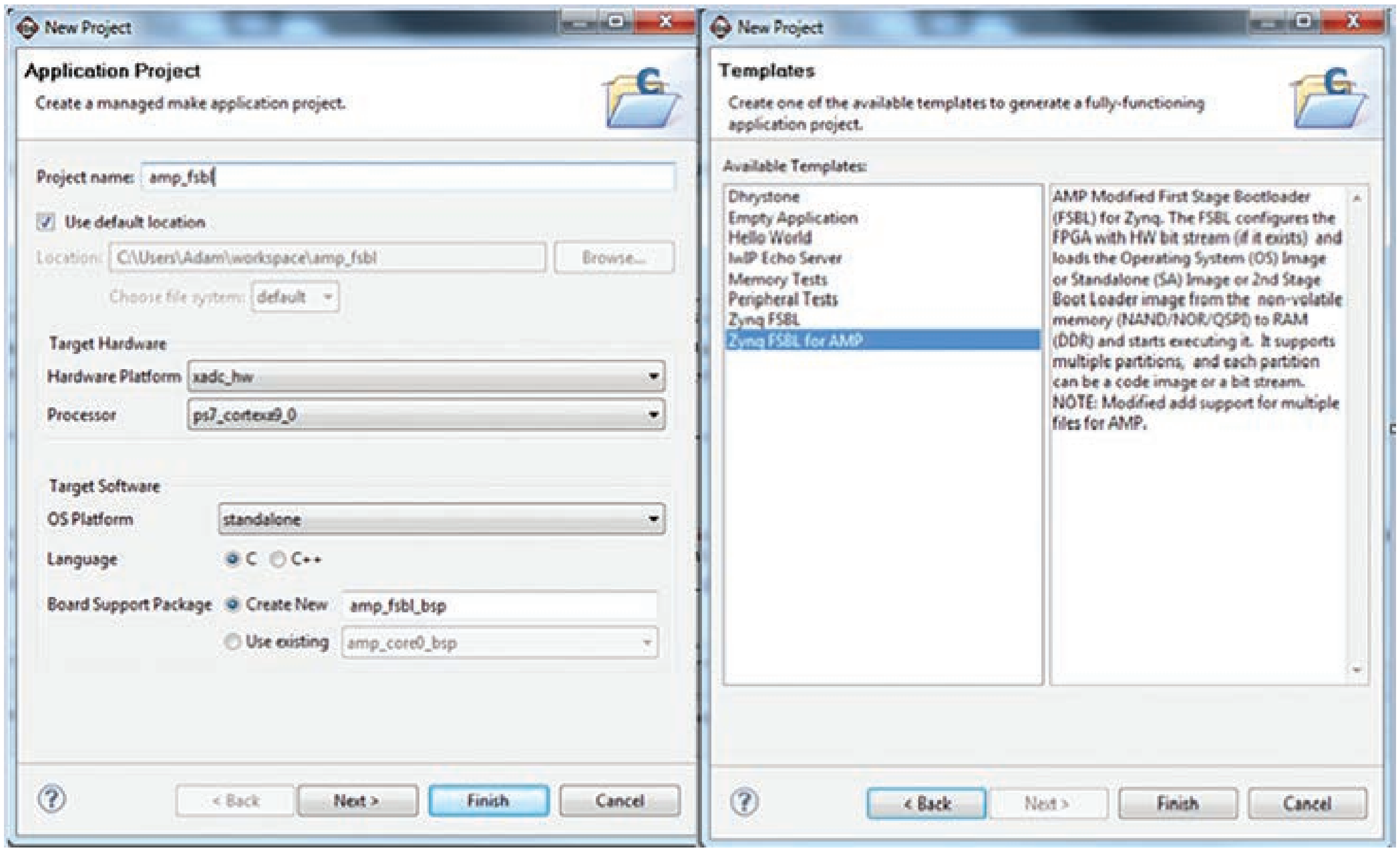
Having added in the repositories, the next stage is to generate three crucial pieces of the AMP solution: the AMP first-stage boot loader, the Core 0 application and the Core 1 application. For each of these items, you will have to generate a different board support package.

The first thing you need to do is to create a new FSBL with the SDK. Selecting "file new application project" enables you to create a FSBL project that supports AMP. This is no different than the process you would follow in creating a normal FSBL. However, this time you will be selecting the "Zynq FSBL for AMP" template as shown in Figure 3.

Following the creation of the AMP FSBL, you will next create the application for the first core. Be sure to select Core 0 and your preferred operating system, and allow it to create its own BSP, as shown in Figure 4.

Having created the application, you need to correctly define the location, with DDR memory, from which it will execute. To do this, edit the linker script as in Figure 5 to show the DDR base address and size. This is important, because if you do not correctly segment the DDR memory for Core 0 and Core 1 applications, you run the risk of one inadvertently corrupting the other.

Having done this segmentation, you can now write the application you wish to execute on Core 0, as this is the core that is in charge within the AMP system. Core 0 must start the execution of the Core 1 application. You need to include the section of code seen in Figure 6 within the application. This code dis-


New Project
Application Project
Create a managed make application project.
Browse...
Choose file system: default
< Back
Next >
Finish
Cancel
New Project
Templates
Create one of the available templates to generate a fully-functioning application project.
< Back
Next >
Finish
Cancel

Project name: amp_fsbl

Use default location [✓]

Location: C:\Users\Adam\workspace\amp_fsbl

Target Hardware

Hardware Platform: xadc_hw

Processor: ps7_cortexa9_0

Target Software

OS Platform: standalone

Language: [✓ C] C++

Board Support Package: [✓ Create New] amp_fsbl_bsp

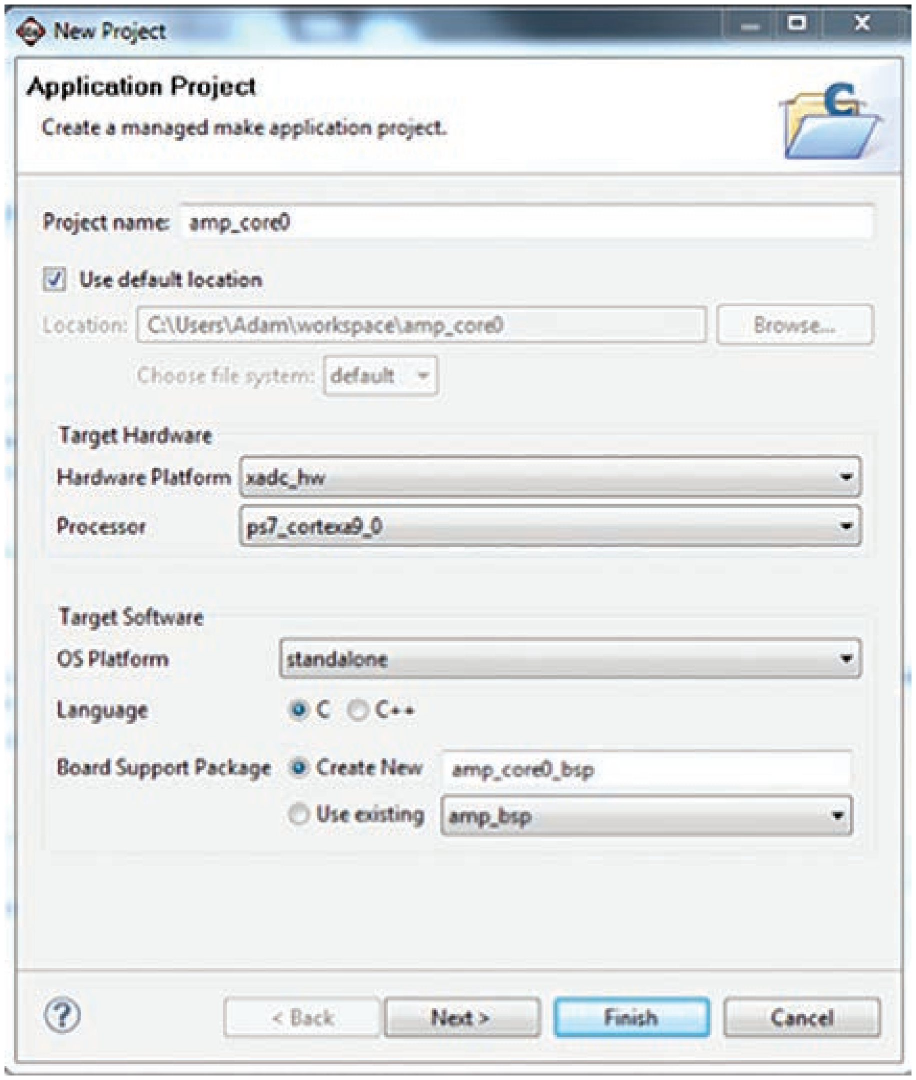
Use existing: amp_core0_bsp

Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- Peripheral Tests
- Zynq FSBL
- [✓ Zynq FSBL for AMP]

AMP Modified First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream. NOTE: Modified add support for multiple files for AMP.

Figure 3 – Selecting the first-stage boot loader for the AMP design



Project name: amp_core0

[✓] Use default location

Location: C:\Users\Adam\workspace\amp_core0

Choose file system: default

Target Hardware

Hardware Platform: xadc_hw

Processor: ps7_cortexa9_0

Target Software

OS Platform: standalone

Language: [✓ C] C++

Board Support Package: [✓ Create New] amp_core0_bsp; Use existing amp_bsp

Figure 4 – Creating the application and BSP for Core 0

ables the cache on the on-chip memory and writes the start address of the Core 1 program to an address that Core 1 will access. Once Core 0 executes the Set Event (SEV) command, Core 1 will start executing its program.

The next step is to create a BSP for Core 1. It's important to use the modified standalone OS (standalone_amp, as shown in Figure 7), which prevents reinitialization of the PS snoop control unit. As such, do not allow automatic generation of the BSP while you create the project, as you did for Core 0. Be sure to select Core 1 in the CPU selection options.

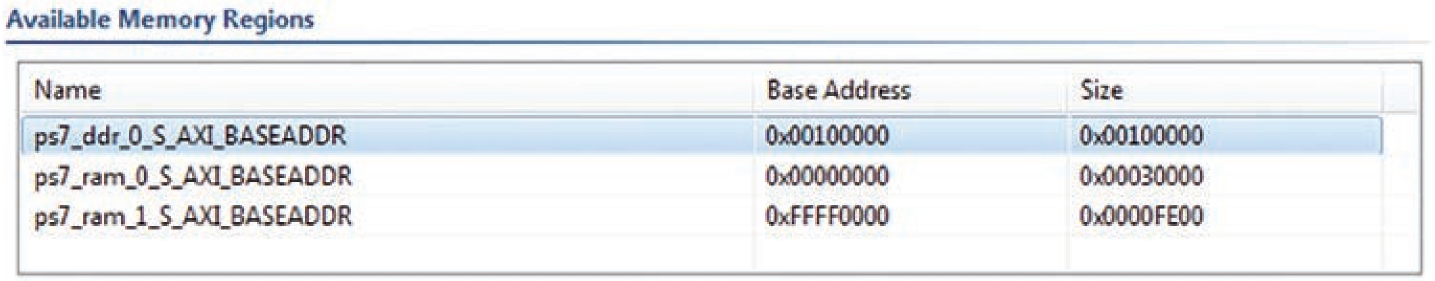
Now that you have created the BSP for Core 1, you need to modify the settings of the BSP before you can progress to creating the application you want to run upon Core 1. Doing so is very simple and requires the addition of an extra compiler flag of –DUSE_AMP=1 to the configuration for the drivers section of the BSP.

With this step completed, you are free to create the application for Core 1. Be sure to select Core 1 as the processor and use the BSP you just created. Again, having created the new application, you need to once more define the correct memory locations within the DDR memory from which the Core 1 program will execute. This is achieved by editing the linker script for the application for Core 1 as you did previously. As with the first core, within this application you must likewise disable the cache on the on-chip memory, which you can use to communicate between the two processors.

PUTTING IT ALL TOGETHER

Once you have completed creation of your applications and built the projects, you should now be in possession of the following components:

- AMP FSBL ELF
- Core 0 ELF
- CORE 1 ELF
- BIT file defining the configuration of the Zynq device upon which you wish to implement AMP


Available Memory Regions

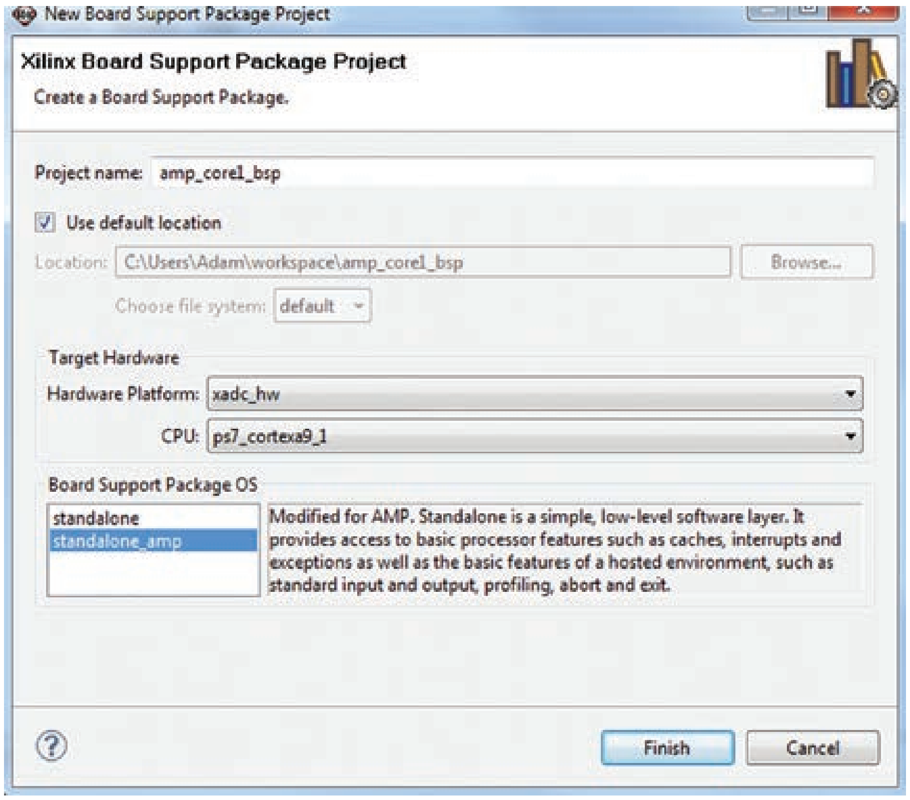
Name	Base Address	Size
ps7_ddr_0_S_AXI_BASEADDR	0x00100000	0x00100000
ps7_ram_0_S_AXI_BASEADDR	0x00000000	0x00030000
ps7_ram_1_S_AXI_BASEADDR	0xFFFF0000	0x0000FE00

Figure 5 – Core 0 DDR location and size

Creating an asymmetric multiprocessing application on the Zynq SoC can be a very simple matter using the tools available.

```
#include <stdio.h>
#include “xil_io.h”
#include ”xil_mmu.h”
#include “xil_exception.h”
#include “xpseudo_asm.h>”
#include “xscugic.h>

#define sev() __asm__(”sev”)
#define CPU1STARTADR 0xfffffff0
#define COMM_VAL  (*(volatile unsigned long *)(0xFFFF0000))


int main()
{

     //Disable cache on OCM
     Xil_SetTlbAttributes(0FFFF0000,0x14de2);      // s=b1 TEX=b100 AP=b11, Domain=b1111, C=b0, B=b0
     Xil_Out32(CPU1STARTADR, 0x00200000);
     dmb(); //waits until write has finished
     sev();
```

Figure 6 – Coding to disable cache on the on-chip memory


New Board Support Package Project

Xilinx Board Support Package Project

Create a Board Support Package.

Project name: amp_core1_bsp

[✓] Use default location

Location: C:\Users\Adam\workspace\amp_core1_bsp

Choose file system: default

Target Hardware

Hardware Platform: xadc_hw

CPU: ps7_cortexa9_1

Board Support Package OS

standalone

standalone_amp

Modified for AMP. Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Figure 7 – Creating the BSP for Core 1

To enable the Zynq SoC to boot from your selected configuration memory, you will need a .bin file. To create it, you will also need a BIF file, which defines the files to be used to create this BIN file and the order in which they go. Rather than use the “create Zynq” boot image within the SDK, you will be using an ISE® Design Suite command prompt and BAT file provided as part of XAPP1079 under the downloaded directory\design\work\bootgen. This directory contains a BIF file and a cpu1_bootvec.bin, which is used as part of the modified FSBL to stop it looking for more applications to load.

To generate the BIN file, you need to copy the three generated ELF files to the bootgen directory and edit the BIF file to ensure the ELF names within it are correct, as shown in Figure 8.

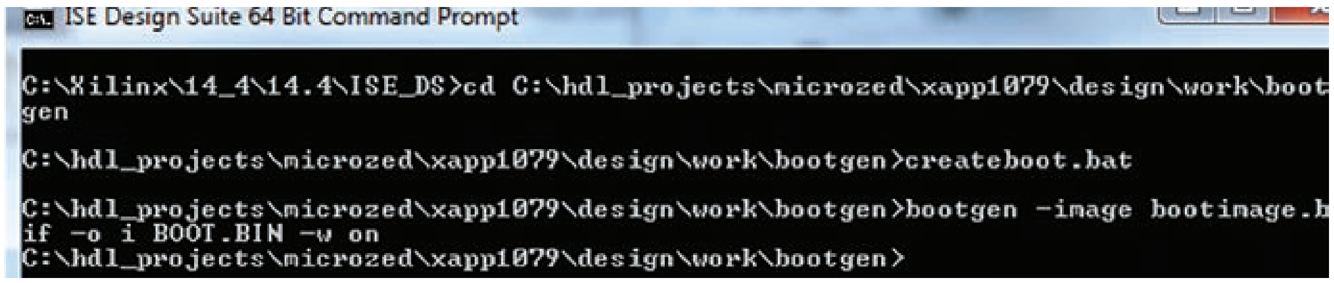
Now you can open an ISE command prompt and navigate to the bootgen directory. There, you should run the createboot.bat. This step will create the boot.bin file as shown in Figure 9.

You can then download this file into the nonvolatile memory on your Zynq SoC. Booting the device will result in both cores starting and executing their respective programs.

Creating an asymmetric multiprocessing application on the Zynq SoC can be a very simple matter using the tools available. It's easy to achieve communication between the two cores using the on-chip memory or even a segmented DDR area.

```
the_ROM_image
{

     [bootloader] amp_fsbl.elf
                download.bit
                  amp_cpu0.elf
                  app_cpu1.elf

  //write start vector address 0xFFFFFFF0 with 0xFFFFFFF00
  //This load address triggers fsbl to continue
  [load = 0xFFFFFFF0] cpu1_bootvec.bin
}
```

Figure 8 – Modifying the BIF file



```
C:\Xilinx\14_4\14.4\ISE_DS>cd C:\hdl_projects\microzed\xapp1079\design\work\boot
gen

C:\hdl_projects\microzed\xapp1079\design\work\bootgen>createboot.bat

C:\hdl_projects\microzed\xapp1079\design\work\bootgen>bootgen -image bootimage.b
if -o i BOOT.BIN -w on
C:\hdl_projects\microzed\xapp1079\design\work\bootgen>
```

Figure 9 –The creation of the boot.bin file that will run on the Zynq SoC

TRACE32®

Debugging Xilinx's Zynq™-7000 family with ARM® CoreSight™

- RTOS support, including Linux kernel and process debugging
- SMP/AMP multicore Cortex®- A9 MPCore™s debugging
- Up to 4 GByte realtime trace including PTM/ITM
- Profiling, performance and statistical analysis of Zynq™'s multicore Cortex®-A9 MPCore™

LAUTERBACH
DEVELOPMENT TOOLS

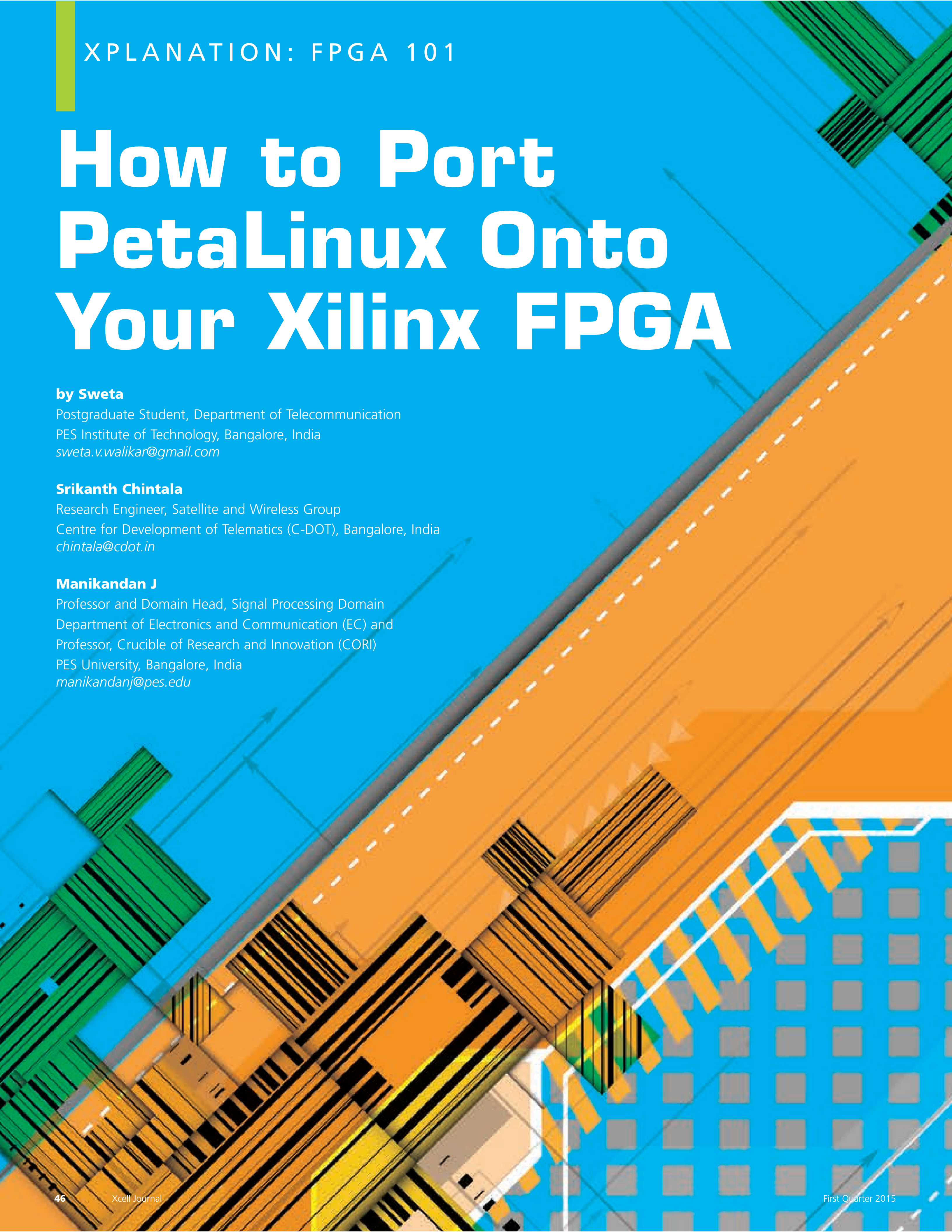
www.lauterbach.com

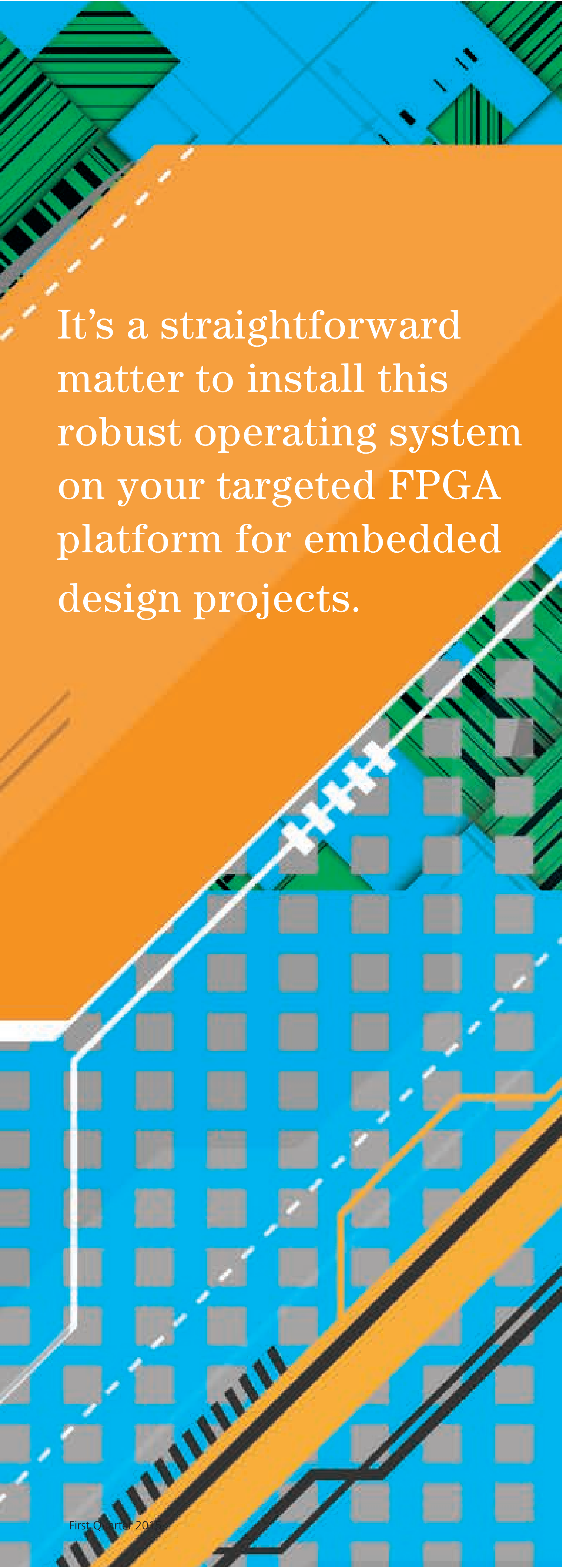
How to Port PetaLinux Onto Your Xilinx FPGA

by Sweta
Postgraduate Student, Department of Telecommunication
PES Institute of Technology, Bangalore, India
sweta.v.walikar@gmail.com

Srikanth Chintala
Research Engineer, Satellite and Wireless Group
Centre for Development of Telematics (C-DOT), Bangalore, India
chintala@cdot.in

Manikandan J
Professor and Domain Head, Signal Processing Domain
Department of Electronics and Communication (EC) and
Professor, Crucible of Research and Innovation (CORI)
PES University, Bangalore, India
manikandanj@pes.edu





It's a straightforward matter to install this robust operating system on your targeted FPGA platform for embedded design projects.

FPGAs have come a long way from their humble beginnings as glue logic. The logic capacity and flexibility of today's FPGAs have catapulted them into a central position in embedded designs. Today, a complete system fits on a single programmable chip, an architecture that facilitates hardware/software co-design and integrates hardware with software applications.

These kinds of FPGA-based embedded designs need a robust operating system. PetaLinux has emerged as a favorite among embedded designers. It is available free of cost as open source and also supports various processor architectures, such as the Xilinx® MicroBlaze® CPU as well as ARM® processors. In order to port PetaLinux onto a particular FPGA, the kernel source code, boot loader, device tree and root file system must be customized, configured and built for the targeted platform.

For a design project here at PES University and C-DOT, our team set out to port PetaLinux and run several PetaLinux user applications on Xilinx's KC705 evaluation board, which features a Kintex®-7 XC7K325T FPGA. It turned out to be a fairly straightforward process.

WHY CHOOSE PETALINUX?

Before going into the details of how we did it, it's worth taking a moment to consider the various OS options available for FPGA-based embedded systems. PetaLinux is one of the most commonly used OSes on FPGAs, along with µClinux and Xilkernel. µClinux is a Linux distribution or ported Linux OS that includes a small Linux kernel and is designed for a processor that does not have a memory-management unit (MMU) [1]. µClinux comes with libraries, applications and tool chains. Xilkernel, for its part, is a small, robust and modular kernel that allows a higher degree of customization than µClinux, enabling users to tailor the kernel to optimize their design in terms of size and functionality [2].

PetaLinux, meanwhile, is a complete Linux distribution and development environment targeting FPGA-based system-on-chip (SoC) designs. PetaLinux consists of preconfigured binary bootable images; fully customizable Linux for Xilinx devices; and an accompanying PetaLinux software development kit (SDK) [3] that includes tools and utilities to automate complex tasks across configuration, build and deployment. The PetaLinux development package, available free of cost and downloadable from Xilinx, includes hardware reference projects designed for various Xilinx FPGA development kits. Also included are a kernel configuration utility for Xilinx FPGAs, software tools such as a cross-compiler, a hardware design creation tool and many more design aids.

It has been reported that Xilkernel performs better than µClinux [4] and that PetaLinux outperforms Xilkernel [5]. For that reason, we chose PetaLinux for our project, especially since the packages were readily available for our Xilinx target board. Another advantage of porting PetaLinux is that the user can have the facility of remote programming. That means you can load the FPGA target board with a new configuration file (or bitstream file) through Telnet using remote access.

There are two approaches to creating a software platform for building a PetaLinux system: PetaLinux commands on a Linux terminal or a GUI with a pulldown menu.

BEGINNING THE INSTALLATION

Let's take a detailed look at how our team installed PetaLinux. For the first step, we downloaded the PetaLinux package 12.12 and the board support package (BSP) for the Kintex-7 target board. We ran the PetaLinux SDK installer and installed the same into the */opt/Petalinux-v12.12-final* directory using the following commands in the console:

```
@ cd /opt
@ cd /opt/PetaLinux-v12.12-final-full.tar.gz
@ tar zxf  PetaLinux-v12.12-final-full.tar.gz
```

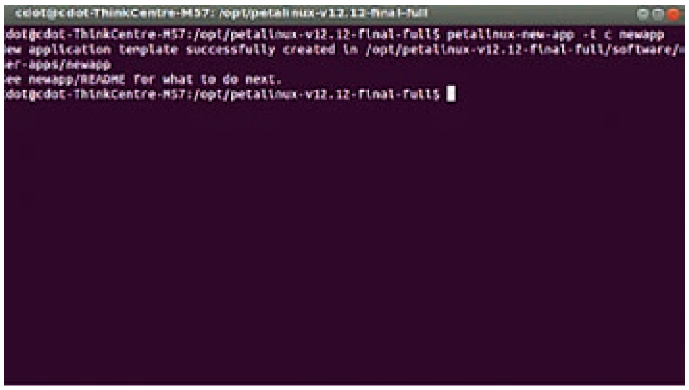
We then copied and pasted the PetaLinux SDK license obtained from the Xilinx website into the .xilinx and .Petalogix folders. Next, we set the SDK working environment by sourcing the appropriate settings using the following commands:

```
@ cd /opt/PetaLinux-v12.12-final
@ source settings.sh
```

In order to verify whether the working environment was set or not, we used the following command:

```
@ echo $PETALINUX
```

If the environment is set properly, the path where PetaLinux is installed will be displayed. In our case, the path where PetaLinux was installed was */opt/PetaLinux-v12.12-final*.



```
cdot@cdot-ThinkCentre-M57: /opt/petalinux-v12.12-final-full
dot@cdot-ThinkCentre-M57:/opt/petalinux-v12.12-final-full$ petalinux-new-app -t c newapp
ew application template successfully created in /opt/petalinux-v12.12-final-full/software/u
er-apps/newapp
ee newapp/README for what to do next.
dot@cdot-ThinkCentre-M57:/opt/petalinux-v12.12-final-full$
```

Figure 1 – Snapshot of a Linux terminal window for user settings

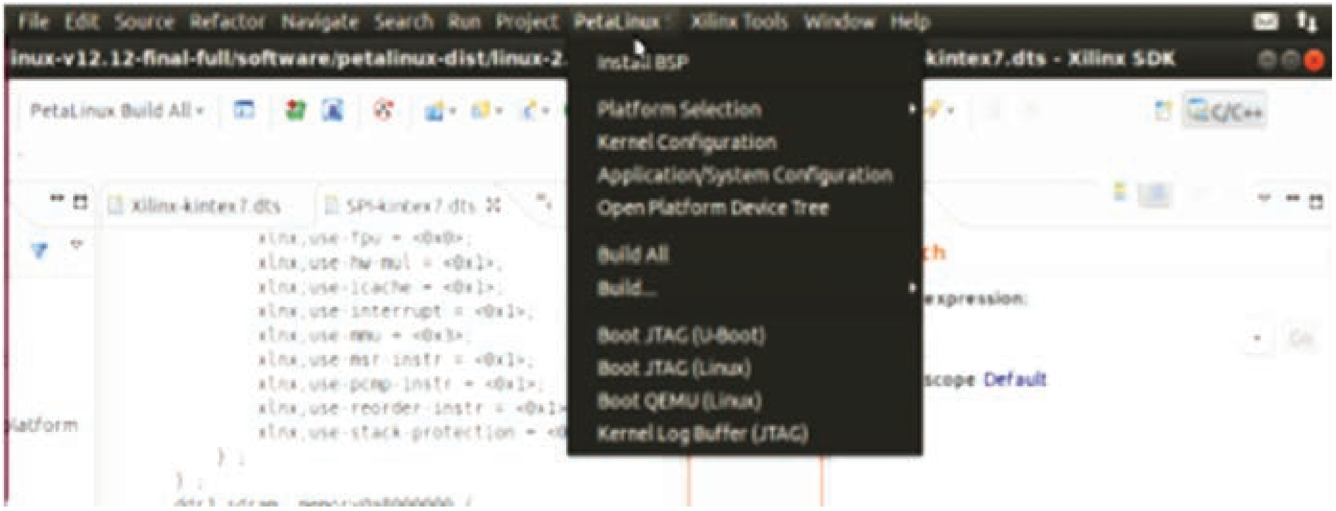
Our next task was to install the BSP, which includes the necessary design files, configuration files and prebuilt hardware and software packages that are already tested and readily available for downloading onto the target board. Packages are also available for booting in the Quick Emulator (QEMU) system simulation environment. In order to install the BSP, we created a folder named "bsp" in the path /opt and copied the ZIP file of the KC705 BSP using the following commands:

```
@ cd  /opt/PetaLinux-v12.12-final-full
@ source settings.sh
@ source /opt/Xilinx/14.4/ISE_DS/settings32.sh
@ PetaLinux-install-bsp /bsp/Xilinx-KC705
    -v12.12-final.bsp
```

There are two approaches to creating and configuring a software platform for building a PetaLinux system customized to a new hardware platform. One method is to use PetaLinux commands in their corresponding path locations using a Linux terminal, as shown in Figure 1. The second approach is to use a GUI with a pulldown menu, as shown in Figure 2. You can use either of these approaches to select the platform, configure the Linux kernel, configure the user application and build images. The PetaLinux console is available once the OS is installed, whereas the GUI is available after installing the PetaLinux SDK plug-in. Once you've installed the plug-in, you can set the configurations using the PetaLinux GUI found in the PetaLinux Eclipse SDK (Figure 2). The GUI has features such as user application and library development as well as debugging, building and configuring PetaLinux and the hardware platform.

BUILDING THE HARDWARE

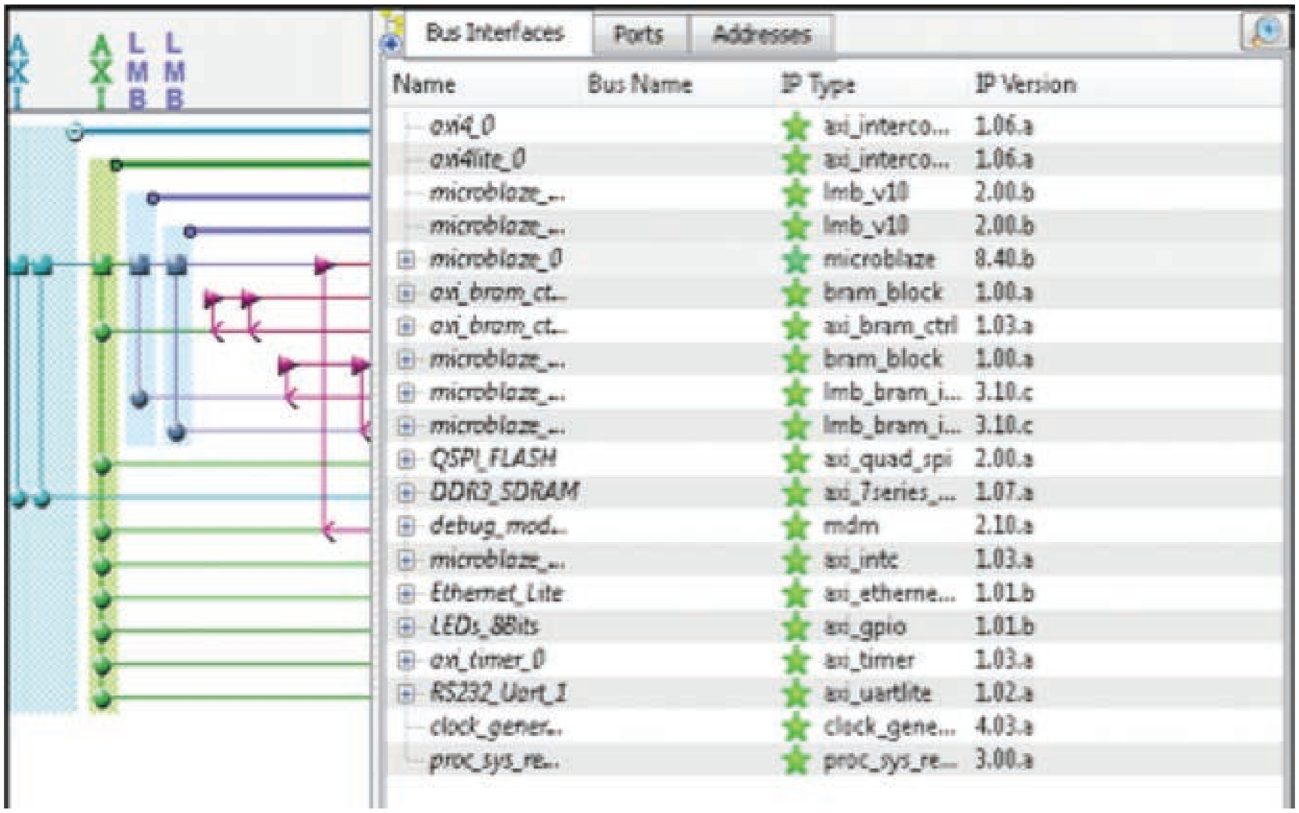
We used the Kintex-7 FPGA-based KC705 evaluation board for our project. The hardware interfaces required for the design included an RS232 interface to monitor the output, a JTAG interface to program the FPGA and an Ethernet interface for remote programming. Besides the PetaLinux SDK, other software required for the proposed design included Xilinx Platform Studio (XPS) [6,7] and the Xilinx Software Development Kit (SDK) [7].



```
xlnx,use-fpu = <0x0>;
xlnx,use-hw-mul = <0x1>;
xlnx,use-icache = <0x1>;
xlnx,use-interrupt = <0x1>;
xlnx,use-mmu = <0x3>;
xlnx,use-msr-instr = <0x1>;
xlnx,use-pcmp-instr = <0x1>;
xlnx,use-reorder-instr = <0x1>
xlnx,use-stack-protection = <0
} ;
} ;
```

Figure 2 – Snapshot of PetaLinux SDK menu for user settings

For the hardware portion of the embedded design, our first task was to design a MicroBlaze processor-based hardware platform using the Base System Builder (BSB) in XPS. The BSB allows you to select a set of peripherals available on the target board. You can add or remove the peripherals based on the demands of the application. The set of cores or peripherals employed for our proposed application included an external memory controller with 8 Mbytes of memory, a timer enabled with interrupts, an RS232 UART with a baud rate of 115,200 bps, Ethernet, nonvolatile memory and LEDs. Once we made our selections, we obtained the hardware peripherals along with their bus interfaces (Figure 3). For designs based on the MicroBlaze processor, PetaLinux requires an MMU-enabled CPU. Hence, we selected low-end



Name	Bus Name	IP Type	IP Version
axi4_0		axi_interco...	1.06.a
axi4lite_0		axi_interco...	1.06.a
microblaze_...		lmb_v10	2.00.b
microblaze_...		lmb_v10	2.00.b
microblaze_0		microblaze	8.40.b
axi_bram_ct...		bram_block	1.00.a
axi_bram_ct...		axi_bram_ctrl	1.03.a
microblaze_...		bram_block	1.00.a
microblaze_...		lmb_bram_i...	3.10.c
microblaze_...		lmb_bram_i...	3.10.c
QSPI_FLASH		axi_quad_spi	2.00.a
DDR3_SDRAM		axi_7series_...	1.07.a
debug_mod...		mdm	2.10.a
microblaze_...		axi_intc	1.03.a
Ethernet_Lite		axi_etherne...	1.01.b
LEDs_8Bits		axi_gpio	1.01.b
axi_timer_0		axi_timer	1.03.a
RS232_Uart_1		axi_uartlite	1.02.a
clock_gener...		clock_gene...	4.03.a
proc_sys_re...		proc_sys_re...	3.00.a

Figure 3 – Hardware configuration of the FPGA

At this point we had the hardware design completed. We could now use the first-stage boot loader to bring up the kernel.

Linux with an MMU by double-clicking on the microblaze_0 instance in the XPS window.

Next, we converted the hardware configuration into a bitstream using a three-step conversion process. First, we used XPS to generate a netlist that represented the embedded hardware platform. Second, we mapped the design into FPGA logic. Finally, we converted the implemented design into a bitstream that could be downloaded onto the FPGA. The final output of XPS was s*ystem.bit and system_bd.bmm* files.

Once we had generated the bitstream, we exported the hardware platform description to the SDK so as to observe the targeted hardware platform in the SDK. The exported system.xml file consisted of information the SDK required to write application software and debug it on the targeted hardware platform. Our next task was to add a PetaLinux repository in the SDK using *Xilinx Tools → Repository → New* and then select the path where PetaLinux was installed. In our case, the path was *$PetaLinux/Hardware/edk_user_repository*.

Next, we created a PetaLinux BSP using *File → Board support package → PetaLinux*. We configured the PetaLinux BSP by selecting necessary drivers based on the application required. Then we built the BSP and created and configured the first-stage boot loader application (fs-boot) to bring up the kernel. The BSP establishes interaction between the hardware and boot application. The output of the SDK is *fs-boot.elf*. A data-to-memory converter command *data2mem* is available that merges *system.bit*, *system_bd.bmm and fs-boot.elf* into a single bitstream file called *download.bit*, which serves as the final FPGA bitstream.

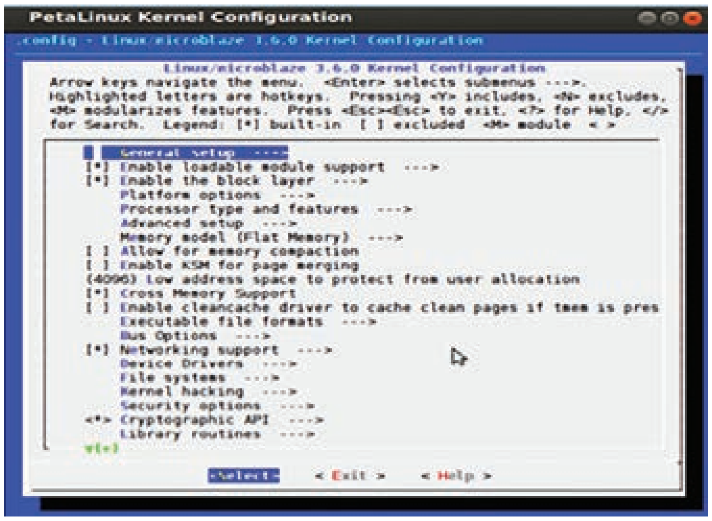
At this point we had the hardware design completed, which among other things included a MicroBlaze core with the PetaLinux OS running on it. We could now use the first-stage boot loader application to bring up the kernel.

BUILDING THE SOFTWARE

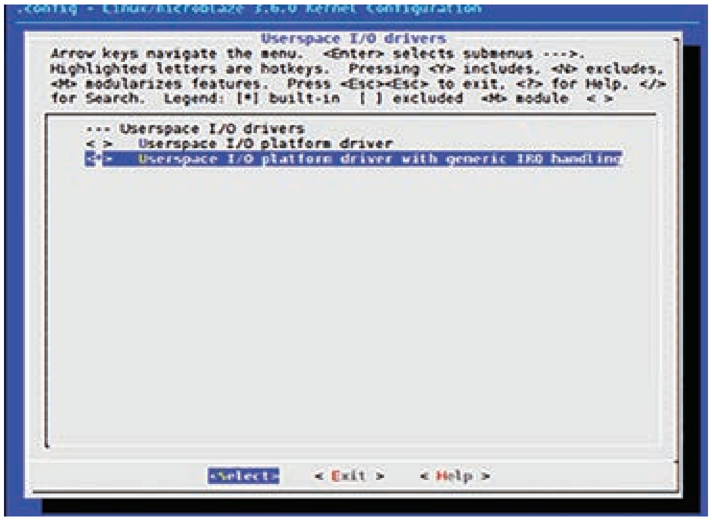
Once our hardware platform was built, we created a customized PetaLinux software platform targeted to the hardware using the following commands:

```
$ cd/opt/PetaLinuxv12.12
$  PetaLinux-new-platform -c <CPU-ARCH> -v
   <VENDOR> -p <PLATFORM>
```

where –c <cpu-arch> is the supported CPU type (here, the MicroBlaze processor), –v <vendor> is the vendor name (here, Xilinx) and –p <platform> is the product name (here, the KC705). The configuration files of the software platform are generated in the directory where PetaLinux is installed, namely */opt/PetaLinuxv12.12/software/ PetaLinux-dist/vendors/Xilinx/ KC705.*



```
PetaLinux Kernel Configuration
.config - Linux/microblaze 3.6.0 Kernel Configuration
                Linux/microblaze 3.6.0 Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

          General setup  --->
      [*] Enable loadable module support  --->
      [*] Enable the block layer  --->
          Platform options  --->
          Processor type and features  --->
          Advanced setup  --->
          Memory model (Flat Memory)  --->
      [ ] Allow for memory compaction
      [ ] Enable KSM for page merging
      (4096) Low address space to protect from user allocation
      [*] Cross Memory Support
      [ ] Enable cleancache driver to cache clean pages if tmem is pres
          Executable file formats  --->
          Bus Options  --->
      [*] Networking support  --->
          Device Drivers  --->
          File systems  --->
          Kernel hacking  --->
          Security options  --->
      <*> Cryptographic API  --->
          Library routines  --->
      v(+)

                <Select>    < Exit >    < Help >
```



```
.config - Linux/microblaze 3.6.0 Kernel Configuration
                      Userspace I/O drivers
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      --- Userspace I/O drivers
      < >   Userspace I/O platform driver
      <*>   Userspace I/O platform driver with generic IRQ handling

                <Select>    < Exit >    < Help >
```

Figure 4 – The kernel configuration menu

To customize the software platform template to match the hardware, we merged the existing platform configuration with the kernel configuration using the command *PetaLinux-copy-autoconfig*. This command generates the hardware configuration files *Xilinx-KC705.dts*, *xparameters.h* and *config.mk*.

We configured the Linux kernel by opening the kernel configuration menu using the GUI (PetaLinux SDK → Kernel Configuration). You can also do it using the following commands in the Linux terminal:

```
$ cd /opt/PetaLinux_v12.12
$  PetaLinux-config-kernel
```

We enabled the drivers for the application in the kernel configuration pop-up window (shown in Figure 4). In order to access devices through the user-space input/output (UIO) interface for the proposed work, we enabled the UIO driver in the kernel configuration menu.

After configuring the kernel, we designed some applications. PetaLinux provides user application templates for C and C++ programming [8]. These templates include application source code and Makefiles, so it was easy to configure and compile applications for the targeted chip and install them into the root file system. You can create a new PetaLinux user application either by using the GUI (File → PetaLinux New Application) or by typing the following commands into the Linux terminal:

```
$ cd /opt/PetaLinux_v12.12
$  PetaLinux-config-apps
```

We then provided a file name to the user application. In our case, we created *gpio-dev-mem-test* and *gpio-uio-test* user applications and modified the template source code based on the application requirements.

Next, we built the PetaLinux system image by using the GUI (as shown in Figure 2). You can also do it by using the *make* command in the Linux terminal, as follows:

```
$ cd $PETALINUX/software/ PetaLinux-dist
$ make
```

Now the software platform with OS and customized user application is ready to be used, along with the hardware design we've already discussed.

TESTING PETALINUX RUNNING ON THE DEVICE

Here's how PetaLinux boots up. The MicroBlaze processor executes the code residing in Block RAM. The first-stage boot loader (*fs-boot*) will initialize basic hardware, execute *fs-boot.elf* and search for the the Universal Bootloader, or U-Boot, address in a flash partition, as the address of U-Boot is specified while configuring *fs-boot*. The *fs-boot* will then fetch the U-boot image from the U-Boot partition in flash, send it to the device's DDR3 memory and run the kernel. Once you have built all the images required for booting, you can test them on hardware via JTAG, Ethernet or the Quick Emulator. QEMU is an emulator and a virtual machine that allows you to run the PetaLinux OS [9]. Let's look at booting methods for all three solutions.

JTAG is the traditional method for programming and testing FPGA designs. To program the FPGA using the JTAG, we used the pulldown menu "Xilinx Tool → Program the FPGA" and downloaded the *download.bit* file that we generated earlier. Then we downloaded the image onto the board using the GUI (PetaLinux SDK → BOOT JTAG [Linux]), as shown in Figure 2. You can also use the following commands in the Linux terminal:

```
$ cd/opt/PetaLinux_v12.12/software/
  PetaLinux-dist
$  PetaLinux-jtag-boot -i images/image.elf
```

Alternatively, you can perform an indirect kernel boot using U-Boot to boot PetaLinux. The system is first bootstrapped by downloading U-Boot via the JTAG interface using either the GUI (PetaLinux SDK → BOOT JTAG [U-Boot]) or the following commands:

```
$ cd $PETALINUX/software/ PetaLinux-dist
$  PetaLinux-jtag-boot -i images/u-boot.elf
```

Figure 6 shows a snapshot of the U-Boot console.

It's worth noting that the FPGA board is connected to the Ethernet interface. You must select the Ethernet interface in the hardware resources part of the XPS. Once U-Boot boots, check whether the IP address of the server and host are the same. If they are not, set the IP of the host using the following commands in the U-Boot terminal:

```
u-boot>print serverip // prints 192.168.25.45(server ip)
u-boot>print ipaddr   // prints IP address
of the board as   // 192.168.25.68
u-boot>set serverip <HOST IP> // Host IP 192.168.25.68
u-boot>set serverip 192.168.25.68
```

Now the server (PC) as well as the host (KC705 board) have the same IP address. Run the netboot command from the server to download the PetaLinux image and boot:

```
u-boot> run netboot
```

After running netboot, you should see the PetaLinux console, as seen in Figure 5.

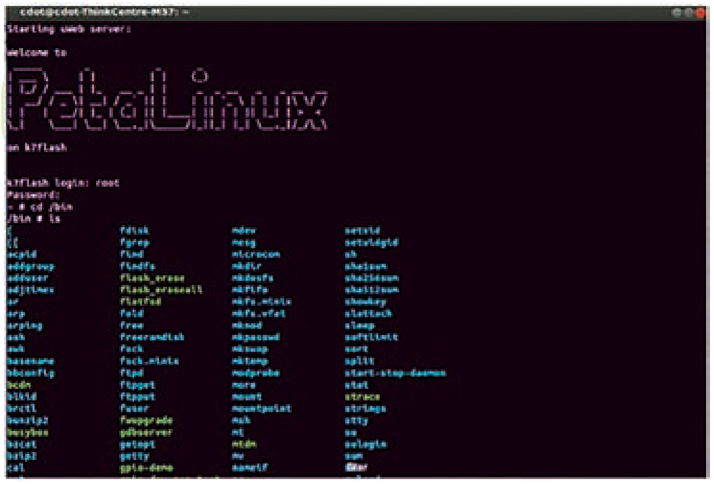
Last but not least, you can perform kernel boot by means of QEMU by using either the GUI (PetaLinux SDK → BOOT QEMU [Linux]) or the following commands:

```
$ cd $ PETALINUX/software/ PetaLinux-dist
$ PetaLinux-qemu-boot -i images/image.elf
```

Using this fast method produces the screen shown in Figure 7.

TESTING APPLICATIONS RUNNING ON THE DESIGN

Once the booting of PetaLinux is tested, the next task is to test the user application designed for PetaLinux. The MicroBlaze processor looks at the hardware peripherals on the Kintex-7 FPGA board as a set of memory registers. Each register has its own base address and end address. In order to access any peripheral, the user must know its base and end addresses. You will find details about the addresses in the device tree source (**.dts*) file. For our design, we developed and tested four applications: Accessing DDR3; Accessing GPIO Using /dev/mem; Accessing GPIO Using UIO; and File Transfer.



```
cdot@cdot-ThinkCentre-M57: ~
Starting uWeb server:

Welcome to

PetaLinux

on k7flash


k7flash login: root
Password:
~ # cd /bin
/bin # ls
[                   fdisk               mdev                setsid
[[                  fgrep               mesg                setuidgid
acpid               find                microcom            sh
addgroup            findfs              mkdir               sha1sum
adduser             flash_erase         mkdosfs             sha256sum
adjtimex            flash_eraseall      mkfifo              sha512sum
ar                  flatfsd             mkfs.minix          showkey
arp                 fold                mkfs.vfat           slattach
arping              free                mknod               sleep
ash                 freeramdisk         mkpasswd            softlimit
awk                 fsck                mkswap              sort
basename            fsck.minix          mktemp              split
bbconfig            ftpd                modprobe            start-stop-daemon
bcdn                ftpget              more                stat
blkid               ftpput              mount               strace
brctl               fuser               mountpoint          strings
bunzip2             fwupgrade           msh                 stty
busybox             gdbserver           mt                  su
bzcat               getopt              mtdm                sulogin
bzip2               getty               mv                  sum
cal                 gpio-demo           nameif              tar
```

Figure 5 – Snapshot of PetaLinux console confirming that the OS has booted

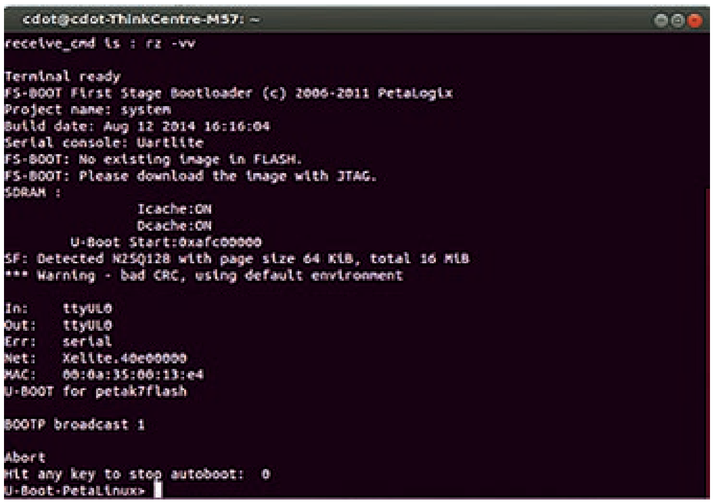
1. Accessing DDR3

We used the PetaLinux application titled *DDR3-test.c* to access the DDR3 memory. The application is designed to write data to and read data from a DDR3 memory location. DDR3 is a dual-in-line memory module that provides SDRAM for storing user code and data. As mentioned earlier, the user should know the start and end addresses of DDR3 memory—*0xC0000000* and *0xC7FFFFFF* respectively. The memory size is 512 Mbytes. The Linux kernel resides in the initial memory locations of DDR3 memory. Hence, the writing location for DDR3 memory is selected in such a way that the Linux kernel is not corrupted. The command we used to write data to DDR3 memory was

```
#DDR3-test -g 0xc7000000 -o 15
```

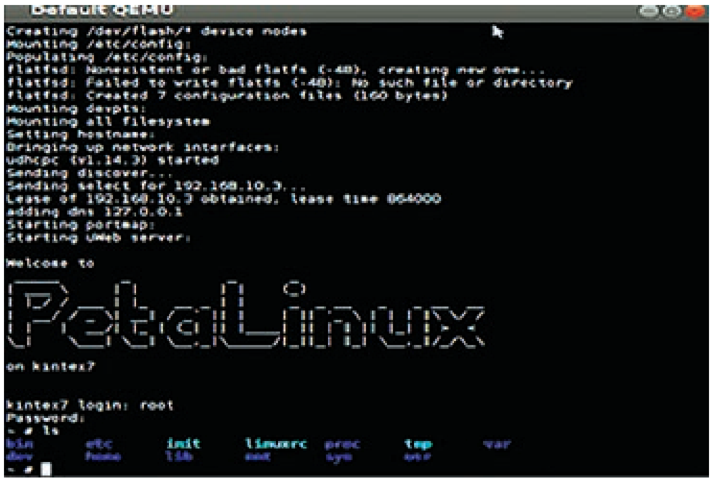
where DDR3-test is the application name, –g is the DDR3 memory physical address, –o is output and 15 is the value expected to be written on the DDR3 memory at the location 0xc7000000. To test whether the value is written at the expected location, we used the following command to read data from DDR3 memory:

```
#DDR3-test -g 0xc7000000 -i
```



```
cdot@cdot-ThinkCentre-M57: ~
receive_cmd is : rz -vv

Terminal ready
FS-BOOT First Stage Bootloader (c) 2006-2011 PetaLogix
Project name: system
Build date: Aug 12 2014 16:16:04
Serial console: Uartlite
FS-BOOT: No existing image in FLASH.
FS-BOOT: Please download the image with JTAG.
SDRAM :
                Icache:ON
                Dcache:ON
        U-Boot Start:0xafc00000
SF: Detected N25Q128 with page size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    ttyUL0
Out:   ttyUL0
Err:   serial
Net:   Xellite.40e00000
MAC:   00:0a:35:00:13:e4
U-BOOT for petak7flash

BOOTP broadcast 1

Abort
Hit any key to stop autoboot:  0
U-Boot-PetaLinux>
```

Figure 6 – Indirect kernel boot via Universal Bootloader (U-Boot)



```
Default QEMU
Creating /dev/flash/* device nodes
Mounting /etc/config:
Populating /etc/config:
flatfsd: Nonexistent or bad flatfs (-48), creating new one...
flatfsd: Failed to write flatfs (-48): No such file or directory
flatfsd: Created 7 configuration files (160 bytes)
Mounting devpts:
Mounting all filesystem
Setting hostname:
Bringing up network interfaces:
udhcpc (v1.14.3) started
Sending discover...
Sending select for 192.168.10.3...
Lease of 192.168.10.3 obtained, lease time 864000
adding dns 127.0.0.1
Starting portmap:
Starting uWeb server:

Welcome to

PetaLinux

on kintex7


kintex7 login: root
Password:
~ # ls
bin       etc       init      linuxrc   proc      tmp       var
dev       home      lib       mnt       sys       usr
~ #
```

Figure 7 – Running PetaLinux through QEMU

The application is designed to control an 8-bit discrete output and is tested by connecting LEDs onboard to the GPIO.

The value 15 was observed in the terminal, which confirms the DDR3 memory read and write operations were operating perfectly.

2. Accessing GPIO Using /dev/mem

For our next application test, we used a PetaLinux application titled *gpio-dev-mem-test.c* to access general-purpose I/O (GPIO). The application is designed to control an 8-bit discrete output and test that output by connecting LEDs onboard to the GPIO. In order to access any device from the user space, open */dev/mem* and then use *mmap()* to map the device to memory. The start and end addresses of the LED GPIO we used are *0x40000000* and *0x4fffffff*, respectively.

The GPIO peripheral has two registers: a data register (GPIO_DATA) and a direction register (GPIO_TRI_OFFSET). In order to read the status of the GPIO, we set the direction bit to 1 (i.e., GPIO_TRI_OFFSET=1) and read the data from the data register. To write data to GPIO, set the bit to 0 and write the value to the data register. Data is written on GPIO using the following command on the PetaLinux terminal:

```
#gpio-dev-mem-test -g 0x40000000 -o 255
```

where *gpio-dev-mem-test* is the application name, *–g* is the GPIO physical address, *–o* is output and 255 is the value transmitted from GPIO, which is connected to LEDs. The results of the test were verified when the LEDs lit up as programmed.

3. Accessing GPIO Using UIO

An alternative way of accessing GPIO is via the userspace input/output. We used a PetaLinux application titled *gpio-uio-test.c* to access the GPIO using UIO. The application is designed to control an 8-bit discrete output and is tested by connecting LEDs onboard to the GPIO. A UIO device is represented as */dev/uioX* in the file system. In order to access GPIO through UIO, we opened */dev/uioX* or *sys/class/uio/ui0* and then used the *mmap()* call. We configured the kernel to support UIO and enabled the UIO framework in the kernel. Then, using a parameter called "Compatibility," we set the LEDs' GPIO to be controlled as the UIO device, instead of the normal GPIO device. We also changed the label of the device from *gpio@40000000* to *leds@40000000*.

We then rebuilt PetaLinux and tested the GPIO access using UIO. We obtained details about the information of UIO modules loaded using

```
# ls /sys/class/uio/
 uio0 uio1 uio2
```

The name of the UIO and its address are found in */sys/class/uio/uioX*. We used the following command to access GPIO LED through the UIO driver:

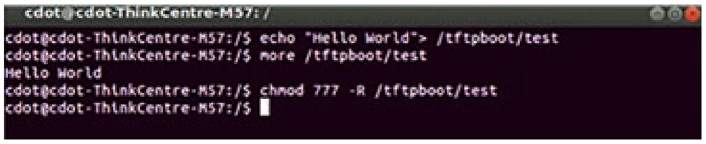
```
# cd "/sys/class/uio/uioX
# gpio-uio-test -d /dev/uio1 -o 255
```

Here, *gpio-uio-test* is the application name, *–d* is the device path, *–o* is the output and 255 is the value passed out to GPIO through UIO. The results were verified by the LEDs glowing based on the data written on GPIO lines using the above command.

4. File Transfer Application

For our last test, we transferred a file from a server to a client, where the server is the host PC and the client is the KC705 board. For this test, we connected the server and client through an Ethernet cable. We used the Trivial File Transfer Protocol (TFTP), which is well known for its simplicity and is generally used for automated transfer of configuration or boot files. In order to test the file transfer from server to client using TFTP, we created a file called *test* in the server PC at */tftpboot*. We used the following commands to write "Hello World" in the file and to view the contents in the same file (as shown in Figure 8):

```
@ echo "Hello World" > /tftpboot/test
@ more /tftpboot/test
```



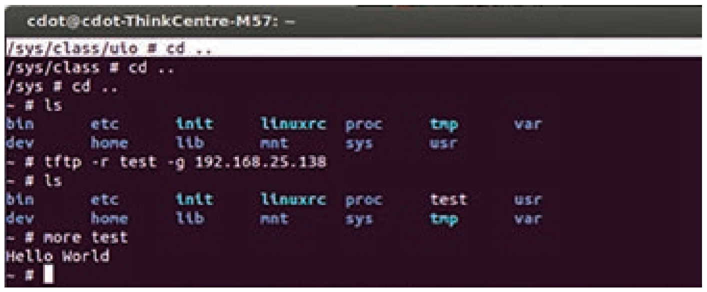
```
cdot@cdot-ThinkCentre-M57: /
cdot@cdot-ThinkCentre-M57:/$ echo "Hello World"> /tftpboot/test
cdot@cdot-ThinkCentre-M57:/$ more /tftpboot/test
Hello World
cdot@cdot-ThinkCentre-M57:/$ chmod 777 -R /tftpboot/test
cdot@cdot-ThinkCentre-M57:/$ █
```

Figure 8 – Snapshot of file creation in the server

To receive this file from the server, we typed the following get command (*-g*) in the PetaLinux terminal window that was running as the client on the KC705 board:

```
# tftp -r test -g 192.168.25.68
# ls -a
```

A new file was created with the filename "test" in the client (as shown in Figure 9). We can view the contents of this file using the *more* command, as seen in Figure 9.

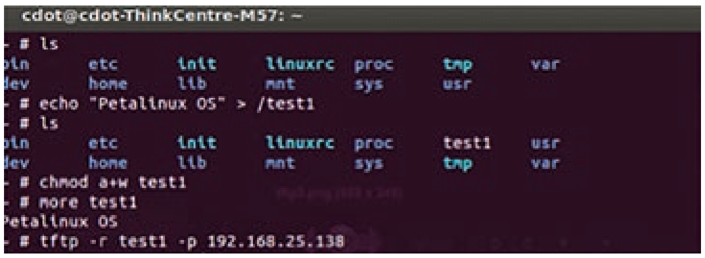


```
cdot@cdot-ThinkCentre-M57: ~
/sys/class/uio # cd ..
/sys/class # cd ..
/sys # cd ..
~ # ls
bin       etc       init      linuxrc   proc      tmp       var
dev       home      lib       mnt       sys       usr
~ # tftp -r test -g 192.168.25.138
~ # ls
bin       etc       init      linuxrc   proc      test      usr
dev       home      lib       mnt       sys       tmp       var
~ # more test
Hello World
~ # █
```

Figure 9 – Snapshot for file reception in the client

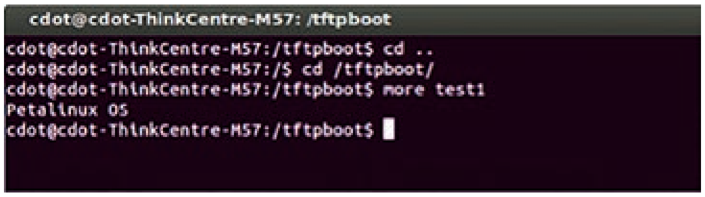
Similarly, transferring a file from the client to the server is done by creating a file called *test1* with the content "PetaLinux OS" in the client machine. To transmit the file from the client to the server, use the following "put" command (*-p*) in the PetaLinux terminal running from the client (as shown in Figure 10):

```
# tftp -r test1 -p 192.168.25.68
```



```
cdot@cdot-ThinkCentre-M57: ~
~ # ls
bin       etc       init      linuxrc   proc      tmp       var
dev       home      lib       mnt       sys       usr
~ # echo "Petalinux OS" > /test1
~ # ls
bin       etc       init      linuxrc   proc      test1     usr
dev       home      lib       mnt       sys       tmp       var
~ # chmod a+w test1
~ # more test1
Petalinux OS
~ # tftp -r test1 -p 192.168.25.138
```

Figure 10 – Snapshot for file transmission from client to server



```
cdot@cdot-ThinkCentre-M57: /tftpboot
cdot@cdot-ThinkCentre-M57:/tftpboot$ cd ..
cdot@cdot-ThinkCentre-M57:/$ cd /tftpboot/
cdot@cdot-ThinkCentre-M57:/tftpboot$ more test1
Petalinux OS
cdot@cdot-ThinkCentre-M57:/tftpboot$ █
```

Figure 11 – Snapshot of file reception in the server

A blank *test1* file is created in the server. Its contents are read after the file transfer operation, and the contents are verified as shown in Figure 11.

Implementing an embedded system and running PetaLinux on an FPGA were pretty straightforward operations. Next, we plan to implement a design using remote programming where the boot files are transferred via Ethernet and the client is capable of running a new application.

REFERENCES

1. Kynan Fraser, "MicroBlaze Running uClinux," Advanced Computer Architecture from *http://www.cse.unsw.edu.au/~cs4211*

2. Xilkernel from Xilinx Inc., Version 3.0, December 2006

3. PetaLinux SDK User Guide from Xilinx Inc., UG976, April 2013

4. Gokhan Ugurel and Cuneyt F. Bazlamacci, "Context Switching Time and Memory Footprint Comparison of Xilkernel and µC/OS-II on MicroBlaze," 7th International Conference on Electrical and Electronics Engineering, December 2011, Bursa, Turkey, pp.52-55

5. Chenxin Zhang, Kleves Lamaj, Monthadar Al Jaberi and Praveen Mayakar, "Damn Small Network Attached Storage (NAS)," Project Report, Advanced Embedded Systems Course, Lunds Tekniska Hogskola, November 2008

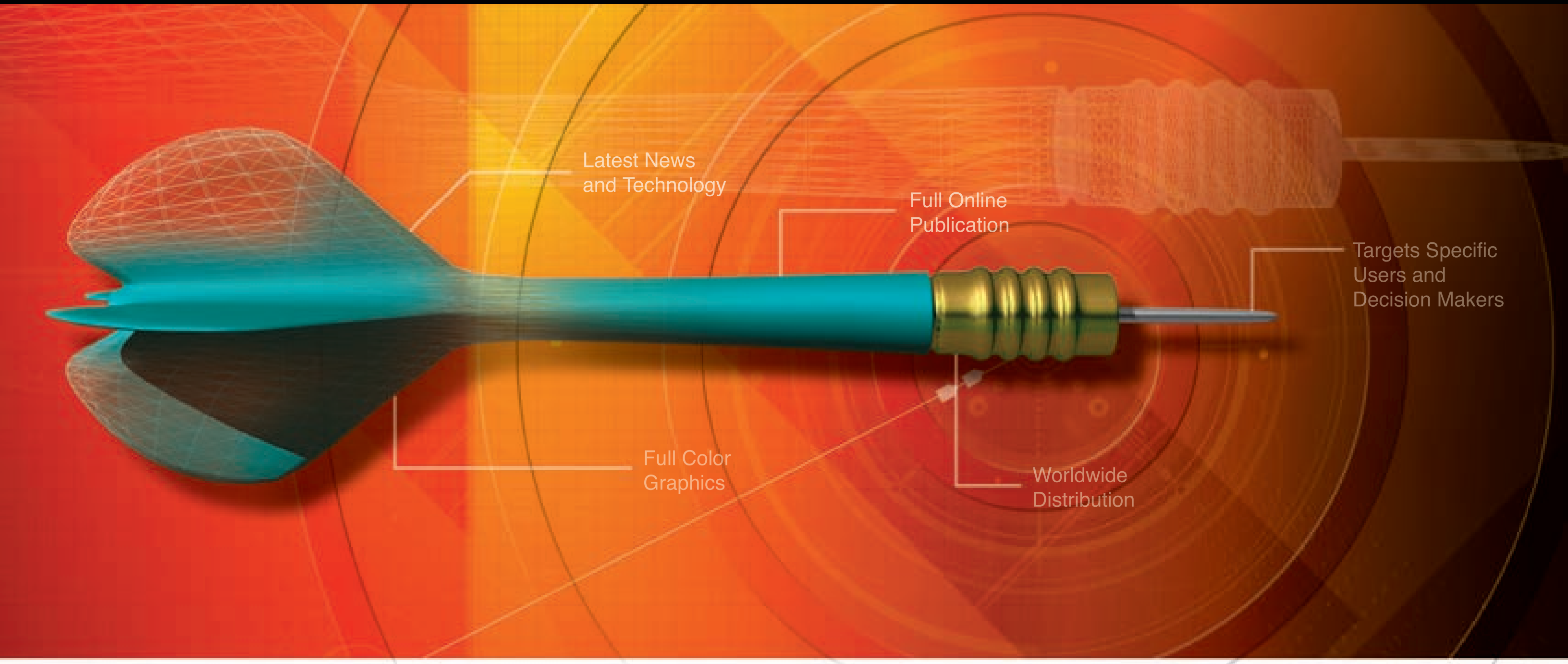
6. Platform Studio User Guide from Xilinx Inc., UG113, Version 1.0, March 2004

7. "EDK Concepts, Tools and Techniques: A Hands-On Guide to Effective Embedded System Design," Xilinx Inc., UG683, Version 14.1, April 2012

8. PetaLinux Application Development Guide from Xilinx Inc., UG981, April 2013

9. PetaLinux QEMU Simulation Guide from Xilinx Inc., UG982, November 2013

Get on Target


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

Let Xcell Publications help you get your message out to thousands of programmable logic users.

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

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

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

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

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

Xcell journal
SOLUTIONS FOR A PROGRAMMABLE WORLD
Xilinx Ships Industry's First 20-nm All Programmable Devices
4.4M Logic Cells
XILINX

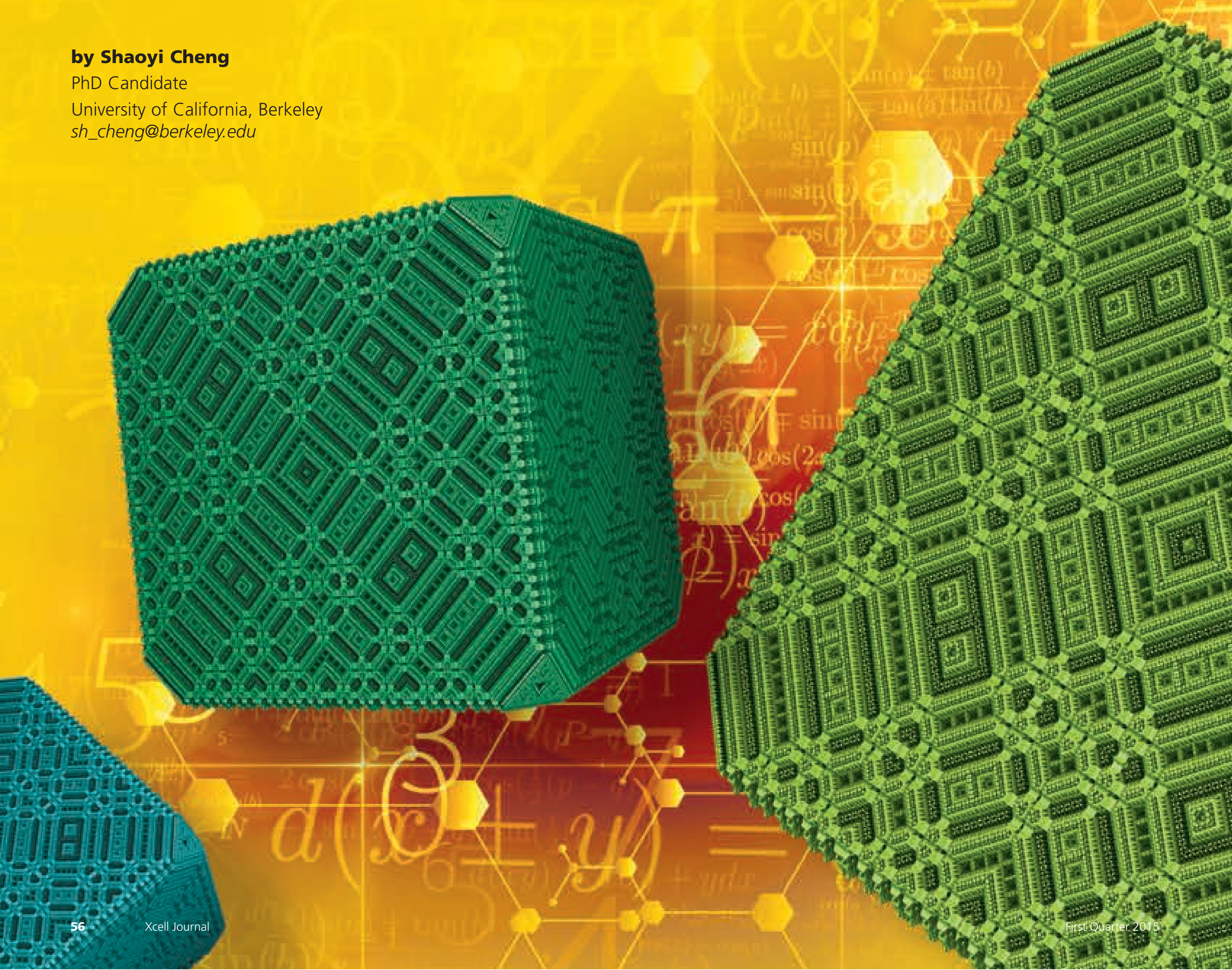
Xcell PUBLICATIONS

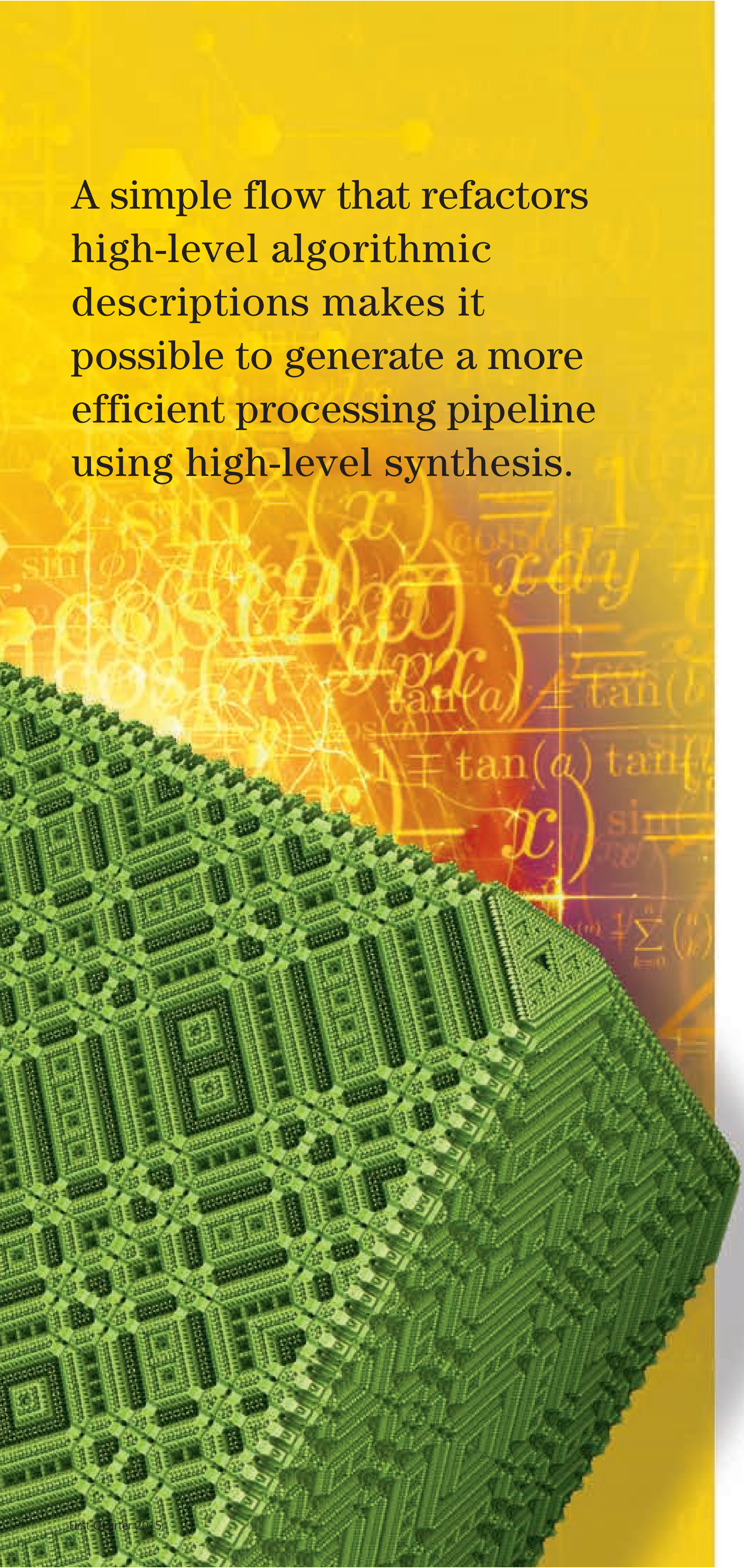
www.xilinx.com/xcell/

XILINX®

Try Algorithm Refactoring to Generate an Efficient Processing Pipeline with Vivado HLS

by Shaoyi Cheng
PhD Candidate
University of California, Berkeley
sh_cheng@berkeley.edu





A simple flow that refactors high-level algorithmic descriptions makes it possible to generate a more efficient processing pipeline using high-level synthesis.

When you are wrestling with a computation kernel with regular memory access patterns and easily extractable parallelism between loop iterations, the Vivado® Design Suite high-level synthesis (HLS) tool can be a great resource for creating high-performance accelerators. By adding a few pragmas to a high-level algorithmic description in C, you can quickly implement a high-throughput processing engine on your Xilinx® FPGA. In combination with DMA mechanisms that are managed by software, the result is an orders-of-magnitude speed-up when compared against general-purpose processors.

However, real-life applications often contain complex memory accesses that are not as easy to deal with, especially if we venture out of the realms of scientific computing and signal-processing algorithms. We have devised a simple technique that you can use in some of these situations to produce efficient processing pipelines. But before we go into the details, let's first take a look at how Vivado HLS works and more importantly, when it doesn't.

HOW DO THE HLS TOOLS WORK?

High-level synthesis attempts to capture parallelism in the control data flow graph (CDFG) described by high-level languages. Compute operations and memory accesses are allocated and scheduled according to the dependency constraints between them and the resource constraints of the target platform. Activation of a particular operation in the circuit is associated with a certain clock cycle, while a central controller that's synthesized alongside the data path orchestrates the execution of the entire CDFG.

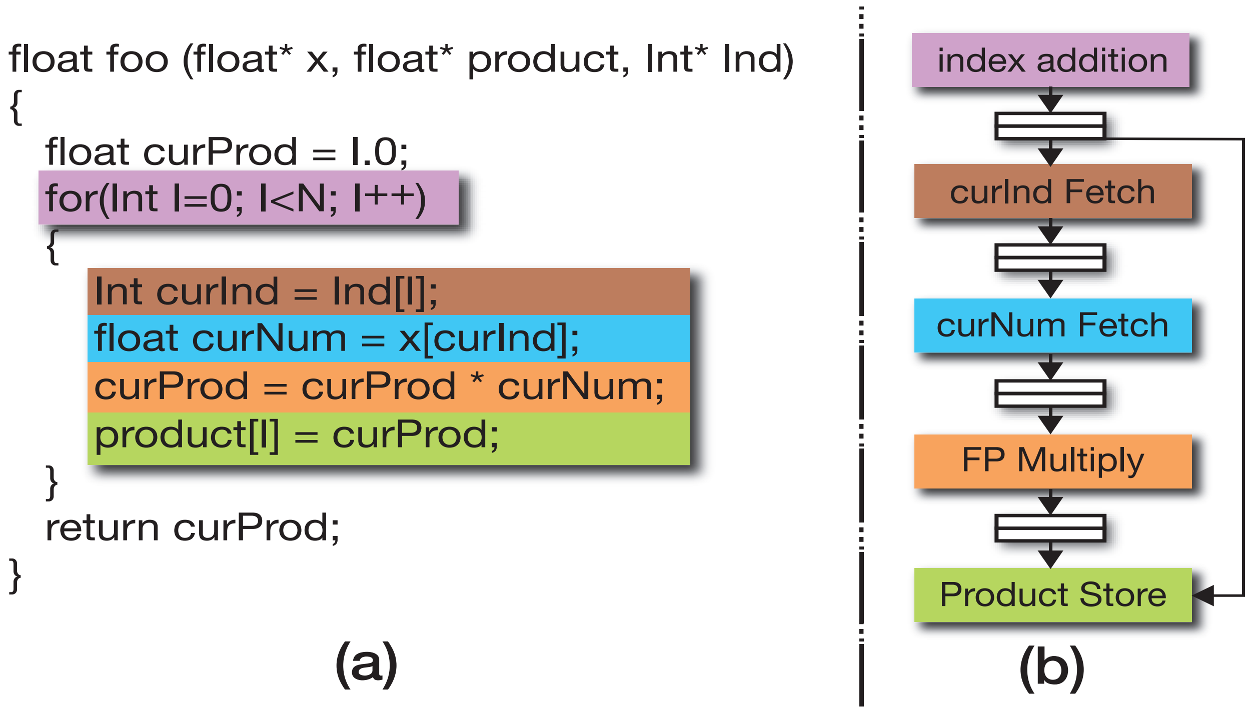
Naively applying HLS on the kernel would create a data path with a lot of instruction-level parallelism. But when it's activated, it would need to stop frequently while waiting for data to be brought in.

As the scheduling is done statically, the run-time behavior of the accelerator is rather simple. Different parts of the generated circuit run in lockstep with each other; no dynamic dependency-checking mechanisms such as those present in high-performance CPUs are needed. In the function shown in Figure 1(a), for instance, the loop index addition and the load of curInd can be parallelized. Also, the next iteration can start before the current iteration finishes.

Meanwhile, because the floating-point multiply always uses the result of multiply from the previous iteration, the shortest interval in which we can start a new iteration is limited by the latency of the floating-point multiplier. The execution schedule of this function is shown in Figure 2(a).

WHEN IS THIS APPROACH SUBOPTIMAL?

The problem with this approach is that the entire data flow graph is running on a rigid schedule. Stalls introduced by off-chip communication propagate to the entire processing engine, resulting in significant performance degradation. This is not an issue when the memory access pattern is known a priori, such that the data can be moved on-chip before it is needed, or if the data set is small enough to be entirely buffered on the FPGA. However, for many interesting algorithms, the data access depends on the result of computation and the memory footprint requires the use of off-chip RAM. Now, naively applying HLS on the kernel would create a data path with a lot of instruction-level parallelism. But when it's activated, it would



```
float foo (float* x, float* product, Int* Ind)
{
   float curProd = I.0;
   for(Int I=0; I<N; I++)
   {
      Int curInd = Ind[I];
      float curNum = x[curInd];
      curProd = curProd * curNum;
      product[I] = curProd;
   }
   return curProd;
}
```

Figure 1 – An example design: (a) A function containing irregular memory access pattern; (b) pipeline structure from a possible refactoring

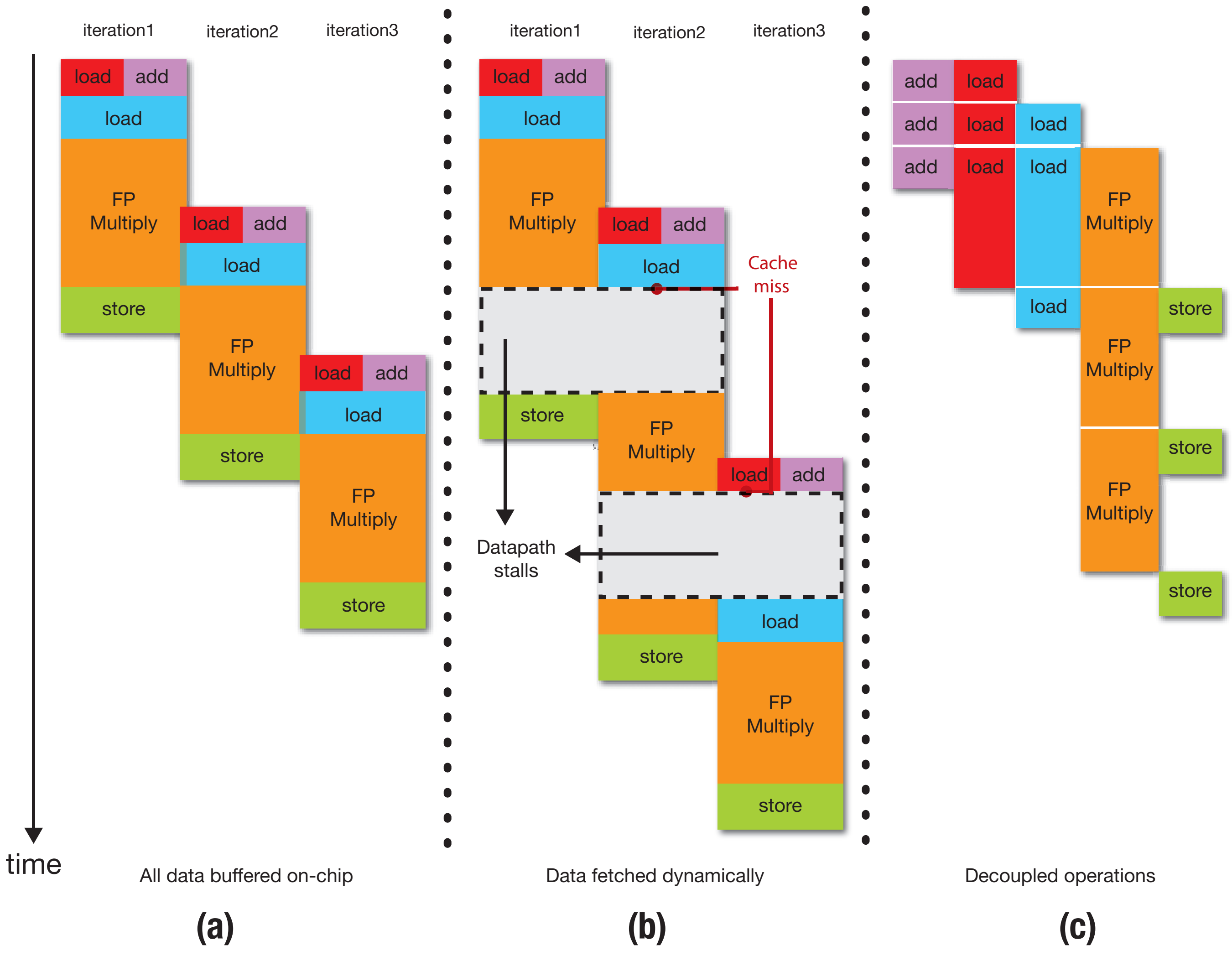


Figure 2 – Execution schedule in different scenarios: (a) when all data is buffered on-chip; (b) with data fetched dynamically; and (c) with decoupled operations

need to stop frequently while waiting for data to be brought in.

Figure 2(b) shows the execution of the generated hardware module for our example function when the data set is too large and needs to be dynamically fetched into the on-chip cache. Note how the slowdown reflects the combination of all cache miss latencies. This does not have to be the case, though, since there are parts of the computation graph whose progress does not need the memory data to be immediately available. These sections should be able to move forward. This extra bit of freedom in the execution schedule can potentially have a significant effect, as we shall see.

MAKING A CASE FOR REFACTORING/DECOUPLING

Let's look at the example function we had earlier. Imagine if the execution of floating-point multiplies and the data accesses are not all tied together by a unified schedule. While one load operator is waiting for data to come back, the other load operator can initiate new memory requests and the multiplier's execution also moves forward. To achieve this, there should be one module responsible for each memory access, running on its own schedule. Also, the multiplier unit should be executing asynchronously with all the memory operations.

The data dependencies between the different modules are communicated through hardware FIFOs. For our example, a possible refactoring is shown in Figure 1(b). The hardware queues used for communication between stages can buffer data already fetched but not yet used. When the memory access parts are stalling for cache misses, the backlog of data already produced so far can continue supplying the multiplier unit. Over a long period of time, the stalls introduced can be shadowed by the long latency of the floating-point multiplication.

Figure 2(c) shows the execution schedule when this decoupled processing pipeline is used. The latencies through the FIFOs are not taken into consideration here, but the effect should be minimal if there are a large number of iterations.

HOW DO WE REFACTOR?

To generate the pipeline of decoupled processing modules, you first need to cluster the instructions in the original CDFG to form subgraphs. To maximize the performance of the resulting implementation, the clustering method must address a few requirements.

First, as we have seen earlier, the Vivado HLS tools use software pipelining to initiate new iterations before previous ones are completed. The latency of the longest circular dependence in the CDFG dictates the minimum interval with which a new iteration can be initiated, which ultimately bounds the overall throughput an accelerator can achieve. It is therefore crucial that these dependency cycles do not traverse multiple subgraphs, as the FIFOs used for communication between modules always add latency.

Second, it is beneficial to separate memory operations from dependency cycles involving long-latency compute, so that cache misses can be "shadowed" by the slow rate of data consumption. Here, "long latency" means the operation takes more than one cycle and for our purpose, the Vivado HLS schedule is used to obtain this metric. So for instance, a multiply would be a long-latency operation while an integer addition is not.

Lastly, to localize the effects of stalls introduced by cache misses, you will also want to minimize the number of memory operations in each subgraph, especially when they address different parts of the memory space.

It's easy to satisfy the first requirement—keeping dependency cycles from traversing multiple subgraphs—by finding strongly connected components (SCCs) in the original data flow graph and collapsing them into nodes before separating them into different clusters. As a result of this process, we wind up with a directed acyclic graph, with some nodes being simple instructions and others being a set of operations dependent on each other.

To satisfy the second and third requirements—separating memory operations and localizing the effects of stalls—we can perform a topological sort of these nodes and then divide them up. The simplest way to do the division is to draw a "boundary" after every memory operation or long-latency SCC node. Figure 3 shows how to apply this approach to our motivating example. The correspondence between this clustering and the pipeline structure in Figure 1 should be apparent. Each of these subgraphs is a new C function that can be pushed through HLS independently. These subgraphs will be executing out of step with one another.

We built a simple source-to-source transformation tool to perform this refactoring. We used Xilinx IP cores for the FIFOs connecting individual modules generated. There is certainly more than one way to refactor a given computation kernel, and the design space exploration is still in progress.

PIPELINED MEMORY ACCESSES

Having an initial implementation of a decoupled processing pipeline, there are optimizations we can perform to improve its efficiency. As we have seen, when a C function is mapped using HLS, the memory reads are blocking. This is still the case for the individual stages in our pipeline. For instance, the module responsible for loading x[curInd] may be stalling while waiting for the data even though the next curInd is already available and there is enough space in the FIFO downstream.

To fix this problem, we can make a transformation to pipeline the memory accesses. Instead of performing a simple memory load in the C function for that particular stage, we replace the load with a push of addresses to a new FIFO. Then, a new hardware module is instantiated separately to read addresses off the address FIFO and send them to the memory subsystem. The data coming back is directly pushed onto the downstream FIFO. Effectively, the memory access is now pipelined.

The push operation for the addresses can be represented by a memory store to a FIFO interface in Vivado HLS, but the

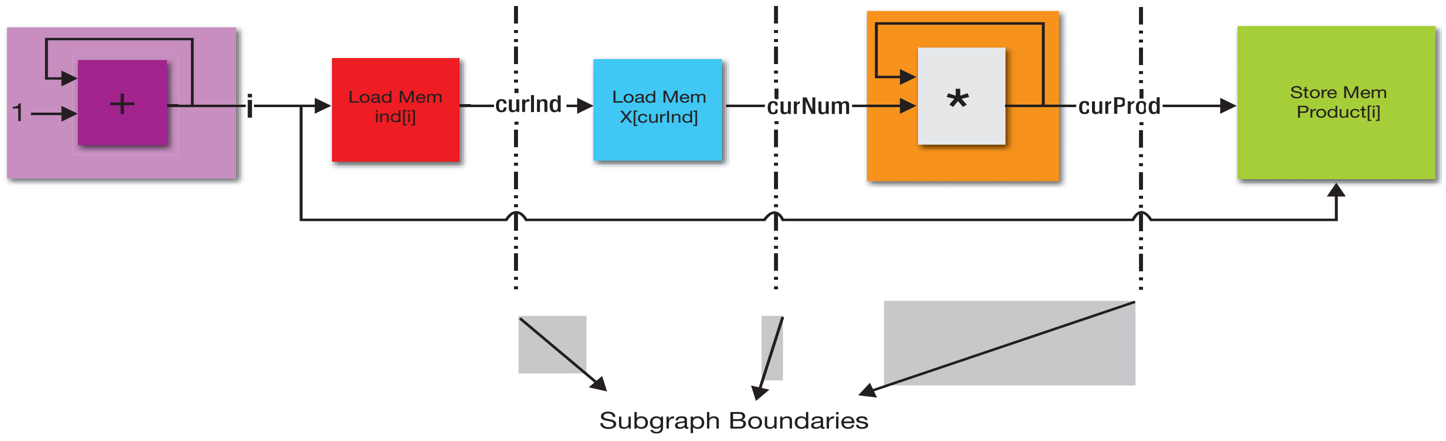


Figure 3 – Refactoring into subgraphs

```
for (w = 1; w <= W; w++) {
  int option1 = opt[n-1][ w];
  int option2 = -999999;
  int opt_without = opt[n-1][ w-cur_weight];
  if (cur_weight <= w)
    option2 = cur_profit + opt_without;
    opt[n][w] = option1> option2? option1:option2;
    sol [n][ w] = option2> oprion1? 1:0;
}
```

Figure 4 – The knapsack problem

hardware module monitoring the downstream FIFO and firing off the memory request is implemented with Verilog. This is because the outgoing addresses and response data are not bundled together in a memory interface synthesizable by Vivado HLS. However, this is a simple module that gets reused many times across different benchmarks, and thus the design effort is amortized.

DUPLICATION OR COMMUNICATION?

The FIFOs introduced to move data between stages represent significant overhead in refactoring the kernel and generating the decoupled processing pipeline. It is often beneficial to eliminate some of these FIFOs by duplicating a few computation instructions, as even a minimal-depth FIFO can cost a nontrivial amount of FPGA resources.

In general, in searching for the absolute best design point in this trade-off, you can apply cost-modeling and formal optimization techniques. But in most benchmarks, just duplicating the simple loop counters for every one of its users can save a lot of area, and that's what we have done. In the motivating example, this optimization involves duplicating the integer adder for i, so the store to Product[i] does not need to get its index from another module.

BURST-MEMORY ACCESS

A third optimization is burst-memory access. To make more efficient use of the memory bandwidth, it is desirable to have one memory transaction carrying multiple words of data. The AXI bus protocol allows you to specify burst length, and with small modifications to the decoupled C functions and the pipelined memory access module, we can take advantage of this capability.

In addition to generating addresses, each memory operator in decoupled C functions also computes a burst length when a contiguous chunk of memory is accessed. The duplication of loop counters also helps in the generation of burst access, as the number of words accessed can be determined locally within each decoupled function.

EXPERIMENTAL EVALUATION

We did a few case studies applying the described approach. To evaluate the benefit of the method, we compare the decoupled processing pipelines (DPPs) generated using our approach with the accelerators generated by naively applying HLS (Naïve). When invoking Vivado HLS for the Naïve/DPP implementations, we set the target clock frequency to 150 MHz and used the highest achievable clock rates after place-and-route. Also, we tried various mechanisms for the interaction between accelerators and the memory subsystem. ACP and HP ports are used, and for each, we also instantiated a 64-kbyte cache on the reconfigurable array.

The physical device we used for the experiments was the Zynq®-7000

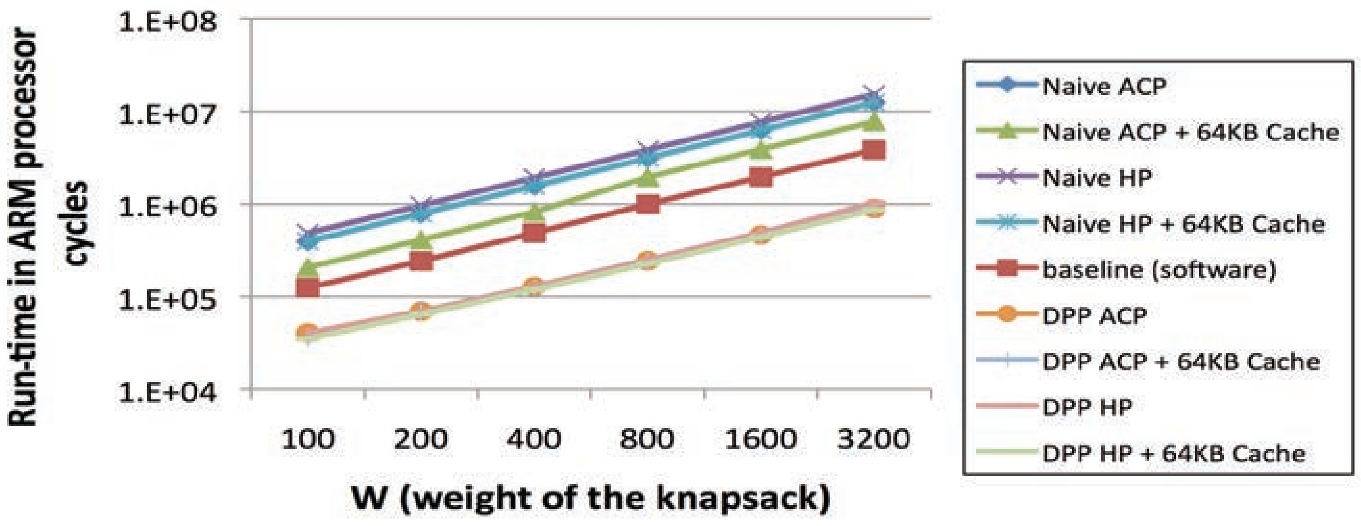


Figure 5 – Run-time comparison for knapsack problem

XC7Z020 All Programmable SoC from Xilinx, installed on the ZedBoard evaluation platform. We also ran the software version of the applications on the ARM® processor on the Zynq SoC and used its performance as the baseline for our experiments. All the accelerators generated are self-contained; they do not need any DMA mechanism to move data in and out of the reconfigurable fabric.

CASE STUDY 1: KNAPSACK PROBLEM

The knapsack problem is a well-known combinatorial problem that can be solved using dynamic programming. The skeleton of the kernel is shown in Figure 4. In particular, the variables in boldface type are all read from memory during run-time. Therefore, the exact locations from which the variable opt_without is loaded are not known a priori. When *w* and *n* are large, we are unable to buffer the entire opt array on-chip. We can only let the computation engine fetch the needed portion of it.

Figure 5 shows the run-time comparison between the accelerators generated using our approach (DPP) and the accelerator generated by naively pushing the function through HLS (Naïve). The chart also shows the performance of running the function on an ARM processor. We fix *n* (number of items) at 40, and vary *w* (total weight of the knapsack) from 100 to 3,200.

It is clear from the comparison that naively mapping the software kernel using Vivado HLS generates an accelerator with performance much slower than the baseline. The superscalar, out-of-order ARM core on the Zynq SoC is capable of exploiting instruction-level parallelism to a good extent and also has a high-performance on-chip cache. The additional parallelism extracted by the Vivado HLS tool is evidently not enough to compensate for the clock frequency advantage the hard processor core has over the programmable logic and the longer data access latency from the reconfigurable array.

When the kernel is decoupled into multiple processing stages, however, the performance is much better than that of the ARM processor (~4.5x). Also, the difference among various memory access mechanisms when DPP is used is rather small—the sensitivity to memory access latency is a lot better with our approach.

CASE STUDY 2: SPARSE MATRIX VECTOR MULTIPLY

Sparse matrix vector (SpMV) multiply is a computation kernel that has been studied, transformed and benchmarked many different ways in various research projects. Our purpose here is not to produce the best-performing SpMV multiply using special data-structure and memory-allocation schemes. Rather, we want to see—given the most basic algorithm description—how much benefit a refactoring pass can provide when using Vivado HLS.

As shown in Figure 6, for our experiment the sparse matrix is stored in compressed sparse row (CSR) format. Loads from an index array are performed before numbers can be fetched for the actual floating-point multiply. Again, the control flow and memory locations accessed depend on values only known during run-time.

For the run-time comparison shown in Figure 7, the matrix is populated with an average density of 1/16, and the dimension is varied between 32 and 2,048.

```
for(s =0; s<dim; s++)
{
  int kend = ptr[s];
  int k;
  float curY = y[s];
  for(k = kbegin; k<kend; k++){
    int curInd = indArray[k];
    curY = curY +valArray[k] * xvec[curInd];
   }
   Y[s] = curY;
   kbegin = kend;
}
```

Figure 6 – Sparse matrix vector multiply

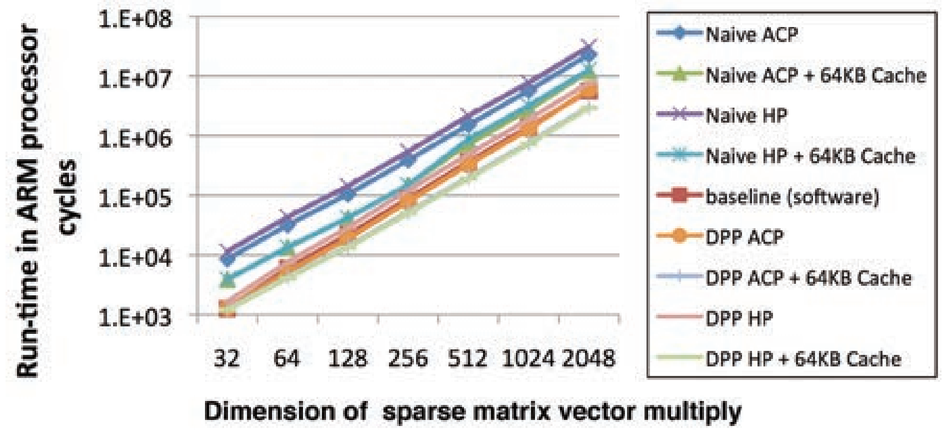


Figure 7 – Run-time comparison for sparse matrix vector multiply

Here, the naïve mapping again lags the software version in performance. The decoupled processing pipeline generated with our approach has about the same performance as the baseline when no on-FPGA cache is used.

With a 64-kbyte cache instantiated on the reconfigurable array, the performance of DPP is close to double that of the baseline. The addition of caches has a more pronounced effect on the performance of the DPP compared with the previous benchmark.

CASE STUDY 3: FLOYD-WARSHALL ALGORITHM

Floyd-Warshall is a graph algorithm that's used to find the pairwise shortest path between any pair of vertices. The memory access pattern is simpler than in previous benchmarks. Thus, there may be a way to devise a DMA-plus-accelerator setup to have a good overlap of computation and off-chip communication. Our approach tries to achieve this overlapping automatically, but we have not done the study to show the gap between the absolute optimal and what we have obtained here.

We do have, however, a comparison of run-time just as with the previous benchmarks. Here, we vary the size of our graph from 40 nodes to 160 nodes. Each node has on average 1/3 of all nodes as its neighbors.

Our results are very similar to those in the knapsack problem. The decoupled processing pipelines achieved performance about 3x that of the software baseline, which has more than twice the throughput of any naïve mappings. The effect on FPGA cache is also small when DPPs are used, demonstrating their tolerance toward memory access latency.

Our simple technique creates processing pipelines that can make better use of memory bandwidth and have better tolerance toward memory latency, thus improving Vivado HLS performance. The described method decouples memory accesses and long dependence cycles in the control data flow graph, such that cache misses do not stall the other parts of the accelerator.

```
for(k=0; k<V; k++)
    for(i=0; i<V; i++)
     if(i!=k) {
        int dik = dis[i][k];
        for(j=0; j<V; j++)
          if(j!=k) {
            int dkj =  dist[k][j];
            int dij =  dist[i][j];
            if(dik +  dkj < dij )
              dist[i][j] = dik + dkj;
          }
     }
```

Figure 8 – The Floyd-Warshall algorithm

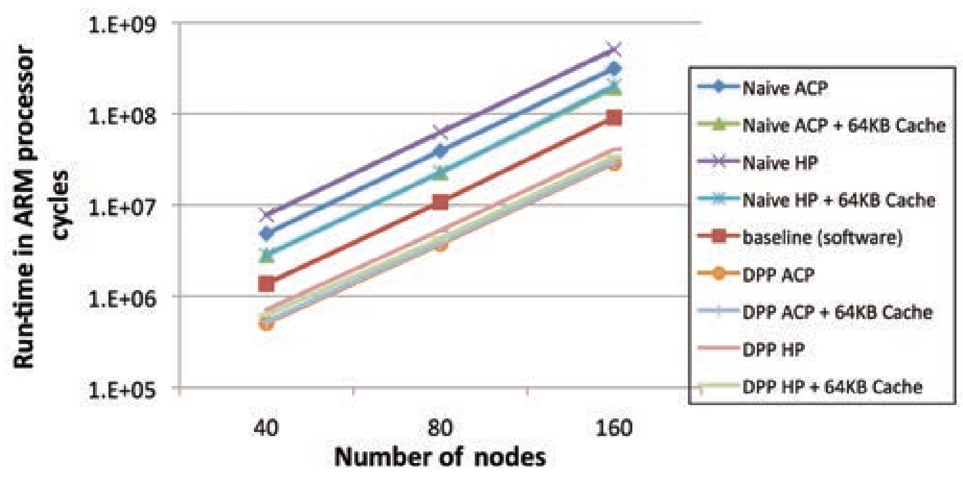


Figure 9 – Run-time comparison for Floyd-Warshall algorithm

Opal Kelly

XEM7350 KINTEX 7

✓ Available K70T, K160T, and K410T
✓ FMC HPC Carrier
✓ USB 3.0 transfers over 340 MB/s
✓ 8 Gigabit Transceiver Lanes
✓ 512 MiB DDR3
✓ **Excellent JESD204B Host**

FrontPanel SDK

Software API and Driver

Device Firmware

FPGA IP

www.opalkelly.com

Latest and Greatest from the Xilinx Alliance Program Partners

Xpedite highlights the latest technology updates from the Xilinx Alliance partner ecosystem.

The Xilinx® Alliance Program is a worldwide ecosystem of qualified companies that collaborate with Xilinx to further the development of All Programmable technologies. Xilinx has built this ecosystem, leveraging open platforms and standards, to meet customer needs and is committed to its long-term success. Alliance members—including IP providers, EDA vendors, embedded software providers, system integrators and hardware suppliers—help accelerate your design productivity while minimizing risk. Here are some highlights.

LINUX-BASED MULTICORE FRAMEWORK FOR ZYNQ ULTRASCALE+ MPSOC

Given that the upcoming Zynq® UltraScale+™ MPSoC from Xilinx is characterized by increased levels of capability, performance and complexity, application developers will need new and improved software development paradigms to efficiently manage and leverage the heterogeneous processing power offered by this device.

The Mentor Embedded Multicore Framework from Mentor Graphics provides an enabling infrastructure to manage the life cycle of compute resources and interprocessor communications in heterogeneous multiprocessing environments. The initial integration of the Mentor portfolio showcases SMP Linux running on the quad ARM® Cortex™-A53 cores managing the life cycle and communications. The Nucleus RTOS runs on the ARM Cortex-R5 core using the Mentor Embedded Multicore Framework.

Mentor's Sourcery Codebench tools provide an integrated development environment for designing asymmetric multiprocessing (AMP) systems. Developers face unique challenges while debugging/profiling heterogeneous software contexts on heterogeneous cores. Mentor's embedded development tools hide these complexities from users and provide deep insight into the system run-time. Offerings include:

- Tools for resource partitioning in AMP systems (available later this year)
- Tools for building and packaging remote firmware/applications
- IDE for debugging each individual software context present in the AMP system
- Ability to profile each OS/application context and analyze data in a unified time reference

For more information, visit http://www.mentor.com/embedded-software/.

MATHWORKS EXPANDS SUPPORT FOR ZYNQ-7000 ALL PROGRAMMABLE SOCS

Xilinx Alliance Program member MathWorks has expanded support for Xilinx's Zynq-7000 All Programmable SoCs in Release 2014b. This newest release of MATLAB® and Simulink® allows engineers and scientists using model-based design to accelerate their time-to-production with a higher degree of confidence, all within a single tool environment.

MathWorks and Xilinx have closely collaborated on joint technical development to bring this novel guided workflow to the market. The workflow allows customers to develop and simulate algorithms that can be validated quickly and deployed onto the Zynq SoC, leveraging automatically generated C and HDL code. This methodology leverages the Zynq SoC's dual-core ARM Cortex-A9 processors coupled with a powerful programmable logic fabric to enable engineers and scientists to design and implement algorithms on a single chip that consumes less space and power in the end system.

In addition to existing support for the Xilinx ISE® Design Suite and the Zynq Intelligent Drives Kit, the extended support in this latest release provides integration with the Xilinx Vivado® Design Suite and the Zynq SDR development platforms. As a result, engineers and scientists can prototype quickly on a hardware development platform and then incorporate the generated C and HDL code into production environments through the use of the Vivado Design Suite.

The expanded support for Xilinx SoCs is available immediately in MATLAB Release 2014b.

In addition, MathWorks also offers a two-day training class to help engineers get up and running quickly on this technology. For more information, visit the Release 2014b highlights page.

INTELLIPROP RELEASES NVME IP CORES FOR XILINX 7 SERIES AND ULTRASCALE FPGAS

Alliance member IntelliProp has collaborated with Xilinx to provide industry-standard NVMe host-interface and device-interface IP cores. IntelliProp's NVMe Host (IPC-NV164-HI) and NVMe Device (IPC-NV163-DT) cores make it possible to communicate with PCIe®-based storage designs implemented onto Xilinx FPGAs. IntelliProp's NVMe IP cores comply fully with the Nonvolatile Memory Express industry specification. They feature an application layer with a processor interface to provide register and memory access. The cores support attachment to the system bus, providing seamless IP access and ease of integration with any system design. They utilize the power of the hard PCIe block made available in Xilinx 7 series and UltraScale FPGAs and are supported in Verilog and VHDL.

IntelliProp's NVMe cores can be integrated onto 7 series and UltraScale FPGAs to provide an industry-compliant PCIe Gen1, Gen2 or Gen3 interface. The NVMe Host IP core is designed to be integrated with an NVMe-compliant host application to interface with an NVMe drive, posting commands to system memory queues and interacting with the endpoint device's register sets. The NVMe Device IP core provides an NVMe-compliant device application to process host commands and perform PCIe data management. Both IP cores are designed to allow for efficient data movement between host system memory and PCIe connected devices.

User applications can be created using the Vivado Design Suite and are packaged for IP Integrator with reference designs for rapid development. IntelliProp's IP cores are competitively priced and available to order immediately. Product details and ordering options are available at *http://www.intelliprop.com/* or *info@intelliprop.com*.

TOPIC SPEEDS EMBEDDED DEVELOPMENT THROUGH ONLINE SHOPPING

Xilinx premier partner Topic Embedded Solutions has opened an online store for high-quality embedded solutions. The company offers a completely modular portfolio of integrated solutions designed to drastically reduce development cycles.

To start, the Zynq SoC-based Miami SoM modules are industrial graded, ready to program and design-in. They come standard with a full, fast-boot mainline Linux board support package. The BSP is updated continuously and is available online to ensure customers stay current with the latest Linux software developments.

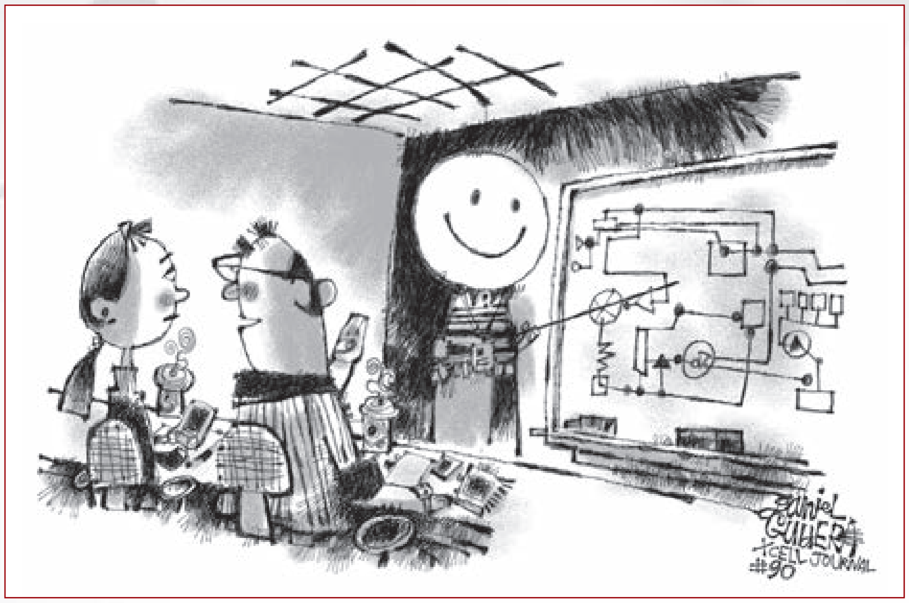
A full line of Florida Carrier Boards is available for complete system integration with Topic's SoMs. Specialized versions for medical and general use provide extensive off-the-shelf capabilities for development, prototyping and even production. Full schematics and layout of the carrier board comes standard with a purchase to ensure fast and successful integration.

Topic has also just released a new PCIe version of Florida that is ideal for data acceleration purposes such as video, signal or high-speed data processing.

A growing range of fully integrated development kits can simplify research, prototyping and fast engineering. These kits include touchscreens, application-specific I/O, all relevant cables and reference designs.

Find Topic Embedded Products at Embedded World in Nuremburg, Germany, in Hall 1, stand No. 1-136, on the Xilinx stand, through one of Topic's worldwide distribution partners or on the Topic online store at *www.topicproducts.com*.

Xpress Yourself in Our Caption Contest



DANIEL GUIDERA

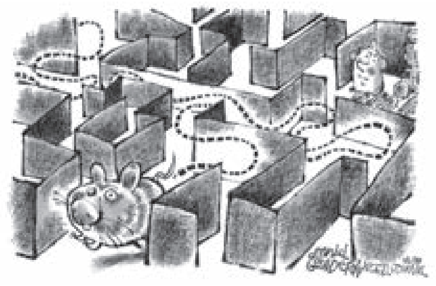
Everyone loves emoticons, but you wouldn't want to look like one. The individual leading this workshop seems to have taken the suggestion to "put on a happy face" a little too far. But don't worry, be happy—just write an engineering- or technology-related caption for our smiley-face cartoon and you could be the lucky winner of this issue's caption contest. The image might inspire a caption like "Max just came back from that Power of Positive Thinking executive retreat. Do you think he's been brainwashed?"

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*. After due deliberation, we will print the submissions we like the best in the next issue of *Xcell Journal*. The winner will receive a Digilent Zynq Zybo board, featuring the Xilinx® Zynq®-7000 All Programmable SoC (*http://www.xilinx.com/products/boards-and-kits/1-4AZFTE.htm*). Two runners-up will gain notoriety, fame and a cool, Xilinx-branded gift from our swag closet.

The contest begins at 12:01 a.m. Pacific Time on Feb. 23, 2015. All entries must be received by the sponsor by 5 p.m. PT on April 2, 2015.

Now, that's something to smile about!

DR. JAMES C. MALONEY, faculty member in the Electrical Engineering and Fuel Cell Engineering Technology and Information Technologies Division of Stark State College (North Canton, Ohio), won a shiny new Digilent Zynq Zybo board with this caption for the rodent-in-a-maze cartoon in Issue 89 of *Xcell Journal*:



"Who moved my cheese?"

Congratulations as well to our two runners-up:

"Gary missed the mark when his boss told him to implement a 'genetic' place-and-route algorithm."

— Michael Costanzo, electrical engineer, Brookhaven National Laboratory, Collider-Accelerator Department, Upton, NY

"This is what we ordered from Mouser Electronics?"

— Chris Lee, hardware engineer, Cisco Systems, San Jose, Calif.

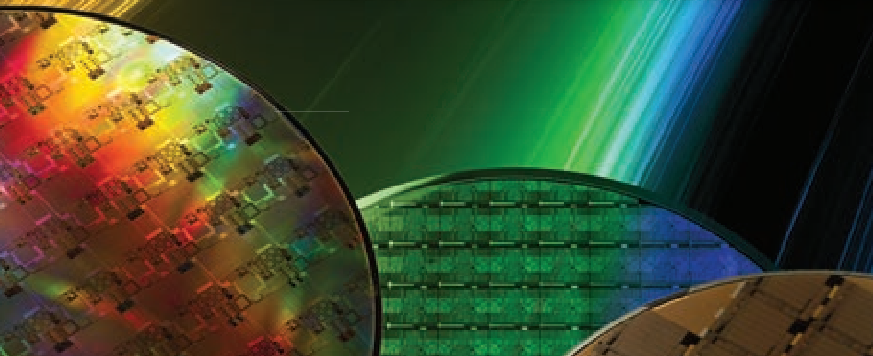
Synplify Premier

Achieve Best Quality of Results and Maximum Performance for Your FPGA Design

FPGAs keep getting bigger, but your schedule is not. There is no time to waste on numerous design iterations and long tool runtimes. Use the hierarchical and incremental techniques available in Synopsys' Synplify® software to bring in your schedule and meet aggressive performance and area goals.

- Unmatched quality of results
- Fastest turnaround time
- Fast and easy debug
- Results preservation from one run to the next

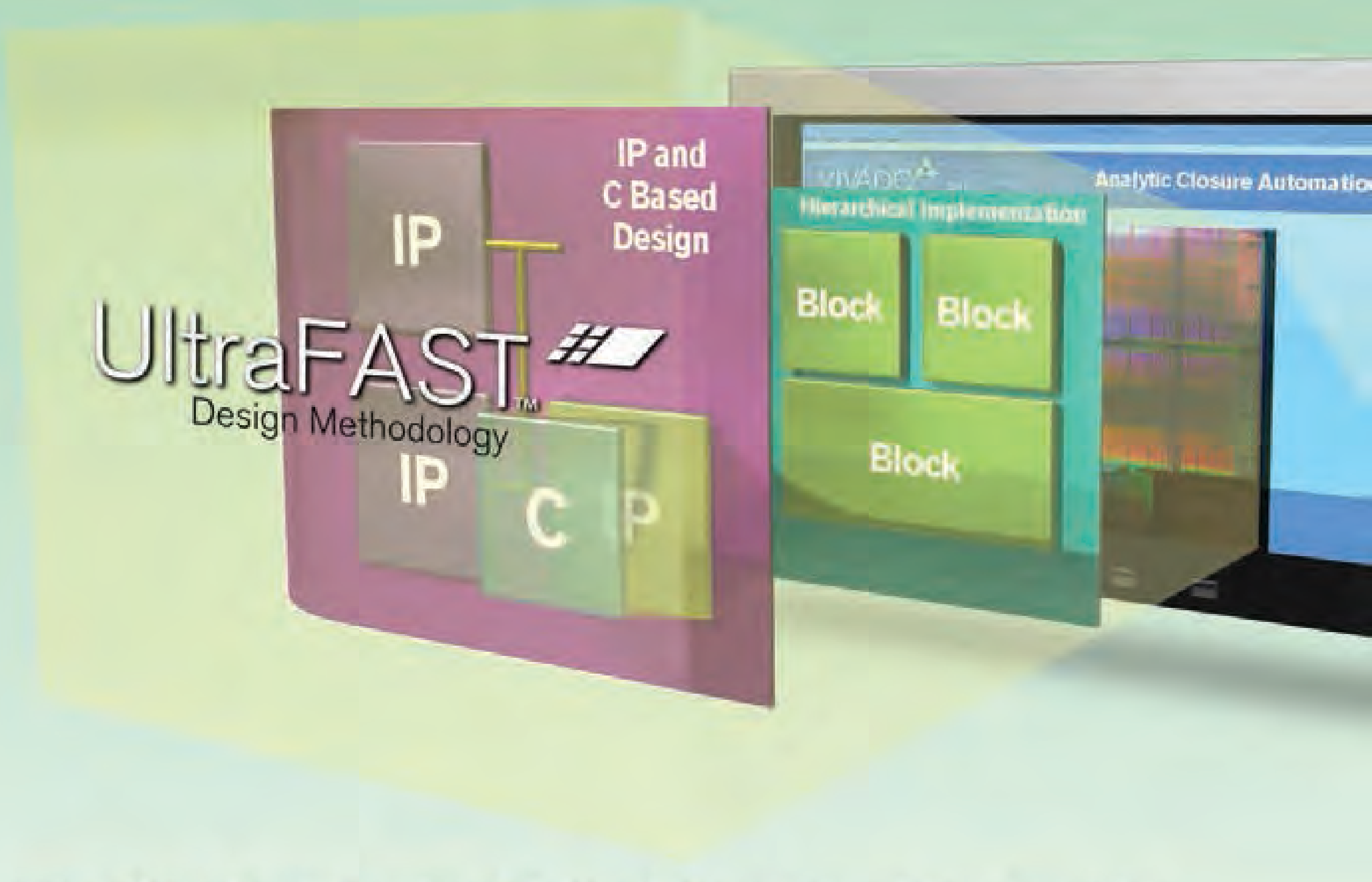
To learn more about how Synopsys FPGA design tools accelerate time to market, visit **www.synopsys.com/fpga**



SYNOPSYS®
Accelerating Innovation

Xilinx Introduces

The **UltraFast™ Design Methodology** for the Vivado® Design Suite



The UltraFast Design Methodology from Xilinx enables accelerated and predictable design cycles.

VIVADO

XILINX
ALL PROGRAMMABLE.