ISSUE 84, THIRD QUARTER 2013

Xcell journal

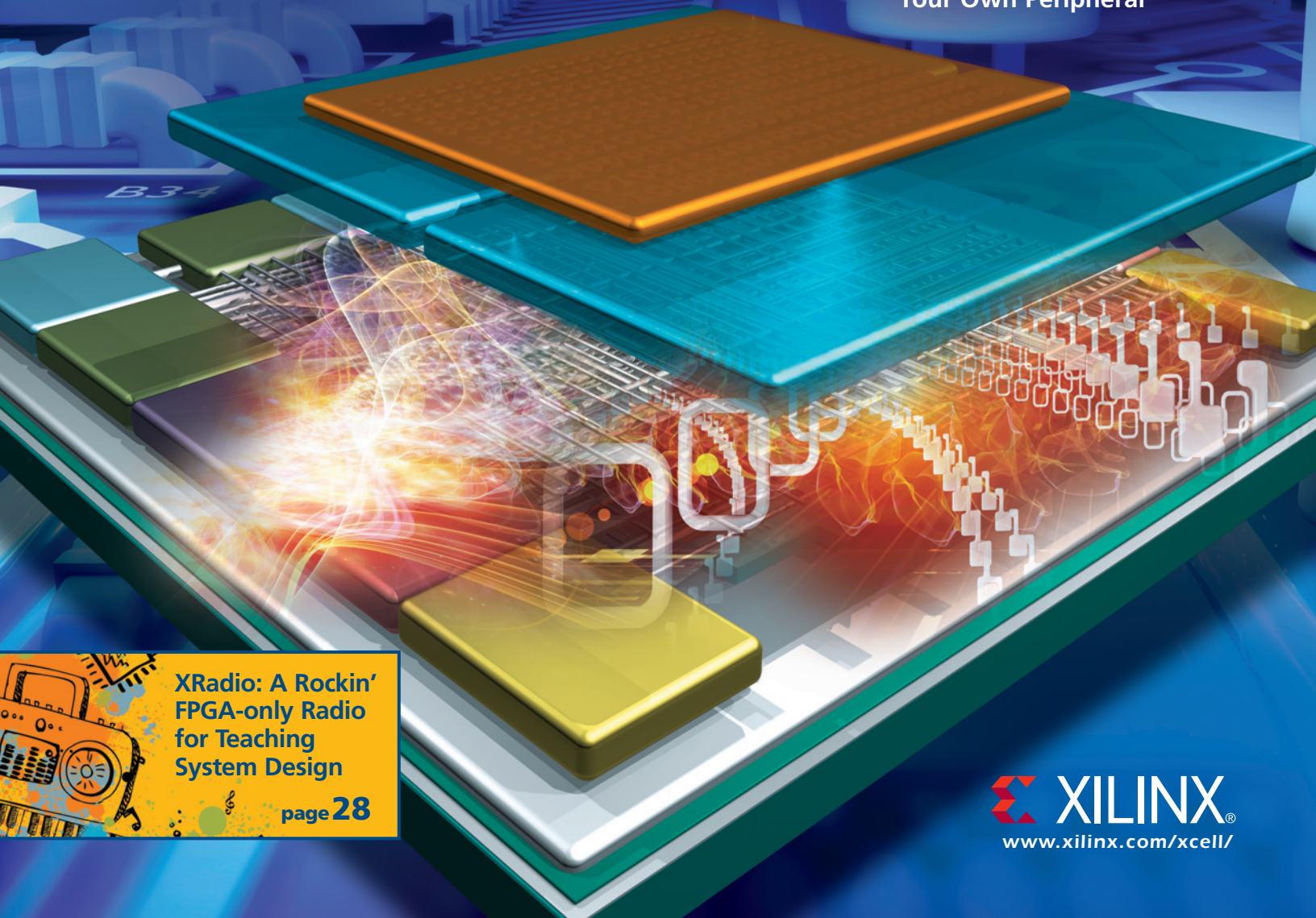
SOLUTIONS FOR A PROGRAMMABLE WORLD

Xilinx Goes UltraScale at 20 nm and FinFET

Efficient Bitcoin Miner System Implemented on Zynq SoC

Benchmark: Vivado's ESL Capabilities Speed IP Design on Zynq SoC

How to Boost Zynq SoC Performance by Creating Your Own Peripheral



XRadio: A Rockin' FPGA-only Radio for Teaching System Design

page 28

XILINX®

www.xilinx.com/xcell/

AVNET electronics marketing

Unlock Your System's True Potential

Xilinx® Zynq®-7000 AP SoC Mini-Module Plus System

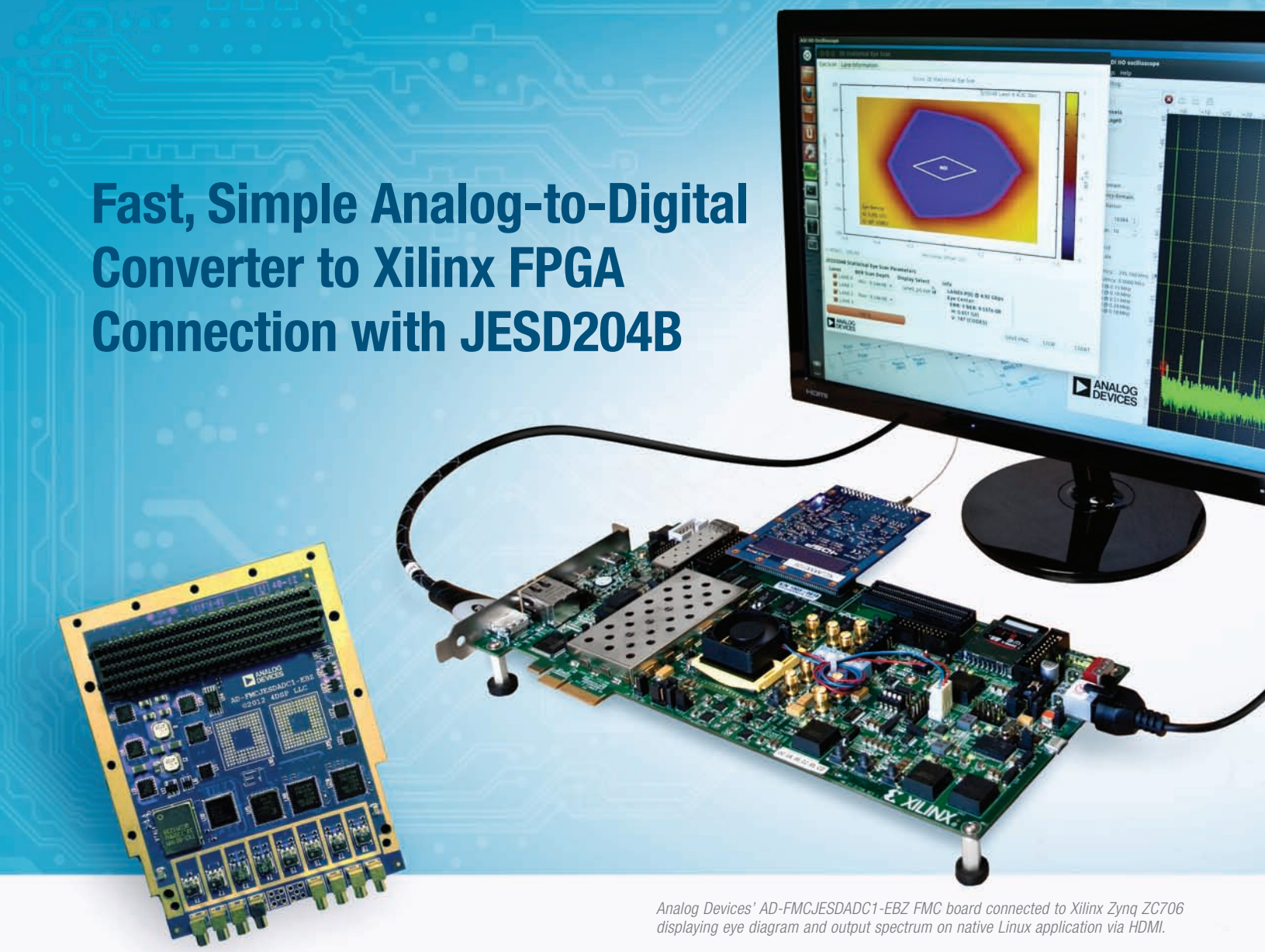
ZYNQ®

www.em.avnet.com/MMP-7Z045-G

The Xilinx® Zynq®-7000 All Programmable SoC Mini-Module Plus designed by Avnet, is a small system-on-module (SOM) that integrates all the necessary functions and interfaces for a Zynq-7000 AP SoC system. For development purposes, the module can be combined with the Mini-Module Plus Baseboard II and a Mini-Module Plus Power Supply to provide an out-of-box development system ready for prototyping. The modular concept also allows the same Zynq-7000 AP SoC to be placed on a user-created baseboard, simplifying the user's baseboard design and reducing the development risk.

DESIGNED BY AVNET

Fast, Simple Analog-to-Digital Converter to Xilinx FPGA Connection with JESD204B

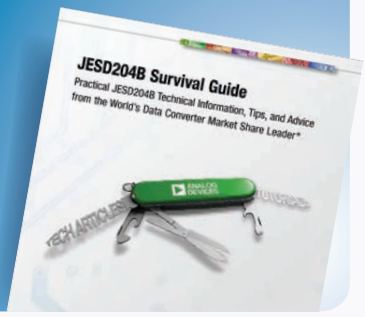


Analog Devices' AD-FMCJESDADC1-EBZ FMC board connected to Xilinx Zynq ZC706 displaying eye diagram and output spectrum on native Linux application via HDMI.

Analog Devices' newest Xilinx FPGA development platform-compatible FPGA mezzanine card (FMC), the AD-FMCJESDADC1-EBZ Rapid Development Board, is a seamless prototyping solution for high performance analog to FPGA conversion.

- Rapidly connect and prototype high speed analog-to-digital conversion to FPGA platforms
- JEDEC JESD204B SerDes (serial/deserializer) technology
- Four 14-bit analog-to-digital conversion channels at 250 MSPS (two AD9250 ADC ICs)
- Free eye diagram analyzer software included in complete downloadable software and documentation package
- HDL and Linux software drivers fully tested and supported on ZC706 and other Xilinx boards.

Everything you need to know about JESD204B in one place. Read the *JESD204B Survival Guide* at *analog.com/JESD204B*



Learn more and purchase at *analog.com/AD9250-FMC-ADC*

ANALOG DEVICES

Xcell journal

PUBLISHER Mike Santarini
mike.santarini@xilinx.com
408-626-5981

EDITOR Jacqueline Damian

ART DIRECTOR Scott Blair

DESIGN/PRODUCTION Teie, Gelwicks & Associates
1-800-493-5551

ADVERTISING SALES Dan Teie
1-800-493-5551
xcelladsales@aol.com

INTERNATIONAL Melissa Zhang, Asia Pacific
melissa.zhang@xilinx.com

Christelle Moraga, Europe/
Middle East/Africa
christelle.moraga@xilinx.com

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/

Xilinx UltraScale Raises Value of All Programmable

If you've been keeping up with developments in the road map for advanced semiconductor process technologies, you're probably aware that the cost of producing silicon is accelerating. Some have asserted that the cost benefit afforded by Moore's Law—namely, that the cost stays the same even though the transistor count doubles—is running out of gas as processes shrink below 28 nanometers. Indeed, for the 20-nm planar and 14/16-nm FinFET generation, industry analysts foresee a tandem trend of rising NRE costs and spiraling price tags for each transistor at each new node.

Analysts say that to make a decent profit on a chip implemented at 20 nm, you will first need to clear an estimated lowball cost of $150 million for designing and manufacturing the device. And to hit that lowball mark, you'd better get it right the first time, because mask costs are approaching $5 million to $8 million per mask.

FinFET looks even more challenging. Analysts predict that the average cost for the 16/14-nm FinFET node will be around $325 million—more than double the cost of 20-nm manufacturing! Why so high? Analysts believe companies will encounter excessive retooling, methodology and IP-requalification costs in switching from planar-transistor processes to FinFETs. Tools and methods will have to deal with new complexities in signal integrity, electromigration, width quantization, and resistance and capacitance. The combination of new transistor, lithography and manufacturing techniques and tools—along with revamped and more expensive IP—represents a perfect storm for design teams.

To say the least, it's hard to justify targeting 16/14-nm FinFET unless the end product will sell in multiple tens of millions of units and have returns counted in billions rather than millions of dollars. As such, eliminating the NRE costs for customers designing on these advanced nodes is a growing sales advantage for all vendors playing in the FPGA space. But if you have been following Xilinx's progress for the last five years, you've seen that the company has clearly demonstrated technical superiority and a unique value proposition. Today's Xilinx® All Programmable devices offer unmatchable system functionality with high capacity and performance, lower power and reduced BOM costs, paired with more flexibility and greater programmability than any other silicon device on the market.

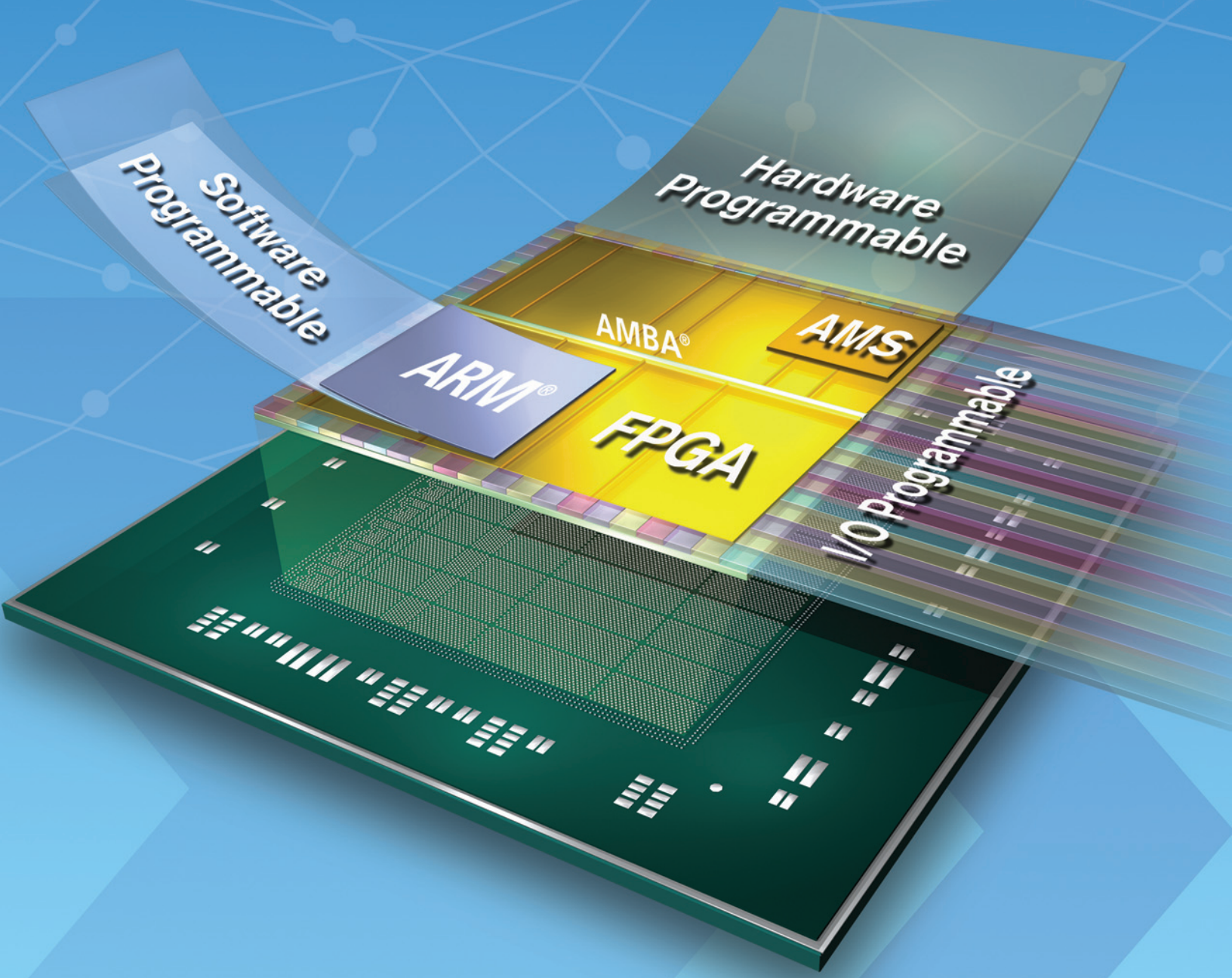
Xilinx executed several breakout moves at the 28-nm node, becoming the first to market with 28-nm FPGAs, 3D ICs and Zynq®-7000 All Programmable SoCs. At the same time, the company introduced the Vivado® Design Suite, incorporating state-of-the-art EDA technologies. There are more breakthroughs in store at the 20-nm and 16-nm FinFET nodes. In July, Xilinx once again became the first merechant chip company to tape out an IC at a new manufacturing node: 20 nm. These new devices will include numerous architectural innovations in Xilinx's new UltraScale™ architecture.

As you will read in the cover story, the UltraScale architecture will deliver many benefits for customers, perhaps the most noteworthy of which is a vast increase in system usage—better than 90 percent. It's worth noting that the power of the UltraScale architecture could not have been fully realized without Vivado. The tool suite has greatly boosted productivity for Xilinx customers using the 28-nm devices. Now, this sophisticated EDA technology is helping Xilinx unleash the benefits of 20-nm and FinFET silicon. Xilinx will deliver that unmatchable advantage to customers—with no NRE charges.

Mike Santarini
Publisher

A Generation Ahead

Hardware, Software and I/O Programmable SoC

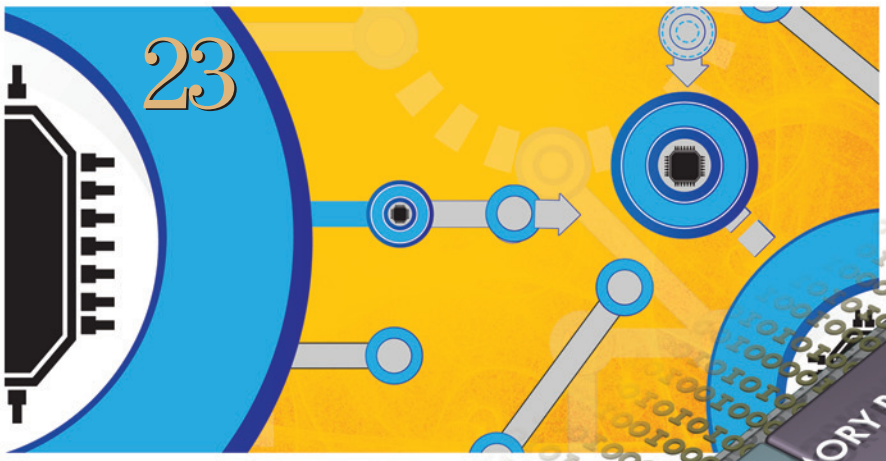


LEARN MORE

XILINX
ALL PROGRAMMABLE™

VIEWPOINTS

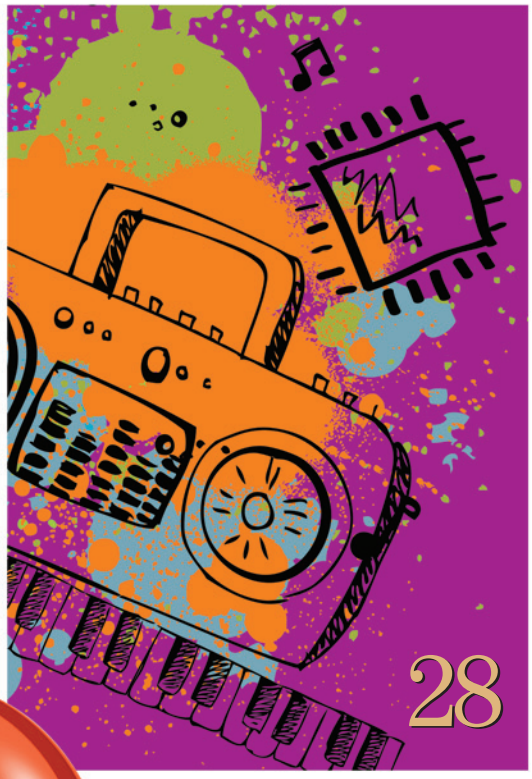
XCELLENCE BY DESIGN APPLICATION FEATURES

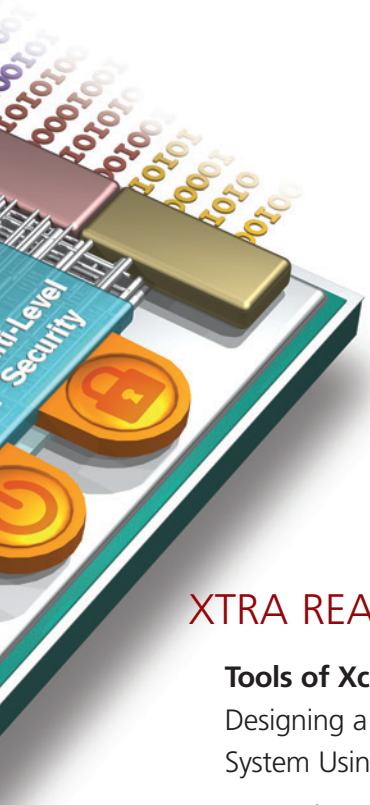


Cover Story

Xilinx 20-nm Planar and 16-nm FinFET Go UltraScale

THE XILINX XPERIENCE FEATURES





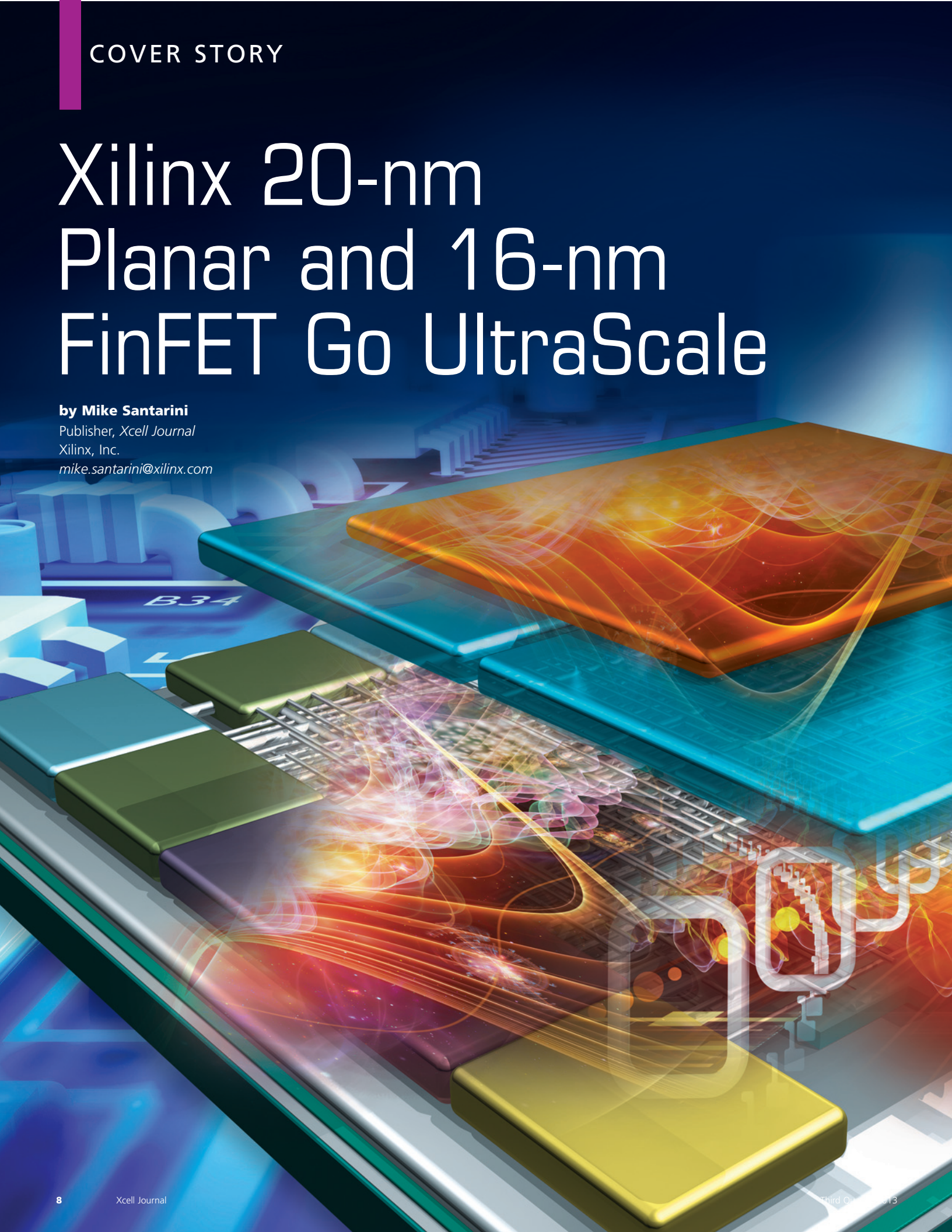
XTRA READING

APEX®
AWARDS FOR PUBLICATION EXCELLENCE
Excellence in Magazine & Journal Writing
2010, 2011

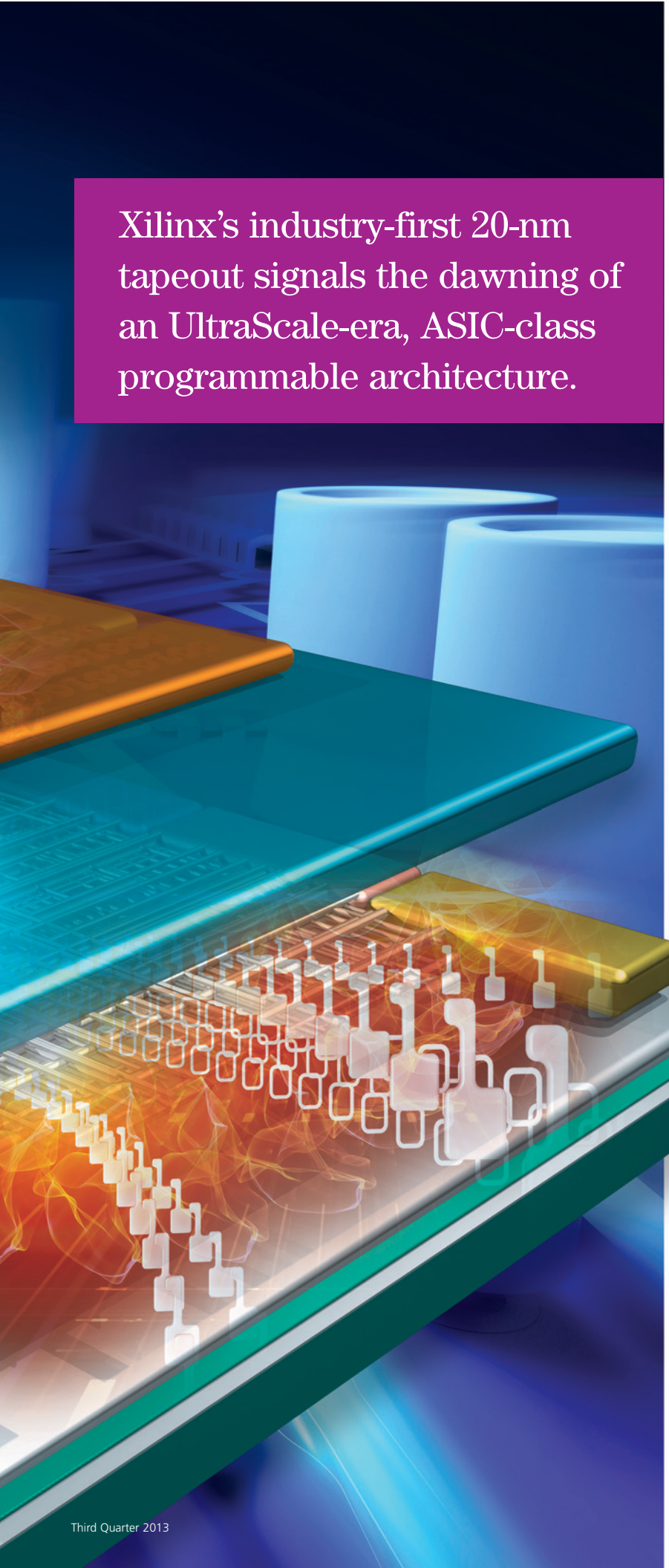
APEX®
AWARDS FOR PUBLICATION EXCELLENCE
Excellence in Magazine & Journal Design and Layout
2010, 2011, 2012

Xilinx 20-nm Planar and 16-nm FinFET Go UltraScale

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



Xilinx's industry-first 20-nm tapeout signals the dawning of an UltraScale-era, ASIC-class programmable architecture.



Building on its breakout move at 28 nanometers, Xilinx® has announced two industry firsts at the 20-nm node. Xilinx is the first merchant chip company to tape out a 20-nm device. What's more, the new device will be the first Xilinx will bring to market using its UltraScale® technology—the programmable industry's first ASIC-class architecture. The UltraScale architecture takes full advantage of the state-of-the-art EDA technologies in the Vivado® Design Suite, enabling customers to quickly create a new generation of All Programmable innovations. The move continues the Generation Ahead advantage over competitors that began at 28 nm, when Xilinx delivered two first-of-their-kind devices: the Zynq®-7000 All Programmable SoC and Virtex®-7 3D ICs.

"Xilinx was the first company to deliver 28-nm devices and at 20 nm, we have maintained the industry's most aggressive tapeout schedule," said Steve Glaser, senior vice president of marketing and corporate strategy at Xilinx. "Our hard work is paying off yet again, and we are clearly a year ahead of our competition in delivering high-end devices and half a year ahead delivering midrange devices."

But as Xilinx did at 28 nm, the company is also adding some industry-first innovations to its new portfolio. Among the first the company is revealing is the new UltraScale architecture, which it will be implementing at 20-nm, 16-nm FinFET and

- Monolithic to 3D IC
- Planar to FinFET
- ASIC-class performance

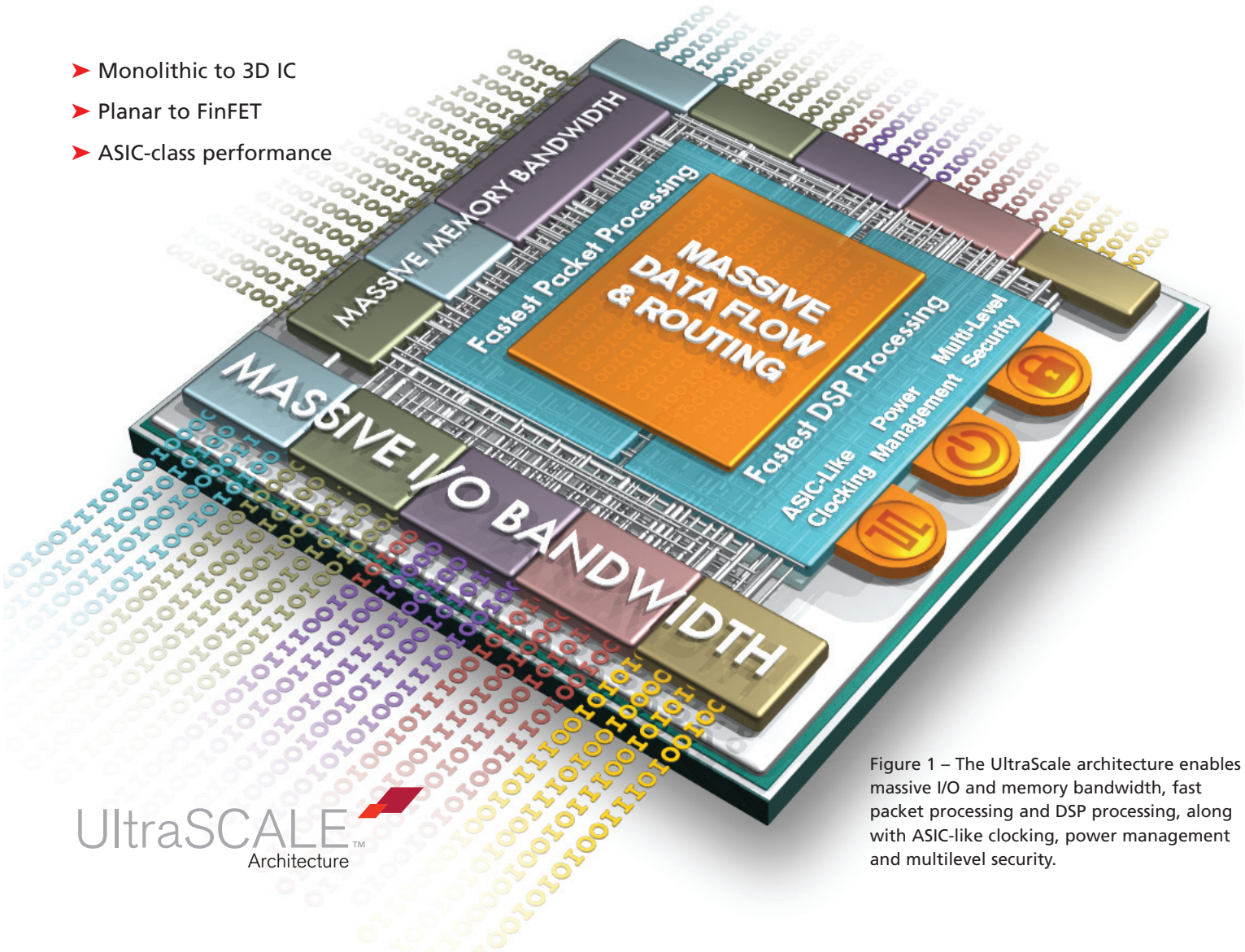


Figure 1 – The UltraScale architecture enables massive I/O and memory bandwidth, fast packet processing and DSP processing, along with ASIC-like clocking, power management and multilevel security.

beyond. "It's the first programmable architecture to enable users to implement ASIC-class designs using All Programmable devices," said Glaser. "The UltraScale architecture enables Xilinx to deliver 20-nm and FinFET 16-nm All Programmable devices with massive I/O and memory bandwidth, the fastest packet processing, the fastest DSP processing, ASIC-like clocking, power management and multilevel security."

AN ARCHITECTURAL ADVANTAGE

The UltraScale architecture includes hundreds of structural enhancements, many of which Xilinx would have been unable to fully implement had it not already developed its Vivado Design Suite, which brings leading-edge EDA tool capabilities to Xilinx customers. For example, Vivado's advanced placement-and-routing capabilities enable users to take full advantage of UltraScale's massive data-processing abilities, so that design teams will be able to achieve better than 90 percent utilization in UltraScale devices while also hitting ambitious performance and power targets. This utilization is far beyond what's possible with competing devices, which today require users to trade off performance for utilization. Customers are thus forced to move to that vendor's bigger and more expensive device, only to find they have to slow down the clock yet again to meet system power goals.

This problem didn't plague Xilinx at the 28-nm process node thanks to Xilinx's routing architecture. Nor will it be an issue at 20 nm, because the UltraScale architecture can implement massive data flow for wide buses to help customers develop systems with multiterabit throughput and with minimal latency. The UltraScale architecture co-optimized with Vivado results in highly optimized critical paths with built-in, high-speed memory cascading to remove bottlenecks in DSP and packet processing. Enhanced DSP subsystems combine critical-path optimization with new 27x18-bit multipliers and dual adders that enable a massive jump in fixed- and IEEE-754 floating-point arithmetic performance and

Typically found only in ASIC-class devices, multiregion clocking allows designers to build high-performance, low-power clock networks with extremely low clock skew in their systems.

efficiency. The wide-memory implementation also applies to UltraScale 3D ICs, which will see a step-function improvement in interdie bandwidth for even higher overall performance.

UltraScale devices feature further I/O and memory-bandwidth enhancements, including support for next-generation memory interfacing that offers a dramatic reduction in latency and optimized I/O performance. The architecture offers multiple hardened, ASIC-class IP cores, including 10/100G Ethernet, Interlaken and PCIe®.

The UltraScale architecture also enables multiregion clocking—a feature typically found only in ASIC-class devices. Multiregion clocking allows designers to build high-performance, low-power clock networks with extremely low clock skew in their systems. The co-optimization of the UltraScale architecture and Vivado also allows design teams to employ a wider variety of power-gating techniques across a wider range of functional elements in Xilinx's 20-nm All Programmable devices to achieve further power savings in their designs.

Last but not least, UltraScale supports advanced approaches to AES bitstream decryption and authentication, key obfuscation and secure device programming to deliver best-in-class system security.

TAILORED FOR KEY MARKETS

Glaser said that UltraScale devices will enable design teams to achieve even greater levels of system integration while maximizing overall system performance, lowering total system power and reducing the overall BOM cost of their systems. "Building on what we did at 28 nm, Xilinx at the 20-nm and FinFET 16/14 nodes is raising the value proposition of our All Programmable technology far beyond the days when FPGAs were considered merely a nice alternative to logic devices," he said. "Our unique system value is abundantly clear across a broad number of applications."

Glaser said devices using Xilinx's 20-nm and FinFET 16-nm UltraScale architecture squarely address a number of growth market segments, such as optical transport networking (OTN), high-performance computing in the network, digital video and wireless communications (see Figure 2). All of these sectors must address increasing requirements for performance, cost, lower power and massive integration.

ULTRASCALE TO SMARTER OTN NETWORKS

The OTN market is currently undergoing a change to smarter networks, as data traffic dictates a massive buildout to create ever-more-advanced wired networking equipment that can move data from 100 Gbps today to 400 Gbps soon and, tomorrow, to 1 Tbps. While the traditional business approach would be to simply throw massive amounts of newer and faster equipment at the problem, today network operators are seeking smarter ways to build out their networks and to improve the networks they have installed while finding better ways to manage costs—all to vastly improve service and profitability.

For example, the data center market is looking to software-defined networking (SDN) as a way to optimize network utilization and cut costs. SDN will allow network managers to leverage the cloud and virtual networks to provide data access to any type of Internet-connected device. The hardware to enable these software-defined networks will have to be extremely flexible, reliable and high performance. Greater pro-

(Continued on page 14)

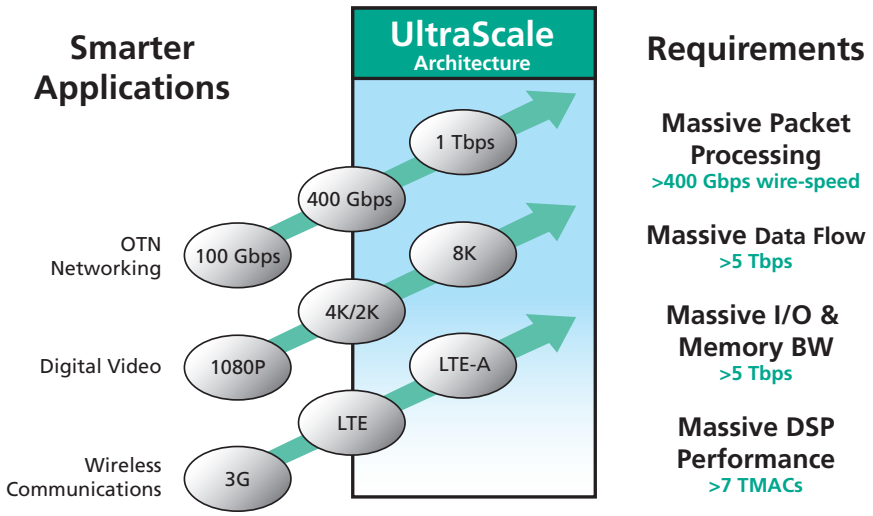


Figure 2 – The UltraScale architecture will speed the development of multiple types of next-generation systems, especially in wired and wireless communications.

THANKS, MR. HOERNI, BUT THE FIN IS IN

Dawning of a new era in transistor technology benefits Xilinx customers.

by Mike Santarini

If you've been keeping up with the latest news in semiconductor process technology, you've likely been reading that the world's most advanced foundries will manufacture devices using new transistor structures called "FinFETs" as the fundamental building blocks for chips they'll produce at what's collectively called the 16/14-nanometer node. But you may be wondering just what FinFETs are, how they differ from standard transistors, and what benefits and challenges they bring.

One of the key inventions in semiconductor manufacturing that has enabled today's $292 billion semiconductor business was Jean Hoerni's creation of the planar process at Fairchild Semiconductor back in the 1950s. The planar process makes it possible to implement ever-smaller transistors, arrange them into a vast array of circuits and interconnect them efficiently on a horizontal plane, rather than having them stick up, as discrete components do on printed-circuit boards. The planar process has enabled the extreme miniaturization of ICs. With the planar process, semiconductors are built in layers on top of, and etched into, ultrapure silicon wafers. Figure 1a shows a planar transistor (essentially an elaborate on/off switch) composed of three main features: source, gate and drain, which have been etched and layered into a silicon substrate. Figure 1b shows a FinFET. Notice that here, the gate surrounds the channel on three sides, whereas in the planar transistor the gate only covers the top of the channel.

The source is where electrons enter the transistor. Depending on the voltage at the gate, the transistor's gate will be either opened or closed in the channel under the gate, similar to the way a light switch can be switched on or off. If the gate allows the signal through the channel, the drain will move the electrons to the next transistor in the circuit. The ideal transistor allows lots of current through when it is "on" and as little current as possible when "off," and can switch between on and off billions of times per second. The speed of the switching is the fundamental parameter that determines the performance of an IC. Chip design companies arrange and group transistors into an array of circuits, which in turn are arranged and grouped into functional blocks (such as processors, memory and logic blocks). Then, these blocks too are arranged and grouped to create the marvelous ICs in the plethora of electronic marvels we benefit from today.

Since the 1960s, the semiconductor industry has introduced a myriad of silicon innovations to keep pace with Moore's Law, which states that the number of transistors in an IC will double every two years. This doubling of transistors means that today's leading-edge ICs contain billions of transistors, crammed into essentially the same die size as 1960s chips but running exponentially faster with exponentially lower power, at 1.2 volts or below. Today, a transistor's features are so small that some are only dozens of atoms wide.

KEEPING UP WITH MOORE

The industry is always struggling to keep pace with Moore's Law in the face of what appear to be true physical limitations to further advancement in semiconductors. Over the last 10 years, process specialists have striven to ensure the electrical integrity of the planar transistor. Simply put, the planar transistor's source, gate and drain features were so small and performance demands so high that these transistors could not adequately control electrons moving through them. The electrons would essentially leak or get pulled through by the drain even if a device was turned off.

For battery-powered devices such as mobile phones, this means the battery would drain charge faster, even if the phone was turned off. For AC-powered devices (those that plug into a wall socket), it means they waste power and run hotter. This heat, if profound and unmanaged by tricks of cooling, can lessen the lifetime of a product, because leakage creates heat and heat increases leakage. Of course, the cooling adds extra cost to products, too. The leakage problem became strikingly noticeable in semiconductors at the 130-nm node. It was in this era of extreme leakage that Microsoft had to recall its Xbox 360 worldwide due to overheating; leakage is one of the reasons why microprocessor companies had to move to less efficient multicore processors over faster but much hotter single-processor architectures.

After 130 nm, several refinements came along to reduce heat and improve planar transistors on all fronts to keep up with Moore's Law. At 90 nm, the industry moved to low-k-dielectric insulators to improve switching, while the 65- and 40-nm nodes introduced further refinements through stressed silicon. At 28 nm, the industry moved to high-k gate dielectrics and metal gates. Xilinx's use of the TSMC 28-nm HPL (high-performance, low-power) process provided the ideal mix of performance and power, giving Xilinx a Generation Ahead advantage over competitors at that geometry.

The planar transistor process will be viable at the 20-nm node thanks to a high-k gate dielectric, metal gates and a

series of more elaborate manufacturing steps such as double patterning. The 20-nm node provides phenomenal benefits in terms of power as well as performance, but the cost is increasing marginally because of elaborate manufacturing to ensure silicon integrity.

Thanks in large part to remarkable research started by Cal Berkeley professor Chenming Hu under a DARPA contract, the 20-nm process will likely be the last hurrah for the planar transistor (at least as we know it today), as the industry moves to FETs built with fins.

INS AND OUTS OF FINS

In a planar transistor of today, electrical current flows from source to drain through a flat, 2D horizontal channel underneath the gate. The gate voltage controls current flow through the channel. As transistor size shrinks with the introduction of each new silicon process, the planar transistor cannot adequately stop the flow of current when it is in an "off" state, which results in leakage and heat. In a FinFET MOSFET transistor, the gate wraps around the channel on three sides, giving the gate much better electrostatic control to stop the current when the transistor is in the "off" state. Superior gate control in turn allows designers to increase the current and switching speed and, thus, the performance of the IC. Because the gate wraps around three sides of the fin-shaped channel, the FinFET is often called a 3D transistor (not to be confused with 3D ICs, like the Virtex-7 2000T, which Xilinx pioneered with its stacked-silicon technology).

In a three-dimensional transistor (see Figure 1b), gate control of the channel is on three sides rather than just one, as in conventional two-dimensional planar transistors (see Figure 1a). Even better channel control can be achieved with a thinner fin, or in the future with a gate-all-around structure where the channel will be enclosed by a gate on all sides.

The industry believes the 16-nm/14-nm FinFET process will enable a 50 percent performance increase at the same power as a device built at 28 nm. Alternatively, the FinFET device will consume 50 percent less power at the same performance. The performance-per-watt benefits added to the continued increases in capacity make FinFET processes extremely promising for devices at 16 or 14 nm and beyond.

That said, the cost and complexity of designing and manufacturing 3D transistors is going to be higher at least for the short term, as EDA companies figure out ways to adequately model the device characteristics of these new processes and augment their tools and flows to account for signal integrity, electromigration, width quantization, resistance and capacitance. This complexity makes designing ASICs and ASSPs even riskier and more expensive than before.

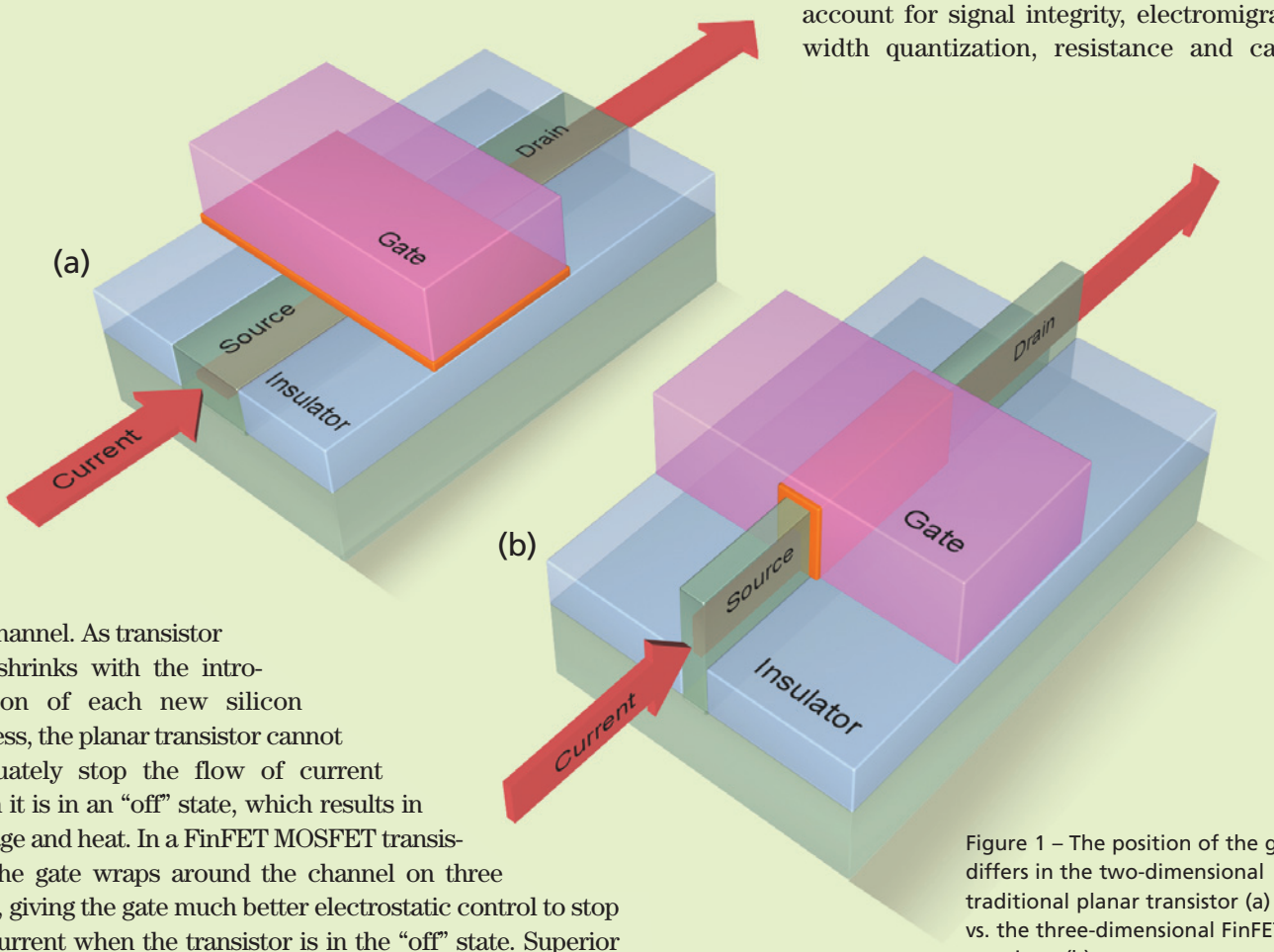


Figure 1 – The position of the gate differs in the two-dimensional traditional planar transistor (a) vs. the three-dimensional FinFET transistor (b).

Xilinx, however, shields users from the manufacturing details. Customers can reap the benefits of increased performance per watt and Xilinx's Generation Ahead design flows to bring innovations based on the new UltraScale architecture to market faster.

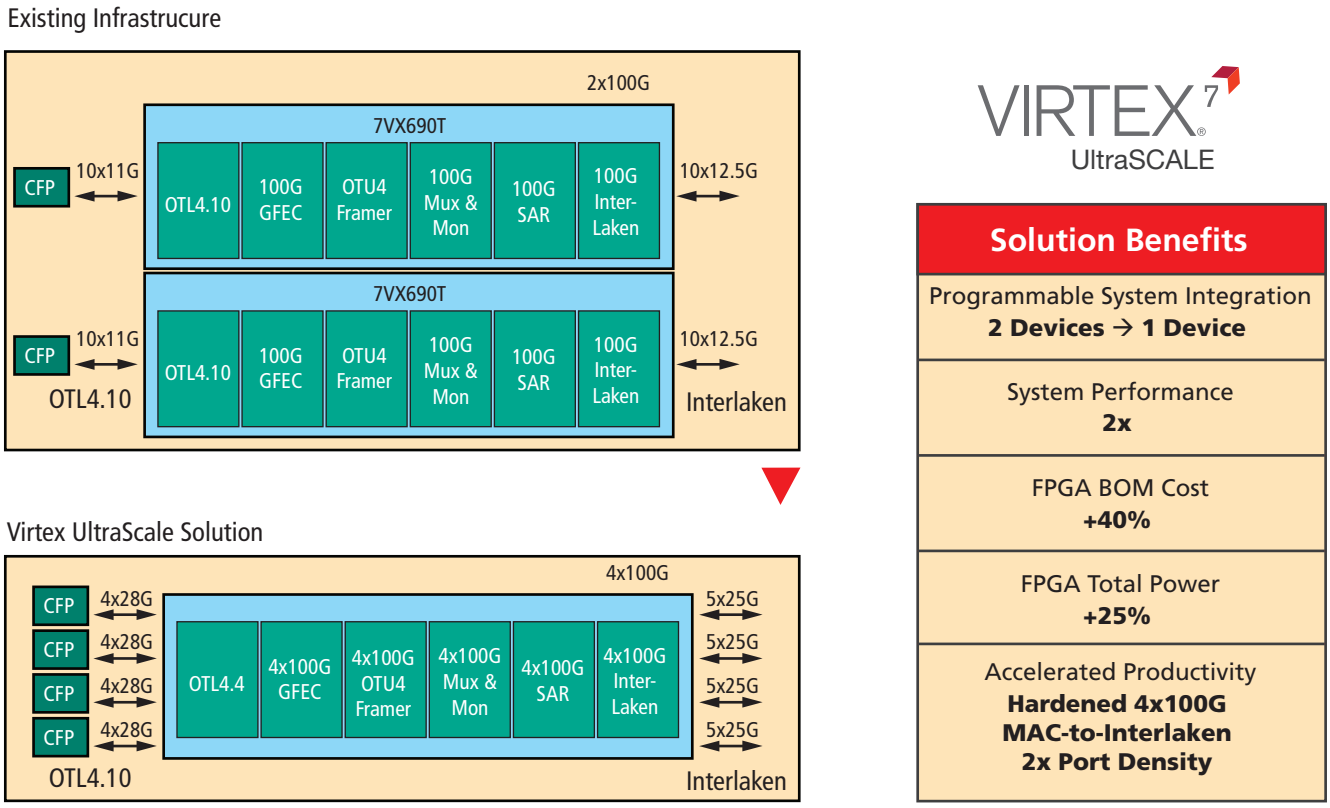


Figure 3 – The UltraScale architecture will enable higher-performance, lower-power single-chip CFP4 modules that will vastly improve the performance of existing OTN equipment with a low-cost card-swap upgrade.

(Continued from page 11)

grammability coupled more tightly with processing will be mandatory.

While network manufacturing companies develop next-generation software-defined networks, their customers are constantly looking to prolong the life of existing equipment. Companies have implemented single-chip CFP2 optical modules in OTN switches with Xilinx 28-nm Virtex-7 F580T 3D ICs to exponentially increase network bandwidth (see cover story, *Xcell Journal* issue 80, *http://issuu.com/xcelljournal/docs/xcell80/8?e=2232228/2002872*). With UltraScale, Xilinx is introducing devices that enable further integration and innovation, allowing companies to create systems capable of driving multiple CFP4 optical modules for yet another exponential gain in data throughput (see Figure 3).

ULTRASCALE TO SMARTER DIGITAL VIDEO AND BROADCAST

Digital video is another market facing a massive buildout. HDTV manufacturers and the supporting broadcast infrastructure are rapidly building TVs and broadcast infrastructure equipment (cameras, communications and production gear) for 1080p, 4k/2k and 8k video. TVs will not only need to conform to these upcoming standards, they must also be feature rich and reasonably priced. The market is extremely competitive, so having devices that can adjust to changing standards quickly and can help manufacturers differentiate their TVs is imperative. And while screen resolution and picture quality will increase greatly with the transition from 1080p to 4k/2k to 8k, TV customers will want ever thinner models. This means new TV designs will require even greater integration while achieving the performance, power and BOM cost requirements the consumer market demands.

While broadcast equipment will need to advance drastically to conform to the requirements for 4k/2k and 8k video, perhaps the biggest customer requirements for these systems will be flexibility and upgradability. Broadcast standards are ever evolving. Meanwhile, the equipment to support these standards is expensive and not easily swapped in and out. As a result, broadcast companies are getting smarter about their buying and are making longevity and upgradability their top requirements. This makes equipment based on All Programmable devices a must-have for all sectors of the next-generation broadcast market, from the camera at the ballgame to the production equipment in the control room to the TV in your living room.

Figure 4 illustrates how a midrange Kintex® UltraScale device will enable professional and "prosumer" camera manufacturers to quickly bring to market new cameras with advanced capabilities for 4k/2k and 8k while dramatically cutting cost, size and weight.

ULTRASCALE TO SMARTER WIRELESS COMMUNICATIONS

Without a doubt, the most rapidly expanding business in electronics is mobile communications. With usage of mobile devices increasing tenfold over the last 10 years and projections for even more dramatic growth in the coming 10, carriers are scrambling to devise ways to woo customers to service plans in the face of increased competition. Wireless coverage and network performance are key differentiators and so carriers are always looking for network equipment that is faster and can carry a vast amount of data quickly. They could, of course, buy more and more higher-performance basestations, but the infrastructure costs of building out and maintaining more equipment cuts into profitability, especially as competitors try to lure customers away by offering more cost-effective plans. Very much like their counterparts in wired communications, wireless carriers are seeking smarter ways to run their business and are looking for innovative network architectures.

One such architecture, called cloud radio-access networks (C-RAN), will enable carriers to pool all the baseband processing resources that would normally be associated with multiple basestations and perform baseband processing "in the cloud." Carriers can automatically shift their resources to areas experiencing high demand, such as those near live sporting events, and then reallocate the processing to other areas as the crowd disperses. The result? Better quality of service for customers and a reduction in capital as well as operating expenditures for carriers, with improved profitability.

Creating viable C-RAN architectures requires highly sophisticated and highly flexible systems-on-chip that integrate high-speed I/O with parallel and serial processing. Xilinx's UltraScale FPGAs and SoCs will play key roles in this buildout.

Xilinx is building UltraScale 20-nm versions of its Kintex and Virtex All Programmable FPGAs, 3D ICs and Zynq All Programmable SoCs. The company is also developing Virtex UltraScale devices in TSMC's "FinFET 16" technology (see sidebar, page 12). For more information on Xilinx's UltraScale architecture, visit Xilinx's UltraScale page and read the backgrounder.

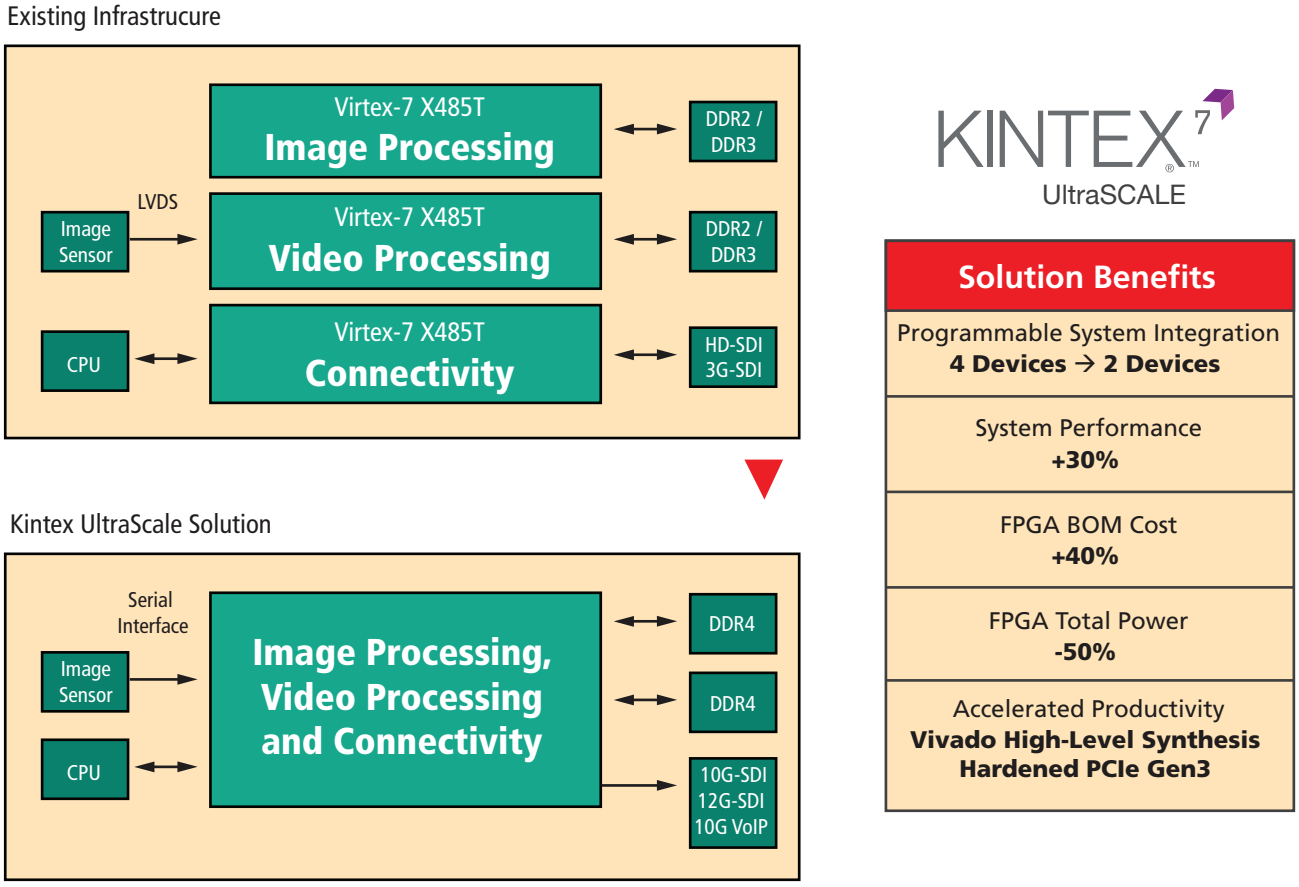


Figure 4 – The UltraScale architecture will let professional and consumer-grade camera manufacturers bring 8k cameras to market sooner, saving power, reducing weight, improving performance and lowering BOM cost.

Efficient Bitcoin Miner System Implemented on Zynq SoC

by Alexander Standridge
MSc Candidate
California State University, Northridge
astandridge@gmail.com

Calvin Ho
MSc Candidate
California State University, Northridge
calvin.ho.67@my.csun.edu

Shahnam Mirzaei
Assistant Professor
California State University, Northridge
shahnam.mirzaei@csun.edu

Ramin Roosta
Professor
California State University, Northridge
ramin.roosta@csun.edu

The integration of programmable logic with a processor subsystem in one device allows for the development of an adaptable and affordable Bitcoin miner.

Bitcoin is a virtual currency that has become increasingly popular over the past few years. As a result, Bitcoin followers have invested some of their assets in support of this currency by either purchasing or "mining" Bitcoins. Mining is the process of implementing mathematical calculations for the Bitcoin Network using computer hardware. In return for their services, Bitcoin miners receive a lump-sum reward, currently 25 Bitcoins, and any included transaction fees. Mining is extremely competitive, since network rewards are divided up according to how much calculation all miners have done.

Bitcoin mining began as a software process on cost-inefficient hardware such as CPUs and GPUs. But as Bitcoin's popularity has grown, the mining process has made dramatic shifts. Early-stage miners had to invest in power-hungry processors in order to obtain decent hash rates, as mining rates are called. Although CPU/GPU mining is very inefficient, it is flexible enough to accommodate changes in the Bitcoin protocol. Over the years, mining operations have slowly gravitated toward dedicated or semi-dedicated ASIC hardware for optimized and efficient hash rates. This shift to hardware has generated greater efficiency in mining, but at the cost of lower flexibility in responding to changes in the mining protocol.

An ASIC is dedicated hardware that is used in a specific application to perform certain specified tasks efficiently. So although ASIC Bitcoin miners are relatively inexpensive and produce excellent hash rates, the trade-off is their inflexibility to protocol changes.

Like an ASIC, an FPGA is also an efficient miner and is relatively inexpensive. In addition, FPGAs are more flexible than ASICs and can adjust to Bitcoin protocol changes. The current problem is to design a completely efficient and fairly flexible mining system that does not rely on a PC host or relay device to connect with the Bitcoin Network. Our team accomplished this task using a Xilinx® Zynq®-7000 All Programmable SoC.

THE BIG PICTURE

To accomplish our goal of implementing a complete mining system, including a working Bitcoin node and a miner that is both efficient and flexible, we needed a powerful FPGA chip not only for flexibility, but also for performance. In addition to the FPGA, we also needed to use a processing engine for efficiency.

At its simplest, the Bitcoin mining process boils down to a SHA-256 process and a comparator. The SHA-256 process manipulates the block header information twice, and then compares it with the Bitcoin Network's expanded target difficulty.

On this complete system-on-chip (SoC), we would need an optimized kernel to run all required Bitcoin activities, including network maintenance and transactions handling. The hardware that satisfied all these conditions was the Zynq-7020 SoC residing on the ZedBoard development board. At around $300 to $400, the ZedBoard is fairly inexpensive for its class (see *http://www.zedboard.org/buy*). The Zynq-7020 SoC chip contains a pair of ARM® Cortex™-A9 processors embedded in 85,000 Artix®-7 FPGA logic cells. The ZedBoard also comes with 512 Mbytes of DDR3 memory that enabled us to run our SoC design faster. Lastly, the ZedBoard has an SD card slot for mass storage. It enabled us to put the entire updated Bitcoin program onto it.

Using the ZedBoard, we implemented our SoC Bitcoin Miner. It includes a host, a relay, a driver and a miner, as seen in Figure 1. We used the non-graphical version of the original Bitcoin client, known as Bitcoind, as a

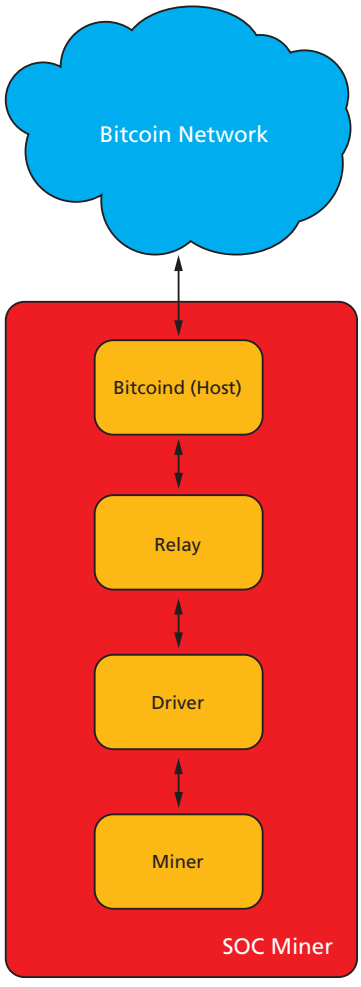


Figure 1 – The basic data flow of the miner showing the relationships between the components and the Bitcoin Network

```
#define ROTR(x,n) ( (x >> n) | (x << (32 - n)))
// Logic operations

#define s0(x) (ROTR(x,7) ^ ROTR(x,18) ^ (x >> 3))
#define s1(x) (ROTR(x,17) ^ ROTR(x,19) ^ (x >> 10))
#define s2(a) (ROTR(a,2) ^ ROTR(a,13) ^ ROTR(a,22))
#define s3(e) (ROTR(e,6) ^ ROTR(e,11) ^ ROTR(e,25))

#define w(a,b,c,d) (s1(d) + c + s0(b) + a)

#define maj(x,y,z) ((x&y)^(x&z)^(y&z))
#define ch(x,y,z) ((x&y)^((~x)&z))

#define t1(e,f,g,h,k,w) (h + s3(e) + ch(e,f,g) + k + w)
#define t2(a,b,c) (s2(a) + maj(a,b,c))

      for( i = 0; i < 64; i++){
// loop 64 times
            if( i > 15) {
                  W[ i] = w(W[ i-16] ,W[ i-15] ,W[ i-7] ,W[ i-
2] );  // Temp values
            }
            unsigned int temp1 = t1(e,f,g,h,K[ i] ,W[ i] );
            unsigned int temp2 = t2(a,b,c);
            h = g;
// Round results
            g = f;
            f = e;
            e = d + temp1;
            d = c;
            c = b;
            b = a;
            a = temp1 + temp2;
      }
```

Figure 2 – Vivado HLS implementation of the SHA-256 process

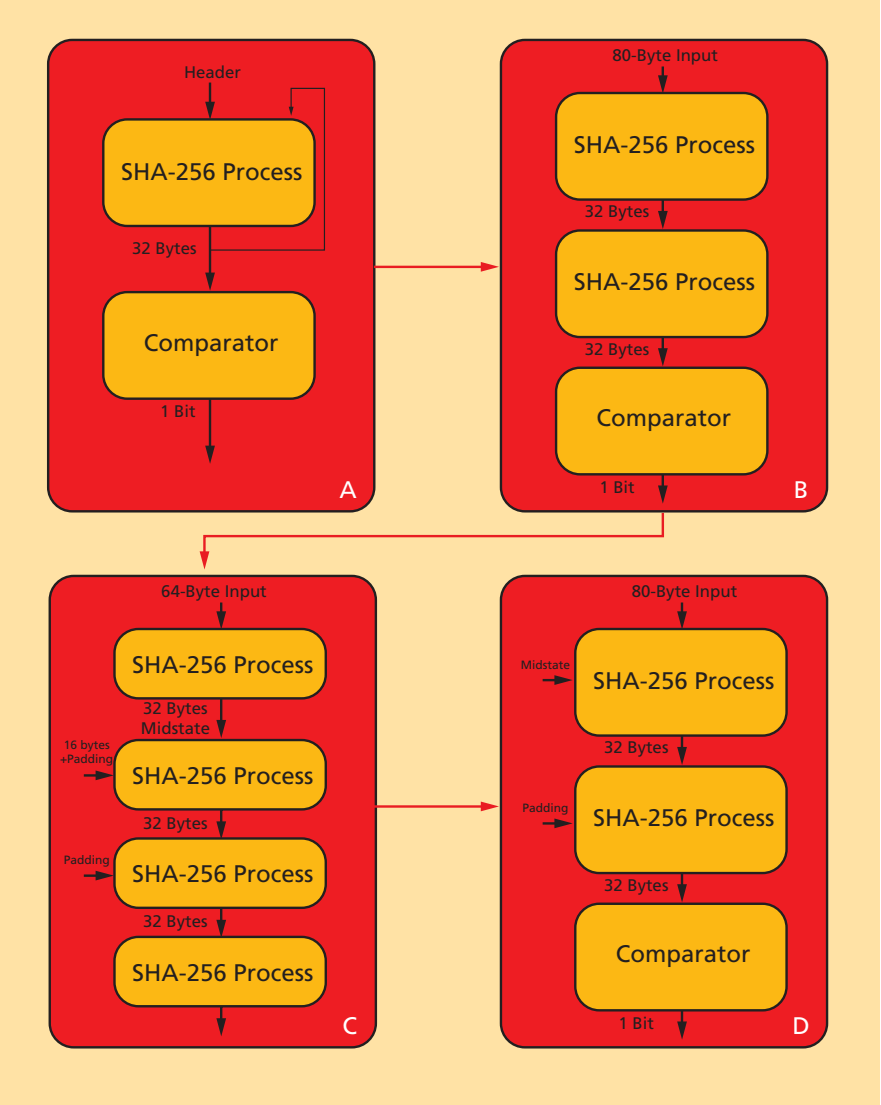


Figure 3 – A (top left) is the simplest form of the core, showing the general flow. B splits the single, universal SHA-256 process module into a pair of dedicated modules for each phase. C breaks down the specific SHA-256 processes into a single generic module that can be repeatedly used. D utilizes a quirk in the block header information allowing the removal of the first generic SHA-256 process.

host to interact with the Bitcoin Network. The relay uses the driver to pass work from the host to the miner.

THE DEPTHS OF THE MINING CORE

We started developing the mining core with Vivado® HLS (high-level synthesis), using the U.S. Department of Commerce's specification for SHA-256. With Vivado HLS' ability of fast behavioral testing, we quickly laid out several prototype mining cores, ranging from a simple single-process to a complex multiprocess system. But before exploring the details of the mining-core structure, it's useful to look at the basics of the SHA-256 process.

The SHA-256 process pushes 64 bytes through 64 iterations of bitwise shifts, additions and exclusive-ors, producing a 32-byte hash. During the process, eight 4-byte registers hold the results of each round, and when the process is done these registers are concatenated to produce the hash. If the input data is less than 63 bytes, it is padded to 63 bytes and the 64th byte is used to store the length of the input data. More commonly, the input data is larger than 63 bytes. In this case, the data is padded out to the nearest multiple of 64, with the last byte reserved for the data length. Each 64-byte chunk is then run through the SHA-256 process one after another, using the output of the previous chunk, known as the midstate, as the basis for the next chunk.

With Vivado HLS, the implementation of the SHA-256 process is a simple "for" loop; an array to hold the constants required by each round; an additional array to hold the temporary values used by later rounds; eight variables to hold the results of each round; and definitions for the logic operations, as seen in Figure 2.

At its simplest, the Bitcoin mining process boils down to a SHA-256 process and a comparator. The SHA-256 process manipulates the block header information twice, and then compares it with the Bitcoin Network's expanded target difficulty, as seen in Figure 3A. This is a simple concept that becomes complicated in hardware, because the first-version Bitcoin block header is 80 bytes long. This means that the initial pass requires two runs of the SHA-256 process, while the second pass requires only a single run, as seen in Figure 3B. This double pass, defining two separate SHA-256 modules, complicates the situation because we want to keep the SHA-256 module as generic as possible to save development time and to allow reuse. This generic requirement on the SHA-256 module defines the inputs and outputs for us, setting them as the 32-byte initial values, 64-byte data and 32-byte hash. Thus, with the core aspect of the SHA-256 process isolated, we can then manipulate these inputs to any layout we desire.

The first of the three prototypes we developed used a single SHA-256 Process module. From the beginning, we knew that this core would be the slowest of the three because of the lack of pipelining; however, we were looking to see how compact we could make the SHA-256 process.

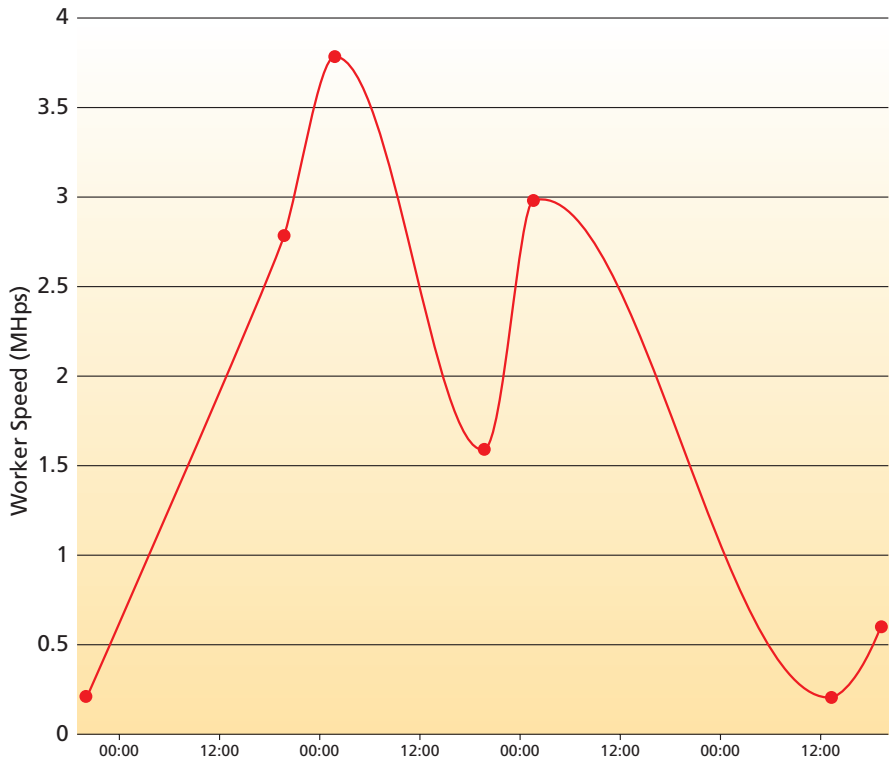


Figure 4 – The hash rate of the mining core over a four-day period of core-optimization testing. The graph is an average over time, giving a general idea of performance of the mining core. The results decreased due to hitting the routing limits of the test platforms.

The second prototype contained three separate SHA-256 Process modules chained together, as seen in Figure 3C. This configuration allowed for static inputs where the first process handles the first 64 bytes of the header information. The second process handles the remaining 16 bytes from the header plus the 48 bytes of required padding. The third process handles the resulting hash from the first two. These three separate processes simplified the control logic and allowed for simple pipelining.

The third and final prototype used two SHA-256 Process modules and took advantage of a quirk in the Bitcoin header information and mining process that the Bitcoin mining community makes use of. The block header information is laid out in such a way that only the first 64 bytes change when the block is modified. This means that once the first 64 bytes are processed, we can save the output data (midstate) and only process the last 16 bytes of the header, for a significant hash rate boost (see Figure 3D).

The comparator is a major component, and properly designed can significantly increase performance. A solution is a hash that has a numeric value less than the 32-byte expanded target difficulty defined by the Bitcoin system. This means we must compare each hash we produce and wait to see if a solution is found. Due to a quirk with the expanded target difficulty, the first 4 bytes will always be zero regardless of what the difficulty is, and as the network's difficulty increases so will the number of leading zeros. At the time of this writing, the first 13 bytes of the expanded target difficulty were zero. Thus, instead of comparing the whole hash to the target difficulty, we check to see if the first x number of bytes are zero. If they are zero, then we compare the hash to the target difficulty. If they are not zero, then we toss the hash out and start over.

With the results of our prototypes in hand, we settled on a version of the third core developed by the Bitcoin open-source community specifically for the Xilinx FPGAs.

ISE DEVELOPMENT

Bitcoin mining is a race to find a number between zero and 2^{32}-1 that gives you a solution. Thus, there are really only two ways to improve mining-core performance: faster processing or divide-and-conquer. Using a Spartan®-3 and a Spartan-6 development board, we tested a series of different frequencies, pipelining techniques and parallelization.

For our frequency tests, we started with the Spartan-3E development board. However, it quickly became apparent that nothing could be done beyond 50 MHz and as such, we left the frequency tests to the Spartan-6.

We tested 50, 100 and 150 MHz on the Spartan-6 development board, which resulted in predictable results, achieving 0.8, 1.6 and 2.4 mega hashes per second (MHps) respectively. We also tried 75 MHz and 125 MHz during the parallelization and pipelining tests, but these frequencies were only a compromise to make the miner fit onto the Spartan-6.

One of the reasons we chose the mining core from the open-source community was that it was designed with a depth variable to control the number of pipelined stages. The depth is an exponential value between 0 and 6, which controls the number of power-two iterations the VHDL "generate" command executes—namely, 2^0 to 2^6. We performed the initial frequency tests with a depth of 0—that is, only a single round implemented for each process.

On the Spartan-3e we tested depths of 0 and 1 before running into timing-constraint issues. The depth of 0 at 50 MHz gave us roughly 0.8 MHps, while the depth of 1 achieved roughly 1.6 MHps.

On the Spartan-6 we were able to reach a depth of 3 before running into

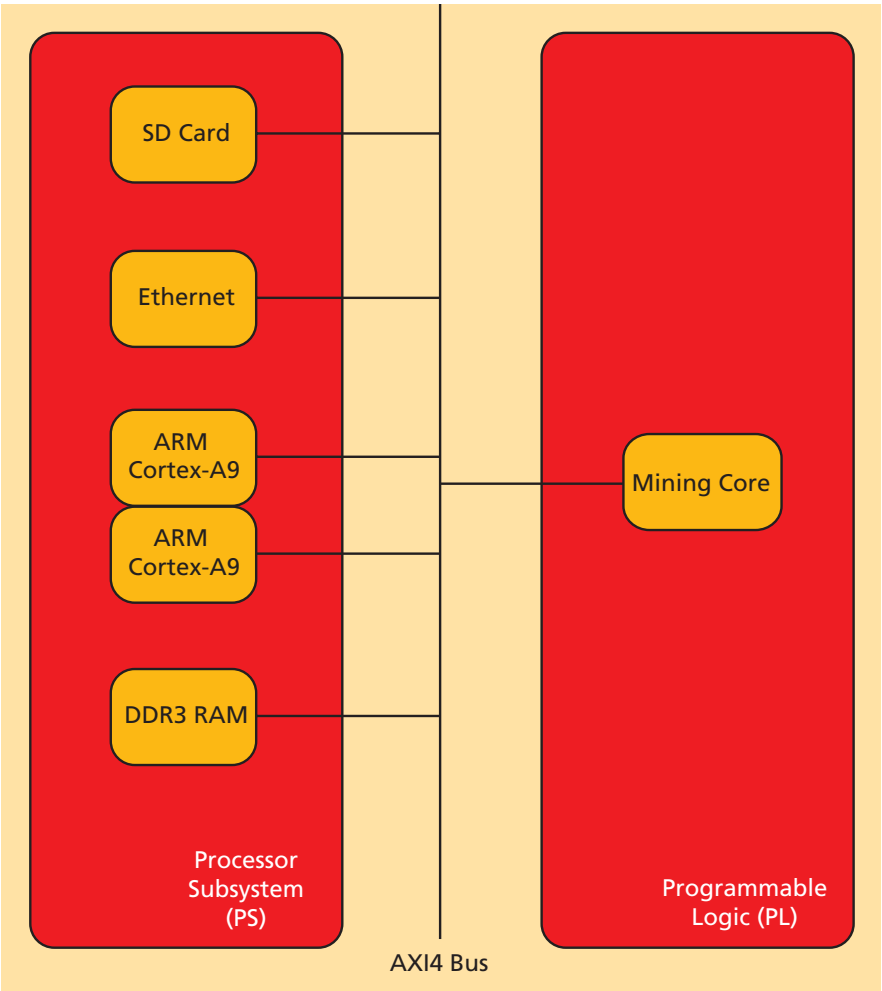


Figure 5 – The relatively new AXI4 interface connects the ZedBoard's hard peripherals with the ARM processors and Artix FPGA logic.

timing-constraint issues. At 50 MHz, the Spartan-6 achieved the same results as the Spartan-3 at a depth of 0 and 1. We noticed an interesting trend at this point. Doubling the frequency had the same effect as increasing the depth of the pipeline. The upper limit was set by the availability of routing resources, with our peak averaging out at roughly 3.8 MHps, as seen in Figure 4.

The SHA-256 Process module has 10 different 32-bit additions. In this implementation, we are attempting to do all of them in a single clock cycle. To shorten the longest path, we tried to divide the adder chains into a series of stages. However, doing so radically changed the control logic for the whole mining core and required a complete rewrite. We dropped these changes to save time and effort.

The final performance improvement we tested was parallelization. With only minor modifications, we doubled and quadrupled the number of SHA-256 sets. A set consists of two SHA-256 Process modules. This modification effectively cut the potential number range for each set into halves, for the doubled number of SHA-256 sets, or fourths for the quadrupled number. To fit these extra SHA processes onto the Spartan-6, we had to reduce the system frequency. Four sets of the SHA-256 Process modules would run at 75 MHz, while two sets would run at 125 MHz. The hash rate improvements were difficult to document. We could easily see the hash rate of an individual SHA-256 set; however, a multiple-set mining core could find a solution faster than the hash rate implied.

USING THE EDK

After FPGA testing, our next step was to tie the mining core into the Zynq SoC's AXI4 bus. Xilinx's Embedded Development Kit (EDK) comes loaded with a configuration utility specifically designed for the Zynq SoC, allowing us to easily configure every aspect. By default, the system has the 512 Mbytes of DDR3, the Ethernet, the USB and the SD interface enabled, which is everything we needed for our Bitcoin SoC.

The twin Cortex-A9 processors feature the AXI4 interface instead of the PLB system used by previous soft-core systems. All of the peripherals are attached to processors through the AXI4 interface, as shown in Figure 5.

The EDK custom peripheral wizard provides code stubs for the different variants of the AXI4 interface and serves as the basis for the development of the mining core's AXI interface. For simplicity, we used the AXI4-Lite interface, which provides simple read and write functionality to the mining core. Ideally, developers would want to use the regular AXI4 interface to take advantage of the advanced interface controls, such as data bursts.

Keeping with the goal of simplicity, we used three memory-mapped registers to handle the mining-core I/O. The first register feeds the host data packet to the miner. This register keeps track of the amount of data passed and locks itself after 11 words have been transferred. This lock is removed automatically when a solution is found. We can remove the lock manually if necessary through the status register.

We defined the second register as the status register, flagging certain bits for the various states the mining core goes through. With the simplicity of our design, we used only three flags: a loaded flag, a running flag and a solution-found flag. The loaded flag is triggered when the mining core receives 11 words, as mentioned before, and is cleared when we write to it. The start/running flag is set by us

to start the mining core, and is cleared by the mining core when a solution is found. The final register is the output register, which is loaded with the solution when one is found.

We tested each stage of the AXI4-Lite interface development before adding the next component. We also tested the AXI4-Lite interface in Xilinx's Software Development Kit using a bare-bones system. The test served two purposes: first, to confirm that the mining core's AXI4-Lite interface was working correctly and second, to ensure that the mining core was receiving the data in the correct endian format.

EMBEDDED LINUX

With the mining core tied into the processors, we moved to software development. Ideally we wanted to build our own firmware, potentially with a Linux core, to maximize performance. For ease of development, we leveraged a full Linux installation. We used Xillinux, developed by Xillybus, an Ubuntu derivative for rapid embedded-system development.

Our first order of business was to compile Bitcoind for the Cortex-A9 architecture. We used the original open-source Bitcoin software to ensure compatibility with the Bitcoin Network.

Testing Bitcoind was a simple matter. We ran the daemon, waited for the sizable block chain to download and then used the line command to start Bitcoind's built-in CPU mining software.

For the Linux driver, you must implement a handful of specific functions and link them to the Linux kernel for correct operation. When the driver is loaded a specific initialization function runs, preparing the system for interaction with the hardware. In this function, we first confirm that the memory address to which the mining core is tied is available for use, then reserve the address. Unlike a bare-bones system, Linux uses a virtual memory address scheme, which means that before we can use the reserved address we have to request an address remap for the mining core. The kernel gives us a virtual address through the remap, which we can then use to communicate to the mining-core registers.

To improve on our current design without jeopardizing the adaptability, we could add more mining cores that connect to the main Bitcoin node. These additions could boost the performance and provide a higher overall hash rate.

Now that we have access to the hardware, the initialization function then runs a simple test on the mining core to confirm that the core is operating properly. If it is, we register the major and minor numbers—which are used to identify the device in user programs—with the kernel. Every function has a counter, and the exit function is the counter to the initialization function. In this function, we undo everything done during initialization—specifically, release the major and minor numbers used by the driver, then release the mapped virtual address.

When we first call the mining core from the relay program, the open function runs. All we do here is to confirm that the self-test run during the initialization function is successful. If the self-test fails, then we throw a system-error message and fail out. The close function is called when the device is released from the relay program. However, since we only check the self-test results in the open function, there is nothing to do in the close function. The read function looks at the data buffer and determines which port the user is reading, and it then gets the data from the mining core and passes it back. The write function determines which register the user is writing to and passes the data to the mining core.

The final component of our system is a small relay program to pass work from Bitcoind to the mining core through the driver, and return results. As a matter of course, the relay program would check to see if Bitcoind was running and that the mining core was ready and operating properly. Since the relay program would normally sit idle for the majority of the time, we will implement a statistics compilation subsystem to gather data and organize it in a log file. Ideally, we would use a Web host interface for configuration, statistical display and device status.

COMPLETE, EFFICIENT MINING SYSTEM

Our solution to an efficient and complete Bitcoin mining system is the Xilinx Zynq-7000 All Programmable SoC on the ZedBoard. This board is flexible to changes in the Bitcoin protocol and also offers a high-performance FPGA solution along with SoC capability. To improve on our current design without jeopardizing the adaptability, we could add more mining cores that connect to the main Bitcoin node. These additions could boost the performance significantly and provide a higher overall hash rate.

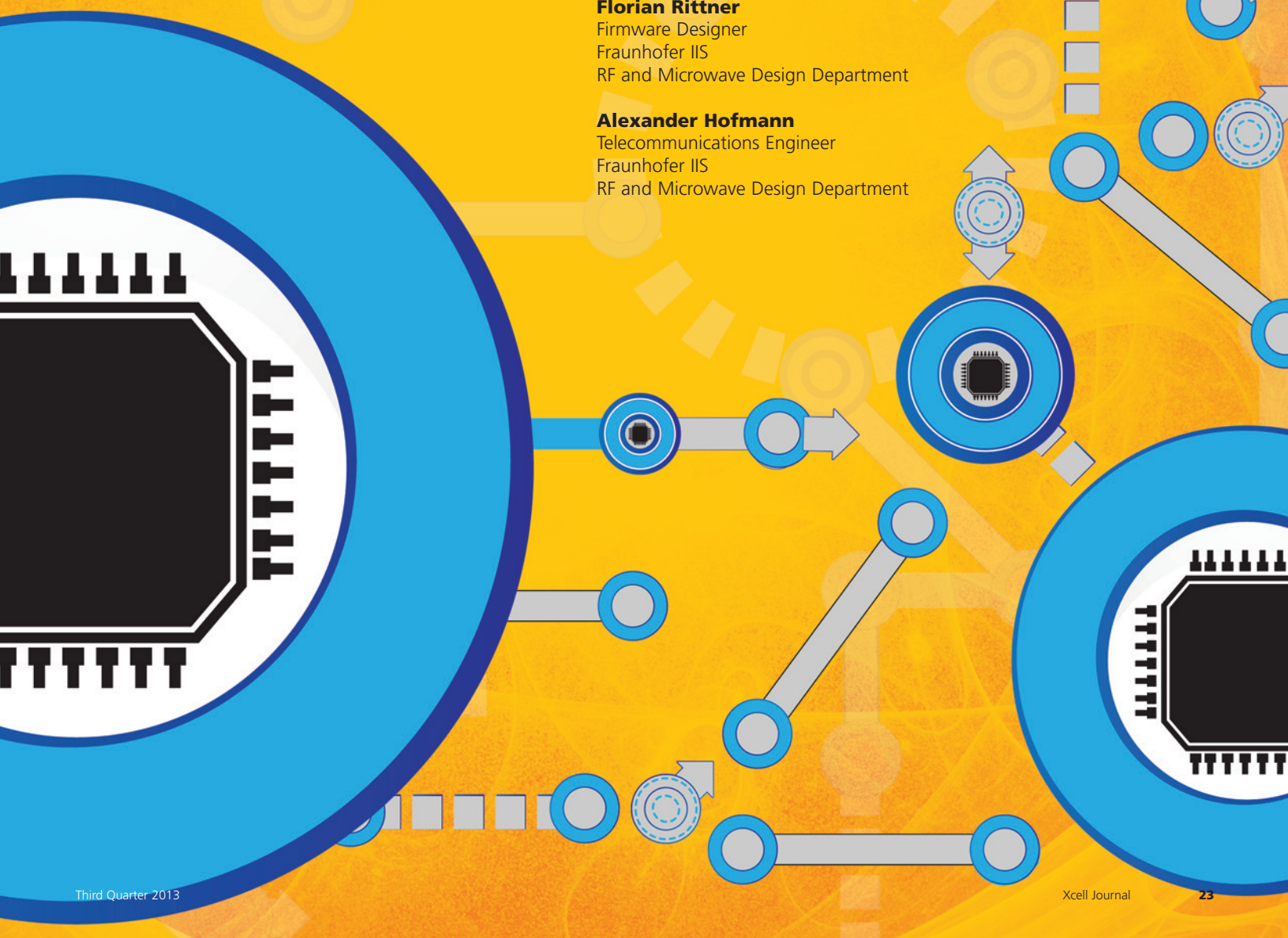
Another improvement that could further optimize this design is to have dedicated firmware. Our current design is running Ubuntu Linux 12.04, which has many unnecessary processes running concurrently with the Bitcoin program, like SSH. It is a waste of the board's resources to run these processes while Bitcoin is running. In a future build, we would eliminate these processes by running our own custom firmware dedicated to only Bitcoin tasks.

Ensuring FPGA Reconfiguration in Space

by Robért Glein
System Engineer
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU), Information
Technology (Communication Electronics)
robert.glein@fau.de

Florian Rittner
Firmware Designer
Fraunhofer IIS
RF and Microwave Design Department

Alexander Hofmann
Telecommunications Engineer
Fraunhofer IIS
RF and Microwave Design Department



A fail-safe method of reconfiguring rad-hard Xilinx FPGAs improves reliability in remote space applications.

In 2007, the German Federal Ministry of Economics and Technology commissioned the German Aerospace Center to perform a feasibility study of an experimental satellite to explore communications methods. Government as well as academic researchers will use the satellite, called the Heinrich Hertz communications satellite, to perform experiments in the Ka and K band. It is scheduled for launch in 2017. A key element of this system is the Fraunhofer-designed onboard processor, which we implemented on Xilinx® Virtex®-5QV FPGAs.

The Fraunhofer onboard processor—which is involved in the in-orbit verification of modules in space—is the essential module of the satellite's regenerative transponder. Because the Heinrich Hertz satellite will be used for a broad number of communications experiments, we chose to leverage the FPGAs' capability to be reconfigured for various tasks. This saves space, reduces weight and enables the use of future communications protocols. Figure 1 illustrates this onboard processor and other parts of the scientific in-orbit verification payload of the satellite.

But because space is perhaps the harshest environment for electronics, we had to take a few key steps to ensure the system could be reconfigured reliably. The radiation-hardened Xilinx FPGAs offer a perfect solution for reconfigurable space applications regarding total ionizing dose, single-event effects, vibration and thermal cycles. The reconfiguration of the digital hardware of these FPGAs is an essential function in this geostationary satellite mission in order to ensure system flexibility. To achieve a robust reconfiguration without a single point of failure, we devised a two-pronged methodology that employs two configuration methods in parallel. Besides a state-of-the-art (SOTA) method, we also introduced a fail-safe reconfiguration method via partial reconfiguration, which enables a self-reconfiguration of the FPGAs.

A straightforward way to configure FPGAs is to use an external radiation-hardened configuration processor with a rad-hard mass memory (for example, flash). In this case, the (partial) bit file is stored in the memory and the processor configures the FPGA with the bitstream. This approach offers good flexibility if your design requires a full FPGA reconfiguration and has many bit files onboard. You can update or verify bit files via a low-data-rate telemetry/telecommand channel or a digital communication up- or downlink with a higher data rate. This SOTA configuration method is nonredundant. Let us assume a failure of the processor or of the configuration interface, as shown in Figure 2. Now it is impossible to reconfigure the FPGA and all Virtex-5QV devices are useless.

OVERALL CONFIGURATION CONCEPT FOR THE FPGA

Figure 2 illustrates both configuration methods for one FPGA—the SOTA method and the fail-safe configuration (also the initial configuration) method, which only needs one radiation-hardened nonvolatile memory (PROM or magnetoresistive RAM) as an external component. This component stores the initial firmware, recognizable as content of the FPGA. We use these two configuration methods in parallel in order to benefit from the advantages of both. With this approach we also increase the reconfiguration reliability and overcome the single-point-of-failure problem.

DETAILS OF FAIL-SAFE CONFIGURATION METHOD

We designed this fail-safe initial configuration method in order to enable self-reconfiguration of the FPGA. The FPGA configures itself with the initial configuration at startup of the onboard processor. We use the serial master configuration protocol for the PROM or the Byte Peripheral Interface for the magnetoresistive RAM (MRAM). However, only one nonvolatile memory per FPGA is required for this fail-safe configuration method.

We separated the firmware of the initial configuration in dynamic (blue) and static (light-red and red) areas as shown in Figure 2. With partial reconfiguration (PR), it is possible to reconfigure dynamic areas while static and other dynamic areas are still running.

After a power-up sequence, the FPGA automatically configures itself with the initial firmware and is ready to receive new partial or even complete (only by MRAM) bit files. A transmitter on Earth packs these bit files into a data stream, encrypts, encodes and modulates it, and sends it over the uplink to the satellite. After signal conditioning and digitizing, the first reconfigurable partition (RP0) inside the FPGA receives the digital data stream. RP0 performs a digital demodulation and decoding. The Block RAM (BRAM) shares the packed bit file with the MicroBlaze™ system. The MicroBlaze unpacks the bit file and stores it in the

SRAM or SDRAM. The MicroBlaze reconfigures a dynamic reconfigurable partition (RP) or all RPs with a direct-memory access (DMA) via the Internal Configuration Access Port (ICAP). The static part of the FPGA should not be reconfigured.

In case of an overwritten RP0, a high-priority command over the satellite control bus could reboot the onboard processor. This command would restore the initial fail-safe configuration.

There are plenty of new possible applications for reconfigurable partitions, including digital transmitters, digital receivers and replacement of a highly reliable communications protocol, among others. You can also reconfigure and switch to new submodules like demappers or decoders for adaptive coding and modulation.

You can even replace the whole fail-safe configuration bit file with a new double-checked initial bit file via MRAM as storage for the initial configuration. With a digital transmitter, the onboard processor can also send the configuration data back to Earth in order to verify bit files on the ground.

In case of an overwrite, a high-priority command over the satellite control bus could reboot the onboard processor and restore the initial fail-safe configuration.

orbit mission with a term of 15 years, reliability of the most important procedures such as configuration depends on the failure-in-time (FIT) rates of both configuration methods. We assume a typical FIT rate of 500 in a representative configuration processor and its peripherals for the SOTA configuration method. The PROM achieves a much better FIT rate of 61 (part stress method for failure-rate prediction at operating conditions regarding MIL-HDBK 217).

Nevertheless, both methods are single points of failure. Only the parallel operation guarantees a significant increase in the reliability of the configuration procedure. Calculating the standalone reliability of the SOTA configuration method results in Rsc = 0.9364 and of the initial method in Ric = 0.9920, with the given mission parameters. The overall configuration reliability for systems in parallel is Rc = 0.9929.

We have to compare Rc and Rsc in order to obtain the gain of the reconfiguration reliability. The achieved gain (Rc - Rsc) is 5.65 percent. The initial configuration has some restrictions in terms of FPGA resources. Nevertheless, you can still reconfigure the whole FPGA anytime with the more complex SOTA configuration method if the external configuration processor is still operating.

Both methods can correct failures of the FPGA caused by configuration single-event upsets. For the SOTA configuration method, the external configuration processor performs a scrubbing, as does the self-scrubbed MicroBlaze for the initial configuration method.

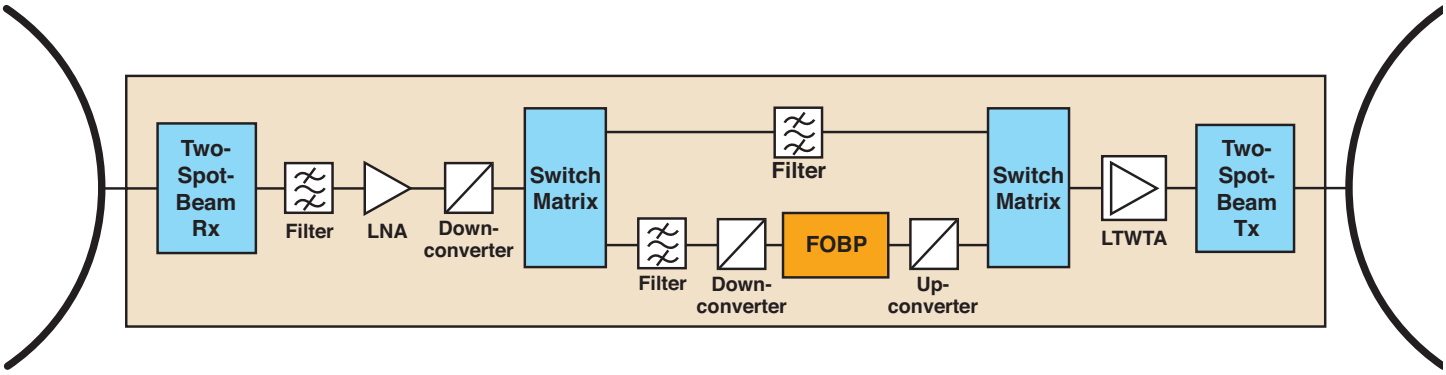


Figure 1 – Part of the scientific in-orbit verification payload of the Heinrich Hertz satellite; the gold box in the middle, labeled FOBP, is the Fraunhofer onboard processor. LTWTA stands for linear traveling-wave tube amplifier.

CONFIGURATION RELIABILITY

A major benefit of using both configuration methods (SOTA and initial) is an increase of the overall system reliability. There is a drawback in the initial fail-safe method regarding the usable FPGA resources. In a geosynchronous

IMPLEMENTING FAIL-SAFE CONFIGURATION

Figure 2 illustrates the detailed architecture of one of the onboard processor's FPGAs. The FPGA is divided into three parts: MicroBlaze, static intellectual-property (IP) cores and RPs. An issue of the MicroBlaze system is memory management. Alongside the PROM or MRAM, we attached an SRAM and an SDRAM. These two different memory types enable applications like temporary packet buffers and content storage (for example, video files). The memory multiplexer block (Mem Mux) enables higher flexibility. It multiplexes the memory to RPs or to the MicroBlaze region. Semaphores control the multiplexer via a general-purpose input/output (GPIO) connection. So, only one region can use the single-port memory at any given time. The initial configuration also contains internal memory with BRAM, presenting an opportunity for a fast data exchange of both regions.

The build wrapper in Xilinx Platform Studio (XPS) enables a universal integration of the MicroBlaze system with support and connection of independent ports.

We implemented IP cores such as BRAM, RocketIO™ and digital clock manager (DCM). The RP switching block connects the RPs by means of settings in the RP switching configuration, which flexibly interconnects the RPs. A data flow control (DFC) manages the data flow among the RPs. The FPGAs communicate to each other by means of RocketIO and low-voltage differential signaling (LVDS).

We divided the RPs into two partitions with 20 percent configurable logic block (CLB) resources of the FPGA and two partitions with 10 percent CLB resources. We also tried to share the DSP48 slices and BRAM equally and maximally to all partitions.

The main part of the implementation is the system design and floorplanning, which we realized with the PlanAhead tool from Xilinx. We combined the MicroBlaze system, the IP cores, the software design and the partial user hardware description language (HDL) modules, called reconfigurable modules. We also defined the location of RPs, planned design runs, chose a synthesis strategy and constrained the design in this step. A BRAM memory map file defines the BRAM resources for data and instruction memory. This enables an integration of the initial MicroBlaze software in the bit file.

Figure 3 shows the implementation result. Currently we are developing a

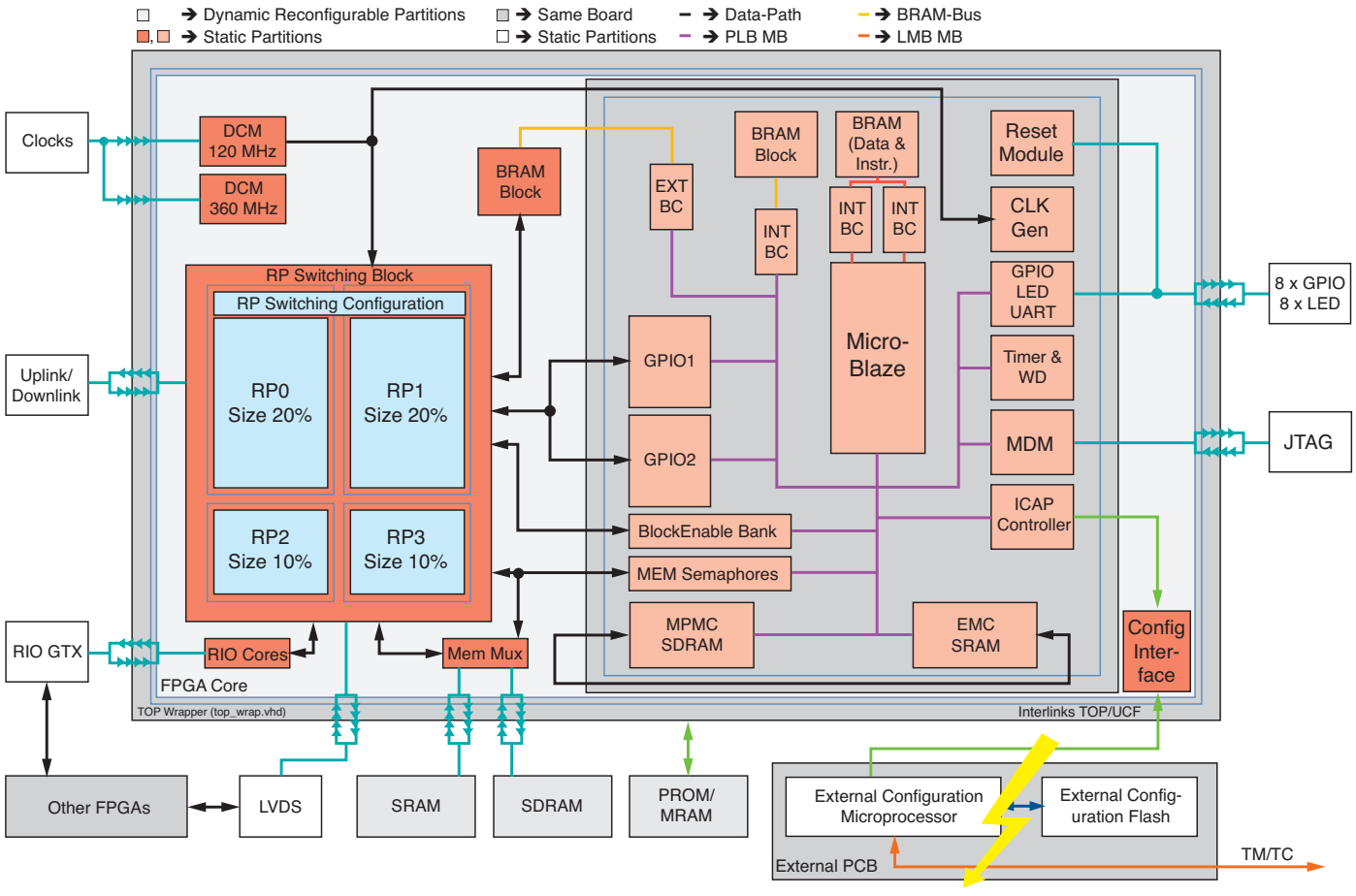


Figure 2 – Architecture of the first FPGA in the initial configuration system. The static sections are in red and light red, and the dynamic portions in light blue (left). The yellow lightning bolt represents a SOTA configuration failure.

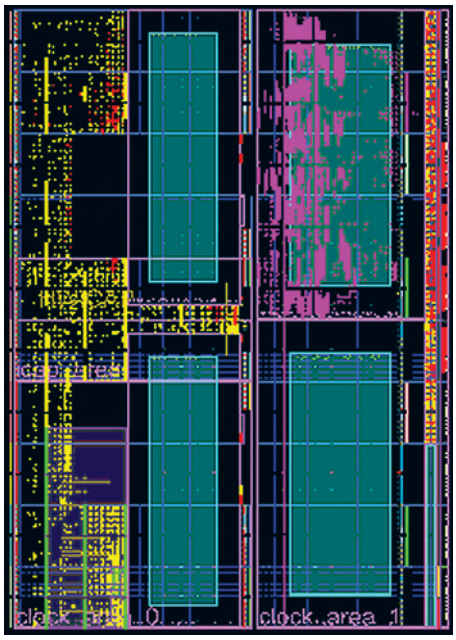


Figure 3 – PlanAhead view of the implemented design. The blue rectangles represent the RP regions. RP0 is seen as purple, the MicroBlaze as yellow, BRAM as green and other static regions as red.

highly reliable receiver for the first partition (RP0), which will be used in the initial configuration (on the satellite) to receive (demodulating, demapping, deinterleaving and decoding) the configuration data.

Table 1 illustrates target and actual sizes of the RPs. We achieved a slightly higher CLB count than we targeted, whereby the DSP slices nearly reach the maximum. The BRAM resources correspond to the CLBs because of the MicroBlaze's demand for BRAMs. The overhead of the static area is approximately 30 percent. We designed this firmware for the commercial Virtex-5 FX130 in order to prove the concept with an elegant breadboard. At this point, we are about to port this design to the Virtex-5QV and implement further features, such as triple-modular redundancy or error-correction codes.

PROBLEM SOLVED

A single point of failure regarding FPGA configuration is a problem in highly reliable applications. You can overcome this issue with an initial configuration, which needs only one additional device per FPGA. Partial reconfiguration enables the remote FPGA configuration via this fail-safe initial configuration. The parallel use of both configurations combines their main advantages: the increase of 5.65 percent in configuration reliability and the flexibility to reconfigure the complete FPGA.

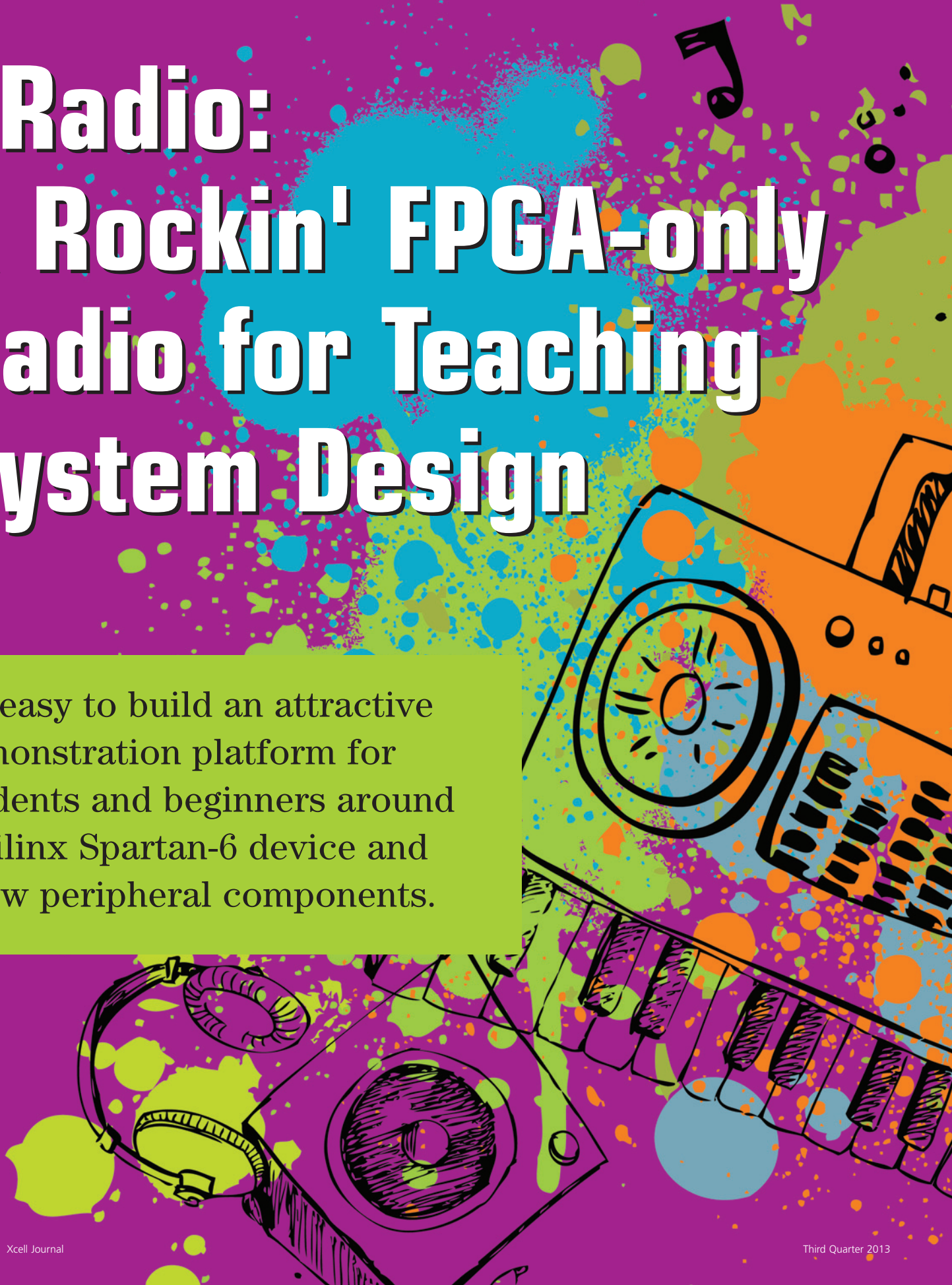
System design and implementation challenges require a trade-off between complexity regarding the routing or clock resources and the usable design resources. An elegant breadboard shows the successful implementation and verification in a real-world scenario.

	CLBs		DSP Slices		BRAM	
	Target	Result	Target	Result	Target	Result
RP0/RP1	4,096	4,400	max.	80	max.	60
	20%	≈ 21%	max.	≈ 25%	max.	≈ 20%
RP2/RP3	2,048	2,720	max.	76	max.	38
	10%	≈ 13%	max.	≈ 24%	max.	≈ 13%
sum of all RP resources	12,288	14,240	max.	312	max.	196
	60%	≈ 70%	max.	≈ 98%	max.	≈ 66%

Table 1 – FPGA resources of the reconfigurable partitions (RPx)

XRadio: A Rockin' FPGA-only Radio for Teaching System Design

It's easy to build an attractive demonstration platform for students and beginners around a Xilinx Spartan-6 device and a few peripheral components.



by Qiongzhi Wu
Lecturer
Beijing Institute of Technology
wqz_bit@bit.edu.cn

Xingran Yang
MS Student
Beijing Institute of Technology
2120120784@bit.edu.cn

R

Recently here at the Beijing Institute of Technology, we set out to develop a digital design educational platform that demonstrated the practical use of FPGAs in communications and signal processing. The platform needed to be intuitive and easy to use, to help students understand all aspects of digital design. It also had to be easy for students to customize for their own unique system designs.

Around that time, we in the electrical engineering department had been having a lively debate as to whether it was possible to use an FPGA's I/O pins as a comparator or a 1-bit A/D converter directly. We decided to test that premise by trying to incorporate this FPGA-based comparator in the design of our XRadio platform, an entirely digital FM radio receiver that we designed using a low-cost Xilinx® Spartan®-6 FPGA and a few general peripheral components. We demonstrated the working radio to Xilinx University Program (XUP) director Patrick Lysaght, during his visit last year to the Beijing Institute of Technology. He was excited by the simplicity of the design, and encouraged us to share the story of the XRadio with the academic community worldwide.

SYSTEM ARCHITECTURE

We built the XRadio platform almost entirely in the FPGA and omitted the use of traditional analog components such as amplifiers or discrete filters, as shown in Figure 1. As a first step, we created a basic antenna by linking a simple coupling circuit with a wire connected to the FPGA's I/O pins. This antenna transmits RF signals to the FPGA, which implements the signal processing of the FM receiver as digital downconversion and frequency demodulation. We then output the audio signal through the I/O pins to the headphones. We added mechanical rotary incremental encoders to control the XRadio's tuning and volume. We designed the system so that the tuning and volume information is displayed on the seven-segment LED module.

Figure 2 shows a top-level logic diagram of the FPGA. In this design, the RF signals coupled into the input buffer of the FPGA are quantified as a 1-bit digital signal. The quantified signal is then multiplied by the local oscillation

Figure 1 – The XRadio hardware concept

signal generated by a numerically controlled oscillator (NCO). The multiplied signal is filtered to obtain an orthogonal IQ (in-phase plus quadrature phase) baseband signal. It then obtains the audio data stream from the IQ signal through the frequency demodulator and low-pass filter. The data stream next goes to the pulse-width modulation (PWM) module, which generates pulses on its output. By filtering the pulses, we obtain a proportional analog value to drive the headphones. We connected a controller module to the mechanical rotary incremental encoders and LEDs. This module gets a pulse signal from incremental encoders to adjust the output frequency of the NCO and audio volume controlled by the PWM module.

IMPLEMENTATION DETAILS

The first challenge we had to overcome was how to couple the signal from the antenna to the FPGA. In our first experimental design, we configured an FPGA I/O as a standard single-ended I/O; then we used resistors R1 and R2 to build a voltage divider to generate a bias voltage between VIH and VIL at the FPGA pin. The signal that the antenna receives can drive the input buffer through the coupling capacitor C1. The signal is sampled by two levels of D flip-flops, which are driven by an internal 240-MHz clock. The output of the flip-flops obtains an equal interval 1-bit sampling data stream.

To test this circuit, we fed the result to another FPGA pin and measured it with a spectrum analyzer to see if the FPGA received the signal accurately. However, it didn't work well because the Spartan-6 FPGA's input buffer has a small, 120-millivolt hysteresis, according to the analyzer. Although the hysteresis is good for

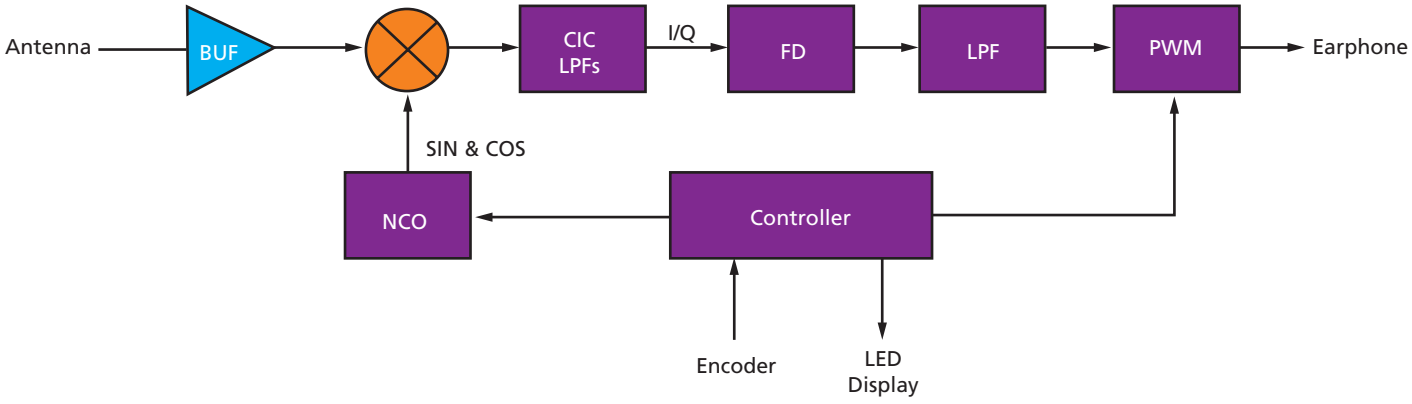


Figure 2 – Top-level diagram of the XRadio FPGA

The design slashes the signal-to-noise ratio because of the quantizing noise generated by 1-bit sampling. But four channels are still distinguishable against the noise floor. Thus, we were able to prove our theory that an FPGA's I/O pins can be used as a comparator or as a 1-bit A/D converter.

preventing noise generally, in this application it wasn't wanted. We had to devise a way to increase the strength of the signal.

To solve this problem, we found that the differential input buffer primitive, IBUFDS, has a very high sensitivity between its positive and negative terminals. According to our test, a differential voltage as low as 1 mV peak-to-peak is enough to make IBUFDS swing between 0 and 1. Figure 3 shows the resulting input circuit. In this implementation, the resistors R1, R2 and R3 generate a common voltage in both the P and N terminals of IBUFDS. The received signals feed into the P sidefile:///app/ds/terminal through coupling capacitor C1. The AC signal is filtered out by the capacitor C2 on the N side, which acts as an AC reference. With this circuit, the FPGA converts the FM broadcast signals into a 1-bit data stream successfully.

The radio picks up about seven FM channels with different amplitudes—including 103.9-MHz Beijing Traffic Radio. The design significantly reduces the signal-to-noise ratio because of the quantizing noise generated by 1-bit sampling. But there are four channels that are still distinguishable against the noise floor. Therefore, we were able to prove our theory that an FPGA's I/O pins can be used effectively as a comparator or a 1-bit A/D converter in our XRadio.

For digital downconversion, we used the DDS Compiler 4.0 IP core to build a numerically controlled oscillator with a 240-MHz system clock. The sine and cosine output frequency ranged from 87 MHz to 108 MHz. The local-oscillator signals that the NCO generates are multiplied with the 1-bit sampling stream and travel through a

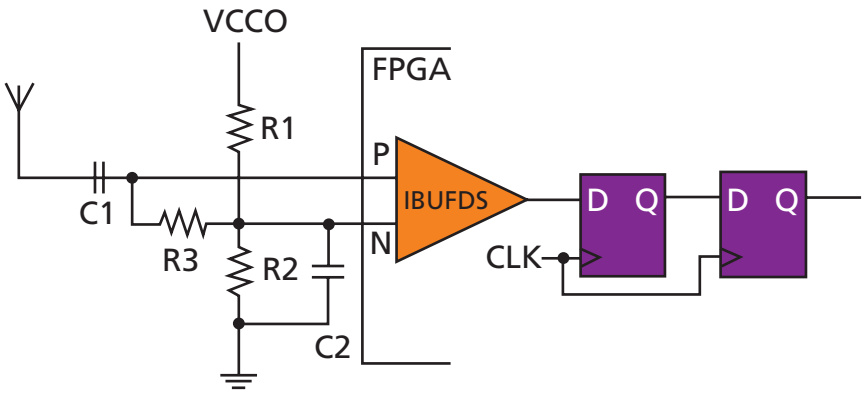


Figure 3 – The antenna input method, making use of the differential input buffer primitive (IBUFDS)

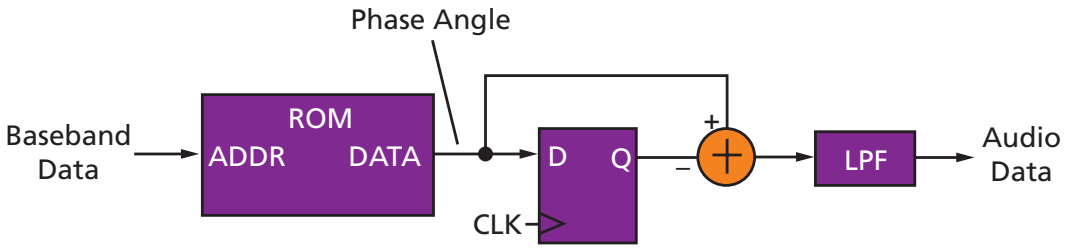


Figure 4 – Frequency demodulator, which converts the frequency of the IQ baseband signal to our audio signal

low-pass filter to obtain the orthogonal baseband signal. Here, we used the CIC Compiler 2.0 IP core to build a three-order, low-pass CIC decimation filter with a downsampling factor R = 240. The sample rate drops to 1 MSPS because the filtered baseband orthogonal signal is a narrowband signal.

Figure 4 shows the frequency demodulator, which converts the frequency of the IQ baseband signal to our audio signal. We extracted the IQ data's instantaneous phase angle using a lookup table in ROM. The IQ data is merged and is used as an address of the ROM; then the ROM outputs both the real and the imaginary part of the corresponding complex angle. Next we have a difference operation in which one tap by a

Slice Logic Utilization	Used	Available	Utilization
Slice registers	1,181	11,440	10%
Slice LUTs	1,062	5,720	18%
Occupied slices	406	1,430	28%
MUXCYs	544	2,860	19%
RAMB16BWERS	2	32	6%
RAMB8Bwers	1	64	1%
BUFG/BUFGMUXs	4	16	25%
PLL_ADVS	1	2	50%

Table 1 – Resource utilization on the XC6SLX9 FPGA

Figure 5 – Co-author and grad student Xingran Yang shows off the working model.

flip-flop delays the phase-angle data. When the delayed data is subtracted from the original data, the result is exactly the wanted audio signal. To improve the output signal-to-noise ratio, we filtered the audio signal through a low-pass filter using simple averaging.

The frequency demodulator's 8-bit audio data stream output is scaled according to the volume control parameter and sent into an 8-bit PWM module. The duty cycle of PWM pulses represents the amplitude of the audio signal. The pulses output at the FPGA's I/O pins and drive the headphone through capacitance. Here, the headphone acts as a low-pass filter, eliminating the high-frequency part of the pulses remaining in the audio signal.

Two rotary incremental encoders control the frequency and volume of the radio. Each of the encoders outputs two pulse signals. The rotational direction and rotational speed can be determined by the pulse width and phase. State machines and counters translate the rotation states into a frequency control word and a volume control word. Meanwhile, the frequency volume values are decoded and then displayed on the seven-segment LEDs.

ROOM TO CUSTOMIZE

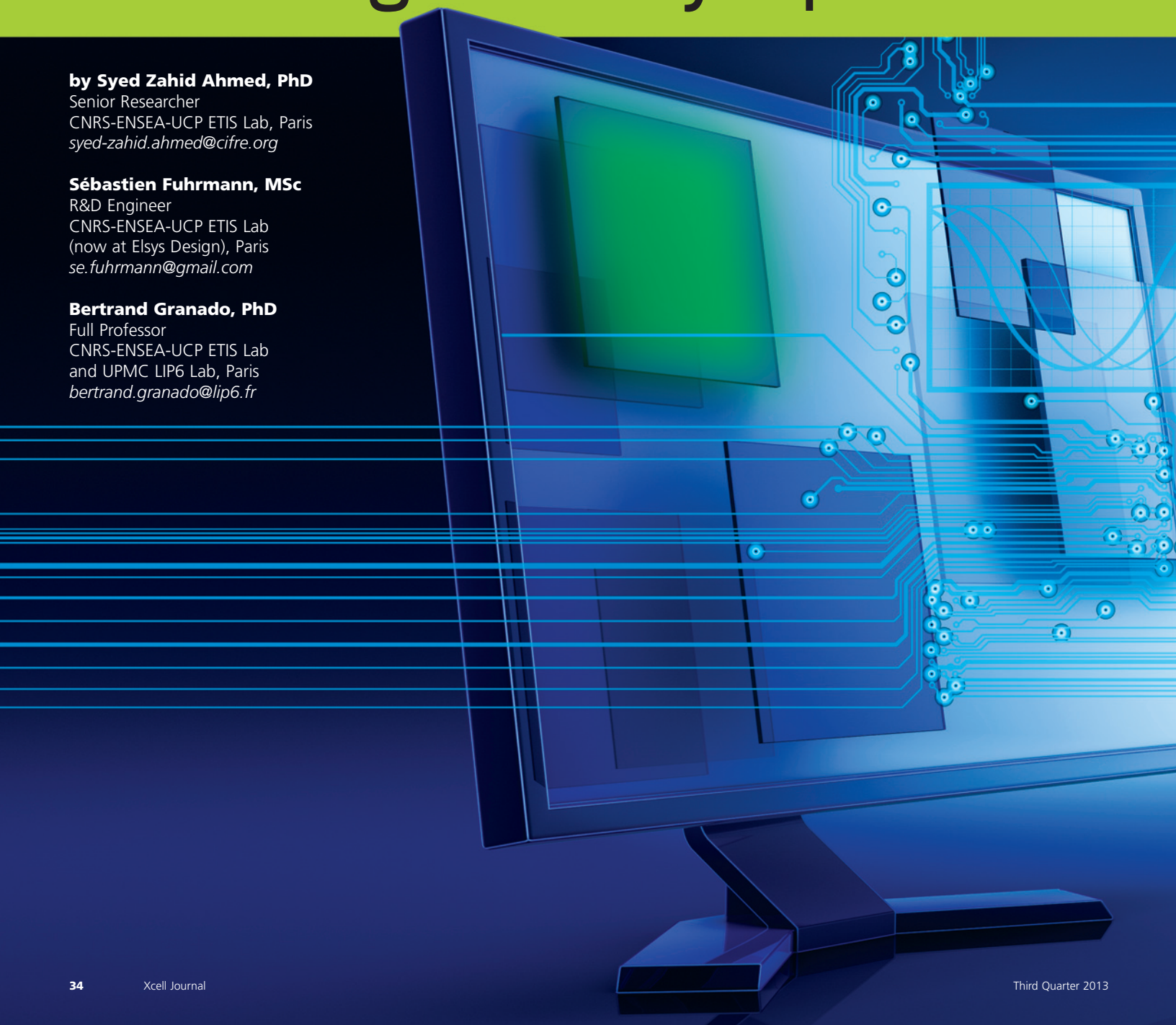
The printed-circuit board for the XRadio platform is shown in Figure 5. With a Spartan-6 XC6SLX9 FPGA, you can implement a radio program that consumes only a small part of the FPGA's logic resources, as shown in Table 1. This means there is room for students to customize the design and make improvements. For example, today with the 1-bit sampling, XRadio's audio quality is not as good as a standard FM radio. However, this leaves room for students to figure out how to improve the XRadio's audio quality and how to implement it in stereo.

Benchmark: Vivado's ESL Capabilities Speed IP Design on Zynq SoC

by Syed Zahid Ahmed, PhD
Senior Researcher
CNRS-ENSEA-UCP ETIS Lab, Paris
syed-zahid.ahmed@cifre.org

Sébastien Fuhrmann, MSc
R&D Engineer
CNRS-ENSEA-UCP ETIS Lab
(now at Elsys Design), Paris
se.fuhrmann@gmail.com

Bertrand Granado, PhD
Full Professor
CNRS-ENSEA-UCP ETIS Lab
and UPMC LIP6 Lab, Paris
bertrand.granado@lip6.fr



Automated methodology delivers results similar to hand-coded RTL for two image-processing IP cores

FPGAs are widely used as prototyping or actual SoC implementation vehicles for signal-processing applications. Their massive parallelism and abundant heterogeneous blocks of on-chip memory along with DSP building blocks allow efficient implementations that often rival standard microprocessors, DSPs and GPUs. With the emergence of the Xilinx® 28-nanometer Zynq®-7000 All Programmable SoC, equipped with a hard ARM® processor alongside the programmable logic, Xilinx devices have become even more appealing for embedded systems.

Despite the devices' attractive properties, some designers avoid FPGAs because of the complexity of programming them—an esoteric job involving hardware description languages such as VHDL and Verilog. But the advent of practical electronic system-level (ESL) design promises to ease the task and put FPGA programming within the grasp of a broader range of design engineers.

ESL tools, which raise design abstraction to levels above the mainstream register transfer level (RTL), have a long history in the industry. [1] After numerous false starts, these high-level design tools are finally gaining traction in real-world applications. Xilinx has embraced the ESL methodology to solve the programmability issue of its FPGAs in its latest design tool offering, the Vivado® Design Suite. Vivado HLS, the high-level synthesis tool, incorporates built-in ESL to automate the transformation of a design from C, C++ or SystemC to HDL.

The availability of Vivado HLS and the Zynq SoC motivated our team to do a cross-analysis of an ESL approach vs. hand-coding as part of our ongoing research into IP-based image-processing solutions, undertaken in a collaborative industrial-academic project. We hoped that our work would provide an unbiased, independent case study of interest to other design teams, since the tools are still new and not much literature is available on the general user experience. BDTI experiments [2, 3] provide a nice overview of the quality and potential of ESL tools for FPGAs at a higher level, but the BDTI team rewrote the source application in an optimal manner for implementation with ESL tools. Our work, by contrast, used the original legacy source code of the algorithms, which was written for the PC.

In comparing hand-coded blocks of IP that were previously validated on a Xilinx competitor's 40-nm midrange products to ESL programming on a Zynq SoC, we found that the ESL results rivaled the manual implementations in multiple aspects, with one notable exception: latency. Moreover, the ESL approach dramatically slashed the development time.

Key objectives of our work were:

- Assess the effort/quality of porting existing (C/C++) code to hardware with no or minimal modification;
- Do a detailed design space exploration (ESL vs. hand-coded IP);
- Investigate productivity trade-offs (efficiency vs. design time);
- Examine portability issues (migration to other FPGAs or ASICs);
- Gauge the ease of use of ESL tools for hardware, software and system designers.

VIVADO-BASED METHODOLOGY

We used Xilinx Vivado HLS 2012.4 (ISE® 14.4) design tools and the Zynq SoC-based ZedBoard (XC7z20clg484-1) for prototyping of the presented experiments. We chose two IP cores out of 10 from our ongoing Image Processing FPGA-SoC collaborative project for scanning applications. The chosen pieces of IP were among the most complex in the project. Optimally designed in Verilog and validated on the Xilinx competitor's 40-nm product, they perform specific image-processing functions for scanning products of the industrial partner, Sagemcom.

Figure 1 outlines the experimentation flow using Xilinx tools. For purposes of our experiments, we began with the original C source code of the algorithms. It is worth noting that the code was written for the PC in a classical way and is not optimal for efficient porting to an embedded system.

Our objective was to make minimal—ideally, zero—modifications to the code. While Vivado HLS supports ANSI C/C++, we did have to make some basic tweaks for issues like dynamic memory allocation and complex structure pointers. We accomplished this task without touching the core of the coding, as recommended by Xilinx documentation. We created the AXI4 master and slave (AXI-Lite) interfaces on function arguments and processed the code within Vivado HLS with varying constraints. We specified these constraints using directives to create different solutions that we validated on the Zynq Z20 SoC using the ZedBoard. The single constraint we had to make for comparison's sake involved latency. We had to assume the Zynq SoC's AXI4 interface would have comparable latency at the same frequency as the bus of the competitor's FPGA in the hand-coded design. Porting the RTL code (the core of the IP) to the Xilinx environment was straightforward, but redesigning the interface was beyond the scope of our work.

We used the Vivado HLS tool's built-in SystemC co-simulation feature to verify the generated hardware before prototyping on the board. The fact that Vivado HLS also generates software drivers (access functions) for the IP further accelerated the time it took to validate and debug the IP.

We used Core-0 of the Zynq device's ARM Cortex™-A9 MPCore at 666.7 MHz, with a DDR3 interface at 533.3 MHz. An AXI4 timer took care of latency measurements. Figure 2 presents the validation flow we used on the FPGA prototype. The image is processed with the original C source on ARM to get the "golden" reference results. We then processed the same image with the IP and compared them to validate the results.

As a standalone project, we also evaluated the latency measurements of the two blocks of IP for a 100-MHz MicroBlaze™ processor with 8+8KB I+D cache, and hardware multiplier and divider units connected directly with DDR3 via the AXI4 bus. We used the latency measurements of the two IP cores on the MicroBlaze as a reference to compare the speedup achieved by the Vivado HLS-generated IP and the ARM processor, which has brought significant software compute power to FPGA designers.

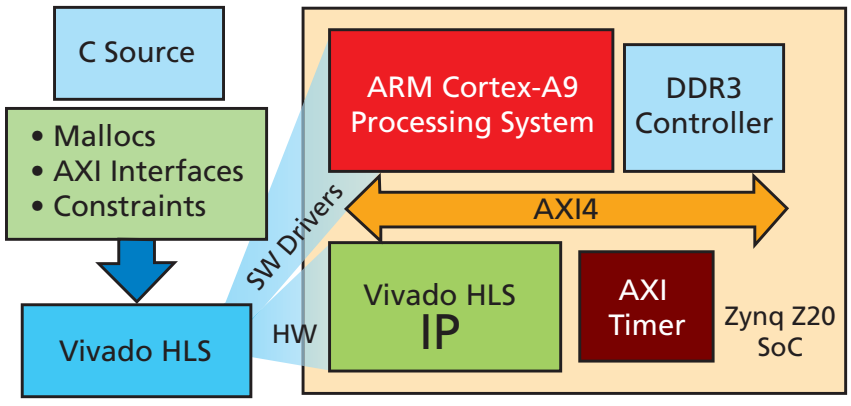


Figure 1 – The test hardware system generation for the Zynq Z20 SoC

EXPERIMENTAL RESULTS

Let's now take a closer look at the design space explorations of the two pieces of IP, which we have labeled IP1 and IP2, using the HLS tools. We've outlined the IP1 experiment in

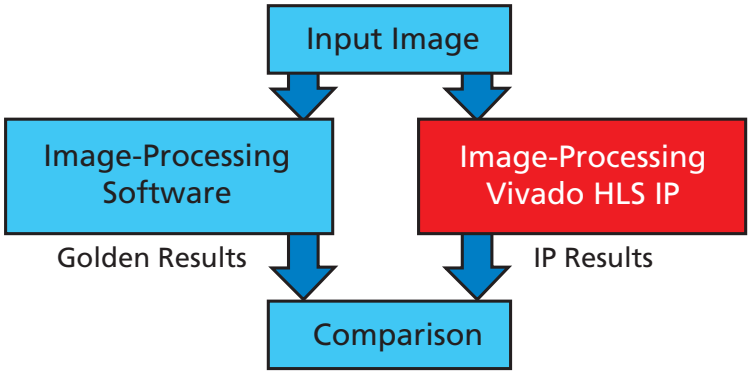


Figure 2 – ESL IP validation flow on the FPGA

detail but to avoid redundancy, for IP2 we've just shown the results. Finally, we will also discuss some feedback we received from the experts at Xilinx to our query regarding the efficiency of the AXI4 "burst" implementation used in the Xilinx ESL tools.

Tables 1 and 2 outline the details of our design space explorations for IP1 in the form of steps (we have listed them as S1, S2, etc.). For our experiments, we used a gray-scale test image of 256 x 256 pixels. It is important to note that the clock period, latency and power figures are just reference values given by a tool after quick synthesis for the cross-analysis of multiple implementations. The actual values can significantly vary after final implementation, as you will see in the discussion of the FPGA prototype results (Table 2).

S1: RAW CODE TRANSFORMATION

In the first step, we compiled the code with no optimization constraints (except the obligatory modifications described earlier). As you can see from Table 1, the hardware is using an enormous amount of resources. When we analyzed the details of resource distribution from the Vivado HLS quick-synthesis report and design-viewer GUI tool, it was clear that the primary contributor to this massive resource usage was several division operations in the algorithm. Unlike the multipliers, there are no hard blocks for division in an FPGA, so the division operations consumed a huge amount of logic. Therefore, the first thing we had to do was to optimize this issue.

S2: S1 WITH SHARED DIVIDER

Intuitively, our first thought for how to decrease the division hardware was to make use of resource sharing, with its trade-off of performance loss. Unfortunately, in the version of the tools we used it is not easy or even possible to apply custom constraints on operators directly. But the workaround was quite simple. We replaced the division operation (/) by a function, and on functions, constraints can be applied. By default, the tool uses functions as inline: For each division, it generates complete division hardware. We applied the "Allocation" directive on the division function, experimented with different values and monitored their effect on latency. Finally, we opted for single, shared division hardware. As you can see in Table 1, this choice drastically reduced lookup tables (LUTs), flip-flops (FFs) and other logic resources with just a negligible impact on latency—a complete win-win situation.

S3: S2 WITH CONSTANT DIVISION TO MULTIPLICATION CONVERSION

In the next step, we opted to transform the division operation into multiplication, which was possible because this application uses division by constant values. Our motivation here was twofold. First, despite deciding to use a single shared divider for all divisions, division hardware is still expensive (around 2,000 LUTs and flip-flops). Second, this shared divider takes 34 cycles for division, which is quite slow, and the version of the Xilinx tools we used supports only one type of divider in the HLS library. But there are several options for multiplication, as we will see in the next steps.

Therefore, in this step we transformed the division operations to multiplications (we used the #ifdef directives of C so as to keep the code portable). The approximation values were provided as function parameters via AXI-Lite slave registers. This is why Table 1 shows a significant improvement in performance and resources at this step, with a slight trade-off in added DSPs due to increased multiplication operations.

S4: S3 WITH BUFFER MEMORY PARTITIONING

The source algorithm uses multiple memory buffers (8, 16 and 32 bits) arranged together in a unified malloc block. During the initial transformation to make the algorithm Vivado HLS ready (Figure 1), that memory block became an internal on-chip memory (Block RAMs).

While a unified malloc block was a normal choice for shared external memory, when using internal on-chip memories, it is much more interesting to exploit memory partitioning for faster throughput. In this step, therefore, we fractured the larger memory block into independent small memories (8, 16 and 32 bits). Table 1 shows the results, highlighting significant performance improvements with a slight increase in the number of BRAMs (Table 2) due to fractured use. This technique also yielded a significant decrease in logic resources, potentially due to simplified control logic for memory access.

IP1: Vivado HLS results for Zynq Z20 SoC	LUTs	FFs	BRAMs	DSP	ClkPr (ns)	Latency (clk cycles)	Speedup	Power*
S1: Raw transform	20,276	18,293	16	59	8.75	8,523,620	NA	1410
S2: S1 with Shared Div	9,158	7,727	16	51	8.75	8,555,668	1	1638
S3: S2 with DivToMult	7,142	6,172	16	84	8.75	3,856,048	2.22	1337
S4: S3 with Buffer Memory Partitioning	4,694	4,773	16	81	8.75	2,438,828	3.51	953
S5: S4 with Shared 32/64bit Multipliers	4,778	4,375	16	32	8.75	3,111,560	2.75	890
S6: S5 with Smart Mult sharing	4,900	4,569	16	40	8.75	2,910,344	2.94	918
S7: S6 with Mult Lat. exp (comb. Mult)	4,838	4,080	16	40	13.56	1,975,952	4.33	893
S8: S6 with Mult Lat. exp (1clk cycle Mult)	4,838	4,056	16	40	22.74	2,181,776	3.92	890
S9: S4 with Superburst	4,025	4,357	80	79	8.75	NA	NA	NA
S10: S4 with Smart Burst	3,979	4,314	24	80	8.75	NA	NA	NA
S11: S6 with Smart Burst	4,224	4,079	24	39	8.75	NA	NA	NA

Table 1 – HLS exploration results for IP1

*Power value of tool for iterations comparison, it has no units

IP1: Zynq Z20 board results for 256x256-pixel test image	LUTs	FFs	BRAMs	DSP	Fmax (MHz)	Fexp (MHz)	Latency (ms)	Speedup	Power** (mW)
ARM CortexA9 MPCore (Core-0 only)	NA	NA	NA	NA	NA	666.7	32.5	15	NA
MicroBlaze (with Cache & int. Divider)	2,027	1,479	6	3	NA	100	487.4	1	NA
Hand-Coded IP (Resources for Ref.)	8798	3,693	8	33	54.2	*50(NA)	*4 (NA)	NA	57
S3: S2 with DivToMult	6,721	6,449	8	72	40.2	40	142.4	3.42	12
S4: S3 with Buffer Mem. Partitioning	4,584	5,361	10	69	84.5	83.3	124.5	3.91	30
S6: S5 with Smart Mult sharing	5,156	4,870	10	32	91.3	90.9	136.9	3.56	26
S8: S6 with Mult Latency exp (1clk)	5,027	4,607	10	33	77.3	76.9	121.2	4.02	22
S9: S4 with Superburst	4,504	5,008	42	70	77.6	76.9	26.5	18.39	NA
S10: S4 with Smart Burst	4,485	4,981	14	73	103	100	22.48	21.68	62
S11: S6 with Smart Burst	4,813	4,438	14	33	101	100	33.7	14.46	42

Table 2 – FPGA prototype results for IP1 explorations

*Reference values, see methodology section

**Dynamic power from XPower Analyzer

S5: S4 WITH SHARED 32/64-BIT SIGNED/UNSIGNED MULTIPLIERS

In this step, motivated by the shared-divider experiments in S2, we focused on optimizing the DSP blocks using shared resources. Our first step was to conduct a detailed analysis of the generated hardware using the Design Viewer utility to cross-reference generated hardware with the source C code. The analysis revealed that in raw form, the hardware is using 26 distinct multiplier units of heterogeneous types, which can be classified as unsigned 16-/32-/64-bit multiplications and 32-bit signed multiplications. As in our S2 experiments, we constructed a system consisting of 32-bit signed, 32-bit unsigned and 64-bit unsigned multipliers (by replacing the "*" operation with a function). We then applied an Allocation constraint of using only one unit for these three distinct operations. Table 1 shows the resulting drastic reduction in DSP usage, albeit with some penalty in performance. This result motivated us to further investigate smart sharing.

S6: S5 WITH SMART MULTIPLICATION SHARING

To mitigate the performance loss arising from shared multiplier usage, we made smart use of multiplier sharing. We created two additional multiplier types: a 16-bit unsigned multiplier and a 32-bit unsigned multiplier with 64-bit return value. After several iterations of smartly using the multipliers in a mutually exclusive manner and changing their quantity, our final solution contained two unsigned 64-bit multipliers, two unsigned 32-bit multipliers, one signed 32-bit multiplier, one unsigned 16-bit multiplier and a single unsigned 32-bit multiplier with 64-bit return value. This technique improved performance slightly, as you can see in Table 1, but with slightly higher DSP usage.

S7: S6 WITH MULTIPLIER LATENCY EXPERIMENTS (COMBINATIONAL MULTIPLIER)

In the final stages of multiplier optimization, we conducted two experiments for changing the latency value

of the multipliers. By default, Vivado HLS used multipliers with latencies ranging from two to five clock cycles. In hard IP, all multipliers are either single-clock-cycle or combinational. In this step, we set the multipliers' latency to 0 by using the "Resource" directive and choosing the corresponding combinational multiplier from the library. The results, as seen in Table 1, show an improvement in latency. However, the design has become much slower from the standpoint of timing, so the increased performance may well turn into a loss due to the slower clock speed of the FPGA design.

S8: S6 WITH MULTIPLIER LATENCY EXPERIMENTS (1-CLOCK-CYCLE MULTIPLIER)

In this step, instead of using a combinational multiplier we selected a single-clock-cycle multiplier from the library. Results are shown in Table 1. We can see that while the latency has slightly increased, the clock period, surprisingly, has significantly increased, which may lead to slower hardware.

S9, S10, S11: BURST-ACCESS EXPERIMENTS

All the exploration steps presented so far required very little or no knowledge of the functionality of the C code. In the final experiments, we explored burst access, which is almost obligatory for IP that is based on shared-memory systems due to high latency of the bus for small or random data accesses. Toward this end, we analyzed the algorithmic structure of the C code to explore the possibility of implementing burst access, since natively there is no such thing as burst access from a general software standpoint. However, it is important to note that even with our burst-access experiments, we never modified the software structure of the code.

We performed our burst-access experiments in two steps. First we used a dump approach (not practical, but a good way to get some technical information), where the entire image was burst into the internal memory buffer and the full results burst out at the end of execution. This was easy to do by using a small image and by the presence of abundant on-chip memory in the FPGA. We called this iteration step "superburst" (S9).

In the second phase, we implemented a smart, interactive burst (S10, S11) within the limitations of not modifying the source code structure. The results are shown in Table 1. We implemented the burst functionality in C using the standard "memcpy" function, based on tutorial examples from Xilinx. Unfortunately, it is not possible to see latency values when you insert the memcpy function in the code. The FPGA implementation (Table 2) will illustrate the obtained behaviors.

FPGA PROTOTYPE OF SELECTED STEPS

One of the greatest benefits of using ESL tools is rapid design space exploration. We were able to evaluate all the exploration steps listed above and the results presented in Table 1 within minutes, avoiding the lengthy implementation phase of an FPGA for only a few, final selected iterations. Table 2 presents the final implementation results of some selected steps of Table 1 on our Zynq Z20 SoC. It also presents the equivalent results (latency) of implementing the original source code on a MicroBlaze processor, as a reference, and on the ARM Cortex-A9, showing the tremendous compute power that the Zynq SoC has brought to FPGA designers. The table also shows estimated dynamic-power values calculated by the Xilinx XPower Analyzer.

We obtained the implementation statistics of the optimal hand-coded IP on the Zynq SoC by implementing their RTL using Xilinx tools. We had to use the latency measurements of our previous experiments with a competitive FPGA as a reference, since porting to the AXI4 bus was beyond the scope of our experiments. This further highlights the potential of HLS tools, which can make it easier for designers to port to multiple protocols.

If we compare the results of Table 2 with those of Table 1, several interesting observations emerge. First, the quick synthesis of ESL gives a relatively good estimate compared with the actual implementation (especially for LUTs and flip-flops); for memory and DSP, the synthesizer sometimes can significantly change the resource count for optimizations.

Second, for final implementation, timing closure can be a big problem. We can see a significant change in achieved frequency after FPGA implementation compared with the HLS estimate. Finally, it is surprising that even with the burst-access experiments, there is a significant difference in the latency values from the expected HLS values and hand-coded IP. This piqued our curiosity about the quality of DMA that the HLS tool generated.

NEW IP, SIMILAR RESULTS

In a similar fashion, we likewise explored our second piece of IP, which we called IP2, in the same detail. The results, illustrated in Tables 3 and 4, are self-explanatory in the light of our in-depth discussion of our experiences with IP1. One point that's interesting to note is step 8 in Table 3. Just as with IP1 (see the S2 and S3 discussions), the divider is expensive in terms of resources and also latency. Unlike IP1, IP2 uses real division, so the divider cannot be removed. Unfortunately, in the current version of the ESL tools, only one divider is present, with 34 clock cycles of latency. S8 shows that if a divider with a latency of a single clock cycle were used, like those available in hand-coded IP, theoretically it could yield a performance improvement of around 30 percent. Table 4 shows FPGA prototype results of selected steps.

IP2: Vivado HLS results for Zynq Z20 SoC	LUTs	FFs	BRAMs	DSP	ClkPr (ns)	Latency (clk cycles)	Speedup	Power*
S1: Raw transform	25,746	23,392	16	76	8.75	8,691,844	NA	NA
S2: Raw with SharedDiv	9,206	7,900	16	68	8.75	11,069,548	1	1643
S3: S2 + const Mult & Div ops reduction	9,703	8,557	16	96	8.75	4,934,764	2.24	1712
S4: S3 with Buffer Memory Partitioning	7,204	7,249	16	93	8.75	3,539,564	3.13	1338
S5: S4 with Smart Mult sharing	7,240	6,663	16	48	8.75	3,882,604	2.85	1357
S6: S4 with Smart Burst	6,816	6,515	24	78	8.75	NA	NA	1240
S7: S5 with Smart Burst	6,846	5,958	24	33	8.75	NA	NA	1262
S8: S5 Theoretical Lat. with 1 clk cycle Div.	NA	NA	NA	NA	NA	2,751,084	4.02	NA

Table 3 – HLS exploration results for IP2

*Power value of tool for iterations comparison, it has no units

IP2: Zynq Z20 board results for 256x256-pixel test image	LUTs	FFs	BRAMs	DSP	Fmax (MHz)	Fexp MHz)	Latency (ms)	Speedup	Power** (mW)
ARM CortexA9 MPCore (Core-0 only)	NA	NA	NA	NA	NA	666.7	37.9	13.39	NA
MicroBlaze (with Cache & int. Divider)	2,027	1,479	6	3	NA	100	507.6	1	NA
Hand-Coded IP (Resources for Ref.)	3,671	2,949	2	16	63.8	*50(NA)	*4 (NA)	NA	25
S3: S2 + const Mult & Div ops reduction	8,620	8,434	8	81	44.2	40	236.8	2.14	24.4
S4: S3 with Buffer Memory Partitioning	6,523	7,403	10	78	88.5	83.3	124.4	4.08	41.5
S5: S4 with Smart Mult sharing	6,722	7,017	10	41	94.1	90.9	124.2	4.09	36.4
S6: S4 with Smart Burst	6,240	6,486	14	69	96.2	90.9	41.2	12.32	56.4
S7: S5 with Smart Burst	6,435	6,174	14	30	95.1	90.9	45.8	11.08	50.1

Table 4 – FPGA prototype results for IP2 explorations

*Reference values, see methodology section

**Dynamic power from XPower Analyzer

VIVADO HLS DMA/BURST EFFICIENCY ISSUE

Comparing the results of the hand-coded IP implementation with ESL techniques, several observations can be made. Despite using the original source code (with almost no modification), which was not written in an optimal manner to exploit embedded systems, the ESL tools delivered results that rivaled optimally hand-coded IP in terms of resource utilization. The significant difference in the explored results for ESL vs. hand-coded IP came in latency, which was far better in the hand-coded IP.

Looking into this issue, we observed that there is a major difference in the way the hand-coded IP and ESL IP are made. (This is also the reason for the big difference in resources consumed for IP2.) Due to the style of the source code, there is only one data bus for input and output in the ESL IP, compared with two for the hand-coded IP. Also, FIFOs handle burst access much more optimally in hand-coded IP. For the ESL IP, the burst is done with buffers, and due to the nature of the original code it was quite hard to optimally create an efficient burst mechanism. The ESL approach entails a penalty in BRAM usage, since the ESL hardware is using these memories for work buffers and burst buffers in accordance with the code structure. In addition, the burst is done using the memcpy function of C, in accordance with Xilinx tutorials for the AXI4 master interface.

We also conferred with Xilinx experts, who recommended using the external DMAs manually for optimal performance, as the AXI4 master interface generated by the version of Vivado HLS we used is in beta mode and will be upgraded in the future. Such factors might have caused a significant difference in latency values and might be a good starting point for future experiments.

PRODUCTIVITY VS. EFFICIENCY

The detailed experiments undertaken in this work revealed several interest-

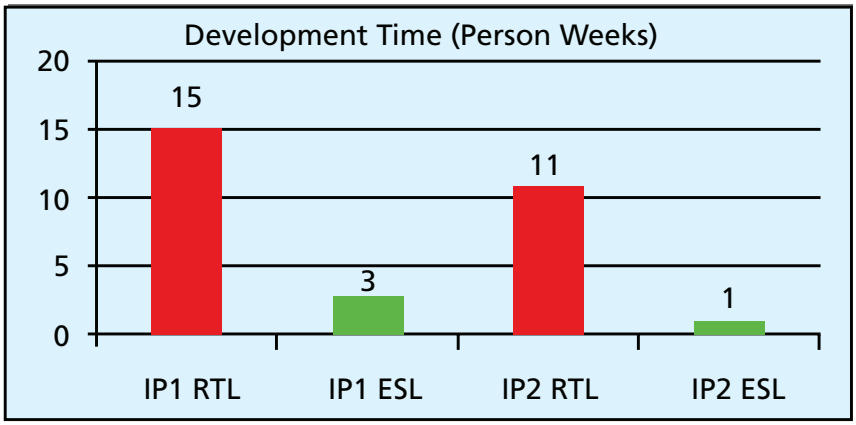


Figure – 3: Hand-coded RTL vs. ESL development time

ing aspects of ESL design and also cleared up some misconceptions. ESL tools have come a long way since the early iterations. They now support ANSI C/C++, give good competitive results and exploit several design decisions (not just the loop exploitations of the past) inspired by hardware/system design practices, in the form of constraints directives.

Indeed, we found that the results obtained from ESL rivaled the optimal results in all aspects except one: latency. Furthermore, using accurate bit types with advanced features of the tools (instead of the generic C types we used in these experiments) may further improve resources, as could partial rewriting of code.

Our explorations also gave us an idea about productivity vs. efficiency trade-offs. Figure 3 illustrates the relative development time of IP blocks (in terms of design, software, FPGA implementation and verification). The fast time-to-solution for ESL-based design is dramatic, especially for verification, with results that rival optimal implementation. It is important to note that design/validation time is highly dependent on the skills of the designers. Since the same group of people was involved in both the hand-coded design and the ESL-based design, Figure 3 gives quite an unbiased overview of how the process plays out. Considering the fact that designers were well acquainted with the classical RTL method of IP design and integration, and were new to ESL tools, it's likely that implementation time could significantly improve in the longer term, once the tools are more familiar.

WHAT NEXT FOR ESL?

From a technology and also a business standpoint, there should be some EDA standardization among ESL tools' constraints structure, making designs easy to port to other FPGAs or ASICs (flexibility that is inherent in RTL-based design tools and in principle, even easier to achieve with ESL tools). ESL tool providers should differentiate in the quality of their tools and results only, as they do for RTL tools.

It's a long and complex discussion, and beyond scope of this short article, to answer questions like: Have ESL tools enabled software designers to program FPGAs? Will ESL eliminate RTL designers/jobs? Who is the biggest winner with ESL tools? In our brief experience with this work, we have found that ESL tools are beneficial for all, especially for the system designers.

For hardware designers, the tools can be an interesting way to create or evaluate small portions of their design. They can be helpful in rapidly creating a bus-interface-ready test infrastructure. Software designers trying to design hardware are one of the key targets for ESL tools. And while ESL tools have come a long way for this purpose, they still have a long way to go. In our project, for example, ELS tools hid the hardware complexities to a great extent from software designers. Nevertheless, good implementation might still require a hardware mind-set when it comes to constraints and optimization efforts. This issue will get better as the ESL tools further evolve.

As for the system designers, who are becoming more and more common due to the increased integration and penetration of HW/SW co-design in the world of SoCs, ESL tools perhaps are a win-win situation. System designers can exploit these tools at multiple levels.

On the hardware side, one of the key reasons to choose the Zynq-7000 All Programmable SoC among Xilinx's 28-nm 7 series FPGAs was to explore the potential of having a hard ARM core onboard the same chip with FPGA logic. As we have seen in these experimental results, the dual-core ARM has brought unprecedented compute power to FPGA designers. Indeed, the Zynq SoC delivers a mini-supercomputer to embedded designers, who can customize it using the tightly integrated FPGA fabric.

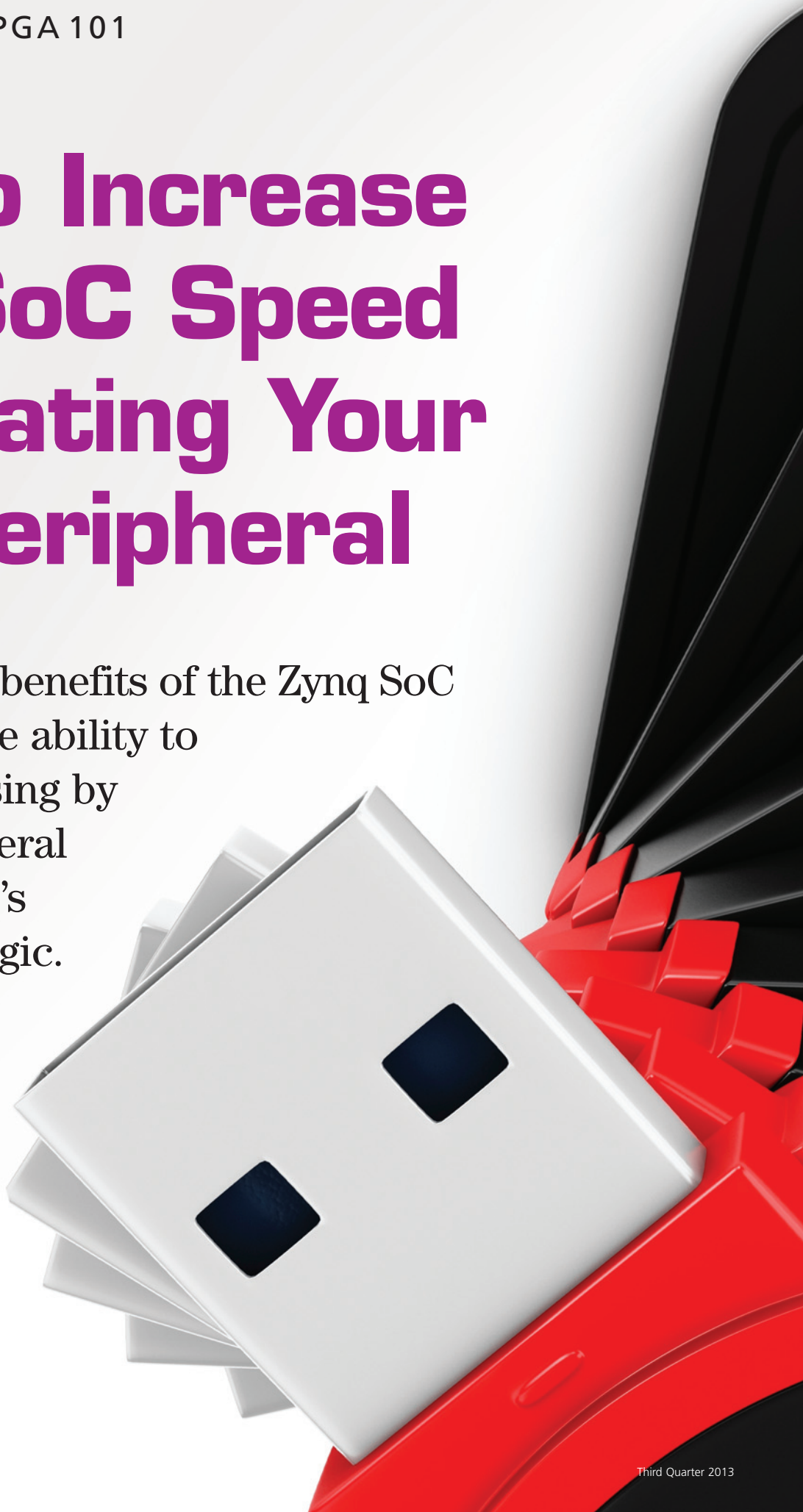
Acknowledgements

We are grateful to our industrial partner, Sagemcom, for funding and technical cooperation. We would also like to thank Xilinx experts in Europe, Kester Aernoudt and Olivier Tremois, for their prompt technical guidance during this work.

References

1. *G. Martin, G. Smith, "High-Level Synthesis: Past, Present and Future," IEEE Design and Test of Computers, July/August 2009*
2. *BDTI report, "High-Level Synthesis Tools for Xilinx FPGAs," 2010*
3. *BDTI report, "The AutoESL AutoPilot High-Level Synthesis Tool," 2010*

How to Increase Zynq SoC Speed by Creating Your Own Peripheral



One of the major benefits of the Zynq SoC architecture is the ability to speed up processing by building a peripheral within the device's programmable logic.

by Adam Taylor
Senior Specialist Engineer
EADS Astrium
aptaylor@theiet.org

This is the third in a planned series of hands-on Zynq-7000 All Programmable SoC tutorials by Adam Taylor. Earlier installments appeared in Xcell Journal *issues 82 and 83. A frequent contributor to* Xcell Journal, *Adam is also a blogger at All Programmable Planet* (www.programmableplanet.com).

One of the real advantages of the Xilinx® Zynq™-7000 All Programmable SoC is the ability to increase the performance of the processing system (PS) side of the device by creating a peripheral within the programmable logic (PL) side. At first you might think this would be a complicated job. However, it is surprisingly simple to create your own peripheral.

Adding a peripheral within the PL can be of great help when you're trying to speed up the performance of your processing system or when you're using the PS to control the behavior of the design within the programmable logic side. For example, the PS might use a number of memory-mapped registers to control operations or options for the design within the programmable logic.

Our example design is a simple module that allows you to turn on and off a memory-mapped LED on the ZedBoard. To create this module, we will use the Xilinx PlanAhead™, XPS and Software Development Kit (SDK) tool flow in a three-step process.

1. Create the module within the Embedded Development Kit (EDK) environment.
2. Write the actual VHDL for our module and build the system.
3. Write the software that uses our new custom module.

CREATING THE MODULE IN EDK

To create your own peripheral, you will first need to open Xilinx Platform Studio (XPS) from the PlanAhead project containing your Zynq SoC design and select the menu option "hardware->create or import peripheral."

For our example module, the seven LEDs on the ZedBoard should demonstrate a walking display, illuminating one after the other, except that the most significant LED will be toggled under control of the software application. While this is not the most exciting application of a custom block, it is a useful simple example to demonstrate the flow required. Once you've grasped that process, you will know everything you need to know to implement more-complex modules.

You create a peripheral by stepping through 10 simple windows and selecting the options you desire.

The first two screens (shown in Figure 1) ask if you want to create or import an existing peripheral and if so, to which project you wish to associate it. The next stage, at screen 3, allows you to name your module and very usefully lets you define versions and revisions. (Note that the options at the bottom of the peripheral flow enable you to recustomize a module you have already created by reading back in the *.cip file.)

Once you have selected your module name, you can then choose the complexity of the AXI bus required, as demonstrated in Figure 2. The one we need for our example design is a simple memory-mapped control register-type interface; hence, we have chosen the top option, AXI4-Lite. (More information on the different types of AXI buses supported is available at *http://www.xilinx.com/support/documentation/white_papers/wp379_AXI4_Plug_and_Play_IP.pdf*.) This step will create a simple number of registers that can be written to and read from over the AXI bus via the processing system. These registers will be located within the programmable logic fabric on the Zynq SoC and hence are also accessible by the user logic application, where you can create the function of the peripheral.

The next six screens offer you the option to customize an AXI master or slave configuration, and the number of registers that you will be using to control your user logic along with the supporting files. For this example we are using just the one register, which will be functioning as an AXI-Lite slave. The most important way to ease the development of your system is to select the "generate template driver files" on the peripheral implementation support tab. Doing so will deliver a number of source and header files to help communicate within the peripheral you create.

At the end of this process, XPS will have produced a number of files to help you with creating and using your new peripheral:

- A top-level VHDL file named after your peripheral
- A VHDL file in which you can create your user logic
- A Microprocessor Peripheral Description, which defines the interfaces to your module such that it can be used with XPS
- A CIP file, which allows recustomization of the peripheral if required
- Driver source files for the SDK
- Example source files for the SDK
- A Microprocessor Driver Definition, which details the drivers required for your peripheral

These files will be created under the PlanAhead sources directory, which in our case is *zed_blog.srcs\sources_1\edk\proc_subsystem*.

Beneath this you will see the Pcores directory, which will contain the files (under a directory named after you peripheral) required for XPS and PlanAhead, along with the drivers directory, which contains the files required for the SDK.

CREATING THE RTL

Within the Pcores directory you will see two files, one named after the component you created (in this case, led_if.vhd) and another called

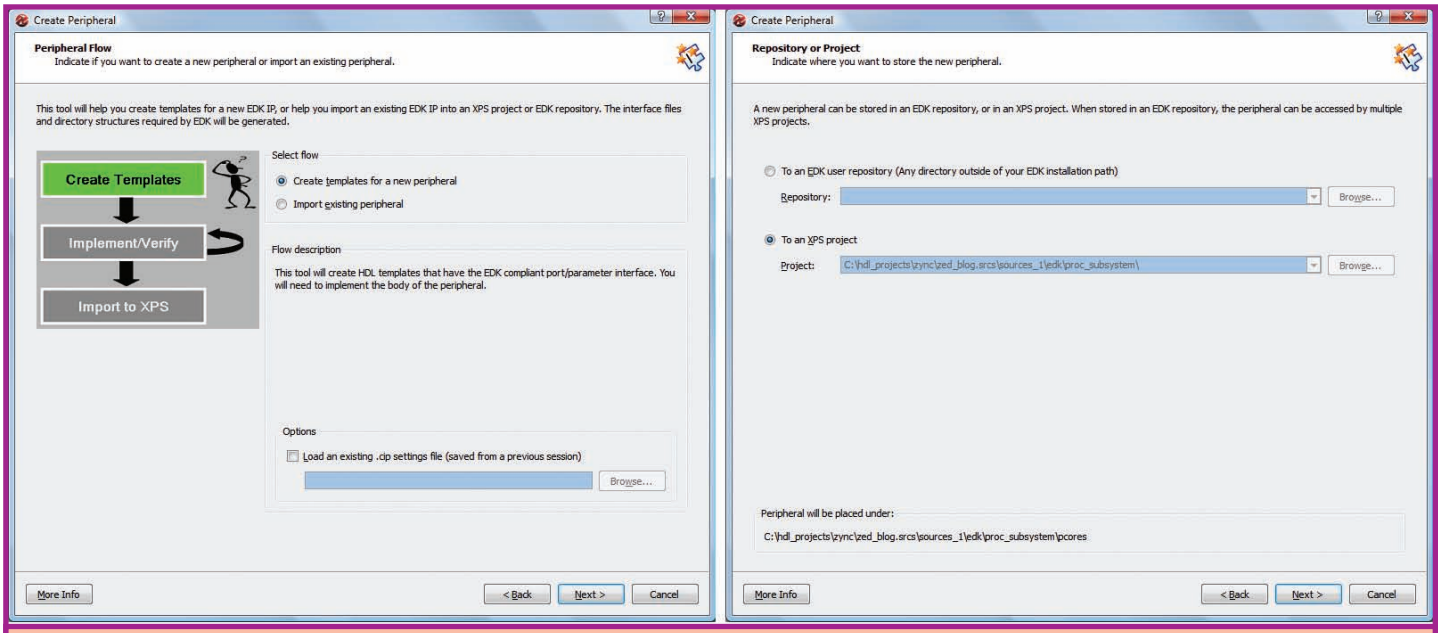


Figure 1 – The first two of the 10 wizard screens

user_logic.vhd. It is in this latter file that you create your design. You will find that the wizard has already given you a helping hand by generating the registers and interfaces necessary to communicate over the AXI bus within this file. The user-logic module is instantiated within led_if.vhd. Therefore, any changes you make to the entity of user_logic.vhd must be flowed through into the port map. If external ports are required—as they are in our example design, to flash the LED—you must also flow through your changes in the entity of led_if.vhd.

Once you have created your logic design, you may wish to simulate it to ensure that it functions as desired. Bus-functional models are provided under the Pcores/devl/bfmsim directory.

USING YOUR MODULE WITHIN XPS

Having created the VHDL file or files and ensuring they function, you will wish to use the module within your XPS project. To correctly do this you will need to ensure the Microprocessor Peripheral Description file is up to date with the ports you may have added during your VHDL design. You can accomplish this task by editing the MPD file within XPS. You will find your created peripheral under the Project Local Pcores->USER->core name. Right-clicking on this name will enable you to view your MPD file.

You will need to add the non-AXI interfaces you included in your design so that you can see the I/Os and deal with them correctly within XPS. You will find the syntax for the MPD file at *http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/psf_rm.pdf*, although it is pretty straightforward, as you can see in Figure 3. The highlighted line (line 59, at the bottom of the central list) is the one I added to bring my LED output port to the module.

When you have updated the MPD file, you must click on "re-scan user IP repositories" to ensure the changes are loaded into XPS. At that point, you can then implement the device into your system just like any other peripheral from the IP library. Once you have it in the address space you want (remember, the minimum is 4K) and have connected your outputs as required to external ports, if necessary you can run the design rule checker. Provided there are no errors, you can close down XPS and return to PlanAhead.

Back within the PlanAhead environment, you need to regenerate the top level of HDL; to do this, just right-click on the processor subsystem you created and select "create top HDL" (see *Xcell Journal* issue 82 for information on creating your Zynq SoC PlanAhead project). The final stage before implementation is to modify the UCF file within PlanAhead if changes have been made to the I/O pinout of your design. Once you are happy with the pinout, you can then run the implementation and generate the bitstream ready for use in the SDK.

USING THE PERIPHERAL IN SDK

Having gotten this far, we are ready to use the peripheral within the software environment. As with the tasks of creation and implementation, this is an easy and straightforward process. The first step is to export the hardware to the Software Development Kit using the "export hardware for SDK" option within PlanAhead. Next, open the SDK and check the MSS file to make sure that your created peripheral is now present. It should be listed with the name of the

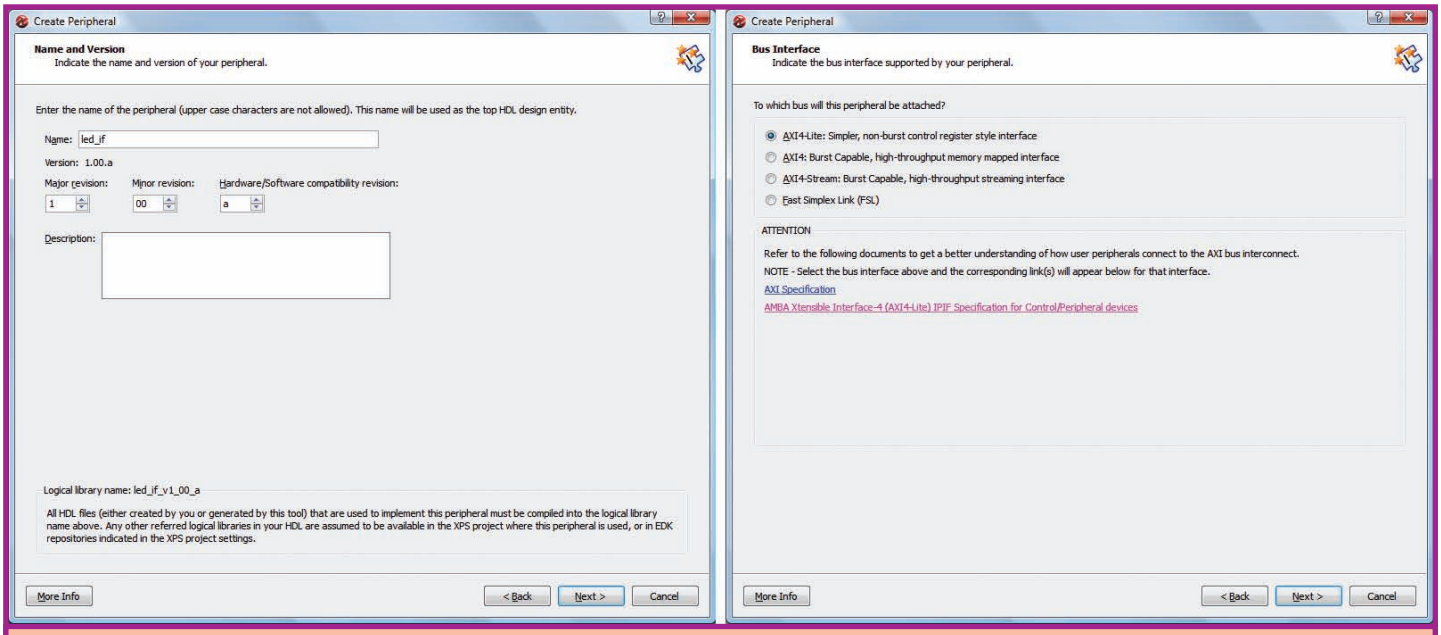


Figure 2 – The next two windows, covering name, revision and bus standard selection

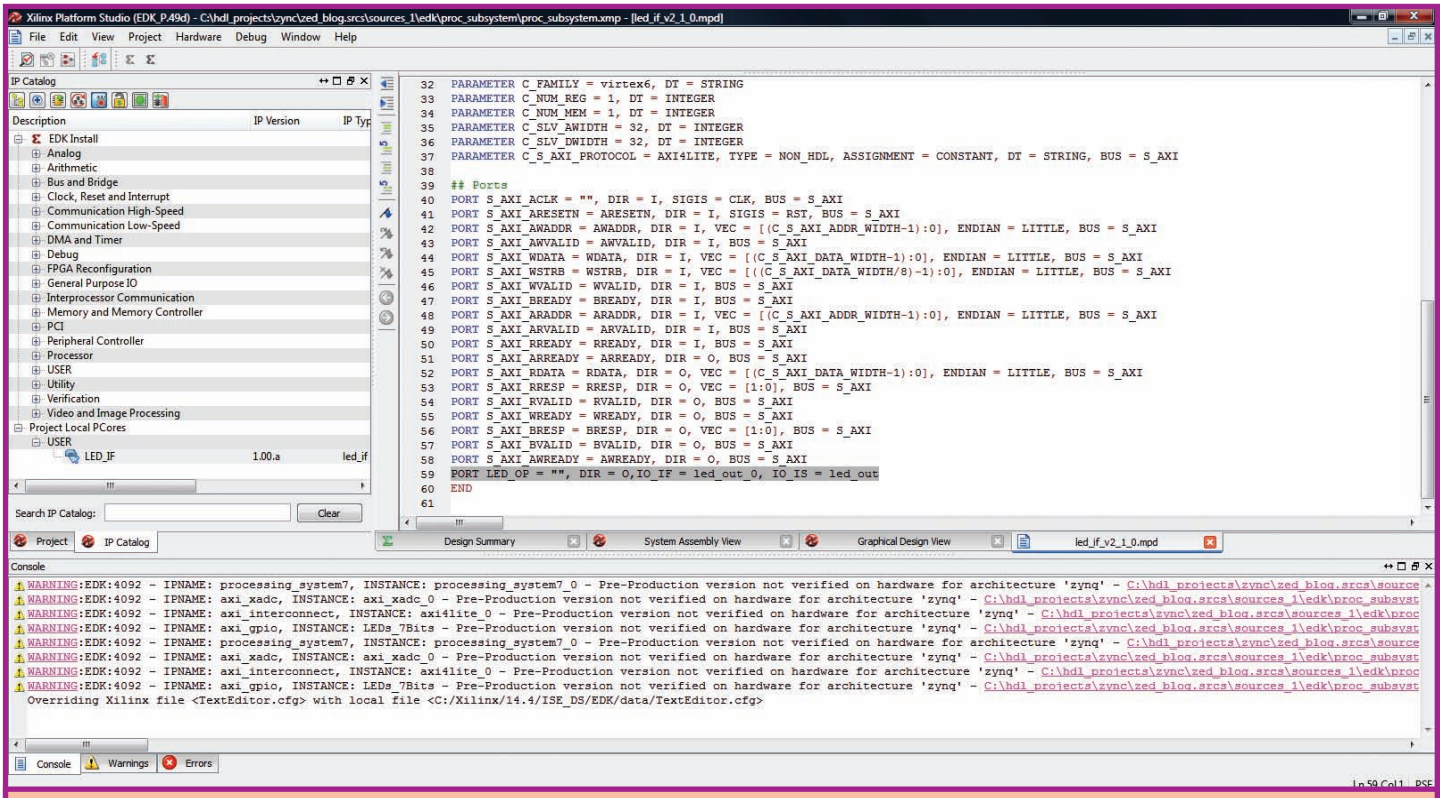


Figure 3 – Editing the MPD file to include your peripheral

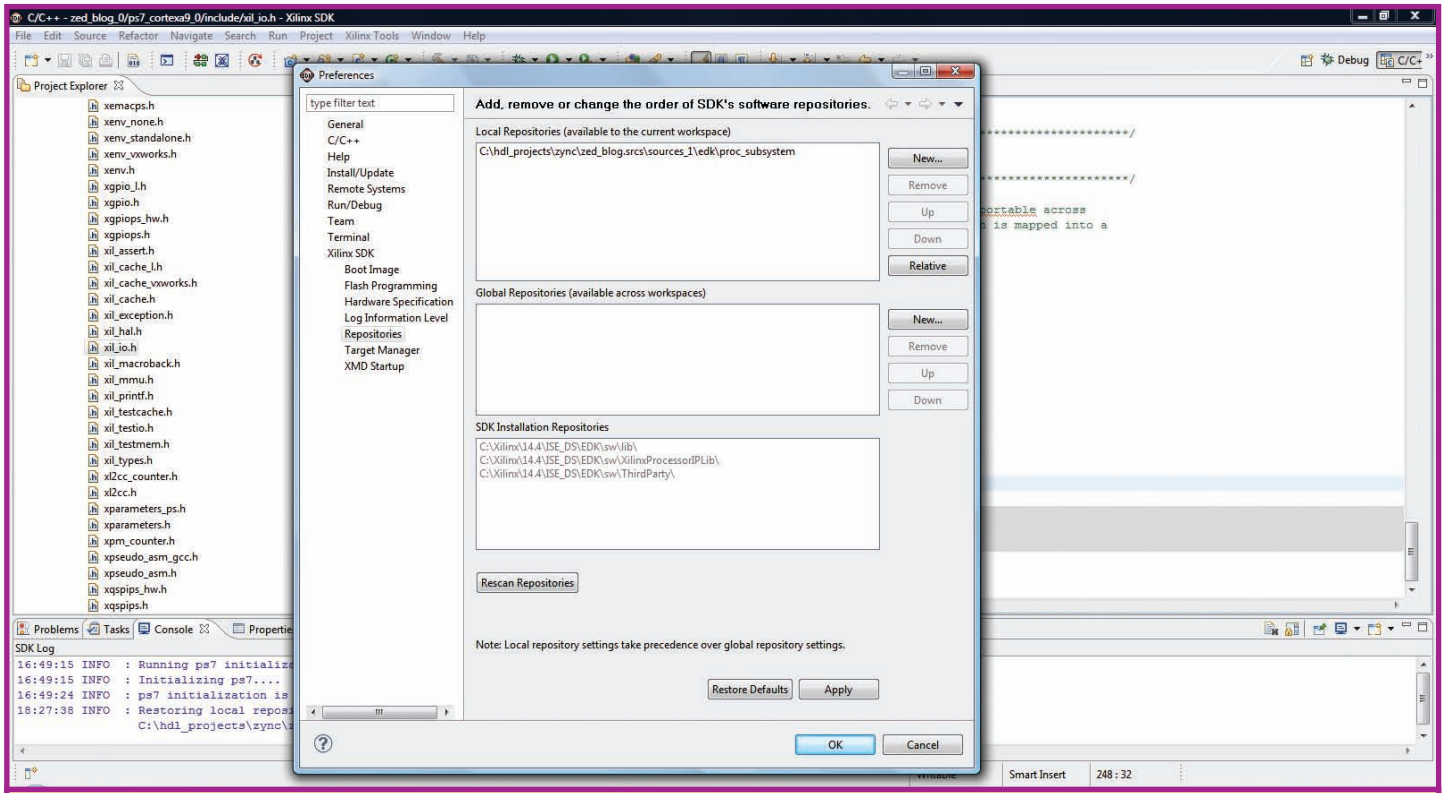


Figure 4 – Importing the software repositories

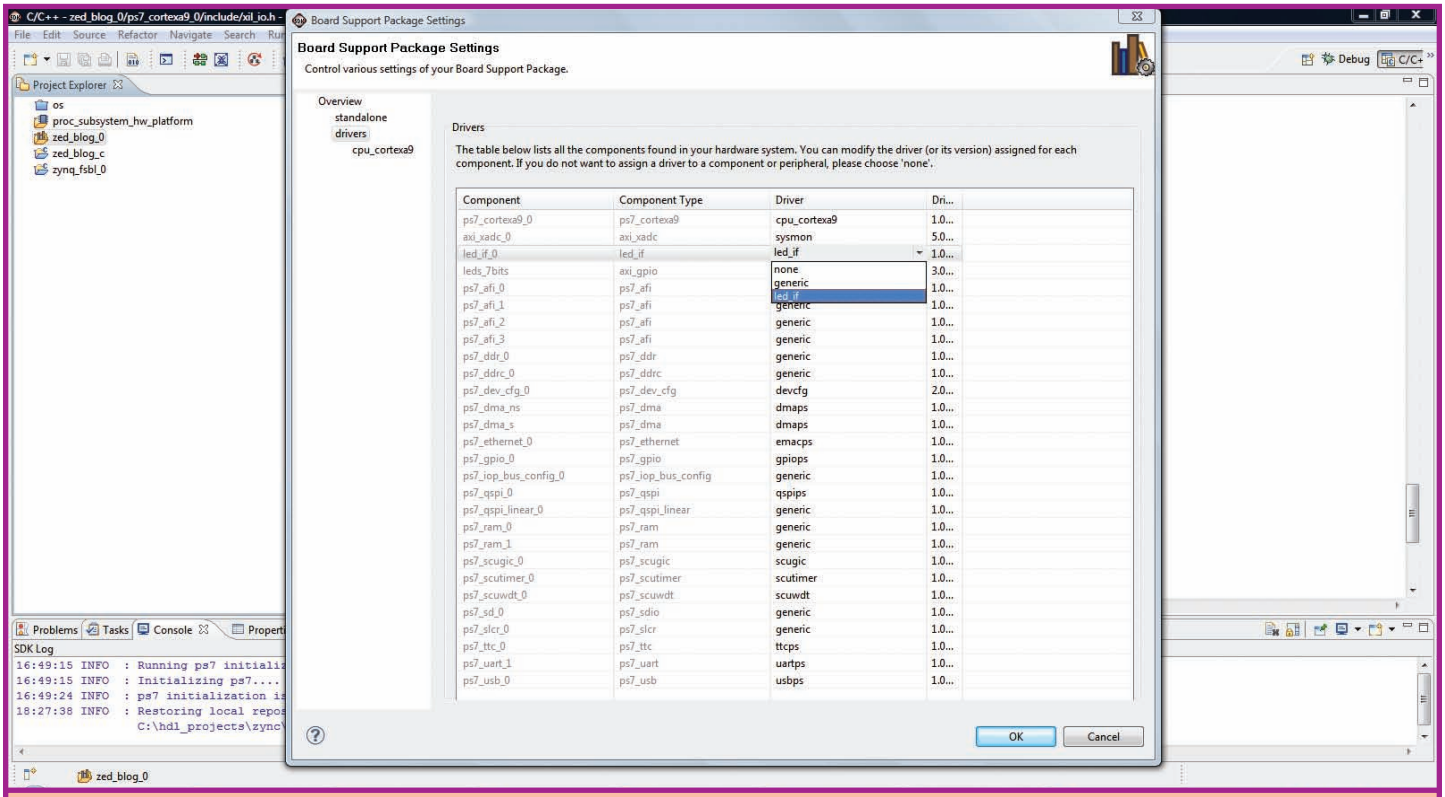


Figure 5 – Selecting the peripheral driver

driver next to it; in the first instance the driver will be named "generic." Using the peripheral within the SDK, the generic driver is not an issue. If you wish to use it, you will need to include the header file #include "xil_io.h." This allows you to use the function calls Xil_in32(addr) or Xil_out32(addr) to read and write from the peripheral.

To use this method you will also need to know the base address of the peripheral, which is given in xparameters.h, and the offset between registers. In this case, because of the 32-bit addressing in our example design, the registers are separated by 0x04. However, if you enabled the driver-creation option when you created the peripheral using XPS, then a separate driver will have been specially built for your peripheral. This driver too will be located within the drivers subdirectory of your PlanAhead sources directory.

You can use this type of specially built driver in place of the generic drivers within your design, but first you must set the software repositories to point to the directory to see the new drivers. You can define the repositories under the Xilinx Tools -> Repositories menu (Figure 4).

Once you have set the repositories to point to the correct directory, you can re-scan the repositories and close the dialog window. The next step is to update the board support package with the new driver selection. Open the board support package settings by right-clicking on the BSP icon under the Project Explorer. Click on the drivers tab and you will see a list of all the peripherals and their drivers. You can change the driver selection for your peripheral by selecting the appropriate driver from a list of drivers applicable to that device (Figure 5).

Select the driver for your peripheral by name and click OK; your project will be recompiled if you have automatic compilation selected. Once the code has been rebuilt, you are then nearly ready to communicate with the peripheral using the supplied drivers. Within your source code you will need to include the driver header file #include "led_if.h" and use the base address of the peripheral from xparameters.h to enable the interface to know the address within the memory space. It is then a simple case of calling the commands provided by the driver within the "include" file. In the case of our design, one example was

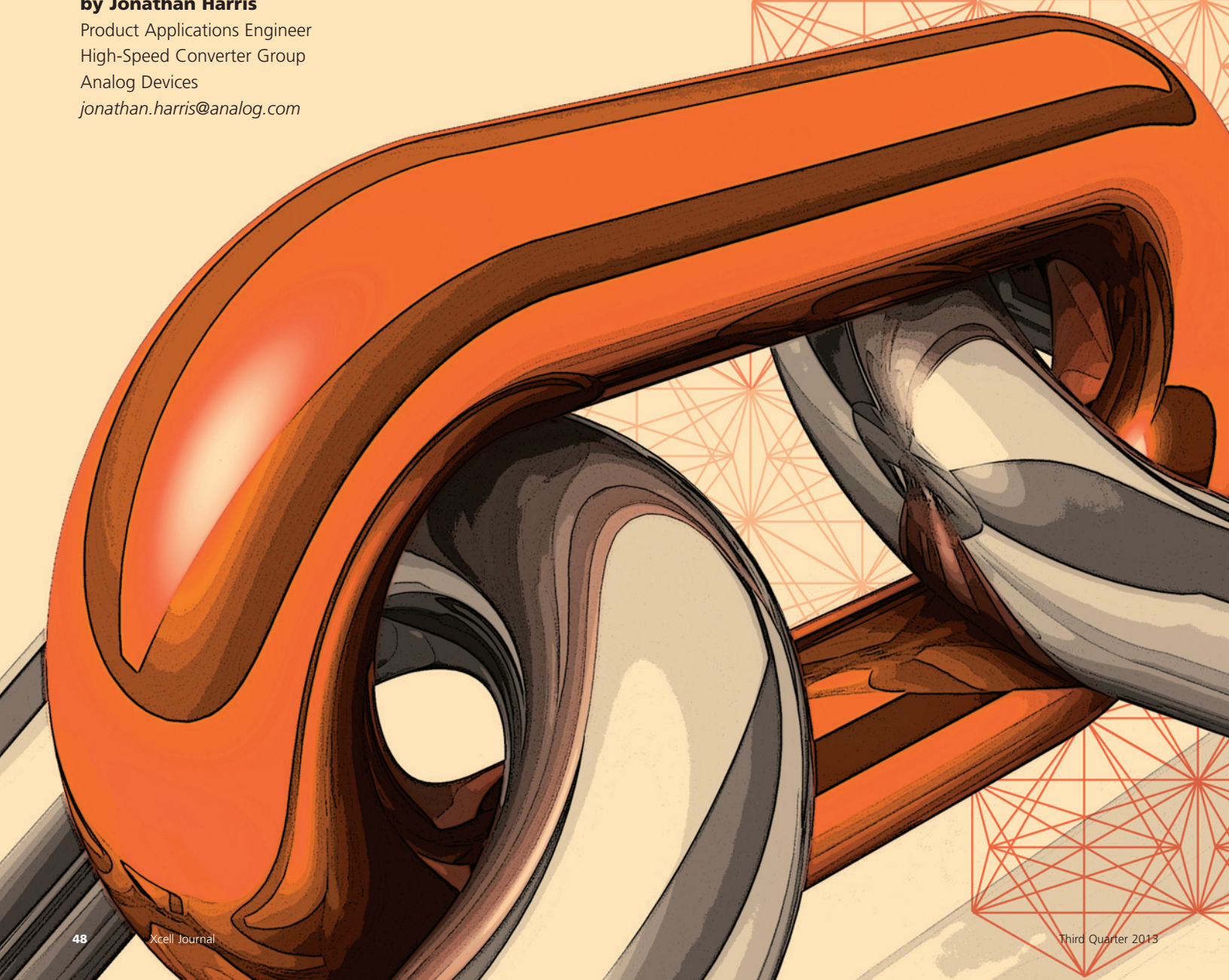
```
LED_IF_mWriteSlaveReg0(LED_
ADDR, LED_IF_SLV_REG0_OFF-
SET, 0x01);
```

With this command written, it is then time to test the software within the hardware environment and confirm that it functions as desired.

As you can see, making your own peripheral within a Zynq SoC design is not as complicated as you may at first think. Creation of a peripheral offers several benefits for your Zynq SoC processing system and, ultimately, your overall solution.

JESD204B: Critical Link in the FPGA-Converter Chain

by Jonathan Harris
Product Applications Engineer
High-Speed Converter Group
Analog Devices
jonathan.harris@analog.com



Data converters that interface with Xilinx FPGAs are fast embracing the new JESD204B high-speed serial link. To use this interface format and protocol, your design needs to take some basic hardware and timing issues into account.

In most designs today, parallel low-voltage differential signaling (LVDS) is the interface of choice between data converters and FPGAs. Some FPGA designers still use CMOS as the interface in designs with slower converters. However, the latest high-speed data converters and FPGAs are migrating from parallel LVDS and CMOS digital interfaces to a serialized interface called JESD204B, a new standard developed by the JEDEC Solid State Technology Association, an independent semiconductor engineering trade organization and standardization body.

As converter resolution and speed have increased, the demand for a more efficient interface has grown. This interface is the critical link between FPGAs and the analog-to-digital or digital-to-analog converters that reside beside them in most system designs. The JESD204B interface brings this efficiency to designers, and offers several advantages over preceding interface technologies. New FPGA designs employing JESD204B enjoy the benefits of a faster interface to keep pace with the faster sampling rates of converters. Also, a dramatic reduction in pin count leads to smaller package sizes and fewer trace routes, making board designs a little less complex.

All this does not come for free.

Each type of interface—including JESD204B—comes with design issues such as timing concerns. The specifics differ somewhat, but each interface brings its own set of parameters that designers must analyze properly to achieve acceptable system performance. There are also hardware choices to be made. For example, not all FPGAs and converters support the JESD204B interface. In addition, you must be up to date on parameters such as the lane data rate in order to select the appropriate FPGA.

Before exploring the details of the new JESD2904B interface, let's take a look at the other two options that designers have long used for FPGA-to-converter links: CMOS and LVDS.

CMOS INTERFACE

In converters with sample rates of less than 200 megasamples per second (MSPS), it is common to find that the interface to the FPGA is CMOS. A high-level look at a typical CMOS driver shows two transistors, one NMOS and one PMOS, connected between the power supply (V_{DD}) and ground, as shown in Figure 1a. This structure results in an inversion in the output, and to avoid it,

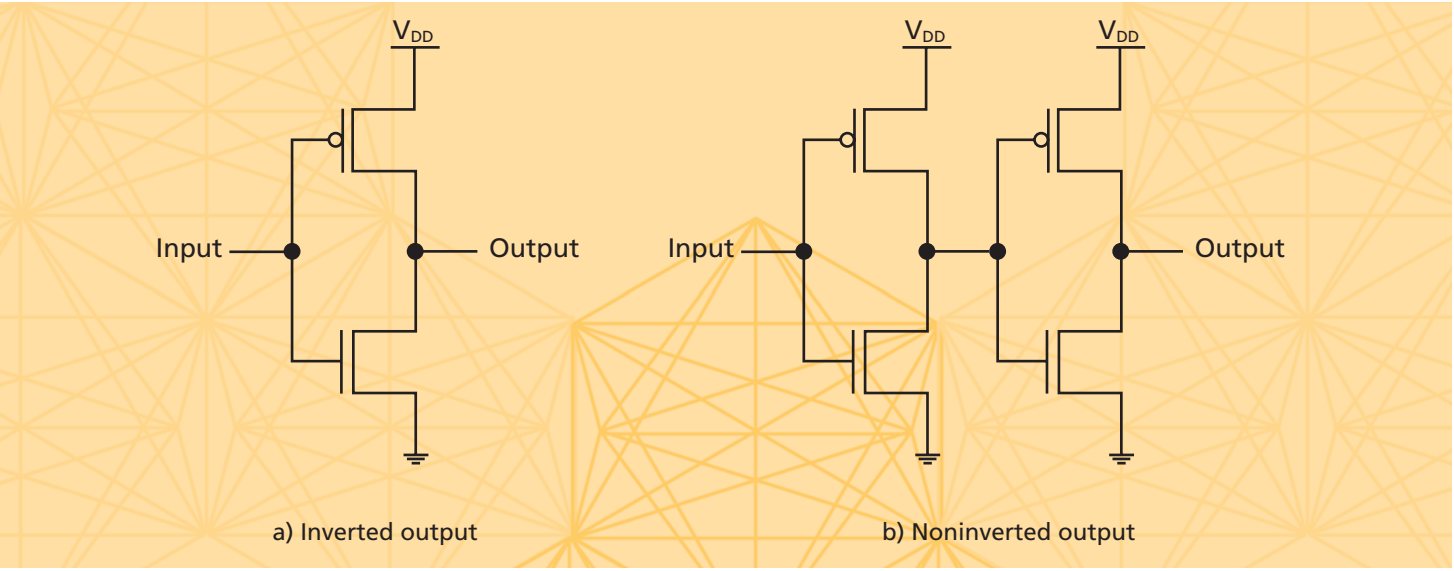


Figure 1 – Typical CMOS digital output driver

you can use the back-to-back structure in Figure 1b instead. The input of the CMOS output driver is high impedance while the output is low impedance. At the input to the driver, the gates of the two CMOS transistors present high impedance, which can range from kilohms to megohms. At the output of the driver, the impedance is governed by the drain current, I_D, which keeps the impedance to less than a few hundred ohms. The voltage levels for CMOS swing from approximately V_{DD} to ground and can therefore be quite large depending on the magnitude of V_{DD}.

Some important things to consider with CMOS are the typical switching speed of the logic levels (~1 V/ns), output loading (~10 pF/gate driven) and charging currents (~10 mA/output). It is important to minimize the charging current by using the smallest capacitive load possible. In addition, a damping resistor will minimize the charging current, as illustrated in Figure 2. It is important to minimize these currents because of how quickly they can add up. For example, a quad-channel 14-bit A/D converter could have a transient current as high as 14 x 4 x 10 mA, which would be a whopping 560 mA.

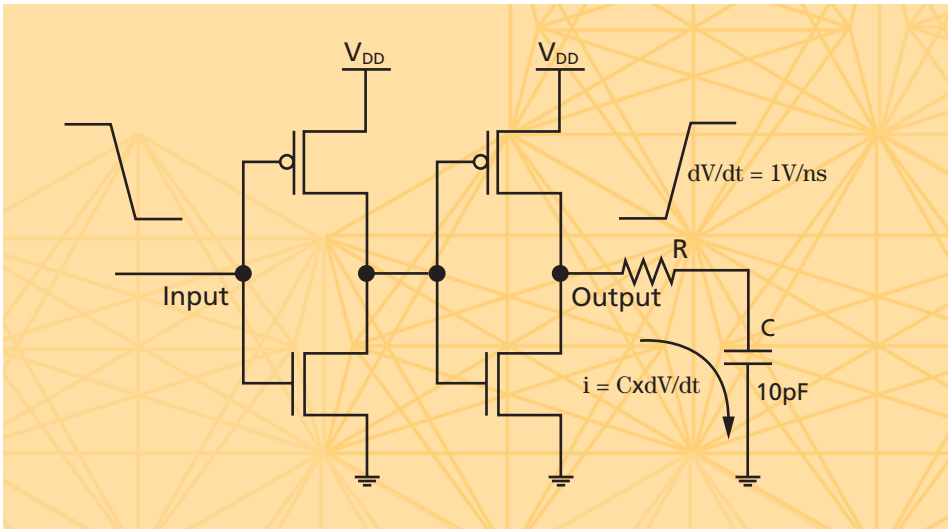


Figure 2 – CMOS output driver with transient-current and damping resistor

The series-damping resistors will help suppress this large transient current. This technique will reduce the noise that the transients generate in the outputs, and thus help prevent the outputs from generating additional noise and distortion in the A/D converter.

LVDS INTERFACE

LVDS offers some nice advantages over CMOS technology for the FPGA designer. The interface has a low-voltage differential signal of approximately 350 mV peak-to-peak. The lower-voltage swing has a faster switching time and reduces EMI concerns. In addition, many of the Xilinx® FPGA families, such as the Spartan®, Virtex®, Kintex® and Artix® devices, support LVDS. Also, by virtue of being differential, LVDS offers the benefit of common-mode rejection. This means that noise coupled to the signals tends to be common to both signal paths and is mostly canceled out by the differential receiver. In LVDS, the load resistance needs to be approximately 100 Ω and is usually achieved by a parallel termination resistor at the LVDS receiver. In addition, you must route the LVDS signals using controlled-impedance transmission lines. Figure 3 shows a high-level view of a typical LVDS output driver.

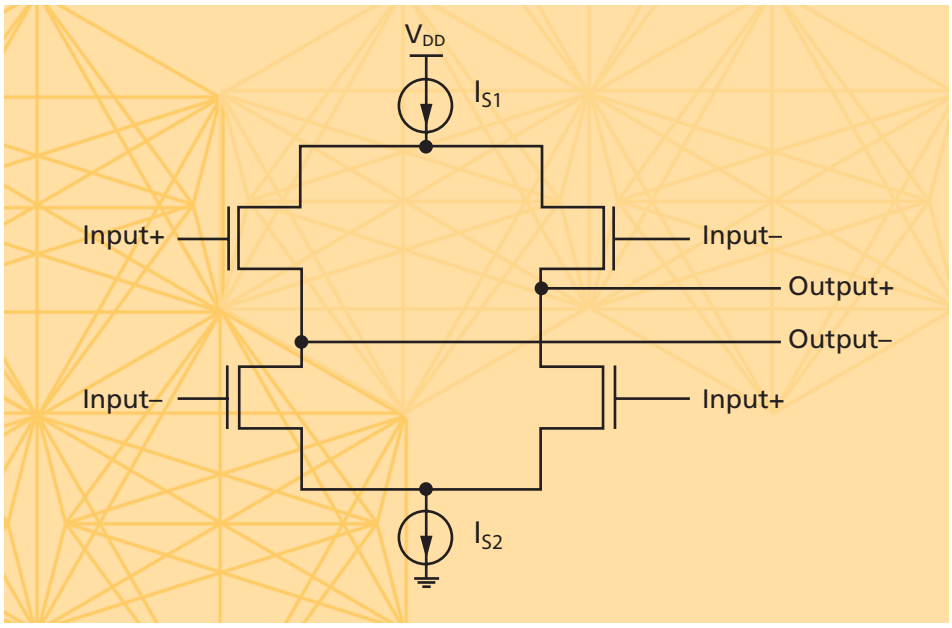


Figure 3 – Typical LVDS output driver

A critical item to consider with differential signals is proper terminations. Figure 4 shows a typical LVDS driver and the termination needed at the receivers. You can use a single differential termination resistor (R_{TDIFF}) or two single-ended termination resistors (R_{TSE}). Without proper terminations, the signal quality becomes degraded, which results in data being corrupted during transmission. In addition to having proper terminations, it is important to give attention to the physical layout of the transmission lines. You can maintain proper impedance by using line widths that are appropriate. The fabrication process variation should not cause wide shifts in the impedance. Avoid abrupt transitions in the spacing of the differential transmission lines in order to reduce the possibility of signal reflections.

JESD204B INTERFACE

The latest development in high-speed converters and FPGAs is the JESD204B serialized interface, which employs current-mode logic (CML) output drivers. Typically, converters with higher resolutions (≥14 bits), higher speeds (≥200 MSPS) and the desire for smaller packages with less power utilize CML drivers. There is some overlap here in high-speed converters around the 200-MSPS speed range that use LVDS. This means that designers moving to JESD204B can use a proven design that was likely based on a high-speed converter with LVDS. The result is to remove a level of complexity out of the design and to mitigate some of the risk.

Combining CML drivers with serialized JESD204B interfaces allows data rates on the converter outputs to go up to 12.5 Gbps. This speed allows room for converter sample rates to push into the gigasample (GSPS) range. In addition, the number of required output pins plummets dramatically. It's no longer necessary to route a separate clock signal, since the clock becomes embedded in the 8b/10b encoded data stream. Since the interface employed with CML drivers is typically serial,

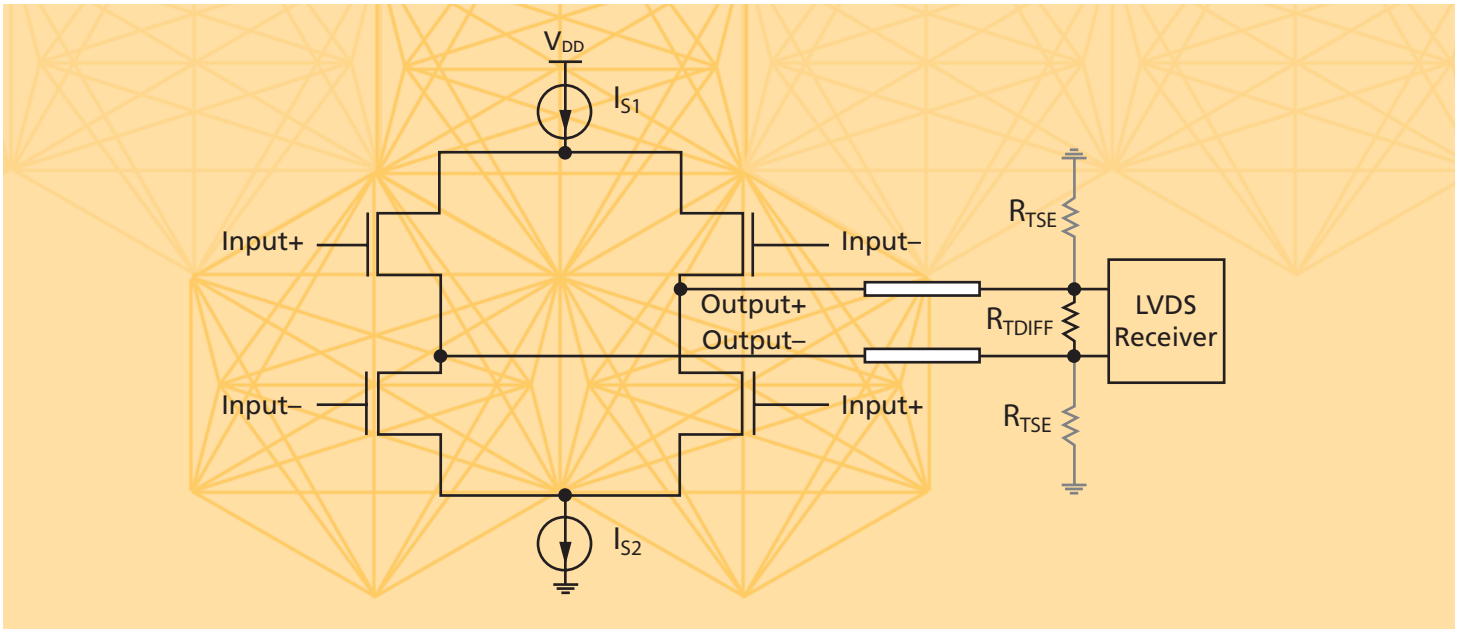


Figure 4 – LVDS output driver with termination

The JESD204B serialized data interface offers a better and faster way to transmit data from converters to receiver devices. The latest converters and FPGAs are being designed with drivers to support JESD204B.

there's a much smaller increase in the number of pins required than would be the case with CMOS or LVDS.

Table 1 illustrates the pin counts for the three different interfaces in a 200-MSPS converter with various channel counts and bit resolutions. The data assumes a synchronization clock for each channel's data, in the case of the CMOS and LVDS outputs, and a maximum data rate of 4 Gbps for data transfer using JESD204B. This lane rate is well under the maximum limits of many of the FPGAs in the Virtex-6, Virtex-7, Kintex-7 and Artix-7 families available today. The reasons for the progression to JESD204B become obvious when you note the dramatic reduction in pin count that you can achieve. The board routing complexity is reduced and less I/O is required from the FPGA, freeing up those resources for other system functions.

Taking this trend one step further and looking at a 12-bit converter running at 1 GSPS, the advantage of using JESD204B becomes even more apparent. This example omits CMOS, because it is just completely impractical to try to use a CMOS output interface with a gigasample converter. Also, the number of converter channels is limited to four and the lane rate is limited to 5 Gbps, which is again compatible with many of the FPGAs in the Virtex-6 and Xilinx 7 series FPGA families. As you can see, JESD204B can significantly reduce the complexity in the output routing due to the reduction in the number of output pins. There is nearly a 6x reduction in pin count for a quad-channel converter (Table 2).

Just as with the LVDS interface, it is important to maintain proper termination impedances and transmission-line impedances when using CML with the JESD204B interface. One method of measuring signal quality is the eye diagram. An eye diagram is a measurement that indicates several parameters about the signal on a link. Figure 5 shows an eye diagram for a properly terminated CML driver on a 5-Gbps JESD204 link. The eye diagram exhibits nice transitions and has a sufficiently open eye such that the receiver in the FPGA should have no difficultly interpreting the data. Figure 6 shows the result of having an improperly terminated CML driver in the same 5-Gbps JESD204 link. The eye diagram has more jitter, the amplitude is reduced and the eye is closing, in which case the receiver in the FPGA will have a difficult time interpreting the data. As you can see, it is very important to use proper terminations to ensure that you achieve the best possible signal integrity.

A BETTER AND FASTER WAY

The three main types of digital outputs employed in converters each have their own advantages and disadvantages when connecting to FPGAs. It is important to keep these pros and cons in mind when utilizing converters that employ CMOS, LVDS or CML (JESD204B) output drivers with an FPGA design. These interfaces are a critical link between the FPGAs and the A/D or D/A converters. Each type of driver has qualities and

Number of Channels	Resolution	CMOS Pin Count	LVDS Pin Count (DDR)	CML Pin Count (JESD204B)
1	14	13	14	4
2	14	26	28	4
4	14	52	56	6
8	14	104	112	6

Table 1 – Pin count comparison for a 200-MSPS A/D converter using CMOS, LVDS and JESD204B

Number of Channels	Resolution	LVDS Pin Count (DDR)	CML Pin Count (JESD204B)
1	14	14	4
2	14	28	4
4	14	56	6
8	14	112	6

Table 2 – Pin count comparison between LVDS and JESD204B for a 1-GSPS converter

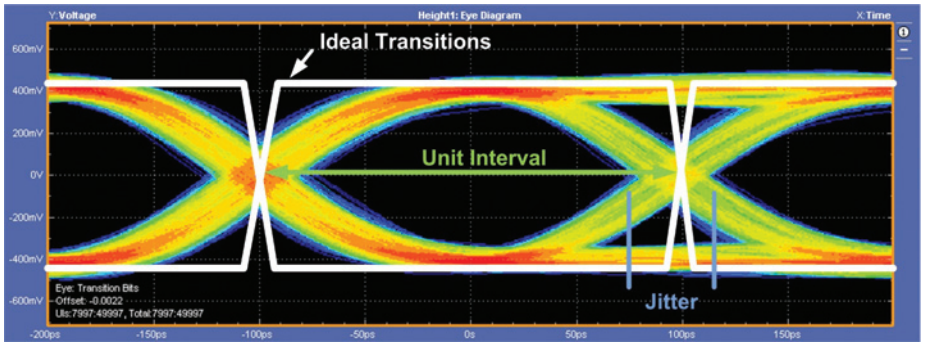


Figure 5 – Eye diagram (5 Gbps) with proper terminations

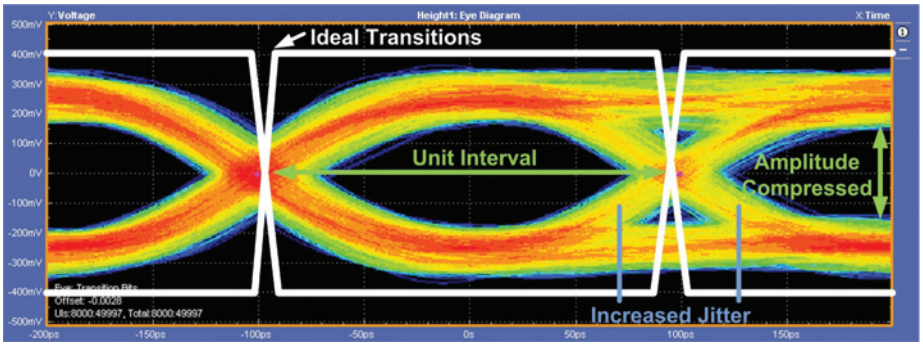


Figure 6 – Eye diagram (5 Gbps) with improper terminations

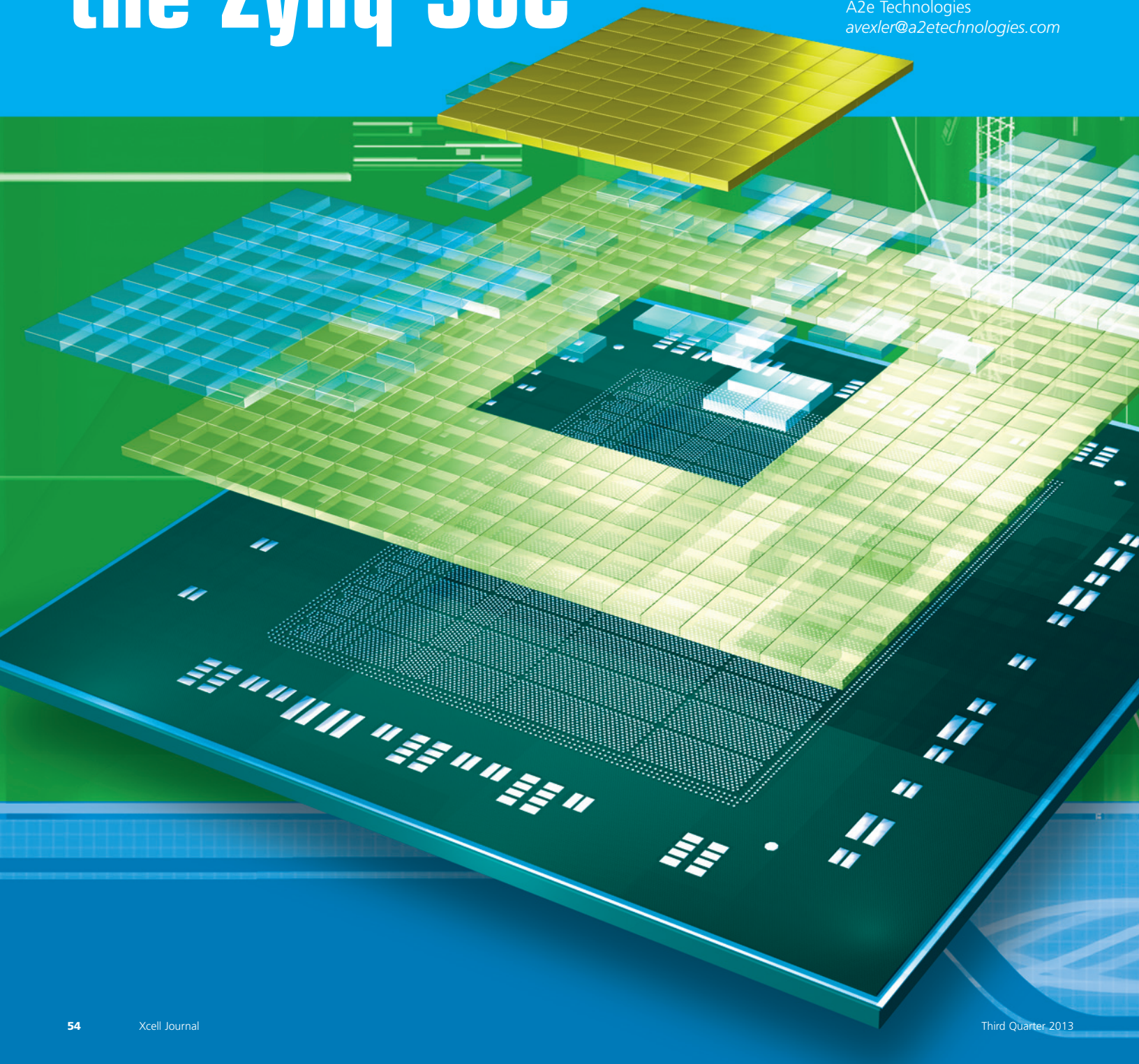
requirements that you must take into account when designing a system, such that the receiver in the FPGA can properly capture the converter data. It is important to understand the loads that must be driven; to utilize the correct terminations where appropriate; and to employ proper layout techniques for the different types of digital outputs the converters use. As the speed and resolution of converters have increased, so has the demand for a more efficient digital interface. As this happens, it becomes even more important to have a properly designed system using optimal layout techniques.

The JESD204B serialized data interface offers a better and faster way to transmit data from converters to receiver devices. The latest converters and FPGAs are being designed with drivers to support JESD204B. The need begins with the system-level demand for more bandwidth, resulting in faster converters with higher sample rates. In turn, the higher sample rates drive the requirement for higher output data rates, which has led to the development of JESD204B. As system designs become more complex and converter performance pushes higher, the JESD204 standard is poised to adapt and evolve to continue to meet the new design requirements. This ability of the JESD204 standard to evolve will increase the use of the interface, eventually propelling it into the role of default interface of choice for converters and FPGAs.

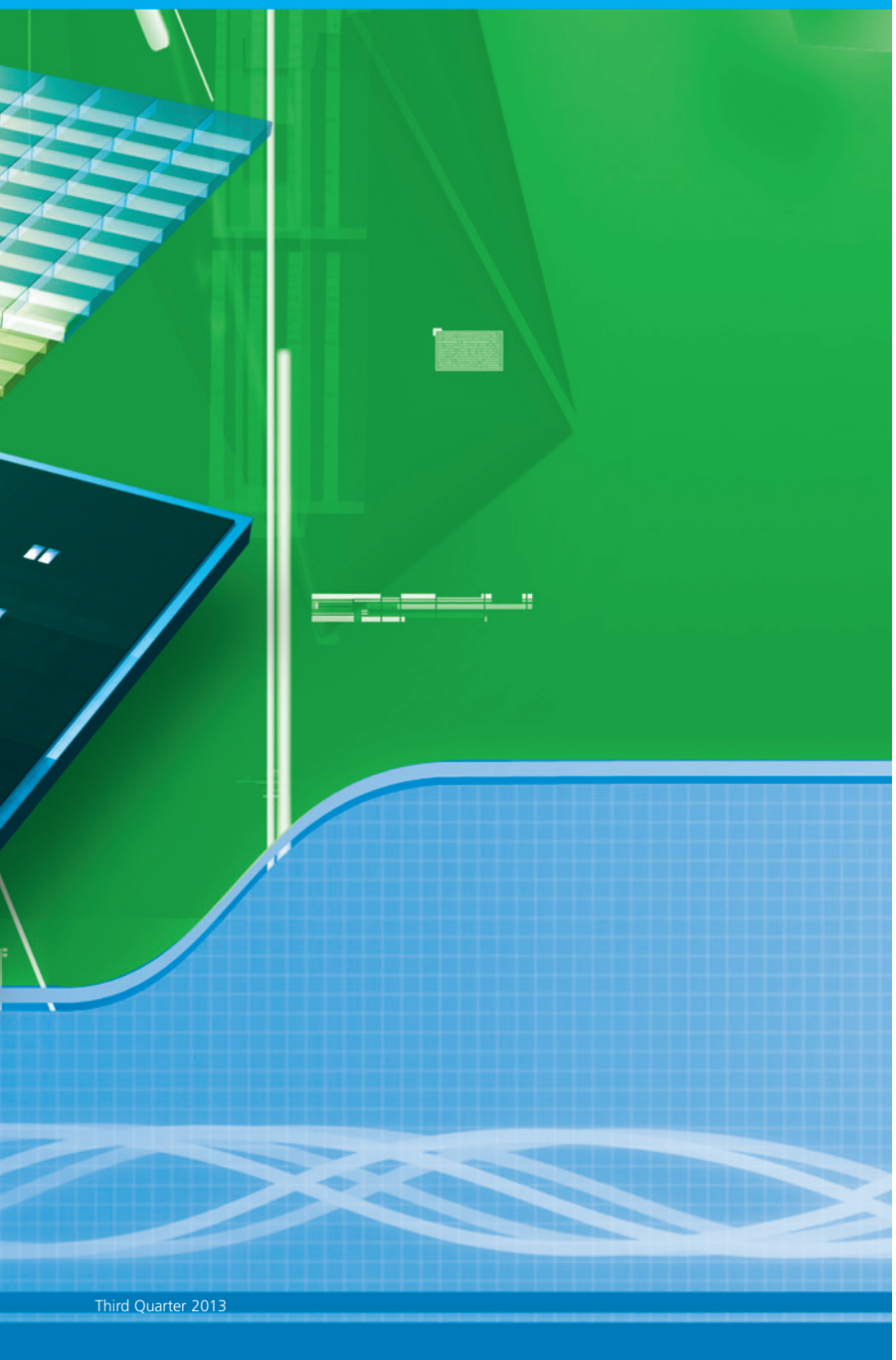
For more information, see the "JESD204B Survival Guide" at *http://www.analog.com/static/imported-files/tech_articles/JESD204B-Survival-Guide.pdf*. Also, Xilinx and Analog Devices offer a three-part YouTube series on rapid prototyping with JESD204B and FPGA platforms at *http://www.youtube.com/playlist?list=PLiwaj4qabLWxRUOHKH2aGMH7kkNr0euo3*

Designing a Low-Latency H.264 System Using the Zynq SoC

by Allen Vexler
CTO
A2e Technologies
avexler@a2etechnologies.com



Micro-footprint H.264 cores combine with Xilinx's Zynq SoC in a small, fast, streaming-video system.



To create an IP-based streaming-video system with a small PCB footprint, designers historically have had to choose between the inflexible architecture of an ASSP or a larger, more flexible system based on an FPGA-microprocessor combo. Placing a soft-core microprocessor inside the FPGA could eliminate the need for a separate processor and DRAM, but the performance of the final system might not match that of a solution built around an external ARM® processor that would also contain USB, Ethernet and other useful peripherals. With the advent of the Xilinx® Zynq®-7000 All Programmable SoC along with small-footprint H.264 cores, it's now possible to construct a system in a very small-form-factor PCB with only one bank of DRAM, but with the power of dual ARM cores and high-speed peripherals that are all interconnected with multiple high-speed AXI4 buses (see Figure 1).

Although H.264 cores for FPGAs have existed for quite some time, up until now none have been fast enough and small enough to convert 1080p30 video and still fit into a small, low-cost device. Using the latest micro-footprint H.264 cores from A2e Technologies in tandem with the Zynq SoC, it's possible to build a low-latency system that can encode and decode multiple streams of video from 720p to 4K at variable frame rates from 15 to 60 frames per second (fps). Placing an A2e Technologies H.264 core in a Zynq SoC device allows for a significant reduction in board space and component count, while the dual ARM cores in the Zynq SoC remove the need for a separate microprocessor and the memory bank to which it must be attached.

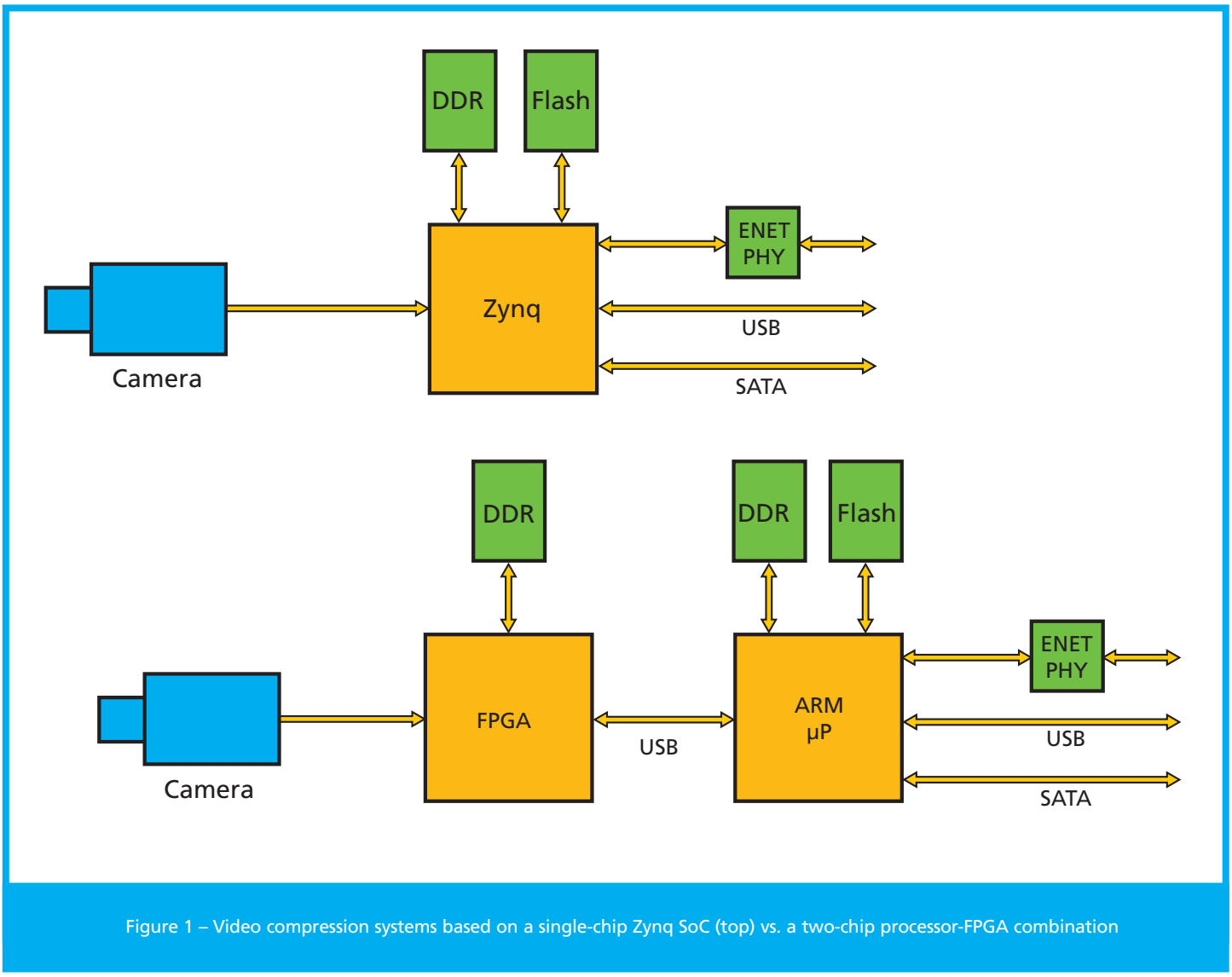


Figure 1 – Video compression systems based on a single-chip Zynq SoC (top) vs. a two-chip processor-FPGA combination

The result is a complete streaming-video system, built around Linux drivers and a Real Time Streaming Protocol (RTSP) server, that is capable of compressing 1080p30 video from two cameras with an end-to-end latency of approximately 10 milliseconds. The A2e Technologies H.264 core is available in encode-only and encode/decode versions. Additionally, there is a low-latency version that reduces encode latency to less than 5 ms. The 1080p30 encode-only version requires 10,000 lookup tables (LUTs), or roughly 25 percent of a Zynq Z7020's FPGA fabric, while the encode/decode version requires 11K LUTs.

SOC-FPGA BASED SYSTEMS

Products based on the Zynq SoC provide significantly more flexibility than those built using an application-specific standard product (ASSP). For example, it's easy to construct a system supporting four 1080p30 inputs by placing four H.264 cores in the FPGA fabric and hooking each camera input to the FPGA. Many ASSPs have only two inputs, forcing designers to figure out how to multiplex several streams into one input. Each A2e Technologies H.264 core is capable of handling six VGA-resolution cameras or one camera with 1080p30 resolution. So, it would be possible to construct a two-core system that could compress video from 12 VGA cameras.

ASSPs typically provide for an on-screen display feature only on the decoded-video side, forcing designers either to send the OSD information as metadata or to use the ARM core to time video frames and programmatically write the OSD data to the video buffer. In an FPGA, adding OSD before video compression is as simple as dropping in a block of IP. You can place additional processing blocks such as fisheye-lens correction before the compression engine with relative ease. FPGAs also allow for in-field, future upgrades of features, such as adding H.265 compres-

sion. Figure 2 is a block diagram of an H.264 compression engine with input from two 1080p30 cameras, with OSD applied to the uncompressed image.

MANAGING LATENCY

Certain applications, such as control of remotely piloted vehicles (RPVs), are based upon feedback from streaming images coming from the remote device. In order to control a remote device, the latency between the time when the sensor sends the video to the compression engine and the point when the decoded image is displayed (referred to as "glass to glass") typically needs to be less than 100 ms. Some designers target numbers in the 50-ms range. Latency is the sum the following:

- Video-processing time (ISP, fisheye-lens correction, etc.)
- Delay to fill frame buffer
- Compression time
- Software delay associated with sending out a packet
- Network delay
- Software delay associated with receiving a packet
- Video decode time

Many systems encode video using hardware but end up decoding it using a standard video player such as VLC running on a PC. Even though the buffer delay for media players can be reduced from the typical 500 to 1,000 ms to something significantly less, the latency is still far greater than 50 ms. To truly control latency and keep it to an absolute minimum, hardware decode of the compressed video stream is required, along with minimal buffering.

Latency for H.264 encoders is typically specified in frames, as a full frame must typically be buffered before compression begins. Assuming you can speed up your encoder, you could decrease the encode latency simply by doubling the frame rate—that is, one frame of latency for 30 fps is 33 ms vs. one frame of latency for 60 fps is 16.5

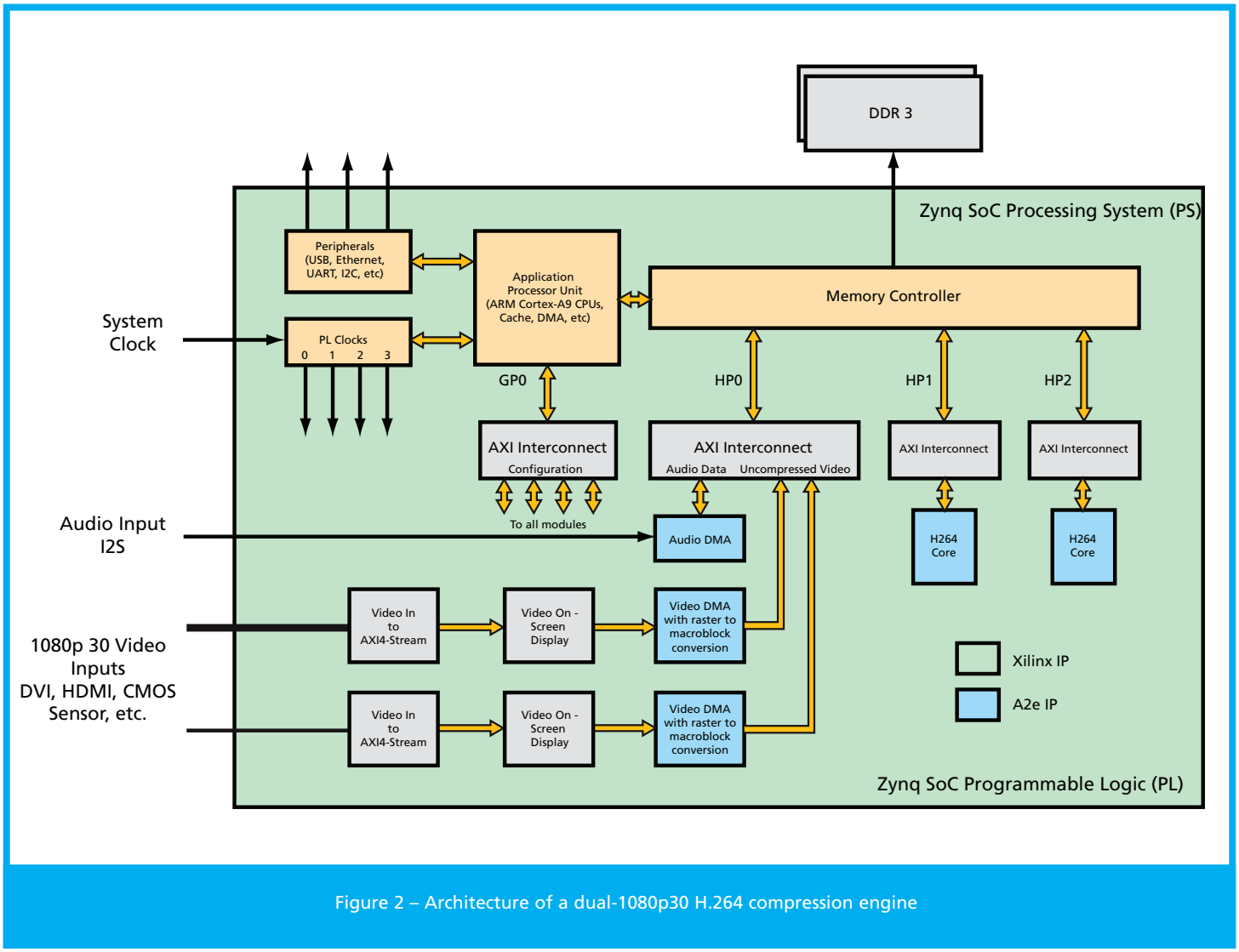


Figure 2 – Architecture of a dual-1080p30 H.264 compression engine

A typical streaming-video system uses an RTSP server to establish a streaming-video connection between a camera and the client (decoding/recording) device. The RTSP server will stream the compressed video to the client for display or storage.

ms. Many times this option is not available due to performance limitations of the camera and encoder. Thus, the solution is an encoder specifically designed for low latency. In the new A2e Technologies low-latency encoder, only 16 video lines need to be buffered up before compression starts. For a 1080p30 video stream, the latency is less than 500 µs. For a 480p30 video stream the latency is less than 1 ms. This low-latency encoder will allow designers to construct systems with low and predictable latency.

In order to minimize total latency, you must also minimize delays introduced by buffers, network stacks and RTSP servers/clients on both the encoding and decoding sides, as it is pointless to use a low-latency encoder with a software path that introduces a large amount of latency. An RTSP server typically is used to establish a streaming-video connection between a server (camera) and the client (decoding/recording) device. After a connection is established, the RTSP server will stream the compressed video to the client for display or storage.

MINIMIZING LATENCY

Typically, the software components in the server and client are designed only to keep up with the bandwidth required to stream compressed video, nor minimize latency. With a non-real-time operating system like Linux, it can be difficult to guarantee latency. The typical solution is to create a low-latency, custom protocol for both the server and client. However, this approach has the downside of not being compatible with industry standards. Another approach is to take a standard like RTSP and make modifications at the lower levels of the software to minimize latency, while at the same time retaining compliance with standards.

However, you can also take steps to minimize copies between kernel and user space, and the delays associated with them. To reduce latency though

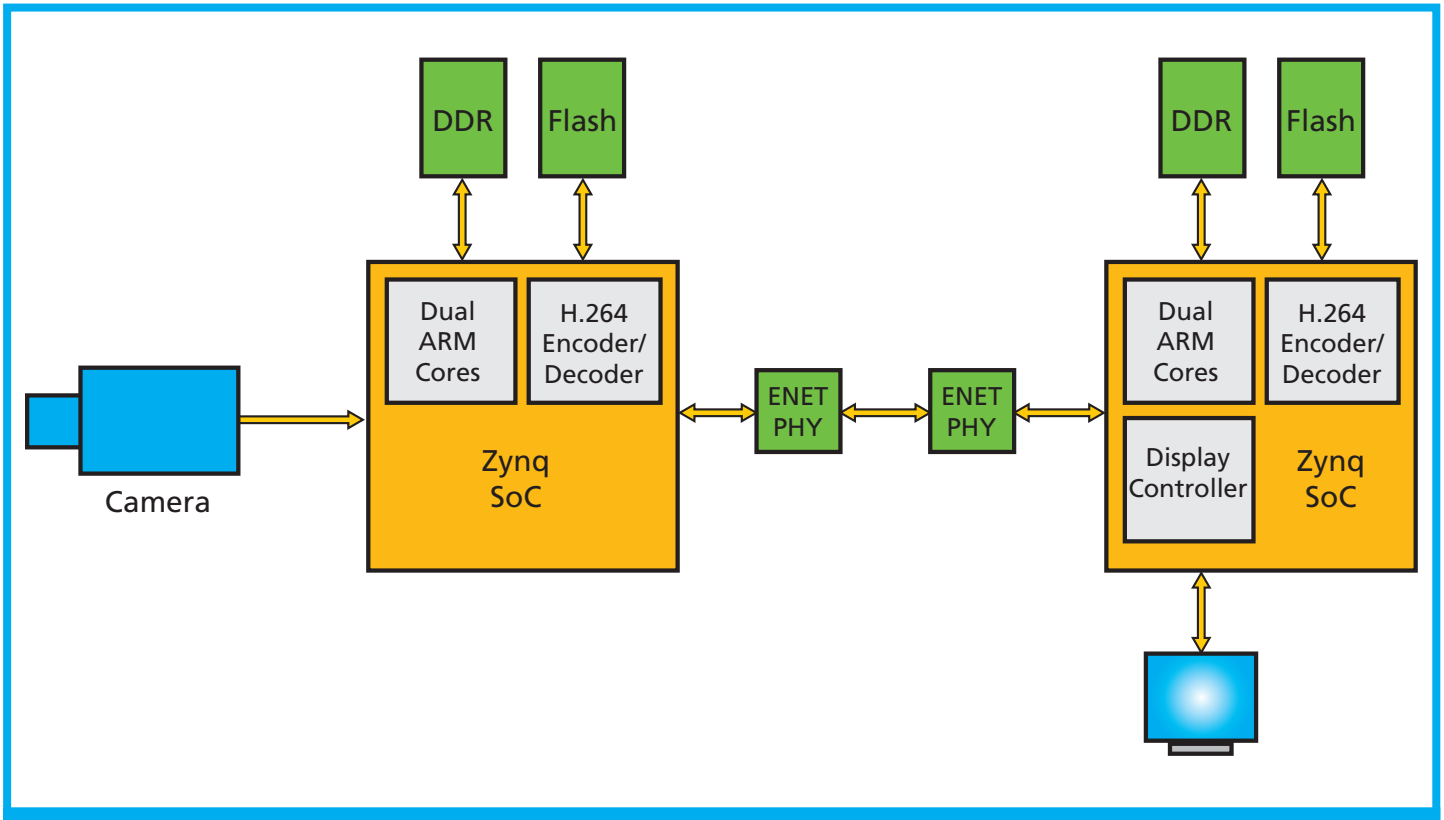


Figure 3 – Block diagram of complete encode-decode system

the whole software path, you will need to decouple the RTSP server and compressed data forwarding so that the Linux driver, not the RTSP server, performs the forwarding.

In the A2e Technologies low-latency RTSP server, we have made two major changes to reduce latency. First, we removed the RTSP server from the forwarding path. The RTSP server continues to maintain statistics using the Real Time Control Protocol (RTCP), and periodically (or asynchronously) updates the kernel drive with the changes in the network destination (i.e., destination IP or MAC address). Second, the kernel driver attaches the necessary headers (based on the information provided by the RTSP server) and immediately forwards the packet by directly injecting it into the network driver (for example, udp_send), thus eliminating the memory copy to/from user space.

Figure 3 shows a complete H.264 IP-based encode/decode system with a combined latency of less than 50 ms. The system is based upon the Zynq SoC, the A2e Technologies low-latency H.264 encoder-decoder and the A2e Technologies low-latency RTSP server/client. Note that the only real difference from a hardware perspective between the encode and decode systems is that the encode side must interface to a camera/sensor and the decode side must be able to drive a flat-panel display. You can easily design a board that has all required hardware features needed for use in both encode and decode.

In order to minimize video compression/decompression latency for real-time control applications, designers need special encoders and optimized software. Utilizing the Xilinx Zynq SoC and the A2e Technologies low-latency H.264 encoder and optimized RTSP server/client, you can create a highly configurable system with extremely low latency in a small PCB footprint.

For more information, visit *http://www.xilinx.com/products/intellectual-property/1-1LK9J1.htm* and *http://www.a2etechnologies.com*.

Innovative Power Design Maximizes Virtex-7 Transceiver Performance

by Takayuki Mizutori
Field Application Engineer
Bellnix Co. Ltd.
mizutori@bellnix.co.jp

Designers must consider voltage rails, voltage levels and high-current requirements when planning their power systems. The point-of-load converter will be a key component.

With semiconductor companies releasing ever-more-advanced devices, circuit designers are constantly challenged to figure out the best ways to ensure their designs run at maximum performance so their systems can reap all the benefits of these new ICs. An often overlooked but critical step to getting the best performance out of the most advanced FPGAs is to create a robust and reliable power supply in your system.

The Xilinx® 7 series FPGAs provide unprecedented low power, high performance and an easy path for migrating designs. This innovative family employs a 28-nanometer TSMC HPL (high-performance, low-power) process and offers a 50 percent reduction in power consumption compared with competing FPGAs.

Yet as FPGAs increase in capacity and performance, FPGA I/O resources are becoming a major performance bottleneck. While transistor counts effectively double every year, the size of an FPGA die stays more or less the same. This is problematic for the I/O carrying signals on and off the chip, as I/O can only increase linearly, which presents a slew of layout, signal integrity and power challenges for systems designers trying to create higher-bandwidth systems. To overcome the issue, Xilinx employs highly sophisticated, multigigabit transceiver (MGT) I/O modules in its state-of-the-art devices. The Virtex®-7 HT FPGA, for example, provides a maximum bandwidth of 2.78 Tbps, using 96 sets of 13.1-Gbps transceivers.

As FPGAs get more sophisticated, the power supply requirements have also become more complex. For today's designs, teams must consider multiple voltage rails corresponding to each circuit and functional block, a plurality of voltage levels, as well as high-current requirements. Let's examine the power requirements associated with the Virtex-7 FPGAs in greater depth.

ACCURACY IN OUTPUT VOLTAGE SETTING

With the evolution of semiconductor manufacturing process technology, the density and performance of semiconductors, especially FPGAs, have improved greatly. At the same time, semiconductor companies have been reducing the core voltages of their devices. Both factors complicate power supply design and make it hard for designers to select the right power circuitry. Board layout and manufacturing technologies are not sufficient to manage voltage within a few millivolts, but the power systems must be measurably accurate and reliable. As a result, it is imperative for design teams to take power supply characteristics and features as well as the device's characteristics into consideration when designing their power systems. For example, a core voltage of 1 V has a tolerance of +/-30 mV. When a current changes by 10 A, a 50-mV voltage drop occurs with a 5-milliohm power line (see Figure 1). As such, it is not easy to manage a power supply voltage within the tolerable values in all the environments in your board designs. In addition, if design teams are using a point-of-load (POL) power supply, they need to be extremely accurate, within +/-1 percent, and have a high-speed transient response. For these and other reasons, design teams should thoroughly investigate the design and options available to them before building their boards.

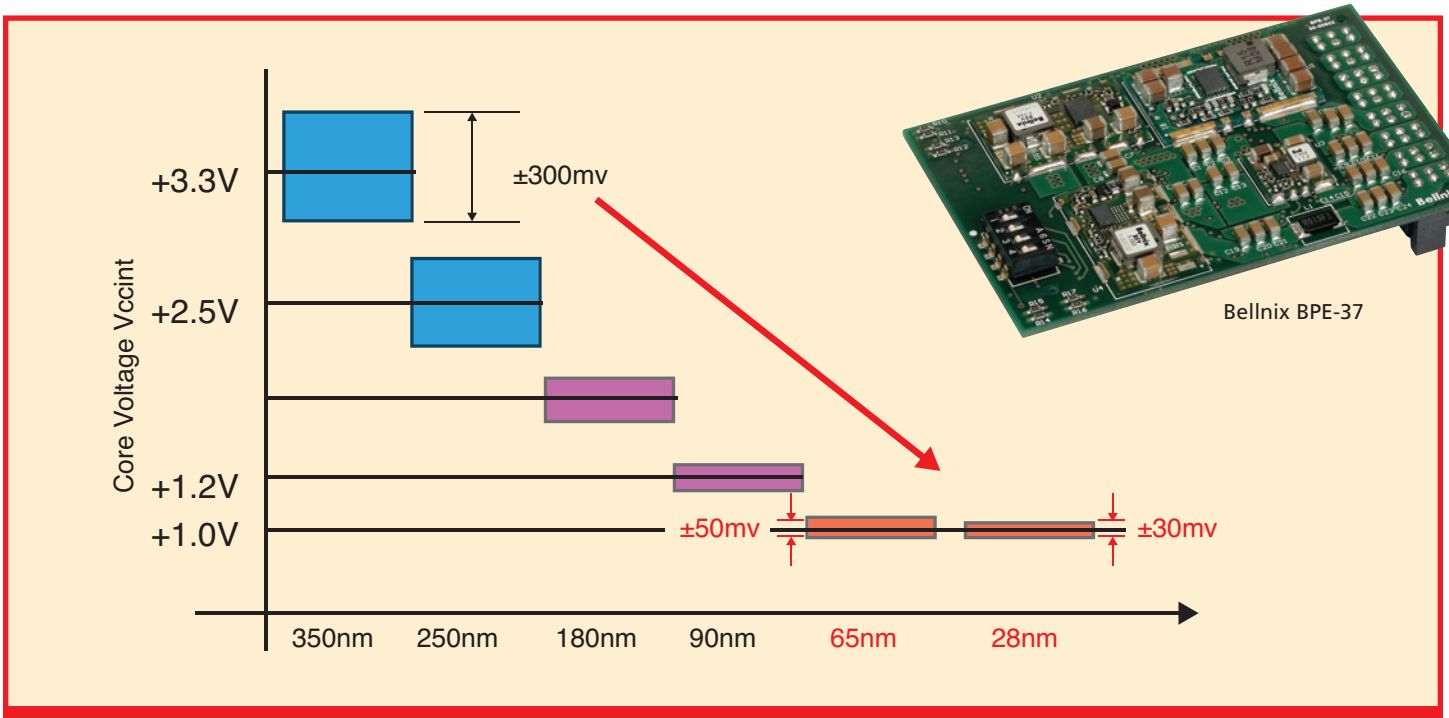


Figure 1 – Evolution of the process and tolerance of the core voltage

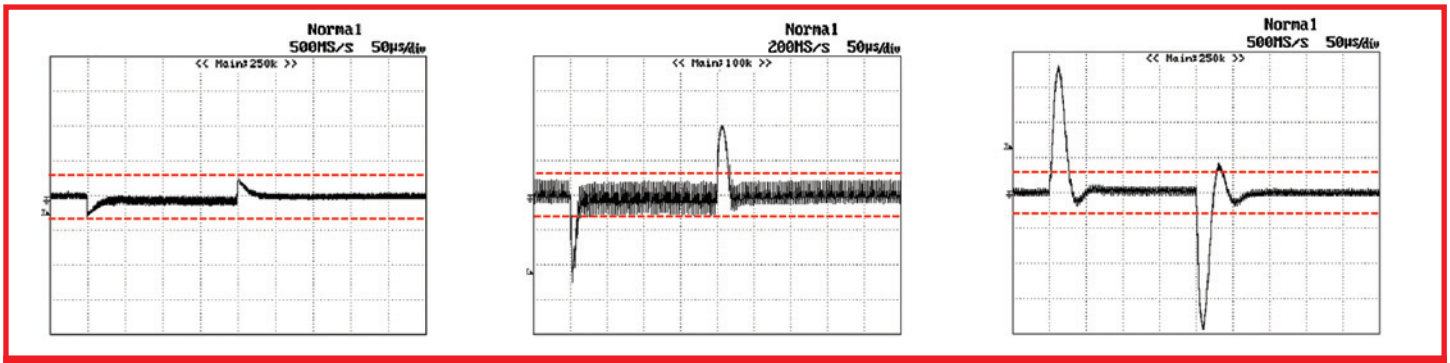


Figure 2 – Examples of transient response in competing point-of-load converters; results for the Bellnix POL converter are at left.

One of the first things design teams need to be aware of when selecting circuitry for their power system is that FPGA designs have strict sequencing requirements for startup. For example, the Virtex-7 FPGA power-on sequence requires one of two choices: VCCINT_VMGTAVCC_VMGTAVTT or VMGTAVCC_VCCINT_VMGTAVTT. Meanwhile, the power-off sequence is the reverse of whichever of these power-on sequences you chose. If you power the device on or off in any sequence other than the ones recommended, the slack current from VMGTAVTT could be more than the specification during power-up and power-down. To satisfy such sequencing rules, designers need to use either a specialized IC for sequencing or the P-Good and on/off features of POLs.

A power system for a design using an FPGA also needs to include circuitry that allows the team to monitor output voltages. Typically, designers check the output voltage during a product's fine-tuning stage and trim the output voltage to manage printed pattern drop voltages. Designers typically trim the voltage with POLs, but of course this means that the POLs you use in your design need to have a feature for output-voltage adjustment. The power system also needs to allow you to monitor your design's voltage and current to check for current time deviations and power malfunctions, and to better predict failures.

High-speed serdes require a high current because they have several channels and a high operating frequency. The low-dropout (LDO) regulators that engineers use for this purpose generate a lot of heat due to poor efficiency and thus are not an ideal power solution for circuits that consume high current. High-speed serdes are quite sensitive to the jitter caused by noise in power circuitry. High jitter will corrupt signals and cause data errors. It is also very important for serdes to have very good signal integrity so as to achieve good high-speed data communications. This of course means that when choosing a POL, designers need to ensure their POL isn't noisy.

SELECTING A HIGH-ACCURACY POL CONVERTER

A crucial step in building a proper power system for a Virtex-7 FPGA is ensuring the POL converters meet a given device's requirements. Fast-transient-response POL converters excel in designs where the current is high and shifts in current are large. Not only are these high-accuracy POL converters with fast transient responses helpful in managing the FPGA's shifting core voltage. They are also useful for managing the VMGTAVCC and VMGTAVTT of an MGT. Figure 2 shows a measurement of voltage response for the dynamic-load change of several POL converters. To keep things fair, we have used the same conditions for all the POL converters. The only exception is that we used the capacitor that each POL converter's manufacturer recommends in their literature.

- Input voltage: 5 V fixed
- Output voltage: 1 V fixed (Virtex-7 core voltage)
- Load current: 0~10 A
- Current change slew rate (current change rate per unit of time): 5 A/µs

Streaming data and numeric processing create changes in dynamic current in FPGAs. Our results show that some POL converters undershoot and overshoot the output voltages, getting beyond 1 V ±30 mV, which is the VCCINT and VMGTAVCC voltage that Xilinx recommends for Virtex-7 FPGAs. Of course, the number might vary depending on usage, board layout and decoupling capacitors surrounding the POLs and the FPGAs. However, it is clear that we need a very good power supply to keep up with the recommended operating-voltage range of the Virtex-7 FPGAs, where large current changes (10 A and 5 A/µs) occur rapidly. The Bellnix BSV series fits these needs and also has a fast transient response, necessary for high-end FPGAs and other high-speed devices, such as discrete DSPs and CPUs.

PROGRAM FOR OUTPUT VOLTAGE AND SEQUENCING

As described above, as FPGA process technologies shrink, their supply voltage gets lower but the number of functions increases, as does their performance. This means the current changes

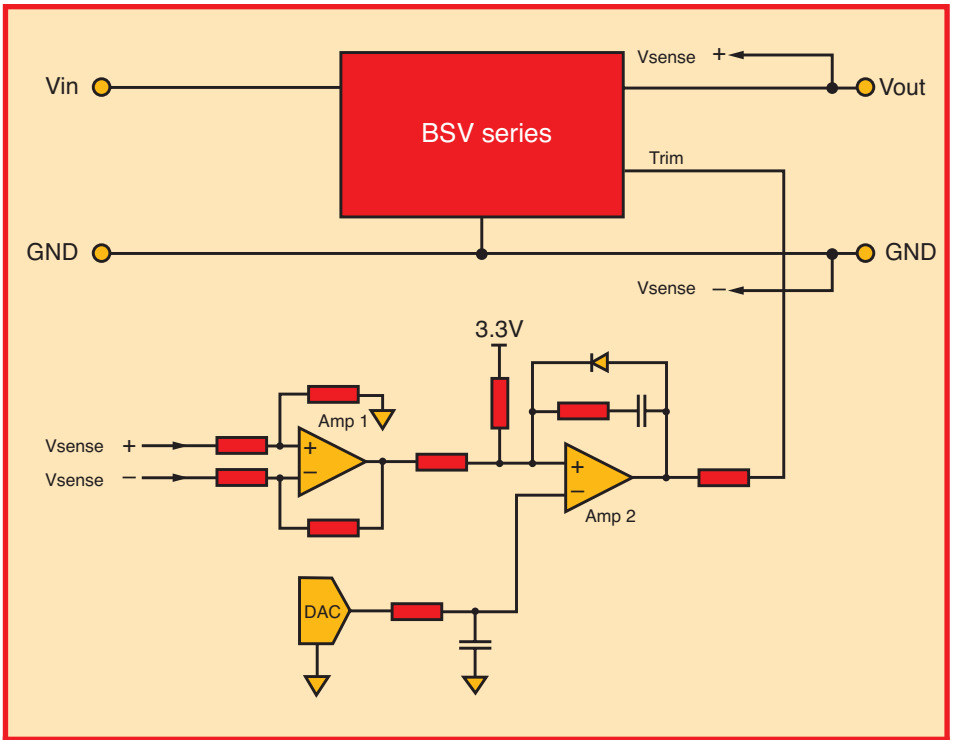


Figure 3 – How to make a BSV-series POL into programmable output voltage

rapidly, increasing power requirements. It is getting more difficult for design engineers to manage voltage within the allowed tolerance that FPGAs require. This is true not only for the core voltage of the FPGA but also for the FPGA's high-speed serdes. There are several important points related to power supplies you need to consider to get the maximum performance from a Virtex-7 transceiver. For example, you need to adjust the POL output voltage when a voltage drop occurs due to power line resistance. Margining testing likewise requires an output-voltage adjustment. Designers also need to monitor and trim the output voltage of the POL to evaluate systems quickly.

Here is a method to adjust the output voltage with a CPU program to satisfy Virtex-7 power requirements. You can use the circuit diagram in Figure 3 to make a BSV-series POL into programmable output voltage.

In the figure, Amp 1 detects the output voltage. You can control the output voltage with high accuracy by remote sensing with a differential amplifier and cancel out the voltage drop by connecting the POL converter to the load. Amp 2 compares the output voltage detected by Amp 1 with the D/A converter's output voltage. We connected Amp 2's output to the POL converter's trim pin. The output voltage of the POL converter changes with the trim pin's input voltage. Thus, the voltage of the load and that of the D/A converter output match.

You can set an output voltage by controlling the D/A converter with the CPU. First, you must set the D/A converter output to 0 V, and then turn the on/off control to ON and set the D/A converter output to the desired value when you turn on the POL converter. If the noninverted input voltage is high when the inverted input of Amp 2 is at 0 V (when the output is OFF), Amp 2 will try to raise the output voltage at the trim pin and cause an overvoltage when you turn on the POL converter. Also, you must connect an RC circuit to the D/A converter output to prevent Amp 2's noninverted input pin from rising high rapidly and creating an overshoot.

At Bellnix, we developed an evaluation power module called the BPE-37 for Xilinx's 7 series GTX/GTH FPGAs. BPE-37 uses the circuit described earlier and has four POL converters required for the MGT. These four POL converters are all BSV-series POLs featuring high accuracy, fast transient response and low ripple noise. The BPE-37 uses a microcontroller to control the D/A converter and to set each output voltage to an adequate level. The microcontroller also starts and stops the POL converters using their on/off pins to control sequencing. This technique allows design teams to modify output voltage, change sequencing and monitor the output voltage and current via PMBus serial communication.

The BPE-37's circuit configuration makes it possible to add digital functionality to the conventional analog-type POLs to satisfy all the power requirements for Xilinx's Virtex-7 series. The GTX/GTH transceivers in these FPGAs require POLs with the fastest transient response. Table 1 shows the BPE-37's key specifications.

For more details on Bellnix's offering, please refer to *http://www.bellnix.com/news/274.html.*

Power supply	Maximum allowed	POL converters	Power supply noise	Dynamic response
MGTAVCC	1.0V±30mV	BSV-1.5S12R0H	8mVp-p	18mV (voltage dip)
MGTAVTT	1.2V±30mV	BSV-3.3S8R0M	8mVp-p	20mV (voltage dip)
MGTVCCAUX	1.8V±50mV	BSV-3.3S3R0M	10mVp-p	10mV (voltage dip)

Table 1 – The BPE-37's key specifications

What's New in the Vivado 2013.2 Release?

Xilinx is continually improving its products, IP and design tools as it strives to help designers work more effectively. Here, we report on the most current updates to Xilinx design tools including the Vivado® Design Suite, a revolutionary system- and IP-centric design environment built from the ground up to accelerate the design of Xilinx® All Programmable devices. For more information about the Vivado Design Suite, please visit www.xilinx.com/vivado.

Product updates offer significant enhancements and new features to the Xilinx design tools. Keeping your installation up to date is an easy way to ensure the best results for your design.

The Vivado Design Suite 2013.2 is available from the Xilinx Download Center at www.xilinx.com/download.

VIVADO DESIGN SUITE 2013.2 RELEASE HIGHLIGHTS

- Full support for Zynq®-7000 All Programmable SoCs
- Vivado IP Integrator now integrated with High-Level Synthesis and System Generator in one comprehensive development environment
- Complete multiprocessor application support, including hardware/software co-debug, for detailed embedded-system analysis
- All 7 series FPGA and Zynq-7000 All Programmable SoC devices supported with PCI-SIG®-compliant PCI Express® cores
- Tandem two-stage configuration for PCI Express 100-ms requirements

Device Support

The following devices are production ready:

- Zynq-7000 All Programmable SoC: 7Z010, 7Z020 and 7Z100
- Defense-Grade Zynq All Programmable SoC: 7000Q, 7Z010, 7Z020 and 7Z030
- Defense-Grade Virtex®-7Q FPGA: VX690T and VX980T
- Defense-Grade Artix®-7Q FPGA: A100T and A200T
- XA Artix-7 FPGA: A100

VIVADO DESIGN SUITE: DESIGN EDITION UPDATES

Vivado IP Integrator

IP Integrator is now in full production release. The tool is available with every seat of Vivado Design Suite. Enhancements include:

- Automated IP upgrade flow for migrating a design to the latest IP version
- Cross-probing now enabled on IP Integrator-generated errors and warnings
- Integration with IP issued by System Generator
- Synthesis run-times up to 4x faster for IP Integrator-generated designs
- Error-correcting code (ECC) now supported in MicroBlaze™ within IP Integrator

Vivado IP Catalog

The Vivado IP flow enables bottom-up synthesis in project-based designs to reduce synthesis run-times (IP that has not changed will not be resynthesized). Significant enhancements to the "Manage IP" flow include:

- Simplified IP creation and management. An IP project is automatically created to manage the netlisting of IP.
- Easy generation of a synthesized design checkpoint (DCP) for IP, to enable IP usage in black-box flows using third-party synthesis tools.
- DCP for IP and the Verilog stub files are now co-located with the IP's XCI file.
- Scripts are delivered for compiling IP simulation sources for behavioral simulation using third-party simulators.

Vivado also offers improved handling of IP constraints. IP DCP contains the netlist and constraints for the IP, and constraints are scoped automatically.

Vivado Debug

Debugging support has been added for the Zynq-7Z100 All Programmable SoC; 7 series XQ package and speed grade changes; and Virtex-7 HT general ES devices (7VHT580T and 7VHT870T).

- Target Communication Framework (TCF) agent (hw_server)
- Vivado serial I/O analyzer support for IBERT 7 series GTZ 3.0
- Added support for debugging multiple FPGAs in the same JTAG chain

Tandem Configuration for Xilinx PCIe® IP

Designers can use tandem configuration to meet the fast enumeration requirements for PCI Express®. A two-stage bitstream is delivered to the FPGA. The first stage configures the Xilinx PCIe IP and all design elements to allow this IP to independently function as quickly as possible. Stage two completes the device configuration while the PCIe link remains functional. Two variations are available: Tandem PROM loads both stages from the same flash device, and Tandem PCIe loads stage two over the PCIe link. All PCIe configurations are supported up to X8Gen3.

- Tandem configuration is now released with production status, for both Tandem PROM and Tandem PCIe. Devices with this status include XC7K325T-FFG900, XC7VX485T-FFG1761 (PCIE X1Y0 location required) and XC7VX690T-FFG1761 (PCIE X0Y1 location required).
- Tandem configuration is available as beta for additional devices, for both Tandem PROM and Tandem PCIe. Hardware testing for these devices has been limited. They include the XC7K160T-FFG676, XC7K410T-FFG676, XC7VX415T-FFG1158 (PCIE X0Y0 location recommended) and XC7VX550T-FFG1158 (PCIE X0Y1 location recommended).

For more information, see PG054 (version 2.1) for Gen2 PCIe IP, or PG023 (version 2.1) for Gen3 PCIe IP.

VIVADO DESIGN SUITE: SYSTEM EDITION UPDATES

Vivado High-Level Synthesis

- Video libraries with OpenCV interfaces are enhanced with support for 12 additional functions.
- Adders can now be targeted for implementation in a DSP48 directly from Vivado HLS.
- The Xilinx FFT core can now be instantiated directly into the C/C++ source code. This feature is beta. Contact your local Xilinx FAE for details on how to use this feature.

Vivado IP Integrator support in System Generator for DSP

- Model packaged as IP with support for gateway inputs and outputs to be packaged as interfaces or raw ports.
- Capability to specify AXI4-Stream interface in gateway blocks and to infer.
- AXI4-Lite and AXI4-Stream interfaces from named gateway blocks.
- System Generator generates example RTL Vivado project and example IP Integrator block diagram to enable easy evaluation of packaged IP.
- New System Generator IP Integrator integration tutorial available with System Generator tutorials.

Introducing support for MATLAB® and Simulink® release R2013a

Blockset enhancements include:

- Support for FIR Compiler v7.1 with introduction of area, speed and custom optimization options.
- Model upgrade flow now includes port and parameter checks to the HTML report.
- Enhanced caching to speed up subsequent model compilation.

UPDATES TO EXISTING IP

AXI Ethernet

New IEEE 1588 hardware time stamping for one-step and two-step

AXI Ethernet Lite

Virtex-7 FPGA in production

Trimode Ethernet MAC

Virtex-7 FPGA, Artix-7 FPGA, Zynq-7000 All Programmable SoC versions in production

GMII2RGMII

Zynq-7000 All Programmable SoC version in production

10G Ethernet MAC, RXAUI, XAUI

Virtex-7 FPGA, Zynq-7000 All Programmable SoC versions in production

PCI32 and PCI64

Artix-7 FPGA in production

LEARN MORE ABOUT THE VIVADO DESIGN SUITE

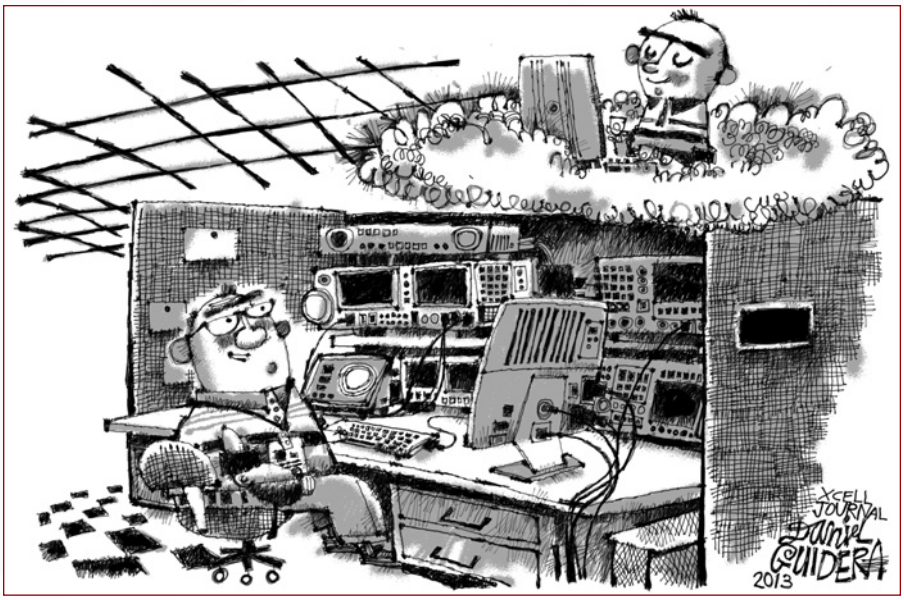
Vivado QuickTake Video Tutorials

Vivado Design Suite QuickTake video tutorials are how-to videos that take a look inside the features of the Vivado Design Suite. Topics include high-level synthesis, simulation and IP Integrator, among others. New topics are updated regularly. See *www.xilinx.com/training/vivado*.

Vivado Training

For instructor-led training on the Vivado Design Suite, please visit *www.xilinx.com/training*.

Xpress Yourself in Our Caption Contest



DANIEL GUIDERA

If you've looked at clouds from both sides now, take another gander. Here's your opportunity to Xercise your funny bone by submitting an engineering- or technology-related caption for this cartoon showing an engineer with his head—and all the rest of him, actually—in the clouds. The image might inspire a caption like "Dave made major strides in cloud computing after studying transcendental meditation."

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 an Avnet ZedBoard, featuring the Xilinx® Zynq®-7000 All Programmable SoC (*http://www.zedboard.org/*). 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 July 15, 2013. All entries must be received by the sponsor by 5 p.m. PT on Oct. 1, 2013.

So, look for that silver lining and get writing!

ROBIN FINDLEY, embedded-systems consultant/contractor with Findley Consulting, LLC in Clarkesville, Ga., won a shiny new Avnet ZedBoard with this caption for the asteroid cartoon in Issue 83 of *Xcell Journal*:

"Paul watches as Engineering lobs another design over the wall."

Congratulations as well to our two runners-up:

"Remember when you asked, 'What could possibly go wrong?'"

– Mark Rives, systems engineer, Micron Technology, Berthoud, Colo.

"Larry, you know how I said our timing looked great???"

– Dave Nadler, owner, Nadler & Associates, Acton, Mass.

Trust Synopsys' FPGA Synthesis Solutions to deliver the fastest time-to-market for your FPGA design

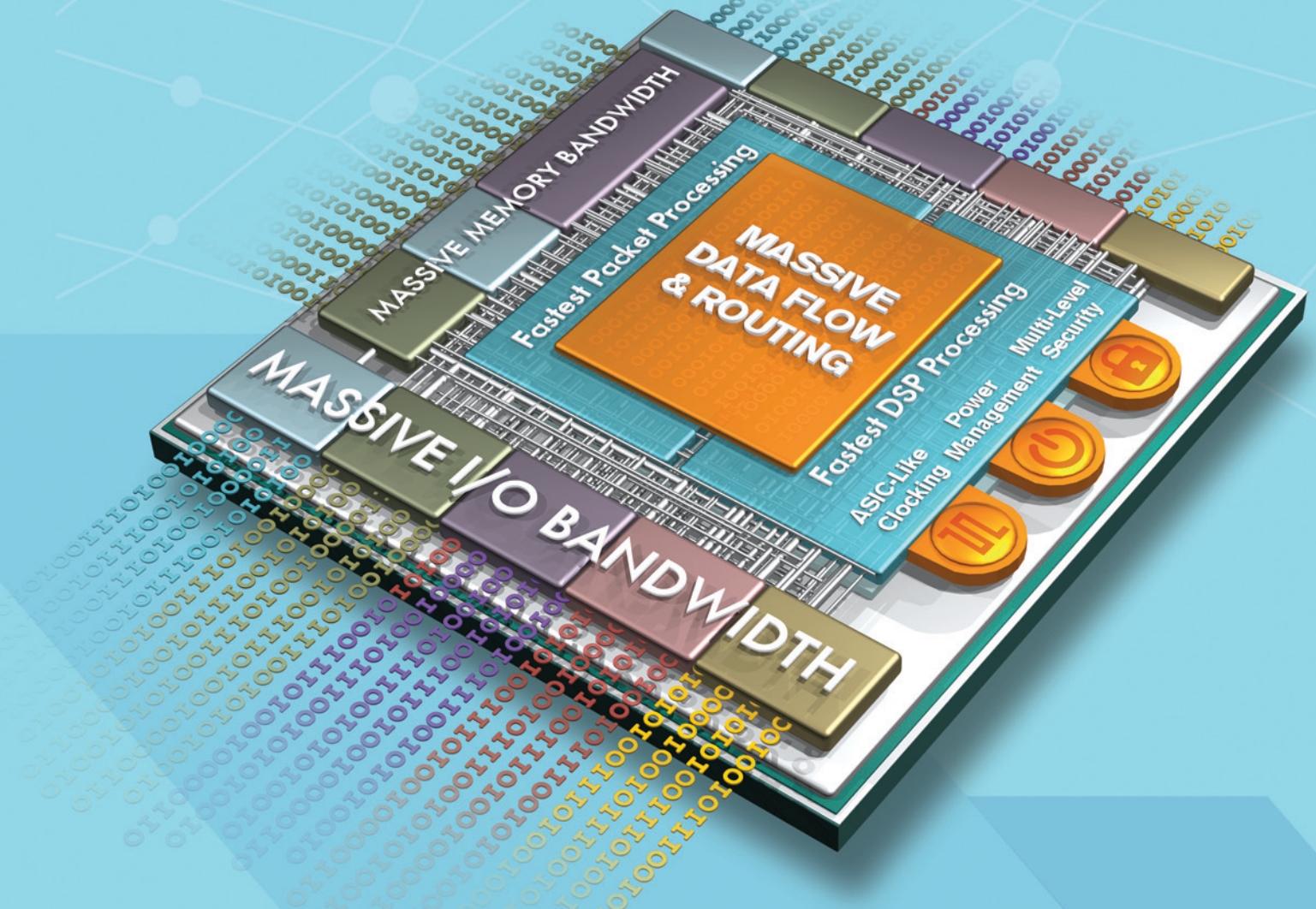
FPGAs keep getting bigger, but your schedule is not. There is no time to waste on numerous design iterations and long tool runtimes. Use the hierarchical and incremental techniques available in Synopsys' Synplify® software to bring in your schedule and meet aggressive performance goals.

To learn more about how Synopsys FPGA design tools accelerate design bring-up, visit **www.synopsys.com/fpgafastturnaround**.

SYNOPSYS®
Accelerating Innovation

A Generation Ahead

First ASIC-Class Programmable Architecture



UltraSCALE™ Architecture

LEARN MORE

XILINX
ALL PROGRAMMABLE™