ISSUE 86, FIRST QUARTER 2014

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

Xilinx Ships Industry's First 20-nm All Programmable Devices

Using Vivado HLS to Design a Median Filter and Sorting Network for Video

Ins and Outs of Testbench Creation

How to Add an RTOS to Your Zynq SoC Design

Kintex-7 Receiver Mines TV 'White Space' for New Comms Services 16

XILINX
ALL PROGRAMMABLE™

www.xilinx.com/xcell

XILINX
ALL PROGRAMMABLE™

AVNET®
electronics marketing

Xilinx® SpeedWay Design Workshops™

Featuring MicroZed™, ZedBoard™ & the Vivado® Design Suite

ZYNQ®

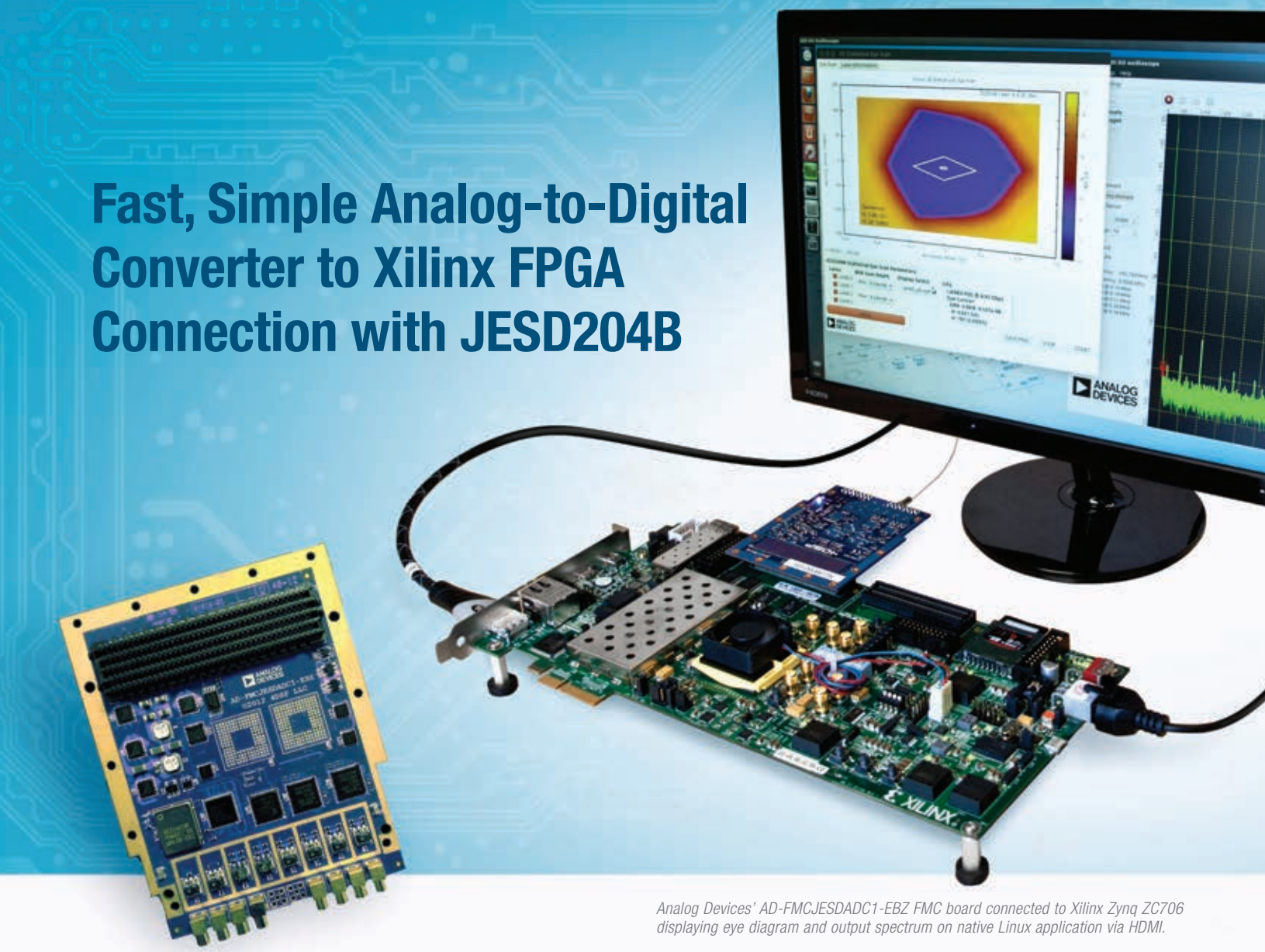
VIVADO™

Avnet Electronics Marketing introduces a new global series of Xilinx® SpeedWay Design Workshops™ for designers of electronic applications based on the Xilinx Zynq®-7000 All Programmable (AP) SoC Architecture. Taught by Avnet technical experts, these one-day workshops combine informative presentations with hands-on labs, featuring the ZedBoard™ and MicroZed™ development platforms. Don't miss this opportunity to gain hands-on experience with development tools and design techniques that can accelerate development of your next design.

em.avnet.com/xilinxspeedways

DESIGNED BY AVNET

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

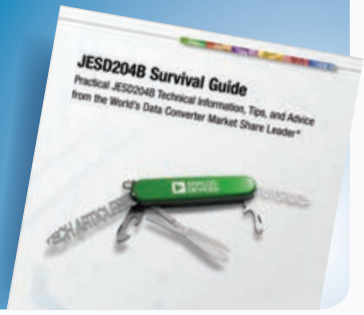


Analog Devices' AD-FMCJESDADC1-EBZ FMC board connected to Xilinx Zynq ZC706 displaying eye diagram and output spectrum on native Linux application via HDMI.

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 Expands Generation-Ahead Lead to Era of UltraScale

Happy New Year! As we move into 2014, I find myself very excited about Xilinx's prospects and predict that this will be banner year for the company and our customers. I have been an eyewitness to the tremendous effort Xilinx has made over the last six years under the leadership of CEO Moshe Gavrielov. In that time, among many accomplishments, Xilinx launched three generations of products and developed an entirely new, best-in-class design tool suite. What's more, by launching our 7 series All Programmable devices a few years ago and being first to ship 20-nanometer devices in December 2013, Xilinx is growing from the FPGA technology leader to a leader in semiconductor and system innovations.

The 7 series, to me, marks the beginning of Xilinx's breakout, as the company not only delivered the best FPGAs to the market at 28 nm, but also innovated two entirely new classes of devices: our 3D ICs and Zynq SoCs.

Meanwhile, our tools group developed from scratch the Vivado® Design Suite and the UltraFast™ Design Methodology to help customers get innovations to market sooner. All these advancements in the 7 series are remarkable in and of themselves, but what's more impressive is that they set a solid road map for future prosperity for Xilinx and a path toward innovation for our customers. It has been deeply rewarding to see this success becoming evident and undeniable to the outside world over the last few quarters, as Xilinx now has more than 70 percent of total 28-nm FPGA market share with our 7 series devices. These numbers truly put wood behind the marketing arrow that Xilinx is a Generation Ahead.

In my days as a reporter, I often witnessed the back-and-forth bravado in this industry between Xilinx and its competition. There was so much of it and with so little concrete proof on either side that it became white noise—like listening to two kids bickering in the backseat over whatever.

Of course, today I'm biased because I work here, but I predict that if Xilinx's momentum and Generation Ahead lead weren't apparent to you in 2013, they will be in 2014. If you look at the numbers and Xilinx's first-to-market delivery of products over the last two generations, it is evident that Xilinx is not content to sit still and wait for the competition to try to leapfrog us. As you will read in the cover story, late last year Xilinx delivered the industry's first 20-nm FPGAs to customers months ahead of the competition (which has yet to ship 20-nm products) and unveiled the portfolios for the 20-nm Kintex® and Virtex® UltraScale™ devices. Included in the UltraScale offering is a Virtex FPGA with a capacity four times greater than the competition's largest device, continuing Xilinx's undisputed density leadership.

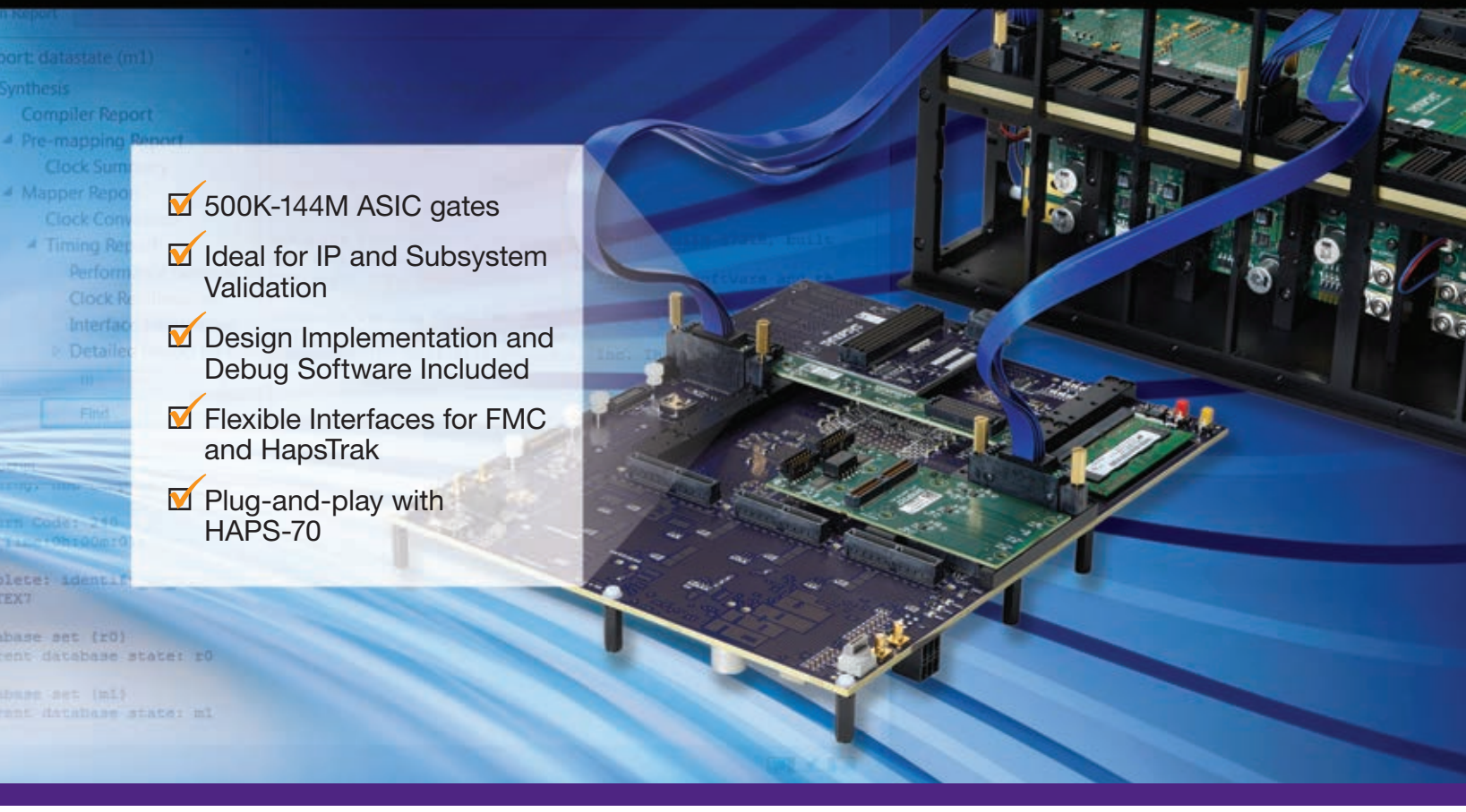
The new UltraScale families are ASIC-class devices that bring incredible value to customers who today are developing the innovations that will shape tomorrow—not just those a year or more from now, but products that will define the future of electronics. What's even more exciting is that these launches represent just the beginning of the UltraScale era at Xilinx.

Mike Santarini
Publisher

Announcing

HAPS Developer eXpress Solution

Pre-integrated hardware and software for fast prototyping of complex IP systems



Designs come in all sizes. Choose a prototyping system that does too. HAPS-DX, an extension of Synopsys' HAPS-70 FPGA-based prototyping product line, speeds prototype bring-up and streamlines the integration of IP blocks into an SoC prototype.

To learn more about Synopsys FPGA-based prototyping systems, visit www.synopsys.com/haps

SYNOPSYS®

Accelerating Innovation

VIEWPOINTS

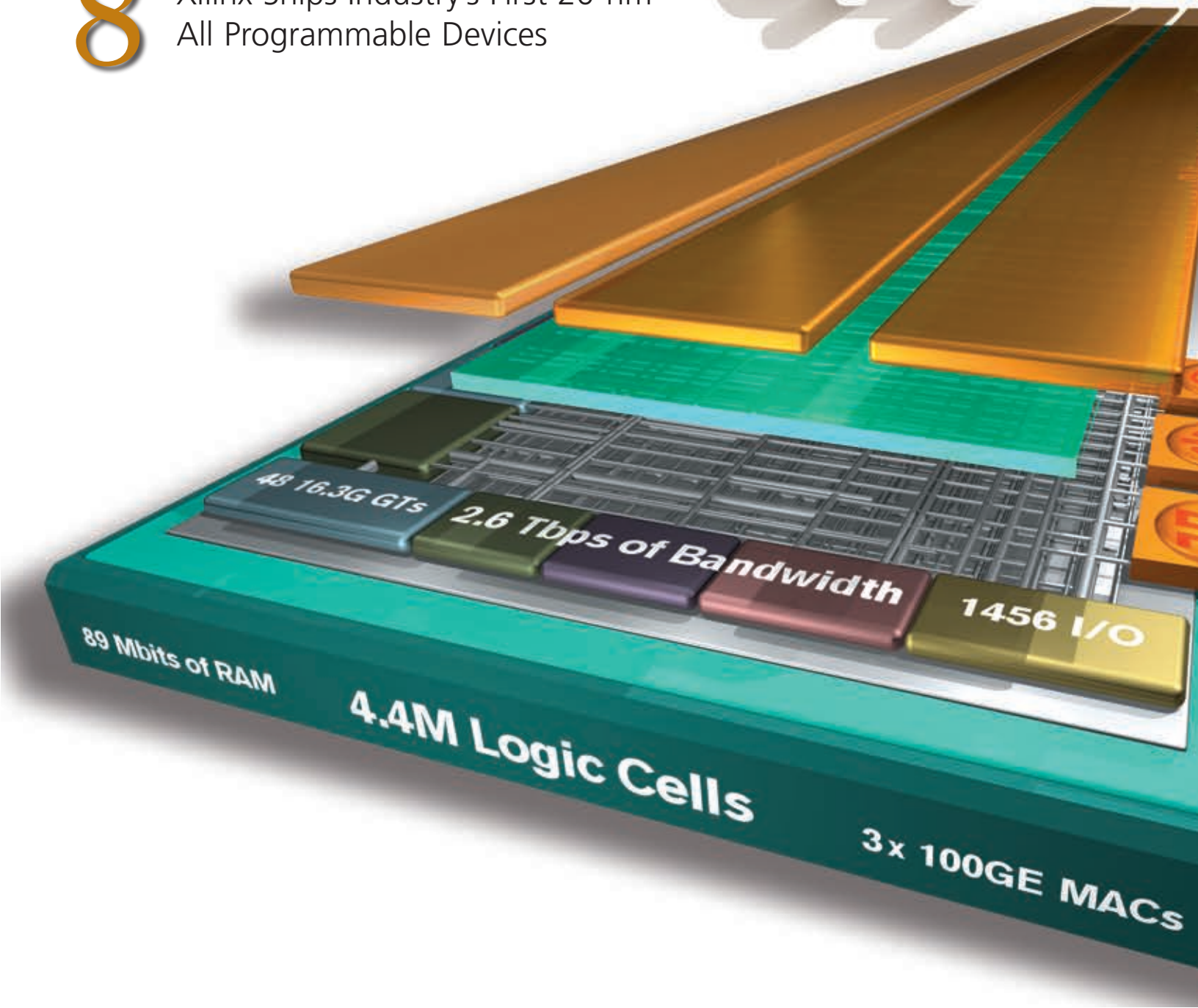
Letter From the Publisher

XCELLENCE BY DESIGN APPLICATION FEATURES

Profiles in Xcellence

16

Cover Story



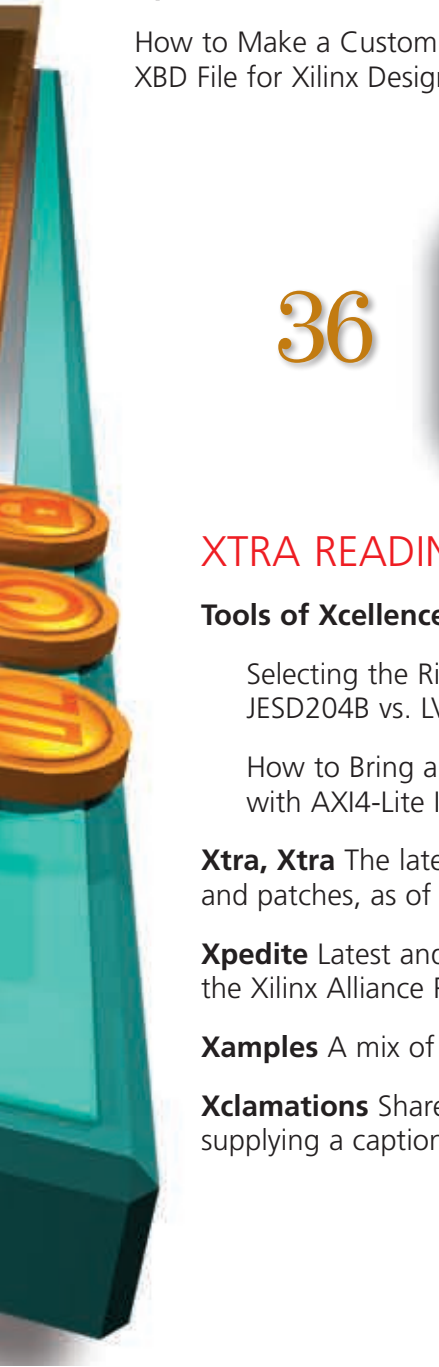
THE XILINX XPERIENCE FEATURES

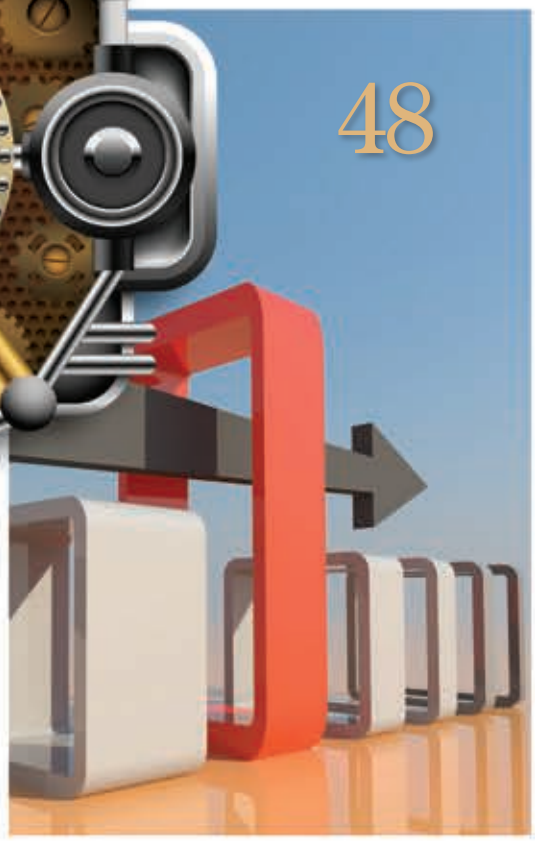
Ask FAE-X

Xplanation: FPGA 101

Xplanation: FPGA 101

Xplanation: FPGA 101





XTRA READING

Tools of Xcellence

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

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

Xilinx Ships Industry's First 20-nm All Programmable Devices

by Mike Santarini
Publisher
Xcell Journal
Xilinx, Inc.

Latest product portfolio sets capacity record with 3D Virtex UltraScale FPGA containing 4.4 million logic cells.

On Nov. 11 of 2013, Xilinx accomplished a key milestone in building on its Generation Ahead advantage by shipping to customers the industry's first 20-nanometer All Programmable FPGA—a Kintex® UltraScale™ XCKU040 device—months ahead of when the competition was purporting to ship its first 20-nm devices. Then, in December, Xilinx built on that accomplishment by unveiling its entire 20-nm Kintex UltraScale and Virtex® UltraScale portfolios, the latter of which includes the Virtex UltraScale VU440. Based on 3D IC technology, this 4.4 million-logic-cell device breaks world records Xilinx already held for the highest-capacity FPGA and largest semiconductor transistor count with its 28-nm Virtex-7 2000T.

"We are open for business at 20 nm and are shipping the first of our many 20-nm UltraScale devices," said Steve Glaser, senior vice president of corporate strategy and marketing at Xilinx. "With the 20-nm UltraScale, we are expanding the market leadership that we firmly established with our tremendously successful 28-nm, 7 series All Programmable devices. Today, we are not only delivering to customers the first 20-nm devices manufactured with TSMC's 20SoC process months ahead of the competition, but are also delivering devices that leverage the industry's most advanced silicon architecture as well as an ASIC- strength design suite and methodology."

All the devices in the 20-nm Kintex UltraScale and Virtex UltraScale portfolios feature ASIC-class performance and functionality, along with lower power and higher capacity than their counterparts in Xilinx's tremendously successful 7 series portfolio (Figure 1). The 7 series devices, built at the 28-nm silicon manufacturing node, currently control more than 70

	KINTEX 7	KINTEX UltraSCALE	VIRTEX 7	VIRTEX UltraSCALE
Logic Cells (LC)	478	1,161	1,995	4,407
Block RAM (BRAM; Mbits)	34	76	68	115
DSP48	1,920	5,520	3,600	2,880
Peak DSP Performance (GMACs)	2,845	8,180	5,335	4,268
Transceiver Count	32	64	96	104
Peak Transceiver Line Rate (Gbps)	12.5	16.3	28.05	32.75
Peak Transceiver Bandwidth (Gbps)	800	2,086	2,784	5,101
PCI Express Blocks	1	6	4	6
100G Ethernet Blocks	-	1	-	7
150G Interlaken Blocks	-	2	-	9
Memory Interface Performance (Mbps)	1,866	2,400	1,866	2,400
I/O Pins	500	832	1,200	1,456

Figure 1 – The 20-nm Kintex and Virtex UltraScale FPGAs deliver industry-leading features complementary to the Kintex and Virtex 7 series devices (max counts listed).

percent of programmable logic device industry market share. What's more, with UltraScale, Xilinx has taken additional steps to refine the device architecture and make ASIC-class improvements to its Vivado® Design Suite. Last October the company introduced a streamlined methodology called the UltraFast Design Methodology (detailed in the cover story of *Xcell Journal* issue 85; *http://issuu.com/xcelljournal/docs/xcell_journal_issue_85/8?e=2232228/5349345*). Xilinx will follow up its 20-nm UltraScale portfolio with 16-nm FinFET UltraScale devices in what it calls its "multinode strategy" (Figure 2).

"We are in a multinode world now where customers will be designing in devices from either our 7 series, our UltraScale 20-nm or our upcoming UltraScale 16-nm FinFET family depending on what is the best fit for their system requirements," said Kirk Saban, product line marketing manager at Xilinx. "For example, if you compare our 20-nm Kintex UltraScale devices with our Kintex-7 devices, we have significantly more logic and DSP capability in Kintex UltraScale than we had in Kintex-7. This is because a vast majority of applications needing high signal-processing bandwidth today tend to demand Kintex-class price points and densities. If the customer's application doesn't require that additional density or DSP capability, Kintex-7 is still a very viable design-in vehicle for them."

Saban said that the multinode approach gives Xilinx's broad user base the most powerful selection of All Programmable devices available in the industry, while the Vivado Design Suite and UltraFast methodology provide unmatched productivity.

"There's a misconception in the industry that Xilinx is going to be leapfrogged by the competition, which is waiting for Intel to complete its 14-nm FinFET silicon design process and then fabricate the competition's next-generation devices," said Saban. "We are certainly not sitting still. We are already offering our 20-nm UltraScale devices, which allow customers to create innovations today. We will be offering our UltraScale FinFET devices in the same time frame as the competition. We are going to build on our Generation Ahead advantage with these devices and the ASIC-class advantage of our Vivado Design Suite and UltraFast methodology."

These advances are built on a foundation of solid manufacturing, Saban said. "We are confident we have the industry's strongest foundry partner in TSMC, which has a proven track record for delivery and reliability," he said. "Foundry is TSMC's primary business, and they manufacture devices for the vast majority of the who's who in the semiconductor industry. What's more, TSMC's former CTO, now adviser, Chenming Hu actually pioneered the FinFET process, and we are very impressed with their FinFET development for next-gen processes."

KINTEX AND VIRTEX ULTRASCALE TRANSISTOR COUNTS

Each silicon process node presents the industry with a new set of manufacturing and design challenges. The 20-nm node is no exception. This geometry introduced new challenges in terms of routing delays, clock skew and CLB packing. However, with the Kintex UltraScale and Virtex UltraScale devices, Xilinx was able to overcome these challenges and greatly improve overall performance and utilization rates. (See the sidebar, "What's the Right Road to ASIC-Class Status for FPGAs?")

The intricacies of the node also allowed Xilinx to make several block-level improvements to the architecture, all of which Xilinx co-optimized with its Vivado tool suite to deliver maximum bandwidth and maximum signal-processing capabilities.

"If we have a closer look at our DSP innovations, we've gone to a wider-input multiplier, which allows us to use fewer blocks per function and deliver higher precision for any type of DSP application," said Saban. "We also included some additional features for our wireless communications customers in terms of FEC, ECC and CRC implementations now being possible within the DSP48 itself."

On the Block RAM front, Xilinx hardened the data cascade outputs and improved not only the power, but also the performance of BRAM with some new, innovative hardened features.

Xilinx is offering two different kinds of transceivers in the 20-nm Kintex and Virtex UltraScale portfolios. Mid- and high-speed-grade Kintex UltraScale devices will support 16.3-Gbps backplane operation. Even the slowest-speed-grade devices in the Kintex UltraScale family will offer impressive transceiver performance at 12.5 Gbps, which is particularly important for wireless applications. Meanwhile, in its Virtex UltraScale products, Xilinx will offer a transceiver capable of 28-Gbps backplane operation, as well as 33-Gbps chip-to-chip and chip-to-optics interfacing.

"We've added some significant integrated hard IP blocks into UltraScale," said Saban. "We've added a 100-Gbps Ethernet MAC as hard IP to both the Virtex and Kintex UltraScale family devices. We also added to both of these UltraScale portfolios hardened 150-Gbps Interlaken interfaces and hardened PCI Ex-

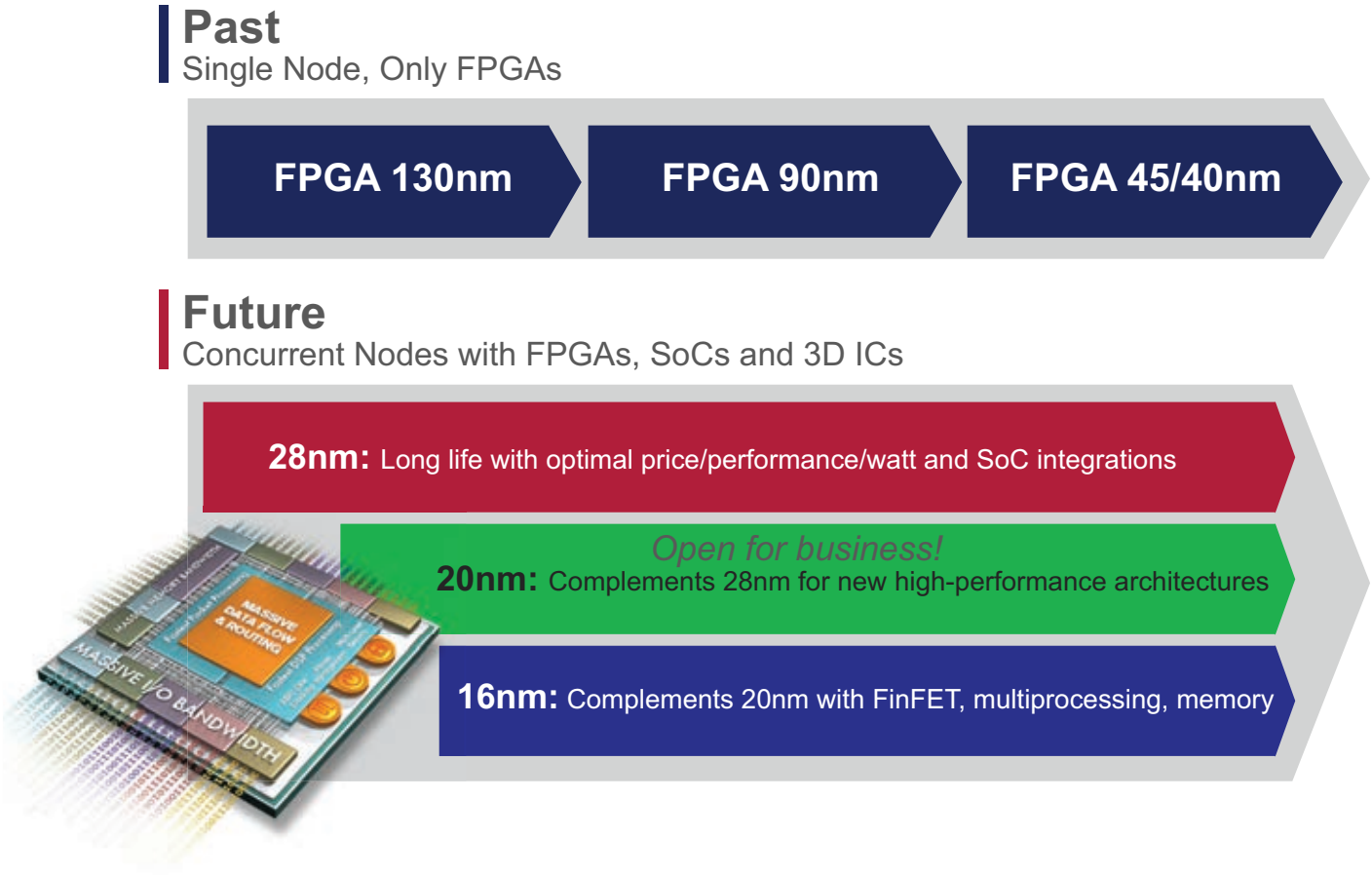


Figure 2 – Xilinx's Generation Ahead strategy favors multinode product development, with the concurrent release of FPGA, SoC and 3D IC product lines on nodes best suited for customer requirements.

What's the Right Road to ASIC-Class Status for FPGAs?

by Steve Leibson
Editor
Xcell Daily Blog
Xilinx, Inc.

Something that happened a while ago to ASICs has now hit FPGAs. What is it? It's the dominance of routing delay in determining design performance. Over the years, Dennard scaling increased transistor speed while Moore's Law scaling increased transistor density per square millimeter. Unfortunately, it works the other way for interconnect. As wires become thinner and flatter with Moore's Law scaling, they get slower. Eventually, transistor delay shrinks to insignificance and routing delay dominates. With the increasing density of FPGAs and with the entry of Xilinx® UltraScale™ All Programmable devices into the realm of ASIC-class design, the same problem has appeared. UltraScale devices have been re-engineered to overcome this problem, but the solution wasn't easy and it wasn't simple. Here's what it took.

STEP 1: COMPACT THE BLOCKS SO THAT SIGNALS DON'T NEED TO TRAVEL AS FAR.

Sounds obvious, right? Necessity is the mother of invention and at UltraScale densities, it was time to act. The CLBs in the UltraScale architecture have been reworked so that the Vivado® Design Suite can pack a logic design into the CLBs more efficiently. Logic-block designs become more tightly packed and less inter-CLB routing resource is required as a result. The routing paths also become shorter. Changes within the UltraScale architecture's CLBs include adding dedicated inputs and outputs to every flip-flop within the CLB (so that the flip-flops can be used independently for greater utilization); adding more flip-flop clock enables; and adding separate clocks to the CLBs' shift registers and distributed RAM components. Conceptually, the improved CLB utilization and packing looks like the diagram in Figure 1.

The example illustration shows that a circuit block formerly implemented with 16 CLBs now fits in nine of the improved UltraScale CLBs. The distribution of the small blue squares and triangles in the illustration shows that the CLB utilization has improved and the reduction in red lines shows that routing requirements are reduced as well.

STEP 2: ADD MORE ROUTING RESOURCES.

A rise in transistor density from Moore's Law scaling causes the number of CLBs to increase proportional to N^2, where N is the linear scaling factor of the IC process technology. Unfortunately, FPGA routing resources tend to scale linearly with N—far more slowly. That's a situ-

Figure 1 – CLB utilization is improved and routing requirements reduced in the UltraScale architecture.

ation with rapidly diminishing returns unless you take action and solve the problem. For the UltraScale architecture, the solution involved adding more local routing resources so that the routability improves more quickly with increasing CLB density. Figure 2 shows the result.

However, it's not sufficient to simply increase the hardware-routing resources. You must also enhance the design tool's place-and-route algorithms so that they will employ these new resources. The Xilinx Vivado Design Suite has been upgraded accordingly.

STEP 3: DEAL WITH INCREASING CLOCK SKEW.

You might not know this, but FPGA clocking has been pretty simplistic because it could be. Earlier FPGA generations relied on a central clock-distribution spine, which would then fan out from the IC's geometric center to provide clocks to all of the on-chip logic. That sort of global clocking scheme just isn't going to work in ASIC-class FPGAs like the ones you find in the Virtex UltraScale and Kintex UltraScale All Programmable device families. Rising CLB densities and increasing clock rates won't permit it. Consequently, UltraScale devices employ a radically improved clocking scheme as seen in Figure 3.

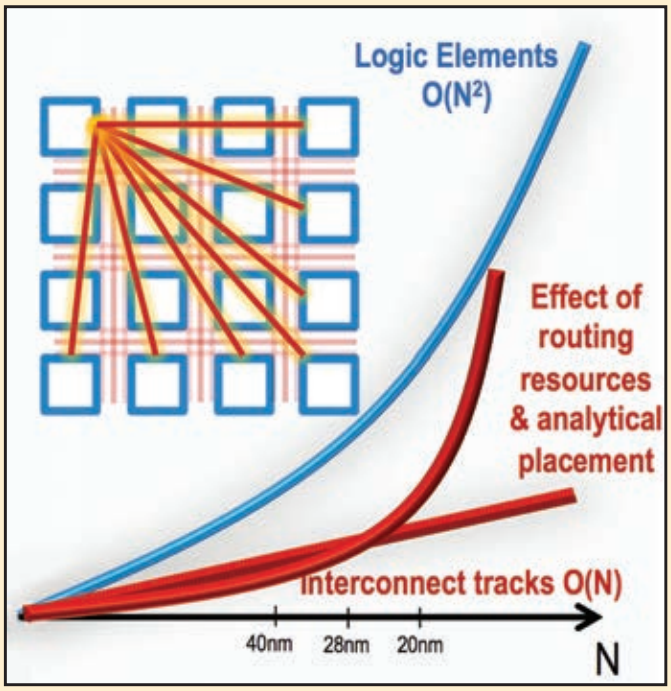


Figure 2 – The blue line shows exponential CLB growth with increasing transistor density. The straight red line shows the slower, linear growth of inter-CLB interconnect using previous-generation routing resources. Note that the straight red line is rapidly diverging from the blue curve. The red curve shows the improved routability of the enhanced local inter-CLB interconnect scheme used in the UltraScale architecture.

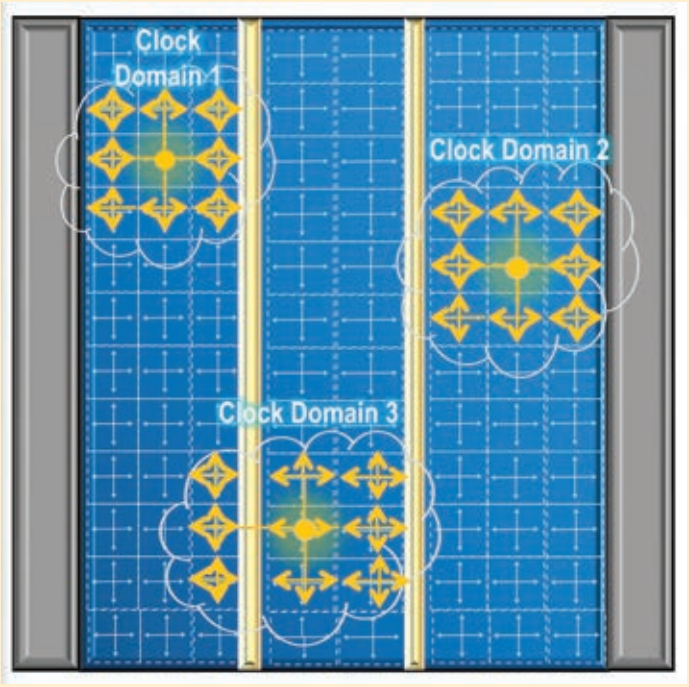


Figure 3 – A radical new clocking scheme can place multiple clock-distribution nodes at the geometric center of many on-chip clock domains.

The UltraScale architecture's clock-distribution network consists of a regionalized, segmented clocking infrastructure that can place multiple clock-distribution nodes at the geometric centers of many on-chip clock domains. The individual clock-distribution nodes then drive individual clock trees built from properly sized infrastructure segments. There are at least three big benefits to this approach:

1. Clock skew quickly shrinks.
2. There's a lot more clocking resource to go around.
3. Timing closure immediately gets easier.

However, it's not sufficient to improve the clocking infrastructure unless the design tools support the new clocking scheme, so the Vivado Design Suite has been upgraded accordingly, as it was for the improved inter-CLB routing discussed above in Step 2.

For each of these three steps, Xilinx had to make big changes in both the hardware architecture and the design tools. That's what Xilinx means when it says that the UltraScale architecture and the Vivado Design Suite were co-optimized. It required a significant effort—which was absolutely mandatory to deliver an ASIC-class All Programmable device portfolio.

For more information, see the white paper "Xilinx UltraScale Architecture for High-Performance, Smarter Systems" (*http://www.xilinx.com/support/documentation/white_papers/wp434-ultrascale-smarter-systems.pdf*).

press® Gen3 blocks that are capable of operation all the way up to Gen3x8."

Both UltraScale families support DDR4 memory, which offers 40 percent higher data rates than what is available in 7 series devices while delivering a 20 percent reduction in overall power consumption in terms of memory interfaces.

On the security front, Saban said Xilinx has added features to provide greater key protection, along with the ability to implement more detailed and sophisticated authentication schemes. He added that the Vivado tool suite supports all of these new enhancements.

Saban noted that the Virtex and Kintex UltraScale devices also share the same fabric performance. "The Kintex UltraScale is really a 'midrange FPGA' only in terms of its density, not in terms of its performance," he said.

Last but not least, the 20-nm UltraScale builds on Xilinx's outstandingly successful 3D IC technology, which it pioneered in the 7 series. With the second generation of Xilinx's stacked-silicon interconnect (SSI) technology, the company made significant enhancements to improve the interdie bandwidth across the multidice technology. "SSI enables monolithic devices to be realized with multiple dice all acting as one within a very large device," Saban said.

Xilinx was able to leverage this SSI technology to once again break its own world records for the largest-capacity FPGA and highest IC transistor count.

WORLD-RECORD CAPACITY

Certainly, a shining star in the 20-nm UltraScale lineup is the Virtex UltraScale XCVU440. Xilinx implemented the device with its award-winning 3D stacked-silicon interconnect technology, which stacks several dice side-by-side on a silicon interposer to which each die is connected. The resulting methodology allowed Xilinx at the 28-nm node to offer "More than Moore's Law" capacity and establish transistor-count and FPGA logic-cell capacity world records with the Virtex-7 2000T, which has 1,954,560 logic cells. With the Virtex UltraScale XCVU440, Xilinx is smashing its own record by offering a 20-nm device with 4.4 million logic cells (the equivalent to 50 million ASIC gates) for programming. The device also is by far the world's densest IC, containing more than 20 billion transistors.

"We anticipate that devices at this level of capacity will be a perfect fit for ASIC, ASSP and system emulation and prototyping," said Saban. Some vendors specialize in creating massive commercial boards for ASIC prototyping, but many more companies build their own prototyping systems. A vast majority of them are looking for the highest-capacity FPGAs for their prototypes. With the Virtex-7 2000T, Xilinx was the hands-down leader in this segment at the 28-

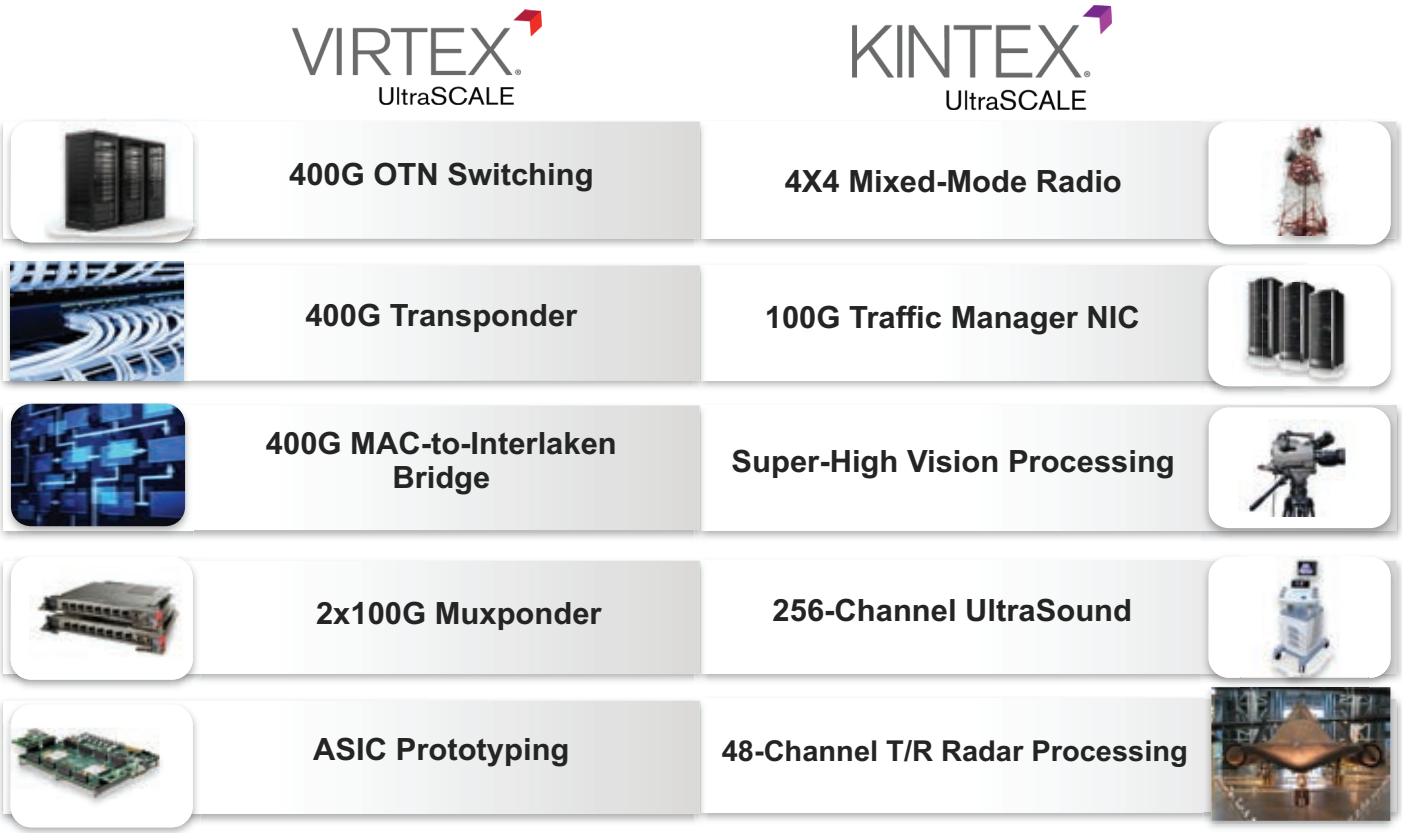


Figure 3 – Xilinx UltraScale devices are ideally suited for the next-generation innovations of smarter systems.

nm node. "I can tell you that prototyping customers are more than pleased at what we are offering with the Virtex UltraScale XCVU440," said Saban.

ULTRASCALE TO 400G WITH CFP4

Saban noted that Xilinx diligently planned the 20-nm UltraScale portfolio to offer next-generation capabilities for its entire user base, so as to enable them to create the next generation of smarter systems (Figure 3). The feature sets of the 20-nm Kintex and Virtex lines are especially well suited for key applications in networking, the data center and wireless communications.

Today the networking space is seeing a tremendous buildout of 100G applications. Saban said that leading-edge systems are already employing 100G technologies, and the technology is fast becoming mainstream and expanding to peripheral markets connecting devices to 100G networks. At the same time, the leading networking customers are already deep into development of next-generation 400G and terabit equipment. The new UltraScale devices are well suited to help those developing 100G solutions and those moving to more advanced 400G technologies.

Xilinx's first-generation SSI technology allowed the company to deliver the award-winning Virtex-7 H580T, which customers were able to leverage to create a 2x100G transponder on-chip for networks employing CFP2 optical modules (see cover story, *Xcell Journal* issue 80; *http://issuu.com/xcelljournal/docs/xcell80/8?e=2232228/2002872)*. Now, with second-generation SSI technology, Xilinx is enabling customers to accomplish an even more impressive feat and deliver single-FPGA transponder line cards and CFP4 optical modules.

"Designers looking to migrate to CFP4 modules and pack a 400G design onto a single FPGA device will need UltraScale for multiple reasons," said Saban. "First, it has a high number of 32G transceivers to interface to CFP4 optics via next-generation chip-to-chip interfaces (CAUI4). Equally important is the ability to support 400G of bandwidth through the device. Next-generation routing and ASIC-like clocking support the massive data flow needed in 400G systems."

In addition, he said, the Virtex UltraScale also supports fractional PLL, which enables a reduction in the number of external voltage-controlled oscillators required in the design. "In UltraScale we can use one VCXO and then internally generate any of the other frequencies needed," said Saban. "From this you'll enable a lower BOM cost and power through system integration—just on the line card itself, not including the cost reduction and power efficiency gained from the CFP4 optics."

ULTRASCALE TO A LOWER-COST, LOWER-POWER NIC

With the current boom in cloud computing applications, IT departments demand ever-more-sophisticated, lower-power and lower-cost compute muscle for their data centers. Xilinx Virtex-7 XT devices, which featured integrated x8 Gen3 blocks, have been a centerpiece in the network interface card at the heart of the most advanced data center architectures.

For network interface cards (NICs), the throughput (egress) on the PCI Express side must keep up with the throughput (ingress) on the Ethernet side. "And for a 100G NIC, multiple PCIe® Gen3 integrated blocks are needed," said Saban. "In the previous generation, the Virtex-7 XT devices were the only devices with integrated x8 Gen3 blocks to meet these requirements."

Now with the UltraScale, it's possible to achieve these same performance requirements with a lower-cost and even lower-power Kintex UltraScale FPGA. Kintex UltraScale has multiple PCI Express Gen3 integrated blocks as well as an integrated 100G Ethernet MAC, which was a soft IP core in the 7 series.

"Implementing a NIC in UltraScale Kintex not only enables the application to be implemented in a midrange device, but frees up logic for additional, differentiating packet-processing functions," said Saban.

ULTRASALE TO WIRELESS COMMUNICATIONS

In the wireless communications equipment industry, vendors are in the process of simultaneously rolling out LTE and LTE Advanced equipment while beginning to develop even more forward-looking implementations. New systems are sure to come down the pike in the form of sophisticated architectures that enable multitransmit, multireceive and beam-forming functionality.

Saban said the latest generation of beam former equipment at the heart of LTE and LTE Advanced systems commonly leverages architectures that rely on two Virtex-7 X690T FPGAs, which customers chose because of their mix of high DSP and BRAM resources. Now, with the Kintex UltraScale, Xilinx can offer a lower-cost single device that will do the same job, Saban said.

"The Kintex Ultrascale offers 40 percent more DSP blocks in a midrange device," he said. "It also has pre-adder squaring and extra accumulator feedback paths, which enables better DSP48 efficiency, folding of equations and more efficient computation. That means that in UltraScale, a two-chip Virtex-class application can be reduced to a single Kintex KU115 for 48-channel processing." The Kintex UltraScale KU115 has 5,520 DSP blocks and 2,160 BRAMs, which Saban called "the highest signal processing available in the industry and much greater than a GPU."

What's more, Xilinx's Vivado Design Suite features a best-in-class high-level synthesis tool, Vivado HLS, which Xilinx has co-optimized to help users efficiently implement complex algorithms for beam forming and other math-intensive applications.

To learn more about Xilinx's 20-nm UltraScale Portfolio, visit *http://www.xilinx.com/products/technology/ultrascale.html*.

Kintex-7 FPGA Receiver Mines TV 'White Space' for New Comms Services

by Mike Santarini
Publisher
Xcell Journal
Xilinx, Inc.

Adaptrum is first to market with a full transmission system for services that capitalize on open spectrum. Its product incorporates Xilinx All Programmable FPGAs.

The Federal Communications Commission's mandatory switchover from analog television to digital back in 2009 was uneventful for most people. But for communications service providers, it marked a significant business growth opportunity. Shortly after the conversion, the FCC auctioned off to communications companies, emergency services and other entities the portions of the analog TV broadcast spectrum that weren't needed for digital broadcasts. But after the auctions, there remained 300 MHz worth of the radio spectrum (the range of 400 to 700 MHz) originally designated for UHF TV channels 20 and beyond, a terrain only sparsely used by digital broadcasters today.

Acknowledging the fact that this prime real estate in the spectrum isn't being used to its full potential, a few years ago the FCC began working with a small cadre of tech giants—among them Google, Microsoft's Xbox group, Samsung, Dell, Intel and Philips—to find viable ways communications companies could share the 400-MHz to 700-MHz spectrum with TV broadcasters. The idea was to let them offer new mobile services in what's called the "white space," or unused channels between the ones that actually carry broadcasts (see Figure 1). Companies like Google and Microsoft want to use this white space (and similar white-space spectrum in countries around the world) to offer a new breed of communication services over what equates to a longer-distance and more signal-robust version of Wi-Fi. And the first equipment vendor to field a full commercial receiver and transmission system that will enable these new services is Xilinx customer Adaptrum (San Jose, Calif.).

"White space is a brand new market," said Adaptrum founder and CEO Haiyun Tang. "It's a lot like the Wild West at this point—there is a growing land rush to capitalize on this open spectrum in white space, and the standards are still evolving."

While the market has emerged relatively recently, Adaptrum was quick to identify the opportunity and has been developing its white-space communications equipment for a number of years. Tang, who holds a PhD in wireless communications from UC Berkeley and became steeped in cognitive-radio technology development in his professional career, started Adaptrum with Berkeley professor emeritus Bob Brodersen in 2005, with initial backing from an Air Force SBIR grant. Since then, the company has made impressive progress and secured follow-up venture financing. In 2008, Adaptrum began working with the FCC to help it form rules for TV white space and secured partnerships with Google, Microsoft and others. In April of 2012, it became one of the first equipment vendors to have a TV-band white-space device certified by the FCC to work in conjunction with the FCC-approved Telcordia database. Now, the company is moving from the technology from proof of concept to commercialization. In November of 2013, the FCC certified Adaptrum's ACRS TV white-space solution with the Google TV white-space database.

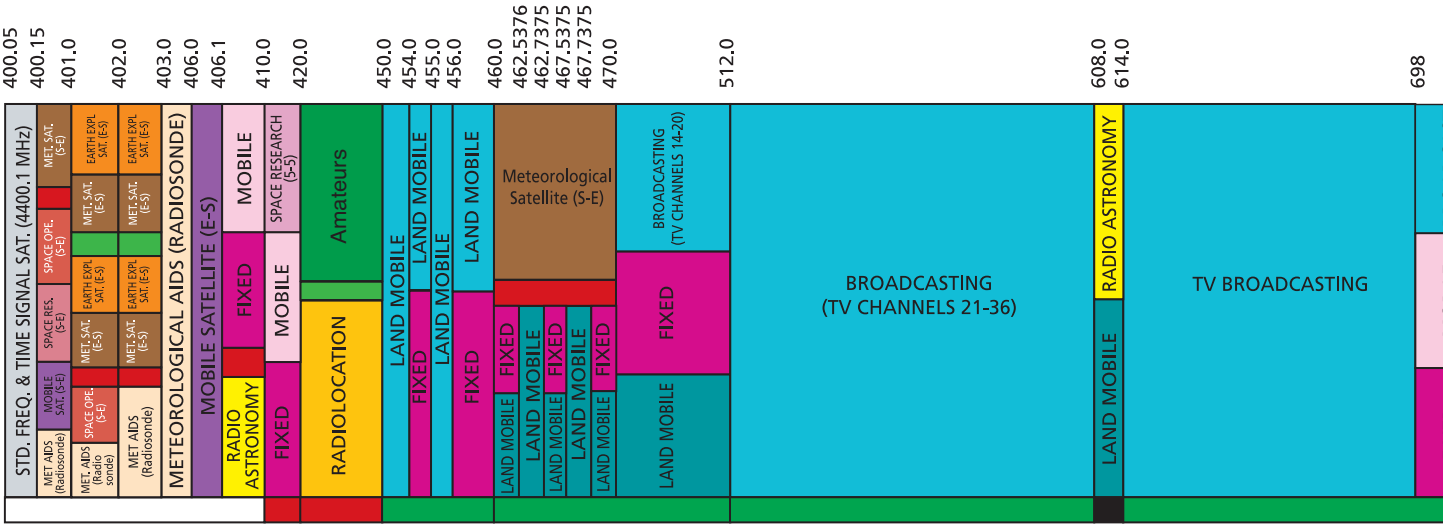


Figure 1 – In the United States, white space is found in roughly the 400k to 700k range of the spectrum, noted in blue above. (*http://www.ntia.doc.gov/files/ntia/publications/spectrum_wall_chart_aug2011.pdf*)

The latest spin of the system is built around the Xilinx® Kintex-®7 FPGA.

The ACRS consists of a lunch box-size base unit that can mount to traditional basestation clusters on poles, sides of buildings, mountains and the like, and a residential receiver—roughly the size of a desktop router or modem—that consumers will have in their homes (see Figure 2). The system will allow service providers to offer connectivity services on the order of 20 Mbits over 6 MHz (3 bits per hertz), which isn't as fast as Wi-Fi but has a much longer range.

"One of the reasons broadcasters in the 1950s selected that portion of the radio spectrum for broadcasting was the spectrum's great propagation characteristics," said Tang. "A TV broadcast can go through trees, walls and even over mountains. Traditional Wi-Fi-based mobile services have a hard time getting past trees, walls and other common obstacles. If you are right next to a Wi-Fi transmitter, you can get 300-Mbit data rates, but that data rate can decrease dramatically to a few megabits or no connection at all when you move away from the transmitter and have a few walls in between. The 400-MHz to 700-MHz spectrum is much more robust. Its propagation characteristics allow us to offer equipment that has a range of up to five miles." As a result, Tang said, "With TV white-space devices using the 400-MHz to 700-MHz spectrum, you'll start with 20-Mbit data rates but it will be the same for miles instead of feet."

Tang said he believes that in areas where Wi-Fi and mobile service are available, service providers can offer the new technology as an additional service to ensure connectivity. But in rural areas and undeveloped countries, the white-space approach might perhaps provide even primary data services, especially in geographies where it may not be economical to deploy wired networks or satellite broadband.

COMPLIANCE AND PERFORMANCE

To create a viable commercial solution, Adaptrum has had to figure out some complex technical challenges involved in using white space that meets the regulatory requirements while also ensuring good performance at a cost level that today's consumers expect. Adaptrum first needed to ensure that the bands its system would use to transmit and receive signals do not in any way interfere with licensed digital broadcasts. An added complexity is that other parts of the 400- to 700-MHz spectrum that aren't being used for broadcast can be randomly occupied by other devices, such as wireless microphones at sporting events, churches and nightclubs, or by medical equipment such as MRI products that use radio waves.

"The number of channels broadcasting and the frequency each is using will vary day to day and region to region," said Tang. "What's more, different companies will have different parts of the spectrum available for white-space equipment, so the equipment needs to be flexible and foolproof."

To address these challenges, Tang said, white-space equipment vendors are developing systems that primarily take one of two approaches. The first is what's called a database approach, in which providers collect TV transmission data from the FCC daily (for transmission in the United States) and employ a propagation model to determine the coverage contour of each tower. Anything outside of that contour is fair game for use by the white-space communications equipment.

The second technique, called sensing, is an autonomous, on-the-fly approach to detecting which channels are available. In this scheme, the white-space equipment continually senses which channels are in use and which are not to determine which channels it can use for communications. At this point, however, Tang said the database approach is the only one that has been proven and certified by the FCC. "It's the default approach thus far, because the FCC hasn't

yet fully tested sensing-based white-space equipment and proven that it is 100 percent reliable," he said.

Adaptrum's system uses a database approach, in which each morning, each basestation downloads an updated database (from the FCC for the U.S. market, for example) of what bands local stations will be using for broadcasting on a given day.

The FCC has certified less than a handful of companies as database service providers. These include Telcordia (Ericsson), Spectrum Bridge and, most recently, Google, which is seemingly looking at TV white space as a way to make its services available more broadly worldwide (see *http://www.google.org/spectrum/whitespace/channel/*).

In addition, there are a number of emerging and competing standards that strive to define how white space should best be used. For example, IEEE 802.22 is trying to define standards for a regional-area network. "Its goal is to provide connectivity over a longer range than conventional Wi-Fi," said Tang. "Its advertised range is at least 10 miles and with a maximum data rate of 20 Mbits over 6 MHz (3 bits per hertz)." Meanwhile, a competing IEEE standard, 802.11AF, is looking to use the TV white space for more traditional Wi-Fi operations. "At this point it is still early and no one knows for sure what standards will be adopted by the market," said Tang. "But there is a lot of excitement about this space because of the range it provides and the fact that outside of metro areas, there is an even greater amount of spectrum available."

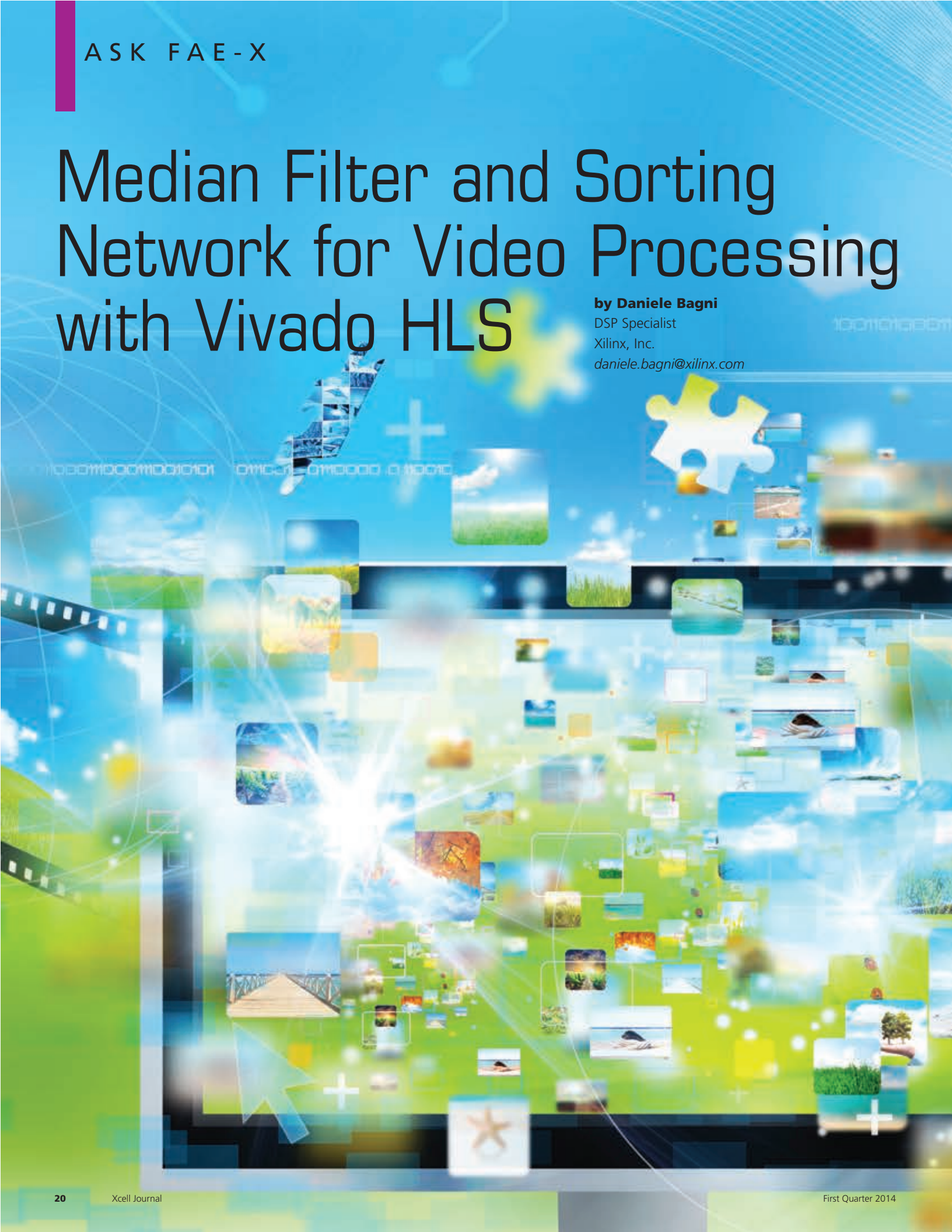
With service providers just now forming and standards still being defined, there are only a scant few companies—mostly startups—developing white-space equipment (transmitters and receivers) at this point, but Adaptrum seems to be a step ahead of the others. One of the reasons for that is the company's use of Xilinx 7 series FPGAs. "We are using the Kintex-7 All Programmable FPGA in our transceiver solution because of its flexibility. The ability to change the design of the device to adapt to changing design specifications, and to adjust it further as the standards get solidified, is really valuable to us," said Tang.

Tang said that the flexibility of the Kintex-7 FPGA will help in selling the system to service providers, as the device offers the maximum reprogrammability. That's because the FPGA can be reprogrammed and upgraded even after service providers sell the white-space communications services to customers. This field upgradability means that carriers can add enhancements and features on the fly over time. In this way, they have the opportunity to "sell higher-value plans to customers or add more services to existing plans, realizing significantly more value out of the platform than that of a fixed hardware solution," Tang said.

Figure 2 – The ACRS 2.0 has an aluminum shell construction sealed and ruggedized for outdoor life. It can be pole-mounted or wall-mounted and is powered over Ethernet.

Median Filter and Sorting Network for Video Processing with Vivado HLS

by Daniele Bagni
DSP Specialist
Xilinx, Inc.
daniele.bagni@xilinx.com



Vivado's high-level synthesis features will help you design a better sorting network for your embedded video app.

A growing number of applications today, from automobiles to security systems to handheld devices, employ embedded video capabilities. Each generation of these products demands more features and better image quality. However, for some design teams, achieving great image quality is not a trivial task. As a field applications engineer specializing in DSP design here at Xilinx, I'm often asked about IP and methods for effective video filtering. I've found that with the high-level synthesis (HLS) capabilities in the new Vivado® Design Suite, it's easy to implement a highly effective median-filtering method based on a sorting network in any Xilinx® 7 series All Programmable device.

Before we dive into the details of the method, let's review some of the challenges designers face in terms of image integrity and the popular filtering techniques they use to solve these problems.

Digital image noise most commonly occurs when a system is acquiring or transmitting an image. The sensor and circuitry of a scanner or a digital camera, for example, can produce several types of random noise. A random bit error in a communication channel or an analog-to-digital converter error can cause a particularly bothersome type of noise called "impulsive noise." This variety is often called "salt-and-pepper noise," because it appears on a display as random white or black dots on the surface of an image, seriously degrading image quality (Figure 1).

To reduce image noise, video engineers typically apply spatial filters to their designs. These filters replace or enhance poorly rendered pixels in an image with the appealing characteristics or values of the pixels surrounding the noisy ones. There are primarily two types of spatial filters: linear and nonlinear. The most commonly used linear filter is called a mean filter. It replaces each pixel value with the mean value of neighboring pixels. In this way, the poorly rendered pixels are improved based on the average values of the other pixels in the image. Mean filtering uses low-pass methods to de-noise images very quickly. However, this performance often comes with a side effect: it can blur the edge of the overall image.

In most cases, nonlinear filtering methods are a better alternative to linear mean filtering. Nonlinear filtering is particularly good at removing impulsive noise. The most commonly employed nonlinear filter is called an order-statistic filter. And the most popular nonlinear, order-statistic filtering method is the median filter.

Median filters are widely used in video and image processing because they provide excellent noise reduction with considerably less blurring than linear smoothing filters of a similar size. Like a mean filter, a median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. But instead of simply replacing the pixel value with the

Figure 1 – Input image affected by impulsive noise. Only 2 percent of the pixels are corrupted, but that's enough to severely degrade the image quality.

Figure 2 – The same image after filtering by a 3x3 median filter; the impulsive noise has been totally removed.

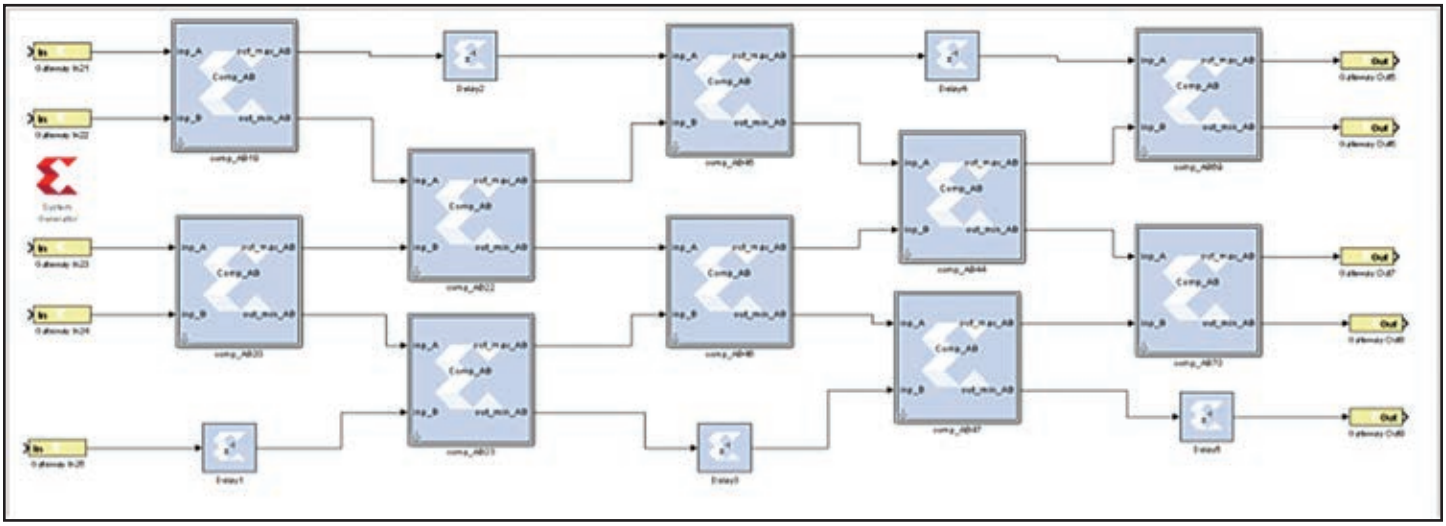


Figure 3 – Block diagram of a sorting network of five input samples. The larger blocks are comparators (with a latency of one clock cycle) and the small ones are delay elements.

mean of neighboring pixel values, the median filter replaces it with the median of those values. And because the median value must actually be the value of one of the pixels in the neighborhood, the median filter does not create new, unrealistic pixel values when the filter straddles an edge (bypassing the blur side effect of mean filtering). For this reason, the median filter is much better at preserving sharp edges than any other filter. This type of filter calculates the median by first sorting all the pixel values from the surrounding window into a numerical order and then replacing the pixel being considered with the middle pixel value (if the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used).

For example, assuming a 3x3 window of pixels centered around a pixel of value 229, with values

39 83 225
5 229 204
164 61 57

we can rank the pixels to obtain the sorted list 5 39 57 61 83 164 204 225 229.

The median value is therefore the central one—that is, 83. This number will replace the original value 229 in the output image. Figure 2 illustrates the effect of a 3x3 median filter applied to the noisy input image of Figure 1. The

```
#define KMED 3 // KMED can be 3, 5, 7, Ö
#define MIN(x,y) ( (x)>(y) ? (y) : (x) )
#define MAX(x,y) ( (x)>(y) ? (x) : (y) )

#ifndef GRAY11
typedef unsigned char pix_t; // 8-bit per pixel
#else
#include <ap_int.h>
typedef ap_int<11> pix_t; // 11-bit per pixel
#endif

pix_t median(pix_t window[KMED*KMED])
{
#pragma HLS PIPELINE II=1
#pragma HLS ARRAY_RESHAPE variable=window complete dim=1

  int const N=KMED*KMED;
  pix_t t[N], z[N];
  char i, k, stage;

  // copy input data locally
  for (i=0; i<KMED*KMED; i++) z[i] = window[i];

  // sorting network loop
  for (stage = 1; stage <= N; stage++)
  {
     if ((stage%2)==1) k=0;
     if ((stage%2)==0) k=1;
     for (i = k; i<N-1; i=i+2)
     {
        t[i  ] = MIN(z[i], z[i+1]);
      t[i+1] = MAX(z[i], z[i+1]);
      z[i  ] = t[i  ];
      z[i+1] = t[i+1];
     }
  } // end of sorting network loop

  // the median value is in location N/2+1,
  // but in C the address starts from 0
  return z[N/2];

} // end of function
```

Figure 4 – Implementation of a median filter via a sorting network in C

larger the window around the pixel to be filtered, the more pronounced the filtering effect.

Thanks to their excellent noise-reduction capabilities, median filters are also widely used in the interpolation stages of scan-rate video conversion systems, for example as motion-compensated interpolators to convert the field rate from 50 Hz to 100 Hz for interlaced video signals, or as edge-oriented interpolators in interlaced-to-progressive conversions. For a more exhaustive introduction to median filters, the interested reader can refer to [1] and [2].

The most critical step in employing a median filter is the ranking method you will use to obtain the sorted list of pixels for each output pixel to be generated. The sorting process can require many clock cycles of computation.

Now that Xilinx offers high-level synthesis in our Vivado Design Suite, I generally tell people that they can employ a simple but effective method for designing a median filter in C language, based on what's called the sorting-network concept. We can use Vivado HLS [3] to get real-time performance on the FPGA fabric of the Zynq®-7000 All Programmable SoC [4].

For the following lesson, let's assume that the image format is 8 bits per pixel, with 1,920 pixels per line and 1,080 lines per frame at a 60-Hz frame rate, thus leading to a minimum pixel rate of at least 124 MHz. Nevertheless, in order to set some design challenges, I will ask the Vivado HLS tool for a 200-MHz target clock frequency, being more than happy if I get something decently larger than 124 MHz (real video signals also contain blanking data; therefore, the clock rate is higher than that requested by only the active pixels).

```
#define MAX_HEIGHT  1080
#define MAX_WIDTH   1920
#define KKMED 1 // KKMED == 1 for 3x3 window

void ref_median(pix_t in_pix[MAX_HEIGHT][MAX_WIDTH],
                pix_t out_pix[MAX_HEIGHT][MAX_WIDTH],
                short int height, short int width) {

short int r, c; //raw and col index
pix_t pix, med, window[KMED*KMED];
signed char x, y;

L1:for(r = 0; r < height; r++) {
#pragma HLS LOOP_TRIPCOUNT min=600 max=1080 avg=720

  L2:for(c = 0; c < width; c++) {
  #pragma HLS LOOP_TRIPCOUNT min=800 max=1920 avg=1280
  #pragma HLS PIPELINE II=1

     if ( (r>=KMED-1)&&(r< height)&&
         (c>=KMED-1)&&(c<=width) )
     {
       for (y=-2; y<=0; y++)
          for (x=-2; x<=0; x++)
             window[(2+y)*KMED+(2+x)]=in_pix[r+y][c+x];
       pix = median(window);
     }
     else
       pix = 0;

     if(r>0 && c>0)
        out_pix[r-KKMED][c-KKMED] = pix;

   } // end of L2
   } // end of L1

} // end of function
```

Figure 5 – Initial Vivado HLS code, which doesn't take video line buffer behavior into account

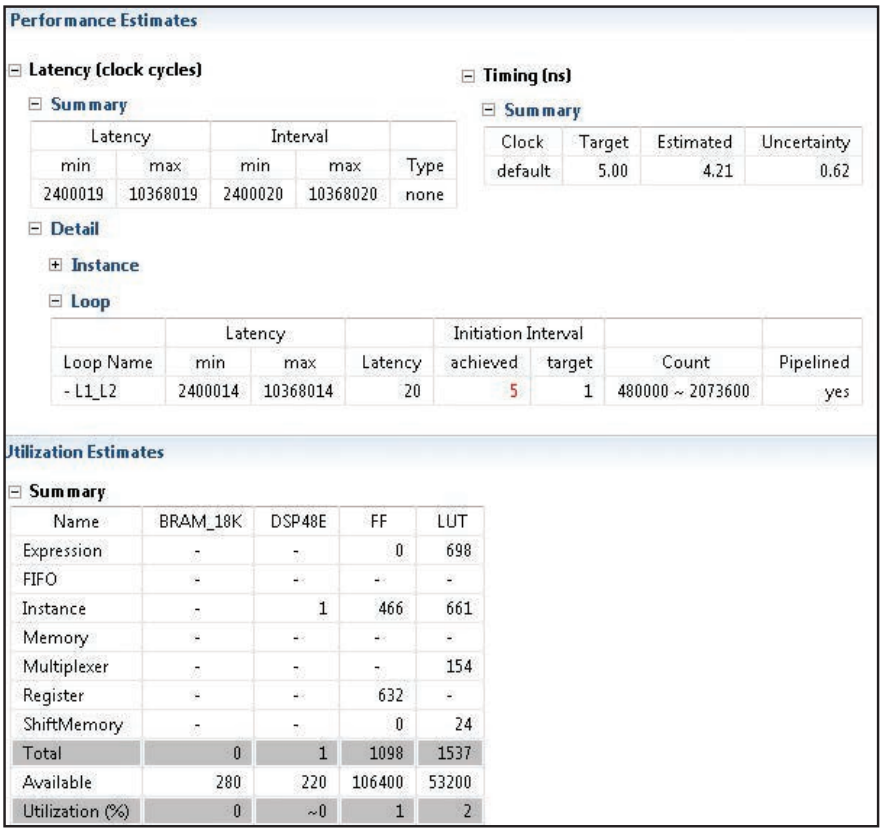
WHAT IS A SORTING NETWORK?

The process of rearranging the elements of an array so that they are in ascending or descending order is called sorting. Sorting is one of the most important operations for many embedded computing systems.

Given the huge amount of applications in which sorting is crucial, there are many articles in the scientific literature that analyze the complexity and speed of well-known sorting methods, such as bubblesort, shellsort, mergesort and quicksort. Quicksort is the fastest algorithm for a large set of data [5], while bubblesort is the simplest. Usually all these techniques are supposed to run as software tasks on a RISC CPU, performing only one comparison at a time. Their workload is not constant but depends on how much the input data can already be partially ordered. For example, given a set of N samples to be ordered, the computational complexity of quicksort is assumed to be N^2, NlogN and NlogN respectively in the worst-, average- and best-case scenarios. Meanwhile, for bubblesort, the complexity is N^2, N^2, N. I have to admit that I have not found a uniformity of views about such complexity figures. But all the articles I've read on the subject seem to agree on one thing—that computing the complexity of a sorting algorithm is not easy. This, in itself, seems like a good reason to search for an alternative approach.

In image processing, we need deterministic behavior in the sorting method in order to produce the output picture at constant throughput. Therefore, none of the abovementioned algorithms is a good candidate for our FPGA design with Vivado HLS.

Sorting networks offer a way to achieve a faster running time through the use of parallel execution. The fundamental building block of a sorting network is the comparator—a simple component that can sort two numbers, a and b, and then output their maximum and minimum respectively to its



Performance Estimates

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2400019	10368019	2400020	10368020	none

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	5.00	4.21	0.62

Detail

Instance

Loop

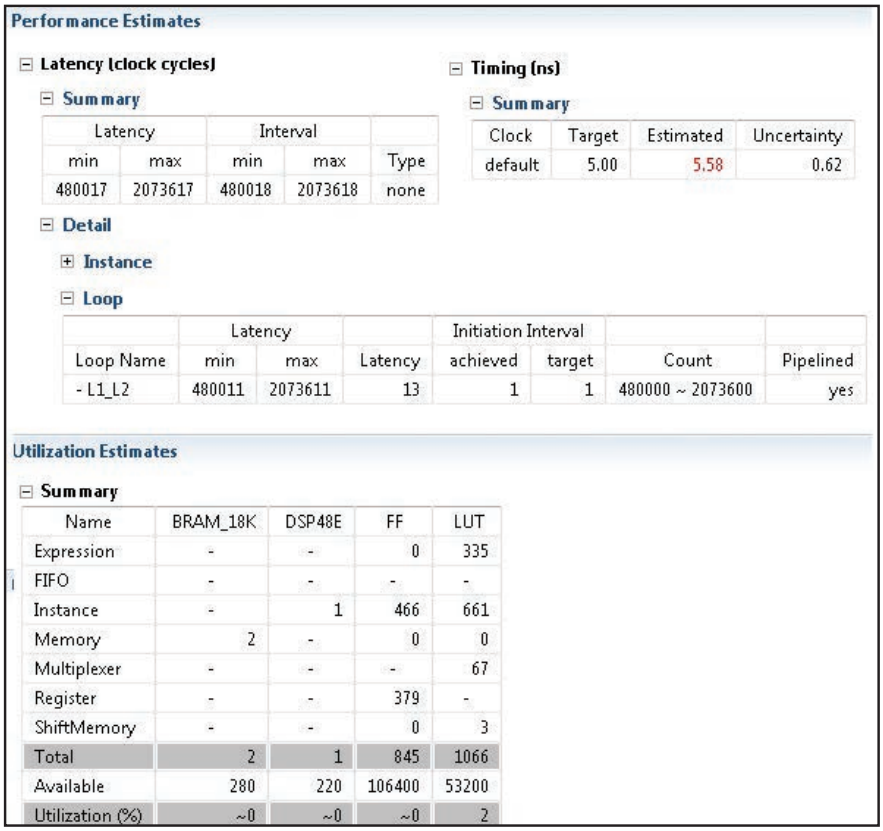
	Latency			Initiation Interval			
Loop Name	min	max	Latency	achieved	target	Count	Pipelined
- L1_L2	2400014	10368014	20	5	1	480000 ~ 2073600	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	698
FIFO	-	-	-	-
Instance	-	1	466	661
Memory	-	-	-	-
Multiplexer	-	-	-	154
Register	-	-	632	-
ShiftMemory	-	-	0	24
Total	0	1	1098	1537
Available	280	220	106400	53200
Utilization (%)	0	~0	1	2

Figure 6 – Vivado HLS performance estimate for the elementary reference median filter if it were to be used as an effective top function; throughput is far from optimal.



Performance Estimates

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
480017	2073617	480018	2073618	none

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	5.00	5.58	0.62

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Latency	achieved	target	Count	Pipelined
- L1_L2	480011	2073611	13	1	1	480000 ~ 2073600	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	335
FIFO	-	-	-	-
Instance	-	1	466	661
Memory	2	-	0	0
Multiplexer	-	-	-	67
Register	-	-	379	-
ShiftMemory	-	-	0	3
Total	2	1	845	1066
Available	280	220	106400	53200
Utilization (%)	~0	~0	~0	2

Figure 7 – Vivado HLS performance estimate for the top-level median filter function; the frame rate is 86.4 Hz, performance even better than what we need.

top and bottom outputs, performing a swap if necessary. The advantage of sorting networks over classical sorting algorithms is that the number of comparators is fixed for a given number of inputs. Thus, it's easy to implement a sorting network in the hardware of the FPGA. Figure 3 illustrate a sorting network for five samples (designed with Xilinx System Generator [6]). Note that the processing delay is exactly five clock cycles, independently of the value of the input samples. Also note that the five parallel output signals on the right contain the sorted data, with the maximum at the top and the minimum at the bottom.

Implementing a median filter via a sorting network in C language is straightforward, as illustrated in the code in Figure 4. The Vivado HLS directives are embedded in the C code itself (#pragma HLS). Vivado HLS requires only two optimization directives to generate optimal RTL code. The first is to pipeline the whole function with an initialization interval (II) of 1 in order to have the output pixel rate equal to the FPGA clock rate. The second optimization is to reshape the window of pixels into separate registers, thus improving the bandwidth by accessing the data in parallel all at once.

THE TOP-LEVEL FUNCTION

An elementary implementation of a median filter is shown in the code snippet in Figure 5, which we will use as a reference. The innermost loop is pipelined in order to produce one output pixel at any clock cycle. To generate a report with latency estimation, we need to instruct the Vivado HLS compiler (with the TRIPCOUNT directive) about the amount of possible iterations in the loops L1 and L2, since they are "unbounded." That is, the limits of those loops are the picture height and width, which are unknown at compile time, assuming the design can work at run-time on image resolutions below the maximum allowed resolution of 1,920 x 1,080 pixels.

```
void top_median(pix_t in_pix[MAX_HEIGHT][MAX_WIDTH],
                  pix_t out_pix[MAX_HEIGHT][MAX_WIDTH],
                  short int height, short int width)
{
#pragma HLS INTERFACE ap_vld register port=width
#pragma HLS INTERFACE ap_vld register port=height
#pragma HLS INTERFACE ap_fifo depth=2 port=out_pix
#pragma HLS INTERFACE ap_fifo depth=2 port=in_pix

  short int r, c; //row and col index
  pix_t pix, med, window[KMED*KMED], pixel[KMED];

  static pix_t line_buffer[KMED][MAX_WIDTH];
#pragma HLS ARRAY_PARTITION variable=line_buffer complete dim=1

  L1:for(r = 0; r < height; r++) {
  #pragma HLS LOOP_TRIPCOUNT min=600 max=1080 avg=720
    L2:for(c = 0; c < width; c++)
    {
#pragma HLS LOOP_TRIPCOUNT min=800 max=1920 avg=1280
#pragma HLS PIPELINE II=1

    // Line Buffers fill
    for(int i = 0; i < KMED-1; i++)
    {
       line_buffer[i][c] = line_buffer[i+1][c];
       pixel[i] = line_buffer[i][c];
    }
    pix = in_pix[r][c];
    pixel[KMED-1]=line_buffer[KMED-1][c]=pix;

    // sliding window
    for(int i = 0; i < KMED; i++)
       for(int j = 0; j < KMED-1; j++)
          window[i*KMED+j] = window[i*KMED+j+1];

    for(int i = 0; i < KMED; i++)
       window[i*KMED+KMED-1] = pixel[i];

    // Median Filter
    med = median(window);

    if v( (r>=KMED-1)&&(r<height)&&
         (c>=KMED-1)&&(c<=width) )
       pix = med;
    else
       pix = 0;

    if(r>0 && c>0)
       // KKMED == 1 for 3x3 window
       // KKMED == 2 for 5x5 window
       out_pix[r-KKMED][c-KKMED] = pix;

   } // end of L2 loop
   } // end of L1 loop
} // end of function
```

Figure 8 – New top-level C code accounting for the behavior of video line buffers

In the C code, the window of pixels to be filtered accesses different rows in the image. Therefore, the benefits of using memory locality to reduce memory bandwidth requirements are limited. Although Vivado HLS can synthesize such code, the throughput will not be optimal, as illustrated in Figure 6. The initialization interval of the loop L1_L2 (as a result of the complete unrolling of the innermost loop L2, automatically done by the HLS compiler) is five clock cycles instead of one, thus leading to an output data rate that does not allow real-time performance. This is also clear from the maximum latency of the whole function. At a 5-nanosecond target clock period, the number of cycles to compute an output image would be 10,368,020, which means a 19.2-Hz frame rate instead of 60 Hz. As detailed in [7], the Vivado HLS designer must explicitly code the behavior of video line buffers into the C-language model targeted for RTL generation, because the HLS tool does not automatically insert new memories into the user code.

The new top-level function C code is shown in Figure 8. Given the current pixel at coordinates (row, column) shown as in_pix[r][c], a sliding window is created around the output pixel to be filtered at coordinates (r-1, c-1). In the case of a 3x3 window size, the result is out_pix[r-1][c-1]. Note that in the case of window sizes 5x5 or 7x7, the output pixel coordinates would be respectively (r-2, c-2) and (r-3, c-3). The static array line_buffer stores as many KMED video lines as the number of vertical samples in the median filter (three, in this current case), and the Vivado HLS compiler automatically maps it into one FPGA dual-port Block RAM (BRAM) element, due to the static C keyword.

It takes few HLS directives to achieve real-time performance. The innermost loop L2 is pipelined in order to produce one output pixel at any clock cycle. The input and output image arrays in_pix and out_pix are mapped as FIFO

Vivado HLS Report Comparison

Performance Estimates

Timing (ns)

Clock		sol2_med_3x3	sol2_med_5x5	sol2_med_7x7
default	Target	5.00	5.00	5.00
	Estimated	3.37	3.37	3.37

Latency (clock cycles)

		sol2_med_3x3	sol2_med_5x5	sol2_med_7x7
Latency	min	8	24	48
	max	8	24	48
Interval	min	1	1	1
	max	1	1	1

Utilization Estimates

	sol2_med_3x3	sol2_med_5x5	sol2_med_7x7
BRAM_18K	0	0	0
DSP48E	0	0	0
FF	457	3673	14257
LUT	656	5424	21216

Figure 9 – Vivado HLS performance estimates comparison for the median filter function standalone in cases of 3x3, 5x5 and 7x7 window sizes.

Vivado HLS Report Comparison

All Compared Solutions

sol2_med_3x3_11bit : xc7z020clg484-1

sol2_med_3x3_k7 : xc7k325tffg900-2

Performance Estimates

Timing (ns)

Clock		sol2_med_3x3_11bit	sol2_med_3x3_k7
default	Target	5.00	2.00

Latency (clock cycles)

		sol2_med_3x3_11bit	sol2_med_3x3_k7
Latency	min	480019	480028
	max	2073619	2073628
Interval	min	480020	480029
	max	2073620	2073629

Utilization Estimates

	sol2_med_3x3_11bit	sol2_med_3x3_k7
BRAM_18K	4	2
DSP48E	1	1
FF	1156	1351
LUT	1407	1046

Figure 10 – Vivado HLS comparison report for the 3x3 top-level function in case of either 11 bits on the 7Z02 or 8 bits on a 7K325 device.

streaming interfaces in the RTL. The line_buffer array is partitioned into KMED separate arrays so that the Vivado HLS compiler will map each one into a separate dual-port BRAM; that increases the number of load/store operations due to the fact that more ports are available (every dual-port BRAM is capable of two load or store operations per cycle). Figure 7 shows the Vivado HLS performance estimate report. The maximum latency is now 2,073,618 clock cycles. At an estimated clock period of 5.58 ns, that gives us a frame rate of 86.4 Hz. This is more than what we need! The loop L1_L2 now exhibits an II=1, as we wished. Note the two BRAMs that are needed to store the KMED line buffer memories.

ARCHITECTURAL EXPLORATION WITH HIGH-LEVEL SYNTHESIS

One of the best features of Vivado HLS, in my opinion, is the creative freedom it offers to explore different design architectures and performance trade-offs by changing either the tool's optimization directives or the C code itself. Both types of operations are very simple and are not time-consuming.

What happens if we need a larger median filter window? Let's say we need 5x5 instead of 3x3. We can just change the definition of KMED in the C code from "3" to "5" and run Vivado HLS again. Figure 9 shows the HLS comparison report for the synthesis of the median filter routine standalone in the case of 3x3, 5x5 and 7x7 window sizes. In all three cases, the routine is fully pipelined (II=1) and meets the target clock period, while the latency is respectively 9, 25 and 49 clock cycles, as one expects from the sorting network's behavior. Clearly, since the amount of data to be sorted grows from 9 to 25 or even 49, the utilization resources (flip-flops and lookup tables) also grow accordingly.

Because the standalone function is fully pipelined, the latency of the top-level function remains constant, while the clock frequency slightly decreases when increasing the window size.

So far we have only discussed using the Zynq-7000 All Programmable SoC as the target device, but with Vivado HLS, we can easily try different target devices in the same project. For example, if we take a Kintex®-7 325T and synthesize the same 3x3 median filter design, we can get a place-and-route resource usage of two BRAMs, one DSP48E, 1,323 flip-flops and 705 lookup tables (LUTs) at a 403-MHz clock and data rate, whereas on the Zynq SoC, we had two BRAMs, one DSP48E, 751 flip-flops and 653 LUTs at a 205-MHz clock and data rate.

Finally, if we want to see the resource utilization of a 3x3 median filter working on a gray image of 11 bits per sample instead of 8 bits, we can change the definition of the pix_t data type by applying the ap_int C++ class, which specifies fixed-point numbers of any arbitrary bit width. We just need to recompile the project by enabling the C preprocessing symbol GRAY11. In this case, the resource usage estimate on the Zynq SoC is four BRAMs, one DSP48E, 1,156 flip-flops and 1,407 LUTs. Figure 10 shows the synthesis estimation reports of these last two cases.

FEW DAYS OF WORK

We have also seen how easy it can be to generate timing and area estimations for median filters with different window sizes and even with a different number of bits per pixel. In particular, in the case of a 3x3 (or 5x5) median filter, the RTL that Vivado HLS automatically generates consumes just a small area on a Zynq SoC device (-1 speed grade), with an FPGA clock frequency of 206 (or 188, for the 5x5 version) MHz and an effective data rate of 206 (or 188) MSPS, after place-and-route.

The total amount of design time to produce these results was only five working days, most of them devoted to building the MATLAB® and C models rather than running the Vivado HLS tool itself; the latter task took less than two days of work.

References

1. "Weighted Median Filters: a Tutorial," Lin Yin, Ruikang Yang, M. Gabbouj, Y. Nuevo, in *IEEE Transactions on Circuits and Systems II, Analog and Digital Signal Processing*, Volume 43, Issue 3, March 1996, pp 157-192

2. "Adaptive Window Multistage Median Filter for Image Salt-and-Pepper Denoising," Weiyang Mul, Jing Jin, Hongqi Feng, Qiang Wang, in Proceedings of 2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), May 2013, pp 1535-1539

3. "Vivado Design Suite User Guide: High-Level Synthesis," UG902 (v2013.2), June 2013

4. "Zynq-7000 All Programmable SoC: Technical Reference Manual," UG585 (v1.6.1) September 2013

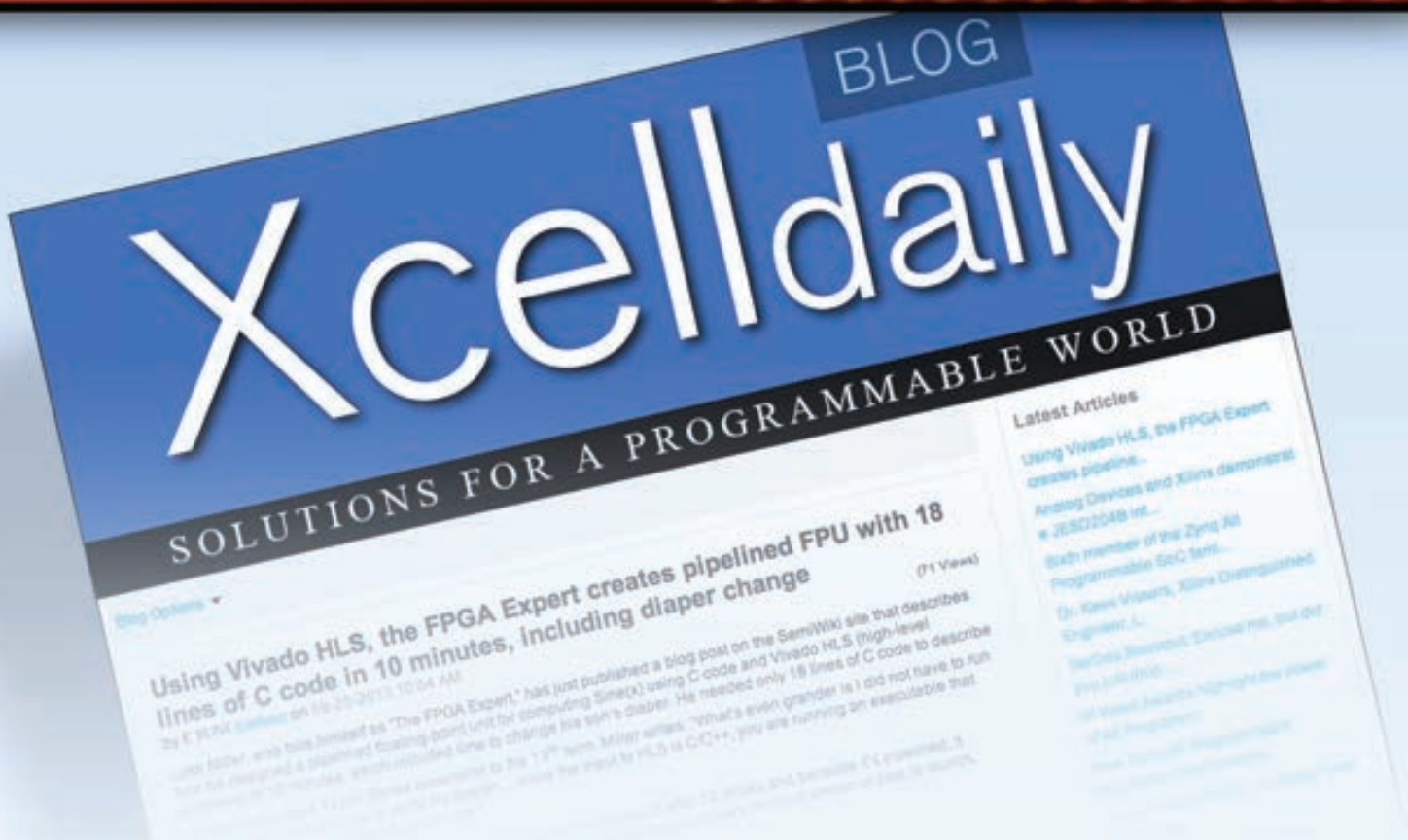
5. S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, W. H. Press, *Numerical Recipes in C, the Art of Scientific Computing*, Cambridge University Press, 1992, second edition, ISBN 0 521 43108 5

6. "Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator," UG897 (v2013.2), June 2013

7. F.M. Vallina, "Implementing Memory Structures for Video Processing in the Vivado HLS Tool," XAPP793 (v1.0), September 2012

Daniele Bagni is a DSP Specialist FAE for Xilinx EMEA in Milan, Italy. After earning a university degree in quantum electronics from the Politecnico of Milan, he worked for seven years at Philips Research labs in real-time digital video processing, mainly motion estimation for field-rate upconverters. For the next nine years Daniele was project leader at STMicroelectronics' R&D labs, focusing on video-coding algorithm development and optimization for VLIW-architecture embedded DSP processors, while simultaneously teaching a course in multimedia information coding as an external professor at the State University of Milan. In 2006 he joined the Xilinx Milan sales office. What Daniele enjoys most about his job is providing customers with feasibility studies and facing a large variety of DSP applications and problems. In his spare time, he likes playing tennis and jogging.

Xcell Journal Adds New Daily Blog



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

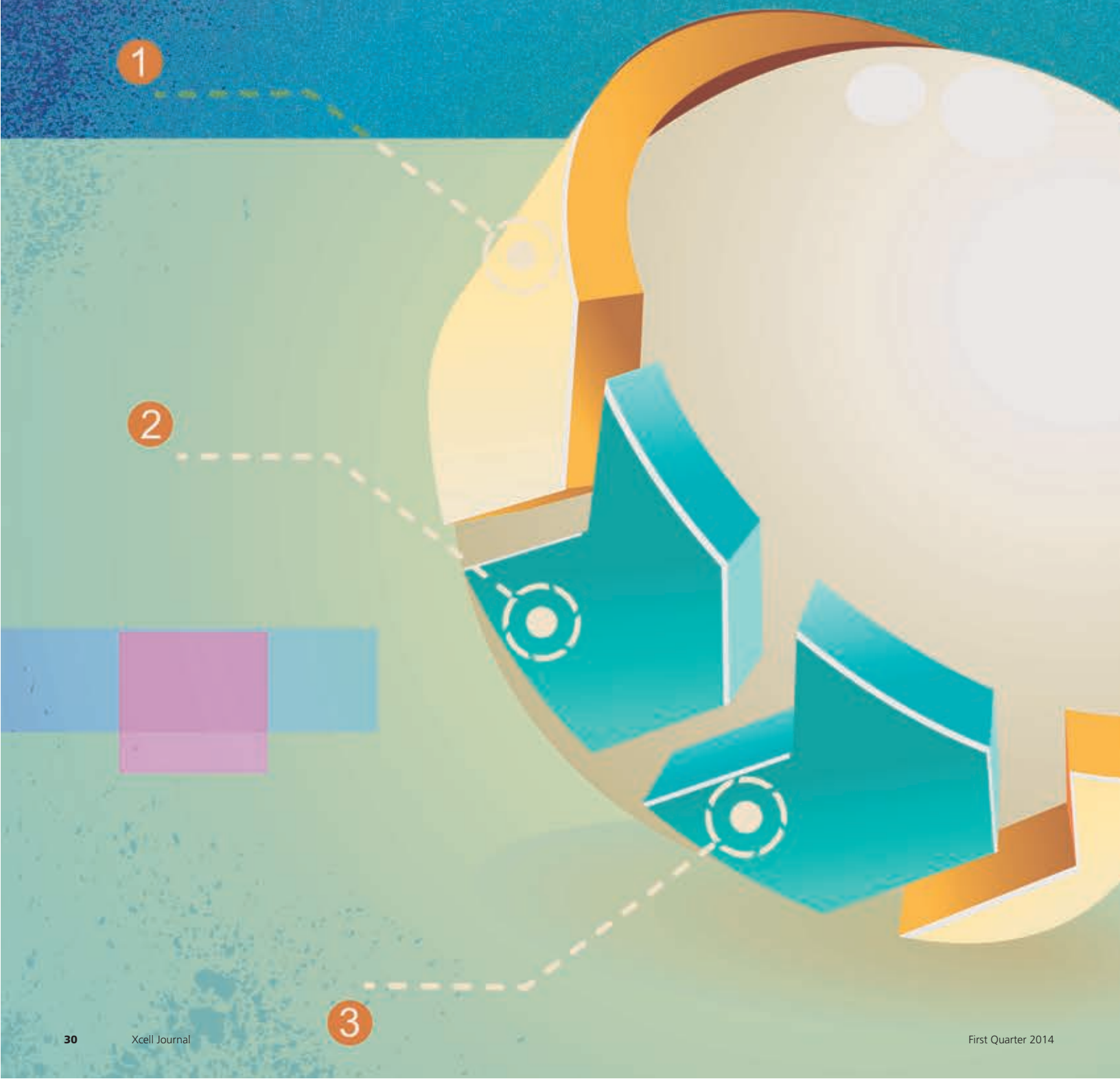
Recent

- *Want the Real Skinny on Commercial 3D ICs? Xilinx's Vincent Tong Forecasts The Future on 3DInCites*
- *SemiWiki's Luke Miller: FPGA Training is the Key to Success. Xilinx and the Free University of FPGA*
- *Fast-Booting Linux on the Zynq SoC: The 13.5-second Excitement (+/- a hundred msec or so)*
- *Zynq-7100 All Programmable SoC Voted "Best FPGA" by readers of Electronic Engineering & Product World in China*
- *Zynq-based ZedBoard + CMOSIS Super35 Image Sensor + Apertus Engineering = Axiom Open 4K Cinema Camera*

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

XILINX
ALL PROGRAMMABLE.

Ins and Outs of Creating the Optimal Testbench



Verifying that your RTL module or FPGA meets its requirements can be a challenge. But there are ways to optimize this process to ensure success.

by Adam P. Taylor
Head of Engineering - Systems
e2v
aptaylor@theiet.org

Verification of an FPGA or RTL module can be a time-consuming process as the engineer strives to ensure the design will function correctly against its requirement specification and against corner cases that could make the module go awry. Engineers traditionally achieve this verification using a testbench, a file that you will devise to test your design. However, testbenches can be simple or complicated affairs. Let us have a look at how we can get the most from our testbench without overcomplicating it.

WHAT IS VERIFICATION?

Verification ensures the unit under test (UUT) meets both the requirements and the specification of the design, and is thus suitable for its intended purpose. In many cases a team independent of the design team will perform verification so as to bring a fresh eye to the project. This way, the people who design the UUT are not the ones who say it works.

With the size and complexity of modern FPGA designs, ensuring that the UUT performs against its specification can be a considerable task. The engineering team must therefore determine at the start of the project what the verification strategy will be. Choices can include a mix of the following tactics:

- Functional simulation only—This technique checks whether the design is functionally correct.
- Functional simulation and code coverage—This method checks that along with the functional correctness, all the code within the design has been tested.
- Gate-level simulation—This technique likewise verifies the functionality of the design. When back-annotated with timing information from the final implemented design, this type of simulation can take considerable time to perform.
- Static timing analysis—This method analyzes the final design to ensure the module achieves its timing performance.
- Formal equivalence checking—Engineers use this technology to check the equivalence of netlists against RTL files.

A boundary condition is a when one of the inputs is at its extreme value, while a corner case is when all inputs are at their extreme value.

Whatever the verification strategy, the engineering team will need to produce a test plan that shows how the individual modules and the final FPGA are to be verified and all requirements addressed.

SELF-CHECKING TESTBENCHES

The testbench that you devise to stimulate the UUT can be either self-checking or not. If you opt for a non-self-checking variety, then you will be required to visually confirm that the testbench functions as desired by eyeballing the test results on a computer monitor. A self-checking testbench differs from a traditional testbench in that along with applying the stimulus, it also checks the outputs from the UUT against expected results, as shown in Figure 1. This allows you to state categorically that the UUT has passed or failed. Couple this capability with the control and reporting via text files, and you can create a very powerful verification tool.

A self-checking testbench offers a number of advantages. For starters, it can be very time-consuming and complicated to visually verify simulation waveforms. A self-checking testbench spares you that labor. Then too, self-checking testbenches can provide an overall pass-or-fail report, which you can save and use as test evidence later in the design flow. And should the UUT require design iterations at a later stage, it will take you far less time to rerun the testbench and determine pass or fail—a process often called regression testing—than would be the case if you had used a traditional testbench.

CORNERS, BOUNDARIES AND STRESS

When using a testbench, the aim is to ensure the module performs as per the functional requirements, that any corner cases are addressed and, most important, that the testbench sufficiently exercises the UUT code. It is therefore common when verifying design modules to use "glass box" testing methodologies, so-called because the contents of the UUT module are known and observed. By contrast, when the UUT is the FPGA, you would use top-level black-box testing due to the increased verification and simulation times at the top level.

To verify the functionality of the design against requirements, the testbench must apply the same kinds of

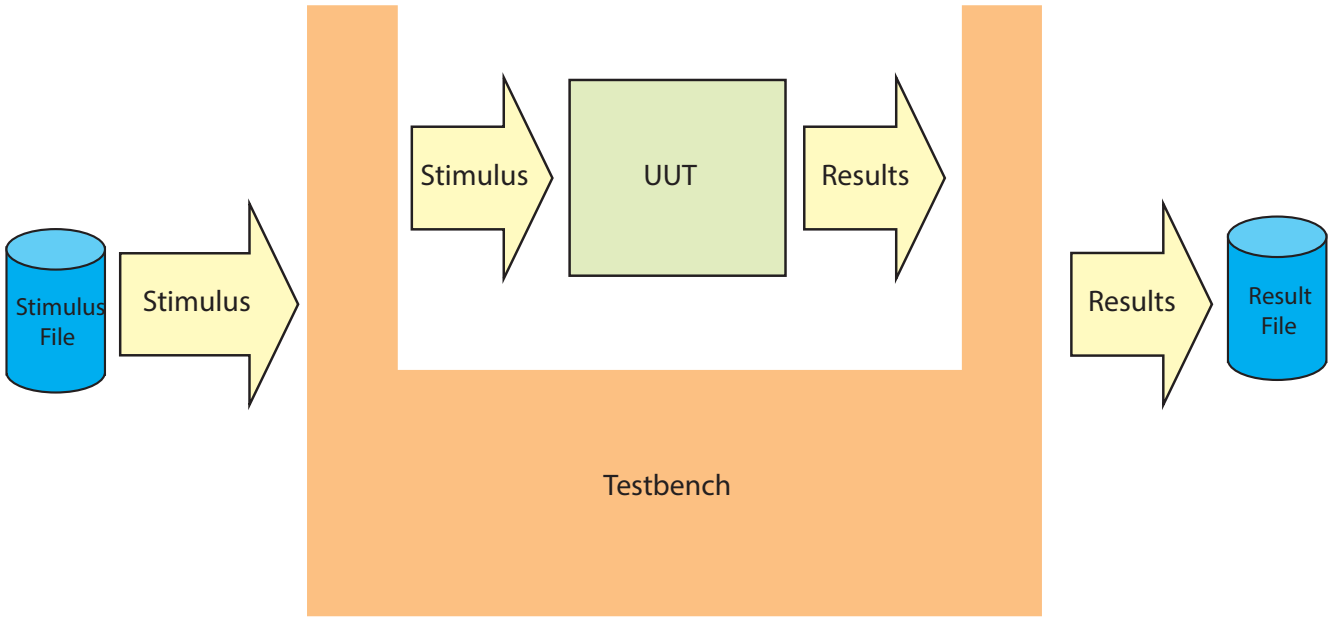


Figure 1 – Self-checking testbench architecture

stimulus the modules expect to see in operation. However, testing all possible inputs to prove functional compliance can be a time-consuming process. For this reason, engineers will often focus upon boundary conditions and corner cases along with testing some typical operating values.

A boundary condition is when one of the inputs is at its extreme value, while a corner case is when all inputs are at their extreme value. A good example of both of these situations is a simple 16-bit adder that sums two numbers and produces a result. In this case, the boundary condition and corner cases are illustrated in Figure 2.

As you can clearly see, the corner cases would be when both A and B inputs equal 0; when A equals 0 and B equals 65535; when A and B equal 65535; and finally, when A equals 65535 and B, 0. The boundary conditions are the values between those corner cases.

In some applications you may also want to stress-test the UUT to ensure that there's a margin above and beyond the normal operation. Stress-testing will change depending upon the module, but it could involve repeated operation, attempting to overflow buffers and FIFOs. Stress-testing a design allows the verification engineer to have a little fun trying to break the UUT.

CODE COVERAGE

The verification team also requires a metric to ensure the testbench has properly exercised the UUT. This metric is typically provided by using code coverage to make sure the UUT is correctly exercised. When looking at code coverage, there are a number of parameters to consider.

- Each executable statement is examined to determine how many times it is executed.
- All possible IF, CASE and SELECT branches are executed.
- Conditions and subconditions within a code branch are tested to see what condition caused this branch to be true.
- All paths through the HDL are traversed to identify any untraveled paths.
- Signals in the sensitivity list of VHDL processes and wait statements are monitored to ensure all possible trigger conditions are tested.

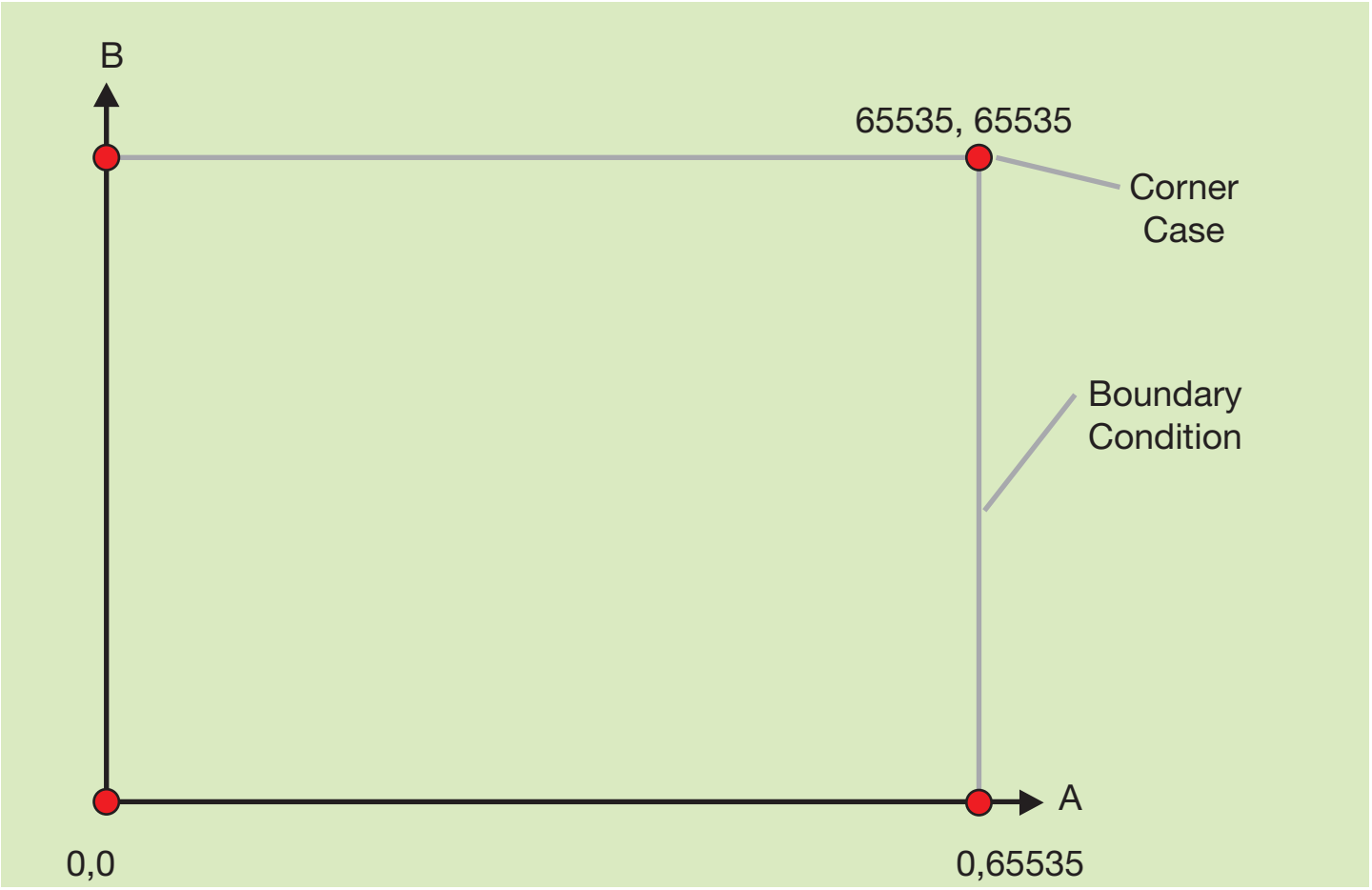


Figure 2 – Boundary condition and corner cases for a 16-bit adder

Achieving 100 percent in the above parameters will not prove the UUT is meeting its functional requirements. However, it will easily identify sections of the UUT that the testbench has not exercised. It is very important at this point to mention that code coverage only addresses what is within the UUT.

TEST FUNCTIONS AND PROCEDURES

The verification team should create a library of commonly used test routines that all members of the team can employ for this project and even across other projects. Possessing this library will not only speed up your development, it will also deliver a standardized output from the testbench. This standardized output aids both analysis and verification of the module.

You can use these standard functions and procedures for numerous applications. A few commonly used methods are:

- Stimulus generators: for instance, a standard method of testing reset application and release or of driving a communication interface
- Output checking: checking the output result against an expected result and reporting via the transcript or file
- Models: standardized models of devices with which the FPGA or module will interface during operation
- Logging functions: a standardized reporting format for the results to enable easier analysis. Logging functions also provide evidence of verification of requirements that can be used to demonstrate compliance.

Good output-checking functions can make use of the VHDL signal attributes "stable," "delayed," "last_value and" "last_event." These attributes can be very important when confirming the UUT is compliant with the timing interfaces required for memories, or with other interfaces. These functions can also be useful for ensuring the UUT has achieved setup-and-hold times, and for reporting violations if it has not.

BEHAVIORAL MODELS

Sometimes you can verify the performance of a UUT against a nonsynthesizable behavioral model that implements the same function as the UUT (see Figure 3). The model and the UUT

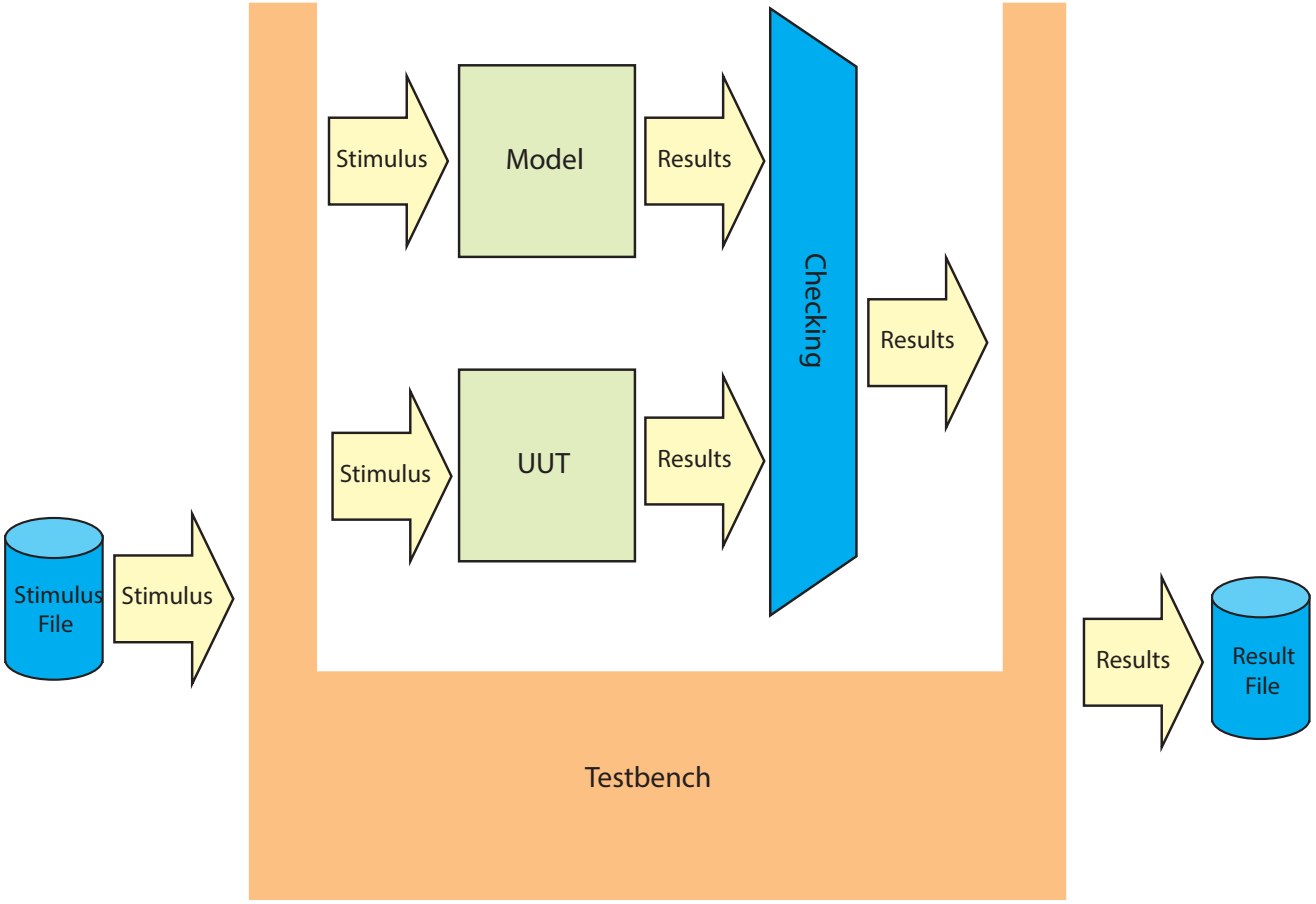


Figure 3 — Model checking against a UUT

Verifying a design can be and often is a larger task than creating the implementable design itself. By carefully thinking about the verification strategy early in the project life cycle, teams can put in place the necessary test plans and infrastructure to support this crucial process.

are then subjected to the same stimulus from the testbench, and the outputs of the two modules are compared on a cycle-by-cycle basis to ensure both behave in the same manner. The model or testbench may need to account for the latency of the UUT to allow comparison of the correct results.

USING TEXTIO FILES

If you are using VHDL to verify your design, you can use the TextIO package to read in stimulus vectors from a text file and record the results, including pass or fail, within a results text file. The verification engineering team is then able to create a number of stimulus test files to test the UUT. This approach is very beneficial, as changing the stimulus becomes as simple as updating a text file. This allows for easy and quick updates, while also offering the ability to test scenarios that could be causing issues in the lab or on the final hardware.

Within VHDL, the STD testing package supports interaction with text files and—if your simulator supports VHDL 2008—allows the use of std_logic and std_logic_vector. If your simulator does not yet support VHDL 2008, you can always fall back on the nonstandard but commonly used std_logic_textio package to provide similar support. Verilog provides the same kinds of tools to read and write text files.

WHAT ELSE MIGHT WE CONSIDER?

The testbench should be time-agnostic, which means that it should be capable of testing either RTL or the behavioral UUT along with post-place-and-route netlists back-annotated with SDF timing information. While simulating at a gate level can take considerable time due to the required resolution, at times it is necessary.

The verification team should also consider the use of Tcl scripts to control the simulation tool as it executes the test. This approach allows for a more repeatable process, as all options that might be set within a GUI and change between simulation runs will be defined within the Tcl file controlling the simulation. These files also become self-documenting, reducing the documentation needed to support the verification of the UUT.

PLANS AND INFRASTRUCTURE

Verifying a design can be and often is a larger task than creating the implementable design itself. By carefully thinking about the verification strategy early in the project life cycle, the design and verification teams can put in place the necessary test plans and infrastructure to support this crucial process. Taking into account the points above can help you create a testbench that provides flexibility for the task at hand.

How to Add an RTOS to Your Zynq SoC Design

by Adam P. Taylor
Head of Engineering—Systems
e2v
aptaylor@theiet.org

To get the maximum advantage from the Zynq All Programmable SoC, you will need an operating system. Here's how to install a real-time version—specifically, the µC/OS-III.

In the quest to gain the maximum benefit from the processing system within a Xilinx® Zynq®-7000 All Programmable SoC, an operating system will get you further than a simple bare-metal solution. Anyone developing a Zynq SoC design has a large number of operating systems to choose from, and depending upon the end application you may opt for a real-time version. An RTOS is your best choice if you are using the Zynq SoC in industrial, military, aerospace or other challenging environments where response times and reliable performance are required to prevent loss of life or injury, or to achieve strict performance goals.

To get a feel for how best to add an RTOS to our Zynq SoC system, we will be using one of the most popular real-time operating systems around, the µC/OS-III from Micrium. This RTOS or earlier versions of it have been used on a number of very exciting systems, including the Mars Curiosity Rover. The latest version is currently in the process of being certified for MISRA-C, DO178B Level A, SIL3/4 and IEC61508 standards, which means it should have a wide appeal to many Zynq SoC users. But before going into the implementation details, it's helpful to review the basics of real-time operating systems.

WHAT IS A REAL-TIME OPERATING SYSTEM?

What makes a real-time operating system different from a standard operating system? Well, a real-time operating system is deterministic, which means that the system responds within a defined deadline. This determinism can be important for a number of reasons, for instance if the end application is monitoring an industrial process and has to respond to events within a specified period of time, as would be the case for an industrial control system.

RTOSes are further subdivided based upon their ability to meet these deadlines. This categorization gives rise to three distinct types of RTOS, each of which addresses the concept of deadlines differently. In the hard RTOS, missing a deadline is seen as a system failure. That's not the case for the firm RTOS, where an occasional missed deadline is acceptable. In the soft RTOS, meanwhile, missing a deadline reduces the usefulness of the results, but the system as a whole can tolerate this.

Real-time operating systems revolve around the concept of running tasks (sometimes called processes), each of which performs a required function. For example, a task might read in data over an interface or perform an operation on that data. A simple system may employ just one task, but it

With time sharing, each task gets a dedicated time slot on the processor. Higher-priority tasks can be allocated multiple time slots.

is more likely for multiple tasks to be running on the processor at any one time. Switching between these tasks is referred to as "context switching," and it requires that the state of the processor associated with each task be stored and added to the task stack.

The process of determining which task is to run next is controlled by the kernel (the core of the RTOS that manages input/output requests from software and translates them into data-processing instructions for the central processing unit and functional elements in the processor). Task scheduling can be complicated, especially if we want to avoid deadlock, a state in which two or more tasks lock each other out. The two basic methods are time sharing and event-driven. With time sharing, each task gets a dedicated time slot on the processor. Higher-priority tasks can be allocated multiple time slots. This time slicing is controlled via a regular interrupt or timer, and is often called "round-robin scheduling." With an event-driven solution, tasks are switched only when one with a higher priority is required to run. This is often called "preemptive scheduling."

DEADLOCKS, RESOURCE SHARING AND STARVATION

When two or more processes need to use the same resource—such as a UART, ADC or DAC—it is possible for them to request this resource at the same time. In such a situation, access needs to be controlled in order to prevent contention. How this is managed is very important. Without the correct management, issues such as "deadlock" or "starvation" might occur, resulting in system failure.

Deadlock occurs when a process is holding one resource and cannot release it, because it is unable to complete its task. It requires another resource that is currently being held by another process. Since the system will remain in this state indefinitely, the application is said to be deadlocked. As you can imagine, deadlock is a bad situation for a real-time operating system to find itself in.

Starvation occurs when a process cannot run because the resources it needs are always allocated to another process.

It probably won't surprise you to hear that there have been a lot of things written on these subjects over the years, and there are many proposed solutions, such as the Dekker algorithm, a classic fix for the mutual-exclusion problem in concurrent programming. The most commonly used method to handle these kinds of situations is semaphores, which are usually of two types—binary semaphores and counting semaphores.

Typically, each resource has a binary semaphore allocated to it. A requesting process will wait for the resource to become available before executing. Once the task is completed, the requesting process will release the resource. These semaphores are commonly known as WAIT and SIGNAL operations. A process will WAIT on a semaphore. If the resource is free, the process will then be given control of that resource and it will run until completed, at which point it will SIGNAL

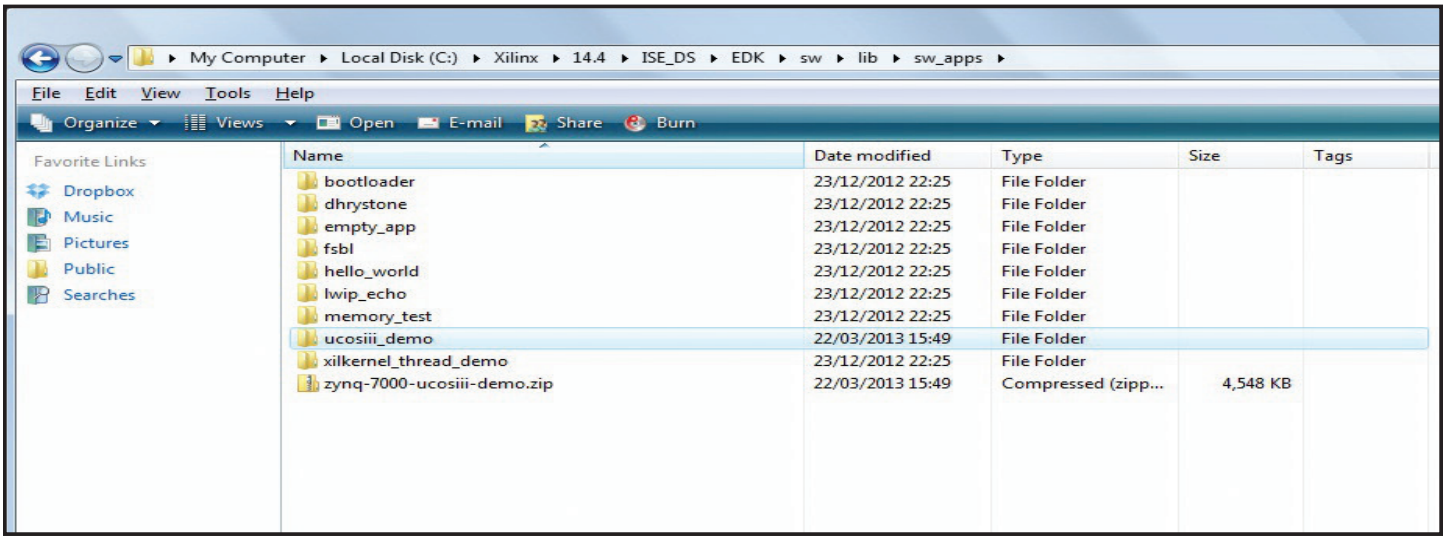


Figure 1 – The directory structure showing the location of the demo files

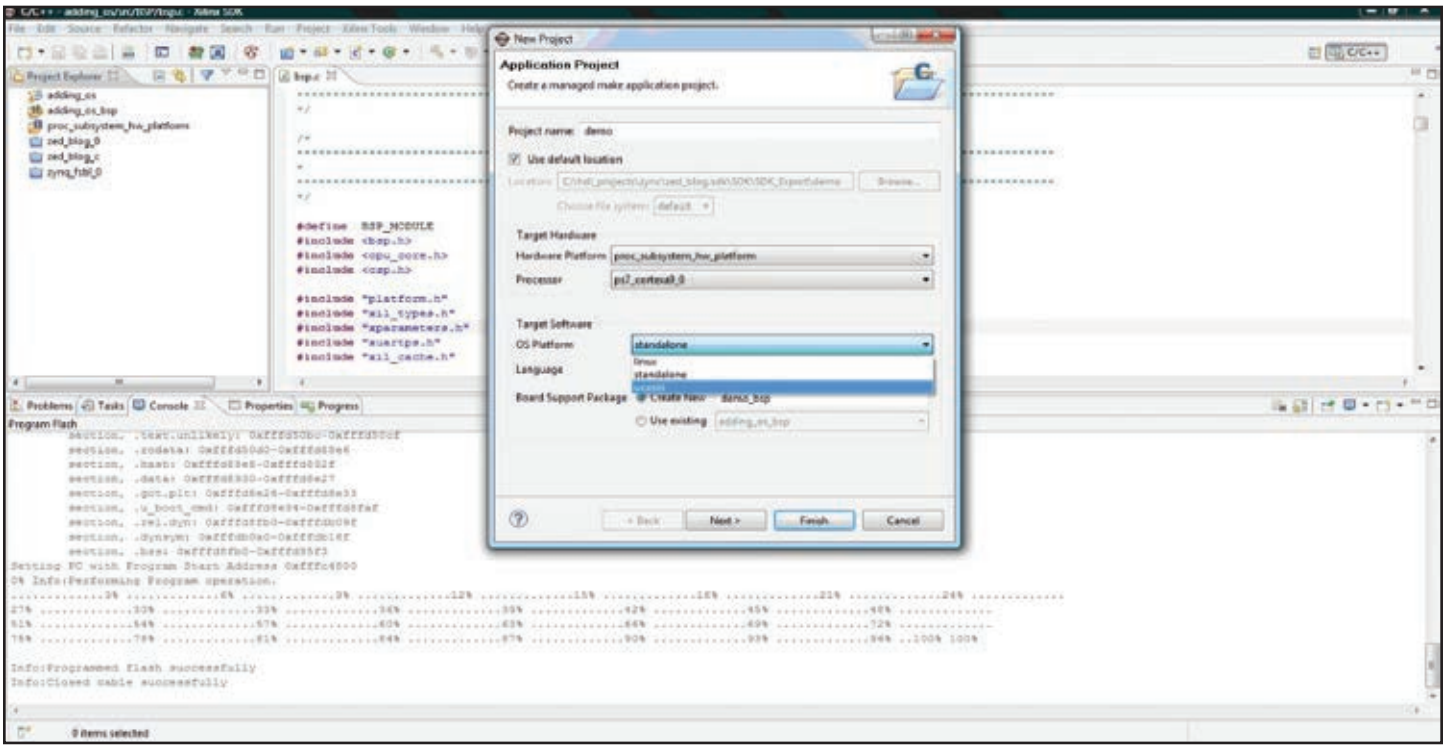


Figure 2 – Selecting the operating system

completion. However, if the resource is already occupied when the process WAITs on the semaphore, then the process will be suspended until the resource becomes free. This could occur as soon as the currently executing process is finished, but there could be a longer wait if this process is preempted by a process of higher priority. A special class of binary semaphores called mutexes (derived from the term "mutual exclusion") are often used to prevent priority inversion.

Counting semaphores work in the same way as binary semaphores, but they are used when more than one instance of a particular type of resource is available (for example, data stores). As each of the resources is allocated to a process, the count is reduced to show the number of resources remaining free. When the count gets to zero, there are no more resources available, and the requesting process will be suspended until one of the resources is released.

It is often necessary for processes to communicate with one another. There are a number of methods that

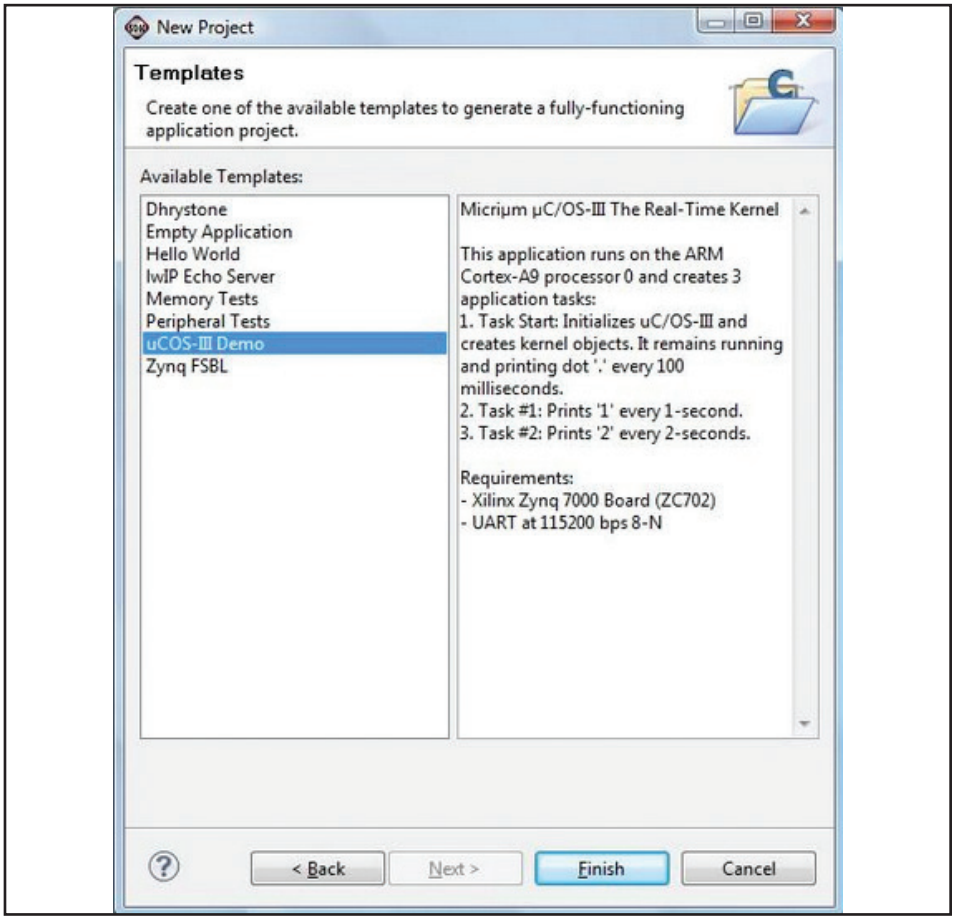


Figure 3 – Selecting the µCOS-III demo

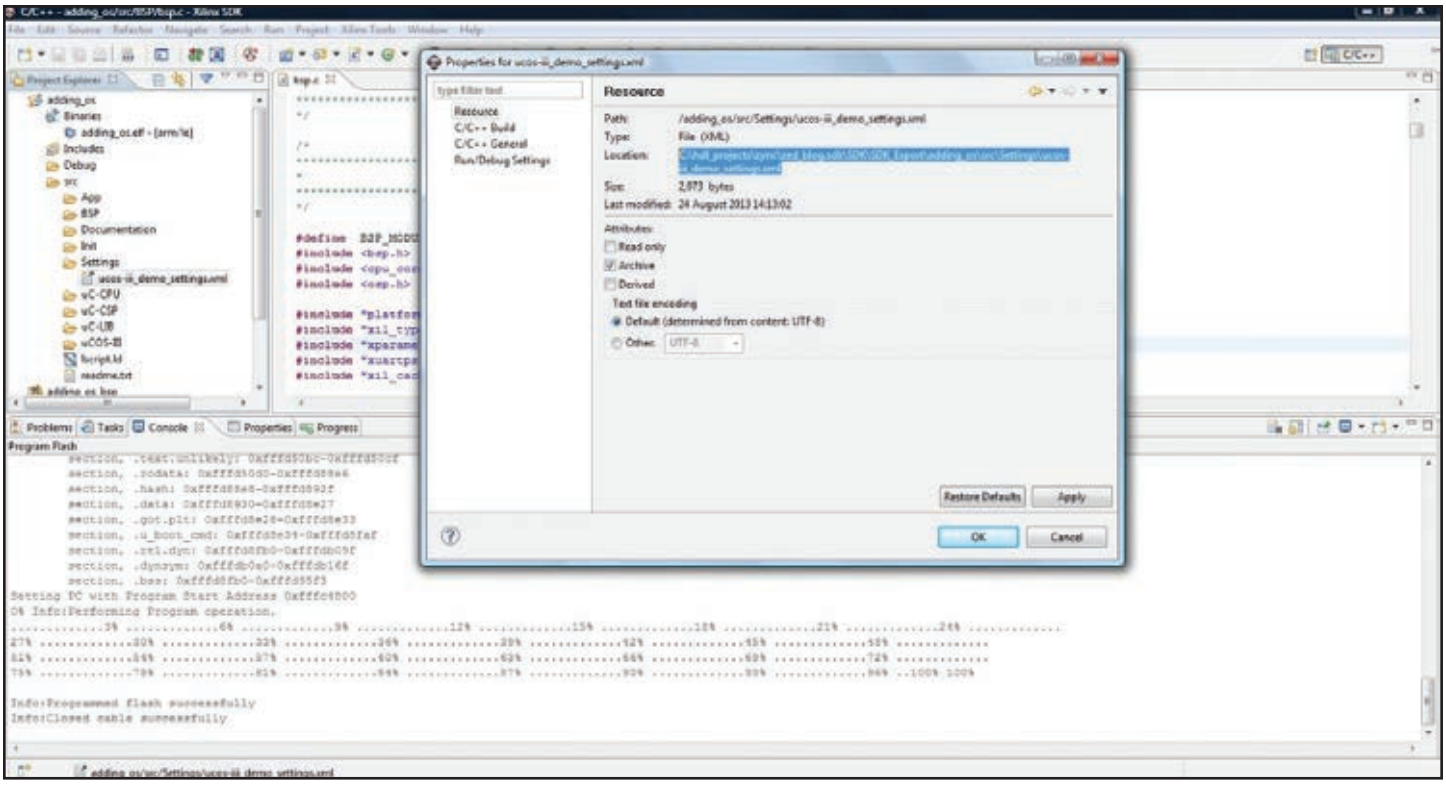


Figure 4 – Getting the settings correct

can be employed, the simplest of which is to use a data store and semaphores as described above. More complex techniques include message queues. With message queues, when one process wishes to send information to another process, it POSTs a message to the queue. When a process wishes to receive a message from a queue, it PENDs on the queue. Message queues therefore work like a FIFO (first-in, first-out) memory.

THE µC/OS-III OPERATING SYSTEM

Micrium's µC/OS-III is a preemptive RTOS, which means it will always run the task with the highest priority that is ready to execute. The first step in adding it to your Zynq SoC system design is to download the µC/OS-III RTOS from the Micrium website. Once you've done that, the installation is very straightforward. You just need to extract a few ZIP files into the correct folders (directories) under your Xilinx installation on your computer.

Make sure that you extract the ZIP file named Zynq-7000-ucosiii-bsp.zip into your \<XILINX> \ISE_DS\EDK\ sw\ lib\bsp\ folder. You will observe a number of the other operating systems under this folder, including the standalone and the xilkernel. Next, extract the ZIP file named Zynq-7000-ucosiii-demo.zip into your \<XILINX> \ISE_DS\EDK\sw\lib\sw_apps\ folder, as shown in Figure 1. Once again, you will see a number of other application demos within this folder.

Having installed these two sets of files, we are ready to begin creating our project within the software development kit (SDK). We will be using the same base hardware that was created before, but we will require a new application and board support package (BSP), since we wish to include the operating system.

Within the SDK, close all open projects except the base hardware design. Next, select the File > New > Application Project option, give the new project a name and select the operating system µC/OS-III (see Figure 2). Then select the demo application for µC/OS-III (see Figure 3)

Once you are happy, click the Finish button. The application and board support package (if you choose that option) will be created within the SDK. If you have the Auto Build option selected, you may find that a few errors are reported. This is because not all the project references are correct yet. To set these project references, you need to import the demo settings, which you will find under the Project > Src > Settings option. Right-click on this XML file and view the Properties. This will allow you to select and copy the location of this file, as Figure 4 illustrates.

Once you have copied this location, right-click on the project and select Properties. Under the heading C/C++ General, select the Paths and Symbol options. Then select Import Settings and paste in the location of the settings file.

It is also important to ensure that the repositories are correctly pointing to the libraries you added earlier. You can check

these by setting Xilinx Tools > Repositories, which should show the location where you installed the µC/OS-III BSP earlier.

Since we wish to use the UART to output the status of the demo—showing the initialization being completed and the tasks running—you may need to set the stdin and stdout to the UART under the BSP settings.

Having performed these actions, you will see that the project can now be built. However, there will still be a few warnings, and if you tried to run this project on your hardware it would not perform as the demo states it should. This is because of a warning over undeclared functions. Including the following statement within the bsp.c file should correct this issue.

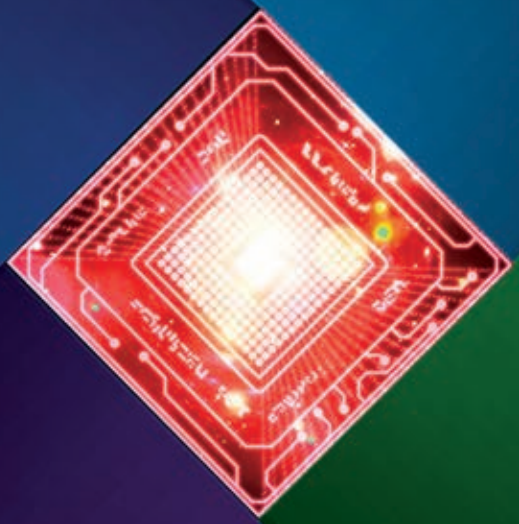
```
#include "xil_cache.h
```

Once I added this "include" header file, the project built and ran as expected on my ZedBoard (see my YouTube video at *http://www.youtube.com/watch?v=uRB4La5ijrA*).

UP AND RUNNING

Having got the example up and running, you now have confidence that the RTOS has been implemented correctly on your system. Now you can proceed to correctly implement the software design on the Zynq SoC. Once you have created the software application and the engineering team is ready to try it out on the hardware, you can create a programming file in exactly the same way as you would for a bare-metal system (see *Xcell Journal* issue 83, "How to Configure Your Zynq SoC Bare-Metal Solution," *http://issuu.com/xcelljournal/docs/xcell_journal_issue_83/40?e=2232228/2101904*), enabling the application with RTOS to boot and execute from the configuration memory.

How to Make a Custom XBD File for Xilinx Designs



Creating a custom Xilinx Board Description file can be a timesaver and can help keep your design project on track. It's not difficult to develop one for any board you are designing.

by Manish Nalamwar, Scientist "D"
Radar Seeker Laboratoryz
Research Centre IMARAT
Defence Research Development Organization
Hyderabad, India
nalamwar.manishkumar@rcilab.in

FPGA vendors provide a number of good evaluation and application-specific boards for those wishing to assess their FPGAs or even as a basis for developing systems. Sometimes, however, a design project may require functionality that isn't offered on an evaluation board or might need a smaller end system. In these cases, design teams must build a custom board.

Each Xilinx® evaluation board comes with a Xilinx Board Description (XBD) file that lists the peripherals, their configuration, control registers and pin locking with the FPGA on the board. XBD files are very useful in that they keep design teams organized and help them formulate the best strategy for current designs and even for future designs that they may implement on the board.

Of course, if you are creating a custom board you won't have an XBD file from Xilinx to fall back on. But it is certainly worthwhile to take the time to develop an XBD file of your own. A properly prepared XBD file helps your design team manage the project and also streamlines your device driver and firmware development. Luckily, with a bit of research and effort, you can build a custom XBD file for your board without undue difficulty. (For those using the Vivado Design Suite, Xilinx offers refined XBD capabilities in a new utility called Board Manager that Xilinx introduced with version 2014.1 of the tool suite. For more information, see the "Vivado Design Suite User Guide," *http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug898-vivado-embedded-design.pdf*.)

Let's look at one method for developing a custom XBD file. For the purposes of this example design, we'll be creating our XBD file for a custom board using a Virtex®-5 FX30T FPGA.

The best place to start is with the documentation from Xilinx and distributor Avnet. Because we'll have to write our own XBD file, we will have to do it in the XBD syntax. Xilinx has documented the XBD syntax in the Platform Specification Format Reference Manual (see *http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/psf_rm.pdf*).

It's very likely that your custom board will require serial communications (RS232 and RS422), analog-to-digital converters (ADCs), digital-to-analog converters (DACs), RAM and flash memory. The evaluation boards from Xilinx and Avnet will also have these peripherals, so finding a board that has similar parts and looking at its related XBD file will speed up the development of your custom XBD file.

Each XBD file will have different blocks that define the FPGA interfaces the board supports, and each block has a list of attributes, parameters and ports. Thus, the first entry in the file starts with global attribute commands, vendor information, the name of the board and its revision number, a Web URL for assistance, and both short and long descriptions of the board.

The local attribute command of the file is defined between a BEGIN-END block and expressed in the specific format that is available in the Platform Specification Reference Manual. For this example, we'll use the ISE® version 12.4 design tools targeting our Virtex-5FX30T FPGA, which includes a hard PowerPC® 440 core apart from its configurable logic cells.

Other than the FPGA, a custom board will contain different peripherals based on the needs of the design, such as a serial communication interface (RS232, RS422), ADC, DAC, SRAM and so on. You can fulfill multiple UART requirements by using specialized intellectual property (IP) blocks for serial communication. For example, you can use external memory controller (EMC) IP for interfacing SRAM with the FPGA, and general-purpose I/O (GPIO) IP to link the ADC and DAC with the FPGA.

For our example design, we prepared our custom XBD file to meet functional and device requirements as listed in the device datasheet. The input clock signal to the FPGA is 20 MHz. The processor is running at 200

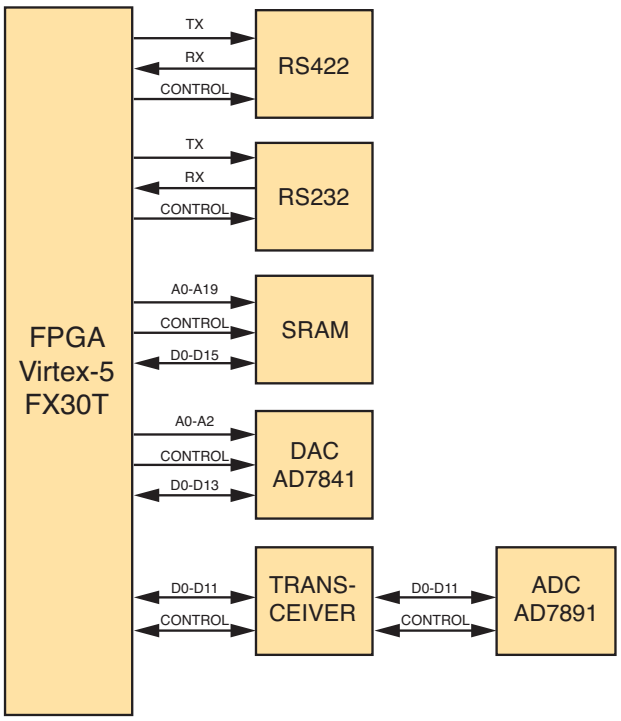


Figure 1 – Block diagram of custom hardware design

MHz and the Processor Local Bus (PLB) is operating at 100 MHz. Based on this information, we can be sure we have maintained local device driver timing. Figure 1 shows a block diagram of the custom hardware.

Let us begin with the custom file. It starts with the global attribute command and after that the clock signal, which is compulsory for all the boards, as follows:

```
ATTRIBUTE VENDOR = Xilinx Board FX30T
ATTRIBUTE NAME = Virtex 5 FX30T
ATTRIBUTE REVISION = A
ATTRIBUTE SPEC_URL = www.xilinx.com
ATTRIBUTE  CONTACT_INFO_URL  =  http://www.
xilinx.com/support/techsup/tappinfo.htm
ATTRIBUTE DESC = Xilinx Virtex 5 FX30T Cus-
tom Platform
ATTRIBUTE LONG_DESC = 'The FX30T board is
intended to showcase and demonstrate Vir-
tex-5 technology.  This board utilizes Xil-
inx Virtex 5 XC5VFX30T-FF665 device. The
board includes ADC, DAC, RS232, RS422, SRAM,
PLATFORM FLASH, CPU Debug (JTAG) and CPU
Trace connectors. '

BEGIN IO_INTERFACE
  ATTRIBUTE INSTANCE = clk_1
  ATTRIBUTE IOTYPE = XIL_CLOCK_V1
   PARAMETER CLK_FREQ = 20000000, IO_IS =
clk_freq, RANGE = (20000000) # 20 Mhz
   PORT USER_SYS_CLK = CLK_20MHZ, IO_IS =
ext_clk
END
```

After this we need to list all the peripherals of the board, one by one, in this file. (Readers interested in the detailed coding information for each peripheral block may refer to the companion PDF titled "Details of XBD Coding for a Custom Board," at *http://www.xilinx.com/publications/xcellonline/xbd_coding.pdf*.)

DIGITAL-TO-ANALOG CONVERTER

Let us begin with the digital-to-analog converter, the AD7841 from Analog Devices. This DAC has eight channels, three address lines and 14-bit data lines, along with a handful of control signals to handle the functioning of the device. It interfaces with the FPGA using a GPIO IP core provided by Xilinx. Address lines of the device (A0-A2) are connected to the processor address lines.

The DAC has four control signals: LDACN, CSN, WRN and CLRN. There are two ways to structure these signals—you could either assign one bit to each signal, or directly frame one register of 4 bits. It all depends on how you want to handle these signals for your application. This DAC is 14 bits, D0-D13.

Now that we have one detail of the first device, the DAC, listed in our XBD file, we need to turn our attention to writing firmware. In developing the device driver, we need to go through the datasheet and timing diagram very carefully. The timing diagram is shown in Figure 2.

The datasheet of the device explains that the user has to read the timing diagram and generate a control signal. Timing information t0-t11 is exactly followed as per the datasheet (AD7841) from Analog Devices. As a first step, set the direction of the signal to out and then pull it to high by writing 1 to corresponding addresses. For example, the direction LDACN signal is set to out and then pulled high with the following statement:

```
XGpio_WriteReg(XPAR_DAC_14BIT_CONTROL_
LDACN_BASEADDR,XGPIO_TRI_OFFSET,0x0);
//Direction is out

XGpio_WriteReg(XPAR_DAC_14BIT_CONTROL_
LDACN_BASEADDR,XGPIO_DATA_OFFSET,1);
//Pulled high
```

Delays between signals are achieved by using either the "for" loop or, alternatively, the "NOP" instruction. Equally important are the sequencing and setup-and-hold timings of each signal; you will find these specs in the device datasheet. Here, each increment corresponds to 5 nanoseconds when the processor is running at 200 MHz.

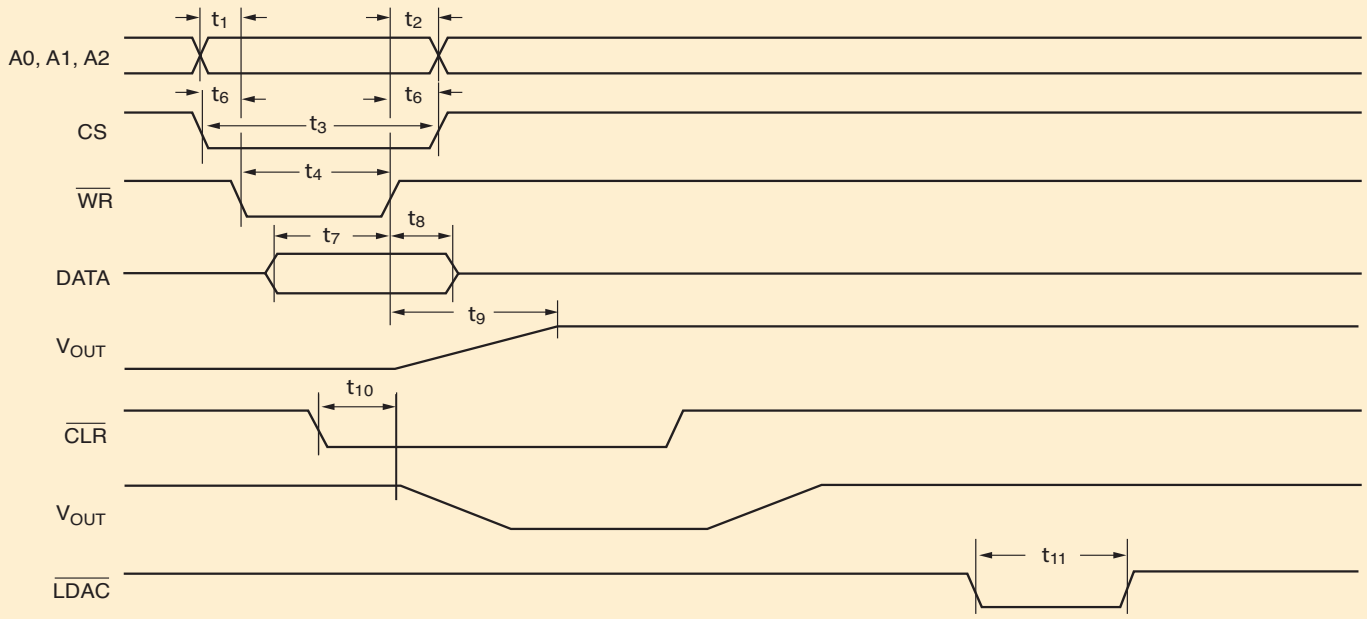


Figure 2 – Timing diagram of the AD7841 DAC

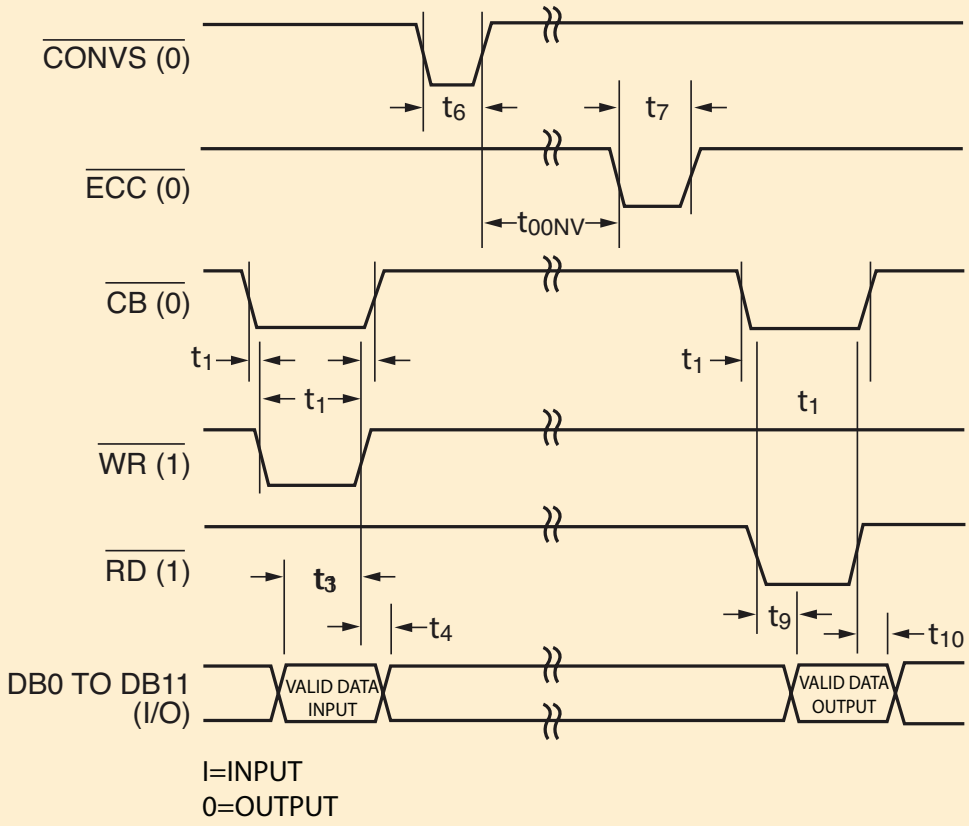


Figure 3 – Timing diagram of the AD7891 ADC

Thanks to the Xilinx EMC IP core and the XBD file, interfacing and controlling memory was an easy task. We found timing constraints in the device datasheet and listed them in the XBD file.

COMMUNICATION AND SRAM INTERFACES

We will use XIL UART IP from Xilinx for interfacing RS232 and RS422 driver ICs with our Virtex-5 FPGA. We chose two Maxim devices—the MAX3079 and MAX3237—for RS422 and RS232 communication, respectively. We generated the control signals of the RS422 IC using the GPIO IP core.

In terms of memory, we chose static RAM from Cypress, the CY7C1061BV33 device, interfacing it with the FPGA using the Xilinx External Memory Controller IP core (XIL_EMC). Thanks to this IP and the XBD file, interfacing and controlling memory was an easy task. We found timing constraints of the device in the device datasheet and listed them in the XBD file. As per the selected device, you need to update different parameters in your XBD file. Because the Xilinx IP is sufficient for handling this memory, a separate device driver was not required.

ADC INTERFACING

In this design, interfacing the ADC with the FPGA was a challenge, because all input analog signals—which are coming from the different sensors—are in the range of ±10 volts. To meet the functional requirements of our example board, we chose the AD7891-1 from Analog Devices. This ADC has eight channels and a 12-bit data bus with the option of serial and parallel interfaces. In this design, the parallel interface proved to be the better option.

This device operates on the 5-V input and since the FPGA I/O voltage is 3.3 V, our design needed a transceiver for interfacing with the FPGA. Subsequently, we will refer to this transceiver as a buffer. On one side, the buffer is connected with the FPGA and on the other side, it connects to the ADC. The FPGA handles the control signals of the buffer. You need to carefully handle the direction pin and output enable pin of the device to control the data flow from the FPGA to the ADC and vice versa. Control signals of the device are grouped as a 5-bit register.

End of conversion (EOCN) of the device should be listed separately in your custom XBD file, since this is an important signal. EOCN indicates that conversion is completed and new data is available to the FPGA for processing.

BUFFER INTERFACING

In this design, we chose a level-shifting transceiver, the SN74ALVC164245 from Texas Instruments, as a buffer. It is a 16-bit noninverting bus transceiver and has two separate ports as well as supply rails. Port B of the device is connected with the 5-V ADC and Port A with the 3.3-V FPGA. This configuration allows signal translation from 5 V to 3.3 V and vice versa.

With the direction pin set low, data at Port B is transferred to Port A. On high, the opposite will happen, and data from Port A will be transferred to Port B. To read the data from the ADC, set the output enable (OEN) and direction (DIR) pins to low. On a write operation, the FPGA will issue a command to the ADC, with the direction pin high and output enable pin low. In our example design, the output enable pins of the buffer are pulled high in order to avoid bus conflicts. We've used two buffers and two ADCs in this design.

Figure 3 is a timing diagram for parallel interface mode, which is what we used for generating control sig-

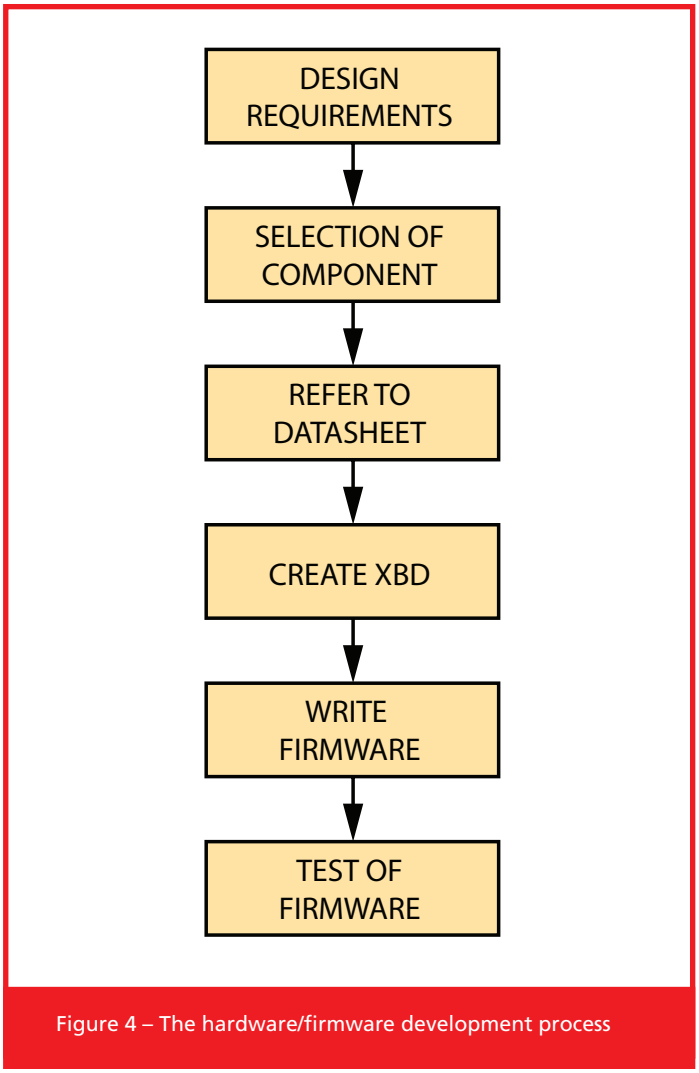


Figure 4 – The hardware/firmware development process

nals. Signal timings from t0 to t11 are referenced with the device datasheet from Analog Devices.

You can create or modify this file using any editing program, such as Notepad or WordPad. You must save the file with the extension ".xbd." Upon completion, the file has to be located in the particular path that will make it visible to the EDK tool directly, while creating a new project. When you start a new project, you need to select this file for the custom board.

For example, to make a hypothetical board called "customboard" visible to the EDK tools, you must follow a specific directory structure. This is a very important step. Our example "customboard_RevX_vX_X_0.xbd," once created, has to be stored in the following directory structure:

Example Path:

```
C:\Xilinx\12.4\ISE_DS\EDK\board\customboard_
RevX\data\custom\ customboard_RevX_vX_X_0.xbd
```

In addition, you need to follow certain steps in the development of hardware to firmware for any application, as seen in Figure 4.

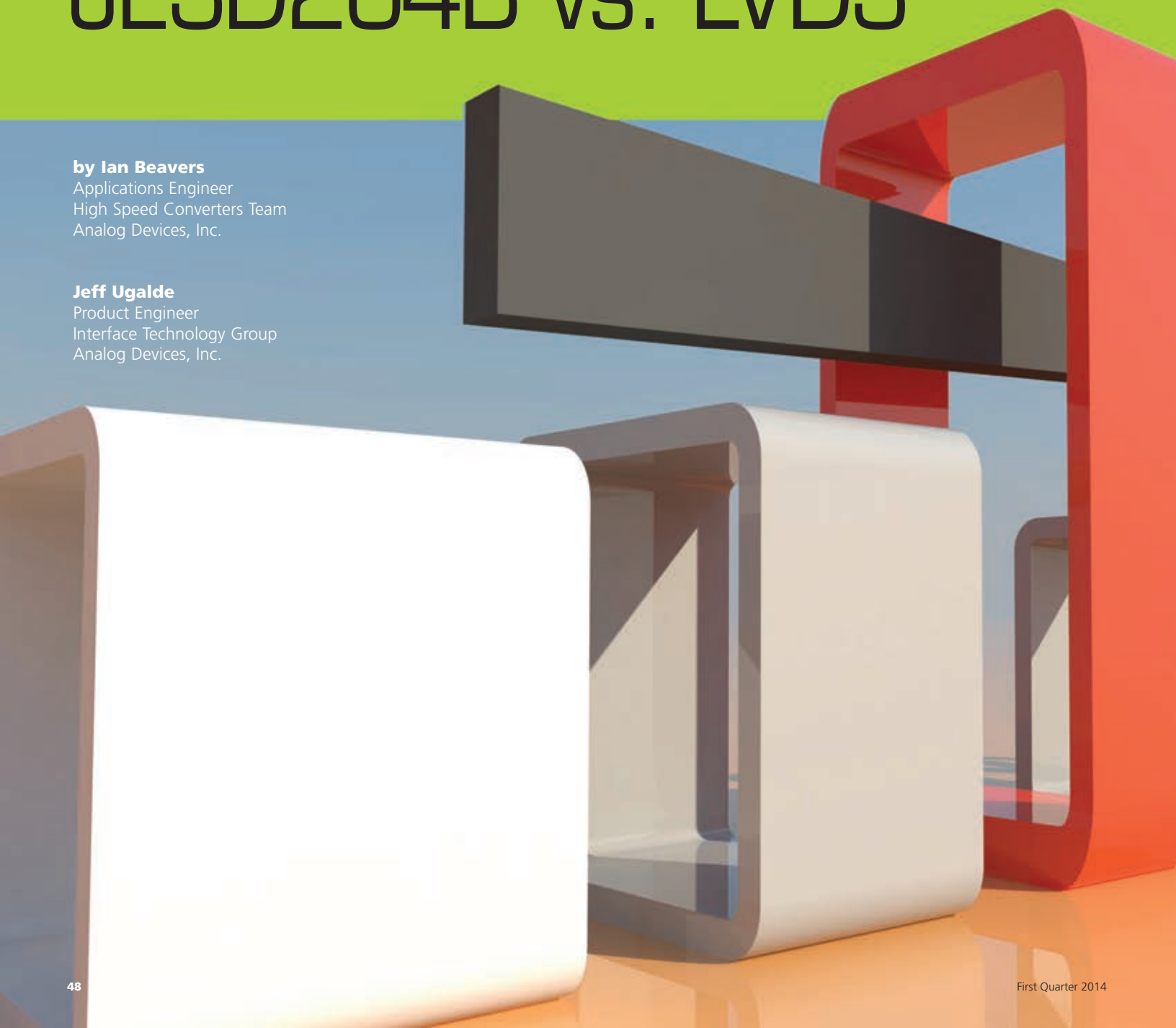
FASTER DEVELOPMENT TIME

It is always a challenging task for an embedded designer making custom hardware to develop device drivers for his or her own particular application. Xilinx provides extensive literature on the developmental process and a generic board-description file for the development. You can easily tailor this file and modify it to meet the needs of your own custom hardware. For any modification or change in hardware configuration, you will then need to look at only one single file and incorporate all the device-related changes into it. In this way, the custom XBD file increases productivity and accelerates development time.

Selecting the Right Converter: JESD204B vs. LVDS

by Ian Beavers
Applications Engineer
High Speed Converters Team
Analog Devices, Inc.

Jeff Ugalde
Product Engineer
Interface Technology Group
Analog Devices, Inc.



Converters built to the new JESD204B standard play well with modern high-speed FPGAs. I/O considerations are a factor when designing with these devices.

As data converter architectures and FPGAs continue to use more advanced and smaller geometries, new data interface challenges present themselves for system designers. Smaller process geometries allow higher-bandwidth converters to operate at increasing resolution and speed, which drives a higher data throughput. However, they also make available higher serialization/deserialization (serdes) rates to accommodate a bandwidth consumption that was unavailable at larger geometries. The smaller area also makes it possible to integrate more data converters into a single device. The interface solution for these data converters needs to allow high data rates, be compatible with complex FPGA devices and maintain a nominal I/O count.

The JESD204B interface, a serdes link specification for converters, permits data transfer at a maximum of 12.5 Gbps. This maximum data rate is possible for converters using advanced processing nodes, such as those at 65 nanometers and smaller, with increasing power efficiency. The 12.5-Gbps channel rate helps system designers take full advantage of this new serdes technology over the alternative low-voltage differential signaling (LVDS) DDR interface.

Several open-market FPGA models offer serial transceivers with data rates up to and beyond 12.5 Gbps, including the Xilinx® Virtex®-7 and Kintex®-7 families. While FPGAs have had this capability for some time, converters are now capable of this same serdes performance. This allows synchronization processing of several converters, possibly with multiple internal channels, within a single FPGA device.

DIFFERENT STROKES FOR DIFFERENT FOLKS

When it comes to high-speed serial transmission on data converters, there are different choices for different applications. Data converter manufacturers have chosen LVDS as the main differential signaling technology for more than a decade. Although some applications of LVDS can use a higher data rate, currently the maximum LVDS data rate available from converter vendors in the market is 0.8 to 1 Gbps. LVDS technology has not been keeping up with the bandwidth demand from converters. LVDS is controlled by the TIA/EIA 644A specification, which is an industry standard for LVDS core manufacturers. The specification acts as a best-practice guide for designers, promoting compatibility of LVDS transmitters and receivers from different manufacturers. In the same manner, designers who do not completely adhere to the LVDS specification build products that are not compliant and have a bigger challenge with compatibility in the market.

Like LVDS, JESD204B exists under the umbrella of a standards organization—Jedec—that provides guidance in the electrical and physical requirements for interoperability among different manufacturers. The JESD204B maximum data rate is defined at 12.5 Gbps, allowing a throughput advantage of more than 10x compared with practical LVDS. This performance decreases the I/O requirements and package size for data converter systems, and provides significant system cost savings with lower static power.

The JESD204B specification allows for AC coupling, which permits compatibility with different technology nodes that use different supply levels. For example, FPGA processing nodes at 28 nm and smaller typically are at the forefront of fabrication technology. Converter transistor nodes often lag behind state-of-the-art FPGAs by a few generations due to the need for custom analog designs. Conversely, LVDS usually uses a DC coupling strategy, which can make interfacing a converter to an FPGA with a lower-power supply more difficult. The larger the mismatches in common-mode voltage, the higher the static-current draw, which will be independent of the data rate. For these reasons, JESD204B is becoming a very

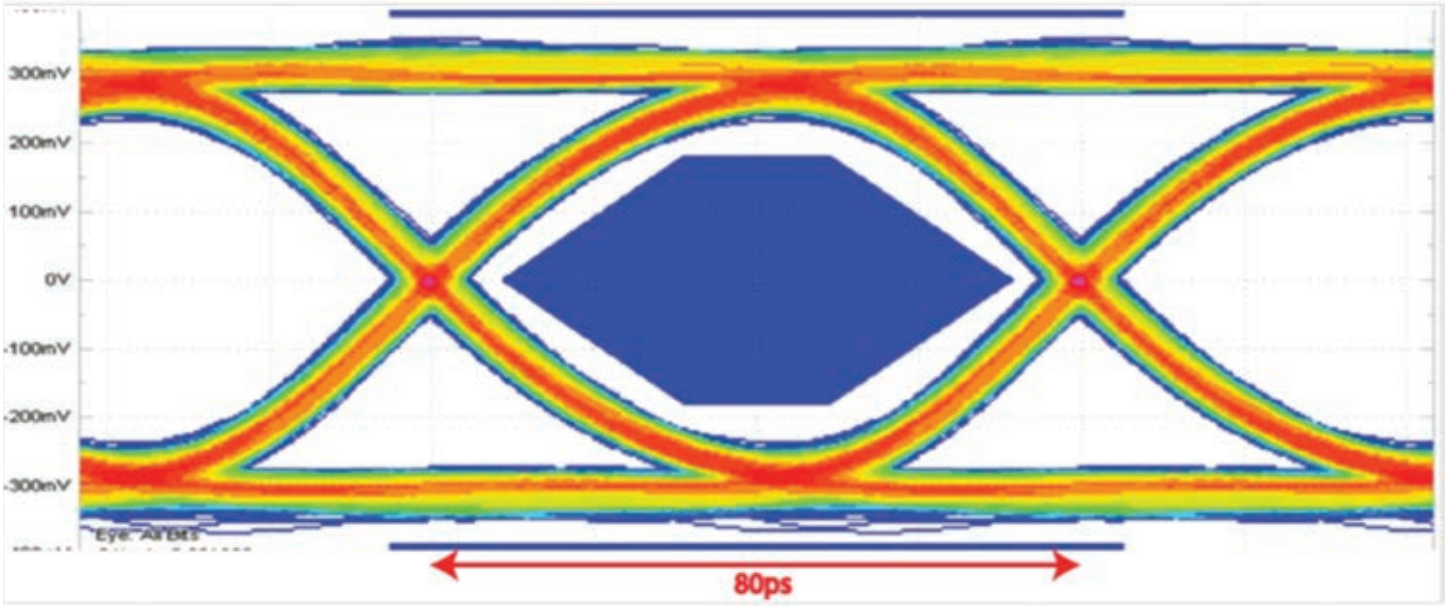


Figure 1 – A 12.5-Gbps JESD204B eye diagram vs. LV-OIF-11G-SR transmit mask

attractive differential-signaling technology for manufacturers of high-resolution and high-speed data converters.

In addition to the electrical specifications, JESD204B has requirements regarding eye diagram performance for three physical-layer variants. The performance metrics include a defined data eye to a mask and total jitter budget. The Optical Internetworking Forum (OIF) has well-established physical-layer (PHY) specifications and eye mask criteria that the JESD204B interface leverages for the same serial data rates. The JESD204B link makes use of the total jitter maximum figure allowed by the OIF's low-voltage 11-Gbit short-reach specification (LV-OIF-11G-SR), which is 30 percent of a unit interval (UI). Figure 1 shows a pristine JESD204B eye diagram at 12.5 Gbps with the eye mask. The eye mask allows a deterministic amount of margin available on both the vertical and horizontal axes. It is important to note that the 12.5-Gbps eye mask meets the specifications from LV-OIF-11G-SR, which is based upon a speed of 11.1 Gbps, a tighter timing requirement than otherwise would be applied at 12.5 Gbps.

THREE PHY VARIANTS

JESD204B supports three physical-layer variants for serial data transmission, defined by the LV-OIF specification and categorized depending on the maximum JESD204B lane rate. The rates that define the categories of the three physical-layer variants are 3.125 Gbps, 6.375 Gbps and 12.5 Gbps, as shown below.

- LV-OIF-SxI5-based operation: from 312.5 Mbps to 3.125 Gbps
- LV-OIF-6G-SR-based operation: from 312.5 Mbps to 6.375 Gbps
- LV-OIF-11G-SR-based operation: from 312.5 Mbps to 12.5 Gbps

Each category has slightly different minimum and maximum electrical specifications accommodating differences needed due to the wide range of data rates supported. Figure 2 shows the electrical specification parameters for the LV-OIF-11G-SR physical-layer variant, which is used for the maximum JESD204B data rate of 12.5 Gbps.

One advantage of having a number of specification variants is that it's possible to support a much wider common-mode voltage range on the link compared with the DC-coupled case. This relaxes the system design requirement on the JESD204B transmitter and receiver, which may come from different vendors, as it provides level shifting as needed. A second advantage to AC-coupled data lanes is that common-mode noise is decoupled between the transmitter and receiver, thus helping ease system designers' concerns about signal quality. DC coupling is more susceptible to common-mode noise coupling into the data lines. A third advantage of AC coupling is that it relaxes termination voltage requirements of different transmitters (V_{tt}) and receivers from multiple vendors, thus allowing the receiver to operate at its optimal common-mode voltage. This allows JESD204B transmitters and receivers to operate under different termination voltages in system designs needing the flexibility of different power supply voltages.

The JESD204B interface can also partition data for many converters onto a single link. As link rates increase to 12.5 Gbps, more converters can fit onto the same link (all other variables being constant; see Figure 3). This is particularly useful for devices with two, four, eight and 16 converters in a single

JESD204B offers a clear specification for sending combined data from multiple converters serially and transmitting across the same pins.

JESD204B Data Outputs				
Parameter	Test Conditions/ Comments	Min	Max	Unit
Data Rate per Channel (NRZ)		1	12.5	Gbps
Unit Interval (UI)		80	1000	ps
Rise Time (t_R) (20%/80%; 100Ω load)		24		ps
Fall Time (t_F) (20%/80%; 100Ω load)		24		ps
Output Common Mode (T_CM)	AC Coupled	0	1.8	V
	V_{tt} = 1.2V	735	1135	mV
	V_{tt} = 1.0V	550	1060	mV
	V_{tt} = 0.8V	490	850	mV
Differential Voltage (V_{diff})		360	770	mV
Differential Impedance		80	120	Ohms
Differential Return Loss (RL_{ddif})		8		dB
Common Mode Return Loss (RL_{dcm})		6		dB

Figure 2 – Electrical specifications for an LV-OIF-11G-SR JESD204B, 12.5-Gbps transmitter show the flexibility of common-mode voltage termination on the link

Converter Channels	Sample Rate (MSPS)	Resolution Max (bits)	LVDS Pins (@ 1Gbps)	JESD204B Pins (@12.5Gbps)
16	78.125	16	40	4
8	156.25	16	40	4
4	312.5	16	40	4
2	625	16	40	4
8	78.125	16	20	2
4	156.25	16	20	2
2	312.5	16	20	2
1	625	16	20	2
1	1,250	16	40	4
1	2,500	16	80	8
1	5,000	16	160	16
1	10,000	16	320	32

Figure 3 – A comparison of converters with different sample rates and channels shows the difference in I/O count. The JESD204B interface operating at 12.5 Gbps consumes 1/10 the number of pins compared with LVDS operating at 1 Gbps

package, and a distinct advantage over an LVDS interface. LVDS can support direct I/O to and from a single converter as an I/O structure, but does not explicitly define a method to combine data from multiple converters across the I/O. With JESD204B, there is a clear specification for sending combined data from multiple converters serially and transmitting across the same pins. The source of each piece of device data does not even need to be an actual fixed hardware converter. It could be from a "virtual converter" filter, where the output data is split into two for a real and complex path, as one of several digitally processed outputs from a single hardware converter. Communication systems that use real and imaginary data (I&Q) for 90-degree phase sampling can also take advantage of virtual converters to send data for multiple converters on a JESD204B link.

RIGHT CONVERTER FOR THE SYSTEM

The insatiable bandwidth demands for higher-speed converters are driving designs to more advanced CMOS process nodes for lower power and increased performance. This trend brings with it new interface challenges. The 12.5-Gbps maximum speed JESD204B interface helps solve some of these challenges that otherwise would require an increasing number of LVDS DDR lanes that do not keep pace with bandwidth speed and capabilities at higher sample rates. Keeping in mind the pin I/O, coupling and supply domain requirements for the converter's digital interface will help you select the right converter for the system.

How to Bring an SMC-Generated Peripheral with AXI4-Lite Interface into the Xilinx Environment

by Sheetal Jain
Senior R&D Engineer
Synopsys
skj@synopsys.com

The SMC Host Interface Block makes it simple to integrate a design you've created with the Synphony Model Compiler into a Xilinx embedded platform.

Synphony Model Compiler (SMC) is a model-based tool from Synopsys that synthesizes designs created in Simulink® and MATLAB® to generate optimized RTL for ASIC and FPGA targets. SMC includes a comprehensive high-level model library for creating math, signal-processing and communications designs in the Simulink environment. This library simplifies the capture of fixed- and floating-point single-rate or multirate algorithms and functionality debug in a high-level model-based design environment. Using these verified models, the SMC RTL Generation Engine automatically creates RTL for hardware implementation and rapid exploration of multiple architectures for area, performance, power and throughput trade-offs. SMC's high-level synthesis engine takes in the top-level design as well as MATLAB language input to generate RTL optimized for the chosen hardware target. Additionally, SMC automatically generates an RTL testbench for the design, along with a bit- and cycle-accurate C model and SystemC wrapper to enable validation of the generated hardware in a SystemC simulation environment.

In many applications, designers create a peripheral to perform some signal-processing function and must configure the peripheral via a host processor such as the Xilinx® MicroBlaze™ soft processor core. The host processor typically connects to the peripheral using a standard bus interface such as AMBA® AXI4 or AXI4-Lite. The SMC library includes a Host Interface Block that implements a slave interface to the host processor. This Host Interface Block supports the AXI4-Lite, APB, Generic Interface and Avalon-MM bus interface protocol standards. The Host Interface Block also implements the necessary memory-map registers to configure the SMC design, including FIR filter coefficients, frequency and phase settings of a numerically controlled oscillator (NCO) and FFT length of the variable-length FFT block. The Host Interface Block can implement these memory-mapped registers at any desired sample rate, including asynchronous, to the bus interface clock. You can specify the bus interface and the memory map settings in the Host Interface Block's UI. Designers can use the Host Interface Block to connect an SMC design to a bus interconnect or to a bus master.

Let's take a closer look at how to import and integrate a peripheral designed with the SMC Host Interface Block into a Xilinx Embedded Development Kit (EDK) project. We will also

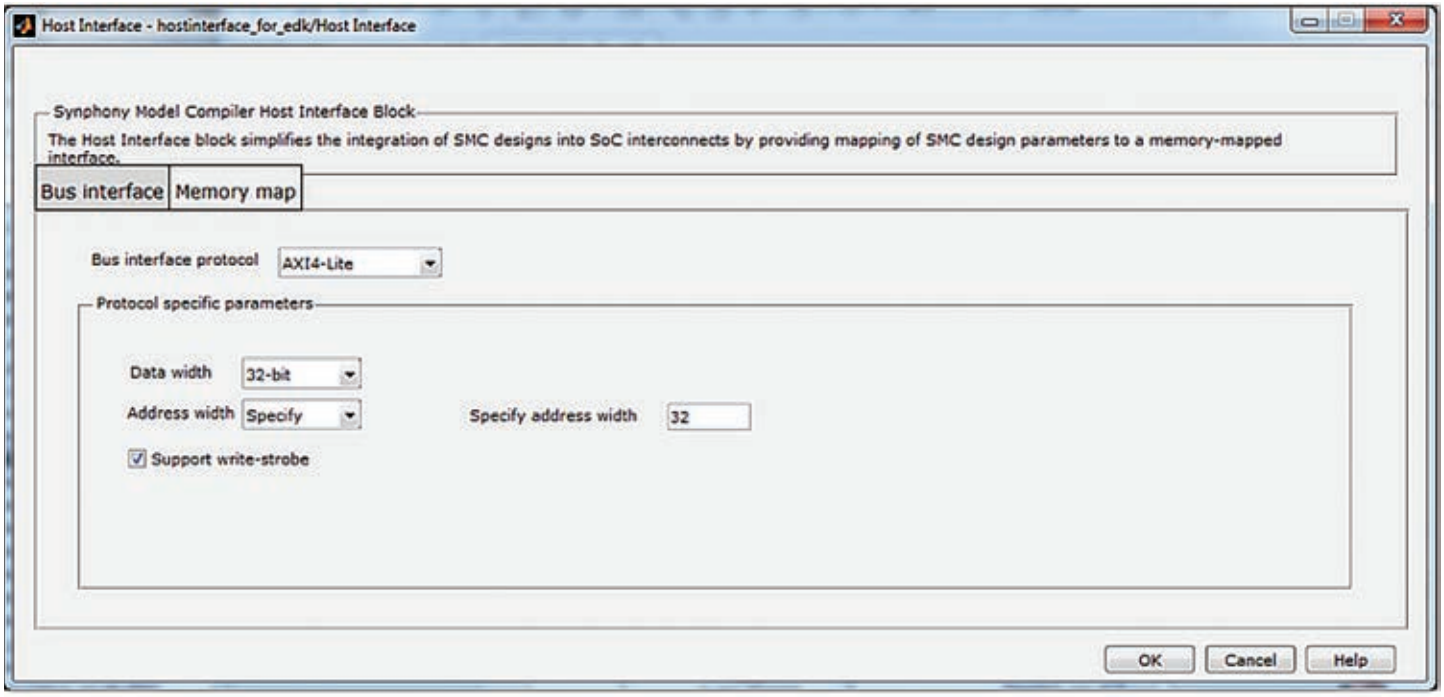


Figure 1 – SMC bus interface protocol specification

examine how to simulate the AXI4-Lite transactions from a MicroBlaze host processor connected to the peripheral via a standard bus interconnect. There are four major steps to this process:

1. Create the peripheral in Simulink using the IP and the Host Interface Block and SMC RTL Generation Engine to generate the optimized RTL implementation for the design.
2. Import the peripheral into the Xilinx EDK project and integrate it with the rest of the design.
3. Develop the software application in the SDK.
4. Generate the RTL and simulate it to check for functional correctness of the hardware and software.

STEP 1: CREATE THE PERIPHERAL USING THE SMC LIBRARY

The first thing you will do is to create the algorithmic implementation of the peripheral using the SMC library blocks, and verify the functionality. Next, configure the SMC Host Interface Block based on two factors: the configuration data of the algorithmic part, which defines the memory-map parameters; and the system's interconnect bus protocol, which defines the bus interface parameters. Then connect the Host Interface Block to the algorithmic part of the peripheral. Some of the parameters of the Host Interface Block (e.g. the bus interconnect, address width and base address) will depend on the platform you are targeting. For our example, we have chosen a Xilinx Virtex®-7 FPGA as the platform and AXI4-Lite as the bus interface. This platform imposes some restrictions on the address width, base address and address space for each peripheral. The address width must be 32 bits, while the base address has to be a multiple of 4 kbytes and the minimum address space available is 4 kbytes. Figures 1 and 2 show how the bus interface protocol and memory map are configured using the Host Interface Block.

For ease of integration, though not mandatory, it is highly recommended that the bus interface ports in the SMC model follow the naming conventions required by the Xilinx EDK. Append "S_AXI_" to the standard AXI4-Lite interface signal names. For example, the address signal of the AXI address write channel (AWADDR) should be named S_AXI_AWADDR. If the signals do not follow the AXI4-Lite naming convention, another opportunity exists to map the port name to the AXI4-Lite signal name while importing the peripheral into the Xilinx EDK. Additionally, do not use capital letters in the Simulink model name because the EDK does not support peripherals with capital letters in their names.

Once you've added, configured and connected the Host Interface Block, generate the RTL for the peripheral using SMC's RTL Generation Engine. Specify the target device, implementation parameters and optimization constraints in the SMC UI to drive the RTL Generation Engine to produce optimized hardware for your target device. In the top RTL that SMC generated, add two virtual parameters (generics in case your top RTL is VHDL) named "C_BASEADDR" and "C_HIGHADDR." Assign their default value to the base address of your memo-

For ease of integration, though not mandatory, it is highly recommended that the bus interface ports in the SMC model follow the naming conventions required by the Xilinx EDK.

ry-mapped space and the max address of the memory-mapped space as required by your IP. This step is necessary to ensure that the EDK identifies your peripheral's memory-mapped address space. An example of the top-level Verilog RTL for the SMC-generated design is shown below, with the two parameters that you need to add highlighted.

```
module host_inferface_for_edk_top
(__list_of_ports_will_be_available_
here__
  );

parameter C_BASEADDR =
32'h41418000;

parameter C_HIGHADDR =
32'h41418fff;
```

STEP 2: IMPORT YOUR PERIPHERAL INTO XILINX EDK AND INTEGRATE IT

The next step is to import the peripheral hardware into the EDK's Xilinx Platform Studio (XPS) and make the necessary connections (bus interface ports as well as functional ports) in the system. In our example, we have created a basic system with the MicroBlaze processor, Block RAM (BRAM) for storing the software executable, a Local Memory Bus (LMB), the AXI4-Lite interconnect and the MicroBlaze debug module.

Select the "Create or Import Peripheral" option in the Hardware category of the XPS GUI. This will open the Create and Import Peripheral wizard. Select the "Import existing peripheral" option in the wizard. Then, specify the path where you want the peripheral to be stored, the design name and the file type (HDL). Now add all the SMC-generated RTL files. Upon successful compilation of the RTL, you will need to identify the bus interface the peripheral supports—namely, the AXI4-Lite slave interface, as shown in Figure 3.

On the next screen in the wizard, select the AXI4-Lite ports of the peripheral and map them to the standard AXI4-Lite ports so that the EDK can connect the bus interface. If the names of the bus interface ports defined in the SMC model match the standard bus port names, the EDK will automatically map the ports (see Figure 4).

You may override the automatic mapping if the port names do not match, as demonstrated for the AXI4-Lite clock (Clk_Div3) and reset (GlobalReset) signals.

Next, specify the Register Space base and high address as the C_BASEADDR and C_HIGHADDR parameters insert-

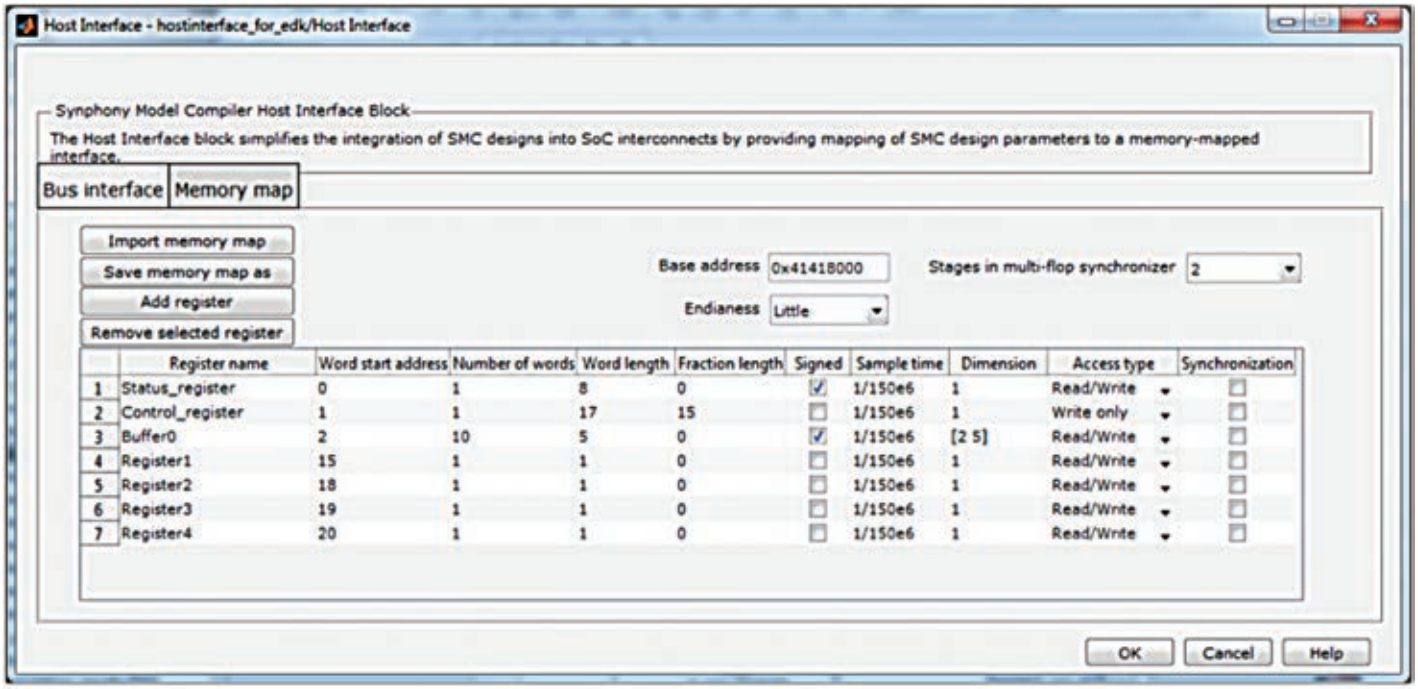


Figure 2 – SMC Host Interface Block memory-map parameter settings

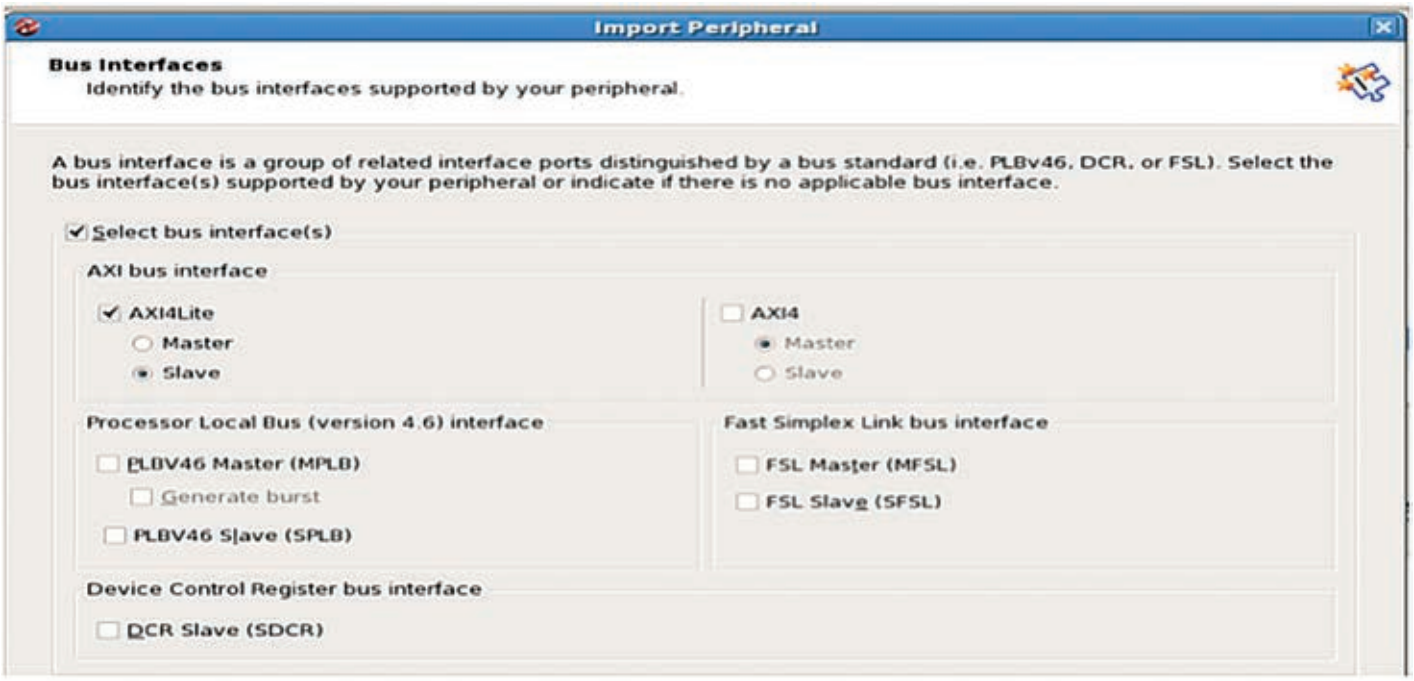


Figure 3 – Specifying the bus interface the peripheral supports—here, AXI4-Lite

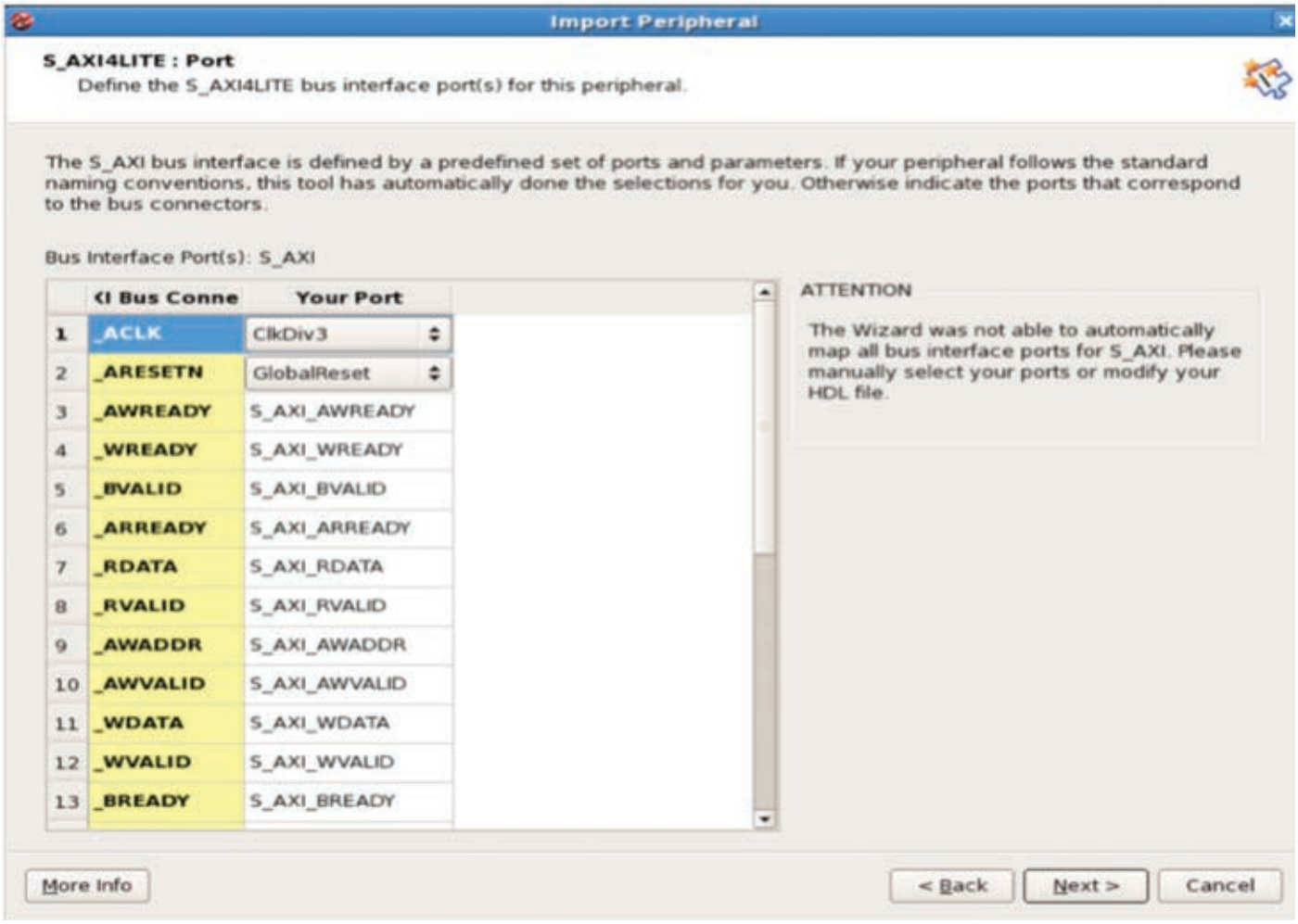


Figure 4 – Mapping the RTL ports to the appropriate AXI4-Lite bus interface signal

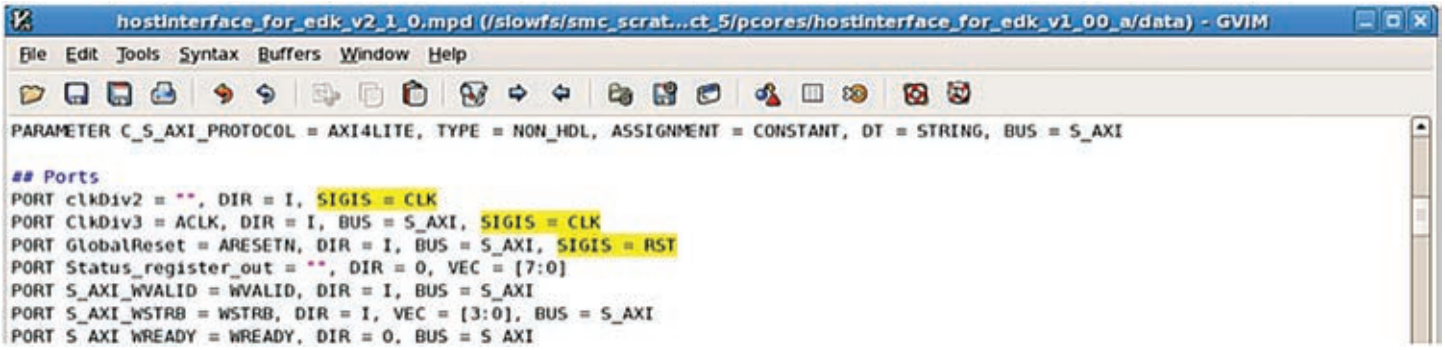


Figure 5 – The *.mpd file with signal types specified for clock and reset ports

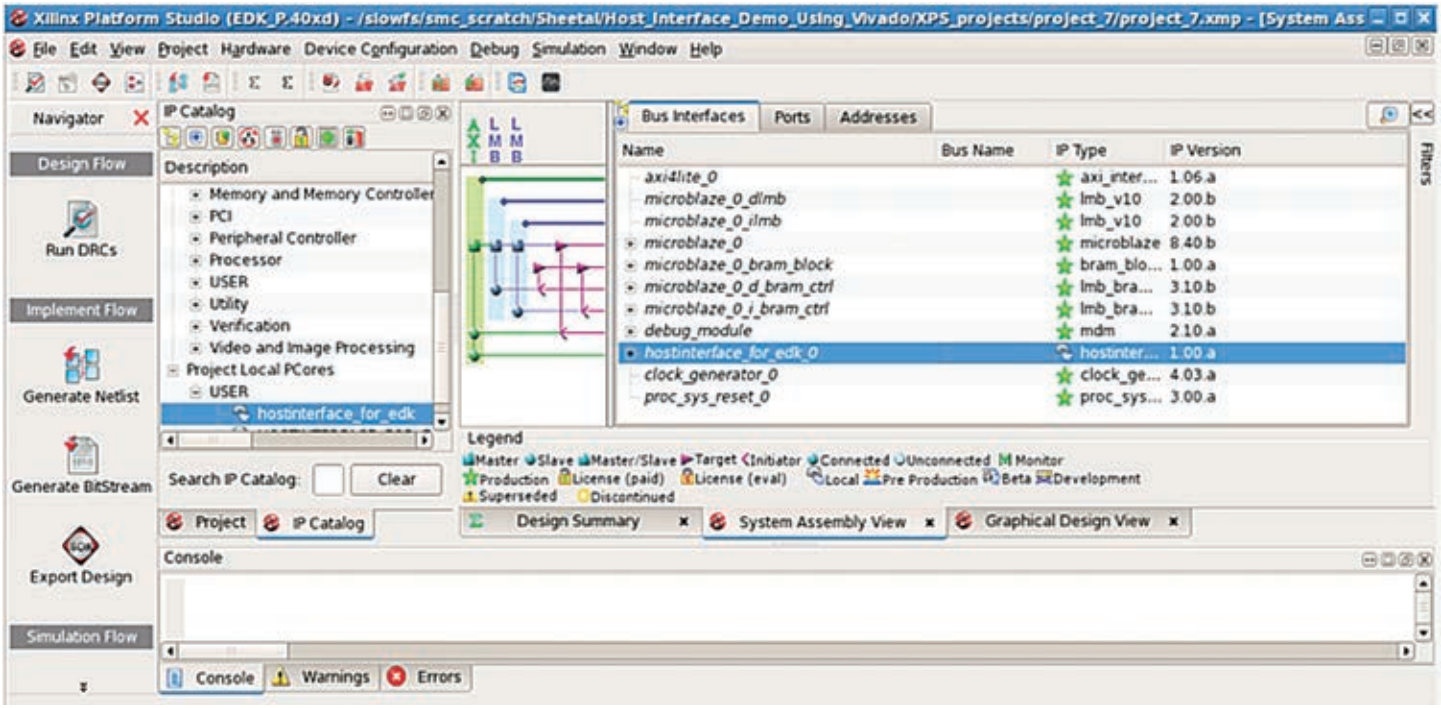


Figure 6 – SMC-generated design connected to the AXI4-Lite bus and MicroBlaze processor

ed into the RTL in step 1. Uncheck the memory space option, because the Host Interface Block has an addressable configuration register space. But keep the default attributes of the RTL parameters unchanged on the next screen to ensure a match with the parameters specified in the Host Interface Block.

The next screen is titled "Port Attributes." Here, you must specify the attributes of clocks or resets on any additional clocks or resets in the design. Click "Finish" on the next screen to add the peripheral to the XPS project. The SMC peripheral has now been successfully imported into XPS. You can verify that this has happened by checking the <project_working_directory>/pcores folder (XPS creates a directory with the name of the peripheral here). Browse through this directory to check that your RTL files are correctly imported.

XPS will also create a directory called "data" in parallel with the HDL directory. This data directory includes the microprocessor peripheral description (*.mpd) file, and information about the peripheral's parameters and its port. Check that the SIGIS = CLK and SIGIS = RST parameters are defined on the clock and reset port. If they are not defined, edit the *.mpd file and add the definitions manually. Figure 5 shows an example of this file with these parameters added.

You will now see your peripheral in the "USER" subcategory under Project Local PCores in the IP Catalog section of the XPS GUI. Right-click on the peripheral name and select "ADD IP." An XPS Core Config window will open. Do not edit any of the parameters in this window—leave the default settings unchanged to comply with the Xilinx EDK flow. If EDK does not accept the specified address space, this indicates a conflict in the specified memory map with some other peripheral in the design. You must go back to SMC, regenerate the RTL with a new base address value and repeat the above steps to import the SMC peripheral. Click "OK" in the Core Config window once

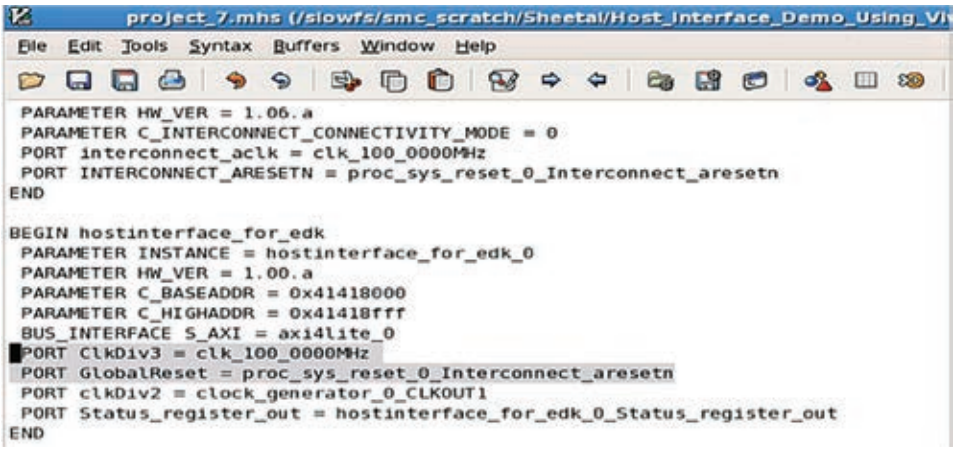


Figure 7 – Clock and reset port connections inserted in the *.mhs file

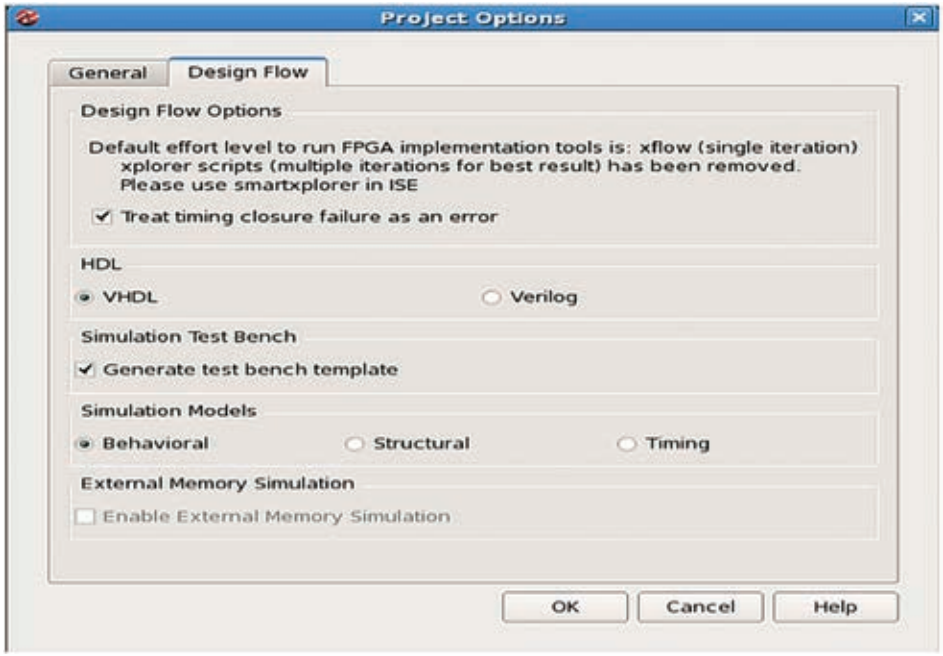


Figure 8 – Project setup to generate the testbench template and behavioral simulation model

the correct base and high address values are available.

XPS will open the "Instantiate and Connect IP" GUI. You can direct the tool to automatically link the peripheral to the interconnect bus driven by an available processor, or you may choose to manually connect the peripheral. Once the connection is complete, you will see the interface connection shown in Figure 6.

Cross-check that the AXI4-Lite related clock and reset are connected in the bus interface connection in the Graphical Design View tab. If they are not automatically connected, edit the microprocessor hardware specification file <project_name>.mhs in the <project_working_directory>. Figure 7 shows the *.mhs file with the clock and reset ports added.

Next, connect the non-AXI4-Lite ports of the design using the Ports tab of the System Assembly View window. In the Addresses tab of the System Assembly View, ensure that the peripheral address space is visible and the address range is locked.

Now, export the hardware to the Xilinx Software Development Kit by selecting the "Export hardware design to SDK" option in the Project category of the XPS GUI. You need not generate the bitstream if you just want to run RTL simulation. After completion, XPS will create an *.xml file that describes the hardware to the SDK. This file is normally created in the <project_ working_directory>/SDK/SDK_Export/hw folder.

STEP 3: DEVELOP SOFTWARE DRIVER USING XILINX SDK

The next step in the integration is to develop the software driver using the Xilinx SDK. Launch SDK and create a hardware platform specification project to source the *.xml file. If you have selected the "Export and Launch" option in EDK, the project is created automatically and the IP blocks and address map information in XPS is now available in the SDK project.

Before creating the board support package (BSP), you must create driver files for your peripheral. A typical driver header file must define the memory-mapped register offset address and the prototype to read and write to those registers.

Copy the driver files to the SDK project repository to identity the driver and then create the BSP project. Open a new, blank application project to create software to read and write data from the peripheral. In this project, specify the hardware target platform that was created as the first task in this step and the BSP that you have just created. In the BSP settings window, see the peripheral driver core for the design.

The application project includes a main.cc file where the software code is written for the application. Once this file is created, the SDK automatically compiles the code and creates an *.elf file that is used to simulate the software code in the RTL simulation environment.

The application project includes a main.cc file where the software code for the application is written. A simple example is a program that writes values into the first two registers of the memory map (the Status_register and the Control_register). Once this file is created, the SDF automatically compiles the code and creates an *.elf file that is used to simulate the software code in the RTL simulation envi-

It's a simple matter to use the Host Interface Block to integrate an SMC-generated peripheral with the Xilinx embedded platform using an AXI4-Lite slave interface.

ronment. This *.elf file can then be used to verify functionality, as shown in Step 4.

STEP 4: GENERATE THE RTL FILES IN XPS

The final step is to generate the RTL in XPS and simulate it to verify that the hardware and software are functionally correct. In the XPS GUI, choose the "Select Elf file" option in the Project category and then "Choose Simulation Elf file" to specify the path of the Elf file that the SDK has created. This file is available in the application project folder. To create the testbench template and behavioral simulation model (see Figure 8), you will choose the Design Flow tab settings of the Project Options in the Project category in the GUI.

Next, create the HDL files by selecting "Generate HDL Files" and launch the ISE® (ISim) HDL simulator to check functional correctness. Figure 9 shows the AXI4-Lite transaction initiated by the MicroBlaze processor for the software C code in the SDK project. Note that the first two registers of the peripheral (Status_register and Control_register) change their value as expected. The corresponding AXI4-Lite signals at the interface of the peripheral show that the SMC-created peripheral is successfully integrated in the embedded project.

POWERFUL TOOL SET

It's a simple matter to use the Host Interface Block to integrate an SMC-generated peripheral with the Xilinx embedded platform using an AXI4-Lite slave interface. The combination of SMC and the Xilinx embedded platform makes a powerful tool set to design and develop DSP peripherals integrated with a host processor. The Host Interface Block in SMC provides the interface necessary to complete the integration seamlessly and create powerful solutions for embedded platforms.

To learn more about SMC, visit *http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/Symphony-Model-Compiler.aspx?cmp=fpga-xcell-86-smc*

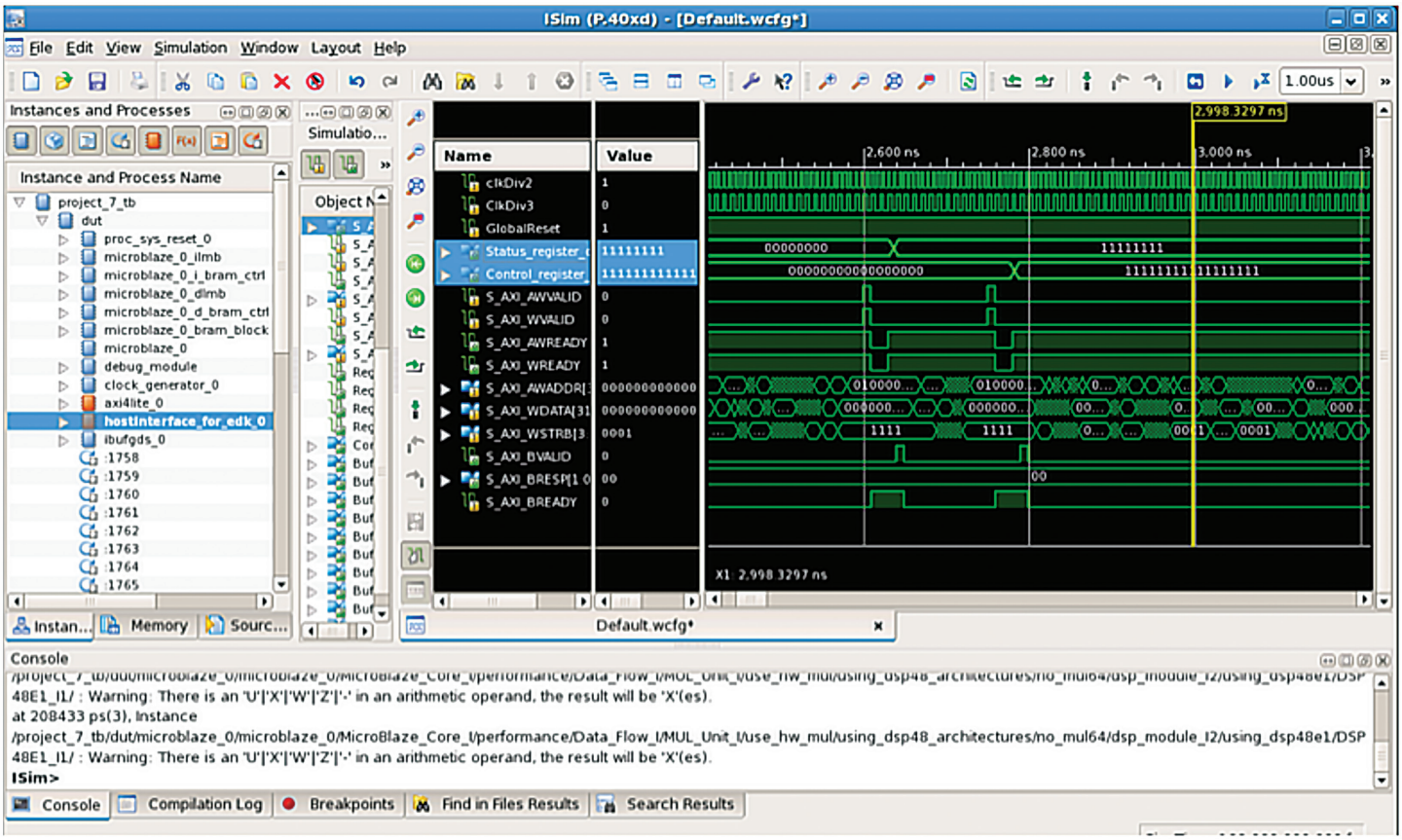


Figure 9 – AXI4-Lite transaction initiated by the MicroBlaze processor

What's New in the Vivado 2013.4 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 2013.4 is available from the Xilinx Download Center at www.xilinx.com/download.

VIVADO DESIGN SUITE 2013.4 RELEASE HIGHLIGHTS

The Vivado Design Suite 2013.4 features support for UltraScale™ devices as well as significant enhancements to IP Integrator, Vivado HLS, Vivado synthesis and the Incremental Design Flow.

Device Support

The following devices are production ready:

- Artix®-7 XC7A35T and XC7A50T FPGAs
- Zynq®-7000 XC7Z015 All Programmable SoC

Tandem Configuration for Xilinx PCIe® IP

The following devices have moved to production status:

- Kintex®-7 7K410T FPGA
- Virtex®-7 X550T FPGA

VIVADO DESIGN SUITE: DESIGN EDITION UPDATES

Vivado IP Integrator

The Vivado Design Suite IP Integrator supports more than 50 new pieces of IP, including:

- Connectivity IP
- CPRI™ and JESD204
- GMII to RGMII
- Virtex®-7 PCIe (Gen2 and Gen3)
- RXAUI and XAUI
- 10Gig Ethernet MAC and PCS PMA
- SelectIO™ Wizard

- An entire Block Design (BD) can be set as an "out-of-context module" to reduce synthesis times on unchanged blocks when doing design iterations.

- User IP can now be repackaged after it has been added to a diagram and all instances of the IP used in that project are updated to reflect the changes.

- Support has been added for "remote sources." Users need to create a temporary project to build the initial BD in a remote location.

- IP Integrator now supports a "non-project flow" using "read_bd."

- New designer assistance has been added around AXI slaves, Block RAM controllers, Zynq All Programmable SoC board presets and AXI-Ethernet.

- IP Integrator now supports address widths between 32 and 64 bits. This is useful for designing multiported memory controllers in IP Integrator.

- CTRL-F can now be used to find an IP or object on the IP Integrator canvas.

- The new "Make Connection" option simultaneously connects multiple objects.

- Users can customize AXI4 interface colors in a diagram based on AXI4 interface types. The default is still to have all interfaces displayed as the same color.

Vivado Synthesis

- Several quality-of-results improvements for DSP, including multiply-accumulate functions, can leverage dynamic opmode and fully map onto a single DSP block.

- Wide multipliers using more than one DSP block are improved through better allocation of pipeline registers.

- FIR filter inference results in push-button QoR (see, for example, the 741-MHz filter described in UG479).

Vivado Implementation Tools

The Incremental Compile flow silently ignores the Pblock constraints when they conflict with reused placement and honoring the Pblock constraints would result in worse timing performance. Better control over Pblock behavior in the Incremental Compile flow will be addressed in a future release. Additional Incremental Compile flow changes include:

- An automatic incremental reuse report is issued after read_checkpoint -incremental.

- A new incremental reuse report section lists conflicts between reused placement and physical constraints in the current design.

VIVADO DESIGN SUITE: SYSTEM EDITION UPDATES

Vivado High-Level Synthesis

Vivado Design Suite 2013.4 HLS updates include:

- Smoother integration of HLS designs into AXI4 systems is provided through new data-packing options that automate the alignment of data to 8-bit boundaries.

- Enhanced functionality is provided for AXI4 master interfaces as the user ports can now be optionally included in the interface.

- Improved resource usage is provided for designs using division operations. These operations now automatically benefit from smaller implementations.

System Generator for DSP

System integration of System Generator for DSP blocks is now faster and easier with AXI4-Lite slave drivers along with the existing bare-metal driver support.

- Verification is improved with the support for non-memory-mapped interfaces in hardware co-simulation.

LEARN MORE ABOUT THE VIVADO DESIGN SUITE AND ULTRAFAST DESIGN METHODOLOGY

UltraFast Design Methodology

In order to further strengthen this offering and enable accelerated and predictable design cycles, Xilinx now delivers the first comprehensive design methodology in the programmable industry. Xilinx has hand-picked the best practices from experts and distilled them into an authoritative set of methodology guidelines known as the UltraFast™ Design Methodology for the Vivado Design Suite.

The UltraFast Design Methodology enables project managers and engineers to accelerate time-to-market and quickly tune their sources, constraints and settings to accurately predict schedules. The new "Design Methodology Guide" covers all the aspects of:

- Board and device planning

- Design creation and IP integration

- Implementation and design closure

- Configuration and hardware debug

Vivado QuickTake 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.

Vivado Training

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

Latest and Greatest from the Xilinx Alliance Program Partners

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

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

AVNET SOFTWARE-DEFINED RADIO KIT SHOWCASES ZYNQ SOC

http://www.zedboard.org/product/zynq-sdr-ii-eval

The Zynq SDR-II Evaluation Kit from Avnet (Phoenix) combines the ZedBoard with the Analog Devices AD-FMCOMMS2-EBZ FMC module, which features the Analog Devices AD9361 integrated RF Agile Transceiver. This kit enables a broad range of transceiver applications for wireless communications. Tuned to a narrow RF range in the 2,400-MHz to 2,500-MHz region, the kit is ideal for the RF engineer seeking optimized system performance that meets datasheet specifications in a defined range of RF spectrum. The kit also includes four Pulse LTE blade antennas, the Xilinx Vivado® Design Edition (device locked to ZC7020), an 8-Gbyte SD card, a "Getting Started" card and downloadable documentation and reference designs.

Avnet's second-generation Zynq SDR kit, the Zynq SDR-II Evaluation Kit, combines the ZedBoard with the Analog Devices AD-FMCOMMS2-EBZ FMC module, which features the Analog Devices AD9361 integrated RF Agile Transceiver. In a more focused RF range, the kit's enhanced functionality enables a wide range of transceiver applications for wireless communications.

OMNITEK ZYNQ SOC BROADCAST DEVELOPMENT KIT

http://omnitek.tv/xilinx-oz745-dev-platform

The OZ745 Kit from OmniTek (Basingstoke, U.K.) incorporates all the basic components of hardware, design tools, IP and preverified reference designs needed to rapidly develop video- and image-processing designs, along with a general board support package. The OZ745 board is delivered with an evaluation reference design that recognizes and displays SDI, HDMI and analog video inputs, and displays a test pattern on the SDI and HDMI video outputs. Also supplied is a demonstration of Xilinx's RTVE 2.1 reference design, which has the OmniTek Scalable Video Processor IP (OSVP) at its core. The reference design deinterlaces and resizes four video inputs and composites them onto the video output. Control software runs on the Zynq® All Programmable SoC's dual-core Cortex™-A9 processor, which uses a Linux build with Qt graphics support. An OmniTek 2D graphics IP core provides graphics acceleration. The control software generates a web page that can be hosted locally and composited over the video on the SDI, HDMI or LVDS flat-panel display output. Control is via mouse and keyboard.

INTOPIX'S VISUALLY LOSSLESS VIDEO COMPRESSION BOASTS TINY FOOTPRINT FOR ARTIX-7 AND SPARTAN-6

http://www.intopix.com/uploaded/Download%20Products/intoPIX-TICO%20FLYER_XILINX.pdf

TICO Compression from intoPIX (Mont-Saint-Guibert, Belgium) is a new patent-pending visually lossless light compression technology specifically designed for the AV industry. This revolutionary technology's extremely tiny footprint means it fits in the smallest Xilinx Artix®-7 and Spartan®-6 devices. Yet, TICO is robust enough for real-time operation with no latency.

Up to now, image and video have been sent or stored uncompressed into many displays and systems such as cameras, videos servers and recorders. TICO is a smart upgrade path for managing higher resolutions (4K, 8K and beyond) and frame rates while assuring visual quality, keeping power and bandwidth within budget, and significantly reducing the complexity and cost of the system.

TICO is ideal for applications from HD to Ultra HD including DVRs, video servers, high-resolution and high-speed cameras, video-over-IP systems, surveillance systems and cable extenders. The technology especially suits applications that support higher data streams on existing networks, because it increases the number of streams in a multistream configuration, slashes the internal video bandwidth and associated power consumption, and reduces the number of lanes need to transport a stream in a display interface.

SILICON SOFTWARE AND MVTEC SOFTWARE DELIVER SMART, COMPACT VISION SYSTEM ON ZYNQ SOC

http://press.xilinx.com/2013-11-21-Xilinx-All-Programmable-Solutions-for-Smarter-Factories-and-Smarter-Vision-Showcases-at-SPS-IPC-Drives-2013

A pair of German companies—Silicon Software (Mannheim) and MVTec Software GmbH (Munich)—have teamed up to deliver a demonstration of high-speed optical character recognition (OCR) systems performing real-time silicon device code recognition. The solution utilizes the Zynq SoC and the HALCON machine vision software from MVTec. The companies accomplished hardware acceleration using VisualApplets from Silicon Software. VisualApplets, a software tool for programming image-processing tasks on Xilinx devices, reduces development time and design complexity.

The functional description takes place on the base of graphical block diagrams; furthermore, with function modules, designs are compiled that can be synthesized into executable hardware code. The modules are organized in thematic libraries and cover the essential functional areas of image processing, from basic to complex operators.

Although knowledge in hardware programming is an advantage, VisualApplets is directed at software developers. With the help of ISE®, the graphical representation of the hardware description (VisualApplets design) is transformed into a Xilinx-targeted bitstream that is compiled to a VisualApplets hardware applet with additional runtime information. Designers can then load this applet to the frame grabber via microDisplay or SDK instructions.

HALCON is the comprehensive standard software for machine vision with an integrated development environment (IDE) that is used worldwide. HALCON's flexible architecture facilitates rapid development of machine-vision, medical-imaging and image-analysis applications. The architecture provides outstanding performance and a comprehensive support of multicore platforms, SSE2 and AVX, as well as GPU acceleration. It serves all industries with a library of more than 1,800 operators for blob analysis, morphology, matching, measuring, identification and 3D vision, to name just a few. Contact Silicon Software (*http://www.silicon-software.info/en/*) and MVTec (*http://www.mvtec.com/products/*) for more information.

Application Notes

If you want to do a bit more reading about how our FPGAs lend themselves to a broad number of applications, we recommend these application notes.

XAPP1179: USING TANDEM CONFIGURATION FOR PCIE IN THE KINTEX-7 CONNECTIVITY TRD

http://www.xilinx.com/support/documentation/application_notes/xapp1179-tandem-config-pcie.pdf

The PCI Express® specification requires the PCIe® link to be ready to connect with a peer within 120 milliseconds after power is stable. Meeting this requirement is a challenge for large FPGAs using flash memory for configuration due to the size of the programming bitstream and the configuration rates available. The Tandem Configuration approach from Xilinx® is a practical way to reduce FPGA configuration time to meet the 120-ms PCIe link-training requirement.

This application note by Sunita Jain, Mrinal Sarmah and David Dye shows how to use the Tandem PROM and the Tandem PCIe configuration methods with the Kintex®-7 Connectivity Targeted Reference Design (TRD) running on the KC705 evaluation board with a Kintex-7 XC7K325T FPGA. The design describes the adjustments made to the TRD to accommodate Tandem Configuration. Using this approach, the base bitstream size, and therefore the initial configuration time, is reduced by more than 85 percent when using Tandem PROM and more than 80 percent when using Tandem PCIe.

XAPP1184: PIPE MODE SIMULATION USING INTEGRATED ENDPOINT PCI EXPRESS BLOCK IN GEN2 X8 CONFIGURATION

http://www.xilinx.com/support/documentation/application_notes/xapp1184-PIPE-mode-PCIe.pdf

Verifying designs involving high-speed serial protocols such as PCI Express can be complex and time-consuming. Many verification projects utilize third-party bus functional models (BFMs) to reduce the complexity of the verification process and to speed up the time spent running the actual simulation. Gigabit transceivers are a particular problem for verification, since the GTs often consume a significant number of processor cycles to simulate. For this reason, and because GTs typically have little impact on the behavior of the upper PCI Express layer functionality, many verification projects bypass them for much of their verification and only simulate with GTs to validate the design at the end of a project

The PHY Interface for the PCI Express Architecture (PIPE) is a specification for linking the PCI Express block and the GTs. This application note by K. Murali Govinda Rao and A. V. Anil Kumar provides a way to connect the PIPE interface of the PCI X-actor BFM (in root complex mode) from Avery Design Systems to the PIPE interface of a Xilinx 7 series FPGA Integrated PCI Express Endpoint Block. When configured with the proper options, the Xilinx PCIe Endpoint will have PIPE ports at the core's top level. You can connect these ports to the X-actor RC BFM to bypass simulating with the GTs.

While this application note demonstrates specific connections to the Avery BFM, it can also serve as a model for how to connect other third-party BFMs to the Integrated PCIe Endpoint Block through the PIPE interface. PIPE-mode simulation is very useful for reducing the simulation time during verification of complex PCI Express applications.

XAPP1097: IMPLEMENTING SMPTE SDI INTERFACES WITH ARTIX-7 FPGA GTP TRANSCEIVERS

http://www.xilinx.com/support/documentation/application_notes/xapp1097-smpte-sdi-a7-gtp.pdf

The serial digital interface (SDI) family of standards from the Society of Motion Picture and Television Engineers (SMPTE) is widely used in professional broadcast studios and video production centers to carry uncompressed digital video, along with embedded ancillary data such as multiple audio channels. The Xilinx SMPTE SD/HD/3G-SDI LogiCORE™

IP is a generic SDI receive/transmit datapath that does not have any device-specific control functions. This application note provides a module containing control logic to couple the Xilinx SDI core with the Artix®-7 FPGA GTP transceivers to form a complete SDI interface. Author John Snow describes this additional control and interface logic and provides the necessary control and interface modules in both Verilog and VHDL source code. Also supplied is a wrapper file that contains an instance of the control module for the GTP transceiver and the SDI core with the necessary connections between them. This wrapper file simplifies the process of creating an SDI interface. The application note also provides two example SDI designs that run on the Artix-7 FPGA AC701 evaluation board.

XAPP1094: CTLE ADAPTATION LOGIC FOR 7 SERIES FPGA GTX TRANSCEIVERS

http://www.xilinx.com/support/documentation/application_notes/xapp1094-ctle-adaptation-gtx.pdf

The 7 series FPGA GTX receiver in digital front-end (DFE) mode contains an automatic gain control (AGC) block and a continuous-time linear equalizer (CTLE) block to compensate for channel loss. Both the AGC block and the CTLE wideband gain stages aim to boost frequencies within the operating frequency range of the GTX transceiver to optimize the eye height of the received signal. Although AGC is auto-adaptive, the CTLE wideband stage is not. By default, the user adjusts the wideband gain by analyzing the channel loss. This application note by David Mahashin explains how to use a module implemented in the FPGA logic to automatically adjust CTLE wideband gain. The one-time calibration occurs after deassertion of GTRXRESET, RXPMARESET or RXDFELPMRESET. This feature allows dynamic adjustment of the equalizer gain without requiring channel-loss analysis. The module is included in the 7 series FPGAs' Transceivers Wizard design as an optional feature.

XAPP1185: ZYNQ-7000 PLATFORM SOFTWARE DEVELOPMENT USING THE ARM DS-5 TOOLCHAIN

http://www.xilinx.com/support/documentation/application_notes/xapp1185-Zynq-software-development-with-DS-5.pdf

This document offers guidance on using the ARM® Development Studio 5 (DS-5) design suite to develop, build and debug bare-metal software for the Xilinx Zynq®-7000 All Programmable SoC, which is based on the ARM Cortex™-A9 processor. Authors Simon George and Prushothaman Palanichamy walk you through the process, beginning with creating a board support package (BSP) for custom hardware design within the Xilinx software development kit (SDK), then importing that BSP into the DS-5 tools for a build. Next comes the creation of the first-stage boot loader and finally, the debugging of the target configuration for your Zynq SoC custom design.

XAPP1182: SYSTEM MONITORING USING THE ZYNQ-7000 AP SOC PROCESSING SYSTEM WITH THE XADC AXI INTERFACE

http://www.xilinx.com/support/documentation/application_notes/xapp1182_zynq_axi_xadc_mon.pdf

This application note by Mrinal J. Sarmah and Radhey S. Pandey describes how to use a Xilinx analog-to-digital converter (XADC) for system monitoring applications. The XADC Wizard IP offers an AXI4-Lite interface that connects to the AXI general-purpose port in a Zynq-7000 All Programmable SoC processing system to get system control information from the XADC. The XADC block provides dedicated alarm output signals that trigger based on preset events. A Linux application running on the Zynq SoC's ARM Cortex-A9 CPU controls the alarm threshold of the XADC and monitors the alarm output. This design also explores the possibility of using an external auxiliary channel through the AXI4-Lite interface and characterizes the maximum signal frequency that can be monitored using that interface.

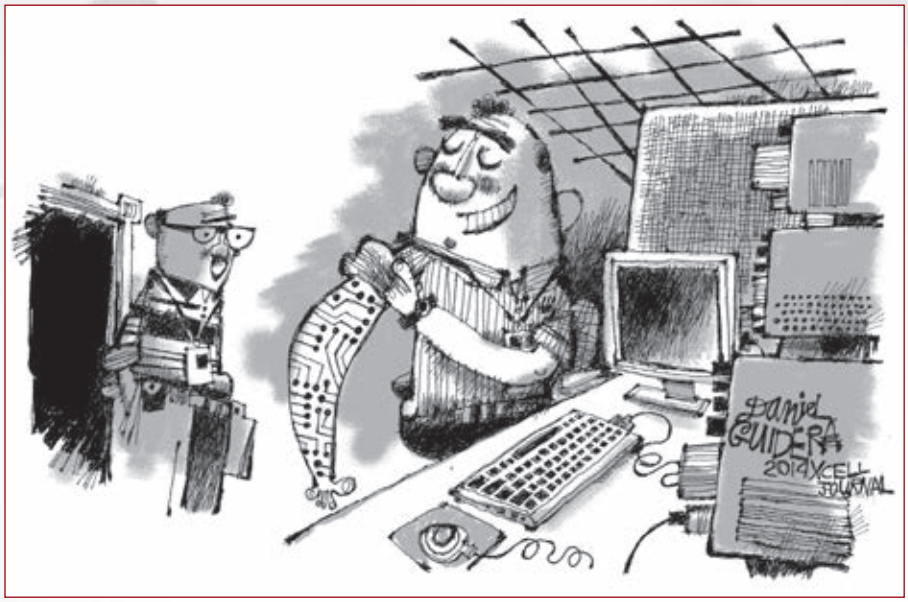
XAPP1183: IMPLEMENTING ANALOG DATA ACQUISITION USING THE ZYNQ-7000 AP SOC PROCESSING SYSTEM WITH THE XADC AXI INTERFACE

http://www.xilinx.com/support/documentation/application_notes/xapp1183-zynq-xadc-axi.pdf

In a second application note, the same authors go on to describe how the XADC acquires analog data using its dedicated Vp/Vn analog input. The design by Mrinal J. Sarmah and Radhey S. Pandey implements a use case where the XADC out data is transferred directly to system memory using the Xilinx direct memory access (DMA) IP. A Linux-based application running on a Zynq-7000 All Programmable SoC processing system reads the buffer from memory. Then, a LabVIEW-based application GUI gathers the data and performs fast Fourier transform (FFT) processing on it to quantify the signal-to-noise ratio (SNR) of the XADC out data.

This design provides a platform for using the AXI4-Stream interface of the XADC Wizard IP for analog data-acquisition applications. The authors show how to use AXI DMA to transfer the XADC samples into processor memory without processor intervention. The design quantitatively analyzes the XADC performance metric for different frequency tones.

Xpress Yourself in Our Caption Contest



DANIEL GUIDERA

Who said tattoos can't be a professional asset? If you've ever wished you had your latest circuit diagram at your fingertips, inking it on your arm might be an alternative. Xercise your funny bone by submitting an engineering- or technology-related caption for this cartoon showing an engineer admiring a colleague's work-inspired tats. The image might inspire a caption like "Nice, right? I got them at a new booth they set up at the IEEE conference on programmable devices I went to last week."

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 Digilent Zynq Zybo board, featuring the Xilinx® Zynq®-7000 All Programmable SoC (*http://www.xilinx.com/products/boards-and-kits/1-4AZFTE.htm*). Two runners-up will gain notoriety, fame and a cool, Xilinx-branded gift from our swag closet.

The contest begins at 12:01 a.m. Pacific Time on Jan. 17, 2014. All entries must be received by the sponsor by 5 p.m. PT on April 1, 2014.

So, think ink … and get writing!

GLENN BABECKI, principal system engineer II at Comcast Corp. (Horsham, Pa.), won a shiny new Avnet ZedBoard with this caption for the ping-pong cartoon in Issue 85 of *Xcell Journal*:

"… and that, my friends, is how I came up with the concept of 'multicore shared ping-pong buffers!' "

Congratulations as well to our two runners-up:

"Pong would have been in 3D from the beginning if Nolan Bushnell had had access to FPGAs back then."

— Wolfgang Friedrich, electrical engineer, product development, SMART Technologies (Calgary, British Columbia, Canada)

"While interviewing for the chief architect position, Jim is asked to prove his multitasking abilities."

— Marcel Ursu, FPGA engineer, AdvancedIO (Vancouver, British Columbia, Canada)

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

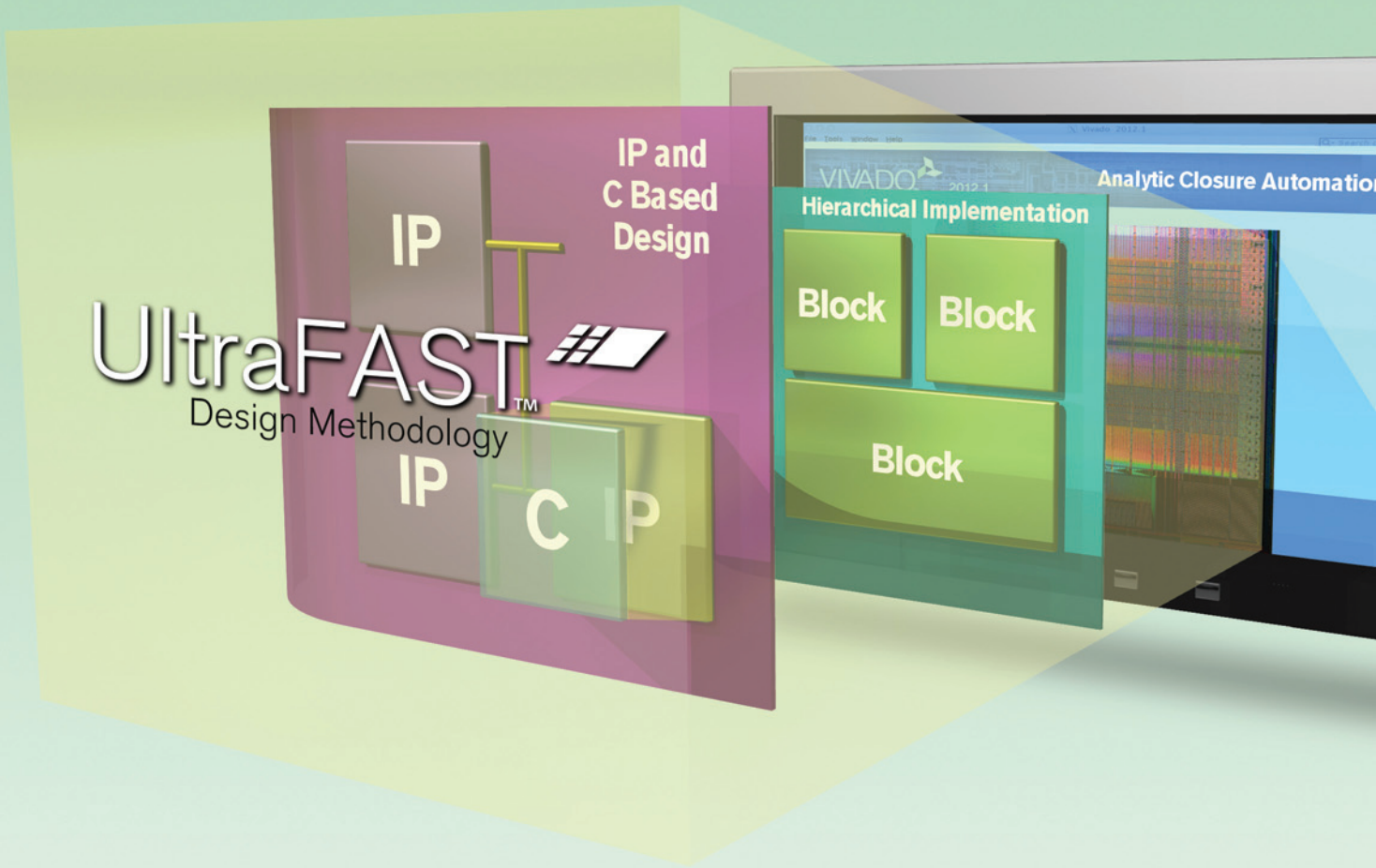
FPGAs keep getting bigger, but your schedule is not. There is no time to waste on numerous design iterations and long tool runtimes. Use the hierarchical and incremental techniques available in Synopsys' Synplify® software to bring in your schedule and meet aggressive performance goals.

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

SYNOPSYS®
Accelerating Innovation

Xilinx Introduces

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



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

VIVADO

XILINX
ALL PROGRAMMABLE™

Learn More: www.xilinx.com/ultrafast