Xcell journal

ISSUE 75, SECOND QUARTER 2011

SOLUTIONS FOR A PROGRAMMABLE WORLD

Xilinx Pioneers New Class of Device in Zynq-7000 EPP Family

High-Performance FPGAs Fly in U.K. Microsatellites

Excerpt: *FPGA-Based Prototyping Methodology Manual*

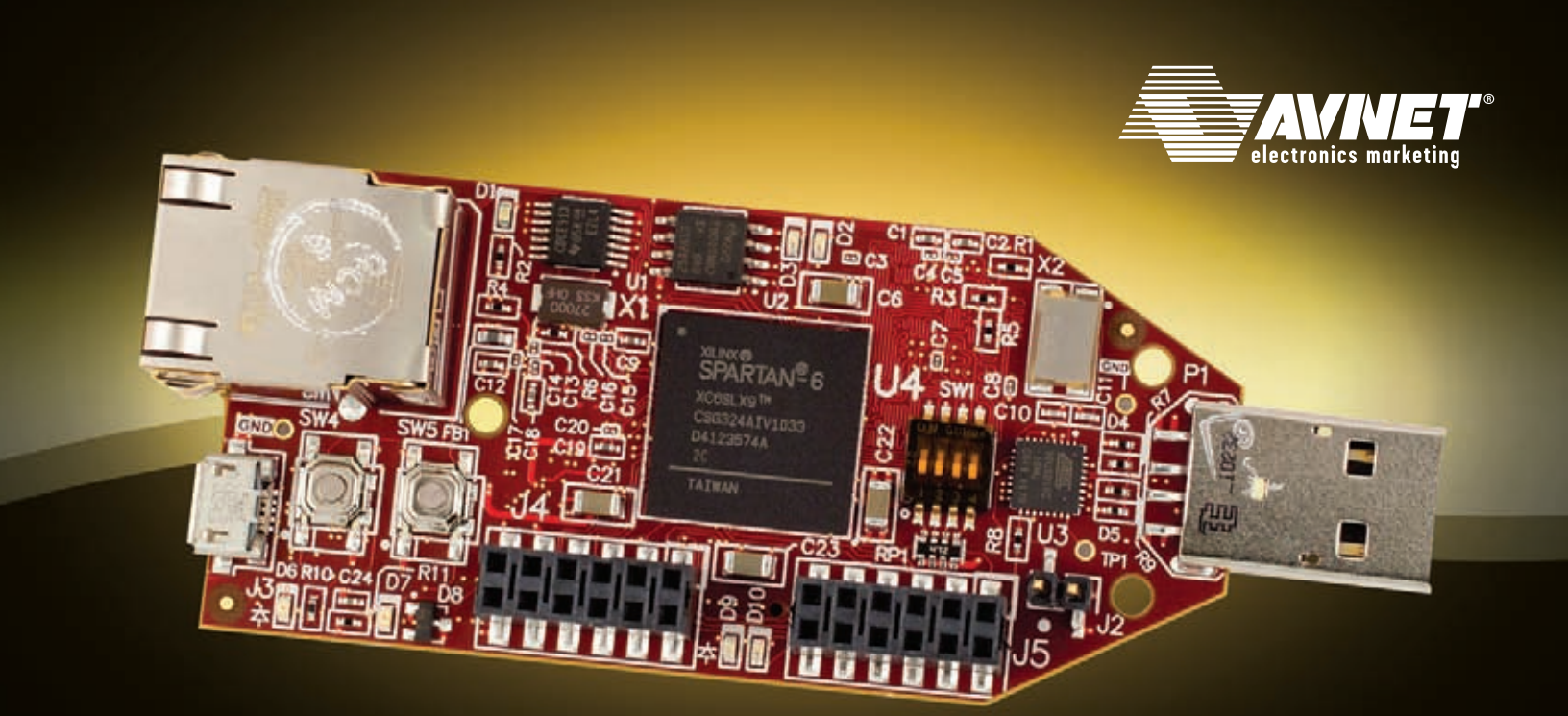
ISE 13.1 Rolls, with PlanAhead Enhancements

Xilinx FPGAs Beam Up Advanced Radio Astronomy in Australia page 30

XILINX®

www.xilinx.com/xcell/

AVNET electronics marketing



DESIGNED BY AVNET

Compact, Easy-to-Use Kit Demonstrates the Versatility of Spartan-6 FPGAs

The low-cost Spartan®-6 FPGA LX9 MicroBoard is the perfect solution for designers interested in exploring the MicroBlaze™ soft processor or Spartan-6 FPGAs in general. The kit comes with several pre-built MicroBlaze "systems" allowing users to start software development just like any standard off-the-shelf microprocessor. The included Software Development Kit (SDK) provides a familiar Eclipse-based environment for writing and debugging code. Experienced FPGA users will find the MicroBoard a valuable tool for general-purpose prototyping and testing. The included peripherals and expansion interfaces make the kit ideal for a wide variety of applications.

Xilinx® Spartan®-6 FPGA LX9 MicroBoard Features

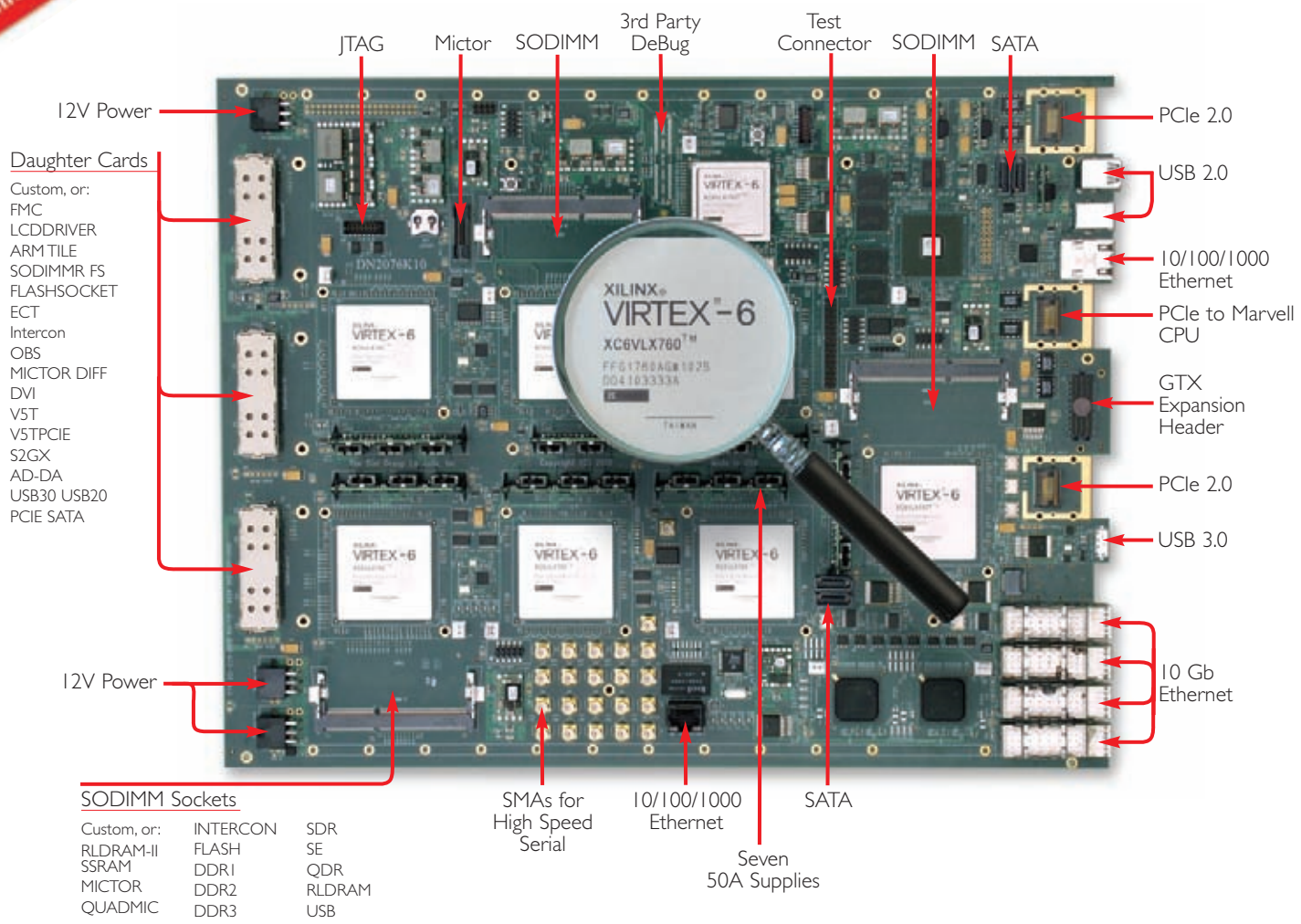
- Avnet Spartan-6 FPGA LX9 MicroBoard
- ISE® WebPACK® software with device locked SDK and ChipScope licenses
- Micro-USB and USB extension cables
- Price: $89

To purchase this kit, visit www.em.avnet.com/s6microboard or call 800.332.8638.

©Avnet, Inc. 2011. All rights reserved. AVNET is a registered trademark of Avnet, Inc.

See us at DAC, and have fun in San Diego www.dinigroup.com/DAC2011

Why build your own ASIC prototyping hardware?



DN2076K10 ASIC Prototyping Platform

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

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

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

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

DINI Group

www.dinigroup.com • 7469 Draper Avenue • La Jolla, CA 92037 • (858) 454-3419 • e-mail: sales@dinigroup.com

Xcell journal

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

EDITOR Jacqueline Damian

ART DIRECTOR Scott Blair

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

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

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

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

Miyuki Takegoshi, Japan
miyuki.takegoshi@xilinx.com

REPRINT ORDERS 1-800-493-5551

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/

© 2011 Xilinx, Inc. All rights reserved. XILINX, the Xilinx Logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

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

It's Spring—Let the Innovation Begin!

It's springtime! The flowers are blooming, the bees are buzzing and the IC design community is readying its latest wares for the beginning of its trade show season. The Embedded Systems Conference and Design Automation Conference are my two favorites, as they display the latest embedded software and hardware design tools from vendors across the world.

At this year's ESC in San Jose (May 2-5), Xilinx will showcase its Zynq™-7000 Extensible Processing Platform (EPP) in Booth #1332. I fully expect this exciting new technology—the subject of this issue's cover story (see page 8)—to be a game-changer for the electronics industry. From silicon decisions to the design software infrastructure, this extremely well-thought-out architecture will immediately snatch up sockets now held by two-chip solutions (pairing an ARM SoC with an FPGA) as well as standalone ASICs, as process technology complexity, cost and thus risk simply become more prohibitive for a growing number of applications. With the Zynq-7000 EPP, design teams get a high-end ARM MPU tightly coupled with programmable logic on the same IC—a powerful one-two punch that simply isn't available with other SoCs.

In fact, I predict that with an architecture that boots the processor first, the Zynq-7000 EPP will appeal to a much broader base of designers, opening up new design possibilities for folks with software backgrounds, tinkerers as well as those traditionally designing and programming ICs. With a starting price at less than $15 (in volume) for the smallest of these relatively immense devices, the Zynq-7000 EPP is simply too cool not to at least try.

In addition to the excitement of the Embedded Systems Conference, I always look forward to the Design Automation Conference (San Diego, June 5-10), at which the small but mighty EDA industry shows its latest wares. Over the last two years, I've served as a member of DAC's Pavilion Panel organizing committee. This year I'm putting together three intriguing panels (see *http://www.dac.com/conference+program+panels.aspx*).

On Monday, Herb Reiter, GSA consultant for 3-D IC programs, will moderate a panel called "3-D IC: Myth or Miracle?" Panelist Ivo Bolsens, Xilinx's own CTO, will be discussing Xilinx's new stacked silicon interconnect technology (see cover story, *Xcell Journal* Issue 74, *http://www.xilinx.com/publications/archives/xcell/issue74/cover-story-stacked-loaded.pdf*). Joining Ivo and Herb will be Riko Radojcic from Qualcomm and TSMC's Suk Lee.

On Tuesday, my buddy over at *EE Journal*, Kevin Morris, is moderating a panel called "C-to-FPGA Tools: Ready for the Mass Market?" This event has a stellar panel, including Jeff Bier of BDTI, Mentor Graphics' Simon Bloch and Synopsys' Johannes Stahl. As the title suggests, they'll be discussing the viability and market opportunities for C-to-FPGA tool flows.

Last but not least, venture capitalist Lucio Lanza will command a panel on Wednesday called "What EDA Isn't Doing Right." This will be a no-holds-barred discussion between Lucio and a group of power EDA users—Behrooz Abdi of NetLogic Microsystems, Charles Matar of Qualcomm and PMC Sierra's Alan Nakamoto—on the real pain points of design today and what EDA needs to do to address them and, in turn, grow the EDA market further.

To learn more, visit *http://esc.eetimes.com/siliconvalley/* and *http://www.dac.com/dac+2011.aspx*. There's a lot to look forward to as the weather warms and the trade show season begins.

Mike Santarini
Publisher

Let our XMC modules do the "talking"...

X6 400m

– the final word in SDR, RADAR, and Medical Imaging.

Features

- Two 400 MSPS 14-bit A/D channels
- Two 500 MSPS 16-bit D/A channels
- Xilinx Virtex6 SX315T/SX475T or LX240T
- 4 Banks of 1GB DRAM (4 GB total)
- 16W typical
- Ruggedized levels available

Extremely Versatile

Complete chassis & embedded PC solutions available! Adapt to VPX, Cabled PCI-Express, PCI-Express, 64-bit PCI, and CompactPCI with our high-quality carriers.

6 VIRTEX

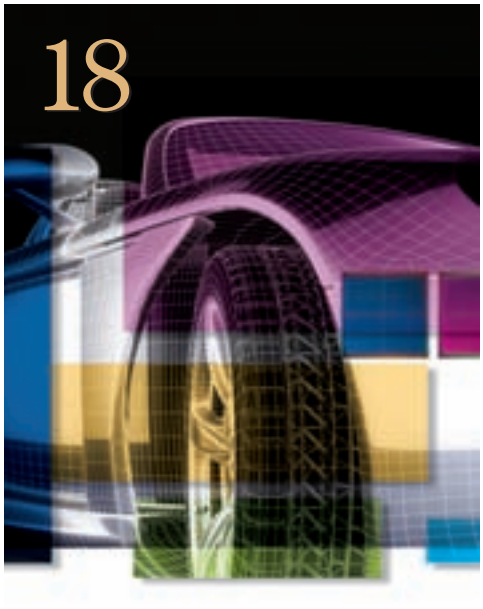
wireless ip cores

FrameWorkLogic

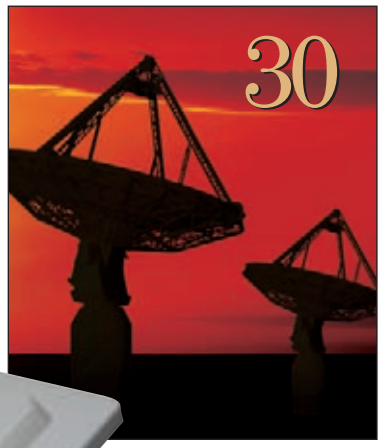
Innovative Integration
a subsidiary of ISI
... real time solutions!

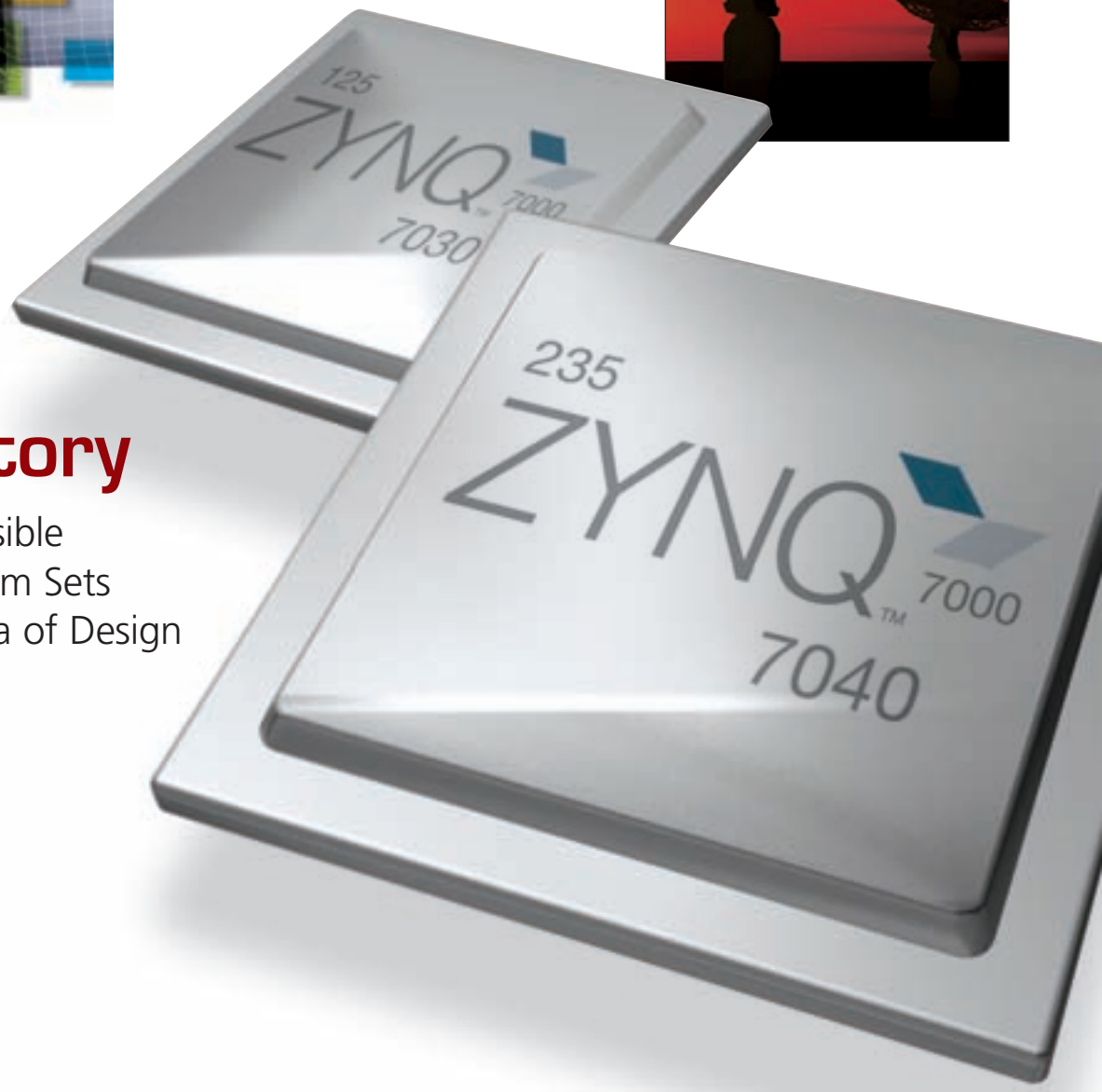
805.520.4260 phone • www.innovative-dsp.com

VIEWPOINTS



XCELLENCE BY DESIGN APPLICATION FEATURES





Cover Story

Zynq-7000 Extensible Processing Platform Sets Stage for New Era of Design

THE XILINX XPERIENCE FEATURES

XTRA READING

2010 APEX® AWARDS FOR PUBLICATION EXCELLENCE

DIGITAL MAGAZINE AWARDS 2010 FINALIST

Zynq-7000 EPP Sets Stage for New Era of Innovations

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

Xilinx's Zynq-7000 Extensible Processing Platform family mates a dual ARM Cortex-A9 MPCore processor-based system on the same device with programmable logic and hardened IP peripherals, offering the ultimate mix of flexibility, configurability and performance.

Xilinx has just unveiled the first devices in a new family built around its Extensible Processing Platform (EPP), a revolutionary architecture that mates a dual ARM Cortex™-A9 MPCore processor with low-power programmable logic and hardened peripheral IP all on the same device (see cover story, spring 2010 issue of *Xcell Journal*, *http://www.xilinx.com/publications/archives/xcell/Xcell71.pdf*). In March of this year, Xilinx officially announced the first four devices of what it has now dubbed the Zynq™-7000 EPP family.

Implemented in 28-nanometer process technology, each Zynq-7000 device is built with an ARM dual-core Cortex-A9 MPCore processing system equipped with a NEON media engine and a double-precision floating-point unit, as well as Level 1 and Level 2 caches, a multimemory controller and a slew of commonly used peripherals (Figure 1). While FPGA vendors have previously fielded devices with both hardwired and soft onboard processors, the Zynq-7000 EPP is unique in that the ARM processor system, rather than the programmable logic, runs the show. That is, Xilinx designed the processing system to boot at power-up (before the FPGA logic) and to run a variety of operating systems independent of the programmable logic fabric. Designers then program the processing system to configure the programmable logic on an as-needed basis.

With this approach, the software programming model is exactly the same as in standard, fully featured ARM processor-based systems-on-chip (SoCs). Previous implementations required designers to program the FPGA logic to get the onboard processor to work. That meant you had to be an FPGA designer to use the devices. This is not the case with the Zynq-7000 EPP.

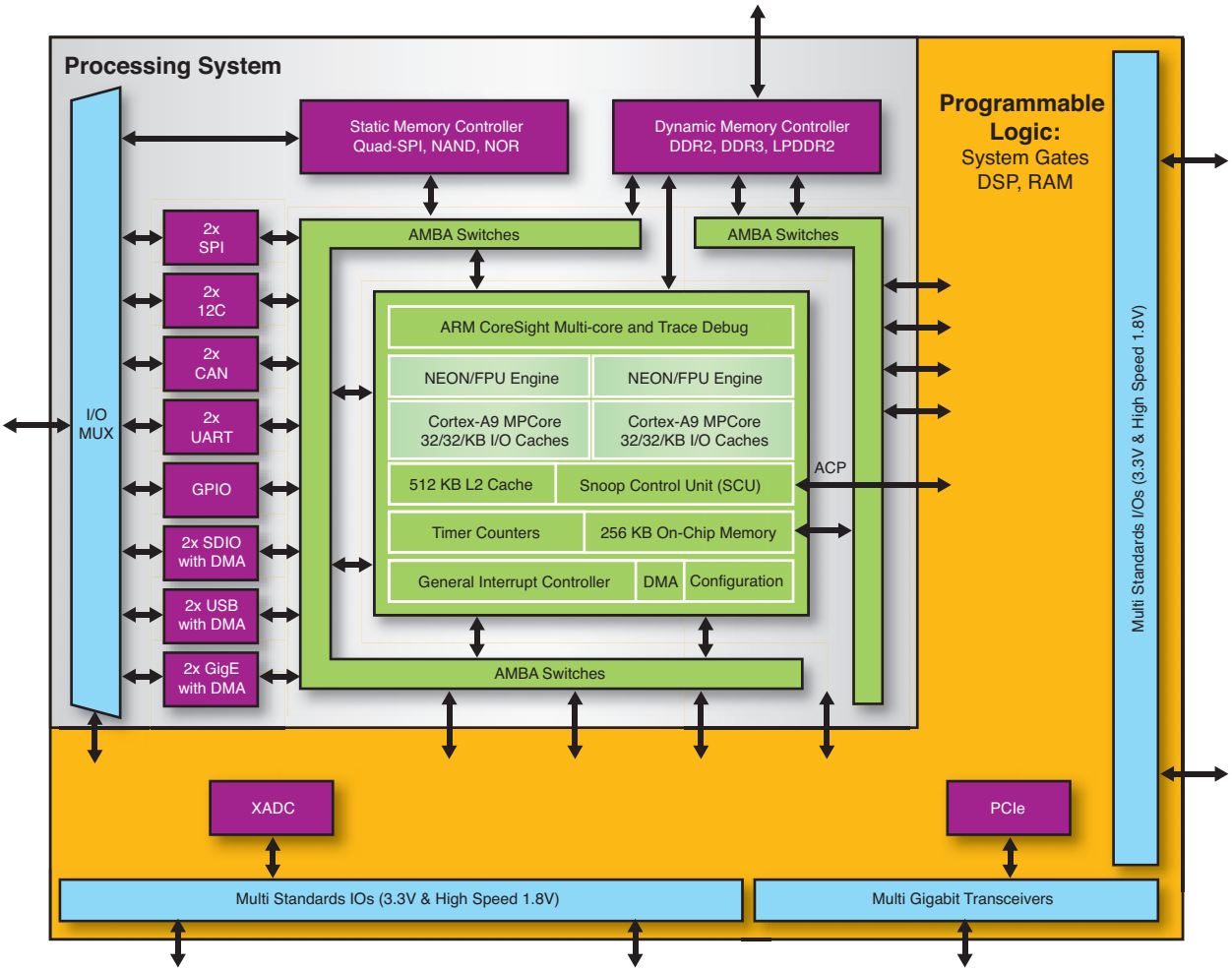


Figure 1 – Unlike previous chips that combine MPUs in an FPGA fabric, Xilinx's new Zynq-7000 EPP family lets the ARM processor, rather than the programmable logic, run the show.

The new product family eliminates the delay and risk of designing a chip from scratch, meaning system design teams can quickly create innovative SoCs leveraging advanced hardware and software programming versatility simply not achievable in any other semiconductor device. As such, the Zynq-7000 EPP stands poised to allow a broader number of innovators—whether they are professional hardware, software or systems designers, or simply "makers"—to explore the possibilities of combining processing plus programmable logic to create applications no one has yet imagined.

"At its most basic level, Zynq-7000 EPP is an entirely new class of semiconductor product," said Larry Getman, vice president of processing platforms at Xilinx. "It is not just a processor and it is not just an FPGA. We are combining the best of both those worlds, and because of that we take away many of the limitations you have with existing solutions, especially two-chip solutions and ASICs."

Getman notes that many electronic systems today pair an FPGA and either a standalone processor or an ASIC with an onboard processor on the same PCB. Xilinx's new offering will allow companies using these types of two-chip solutions to build next-generation systems with just one Zynq-7000 chip, saving bill-of-material costs and PCB space, and reducing overall power budgets. And because the processor and FPGA are on the same fabric, the performance increase is immense.

Zynq-7000 EPP will also speed up the natural market migration from ASICs to FPGAs, Getman said. Implementing ASICs in the latest process technologies is too expensive and too risky for a growing number of applications. As a result, more and more companies are embracing

'With a starting price point below $15, we are really making it hard for companies to justify the cost and risk of designing any ASIC that is not extremely high volume.'

FPGAs. Many of those attempting to hold onto their old ASIC ways are implementing their designs in older process geometries in what analysts call "value-minded SoC ASICs." Any ASIC still requires lengthy design cycles and is at risk of multiple respins—which can be expensive and can delay products from going to market in a timely manner. "With Zynq-7000 EPP on 28 nm, the programmable logic portion of the device has no size or performance penalty vs. older technologies, and you also get the added benefit of a hardened 28-nm SoC in the processing subsystem. With a starting price point below $15, we are really making it hard for companies to justify the cost and risk of designing any ASIC that is not extremely high volume," said Getman. "You can get your software and hardware teams up and running from day one. That alone makes it hard for engineering teams to justify staying with ASICs."

Getman notes that ever since Xilinx announced the architecture last year, interest in and requests for the Zynq-7000 EPP have been remarkable. "A select number of alpha customers are already prototyping systems that will use Zynq-7000 devices. The technology is very exciting."

SMART ARCHITECTURAL DECISIONS

The Zynq-7000 EPP design team, under the direction of Vidya Rajagopalan, vice president of processing solutions at Xilinx, designed a very well-thought-out architecture for this new class of device. Above and beyond choosing the ubiquitous and immensely popular ARM processor system, a key architectural decision was to extensively use the high-bandwidth AMBA® Advanced Extensible Interface (AXI™) interconnect between the processing system and the programmable logic. This enables multigigabit data transfers between the ARM dual-core Cortex-A9 MPCore processing subsystem and the programmable logic at very low power, thereby eliminating common performance bottlenecks for control, data, I/O and memory.

In fact, Xilinx teamed with ARM to make the ARM architecture an even better fit for FPGA applications. "AXI4 has a memory-mapped version and a streaming version," said Rajagopalan. "Xilinx drove the streaming definition for ARM because a lot of IP that people build for applications such as high-bandwidth video is streaming IP. ARM didn't have a product that had this streaming interface so they partnered with us to do it."

Getman said another key aspect of the architecture is that Xilinx hardened a healthy mix of standard interface IP into Zynq-7000 EPP silicon. "We tried to choose peripherals that were more ubiquitous—things like USB, Ethernet, SDIO, UART, SPI, I2C and GPIO are all pretty standard," said Getman. "The one exception is that we also added CAN to the device. CAN is one of the more specialized hardened cores, but it is heavily used in two of our key target markets: industrial and automotive. Having it hardened in the device is just one more comfort factor of the Zynq-7000 EPP."

In terms of memory, Zynq-7000 devices offer up to 512 kbytes of L2 cache that is shared by both processors. "The Zynq-7000 EPP devices have 256 kbytes of scratchpad, which is a shared memory that the processor and FPGA can both access," said Getman.

A unique multistandard DDR controller supports three types of double-data-rate memory. "Where most ASSPs target a particular segment of a market, we target LPDDR2, DDR2 and DDR3, so the user can make the trade-off of whether they want to go after power or performance," said Rajagopalan. "It is a multistandard DDR controller and we are one of the first companies to offer a controller like this."

In addition to being a new class of device, Zynq-7000 EPP is also the latest Xilinx Targeted Design Platform, offered with base development boards, software, IP and documentation to get customers up and running quickly. Further, the company will roll out over the coming years vertical-market- and application-specific Zynq-7000 EPP Targeted Design Platforms—boards or daughtercards, IP and documentation—all to help design teams get products to market faster (see cover story, *Xcell Journal* Issue 68, *http://www.xilinx.com/publications/archives/xcell/Xcell68.pdf*).

Xilinx Alliance Program members and the ARM Connected Community will also offer customers a wealth of Zynq-7000 EPP resources, including popular operating systems, debuggers, IP, reference designs and other learning and development materials.

In addition to creating great silicon and the tools to go with it, Xilinx has meticulously put together user-friendly design and programming flows for the Zynq-7000 EPP.

PROCESSOR-CENTRIC DEVELOPMENT FLOW

The Zynq-7000 EPP relies on a familiar tool flow that allows embedded-software and hardware engineers to perform their respective development, debug and implementation tasks in much the same way as they do now—using familiar embedded-design methodologies already delivered through the Xilinx® ISE® Design Suite and third-party tools (Figure 2).

Getman notes that software application engineers can use the same development tools they have employed for previous designs. Xilinx provides the Software Development Kit (SDK), an Eclipse-based tool suite, for embedded-software application projects. Engineers can also use other third-party development environments, such as the ARM Development Studio 5 (DS-5™), ARM RealView Development Suite (RVDS™) or any other development tools from the ARM ecosystem

Linux application developers can fully leverage both the Cortex-A9 CPU cores in Zynq-7000 devices in a symmetric-multiprocessor mode for the highest performance. Alternatively, they can set up the CPU cores in a uniprocessor or asymmetric-multiprocessor mode running Linux, a real-time operating system (RTOS) like VxWorks or both. To jump-start software development, Xilinx provides customers with open-source Linux drivers as well as bare-metal drivers for all the processing peripherals (USB, Ethernet, SDIO, UART, CAN, SPI, I2C and GPIO). Fully supported OS/RTOS board support packages with middleware and application software will also be available from the Xilinx and ARM partner ecosystem.

Meanwhile, the hardware design flow is similar to the embedded-processor design flow in the ISE Design Suite, with a few new steps for the Extensible Processing Platform. The processing subsystem is a complete dual-processor

One Processing System, Four Devices

Each of the Zynq-7000 EPP family's four devices has the exact same ARM processing system, but the programmable logic resources vary for scalability and fit different applications (see figure).

The Cortex-A9 Multi-Processor core (MPCore) consists of two CPUs—each a Cortex A9 MPCore processor with dedicated NEON coprocessor (a media and signal-processing architecture that adds instructions targeted at audio, video, 3-D graphics, image and speech processing) and a double-precision floating-point unit. The Cortex-A9 processor is a high-performance, low-power, ARM macrocell with a Level 1 cache subsystem that provides full virtual-memory capabilities. The processor implements the ARMv7™ architecture and runs 32-bit ARM instructions, 16-bit and 32-bit Thumb instructions, and 8-bit Java byte codes in Jazelle state. In addition, the processing system includes a snoop control unit, a Level 2 cache controller, on-chip SRAM, timers and counters, DMA, system control registers, device configuration and an ARM CoreSight™ system. For debug, it contains an embedded trace buffer, instrumentation trace macrocell and cross-trigger module from ARM, along with AXI monitor and fabric trace modules from Xilinx.

The two larger devices, the Zynq-7030 and Zynq-7040, include high-speed, low-power serial connectivity with built-in multigigabit transceivers operating at up to 10.3125 Gbits/second. These devices offer approximately 1.9 million and 3.5 million equivalent ASIC gates (125k and 235k logic cells) respectively, along with DSP resources that deliver 480 GMACs and 912 GMACs respectively of peak performance. The two smaller devices, the Zynq-7010 and Zynq-7020, provide roughly 430,000 and 1.3 million ASIC-gate equivalents (30k and 85k logic cells) respectively, with 58 GMACs and 158 GMACs of peak DSP performance.

The Zynq-7000 Extensible Processing Platform debuts with a family of four devices. All sport the same ARM processing system but vary in programmable logic resources. Gate counts range from 430,000 to 3.5 million equivalent ASIC gates.

Each device contains a general-purpose analog-to-digital converter (XADC) interface, which features two 12-bit, 1-Msample/s ADCs, on-chip sensors and external analog input channels. The XADC offers enhanced functionality over the system monitor found in previous generations of Virtex® FPGAs. The two 12-bit ADCs, which can sample up to 17 external-input analog channels, support a diverse range of applications that need to process analog signals with bandwidths of less than 500 kHz.

‘The most important thing for us was to give software developers and hardware designers their comfortable design environments.’

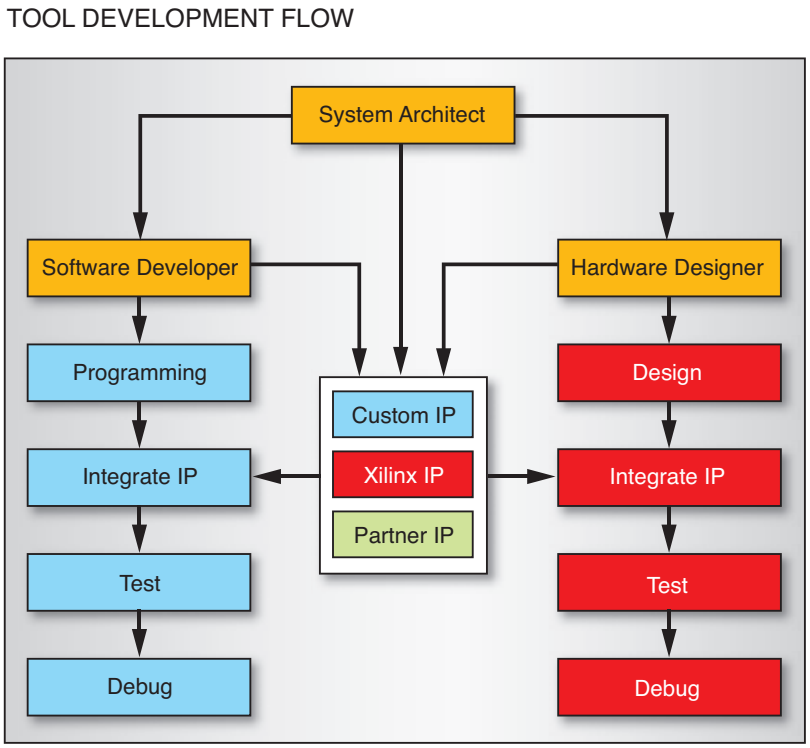
system with an extensive set of commonly used peripherals. Hardware designers can extend the processing power by attaching additional soft-IP peripherals in the programmable logic to the processing subsystem. The hardware development tool Xilinx Platform Studio automates many of the common hardware development steps and can also assist designers with optimized device pinouts. “We’ve also added to ISE some abilities to co-debug for hardware breakpoints and cross-triggering,” said Getman. “The most important thing for us was to give software developers and hardware designers their comfortable design environments.”

A MINDFUL PROGRAMMING METHODOLOGY

In the Xilinx scheme of things, users can configure the programmable logic and connect it to the ARM core through AXI “interconnect” blocks to extend the performance and capabilities of the processor system. The Xilinx and ARM partner ecosystem provides a large set of soft AMBA interface IP cores for implementation in the FPGA programmable logic. Designers can use them to build any custom functions their targeted application requires. Because the device uses familiar programmable logic structures found in the 7 series FPGAs, designers can load a single static programmable logic configuration, multiple configurations or even employ partial reconfiguration techniques to allow the device to reprogram programmable logic functionality as needed on the fly.

The interconnect operation between the two regions of the device is largely transparent to the designers.

Figure 2 – The Zynq-7000 EPP relies on a familiar tool flow for system architects, software developers and hardware designers alike.



Access between master and slave is routed through the AXI interconnect based on the address range assigned to each slave device. Multiple masters can access multiple slaves simultaneously, and each AXI interconnect uses a two-level arbitration scheme to resolve contention.

GET READY, GET IN EARLY…

Customers can start evaluating the Zynq-7000 EPP family today by joining the Early Access program. First silicon devices are scheduled for the second half of 2011, with general engineering samples available in the first half of 2012. Designers can immediately use tools and development kits that support ARM to familiarize themselves with the Cortex-A9 MPCore architecture and begin porting code.

Pricing varies and depends on volume and choice of device. Based on forward volume production pricing, the Zynq-7000 EPP family will have an entry point below $15 in high volumes. Interested customers should contact their local Xilinx representative. For more information, please visit *www.xilinx.com/zynq*.

High-Performance FPGAs Take Flight in Microsatellites

Artist's rendition of the UKube1 CubeSat in flight

Utilizing Xilinx Virtex-4 devices in a U.K. CubeSat mission presents some interesting design challenges

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

The UKube1 mission is the pilot mission for the U.K. Space Agency's planned CubeSat program. CubeSats are a class of nanosatellites that are scalable from the basic 1U satellite (10 x 10 x 10 cm) up to 3U (30 x 10 x 10 cm) and beyond, and which are flown in low-earth orbit. The typical development cost of a CubeSat payload is less than $100,000, and development time is short. This combination makes CubeSats an ideal platform for verifying new and exciting technologies in orbit without the associated overhead or risks that would be present in flying these payloads on a larger mission. Of course, this class of satellites can present its own series of design challenges for the engineers involved.

The EADS Astrium payload for the UKube1 mission comprises two experiments, both of which are FPGA based. The first experiment is the validation of a patent held by Astrium on random-number generation. True random-number generation is an essential component of secure communications systems. The second experiment is the flight of a large, high-performance Xilinx® Virtex®-4 FPGA with the aim of achieving additional in-flight experience with this technology while gaining an understanding of the device's radiation performance and capabilities in the low-earth orbit (LEO). Figure 1 shows the architecture of the payload.

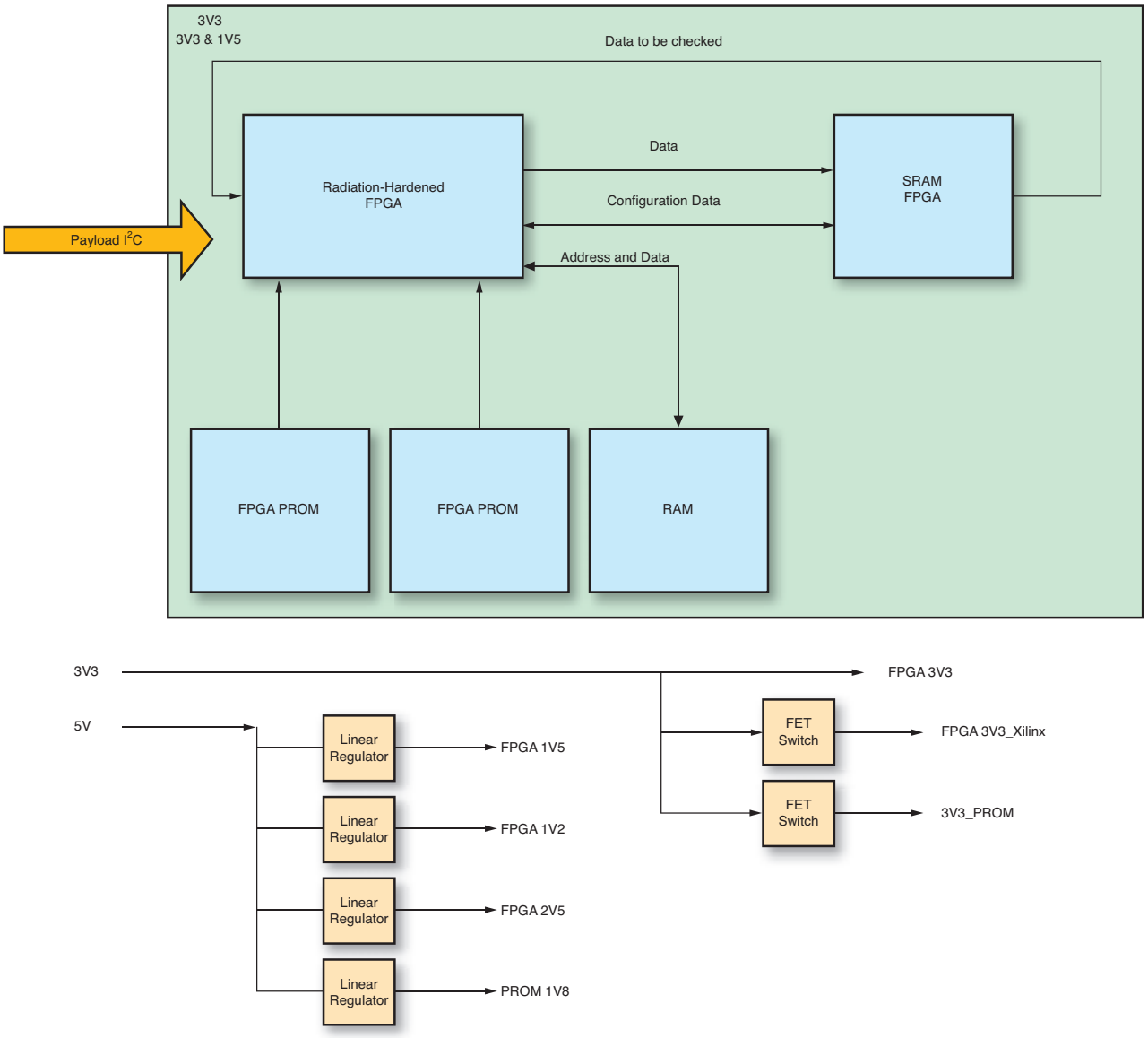


Figure 1 – The EADS Astrium payload for the UKube1 mission comprises two experiments, both of which are FPGA based.

REQUIREMENTS AND CHALLENGES

Designing for a CubeSat mission provides the engineers with many challenges, not least of which is the available power—there is not much of it, at 400 milliwatts on average in sunlight orbit for the UKube1. Weight restrictions come in at a shade over 300 grams for a 3U, 4.5-kg satellite. Combined with the space envelope available for a payload, these limitations present the design team with an interesting set of challenges to address if they are to develop a successful payload. The engineers must also address single-event upsets (SEUs) and other radiation effects, which can affect the performance of a device in orbit regardless of the class of satellite.

The power architecture in UKube1 provides regulated 3.3-, 5- and 12-volt supplies to each of the payloads. It is permissible to take up to 600 mA from each of these rails. However, the sunlit-orbit average must be less than 400 mW. Unfortunately, the voltages supplied are not at levels suitable to supply today's high-performance space-grade FPGAs, which typically require 1.5 V or less to supply the core voltage. For example, the Virtex-4 space-grade device we selected for this mission, the XQR4VSX55 FPGA, requires a core voltage of 1.2 V along with supporting voltages of 2.5 and 3.3 V. The configuration PROMs needed to support the FPGA require 1.8 V.

We selected the XQR4VSX55 because it was the largest high-performance

FPGA that could be accommodated within the UKube1 payload while still achieving both the footprint and power requirements. The power engineer and the FPGA engineer must give considerable thought to the power architecture to ensure all of the power constraints are achieved. Tools such as the PlanAhead™ and XPower Analyzer software are vital for the FPGA engineer to provide FPGA power budgets to the power engineer. We chose high-efficiency switching regulators for this mission to ensure we could achieve the currents required by the SX55 FPGA could be achieved.

The space available to implement the FPGA and its supporting functions is very limited, with the payload being constrained to a PC104-size printed-circuit board. However, mezzanine cards are permitted provided they do not exceed the height restrictions of 35 mm. Therefore, we developed the UKube1 payload to include a mezzanine card that contained the FPGAs, SRAM and flash memory, while the lower board contained all of the power management and conversion functionality (see Figure 2).

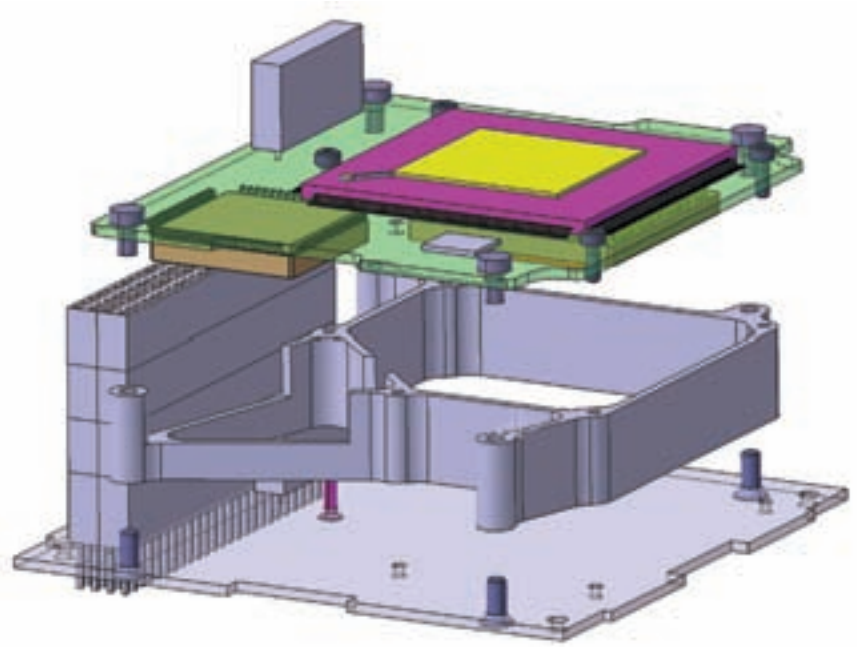


Figure 2 – The FPGAs, SRAM and flash memory reside on a mezzanine card (top), while the power management and conversion functionality occupies the lower board.

RADIATION EFFECTS

One of the most important aspects of this mission is to gain an understanding of the performance of the Virtex-4 device in a LEO environment. While Xilinx has hardened this FPGA for spaceflight, SEUs will still occur and affect both the configuration data and the FPGA registers, RAM and digital clock managers (DCMs). This mission therefore configures the FPGA to use most of the internal logic, RAM and DCMs. We then monitor the performance of this device using another one-time programmable hardened device, which passes performance statistics back down to the ground for analysis. Interestingly, on this mission we are less concerned with SEU- and radiation-mitigation design techniques than on ensuring that these events can be captured such that they can be detected by a monitoring FPGA, allowing for the generation of real in-flight performance statistics.

This aspect of the design presented some interesting engineering challenges. For example, an SEU could affect signals such as the DCM locked signal and incorrectly indicate that the DCM has been affected by an SEU, when in reality it has not. To counter this potential problem, the FPGA design engineers came up with a method of using counters to monitor the DCMs and make sure they are locked to the correct frequency. Should the counter freeze or increment at a different frequency, it is indicated by comparing it with the other counters, allowing for an FPGA reconfiguration if required.

FUTURE DEVELOPMENTS

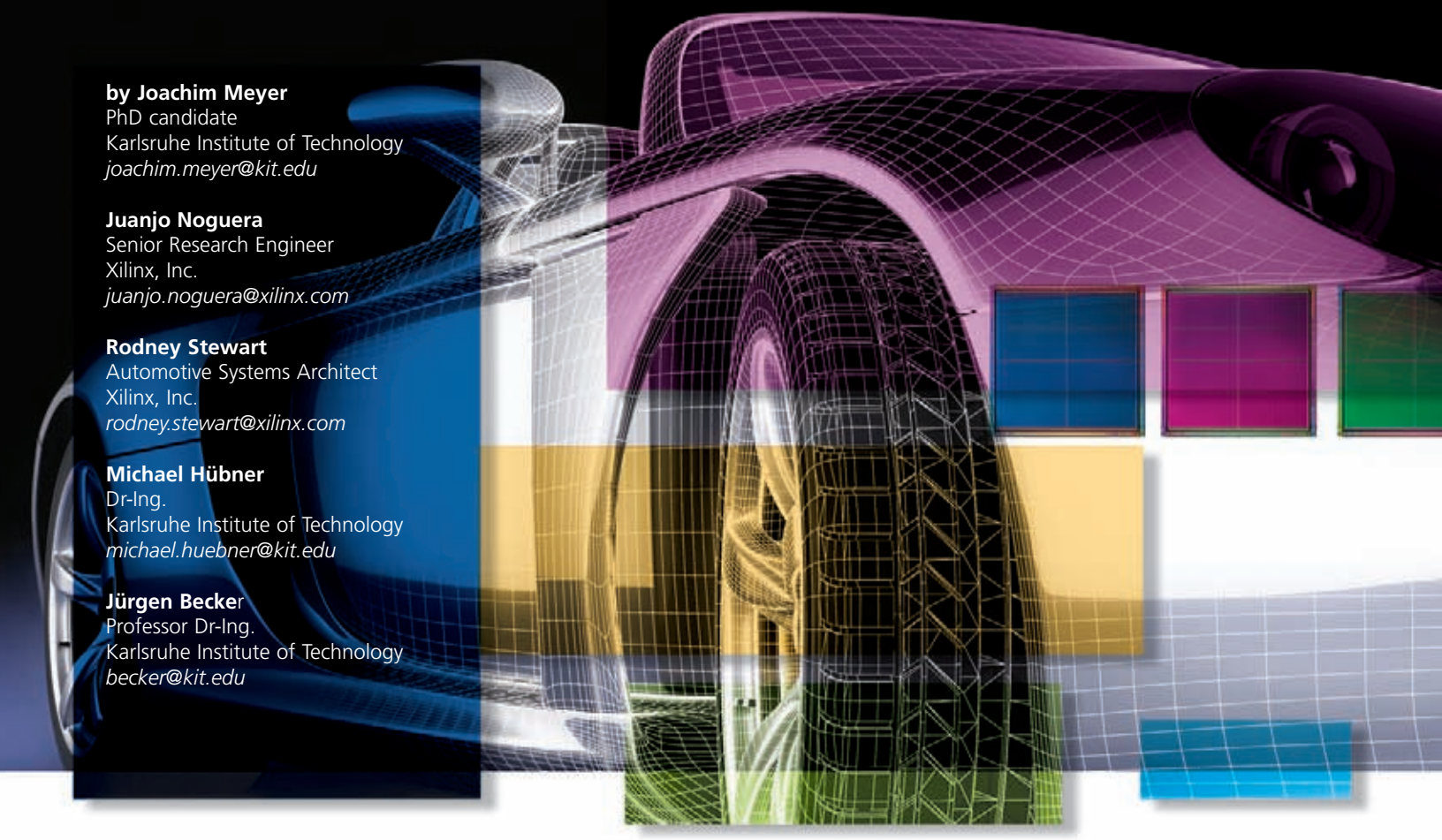
The U.K. Space Agency will launch UKube1 in January 2012. The CubeSat will provide data on both of our experiments for at least the mission lifetime of one year. This mission will demonstrate the suitability of high-performance FPGAs for use in LEO missions and microsatellite architectures as well as hopefully allowing the use of high-performance FPGAs in other missions of longer duration.

The UKube1 is expected to be just the first of a national CubeSat program in the U.K. The architecture developed for UKube1 lends itself to adaptation for use on future missions to gain further understanding of, and experience in, using large, high-performance FPGAs in orbit. Potential adaptations for the next mission would include a demonstration of in-orbit partial reconfiguration and in-orbit readback and verification of the device configuration. Unfortunately, due to the time scales involved in the development of UKube1, these experiments could not be incorporated on this maiden voyage.

The UKube1 architecture using the Xilinx Virtex-4 family of devices also lends itself to evolution into a system-on-chip-based controller that could serve as a micro- or nanosatellite mission controller.

Fast Startup for Xilinx FPGAs

Advanced research work demonstrates a new, two-step configuration method that meets the tight startup timing specs of automotive and PCIe applications.



by Joachim Meyer
PhD candidate
Karlsruhe Institute of Technology
joachim.meyer@kit.edu

Juanjo Noguera
Senior Research Engineer
Xilinx, Inc.
juanjo.noguera@xilinx.com

Rodney Stewart
Automotive Systems Architect
Xilinx, Inc.
rodney.stewart@xilinx.com

Michael Hübner
Dr-Ing.
Karlsruhe Institute of Technology
michael.huebner@kit.edu

Jürgen Becker
Professor Dr-Ing.
Karlsruhe Institute of Technology
becker@kit.edu

In many modern applications, embedded systems have to meet extremely tight timing specifications. One of these timing requirements is the startup time—that is, the time it takes for the electronic system to be operative after power-up. Examples of electronic systems with a tight startup-timing specification are PCI Express® products or CAN-based electronic control units (ECUs) in automotive applications.

Just 100 milliseconds after powering up a standard PCIe® system, the root complex of the system starts scanning the bus in order to discover the topology and, in so doing, initiate configuration transactions. If a PCIe device is not ready to respond to the configuration requests, the root complex does not discover it and treats it as nonexistent. The device won't be able to participate on the PCIe bus system. [1]

The automotive scenario is similar. In CAN-based networks, ECUs go into a sleep mode where they shut down and disconnect from their power supply. Only a small bit of circuitry remains alert to detect wake-up signals. Whenever a wake-up event occurs, ECUs reconnect to power and start booting. While it is acceptable to miss any messages during the first 100 ms after the wake-up event, afterward all ECUs must be fully operational on the network (for example, CAN).

Advanced development work between Xilinx Automotive, Xilinx Research Labs and the Karlsruhe Institute of Technology in Karlsruhe, Germany is addressing this problem with a two-step FPGA configuration methodology.

The technology trends in the semiconductor industry have enabled FPGA manufacturers to significantly increase the resources in their devices. But the bitstream size grows proportionally, and so does the time it takes to configure the device. Therefore,

even with midsize FPGAs, it is not possible to meet demanding startup-timing specifications using low-cost configuration solutions. Figure 1 shows the configuration time for different Xilinx® Spartan®-6 FPGAs using the low-cost SPI/Quad-SPI configuration interface. Even when using a fast configuration solution (namely, Quad-SPI running at a 40-MHz configuration clock), only the small FPGAs meet the 100-ms startup-timing specification. For Xilinx Virtex®-6 devices the results look even more challenging, since these devices provide an abundance of FPGA resources.

To address this challenge, Fast Startup configures the FPGA in two steps, rather than a single (monolithic) full-device configuration. In this novel approach, the strategy is to load only the timing-critical modules at power-up using the first high-priority bitstream, and then load the non-timing-critical modules afterward. This technique minimizes the initial configuration data and, thus, minimizes the FPGA startup time for the timing-critical design.

FAST STARTUP VS. PARTIAL RECONFIGURATION

Fast Startup enables an FPGA design to start critical modules in the design as fast as possible, much faster than in a standard full-configuration technique. [2] Although Fast Startup fundamentally makes use of partial reconfiguration, it differs from the traditional concepts of this technique. Partial reconfiguration intends a full design to be used as an initial configuration that can be modified during runtime. By contrast, Fast Startup already uses an initial partial bitstream in order to configure just one specific (small) region of the FPGA at power-up. The first configuration contains only those parts of the full FPGA design that must be up and running quickly. The remainder is configured later, during runtime, using partial reconfiguration. Figure 2 illustrates this sequential concept.

TOOL FLOW OVERVIEW

The tool flow for Fast Startup relies on the design preservation flow to create the partial bitstreams for both the timing-critical and the non-timing-critical subsystems.

The design preservation flow divides the FPGA design into logical modules called partitions. Partitions form hierarchical boundaries, isolating the modules inside from the other components of your design. A partition, once implemented (that is, placed and routed), can be imported by other implementation runs in order to implement the modules of the partition in exactly the same way in each instance. [3]

Therefore, the first step in using the Fast Startup technique is to separate the complete FPGA design into two sections: a high-priority partition containing the timing-critical subsystem and a low-priority partition for the rest of the components.

IMPLEMENTING THE HIGH-PRIORITY PARTITION

To obtain a partial bitstream for the high-priority partition that is as small as possible, there are some general design considerations to take into account. First of all, the partition must include only components that are timing-critical or that the system needs to perform the partial reconfiguration of the low-priority section (for example, the ICAP). The key to obtaining a small initial partial bitstream is to use an area as small as possible to implement the high-priority partition. That means you must constrain this partition to an appropriate region in the FPGA.

In order to find a good physical location on the FPGA, this area should provide just the right amount of resources you need for the design. Accessing resources located outside of this area is possible but discouraged—though generally will be

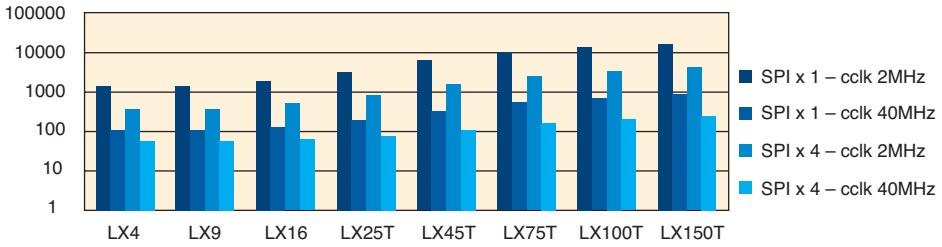


Figure 1 – Logarithmic illustration of calculated Spartan-6 configuration times (worst-case calculation)

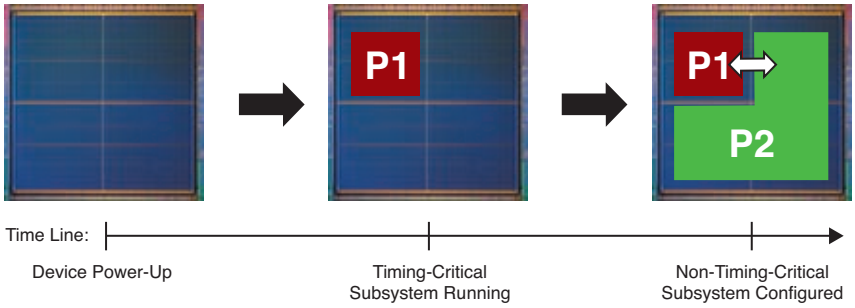


Figure 2 – Fast Startup concept: a sequential configuration

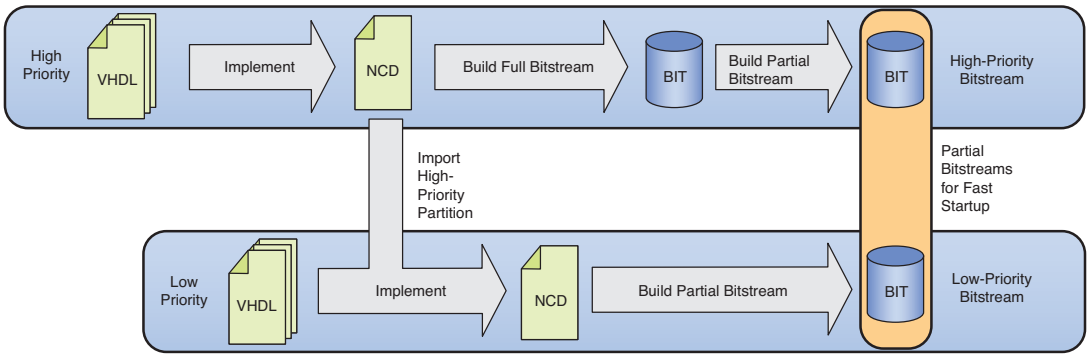


Figure 3 – Fast Startup tool flow

inevitable in the case of I/O pins. When looking for an appropriate area, also keep in mind that this FPGA region might block resources for the non-timing-critical part of the FPGA design.

When you have partitioned the FPGA and found appropriate areas for the partitions, the next step is to implement the high-priority partition, using an empty (black box) low-priority partition. The resulting bitstream contains a lot of configuration frames for resources that are not used. You can remove those frames in order to get a valid partial bitstream for the initial configuration of the high-priority partition. [4]

IMPLEMENTING THE LOW-PRIORITY PARTITION

To create the partial bitstream for the low-priority partition, first you have to create an implementation of the full FPGA design containing both partitions, the high and the low priority. Import the high-priority partition from the previous implementation in order to make sure it is implemented in the same way as before.

For Virtex-6, you will use the partial reconfiguration (PR) flow for all the described implementations. As a result, the partial bitstream for the low-priority partition is obtained automatically. Since the Spartan-6 device family does not support the PR flow, we used the BitGen option for difference-based partial reconfiguration to obtain the partial bitstream for the low-priority partition when implementing Fast Startup for our Spartan-6 design. [5] Figure 3 gives a high-level overview of the tool flow.

EXPERIMENTS AND RESULTS

For verifying the Fast Startup configuration technique in hardware, our team implemented it on a Virtex-6 ML605 board as well as on a Spartan-6 SP605 board.

The application scenario for a Virtex-6 implementation derives from the video domain. Whenever users power

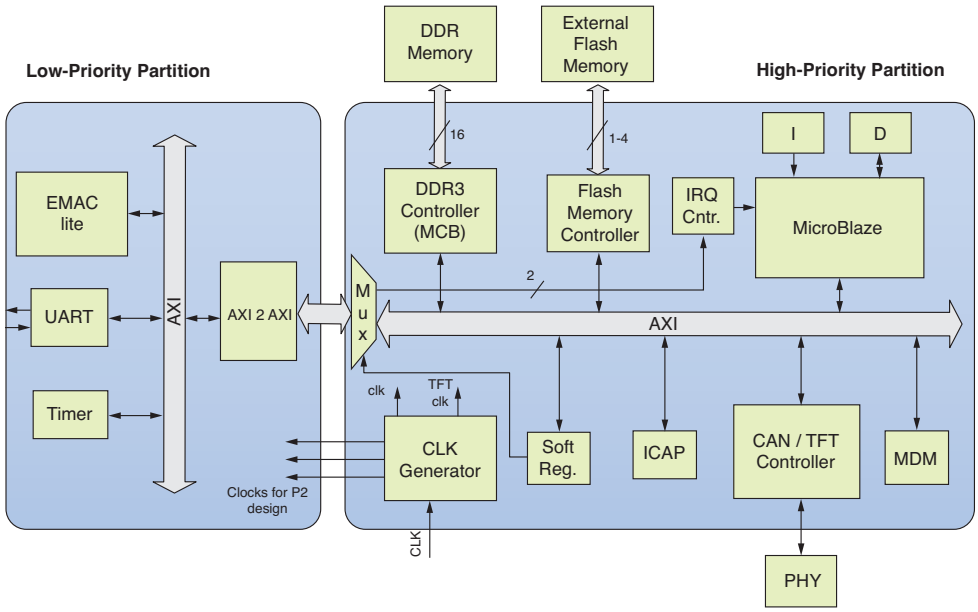


Figure 4 – Basic block diagram of Virtex-6 and Spartan-6 demonstrator (the Virtex-6 includes the TFT block; the Spartan-6 includes the CAN block only)

up a video system they want to see an immediate response, and not wait several seconds. Therefore, in the system illustrated in Figure 4, a high-priority processor subsystem equipped with a TFT controller brings up a TFT screen as quickly as possible. For additional, low-priority applications, the second design provides access to an Ethernet core, UART and hardware timer.

For this demonstrator we used an external flash memory with BPI as our configuration interface. As soon as the initial high-priority bitstream configures the processor subsystem, software running out of BRAM initializes the TFT controller and writes data into the frame buffer located in the DDR memory. This ensures the screen will show up on the TFT as fast as possible at startup. Afterward, the second bitstream is read from the BPI flash memory and the low-priority partition is configured to enable the processor subsystem to run other applications, such as a Web server.

For easy expansion and a clean separation of the two partitions, we used an AXI-to-AXI bridge. This also minimized the nets crossing the border of the two design partitions. The low-priority partition shares its system clock with the high-priority partition.

Resource Type	Partition			
	High Priority	%	Low Priority	%
Flip-Flops	8,849	2.9	1,968	0.7
Lookup Tables	7,039	4.7	2,197	1.5
I/Os	135	22.5	20	3.3
RAMB36s	34	8.2	2	0.5

Table 1 – Occupied FPGA resources for XC6VLX240T

XC6VLX240T	Configuration Technique		
Configuration Interface	Traditional 8.9 MB	Compressed 2.0 MB	Fast Startup 1.4 MB
BPIx16 CR2	1,740 ms	389 ms	278 ms
BPIx16 CR10	450 ms	112 ms	84.4 ms

Table 2 – Measured configuration times for Virtex-6 video design

Table 1 shows the FPGA resource utilization while Table 2 presents configuration times for the traditional startup, the startup with a compressed bitstream of the high-priority partition only [6] and the Fast Startup configuration technique. In all cases, the configuration interface was BPIx16, while the applied configuration rate—an option to determine the target configuration clock frequency—was 2 and 10 MHz. We measured the data by using an oscilloscope to capture the "init" and the "done" signals of the FPGA.

The "Compressed" column in Table 2 represents a compressed bitstream of the high-priority partition only. A compressed bitstream of the full FPGA design containing both partitions would be 3.1 Mbytes.

SPARTAN-6 AUTOMOTIVE ECU DESIGN

To verify the Fast Startup technique for Spartan-6, we chose the ECU application scenario from the automotive domain. Whenever you find an FPGA in an automotive electronic control unit, usually it is used by the

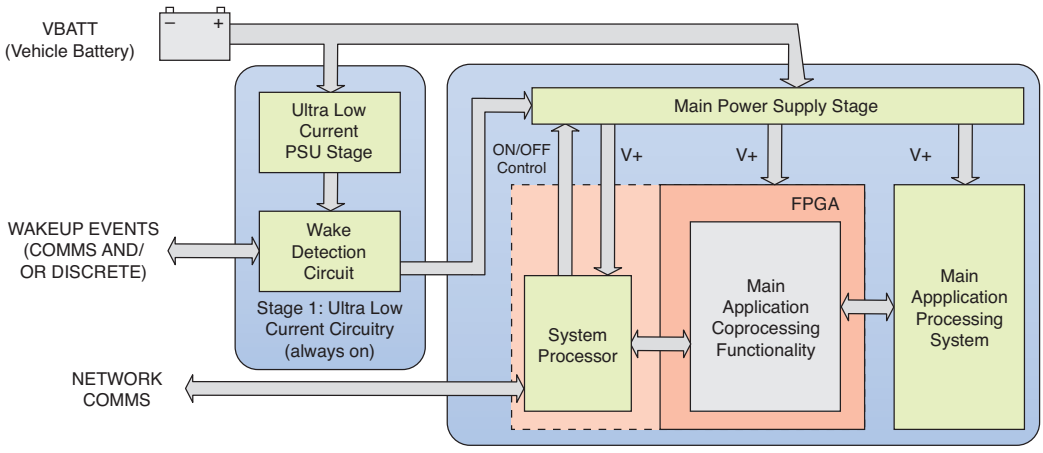


Figure 5 – FPGA use in today's automotive ECUs and with the processor incorporated into the FPGA (dotted lines)

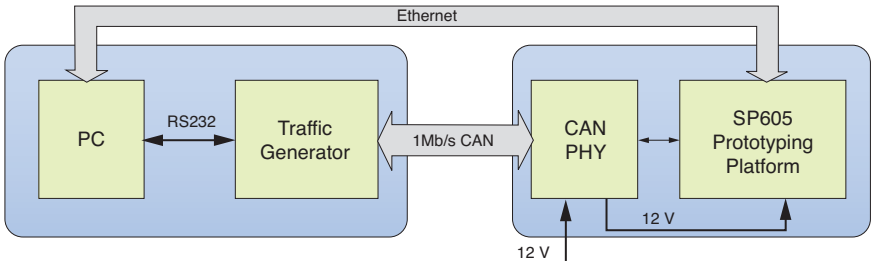


Figure 6 – Measurement setup for the automotive ECU

main application processing unit of the ECU only (see Figure 5). Our goal was to implement a design that also put the system processor inside the FPGA. By doing so, we could eliminate the need for an external processor, thereby reducing overall system cost, complexity, space and power consumption.

SYSTEM PARTITIONING

For this scenario, the partitioning of the system is obvious. We separated our ECU design into a system processor part as our high-priority partition and an application-processing part as the low-priority partition.

This design had a lot of similarities to the Virtex-6 design, but instead of BPI, we used SPI as an interface to the external flash memory and instead of a TFT controller, a CAN controller was necessary. Upon power-up, the system controller has only a limited amount of time to boot and be ready to process the first communication data. For ECUs using the CAN bus for communications this boot time limit is typically 100 ms. With traditional configuration techniques it is hard to beat this tight timing specification using a big Spartan-6 with a low-cost configuration interface like SPI or Quad-SPI. But using a faster and therefore more expensive configuration interface is unacceptable in the automotive domain.

MEASUREMENT SETUP

For the SP605 automotive ECU demonstrator we performed measurements in our labs; Figure 6 presents the measurement setup. On the left side is an X1500 automotive platform based on a Spartan-3 implementing a traffic generator for the CAN bus, which is able to send and receive CAN messages and measure time between them using hardware timers. On the right side is the target platform, which is not connected directly to the CAN bus but uses the CAN transceiver from an additional custom board. Besides providing a CAN PHY, this custom board also controls the power supply of the target board.

The procedure to measure the configuration time starts with the traffic generator in idle status, the CAN transceiver on the CAN PHY board in sleep mode and therefore the SP605 disconnected from power. In the next step, the traffic generator starts a hardware timer and sends a CAN message. Recognizing the activity on the CAN bus, the CAN PHY awakens and reconnects the SP605 to the power supply. The FPGA then starts to load the initial bitstream from SPI flash.

Because there is no receiver acknowledging the message sent by the traffic generator, the message is resent immediately until the FPGA has finished its configuration and also configured the CAN core with the valid baud rate. Whenever the CAN core of the Spartan-6 design acknowledges the message, the CAN core of the traffic generator triggers an interrupt, which stops the hardware timer. This timer is now holding the boot time for the SP605 design. Measurements, which included an additional hardware timer inside the SP605 design, have shown that when executing the software to configure

Resource Type	Partition			
	High Priority	%	Low Priority	%
Flip-Flops	3,480	6%	1,941	4%
Lookup Tables	3,507	13%	1,843	7%
I/Os	58	20%	20	7%
RAMBs	12	10%	2	2%

Table 3 – Occupied FPGA resources in Spartan-6 design

Configuration Interface	Configuration Technique		
	Traditional 1,450 KB	Compressed 920 KB	Fast Startup 314 KB
SPIx1 CR2	5,297 ms	3,382 ms	1,157 ms
SPIx1 CR26	292 ms	196 ms	85 ms
SPIx2 CR2	2,671 ms	1,699 ms	596 ms
SPIx2 CR26	161 ms	113 ms	58 ms
SPIx4 CR2	1,348 ms	872 ms	311 ms
SPIx4 CR26	97 ms	73 ms	45 ms

Table 4 – Measured Spartan-6 configuration times

the CAN core from internal BRAM memory, the software startup time was negligible.

Table 3 presents the FPGA resource consumption for each partition. The percentage information refers to the total amount of available resources of the used XC6S45LXT device.

Table 4 shows the results of the configuration time measurements. For these measurements, we implemented and compared a standard bitstream of the full design, a compressed bitstream of the full design and the Fast Startup technique using a partial initial bitstream. The table lists the configuration times for different SPI bus widths and different configuration rate (CR) settings. As expected, the configuration times are proportional to the bitstream sizes. Because using a fast configuration clock does not affect the housecleaning process, the ratio in percentage changes for high-CR settings.

VERIFIED IN HARDWARE

The advanced configuration technique that we developed might be called prioritized FPGA startup, since it configures the device in two steps. Such a technique is essential to address the challenge of increasing configuration time in modern FPGAs and to enable their use in many modern applications, like PCI Express or CAN-based automotive systems.

Besides proposing the high-priority initial configuration technique, we verified it in hardware. We have used and tested the tool flow and methods for Fast Startup to implement a CAN-based automotive ECU on a Spartan-6 evaluation board (the SP605) as well as a video design on a Virtex-6 prototyping board. By using this novel approach, we decreased the initial bitstream size, and hence, achieved a configuration time improvement of up to 84 percent when compared with a standard full-configuration solution.

Xilinx will support the Fast Startup concept for PCI Express applications in the software for the new 7 series FPGAs, with improved implementation techniques to simplify its use. In the 7 series, the new two-stage bitstream method is the simplest and least expensive technique to implement. The user directs the implementation tools to create a two-stage bitstream via a simple software switch when building the FPGA design. The first stage of the bitstream contains just the configuration frames necessary to configure the timing-critical modules. When configured, an FPGA STARTUP sequence occurs and the critical blocks becomes active, thus easily satisfying the 100-ms timing requirement. While the timing-critical modules are working (e.g., PCI Express enumeration/configuration system process is occurring), the remainder of the FPGA configuration gets loaded. The two-stage bitstream method can use an inexpensive flash device to hold the bitstreams.

References

[1] PCI Express Base Specification, Rev. 1.1, PCI-SIG, March 2005

[2] M. Huebner, J. Meyer, O. Sander, L. Braun, J. Becker, J. Noguera and R. Stewart, "Fast sequential FPGA startup based on partial and dynamic reconfiguration," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2010

[3] Hierarchical Design Methodology Guide, UG748, v12.1, Xilinx, May 2010

[4] B. Sellers, J. Heiner, M. Wirthlin and J. Kalb, "Bitstream compression through frame removal and partial reconfiguration," International Conference on Field Programmable Logic and Applications (FPL), September 2009

[5] J. Meyer, J. Noguera, M. Huebner, L. Braun, O. Sander, R. Mateos Gil, R. Stewart, J. Becker, "Fast Startup for Spartan-6 FPGAs using dynamic partial reconfiguration," Design, Automation and Test in Europe (DATE '11), 2011

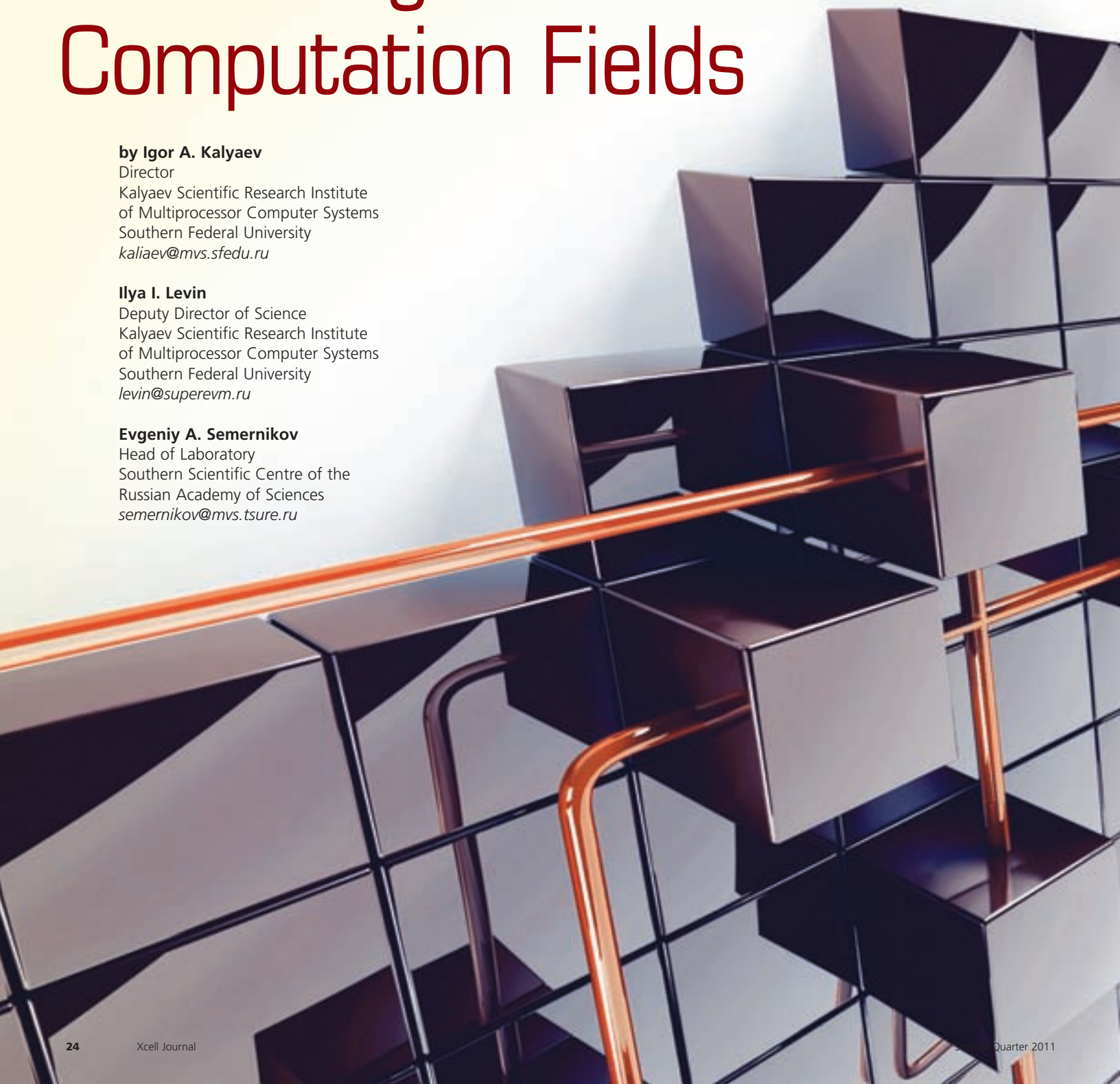
[6] "Fast Configuration of PCI Express Technology through Partial Reconfiguration," XAPP883, v1.0, Xilinx, November 2010, http://www.xilinx.com/support/documentation/application_notes/xapp883_Fast_Config_PCIe.pdf.

Reconfigurable System Uses Large FPGA Computation Fields

by Igor A. Kalyaev
Director
Kalyaev Scientific Research Institute
of Multiprocessor Computer Systems
Southern Federal University
kaliaev@mvs.sfedu.ru

Ilya I. Levin
Deputy Director of Science
Kalyaev Scientific Research Institute
of Multiprocessor Computer Systems
Southern Federal University
levin@superevm.ru

Evgeniy A. Semernikov
Head of Laboratory
Southern Scientific Centre of the
Russian Academy of Sciences
semernikov@mvs.tsure.ru



By interconnecting multiple Xilinx FPGAs, designers can build a new kind of supercomputer and tailor it for jobs in a wide range of application areas.

Over the last two decades, many supercomputer architectures have included both microprocessors, which serve as the main brains of the system, and FPGAs, which typically offload some of the compute tasks from those CPUs. In fact, this pairing of FPGAs with standalone microprocessors has proven to be a winning combination for many generations of supercomputers.

But recently, our group of scientists at the Kalyaev Scientific Research Institute of Multiprocessor Computer Systems at Southern Federal University (Taganrog, Russia) designed a reconfigurable computer system (RCS) based largely on Xilinx® FPGAs. By interconnecting multiple FPGAs, we can create special-purpose computational structures that are well suited to solving problems in a broad number of application areas.

An FPGA-based RCS has several interesting design considerations and advantages. At its heart are several FPGAs that we've united to form a single computational system. For our reconfigurable systems we use an orthogonal communication system, placing FPGAs in the rectangular lattice points. Direct links interconnect adjacent FPGAs.

In addition, we create a parallel-pipeline computational structure in the FPGA computation field by using computer-aided design tools and tools for structural programming. The architecture of this parallel-pipeline computational structure is similar to the task structure, and this similarity ensures the highest performance of the RCS.

If hardware limitations do not allow mapping of the entire informational graph of a task we are trying to perform or a problem we are attempting to solve, then the system will automatically split the graph into a number of subgraphs. The system's

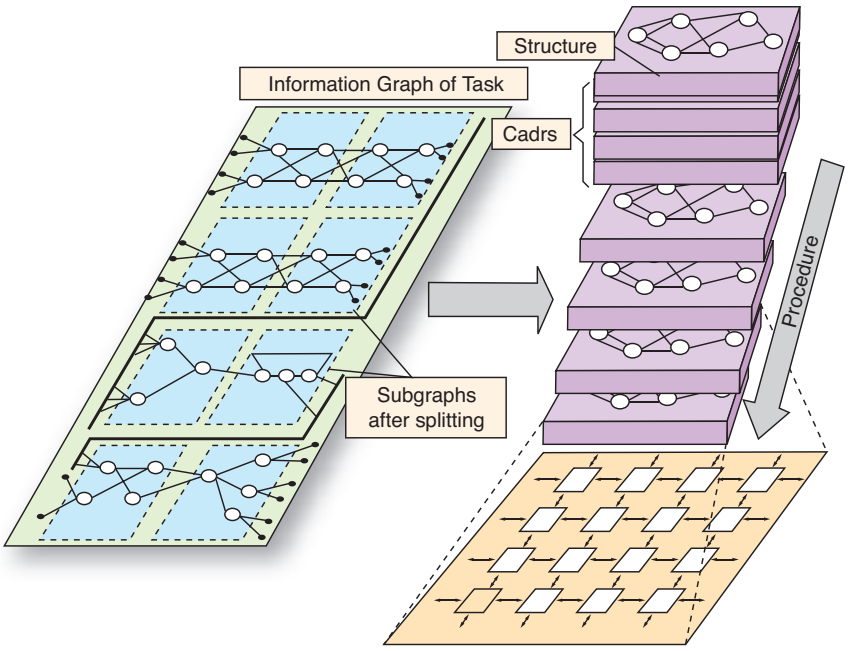


Figure 1 – Computational structures flow of the reconfigurable computer system (RCS)

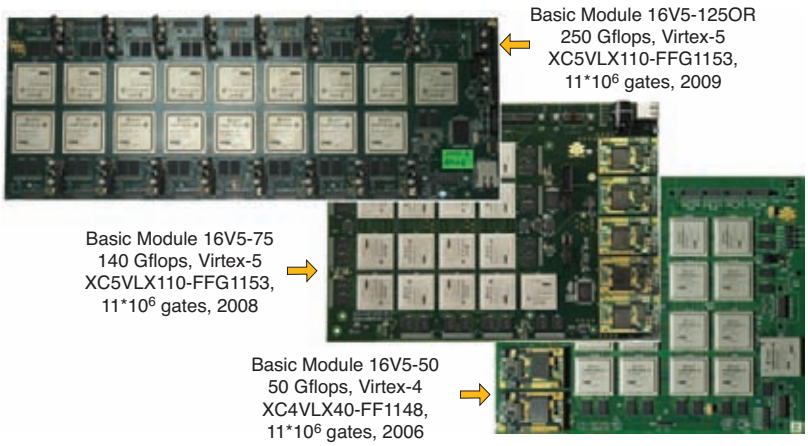


Figure 2 – Several basic RCS modules, all built around Virtex FPGAs

computation field sees each of these subgraphs as individual computational structures, organizes the subgraphs sequentially and solves them sequentially. Figure 1 illustrates the computational structures flow of the RCS and the procedures of data input and output. As you can see, the size of each subgraph is proportional to the size of an FPGA field. Tools we have developed for structural programming automatically split a problem into a smaller number of larger subgraphs and map them into hardware for efficient solving, thus dramatically speeding RCS performance.

In more detail, the system interprets the informational graph of a given task or problem as a sequence of isomorphic basic subgraphs, each of which can be independent or dependent (that is, information-related) on one another. The system transforms each informational subgraph into a special computational structure called "cadr." Essentially, a cadr is a hardwired task subgraph with an operands flow running through it. Each group of operands (or results) corresponds to input (or output) nodes of the specified subgraph in the sequence. The special parallel program (or "procedure," as we call it) executes the sequence of cadrs. Only one parallel program is needed for the whole RCS.

We designed our RCS for high-performance computing in a modular architecture. The RCS consists of basic modules that host fragments of the computation field and have connections with other modules and auxiliary subsystems. Figure 2 shows several basic modules designed at the Kalyaev Scientific Research Institute of Multiprocessor Computer Systems at Southern Federal University (SRI MCS SFU).

The FPGAs in the basic modules communicate via LVDS channels running at a frequency of 1.2 GHz. All modules in an RCS communicate by

We designed our RCS for high-performance computing in a modular architecture. It consists of basic modules that host fragments of the computation field and have connections with other modules and auxiliary subsystems.

means of these channels. Each basic module also includes discrete distributed memory units along with a basic module controller to perform resource management, including loading fragments of parallel applications into the distributed memory controllers and handling data input and output.

Our 16V5-75 is the main module in our RCS family. The RCS series consists of systems with various performance levels ranging from 50 gigaflops (using 16 Xilinx Virtex®-5 FPGAs) to 6 teraflops (1,280 Virtex-5 FGPAs). Our highest-end model holds five ST-1R racks, each rack containing four 6U blocks of RCS-0.2-CB and each block consisting of four basic 16V5-75 modules. The peak performance of the basic module is up to 200 gigaflops for single-precision data. So, the computation field of the ST-1R rack contains 256 Virtex-5 FPGAs (XC5VLX110) connected by high-speed LVDS (Figure 3).

Our Orion RCS (Figure 4), meanwhile, boasts performance of 20 teraflops. It consists of 1,536 Virtex-5 FPGAs placed in a 19-inch rack. The main element of the system is a basic module 16V5-125OR with performance of 250 gigaflops. Basic modules are connected into a single computational resource via high-speed LVDS. Four basic 16V5-125OR modules are connected within a 1U block. In contrast to the basic module 16V5-75, which has only four of 16 FPGAs that can be connected with other basic modules, all 16 FPGAs of the basic module 16V5-125OR have LVDS for computation field expandability. This provides high data-exchange rates between the basic modules within the Orion block.

Table 1 shows the real performance of the block RCS-0.2-CB and Orion at solving various tasks: digital signal processing, linear algebra and mathematical physics.

PROGRAMMING THE RCS

RCS programming significantly differs from programming traditional cluster-

Tasks	Specific performance (Gflops/dm³)	Real performance at task solving (Gflops)		
		DSP	Linear algebra	Mathematical physics
RCS-0.2-CB	16.7	647.7	423.2	535.2
Orion	64.9	809.6	528.7	669.0

Table 1 – Performance of the block RCS-0.2-CB and the computing block Orion

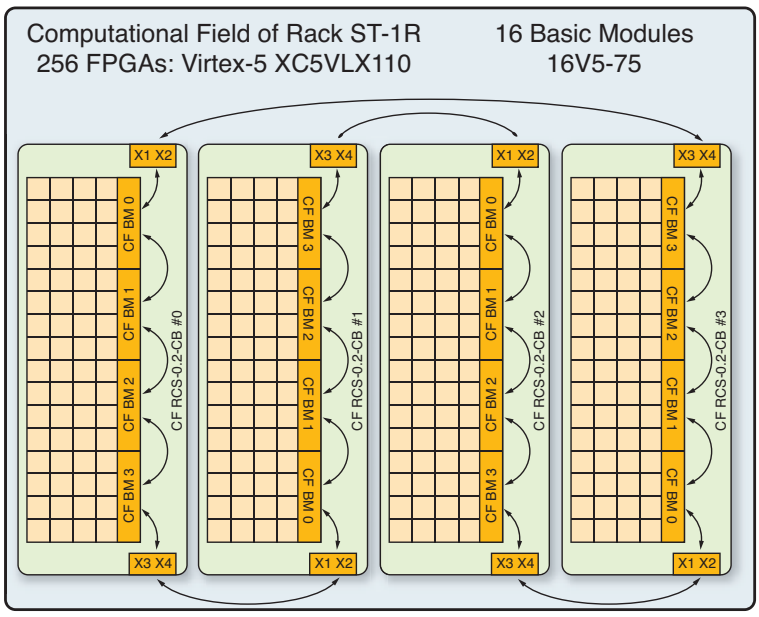


Figure 3 – Computation field of the ST-1R rack containing 256 Virtex-5 FPGAs connected by high-speed LVDS

We sought to make the RCS easier to program by developing a special software suite. The suite allows users to program the system without requiring any specialized knowledge of FPGA and hardware design.

based supercomputer architectures. Traditional supercomputer architectures have fixed hardware circuitry and can only be programmed at a software level, traditionally by software engineers. An FPGA-based RCS has programmable hardware as well software. Therefore, we divide the RCS into two kinds of programming: structural and procedural. Structural programming is the task of mapping the informational graph into the RCS structure's computational field (programming the FPGAs). Procedural programming is traditional programming in which users describe the organization of computational processes in the system. Typically, software engineers have struggled when it comes to programming FPGAs, and so to competently do the job they must have some hardware design skills.

Knowing this, we sought to make the RCS easier to program by developing a special software suite. The suite allows users to program the system without requiring any specialized knowledge of FPGA and hardware design. Using this suite, the software developer will get the solution two to three times faster than an engineer who designs a circuit solution of the task and uses traditional methods and tools.

Based on the development environment Fire!Constructor for circuit design, the suite includes a high-level programming language called COLAMO and the Argus integrated development environment. Users create programs in COLAMO and a built-in compiler/translator/synthesizer automatically creates code for both the structural and procedural components, with the help of an IP library of computational units and interfaces.

Libraries of basic module descriptions, unit descriptions and a description of the entire RCS (RCS passport) make it possible to port parallel applications to the RCS from various architecture platforms. The COLAMO translator breaks the system description down into four data categories: control, procedural, stream and structural.

The control data is translated into Pascal and is executed in master controllers, which, depending on the RCS hardware platform, are contained in basic modules or computing blocks. The control data configures the FPGAs

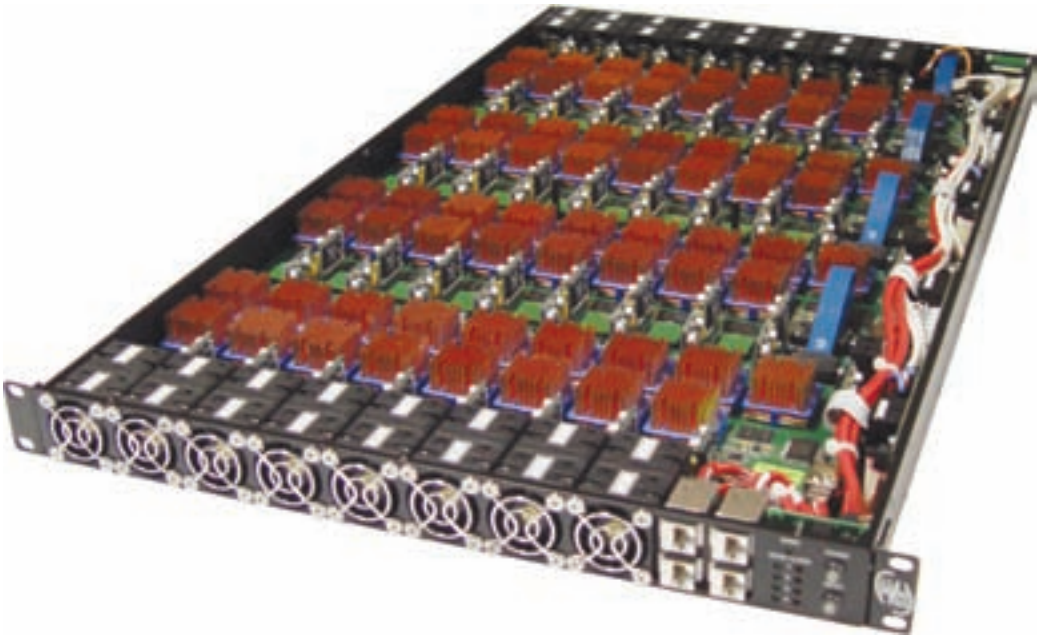


Figure 4 – The 20-Tflops Orion computing block

for the computational tasks at hand, programs distributed memory and maps the data exchange between blocks and basic modules.

Meanwhile, the procedural and stream components are translated into the Argus assembler language. They are executed by distributed memory controllers, which specify the cadrs' execution and create parallel data flows in the cadrs' computational structures.

The most interesting feature of the suite is the synthesis tool called Fire!Constructor that we developed to help users program the FPGAs that lie at the heart of the RCS. Using the IP library, Fire!Constructor generates a VHDL description (*.vhd file) for each FPGA of the RCS. The Xilinx ISE® suite uses these files to create configuration files (*.bit) for all the FPGAs.

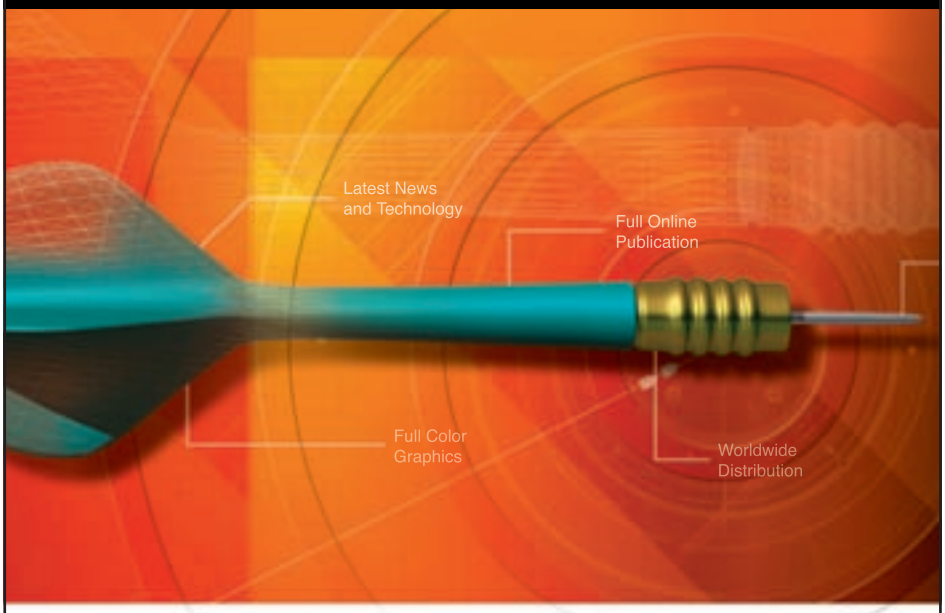
EFFECTIVE PROBLEM SOLVING

Owing to the principle of structural procedural calculations, we can effectively solve various problems on the RCS. The most suitable are the problems that have an invariable information graph of the algorithm and variable sets of input data. For instance, problems of drug design, digital signal processing, mathematical simulation and the like lend themselves to RCS solution.

A spin-off company called Scientific Research Centre of Supercomputers and Neurocomputers (Taganrog, Russia) delivers these reconfigurable systems, manufacturing each RCS according to preorder specifications. The company can manufacture about 100 product units per month. Performance of our RCS, based on Virtex-5 and Virtex-6 FPGAs, is from 100 Gflops to 10 Tflops.

Overall, FPGA-based reconfigurable computer systems offer unique advantages for solving tasks of various classes. With an RCS, users can tailor computation to quickly tackle them. To learn more about our RCS, visit *http://parallel.ru/index_eng.html*.

GET ON TARGET



IS YOUR MARKETING MESSAGE REACHING THE RIGHT PEOPLE?

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

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

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

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

Xcell PUBLICATIONS

See all the new publications on our website.

www.xilinx.com/xcell

Xilinx FPGAs Beam Up Next-Gen Radio Astronomy

Researchers in Australia are using Virtex-6 FPGAs to economically meet the demanding requirements of an advanced telescope called ASKAP.

by Grant Hampson, PhD
ASKAP Digital Systems Lead Engineer
CSIRO Astronomy and Space Science
Grant.Hampson@csiro.au

John Tuthill, PhD
Hardware and Firmware Engineer
CSIRO Astronomy and Space Science
John.Tuthill@csiro.au

Andrew Brown
Hardware and Firmware Engineer
CSIRO Astronomy and Space Science
Andrew.Brown@csiro.au

Stephan Neuhold
Firmware Engineer
CSIRO Astronomy and Space Science
Stephan.Neuhold@csiro.au

Timothy Bateman
Firmware Engineer
CSIRO Astronomy and Space Science
Tim.Bateman@csiro.au

John Bunton, PhD
ASKAP Project Engineer
CSIRO Information and Communication Technologies Centre
John.Bunton@csiro.au

As the largest and most sensitive telescope ever envisioned, the Square Kilometre Array (SKA), a radio telescope under development by a consortium of 20 countries, will revolutionize our knowledge of the universe through large-area imaging from 70 MHz to a 10 GHz when it is deployed in the middle of the next decade. On the path to the development of the SKA, the Australian SKA Pathfinder (ASKAP) is at a key SKA science frequency band of 700 MHz to 1.8 GHz. ASKAP seeks to, achieve instantaneous wide-area imaging through the development and deployment of phased-array feed systems on parabolic reflectors. This large field of view makes ASKAP an unprecedented survey telescope poised to achieve substantial advances in SKA key science.

The Commonwealth Scientific and Industrial Research Organisation, or CSIRO, is an Australian government-funded research agency. A division called CSIRO Astronomy and Space Science (CASS) is undertaking construction of the ASKAP (see *www.csiro.au/org/CASS.html*). The combination of location, technological innovation and scientific program will ensure that ASKAP is a world-leading radio astronomy facility, closely aligned with the scientific and technical direction of the SKA. CSIRO is also involved in the development of the SKA, which is anticipated to be operational in 2024.

Currently under construction in outback Western Australia at the Murchison Radio Astronomy Observatory (MRO), ASKAP will consist of an array of thirty-six 12-meter dishes (Figure 1), each employing advanced phased-array feeds (PAFs) to provide a wide field of view: 30 square degrees. As opposed to conventional antenna optics, PAFs allow for multiple configurable beam patterns using digital

electronics to implement beamformers. ASKAP's location in the Southern Hemisphere is advantageous for astronomy. Combined with a very large radio quiet zone (an area where radio frequency emissions are managed), the result will be a radio telescope with unprecedented performance.

ASKAP's wide field of view and broad bandwidth will enable astronomers to survey the sky very quickly, but present a signal-processing challenge two orders of magnitude greater than that facing existing telescopes. To satisfy performance demands of approximately 2 peta operations/second with 100-Tbit/s communications, Xilinx® FPGAs deliver a well-balanced mix of these features. FPGAs provide an economical solution for both power consumption and capital cost compared with other alternative solutions, such as high-performance computing (HPC).

Figure 1 – The first ASKAP reflector surface is being installed in Western Australia; researchers will later add a phased-array feed at the focus of the antenna.

STATEMENT OF THE PROBLEM

There are two major problems to solve with ASKAP: huge compute capacity and very high-bandwidth data communications. It is easy to find a solution to one, but not to both challenges simultaneously. The Xilinx Virtex®-6 family of FPGAs provides an excellent balance of I/O bandwidth to computational capacity and is well suited to the repetitive parallel operations in filter banks and beamformers.

The ASKAP design is also constrained by a number of other factors that influence the implementation. Power costs at the remote location and the climate provide extra challenges for cooling instrumentation. Radio astronomy instrumentation also requires an operating environment very low in electromagnetic interference (EMI)—telescopes are sensitive to extremely low levels of radiation. The technological advances in Virtex-6 FPGAs that reduce operating voltages and I/O swings help reduce EMI further. A second advantage in using Virtex-6 FPGAs is that the resulting lower-power systems require less cooling, a double saving for an astronomy instrument that operates 24/7.

SYSTEM DESCRIPTION

We have taken the divide-and-conquer approach to determine a cost-effective on-time solution. Figure 2 illustrates that there are two digital signal-processing locations—antenna-based electronics and a central processing site. The antenna-based electronics sample and process the outputs of 188 PAF receptors plus four calibration signals. It transports the data over optical fiber to the central processing site.

In the central site, beamformers for each antenna form up to 36 dual-polarization beams on the sky. The outputs of all the antenna beamformers then connect to a correlator, which forms complex-valued visibilities that are used to create images of the sky.

Within each antenna are a number of receiving cards that process a small number of PAF receptors or, in the case of the beamformer, a small frequency slice of the overall input bandwidth. This technique divides the data communications problem while also reducing the cost of each processing card.

DIGITAL RECEIVER DESIGN

The digital receiver subsystem contains hardware to sample the intermediate-frequency (IF) signals from the PAF and firmware to divide the IF bandwidth into 1-MHz bands and transport the resulting data to the beamformer at the central site. The subsystem is divided into four 3U x 160-mm Eurocard subracks, each consisting of 12 digital receiver cards, a power card and a control interface card. The digital receiver system is shown in Figure 3.

The power card is responsible for converting a 48-VDC input to the voltages required by the digital receiver cards. The control card contains a 1-Gbit Ethernet port to allow local control and monitoring over UDP packets. We implemented the relatively simple request/response control firmware on a low-cost Virtex-5 XC5VLX30T-1FF665C FPGA. Control and monitoring links fan out from the control card to the digital receiver cards via the backplane.

Each digital receiver card samples four analog IF receptor signals from the PAF. The card consists of three major sections: analog-to-digital converters, a Virtex-6 LX130T FPGA and 10-Gbps optical data transmission components. Sampling is performed using two dual-input 8-bit ADCs from the semiconductor company e2v, sampling at a rate of 768 Msamples/s and providing a total bandwidth of 384 MHz. The Xilinx FPGA receives data from the ADCs as four source-synchronous 384-MHz DDR data streams. IDELAYs on these data lines are calibrated using an ADC test pattern.

The ADC data is piped into a custom Radix-3 384-channel polyphase filter bank, resulting in 1-MHz-wide frequency channels. The polyphase filter bank oversamples the channels at a rate of 32/27 and thus runs at a clock rate of 303.4 MHz. The oversampling operation of the filter bank ensures that we lose minimal information at the channel edges prior to the downstream fine-filter-bank operation that follows the beamformer.

We chose the Xilinx Virtex-6 XC6VLX130T-1FF1156C for the digital receiver because it had sufficient multipliers and memory to fit this particular size of polyphase filter bank. There is some spare capacity for a number of monitoring functions such as histograms and spectral integrators. For many radio astronomy applications the ratio of RAM to DSP is typically 1:1, but the relationship must be proportional to I/O bandwidth. The lowest-speed-grade device achieves 384-MHz performance at the lowest cost, and the footprint is compatible with five other larger devices.

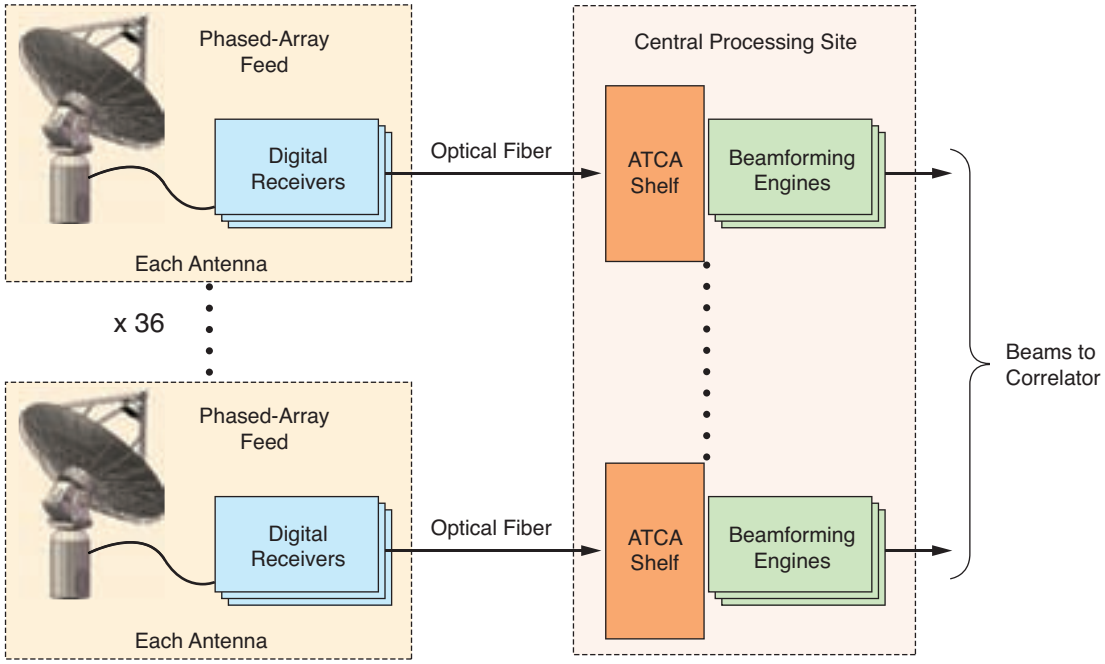


Figure 2 – Up to 8 km of fiber separates the antenna-based processing from the common central processing site of the beamformers and correlators.

ANTENNA TO CENTRAL-SITE DATA COMMUNICATIONS

One of the significant challenges for the ASKAP project team is to transport and distribute the vast amounts of real-time data flowing from the PAFs on the array of antennas through to the digital signal-processing chain at the central site. An overview of the data transport and distribution architecture for one of the 36 PAFs is shown in Figure 4.

Data leaves the digital receiver's Virtex-6 FPGA by means of 16 GTX transceivers, each operating at a line rate of 3.1875 Gbps. A 159.375-MHz reference clock for the serializers is generated locally on each digitizer card, so there is no common clock reference that could correlate and cause artifacts in the data. Each link carries 1/16 (19 MHz) of the total processing bandwidth for the four PAF ports the digital receiver card processes. Since each beamformer card also processes 1/16 of the bandwidth, each output link from the digitizer FPGA is destined for a different beamformer card. The serial output data from the digital receiver FPGA is packetized and aggregated into a custom XAUI (10G Attachment Unit Interface) packet structure. Since the 16 links are unidirectional, point-to-point interconnects, there is no need for a full packet-switched protocol.

The 16 serial output links from the FPGA are grouped into four sets of four lanes, each with XAUI frame encoding. This passes to a Netlogic XAUI transceiver device onboard the digital receiver card that reserializes the four input streams to a 10.51875-Gbps 64b/66b encoded output serial stream (Fibre Channel line rate). The four 10G transceivers drive four Finisar SFP+ optical transceivers.

Each antenna has 192 (48 digitizers/antenna x 4 fibers/digitizer) single-mode fibers connecting to the central site, for a total data rate of 2 Tbps. The fiber spans vary from hundreds of meters for the closest antennas up to 8 km for the farthest.

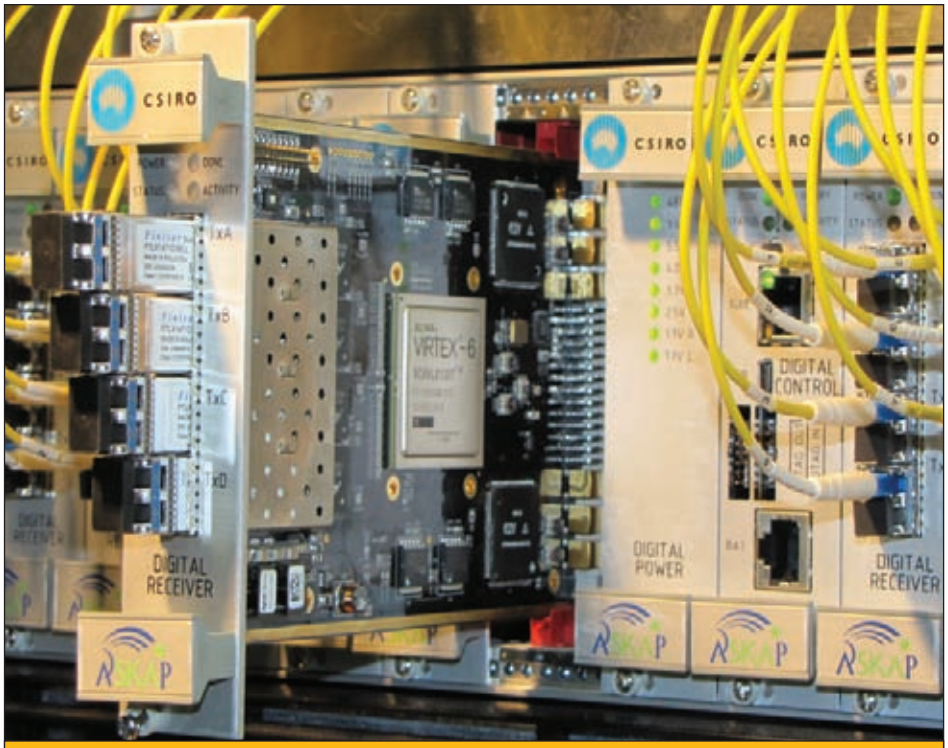


Figure 3 – The digital receiver card samples, filters and transmits PAF receptor data to the central processing site.

BEAMFORMER DATA ROUTING

Two key components of the data-routing and distribution architecture are the Vitesse 6.5-Gbps 72 x 72 cross-point switch and the industry-standard Advanced Telecommunications Computing Architecture (ATCA) backplane. Together, these components make it possible to route individual 3.1875-Gbps serial data streams from the digital receivers directly to the appropriate beamformer card for further processing.

We use discrete ROSA (receiver optical subassembly) devices to convert the 10-Gbps optical data signals from the antennas back to electrical signals, then deserialize them to four 3.1875-Gbps signals via the same XAUI transceiver device used on the digital receiver. This circuitry resides on the ATCA rear transition module, with each RTM designed to accept 12 x 10-Gbps data streams.

The ATCA backplane supports up to 16 cards, each with four duplex 3-Gbps data links to every other card, meaning

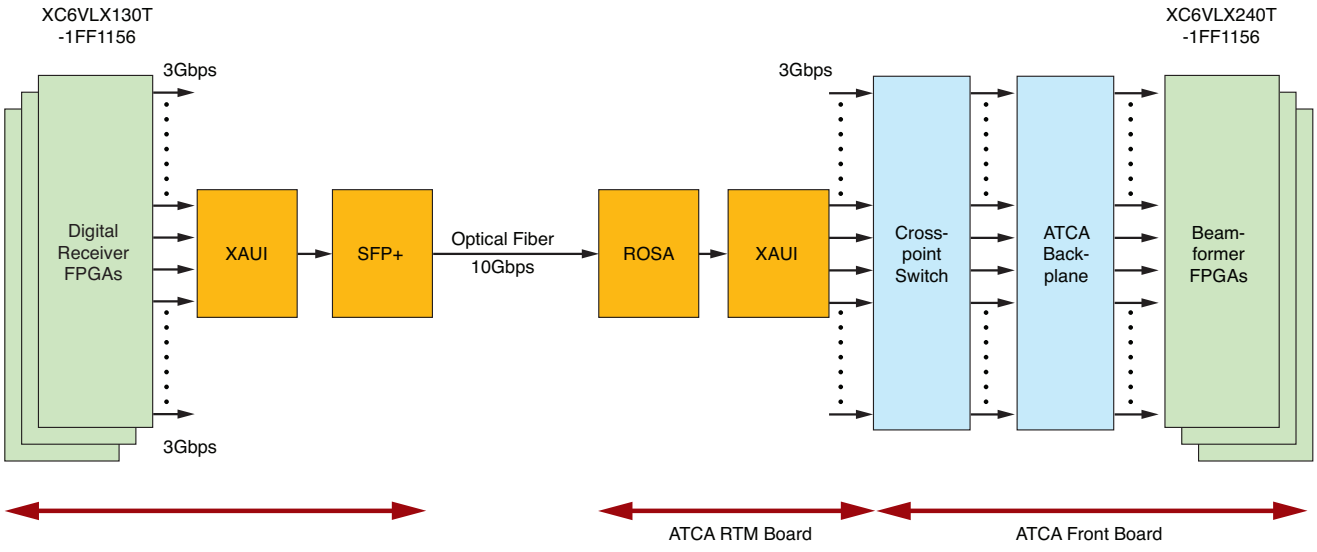


Figure 4 – Each antenna passes 2 Tbps of data to the beamformer, which distributes it using an ATCA backplane.

that on a single card there are (16-1) x 4 = 60 duplex links. One of the challenges of using the ATCA backplane is that the full-mesh connections are different on each slot, so that the signal routing must be dynamically assigned at startup depending on which slot a particular card is located. It is here that we use the 72 x 72 crosspoint switch to set the routing matrix of connections to each card.

Once a 3.1875-Gbps stream reaches its destination beamformer card, it is routed to a GTX receiver on one of the four LX240T processing FPGAs. Here the 3.1875-Gbps stream is deserialized and the data packet decoded. Since each beamformer FPGA only processes approximately 5 MHz of the total 304-MHz bandwidth, three-quarters of the 19-MHz bandwidth arriving on the 3.1875-Gbps streams is forwarded to the other three beamformer FPGAs on the same card. This final hop to the beamformer subsystem also makes use of packetized links, but in this case the physical transport is via LVDS I/O interconnects between each of the DSP FPGAs.

The reference clock for the GTX deserializers is generated locally on each DSP card. Hence, the data transport design makes use of the clock-correction and elastic-buffer features within the Virtex-6 GTX transceiver core.

BEAMFORMER PROCESSING

The data communications system results in all PAF receptors for a common frequency channel arriving at a particular beamformer FPGA. Timing signals injected in the digital receiver are used to synchronize all 192 data streams within the beamformer. Although the main function of the beamforming system is to form multiple beams, there are also a number of PAF support functions that go with calibration and monitoring of the system.

Figure 5 – A 16-slot ATCA shelf holds the beamforming hardware, the heart of which is a Virtex-6 LX240T.

The beamforming operation takes the serial streams of PAF data and buffers them in a dual-port RAM. While data is flowing in from the communication links it is possible to form beams on the previously stored time slice. A set of beamformer weights contains a list of receptors to be included in the beamforming computation and an associated complex weight. DSP48 elements then multiply and accumulate the result for each time slice.

Data from each of the beamformers (there can be up to 36 beams in parallel) is then stored in external DDR3 memory. Each FPGA is directly connected to one stick of DDR3 with an interface operating at 64 bits x 800 Mbps.

Before the correlator processes the beam data, we apply a second polyphase filter bank to set the final frequency resolution of the beams. The relatively slow data input into this second filter bank and the large number of beams necessitate a larger storage area. Xilinx FPGA internal block RAM operates very differently than DDR3 memory. BRAM has a very high bandwidth but limited depth, vs. a relatively low bandwidth and essentially unlimited depth of DDR3. Beam data

Figure 6 – Anchoring the beamforming hardware are the ATCA mesh, crosspoint switch, Virtex-6 FPGAs, DDR3 memory and SFP+ optics.

is written into DDR3 memory and read out in an order appropriate for the filter bank and correlator.

One of the support functions the beamformer FPGA also performs is the calculation of an array co-variance matrix (ACM). The ACM is used to calibrate the array, and the calculation process takes each PAF receptor and cross-correlates it with every other receptor. The result is integrated for several seconds so as to allow the result to converge and also to reduce the output data rate. Both the beamformer and the ACM process data at 318.75 MHz (twice the basic communications clock of 159.375 MHz). Once the ACM is calculated, it is output from each beamformer card over Gigabit Ethernet using UDP packets.

The beamformer card (see Figure 6) is a standard ATCA card (8U x 280 mm) consisting of four signal-processing FPGAs (Virtex-6 XC6VLX240T-1FF1156C), one control FPGA (Virtex-5 LX50T), a crosspoint switch (Vitesse VSC3172), four sticks of DDR3 PC3-8500 memory and eight SFP+ ports (four full duplex, four transmit only).

DEVELOPMENT SOFTWARE

The design flow that we used for all digital processing blocks (test signal generators, polyphase filters, beamformers) centers on Xilinx System Generator for MATLAB®/Simulink®. We wrote data communications, control and monitoring, and interfaces in VHDL, using System Generator to simulate the signal-processing blocks and ModelSim to simulate the HDL-based blocks.

After simulation, we independently verified each block in hardware and then integrated them with other completed blocks to confirm that the complete data and processing path is operating as expected. The ChipScope™ integrated logic analyzer tools are in heavy use during this stage to track down bugs and also for easy visualization of control and data processes on-chip.

We stitch together the entire design by incorporating the DSP netlists from System Generator (in NGC format) via VHDL wrappers, along with all the other HDL-based blocks. We perform synthesis and implementation using XST and the ISE® Project Navigator.

The digital receiver and beamformer designs are very BRAM and DSP48E intensive (see Table 1), which tends to make it difficult to meet timing with simple constraints. Typically we need to apply some floorplanning. The primary goal of floorplanning is to improve timing by reducing route delay. The PlanAhead™ tool provides the insight into our design that we need to be able to apply pipelining on critical nets and guide the map and place-and-route tools to meet our timing requirements. Usually this means optimally placing key sets of BRAMs in addition to block-based design area constraints. This kind of design analysis also provides the ability to greatly reduce implementation runtime.

Table 1 provides a summary of the clock speeds for each FPGA device together with the utilization statistics for the FPGA resources. The total number of FPGAs in the digital receiver and beamforming systems is also listed for reference.

Device	Quantity	Clocks (MHz)	RAM	DSP	I/O	Slices	GTX
Digital receiver LX130T	36x48 =1728	159, 256, 303, 384	64%	74%	35%	60%	100%
Beamformer LX240T	36x16x4 =2304	159, 318, 400	60%	64%	73%	64%	100%

Table 1 – FPGA clock speeds together with utilization statistics for the FPGA resources

LOOKING AHEAD

Xilinx Virtex-6 FPGAs have proven to be a success for the ASKAP digital back end, both in processing performance and I/O bandwidth for data communications. The release of the Virtex-7 device family has our research team excited about even faster data rates, higher processing density and lower power consumption. Larger Virtex-7/Kintex-7 devices with enhanced I/O will significantly change system architectures and lead to boards that are less expensive to design and manufacture.

With the rapid development and adoption of software-defined radio in the commercial world, SKA prototype implementations leveraging this new technology also become a strong possibility as faster, lower-cost, lower-power ADCs are released in combination with Xilinx's newer FPGAs. This will lead to a complete digital solution for direct sampling and processing of PAF data. All of which augurs well for the massive digital systems required for the ultimate radio telescope–the SKA (see *www.skatelescope.org*).

Acknowledgements

The authors are part of the Digital Systems group, an arm of a larger team within CASS working on the ASKAP instrument. We would like to acknowledge the work of other members of the Digital Systems group, including Alan Ng, Ron Koenig, Matt Shields and Evan Davis. We'd also like to acknowledge the Wajarri Yamatji people as the traditional owners of the observatory site.

HOPE – がんばろう 日本 –

The people of Japan experienced a most humbling show of Nature's force from the March 11, 2011 _To-hoku earthquake and tsunami.

We wish to express our appreciation to every person from around the world who sent kind words and help to those afflicted by this unprecedented disaster.

Your overwhelming and personal show of support continues to comfort us during our ordeal, dramatically magnifies our strength and resolve to overcome this crisis and guarantees our success.

With sincere and deepest thanks – Tokyo Electron Device Ltd.

TOKYO ELECTRON DEVICE LIMITED

Head Quarter : Yokohama,Japan
PLD Solution Dept.

E-mail:psd-sales@teldevice.co.jp
http://www.inrevium.jp/eng/x-fpga-board/index.html

Spartan-6 FPGA Consumer Video Kit 2.0
Part Number : TB-6S-CVK2-PRO / TB-6S-CVK2-FND

Spartan-6 FPGA Broadcast Connectivity Kit
Part Number : TB-6S-BCK-FND

XILINX® ALLIANCE PROGRAM PREMIER MEMBER

inrevium
A Revolutionary Innovation Platform

SYNOPSYS®
Predictable Success

Synphony High-Level Synthesis

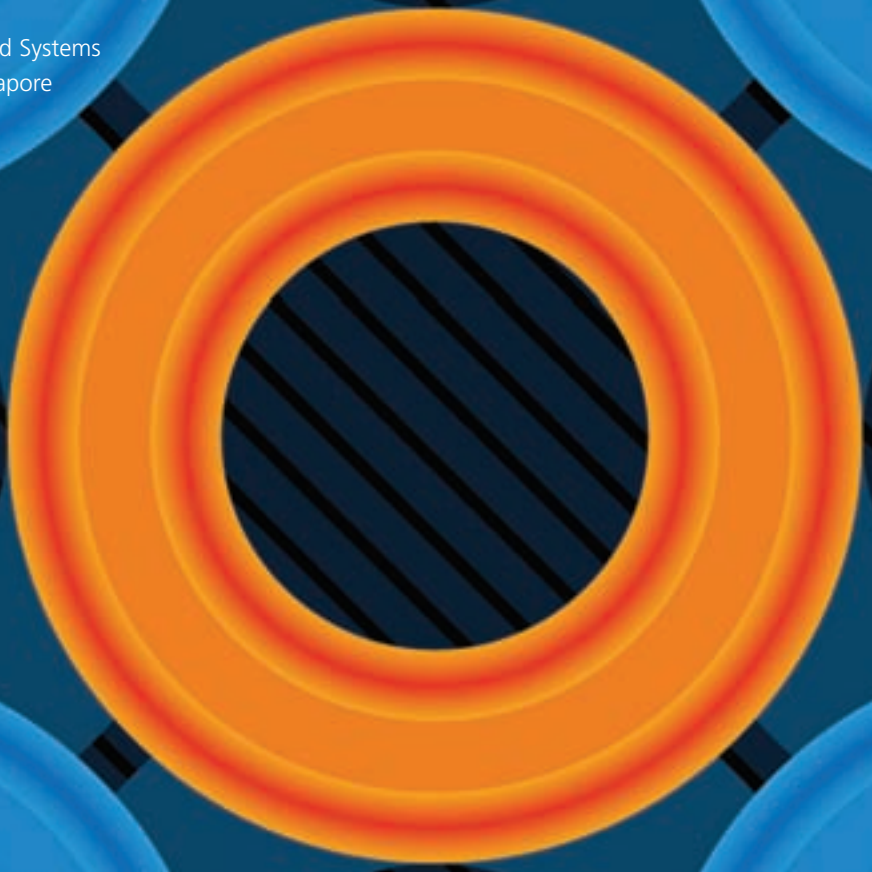
Transform Your Ideas into Reality

- Quickly create synthesizable algorithms using high-level C/C++ or model-based design
- Use synthesizable signal processing libraries for fast design capture
- Rapidly explore FPGA and ASIC implementation tradeoffs in speed, area and power
- Eliminate months of RTL coding and verification effort while reducing risk

Visit www.synopsys.com/HLSinfo

Area-Efficient Design of the SAD Function on FPGAs

by Sharad Sinha
PhD Candidate
Center for High-Performance Embedded Systems
Nanyang Technological University, Singapore
sharad_sinha@pmail.ntu.edu.sg



Here is a design methodology to reduce resource utilization in FPGAs, along with an example implementation of the sum-of-absolute-differences algorithm.

FPGA-based designs require a careful examination of how much area a design will occupy and the timing performance of the design when implemented. The primary goal is to ensure the design fits in the target device while meeting its timing or throughput requirements. In commercial FPGA-based designs, designers generally cap the utilization at around 80 percent of the device's lookup table (LUT) resources so as to ensure that resources exist on the device for future upgrades and feature enhancements, and to facilitate timing closure. The roughly 20 percent or so of free LUTs—and, therefore, free routing resources—can help in meeting tight timing constraints.

FPGA designers know that the more logic they build into the FPGA fabric, the more routing resources they will use. Hence, it is quite possible that while the synthesis tool may succeed in mapping additional logic to LUTs and other resources, it may fail to route connections between them. Because the interconnect utilization is already high due to the existing logic, there are no end-to-end paths to route the signal through for the additional connections. The emphasis is on end-to-end paths because even though free routing segments may exist, the router can't find a way to link them to make end-to-end connections. In this sense, although LUT resources are available, the design is "interconnect limited" and hence probably cannot be expanded.

The reduction in area has an economic impact as well. If the design could be made smaller in some way while still meeting the application requirements, a smaller FPGA device would suffice and this would translate into a lower design and product cost. Of course, if the design has special needs such as Ethernet modules, embedded processor or transceivers, you need to select an FPGA that supports these modules through hard or soft IP. Nevertheless, reduction in area utilization for the remaining parts of the design can still help in picking a particular device in a specific device family that supports such special modules.

It is possible to analyze the application to be implemented before RTL is made ready. If we analyze the application for inherent parallelism, we can exploit that in the RTL design.

RESOURCE SHARING

Resource sharing has traditionally been one way of reducing area or resource utilization while maintaining functionality. It involves sharing of arithmetic operators such as adders, multipliers and the like by mapping more than one operation to one instance of an operator. For example, three adders may perform six rather than three addition operations when shared, halving the number of adders used and thus reducing resource utilization. Xilinx® ISE® software lets you share resources when you enable the corresponding switch (-resource_sharing) in the synthesis-properties dialog box. The tools can detect and implement resource sharing when you describe mutually exclusive assignments in an if-else block (Figure 1) or a case-endcase block (Figure 2).

As you can see, the assignments are mutually exclusive. Therefore, eight addition operations can share two adders when resource sharing is enabled. This type of resource sharing is dependent on identification of mutually exclusive assignments that may exist in the RTL design. However, how can resources be shared if mutually exclusive assignments do not exist and what are the pros and cons of doing so? To answer this question, let's delve into resource sharing at a higher level of abstraction.

A STEP ABOVE RTL

"Higher level of abstraction" refers to design description at a level above RTL. It is possible to analyze the application to be implemented before RTL is made ready. If we analyze the application for inherent parallelism, we can exploit that in the RTL design. At the same time, an analysis of the data flow graph of the application can help us in designing a resource-shared implementation.

This type of graph gives information about the flow of data in the application. Data flows from one operation to another, and hence there can be data dependencies between operations. Operations that are not dependent on each other could execute in parallel. However, parallel execution almost always results in high resource utilization; because our goal is to reduce resource utilization, we will sidestep parallel execution models.

Instead, let's take as an example our implementation of the sum-of-absolute differences (SAD) algorithm to show how to analyze resource sharing at a higher level of abstraction than RTL, reducing resource utilization compared with relying on mutually exclusive assignments in RTL description.

```
if ( <condition> == 2'b01)
      <registerD> = <registerA> +<registerB> + <registerC>;
else if (<condition> == 2'b10)
      <registerD> = <registerX> + <registerY> + <registerZ>;
else if (<condition> == 2'b11)
      <registerD> = <registerP> + <registerQ> + <registerR>;
else
      <registerD> = <registerL> + <registerM> + <registerN>;
```

Figure 1 – The if-else block

```
case(condition)
   2'b01: <registerD> = <registerA> + <registerB> + <registerC>;
   2'b10: <registerD> = <registerX> + <registerY> + <registerZ>;
   2'b11: <registerD> = <registerP> + <registerQ> + <registerR>;
   2'b00: <registerD> = <registerL> + <registerM> + <registerN>;
endcase
```

Figure 2 – The case-endcase block

SAD is an important algorithm in the motion-estimation block in the MPEG-4 codec. It can also be used for object recognition. In this pixel-based method, the value of each pixel in an image block is subtracted from the value of the corresponding pixel in another image block to determine which parts of the image have moved from one frame to another. If the image block size is 4 x 4, then all the 16 absolute-difference values are summed at the end. You'll find example code in the sad.c file of the open-source xvid codec.

Figure 3 shows the data flow graph (DFG) of this algorithm for processing a 4 x 4 image block. You can implement the subtraction operations in parallel because no subtraction operation depends on any other subtraction operation. However, as we were interested in reducing resource utilization, we did not implement the parallel-execution model. Instead, we used the resource-allocation and binding algorithm called Extended Compatibility Path Based (ECPB) hardware binding that my colleagues and I have proposed. [1]

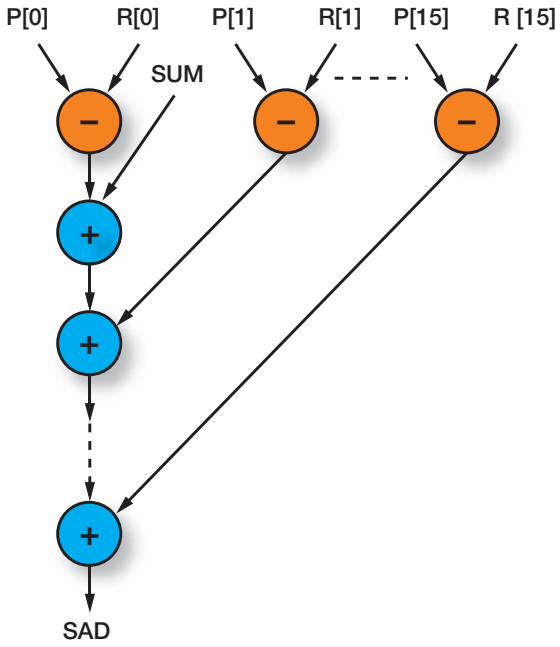


Figure 3 – Data flow graph of the SAD algorithm

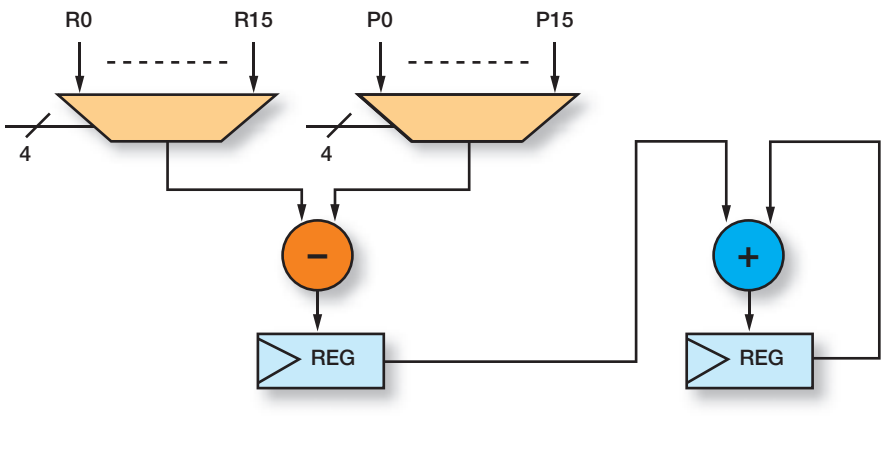


Figure 4 – Datapath of implemented SAD algorithm

Resource allocation and binding basically refer to allocating resources like adders, multipliers and so on to operations in the DFG, and then binding these resources to operations to either reduce device resource utilization, increase maximum clock frequency or both. We focused on reducing area subject to the final design meeting our functional constraints. The algorithm exploits interoperation flow dependency and analyzes intra-operation flow dependency in a scheduled DFG—that is, one whose operations have been scheduled in different time steps or clock cycles.

The algorithm schedules operations that can execute in parallel in the same time step, while operations that depend on data from other operations are scheduled in different time steps. The algorithm constructs a weighted and ordered compatibility graph (WOCG) for each kind of operation in the scheduled DFG. Thus, there is one WOCG for the subtraction operation and another for addition. The methodology uses the weight relation $wij = 1 + \alpha*Fij + \beta*Nij + \gamma*Rij$ to assign weights to the edges in the WOCG. Here, wij is the weight value between operations "i" and "j" of the same type. Fij is the measure of flow dependency and Nij is the number of common inputs between operations "i" and "j." Rij equals 1 if the output of operations "i" and "j" can be stored in the same register, otherwise it equals 0. We set the tuning parameters, α, β and γ, to 1, 1 and 2 respectively for the case we are studying here.

The next step involves finding a longest path in the WOCG using the longest-path algorithm. All operations in that longest path are mapped to just one operator. After removing the bound operations from the WOCG, we repeat the longest-path and mapping procedure, continuing until we have processed all WOCGs. Since the algorithm maps more than one operation to the same resource/operator, the operator size is such that it accommodates the biggest operation. In the case of the SAD implementation, we handled all subtraction operations on 8-bit data (gray-scale images) and hence the subtractor operator had 8-bit-wide inputs. We sized the accumulator operator, which sums up SAD value iteratively, to input bit widths of 23 bits and 8 bits.

In Figure 4, which shows the datapath of the implementation, you will see that we used only one subtractor and one adder/accumulator in this design. There is a very high degree of resource sharing among the operations. This kind of resource sharing would not have been possible if the design were described directly at the behavioral RTL level, because mutually exclusive assignments do not exist in the SAD algorithm. Once we generated this datapath, we implemented the design hierarchically using adder, subtractor and multiplexer modules. This implementation involving instantiation of modules is more structural than behavioral and exactly resembles the datapath.

While some readers may find our implementation simple, it should be noted that in large data-dominated applications, this kind of preprocessing at a higher level can help reduce resource utilization. Moreover, it's very easy to automate this method.

To compare the physical-synthesis results obtained for the datapath that the ECPB algorithm generated, we also implemented a completely parallel implementation and two others with datapaths the same as in Figure 4,

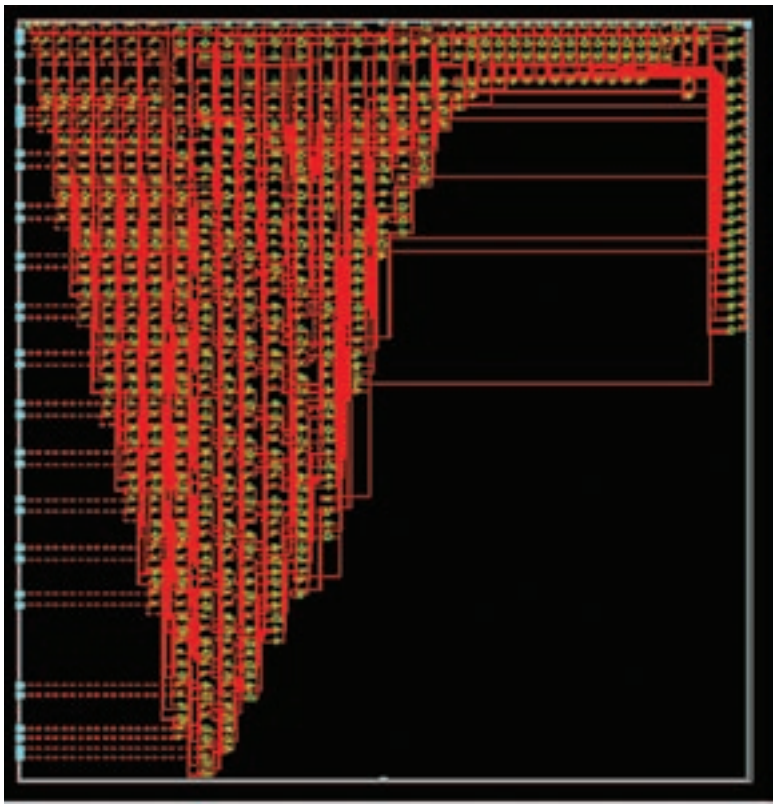


Figure 5 – Honeycomb-like technology layout of Design I(RS)

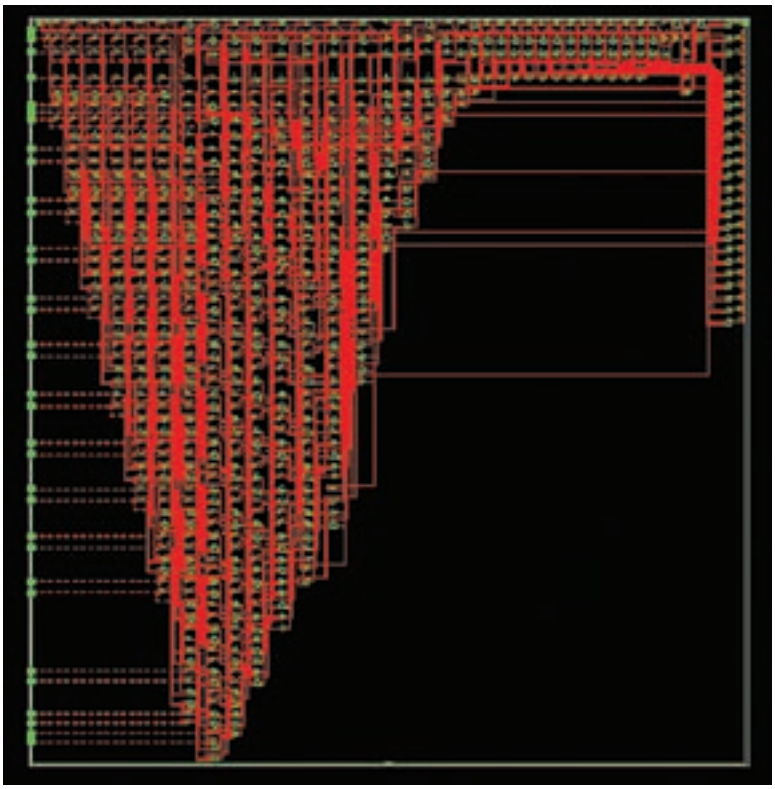


Figure 6 – Honeycomb-like technology layout of Design II(RS)

	Parallel implementation		Design I (case-endcase)		Design II (if-else)		ECPB (hierarchical with instantiations)	
	RS	NRS	RS	NRS	RS	NRS	RS	NRS
Four-input LUTs	368	368	193	272	193	272	160	160
Slices	200	200	110	149	110	149	93	93
Slice flip-flops	391	391	54	54	54	54	54	54
Best achievable clock period (ns)	3.609	3.609	2.662	3.592	2.662	3.592	2.450	2.450

Table I – Comparison of results (RS and NRS stand for resource sharing enabled or disabled, respectively)

but where the final RTL description was not hierarchical. The latter two were flat behavioral designs with "forced," mutually exclusive assignments to account for the muxes. We say "forced" because we tried to implement the same datapath as in Figure 4 but using a behavioral description with mutually exclusive assignments. One design had an if-else construct and the other had case-endcase. Table 1 shows the post-place-and-route results obtained using Xilinx ISE 12.2 (M.63C) software with default settings, Virtex®-4XC4VFX140-11FF1517 as the target device and no timing constraints. There was no reset in any of the implementations and hence internal registers were initialized suitably.

In the table, RS and NRS stand for resource sharing enabled and disabled, respectively, in Xilinx ISE software. We explored designs I and II because different synthesis tools can infer different types of muxes from different styles of HDL code (if-else; case-endcase). [2] The clock period does not take jitter into account and should be derated by an amount according to the clock source specification. Also, we did not use conditioning registers in any implementation for the inputs.

MAJOR RESOURCE SAVINGS

As the table shows, we were able to bring down LUT consumption significantly, at most 56 percent and at the least, 20 percent. However, unlike a completely parallel implementation where data samples can be made available every clock cycle in general, resource sharing puts some restrictions on the way data samples are made available for processing. Because resources are now shared, a new data sample can be processed only after the previous data sample has been partially or fully processed depending on the final datapath. In our ECPB implementation, a new set of P and R values can be made available only after a minimum of 16 clock cycles.

While this may not always be acceptable, it is definitely a technique suitable for use in applications that will allow such sample rates. For instance, the ECPB design can easily process a DV-PAL frame of size 720 x 576 in 1.016 milliseconds without any impact on the PAL frame rate of 25 frames per second.

Digressing a bit from the technical implementation details, Figures 5 and 6 show the honeycomb-like technology layouts for our two designs, I(RS) and II(RS). Some of you might be aware of the recent exhibition of science and technology art at the Museum of Modern Art in New York, which highlighted images and artwork inspired by scientific work. This kind of art gives a different perspective on the work done by engineers and scientists and can help us in appreciating our field of endeavor even more.

References

[1] Dhawan, U., Sinha, S., et al., "Extended compatibility path based hardware binding for area-time efficient designs," Quality Electronic Design (ASQED) 2010, Second Asia Symposium, Penang.

[2] Stephenson, J. and Metzgen, P., "Logic optimization techniques for multiplexers," CP-01003-1.0, Altera Corp., March 2004.

Demystifying FPGAs for Software Engineers

by Glenn Steiner
Senior Manager of Technical Marketing
Xilinx, Inc.
glenn.steiner@xilinx.com

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

Here are some practical tips on how to develop software for FPGA embedded processors.

As product designs increase in complexity, there is a need to use integrated components, such as application-specific standard products (ASSPs), to address design requirements. Years ago, engineers chose individual components for processor, memory and peripherals, and then pieced these elements together with discrete logic. More recently, they would search through catalogs of ASSP processing systems attempting to find the nearest match to meet system requirements. When they need additional logic or peripherals, they often mate an FPGA with an ASSP to complete the solution. Indeed, surveys indicate that FPGAs are now a part of 50 to 70 percent of all embedded systems.

Over the last few years, FPGA sizes have increased, providing sufficient space to accommodate complete processor and logic systems within a single device. Software engineers are now faced with developing and debugging code targeting a processor inside of an FPGA—and in some cases they fear doing so. But a grasp of FPGA basics and an understanding of how to create and debug code for FPGA embedded processors will go a long way to settle their nerves.

WHAT IS AN FPGA?

A field-programmable gate array (FPGA) is an integrated circuit containing logic that may be configured and connected after manufacturing, or "in the field." Where in the past engineers purchased a variety of logic devices from a catalog and then assembled them into a logic design via connections on a printed-circuit board, today hardware designers can implement complete designs within a single device. In their simplest form FPGAs contain:

- Configurable logic blocks consisting of AND, OR, Invert and many other logic functions
- Configurable interconnect enabling logic blocks to be connected together
- I/O interfaces

With these elements, users may create an arbitrary logic design.

Hardware engineers usually write code in HDL (typically either Verilog or VHDL) and then compile the design into an object file that they load into the device for execution. On the surface the HDL programs can look very much like high-level languages such as C. Take, for example, the following implementation of an 8-bit counter written in Verilog (courtesy of *www.asic-world.com*). In it you can see many constructs taken from today's high-level languages:

```
//-----------------------------------------
// Design Name : up_counter
// File Name   : up_counter.v
// Function    : Up counter
// Coder       : Deepak
//-----------------------------------------
module up_counter    (
out      ,  // Output of the counter
enable   ,  // enable for counter
clk      ,  // clock Input
reset       // reset Input
);
//----------Output Ports-------------------
    output [7:0] out;
//------------Input Ports------------------
     input enable, clk, reset;
//------------Internal Variables-----------
    reg [7:0] out;
//-------------Code Starts Here------------
always @(posedge clk)
if (reset) begin
  out <= 8'b0 ;
end
 else if (enable) begin
  out <= out + 1;
end
endmodule
```

TECHNICAL ADVANTAGES OF AN FPGA

Short of using an ASIC with its associated high mask charges, FPGAs are the most flexible, high-performance and cost-effective method of implementing data-processing elements. FPGAs, due to their flexible architecture, enable hardware designers to implement processing systems consisting of elements that are both paralleled and pipelined. This allows designers to tune a system for both performance and latency. Typically these data-processing systems provide higher levels of performance than is obtainable using general-purpose processors, and at a lower cost.

You can couple an external microprocessor to a data-processing system in an FPGA and use it for control. However, having a processor inside the FPGA can present several advantages. An internal processor dramatically reduces the control latency between the processor and the data-processing system, to the point of eliminating many processor cycles. The communication channel between the processor and data-processing system may be 32 or more bits, with additional wires for address and control. For an external processor, these additional wires may necessitate a larger package for both the processor and FPGA, driving up system cost. Alternatively, PCI Express® (PCIe®) can provide a dramatic reduction in the number of pins. Unfortunately, being a relatively new interface, not all processors and FPGAs support PCIe. While inherently high performance, PCIe is a serial interface and thus adds latency between a processor and the data-processing system.

Implementation of a processor inside an FPGA along with the data-processing elements reduces parts count, board space and sometimes power requirements. This can result in a much lower-cost solution. Hardened implementations of processors such as the ARM® Cortex™-A9 processor, or soft implementations such as the Xilinx® MicroBlaze™ processors, are available in FPGAs. FPGA-based processors may also be configured based upon application requirements. FPGA-based systems enable system-level tuning by providing the ability to move decision and computational functions between the processor and the FPGA logic.

IMPLEMENTATION TECHNIQUES

Multiple methods exist for the implementation of FPGA embedded processing systems, but they generally fall into three categories: assembling the system from scratch, putting it together by using wizards or getting there by modifying an existing design.

FPGA tools allow you to assemble a processing system from scratch by selecting the necessary IP from a list, and then connecting the IP via buses and wires. Such assembly, while effective, can be time-consuming.

To speed things up, FPGA tools also allow the rapid assembly of a microprocessor system via wizards. Using drop-down lists or checkboxes, you simply specify the targeted part and the desired processor and peripherals. Figure 1 presents the introductory window to start the wizard and shows the final system the wizard has built. Similarly, you may use tools such as MATLAB® software to rapidly assemble a data-processing system with interfaces to the processor bus for control. You can then connect the processor and data-processing system by simply matching bus interfaces.

The third way of implementing an embedded processing system is to modify an existing reference design or add to an existing hard-processor system. FPGA reference designs and hard-processor systems continue to evolve and many are becoming more market focused. In many cases the designs are so complete that the hardware designer need not add any extra components. The software designer typically will find complete drivers and prebuilt operating systems for these reference designs.

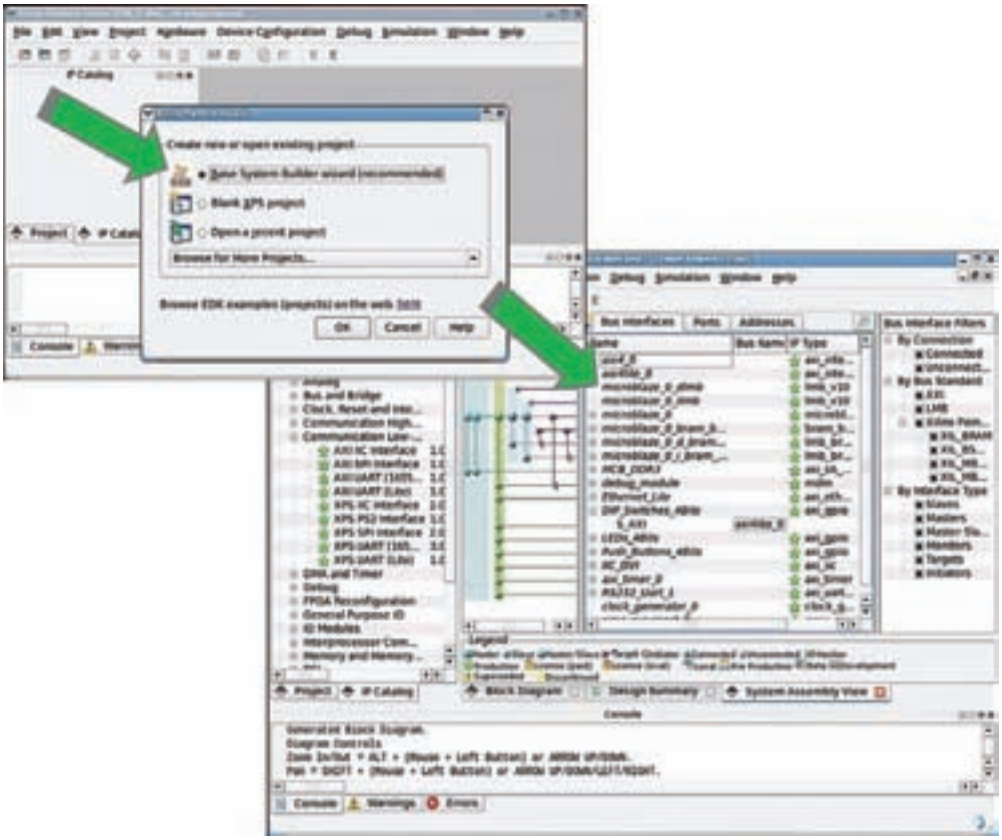


Figure 1 – Wizard start screen and completed system

The first two methods noted above are both effective ways to create a processor system. However, the third method begins with an existing, verified reference design and significantly reduces the development time for both hardware and software engineers.

DISPELLING MYTHS

A number of myths have sprung up in the engineering community about the presumed difficulty of developing code for processors inside of FPGAs. We would like to dispel these myths.

Myth: Coding for a processor in an FPGA is difficult.
Truth: Most FPGA embedded processing development is done in C or C++ within modern software development environments.

Most FPGA suppliers now support software development using Eclipse. Eclipse is a flexible software development environment supporting plug-ins and providing text editors, compilers, linkers, debuggers, trace modules and code management.

As an open environment, Eclipse enjoys a large community of developers that are constantly adding new capabilities. For example, if a coder does not like the provided editor they can install an editor that better meets their needs. Figure 2 shows the Eclipse code editor with the "hello world" program.

Myth: There are no ASSP-like processor systems for FPGAs.
Truth: There are now prebuilt FPGA soft embedded processor designs as well as hard-processor designs with ASSP-like peripheral sets.

FPGAs with soft or hard processors add an extra dimension of capability. FPGA embedded soft-processor reference designs integrate 32-bit RISC processors, memory interfaces and industry-standard peripherals. The flexible nature of such processors allows users to trade off logic for additional performance capabilities, such as the addition of an MMU supporting a modern operating system. A broad array of FPGA choices allows users to select a processor configuration, peripherals, data-processing logic and logic performance levels to meet their system requirements. Prebuilt ASSP-like reference designs enable software designers to begin coding without a hardware engineer having to first implement a processor system. In many cases, the prebuilt design will meet the embedded processor system requirements, eliminating the need for the hardware engineer to do further processor system design. In other cases, the hardware engineer has an excellent platform upon which to add peripherals and connect custom hardware accelerators.

Myth: Debugging code with a processor in an FPGA is hard.
Truth: Software debug of an FPGA embedded processor is as easy as debugging a non-embedded processor. Debuggers support downloading of code, running programs, single-stepping at the source-code and object-code levels, setting breakpoints, examining memory and registers. Additional tools are available for code profiling and trace.

Myth: My favorite OS is not supported.
Truth: The most popular operating systems are supported for the most popular embedded processors, and the list is growing. The Xilinx MicroBlaze supports Linux, ThreadX, MicroC/OS-II and eCos, among other operating systems.

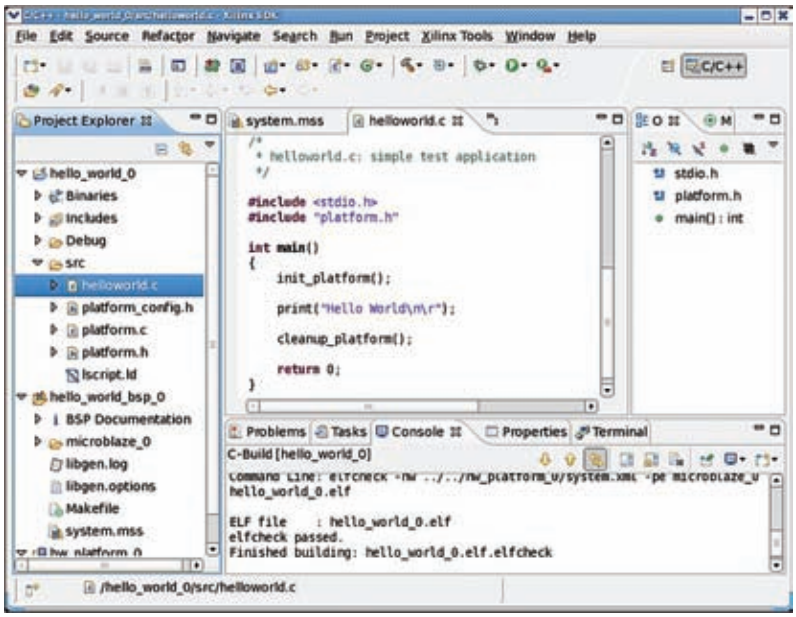


Figure 2 – Eclipse IDE with code

Myth: Drivers don't exist.
Truth: FPGA embedded processors have large libraries of peripherals with included drivers. Table 1 is a representative list of some of the soft peripherals available for an FPGA soft processor. Appropriate drivers exist for all of these devices.

Myth: Hardware engineers have to build it before I can code.
Truth: Prebuilt and tested processor system designs are available, enabling immediate software development.

These prebuilt ASSP-like processor systems include the processor, memory controller and memory, flash memory controllers and peripherals such as UARTs, GPIOs and Ethernet interfaces. These systems are delivered with reference software design examples including demonstrating Linux boot.

Myth: I can't profile or trace with FPGA embedded processors.
Truth: Profiling and trace tools are also available. Profiling allows developers to see how much time the processor has spent in what functions as well as how many calls it has made to any given function.

Myth: FPGA software development tools are expensive.
Truth: Both ASSP and FPGA suppliers price their embedded-software development capabilities between $200 and $500. In addition, many vendors offer trial and free or limited versions as well as discounted evaluation kits.

CREATING AND DEBUGGING CODE

The software development process for FPGA embedded processor systems follows a few general steps:

1. Create a software development workspace and import the hardware platform.
2. Create the software project and board support package.
3. Create the software.
4. Run and debug the software project.
5. Optional: Profile the software project.

Steps 3, 4 and 5 are familiar to most developers. Steps 1 and 2 may be new to some developers but they are straightforward. We will use the Eclipse development environment as an example in looking more closely at each of the steps.

Creating a Workspace and Importing the Hardware Platform

After starting Eclipse, the user is prompted for a workspace to use. A workspace is simply a directory path where project

COMMUNICATION	PERIPHERALS	SYSTEM
PCI	General-Purpose I/O	Debug Module
PCI Express	UART 16550	Interrupt Controller
USB	Delta/Sigma ADC, DAC	Timer/PWM
I2C	External Peripheral Controller	Timebase/Watchdog
SPI	ADC	Mailbox
Gigabit Ethernet MAC	**MEMORY**	Mutex
Ethernet MAC Lite	Multiport Memory Controller (SDRAM, DDR, DDR2, DDR3)	Clock generator
CAN	External Memory Controller (Flash, SRAM)	System reset block
FlexRay	Compact Flash	Central DMA
MOST	On-Chip Memory	

Table 1 – Sample soft-processor peripheral list

files are to be stored. Next, the user specifies the hardware platform (design). The hardware development tools automatically generate this file, which describes the processor system including memory interfaces and peripherals along with memory maps. The file is output from the hardware development tools and the hardware engineer will typically supply this file to the software developer. Once specified, the hardware platform is imported and this step is complete.

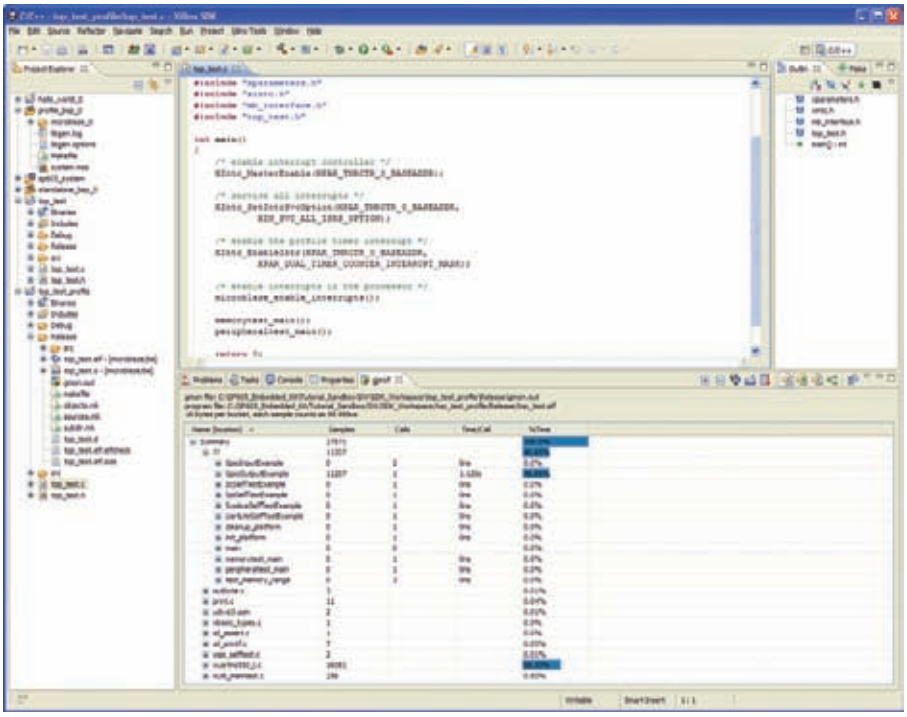


Figure 3 – Eclipse profile perspective

Creating the Software Project and Board Support Package

The board support package (BSP) contains the libraries and drivers for software applications use. A software project is the software application source and settings.

In versions of Eclipse customized for Xilinx projects, you select File ➔ New ➔ Xilinx C Project. For Xilinx C projects, Eclipse automatically creates Makefiles that will compile the source files into object files and link those object files into an executable. During this step the user defines the Project Name, associates it with the hardware platform by providing the hardware platform name created in Step 1 and then names the project.

Next, the system confirms the generation of the BSP and automatically loads the applicable drivers based upon the defined hardware platform and operating system. These drivers are then compiled and the BSP has been generated.

Creating the Software

At this point, you may either import a software example or create code from scratch. As you save code, Eclipse automatically compiles and links the code, reporting any compiler or linker errors.

Running and Debugging the Software Project

With FPGAs there is one step that you must complete prior to executing code—programming the FPGA. In Eclipse, you simply selects Tools ➔ Program FPGA. This step takes the hardware design that the hardware engineer has created and downloads it to the FPGA. Once that job is completed, you may select the type of software to be built. "Debug" turns off code optimization and inserts debugging symbols, while "Release" turns on code optimization. For profiling, use the –pg compile option.

Finally, you may run the code by selecting Run and defining the type of run configuration and compiler options. If you have selected Release, the processor will immediately begin code execution. Otherwise, the processor will execute a few boot instructions and will stop at the first line of source code and the Debug perspective will appear in Eclipse.

The Debug perspective gives you a view of the source or object code, registers, memory and variables. You may single-step code at either the source or object level and may set breakpoints for code execution.

Profile the Software Project

If you wish, at this point you may profile code and view the number of function calls as well as see the percentage of time spent in any given function. Figure 3 shows an example of the Eclipse Profile Perspective including profiler results.

THE FPGA ADVANTAGE

For cost, power, size and overall system efficiency, FPGA embedded processors are becoming primary design choices. The good news is that software engineers do not need to consider an FPGA embedded processor as a mystery or any more difficult to program than an external processor.

FPGA vendors are providing industry-standard development environments such as Eclipse at competitive costs and customized for FPGA embedded processing. Within these environments users can create, compile, link and download code, and as necessary debug their designs in the same manner as they have done in the past with external processors. Prebuilt processor reference designs enable software engineers to start coding and testing without the hardware engineer having to provide the final design. Finally, FPGA embedded processors have extensive IP libraries, drivers and OS support.

What Can FPGA-Based Prototyping Do for You?

The greatest benefit may be when first SoC silicon arrives in the lab and the embedded software is running that same day.

By Doug Amos
Business Development Manager
Solutions Marketing
Synopsys Inc.
damos@synopsys.com

René Richter
Corporate Application Engineering Manager
Hardware Platforms Group
Synopsys Inc.
Rene.Richter@synopsys.com

Austin Lesea
Principal Engineer
Xilinx Labs
Austin@xilinx.com

Xcell Journal is pleased to run in its entirety Chapter 2 of an exciting new book called *FPGA-Based Prototyping Methodology Manual* (FPMM). Written by Synopsys' Doug Amos and René Richter and Xilinx's own Austin Lesea, the FPMM is a comprehensive and practical guide to Design-for-Prototyping. To learn more about the FPMM and to download it as an e-book, go to *www.synopsys.com/fpmm*. Print copies are available from Amazon.com, Synopsys Press and other bookstores. While the book presents examples using Xilinx and Synopsys products, the methodology and lessons are applicable to any prototyping projects, regardless of the tools or FPGAs used.

As advocates for FPGA-based prototyping, we may be expected to be biased toward its benefits and blind to its deficiencies. However, that is not our intent. Our *FPGA-Based Prototyping Methodology Manual* (FPMM) is intended to give a balanced view of the pros and cons of FPGA-based prototyping because, at the end of the day, we do not want people to embark on a long prototyping project if their aims would be better met by other methods, for example by using a SystemC-based virtual prototype.

Let's take a deeper look at the aims and limitations of FPGA-based prototyping and its applicability to system-level verification and other goals. By staying focused on the aim of the prototype project, we can simplify our decisions regarding platform, IP usage, design porting, debug and other aspects of design. In this way we can learn from the experience others have had in their projects by examining some examples from prototyping teams around the world.

FPGA-BASED PROTOTYPING FOR DIFFERENT AIMS

Prototyping is not a pushbutton process and requires a great deal of care and consideration at its different stages. As well as explaining the effort and expertise involved in the process, we should also offer some incentive as to why we should (or maybe should not) perform prototyping during our SoC projects.

In conversation with prototypers over many years, one of the questions we liked to ask was "why do you do it?" There are many answers, but we are able to group them into the general reasons shown in Table 1. So for example, "real-world data effects" might describe a team that is prototyping in order to have an at-speed model of a system available to interconnect with other systems or peripherals, perhaps to test compliance with a particular new interface standard. Their broad reason to prototype is "interfacing with the real world" and prototyping does indeed offer the fastest and most accurate way to do that in advance of real silicon becoming available.

A structured understanding of these project aims and why we should prototype will help us to decide if FPGA-based prototyping is going to benefit our next project.

Let us, therefore, explore each of the aims in Table 1 and how FPGA-based prototyping can help achieve them. In many cases we shall also give examples from the real world and the authors wish to thank in advance those who have offered their own experiences as guides to others in this effort.

Project Aim	Why Prototype?
Test real-time dataflow Early hardware-software integration Early software validation	High performance and accuracy
Test real-world data effects Test real-time human interface Debug rare data dependencies	Interfacing with the real world
Feasibility (proof of concept) Testing algorithms	In-lab usage
Public exhibitions Encourage investors	Out-of-lab demonstration
Extended RTL test and debug	Other aims

Table 1 – General aims and reasons to use FPGA-based prototypes

HIGH PERFORMANCE AND ACCURACY

Only FPGA-based prototyping provides both the speed and accuracy necessary to properly test many aspects of the design. We put this reason at the top of the list because it is the most likely underlying reason of all for a team to be prototyping, despite the many given deliverable aims of the project. For example, the team may aim to validate some of the SoC's embedded software and see how it runs at speed on real hardware, but the underlying reason to use a prototype is for both high performance and accuracy. We could validate the software at even higher performance on a virtual system, but we lose the accuracy that comes from employing the real RTL.

REAL-TIME DATAFLOW

Part of the reason that verifying an SoC is hard is because its state depends upon many variables, including its previous state, the sequence of inputs and the wider system effects (and possible feedback) of the SoC outputs. Running the SoC design at real-time speed connected into the rest of the system allows us to see the immediate effect of real-time conditions, inputs and system feedback as they change.

A very good example of this is real-time dataflow in the HDMI prototype performed by the Synopsys IP group in Porto, Portugal. Here, a high-definition (HD) media data stream was routed through a prototype of a processing core and out to an HD display, as shown in the block diagram in Figure 1. Notice that across the bottom of the diagram there is audio and

HD video dataflow in real time from the receiver (from an external source) through the prototype and out to a real-time HDMI PHY connection to an external monitor.

By using a presilicon prototype, we can immediately see and hear the effect of different HD data upon our design, and vice versa. Only FPGA-based prototyping allows this real-time dataflow, giving great benefits not only to such multimedia applications but to many other applications where real-time response to input dataflow is required.

HARDWARE-SOFTWARE INTEGRATION

In the above example, readers may have noticed that there is a small MicroBlaze™ CPU in the prototype along with peripherals and memories, so all the familiar blocks of an SoC are present. In this design the software running in the CPU is used mostly to load and control the A/V processing. However, in many SoC designs it is the software that requires most of the design effort.

Given that software has already come to dominate the SoC development effort, it is increasingly common that the software effort is on the critical path of the project schedule. It is software development and validation that govern the actual completion date when the SoC can usefully reach volume production. In that case, what can system teams do to increase the productivity of software development and validation? To answer this question, we need to see where software teams spend their time.

MODELING AN SoC FOR SOFTWARE DEVELOPMENT

Software is complex and hard to make perfect. We are all familiar with the software upgrades, service packs and bug fixes in our normal day-to-day use of computers. However, in the case of software embedded in an SoC, this perpetual fine-tuning of software is less easily achieved. On the plus side, the system with which the embedded

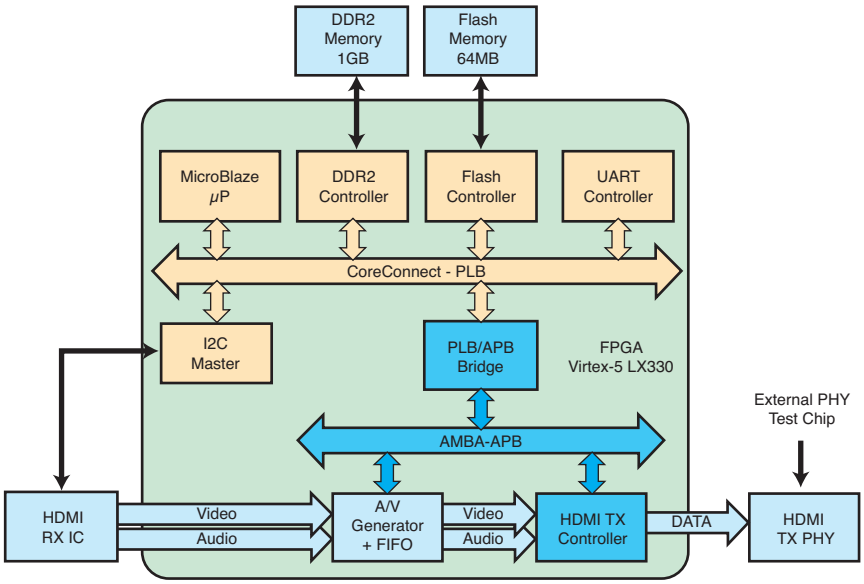


Figure 1 – Block diagram of HDMI prototype

software interacts, its intended use modes and the environmental situation are all usually easier to determine than for more general-purpose computer software. Furthermore, embedded software for simpler systems can be kept simple itself and so is easier to fully validate. For example, an SoC controlling a vehicle subsystem or an electronic toy can be fully tested more easily than a smartphone running many apps and processes on a real-time operating system (RTOS).

If we look more closely at the software running in such a smartphone, for example the Android software shown in Figure 2, then we see a multilayered arrangement, called a software stack. (This diagram is based on an original by software designer Frank Abelson in his book *Unlocking Android*.)

Taking a look at the stack, we should realize that the lowest levels—i.e., those closest to the hardware—are dominated by the need to map the software onto the SoC hardware. This requires absolute knowledge of the hardware to an address and clock-cycle level of accuracy. Designers of the lowest level of a software stack, often calling themselves platform engineers, have the task of describing the hardware in terms that the higher levels of the stack can recognize and reuse. This description is called a BSP (board support package) by some RTOS vendors and is also analogous to the BIOS (basic input/output system) layer in our day-to-day PCs.

The next layer up from the bottom of the stack contains the kernel of the RTOS and the necessary drivers to interface the described hardware with the higher-level software. In these lowest levels of the stack, platform engineers and driver developers will need to validate their code on either the real SoC or a fully accurate model of the SoC. Software developers at this level need complete visibility of the behavior of their software at every clock cycle.

At the other extreme for software developers, at the top layer of the stack, we find the user space, which may be running multiple applications concurrently. In the smartphone example these could be a contact manager, a video display, an Internet browser and, of course, the phone subsystem that actually makes calls. Each of these applications does not have direct access to SoC hardware and is

actually somewhat divorced from any consideration of the hardware. The applications rely on software running on lower levels of the stack to communicate with the SoC hardware and the rest of the world on its behalf.

We can generalize that, at each layer of the stack, a software developer only needs a model with enough accuracy to fool his own code into thinking it is running in the target SoC. More accuracy than necessary will only result in the model running more slowly on the simulator. In effect, SoC modeling at any level requires us to represent the hardware and the stack up to the layer just below the current level to be validated and optimally, we should work with just enough accuracy to allow maximum performance.

For example, application developers at the top of the stack can test their code on the real SoC or on a model. In this case the model need only be accurate enough to fool the application into thinking that it is running on the real SoC; it does not need cycle accuracy or fine-grained visibility of the hardware. However, speed is important because multiple applications will be running concurrently and interfacing with real-world data in many cases.

This approach of the model having "just enough accuracy" for the software layer gives rise to a number of different modeling environments being used by different software developers at different times during an SoC project. It is possible to use transaction-level simulations, modeled in languages such as SystemC, to create a simulator model that runs with low accuracy but at high enough speed to run many applications together. If handling of real-time, real-world data is not important, then we might be better considering such a virtual prototyping approach.

However, FPGA-based prototyping becomes most useful when the whole software stack must run together or when real-world data must be processed.

EXAMPLE PROTOTYPE USAGE FOR SOFTWARE VALIDATION

Only FPGA-based prototyping breaks the inverse relationship between accuracy and performance inherent in modeling methodologies. By using FPGAs we can achieve speeds up to real-time and yet still be modeling at the full RTL cycle accuracy. This enables the same prototype to be used not only for the accurate models required by low-level software validation but also for the high-speed models needed by the high-level application developers. Indeed, the whole SoC software stack can be modeled on a single FPGA-based prototype. A very good example of this software validation using FPGAs is seen in a project performed by Scott Constable and his team at Freescale Semiconductor's Cellular Products Group in Austin, Texas.

Freescale was very interested in accelerating SoC development because short product life cycles of the cellular market demand that products get to market quickly not only to beat the competition, but also to avoid quickly becoming obsolete. Analyzing the biggest time sinks in its flow, Freescale decided that the greatest benefit would be achieved by accelerating their cellular 3G protocol testing. If this testing could be performed presilicon, then Freescale would save considerable months in a project schedule. When compared to a product lifetime that is only one or two years, this is very significant indeed.

Protocol testing is a complex process that even at high real-time speeds requires a day to complete. Using RTL simulation would take years and running on a faster emulator would take weeks, neither of which was a practical solution. FPGAs were chosen because that was the only way to achieve the nec-

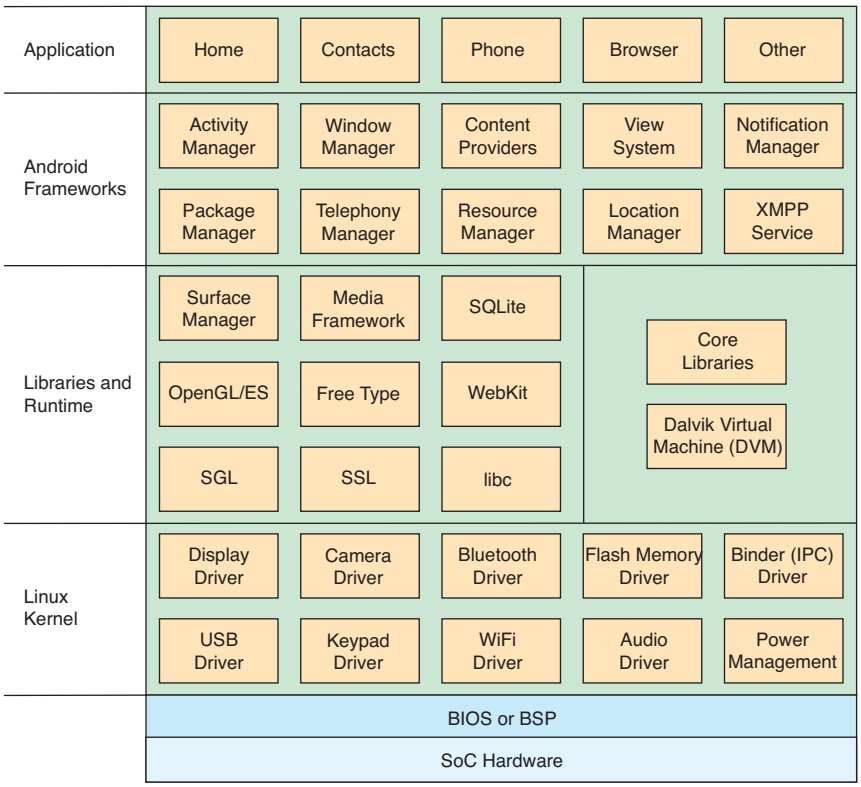


Figure 2-- The Android software stack

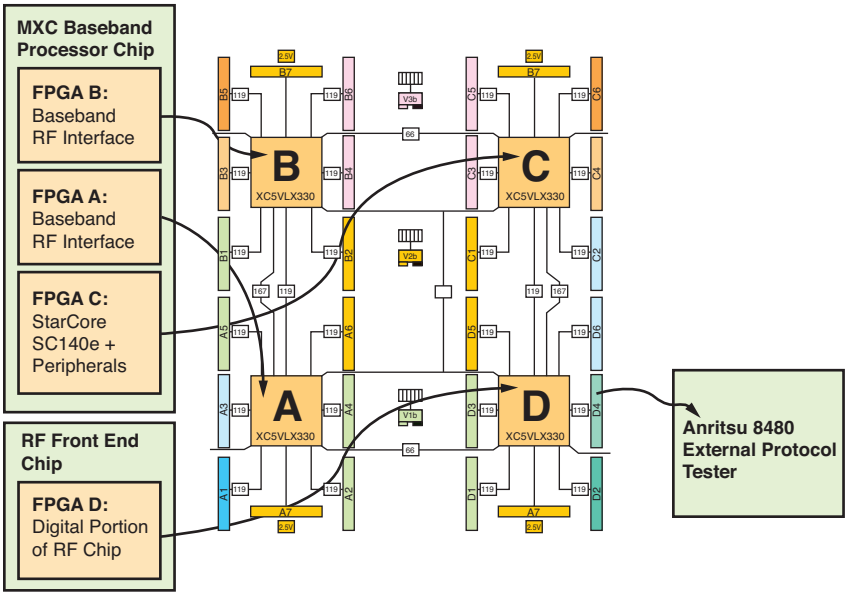


Figure 3 -- The Freescale SoC design partitioned into a HAPS-54 board

essary clock speed to complete the testing in a timely manner.

Protocol testing requires the development of various software aspects of the product, including hardware drivers, operating system and protocol stack code. While the main goal was protocol testing, as mentioned, by using FPGAs all of these software developments would be accomplished presilicon and greatly accelerate various end-product schedules.

Freescale prototyped a multichip system that included a dual-core MXC2 baseband processor plus the digital portion of an RF transceiver chip. The baseband processor included a Freescale StarCore DSP core for modem processing and an ARM®926 core for user application processing, plus more than 60 peripherals.

A Synopsys HAPS-54 prototype board was used to implement the prototype, as shown in Figure 3. The baseband processor was more than 5 million ASIC gates and Scott's team used Synopsys Certify tools to partition this into three of the Xilinx® Virtex®-5 FPGAs on the board, while the digital RF design was placed in the fourth FPGA. Freescale decided not to prototype the analog section but instead delivered cellular network data in digital form directly from an Anritsu protocol test box.

Older cores use some design techniques that are very effective in an ASIC, but they are not very FPGA friendly. In addition, some of the RTL was generated automatically from system-level design code, which can also be fairly unfriendly to FPGAs owing to overly complicated clock networks. Therefore, some modifications had to be made to the RTL to make it more FPGA compatible, but the rewards were significant.

Besides accelerating protocol testing, by the time Freescale engineers received first silicon they were able to:

- Release debugger software with no major revisions after silicon.
- Complete driver software.
- Boot up the SoC to the OS software prompt.
- Achieve modem camp and registration.

The Freescale team was able to reach the milestone of making a cellular phone call through the system only one month after receipt of first silicon, accelerating the product schedule by more than six months.

As Scott Constable said, "In addition to our stated goals of protocol testing, our FPGA system prototype delivered project schedule acceleration in many other areas, proving its worth many times over. And perhaps most important was the immeasurable human benefit of getting engineers involved earlier in the project schedule, and having all teams from design to software to validation to applications very familiar with the product six months before silicon even arrived. The impact of this accelerated product expertise is hard to measure on a Gantt chart, but may be the most beneficial.

"In light of these accomplishments, using an FPGA prototype solution to accelerate ASIC schedules is a no-brainer. We have since spread this methodology into the Freescale Network and Microcontroller Groups and also use prototypes for new IP validation, driver development, debugger development and customer demos."

This example shows how FPGA-based prototyping can be a valuable addition to the software team's tool box and brings significant return on investment in terms of product quality and project time scales.

INTERFACING BENEFIT: TEST REAL-WORLD DATA EFFECTS

It is hard to imagine an SoC design that does not comply with the basic structure of having input data upon which some processing is performed in order to produce output data. Indeed, if we push into the SoC design we will find numerous sub-blocks that follow the same structure, and so on down to the individual gate level.

Verifying the correct processing at each of these levels requires us to provide a complete set of input data and to observe that the correct output data are created as a result of the processing. For an individual gate this is trivial, and for small RTL blocks it is still possible.

However, as the complexity of a system grows it soon becomes statistically impossible to ensure completeness of the input data and initial conditions, especially when there is software running on more than one processor.

There has been huge research and investment in order to increase efficiency and coverage of traditional verification methods and to overcome the challenge of this complexity. At the complete SoC level, we need to use a variety of different verification methods in order to cover all the likely combinations of inputs and to guard against unlikely combinations.

This last point is important because unpredictable input data can upset all but the most carefully designed critical SoC-based systems. The very many possible previous states of the SoC coupled with new input data, or with input data of an unusual combination or sequence, can put an SoC into a nonverified state. Of course, that may not be a problem and the SoC recovers without any other part of the system, or indeed the user, becoming aware.

However, unverified states are to be avoided in final silicon and so we need ways to test the design as thoroughly as possible. Verification engineers use powerful methods such as constrained-random stimulus and advanced test harnesses to perform a wide variety of tests during functional simulations of the design, aiming to reach an acceptable coverage. However, completeness is still governed by the direction and constraints given by the verification engineers and the time available to run the simulations themselves. As a result, constrained-random verification is never fully exhaustive but it will greatly increase confidence that we have tested all combinations of inputs, both likely and unlikely.

In order to guard against corner-case combinations, we can complement our verification results with observations of the design running on an FPGA-based prototype running in the real world. By placing the SoC design into a prototype, we can run at a speed and accuracy that compare very well with the final silicon, allowing "soak" testing within the final ambient data, much as would be done with the final silicon.

One example of this immersion of the SoC design into a real-world scenario is the use made of FPGA-based prototyping at DS2 in Valencia, Spain.

EXAMPLE: IMMERSION IN REAL-WORLD DATA

Broadband-over-power line (BPL) technology uses normally undetectable signals to transmit and receive information over electrical-mains power lines. A typical use of BPL is to distribute HD video around a home from a receiver to any display via the mains wiring, as shown in Figure 4.

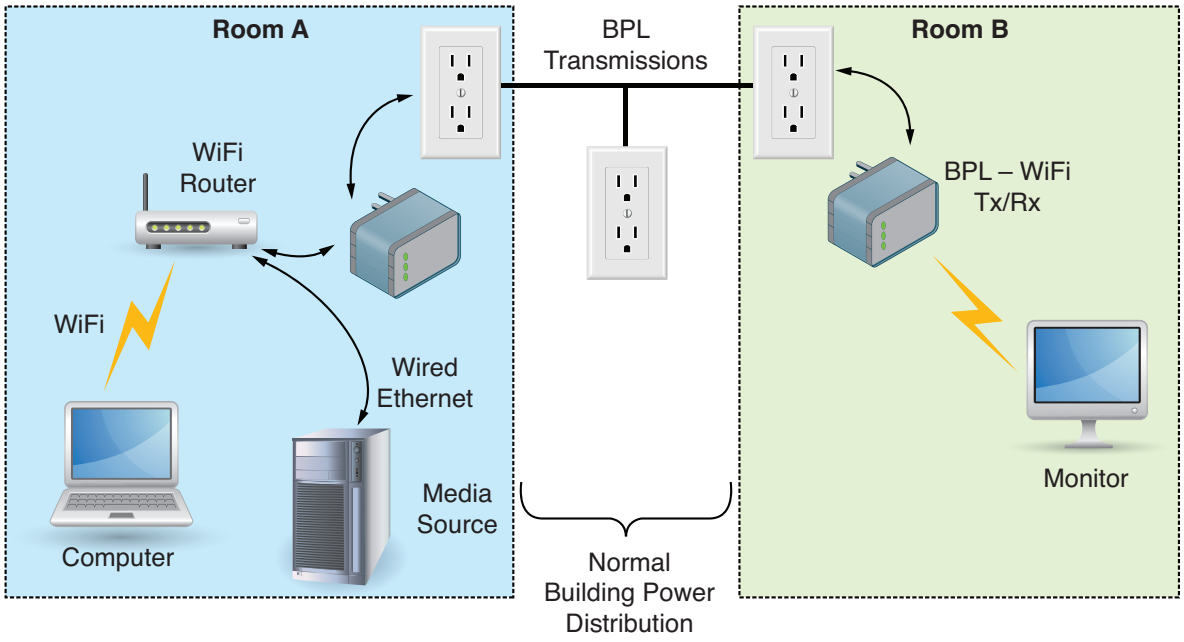


Figure 4 – Broadband-over-power line (BPL) technology used in a WiFi range extender

At the heart of DS2's BPL designs lie sophisticated hardware and embedded software algorithms, which encode and retrieve the high-speed transmitted signal into and out of the power lines. These power lines can be

very noisy electrical environments, so a crucial part of the development is to verify these algorithms in a wide variety of real-world conditions.

Javier Jimenez, ASIC design manager at DS2, explains what FPGA-based prototyping did for them: "It is necessary to use a robust verification technology in order to develop reliable and high-speed communications. It requires very many trials using different channel and noise models, and only FPGA-based prototypes allow us to fully test the algorithms and to run the design's embedded software on the prototype. In addition, we can take the prototypes out of the lab for extensive field testing. We are able to place multiple prototypes in real home and workplace situations, some of them harsh electrical environments indeed. We cannot consider emulator systems for this purpose because they are simply too expensive and are not portable."

This usage of FPGA-based prototyping outside of the lab is instructive because we see that making the platform reliable and portable is crucial to success.

BENEFITS FOR FEASIBILITY LAB EXPERIMENTS

At the beginning of a project, fundamental decisions are made about chip topology, performance, power consumption and on-chip communication structures. Some of these are best performed using algorithmic or system-level modeling tools, but some extra experiments could also be performed using FPGAs. Is this really FPGA-based prototyping? We are using FPGAs to prototype an idea but it is different than using algorithmic or mathematical tools because we need some RTL, perhaps generated by those high-level tools. Once in FPGA, however, early information can be gathered to help drive the optimization of the algorithm and the eventual SoC architecture. The extra benefit that FPGA-based prototypes bring to this stage of a project is that more accurate models can be used, which can run fast enough to interact with real-time inputs.

Experimental prototypes of this kind are worth mentioning, as they are another way to use FPGA-based prototyping hardware and tools in between full SoC projects, hence deriving further return on our investment.

PROTOTYPE USAGE OUT OF THE LAB

One truly unique aspect of FPGA-based prototyping for validating SoC design is its ability to work standalone. This is because the FPGAs can be configured, perhaps from a flash EEPROM card or other self-contained medium, without supervision from a host PC. The prototype can therefore run standalone and be used for testing the SoC design in situations quite different than those provided by other modeling techniques, such as emulation, which rely on host intervention.

In extreme cases, the prototype might be taken completely out of the lab and into real-life environments in the field. A good example of this might be the ability to mount the prototype in a moving vehicle and explore the dependency of a design to variations in external noise, motion, antenna field strength and so forth. For example, the authors are aware of mobile-phone baseband prototypes that have been placed in vehicles and used to make on-the-move phone calls through a public GSM network.

Chip architects and other product specialists need to interact with early-adopter customers and demonstrate key features of their algorithms. FPGA-based prototyping can be a crucial benefit at this very early stage of a project, but the approach is slightly different than the mainstream SoC prototyping.

Another very popular use of FPGA-based prototypes out of the lab is for preproduction demonstration of new product capabilities at trade shows. Let's consider a use of FPGA-based prototyping by the research-and-development division of the BBC in England (yes, that BBC) that illustrates both out-of-lab usage and use at a trade show.

EXAMPLE: A PROTOTYPE IN THE REAL WORLD

The powerful ability of FPGAs to operate standalone is demonstrated by a BBC R&D project to launch DVB-T2 in the United Kingdom. DVB-T2 is a new, state-of-the-art open standard that allows HD television to be broadcast from terrestrial transmitters.

The reason for using FPGA-based prototyping was that, like most international standards, the DVB-T2 technical specification took several years to complete, in fact 30,000 engineer-hours by researchers and technologists from all over the world. Only FPGAs gave the flexibility required in case of changes along the way. The specification was frozen in March 2008 and published three months later as a DVB Blue Book on June 26.

Because the BBC was using FPGA-based prototyping, in parallel with the specification work, a BBC implementation team, led by Justin Mitchell from BBC Research & Development, was able to develop a hardware-based modulator and demodulator for DVB-T2.

The modulator is based on a Synopsys HAPS-51 card with a Virtex-5 FPGA from Xilinx. The HAPS-51 card was connected to a daughtercard that was designed by BBC Research & Development. This daughtercard provided an ASI interface to accept the incoming transport stream. The incoming transport stream was then passed to the FPGA for encoding according to the DVB-T2 standard and passed back to the daughtercard for direct upconversion to UHF.

The modulator was used for the world's first DVB-T2 transmissions from a live TV transmitter that were able to start the same day that the specification was published.

The demodulator, also using HAPS as a base for another FPGA-based pro-

totype, completed the working end-to-end chain and this was demonstrated at the IBC exhibition in Amsterdam in September 2008, all within three months of the specification being agreed upon. This was a remarkable achievement and helped to build confidence that the system was ready to launch in 2009.

BBC Research & Development also contributed to other essential strands of the DVB-T2 project, including a very successful plugfest in Turin in March 2009, at which five different modulators and six demodulators were shown to work together in a variety of modes. The robust and portable construction of the BBC's prototype made it ideal for this kind of plugfest event.

Justin Mitchell had this to say about FPGA-based prototyping: "One of the biggest advantages of the FPGA was the ability to track late changes to the specification in the run-up to the transmission launch date. It was important to be able to make quick changes to the modulator as changes were made to the specification. It is difficult to think of another technology that would have enabled such rapid development of the modulator and demodulator and the portability to allow the modulator and demodulator to be used standalone in both a live transmitter and at a public exhibition."

WHAT CAN'T FPGA-BASED PROTOTYPING DO FOR US?

We started this chapter with the aim of giving a balanced view of the benefits and limitations of FPGA-based prototyping, so it is only right that we should highlight here some weaknesses to balance against the previously stated strengths

First and foremost, an FPGA prototype is not an RTL simulator. If our aim is to write some RTL and then implement it in an FPGA as soon as possible in order to see if it works, then we should think again about what is being bypassed. A simulator has two basic components; think of them as the engine and the dashboard. The engine has the job of stimulating the model and recording the results. The dashboard allows us to examine those results. We might run the simulator in small increments and make adjustments via our dashboard; we might use some very sophisticated stimulus—but that's pretty much what a simulator does. Can an FPGA-based prototype do the same thing? The answer is no.

It is true that the FPGA is a much faster engine for running the RTL "model," but when we add in the effort to set up that model, then the speed benefit is soon swamped. On top of that, the dashboard part of the simulator offers complete control of the stimulus and visibility of the results. We shall consider ways to instrument an FPGA in order to gain some visibility into the design's functionality, but even the most instrumented design offers only a fraction of the information that is readily available in an RTL simulator dashboard. The simulator is therefore a much better environment for repetitively writing and evaluating RTL code and so we should always wait until the simulation is mostly finished and the RTL is fairly mature before passing it over to the FPGA-based prototyping team.

AN FPGA-BASED PROTOTYPE IS NOT ESL

Electronic system-level (ESL) or algorithmic tools such as Synopsys' Innovator or Synphony allow designs to be entered in SystemC or to be built from a library of predefined models. We then simulate these designs in the same tools and explore their system-level behavior including running software and making hardware-software tradeoffs at an early stage of the project.

To use FPGA-based prototyping we need RTL, therefore it is not the best place to explore algorithms or architectures, which are not often expressed in RTL. The strength of FPGA-based prototyping for software is when the RTL is mature enough to allow the hardware platform to be built; then software can run in a more accurate and real-world environment. There are those who have blue-sky ideas and write a small amount of RTL for running in an FPGA for a feasibility study. This is a minor but important use of FPGA-based prototyping, but is not to be confused with running a system-level or algorithmic exploration of a whole SoC.

CONTINUITY IS THE KEY

Good engineers always choose the right tool for the job, but there should always be a way to hand over work-in-progress for others to continue. We should be able to pass designs from ESL simulations into FPGA-based prototypes with as little work as possible. Some ESL tools also have an implementation path to silicon using high-level synthesis, which generates RTL for inclusion in the overall SoC project. An FPGA-based prototype can take that RTL and run it on a board with cycle accuracy. But once again, we should wait until the RTL is relatively stable, which will be after completion of the project's hardware-software partitioning and architectural exploration phase.

SO WHY USE FPGA-BASED PROTOTYPING?

Today's SoCs are a combination of the work of many experts, from algorithm researchers to hardware designers, to software engineers, to chip layout teams—and each has its own needs as the project progresses. The success of an SoC project depends to a large degree on the hardware verification, hardware-software co-verification and software validation methodologies used by each of the above experts. FPGA-based prototyping brings different benefits to each of them.

For the hardware team, the speed of verification tools plays a major role in verification throughput. In most

SoC developments it is necessary to run through many simulations and repeated regression tests as the project matures. Emulators and simulators are the most common platforms used for that type of RTL verification. However, some interactions within the RTL or between the RTL and external stimuli cannot be re-created in a simulation or emulation, owing to long runtime, even when TLM-based simulation and modeling are used. FPGA-based prototyping is therefore used by some teams to provide a higher-performance platform for such hardware testing. For example, we can run a whole OS boot in relatively real-time, saving days of simulation time it would take to achieve the same thing.

For the software team, FPGA-based prototyping provides a unique presilicon model of the target silicon, fast and accurate enough to enable debug of the software in near-final conditions.

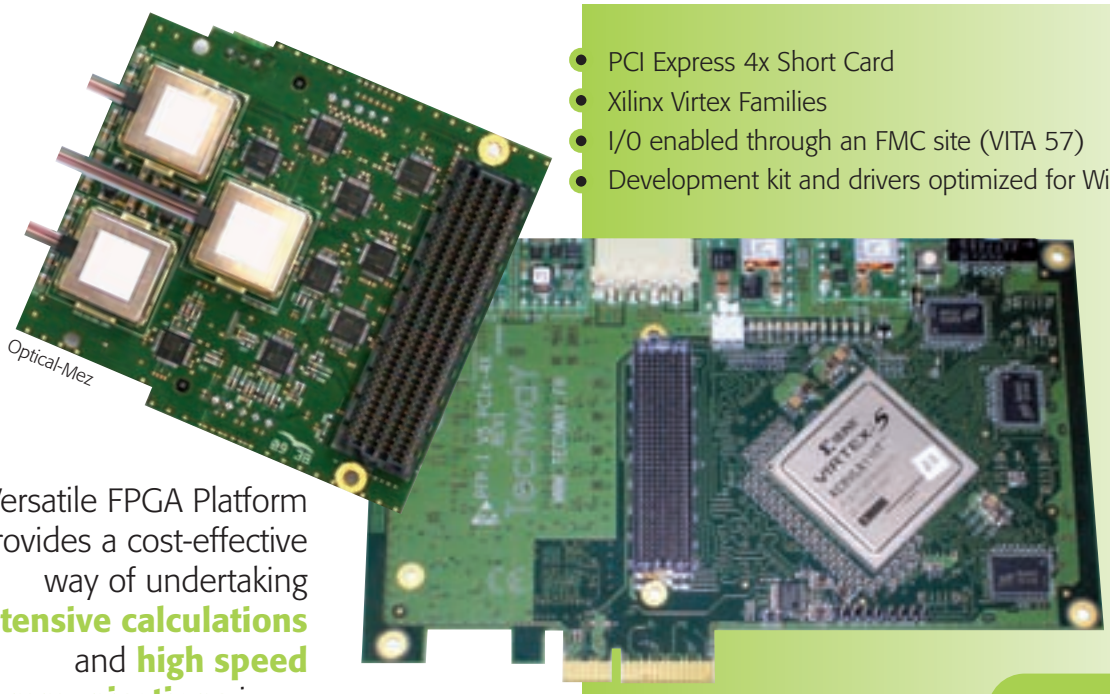
For the whole team, a critical stage of the SoC project is when the software and hardware are introduced to each other for the first time. The hardware will be exercised by the final software in ways that were not always envisaged or predicted by the hardware verification plan in isolation, exposing new hardware issues as a result. This is particularly prevalent in multicore systems or those running concurrent real-time applications. If this hardware-software introduction were to happen only after first silicon fabrication, then discovering new bugs at that time is not ideal, to put it mildly.

An FPGA-based prototype allows the software to be introduced to a cycle-accurate and fast model of the hardware as early as possible. SoC teams often tell us that the greatest benefit of FPGA-based prototyping is that when first silicon is available, the system and software are up and running in a day.

The authors gratefully acknowledge significant contribution to this chapter from Scott Constable of Freescale Semiconductor, Austin, Texas; Javier Jimenez of DS2, Valencia, Spain; and Justin Mitchell of BBC Research & Development, London. For further information, see www.synopsys.com/fpmm.

TechwaY
The way of innovation

Versatile FPGA Platform



- PCI Express 4x Short Card
- Xilinx Virtex Families
- I/O enabled through an FMC site (VITA 57)
- Development kit and drivers optimized for Windows and Linux

The Versatile FPGA Platform provides a cost-effective way of undertaking **intensive calculations** and **high speed communications** in an industrial environment.

www.techway.eu

ISE Design Suite 13

System Level Productivity with PlanAhead

With the release of Xilinx ISE® Design Suite 13, comes the powerful PlanAhead™ unified design environment and analysis tool. PlanAhead offers a push-button RTL-to-bitstream design flow with new and improved user interface and project management capabilities. In addition, PlanAhead software enables you to get the most out of your design through floor planning, multiple implementation runs, hierarchy exploration, quick timing analysis, and block based implementation.

Download ISE Design Suite 13, and start your design with PlanAhead today.

www.xilinx.com/planahead

ISE® DESIGN SUITE

XILINX®

© Copyright 2011. Xilinx, Inc. XILINX, the Xilinx logo, Artix, ISE, Kintex, Virtex, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Xilinx Tool & IP Updates

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 the flagship FPGA development environment, the ISE® Design Suite, as well as to Xilinx® IP, as of March 2011. Product updates offer significant enhancements and new features to three versions of the ISE Design Suite: the Logic, Embedded and DSP editions. Keeping your installation of ISE up to date is an easy way to ensure the best results for your design. Updates to the ISE Design Suite are available from the Xilinx Download Center at www.xilinx.com/download. *For more information or to download a free 30-day evaluation of ISE, visit* www.xilinx.com/ise.

Xilinx has developed an application called Documentation Navigator that allows users to view and manage Xilinx design documentation (software, hardware, IP and more) from one place, with easy-to-use download, search and notification features. To try the new Xilinx Documentation Navigator, now in open beta release, use the download link at *www.xilinx.com/support*.

ISE DESIGN SUITE: LOGIC EDITION

Front-to-Back FPGA Logic Design

Latest version number: 13.1
Date of latest release: March 2011
Previous release: 12.4
URL to download the latest patch:
www.xilinx.com/download

Revision highlights:

A newly redesigned user interface for PlanAhead™ and the IP suite improves productivity across SoC design teams and contributes to the progression toward true plug-and-play IP that targets Spartan®-6, Virtex®-6 and 7 series FPGAs, including the industry-leading Virtex-7 2000T device, with 2 million logic cells.

PlanAhead design and analysis tool: Xilinx has further enhanced PlanAhead's graphical user interface (GUI) to provide an intuitive environment for both new and advanced users. PlanAhead release 13 has power estimation available on Virtex-5, Virtex-6 and Spartan-6 device families. The GUI's new Clocking Resource view aids in the visualization and assignment of clocking-related sites. Also, PlanAhead release 13 integrates the Xilinx ISE Simulator (ISim) into the design flow and adds the ability to tag RTL nets using a new HDL Debug Probe feature. In addition, PlanAhead release 13 supports hierarchical design.

Team design: New to ISE Design Suite 13 is a team design methodology. Using PlanAhead, it addresses the challenge of multiple engineers working on a single project by providing a methodology for groups of developers to work in parallel. Building on the Design Preservation capability made available in ISE Design Suite 12, the team design flow provides additional functionality in allowing early implementation results on completed portions of the design to be locked down without having to wait for the rest of the design team. This new capability facilitates faster timing closure and timing preservation for the remainder of the design, increasing overall productivity and reducing design iterations.

ISE Simulator (ISim): PlanAhead release 13 has integrated the Xilinx ISE Simulator (ISim) into the design flow. The Flow Navigator provides access to this tool.

Project Navigator: Improvements in Embedded Development Kit (EDK) integration include support for multiple ELF files and automatic detection of ELF files referenced by EDK designs.

FPGA Editor: A new Lock Layers toolbar button locks the current layer's visibility settings for all zoom levels.

Xilinx Power Analyzer (XPA): This tool boasts improved vectorless algorithms for activity propagation.

iMPACT: The tool now supports SPI/BPI programming.

ISE DESIGN SUITE: EMBEDDED EDITION

Integrated Embedded Design Solution

Latest version number: 13.1
Date of latest release: March 2011
Previous release: 12.4
URL to download the latest patch:
www.xilinx.com/download

Revision highlights:

All ISE Design Suite editions include the enhancements listed above for the Logic Edition. The following

enhancements are specific to the Embedded Edition.

Project Navigator: Usability improvements make it possible to select a Xilinx board during project creation. Software ELF files for simulation and implementation can now be source files. Multiple processor instances are now recognized and displayed in the source window. Designers can also export hardware design capability to the SDK and launch the SDK from Project Navigator.

Xilinx Platform Studio (XPS): This tool also has usability improvements, including a wizard for custom AXI slave peripheral creation, testbench generation with AXI BFM for ModelSim, ISim and NC Sim, and automatic bus connections to easily create systems with multiple MicroBlaze™ processors. Moreover, AXI systems are now the default in Base System Builder for Spartan-6 and Virtex-6 designs. Base System Builder supports AXI systems only for 7 series designs.

Configuration wizard for Zynq-7000: The wizard allows the user to select and configure peripherals.

EDK overall enhancements: The Embedded Development Kit now offers consistent SDK workspace selection behavior across Project Navigator, Xilinx Platform Studio (XPS) and the SDK, along with TDP device-based licensing support.

SDK enhancements: The Software Development Kit offers initial support of the 7 series in the Xilinx Microprocessor Debugger (XMD).

Project Navigator/EDK integration: Enhancements include recognition of processor instances in .xmp and separate ELF sources for implementation and simulation.

MicroBlaze: The new version 8.10.a of the embedded processor supports 7 series Virtex devices. AXI is now the default interface for 7 series designs. Also, the MicroBlaze configuration wizard supports fault-tolerant features.

ISE DESIGN SUITE: DSP EDITION

For High-Performance DSP Systems

Latest version number: 13.1
Date of latest release: March 2011
Previous release: 12.4
URL to download the latest patch:
www.xilinx.com/download

Revision highlights:
All ISE Design Suite editions include the enhancements listed above for the Logic Edition. Enhancements specific to the DSP Edition deliver a 33 percent improvement in first-time initialization of a typical model and a 1.5x to 3x improvement in simulation speed on typical models. The DSP Edition now also supports the fast simulation model of the AXI4 fast Fourier transform, for a 42x speedup.

System Generator for DSP: This tool supports MATLAB®/Simulink® 2011a. All System Generator blocks now support Kintex-7 and Virtex-7 devices. New blocks include the 7 series DSP48E1, Complex Multiply 5.0, DSP48 Macro 2.1, FIR Compiler 6.2 and VDMA Interface 3.0. Also new in this release is System Generator support for the AXI PCore and for hardware co-simulation.

XILINX IP UPDATES

Name of IP: ISE IP Update 13.1
Type of IP: All

Targeted application: Xilinx develops IP cores and partners with third-party IP providers to decrease customer time-to-market. The powerful combination of Xilinx FPGAs with IP cores provides functionality and performance similar to ASSPs, but with flexibility not possible with ASSPs.

Latest version number: 13.1
Date of latest release: March 2011
URL to access the latest version:
www.xilinx.com/download

Informational URL:
www.xilinx.com/ipcenter/coregen/updates_13_1.htm

Installation instructions:
www.xilinx.com/ipcenter/coregen/ip_update_install_instructions.htm

Listing of all IP in this release:
www.xilinx.com/ipcenter/coregen/13_1_datasheets.htm

Revision highlights:
Starting with Release 13.1, all ISE CORE Generator™ IP supports Kintex™-7 and Virtex-7 devices.

Audio, video and image processing IP: Object segmentation v1.0 (AXI4-Lite), used in conjunction with the image-characterization LogiCORE™ IP, converts statistical data provided into a list of objects that meet a user-defined set of object characteristics. AXI video direct memory access v1.0 (AXI4, AXI4-Stream, AXI4-Lite) provides a flexible interface for controlling and synchronizing video frame stores from external memory. Designers can link together multiple VDMAs from different clock domains to control frame store reads and writes from multiple sources.

Communication DSP building blocks: The Linear Algebra Toolkit v1.0 (AXI4-Stream) implements basic matrix operations including matrix-matrix addition, subtraction, matrix-scalar multiplication and matrix-matrix multiplication. This IP provides flexible and optimized building blocks for developing complex composite functions for various signal- and data-processing applications.

FPGA features and support: The 7 series FPGA transceiver wizard v1.3 configures one or more Virtex-7 and Kintex-7 FPGA GTX transceiver, either from scratch or using industry-standard templates, by means of a custom Verilog or VHDL wrapper. The wizard also provides an example design, testbench and scripts that allow you to observe the transceivers operating in simulation and in hardware. The XADC wizard v1.2 generates an HDL wrapper to configure a single 7 series FPGA XADC primitive for user-specified channels and alarms.

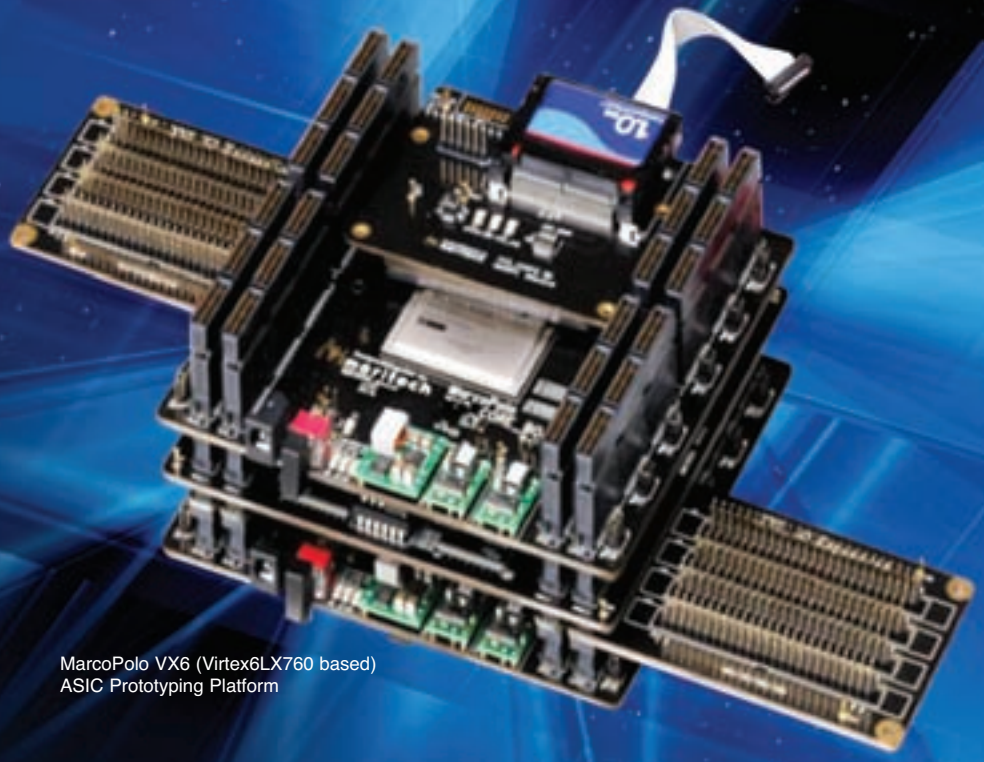
Standard bus interfaces and I/O: The 7 series integrated block for PCI Express® (PCIe®) v1.0 (AXI4-Stream) implements one-, two-, four- or eight-lane configurations. The IP uses the 7 series' integrated hard-IP block for PCI Express in conjunction with flexible architectural features to implement a PCI Express Base Specification v2.1-compliant PCIe endpoint or root port. The LogiCORE IP for PCI Express employs the high-performance AXI interface. It uses optimal buffering for high-bandwidth applications, along with BAR checking and filtering.

Wireless IP: Triple-rate SDI v1.0 (AXI4-Stream) IP provides receiver and transmitter interfaces for the SMPTE SD-SDI, HD-SDI and 3G-SDI standards. The triple-rate SDI receiver and transmitter come as unencrypted source code in both Verilog and VHDL, allowing you to fully customize these interfaces as required by your specific applications. The 3GPP LTE PUCCH Receiver v1.0 (AXI4-Stream) provides designers with an LTE Physical Uplink Control Channel receiver block for the 3GPP TS 36.211 v9.0.0 Physical Channels and Modulation (Release 9) specification. The receiver block supports channel estimation, demodulation and decoding.

Additional IP supporting AXI4 interfaces: Xilinx has updated the latest versions of CORE Generator IP with production AXI4 interface support. For more detailed support information, see *www.xilinx.com/ipcenter/axi4_ip.htm*. In general, the AXI4 interface will be supported by the latest version of an IP block, for Virtex-7, Kintex-7, Virtex-6 and Spartan-6 device families. Older production versions of IP will continue to support the legacy interface for the respective core on Virtex-6, Spartan-6, Virtex-5, Virtex-4 and Spartan-3 device families only. For general information on Xilinx AXI4 support see *www.xilinx.com/axi4.htm*. You can view a comprehensive listing of cores updated in this release at *www.xilinx.com/ipcenter/coregen/13_1_datasheets.htm*. For more information on LogiCORE IP in this release, see *www.xilinx.com/ipcenter/coregen/updates_13_1.htm*.

CORE Generator and PlanAhead design flow enhancements: CORE Generator and PlanAhead now support IP-XACT-based IP repositories for Xilinx and Alliance Program member IP. (Their use requires no changes to existing CORE Generator, PlanAhead and Project Navigator user flows.) A new Manage IP pull-down menu provides a repository and IP management features. Display of AXI4 support by each core in the IP catalog now places the various AXI4 interfaces in separate sortable columns: AXI4, AXI4-Stream and AXI4-Lite. Individual ports in IP symbols can now be grouped into AXI4 channels for simplified symbol views. In this release, PlanAhead also adds support for an automatic IP upgrade flow.

Still wondering how to build your own ASIC prototyping platform quickly?



MarcoPolo VX6 (Virtex6LX760 based)
ASIC Prototyping Platform

Here we have *MarcoPolo VX* series, the market proven solution!

Have you ever dreamed of a more flexible FPGA design platform based on Xilinx Virtex-6 LX760 or Virtex-5 LX330? Have you ever experienced the shortcomings of most conventional platforms available today?

- Redundancy FPGAs?
- Restricted FPGA interconnections?
- Not enough from the conventional platform support?

Meritech is proud to introduce ***MarcoPolo*** platform which solves these common issues, and is revolutionizing the way design engineers prototype FPGA based IP verification systems. It is already adopted by Samsung Electronics and world MP3 & SmartPhone OEM industry-leaders. ***MarcoPolo*** is quickly becoming industry standard development platform.

Suitable for projects ranging from commercial grade SoC through leading-edge high speed FPGA systems. You will be impressed by the flexibility, convenience and power of ***MarcoPolo*** platform. If you are about to embark on the design of your next-generation system, we urge you to first learn more about how ***MarcoPolo*** solution can save your time and money.

Life just got easier…. Just click on www.marcopolofpga.co.kr

meritech
an alliance of Xilinx

www.meritech.co.kr 400-14 Singaldong, Giheunggu, Yongincity, Kyungkido Korea, 446-599
(82)708233-6655 or (82)104481-4818 e-mail: sales@meritech.co.kr or sean@meritech.co.kr

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 and the white paper.

FEATURED WHITE PAPER: LOWERING POWER AT 28 NM WITH XILINX 7 SERIES FPGAS

http://www.xilinx.com/support/documentation/white_papers/wp389_Lowering_Power_at_28nm.pdf#MarNL

This new white paper by Jameel Hussein, Matt Klein and Michael Hartwhich discusses several aspects of power related to the Xilinx® 28-nanometer 7 series FPGAs, including the TSMC 28-nm high-k metal gate (HKMG), high-performance, low-power (HPL) process choice. The paper describes the power benefits of the 28-nm HPL process and its usefulness across Xilinx's full range of product offerings, as well as the architectural innovations and features for power reduction across the dimensions of static power, dynamic power and I/O power.

Xilinx has achieved a dramatic power reduction in the 7 series by applying a holistic approach to curbing FPGA and system power. The FPGAs offer up to 50 percent total power reduction—and an even greater power reduction at maximum (worst-case) process—compared with the previous generation. Designers can further cut power by leveraging new I/O features and advanced clock- and logic-gating software, resulting in the industry's lowest total FPGA power ratings.

XAPP883: FAST CONFIGURATION OF PCI EXPRESS TECHNOLOGY THROUGH PARTIAL RECONFIGURATION

http://www.xilinx.com/support/documentation/application_notes/xapp883_Fast_Config_PCIe.pdf

The PCI Express® specification requires ports to be ready for link training at a minimum of 100 milliseconds after the power supply is stable. This becomes a difficult task due to the ever-increasing configuration memory size of each new generation of FPGA, such as the Xilinx Virtex®-6 family. One innovative approach to addressing this challenge is leveraging the advances made in the area of FPGA partial reconfiguration to split the overall configuration of a PCIe® specification-based system in a large FPGA into two sequential steps: initial PCIe system link configuration and subsequent user application reconfiguration.

As Simon Tam and Martin Kellermann explain in this application note, it is feasible to configure only the FPGA PCIe system block and associated logic during the first stage within the 100-ms window before the fundamental reset is released. Using the Virtex-6 FPGA's partial-reconfiguration capability, the host can then reconfigure the FPGA to apply the user application via the now-active PCIe system link.

This methodology not only provides a solution for faster PCIe system configuration, it enhances user application security because the bitstream is accessible only by the host and can be better encrypted. This approach also helps lower the system cost by reducing external configuration component costs and board space.

This application note describes the methodology for building a Fast PCIe Configuration (FPC) module using this two-step configuration approach. A reference design is available to help designers quick-launch a PlanAhead™ software partial-reconfiguration project. The reference design implements an eight-lane PCIe technology generation-1 link and targets the Virtex-6 FPGA ML605 Evaluation board.

XAPP1151: PARAMETERIZABLE CONTENT-ADDRESSABLE MEMORY

http://www.xilinx.com/support/documentation/application_notes/xapp1151_Param_CAM.pdf

This application note by Kyle Locke describing a parameterizable content-addressable memory (CAM) is accompanied by a reference design that replaces the CAM core previously delivered through the CORE Generator™ software. Designers should use this CAM reference design for all new FPGA designs targeting Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3, Spartan-3E, Spartan-3A and Spartan-3A DSP FPGAs, and newer architectures. All the features and interfaces included in the reference design are backward compatible with the LogiCORE™ IP CAM v6.1 core. In addition, because the reference design is provided in plain-text VHDL format, the implementation of the function is fully visible, allowing for easy debug and modification of the code.

Unlike standard memory cores, a CAM performs content matching rather than address matching. Using content values as an index into a database of address values, the content-matching approach enables faster data searches than are possible by sequentially checking each address location in a standard memory for a particular value. The additional ability to compare content in parallel enables even higher-speed searches. A set of scripts included with the CAM reference design allows the customization of width, depth, memory type and optional features.

You can configure the CAM using one of two memory implementations: SRL16E-based CAM with a 16-clock-cycle write operation and a one-clock-cycle search operation, or BRAM-based CAM with only a two-clock-cycle write operation and a one-clock-cycle search operation. The BRAM-based CAM also supports an optional additional output register that adds a latency of one clock cycle to all read operations.

XAPP19: WRITING EFFICIENT TESTBENCHES

http://www.xilinx.com/support/documentation/application_notes/xapp199.pdf

This application note is written for logic designers who are new to HDL verification flows, and who do not have extensive experience writing testbenches to verify HDL designs. Author Mujtaba Hamid provides guidelines for laying out and constructing efficient testbenches, along with an algorithm to develop a self-checking testbench for any design.

Due to increases in design size and complexity, digital design verification has become an increasingly difficult and laborious task. To meet this challenge, verification engineers rely on several tools and methods. For large, multimillion-gate designs, engineers typically use a suite of formal verification tools. However, for smaller designs, design engineers usually find that HDL simulators with testbenches work best. Thus, testbenches have become the standard method to verify high-level language designs. Typically, testbenches perform the following tasks:

- Instantiate the design under test (DUT)
- Stimulate the DUT by applying test vectors to the model
- Output results to a terminal or waveform window for visual inspection
- Optionally compare actual results to expected results

Typically, designers write testbenches in the industry-standard VHDL or Verilog hardware description languages. Testbenches invoke the functional design, then stimulate it. Complex testbenches perform additional functions—for example, they contain logic to determine the proper design stimulus for the design or to compare actual to expected results.

Testbenches provide engineers with a portable, upgradable verification flow. With the availability of mixed-language simulators, designers are free to use their HDL language of choice to verify both VHDL and Verilog designs. High-level behavioral languages facilitate the creation of testbenches that use simple constructs and require a minimum amount of source code.

This application note describes the structure of a well-composed testbench and provides an example of a self-checking testbench—one that automates the comparison of actual to expected testbench results. Designs benefit from self-checking testbenches that automate the verification of correct design results during simulation.

XAPP1076: IMPLEMENTING TRIPLE-RATE SDI WITH SPARTAN-6 FPGA GTP TRANSCEIVERS

http://www.xilinx.com/support/documentation/application_notes/xapp1076_S6GTP_TripleRateSDI.pdf

The triple-rate serial digital interface (SDI) supporting the SMPTE SD-SDI, HD-SDI and 3G-SDI standards enjoys wide use in professional broadcast video equipment. Broadcast studios and video production centers use it to carry uncompressed digital video, along with embedded ancillary data, such as multiple audio channels.

Spartan-6 FPGA GTP transceivers are well-suited for implementing triple-rate SDI receivers and transmitters, providing a high degree of performance and reliability in a low-cost device. In this application note, Reed Tidwell describes the triple-rate SDI receiver and transmitter reference designs for the GTP transceivers in Spartan-6 devices.

The Spartan-6 FPGA triple-rate SDI reference design supports SD-SDI, HD-SDI and 3G-SDI (both level A and level B). The Spartan-6 FPGA GTP transceiver triple-rate SDI transmitter has a 20-bit GTP interface, supported on Spartan-3 and faster Spartan-6 devices. Only two reference clock frequencies are required to support all SDI modes: 148.5 MHz (for SD-SDI at 270 Mbits/s, HD-SDI at 1.485 Gbits/s and 3G-SDI at 2.97 Gbits/s); and 148.5/1.001 MHz (for HD-SDI at 1.485/1.001 Gbits/s and 3G-SDI at 2.97/1.001 Gbits/s). The transmitter directly supports 3G-SDI level A transmission of 1080p 50-Hz, 59.94-Hz and 60-Hz video as well as preformatted dual-link HD-SDI streams via either dual-link HD-SDI or 3G-SDI level B formats.

The receiver has a bit rate detector that can distinguish between the two HD-SDI and two 3G-SDI bit rates. An output signal from the receiver indicates which bit rate is being received. The receiver automatically detects the SDI standard of the input signal (3G-SDI, HD-SDI or SD-SDI) and reports the current SDI mode on an output port. It detects and reports the video transport format (e.g., 1080p 30 Hz, 1080i 50 Hz), and supports both 3G-SDI level A and level B formats, automatically detecting whether 3G-SDI data streams are level A or B. The receiver performs CRC error checking for HD-SDI and 3G-SDI modes.

XAPP978: FPGA CONFIGURATION FROM FLASH PROMS ON THE SPARTAN-3E 1600E BOAR

http://www.xilinx.com/support/documentation/application_notes/xapp978.pdf

This application note describes three FPGA configuration modes using flash PROMs: BPI Up mode, BPI Down mode and SPI mode. Author Casey Cain details the step-by-step process to boot-load a software application in flash in each of the configuration modes. The process includes creating the boot loader, programming the software application files and the system configuration into flash memory, and setting which configuration mode the FPGA will use when the board is powered on. A reference system targeted for the Spartan-3E 1600E development board is configured for programming the Intel StrataFlash parallel NOR flash PROM on the SP3E1600E board. Sample software applications illustrate the boot-loading process.

Xpress Yourself in Our Caption Contest

DANIEL GUIDERA

If you have a yen to Xercise your funny bone, here's your opportunity. We invite readers to step up to our verbal challenge and submit an engineering- or technology-related caption for this editorial cartoon by Daniel Guidera, featuring a meeting room presentation by an elephant. The image might inspire a caption like "He can't program worth a damn but he has an incredible memory for circuit theory."

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* and award the winner the new Xilinx® SP601 Evaluation Kit, our entry-level development environment for evaluating the Spartan®-6 family of FPGAs (approximate retail value, $295; see *http://www.xilinx.com/sp601*). Runners-up will gain notoriety, fame and a cool, Xilinx-branded gift from our SWAG closet.

The deadline for submitting entries is 5:00 pm Pacific Time (PT) on July 1, 2011. So, get writing!

Austin, Texas-based hardware engineer LARRY A. STELL won an SP601 Evaluation Kit with his caption for the cartoon of the overactive office chair in Issue 74 of *Xcell Journal*.

DANIEL GUIDERA

"Now, that's the worst case of ground bounce I've ever seen!"

Congratulations as well to our two runners-up:

"HR just found out he turned 55 today."

– Vinay Agarwala, MTS, SeaMicro Inc. (Santa Clara, Calif.)

"While developing the DSP algorithm for the motor speed control of the new massage-chair project, Rodney learns the hard way what can happen when an unexpected accumulator overflow occurs."

– Steven Spain, hardware design engineer, Kay Pentax Corp. (Lincoln Park, NJ)

NO PURCHASE NECESSARY. You must be 18 or older and a resident of the fifty United States, the District of Columbia, or Canada (excluding Quebec) to enter. Entries must be entirely original and must be received by 5:00 pm Pacific Time (PT) on July 1, 2011. Official rules available online at *www.xilinx.com/xcellcontest*. Sponsored by Xilinx, Inc. 2100 Logic Drive, San Jose, CA 95124.

XILINX

SYNOPSYS®
Predictable Success

Now Available!

The industry's first methodology manual for FPGA-based prototyping of SoC Designs

FPGA-Based Prototyping Methodology Manual:

Best Practices in Design-for-Prototyping

Go to:
http://www.synopsys.com/fpmm

- Purchase the book or download a free ebook version
- Become involved with the online community for prototypers

Introducing ZYNQ, the new element in processing. Finally, the processor comes together with the FPGA in a fully extensible processing platform called Zynq.™ More intuitive to program in the way you already know. Fully customizable to your requirements. Faster to implement and get to market. As a software engineer, if you know ARM® Cortex,™ you already know Zynq. And if you know Xilinx, you already know this is innovation you can count on. Visit us at www.xilinx.com

XILINX
ZYNQ™

© Copyright 2011. Xilinx, Inc. XILINX, the Xilinx logo, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners. ARM is the registered trademark of ARM Limited in the EU and other countries. Cortex is the trademark of ARM Limited in the EU and other countries.

PN 2485