ISSUE 82, FIRST QUARTER 2013

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

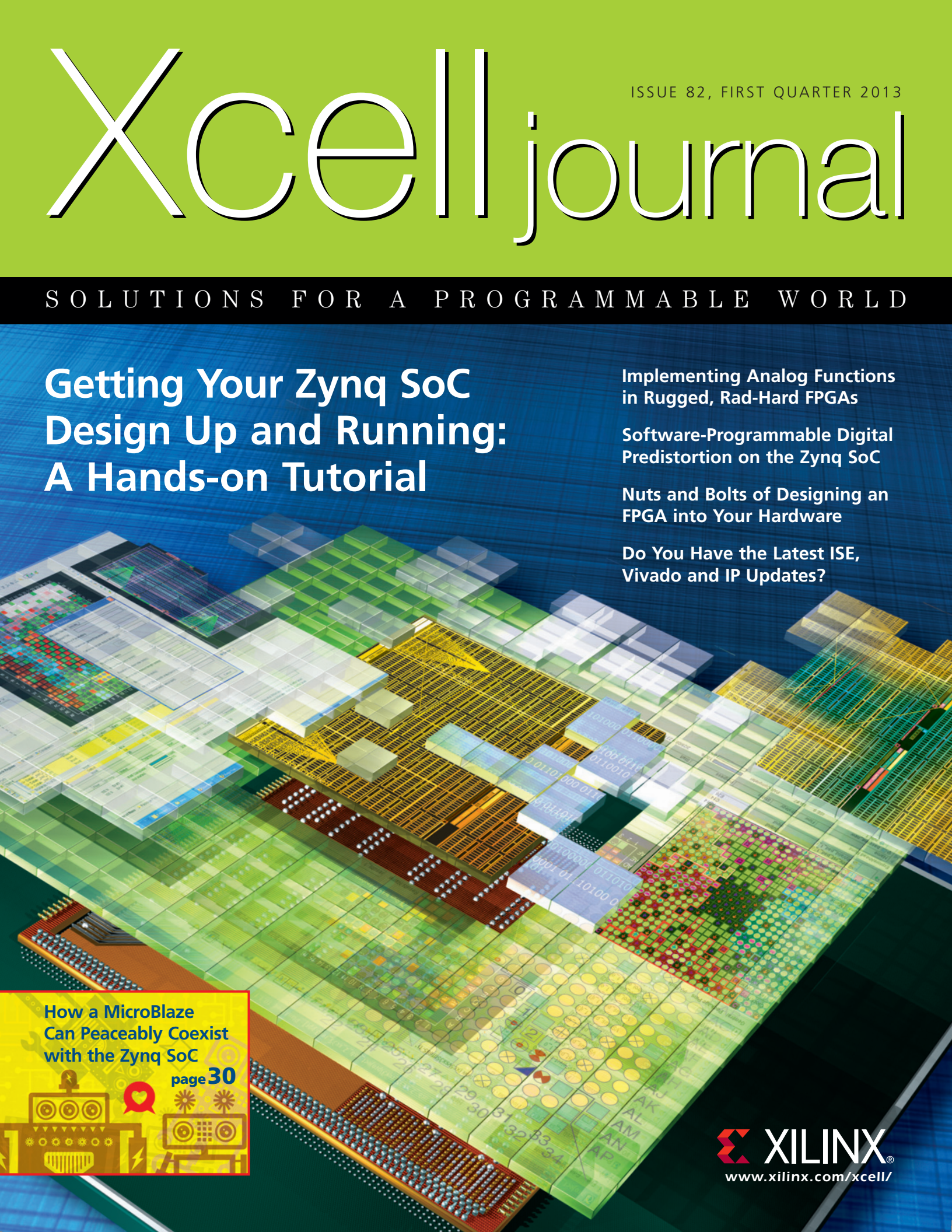
Getting Your Zynq SoC Design Up and Running: A Hands-on Tutorial

Implementing Analog Functions in Rugged, Rad-Hard FPGAs

Software-Programmable Digital Predistortion on the Zynq SoC

Nuts and Bolts of Designing an FPGA into Your Hardware

Do You Have the Latest ISE, Vivado and IP Updates?



How a MicroBlaze Can Peaceably Coexist with the Zynq SoC page 30

XILINX®

www.xilinx.com/xcell/

AVNET electronics marketing

www.zedboard.org

ZedBoard™

Avnet Electronics Marketing presents a new series of video-based SpeedWay Design Workshops™, featuring the new Xilinx Zynq™-7000 All Programmable SoC Architecture. This workshop series includes three online courses progressing from introductory, through running a Linux operating system on the Zynq-7000 SoC, and finally to the integration of the high-speed analog signal chain with digital signal processing from RF to baseband for wireless communications.

SPEEDWAY DESIGN WORKSHOPS™

www.zedboard.org/trainings-and-videos

Accelerating Your Success™

MathWorks® ANALOG DEVICES XILINX ALL PROGRAMMABLE™

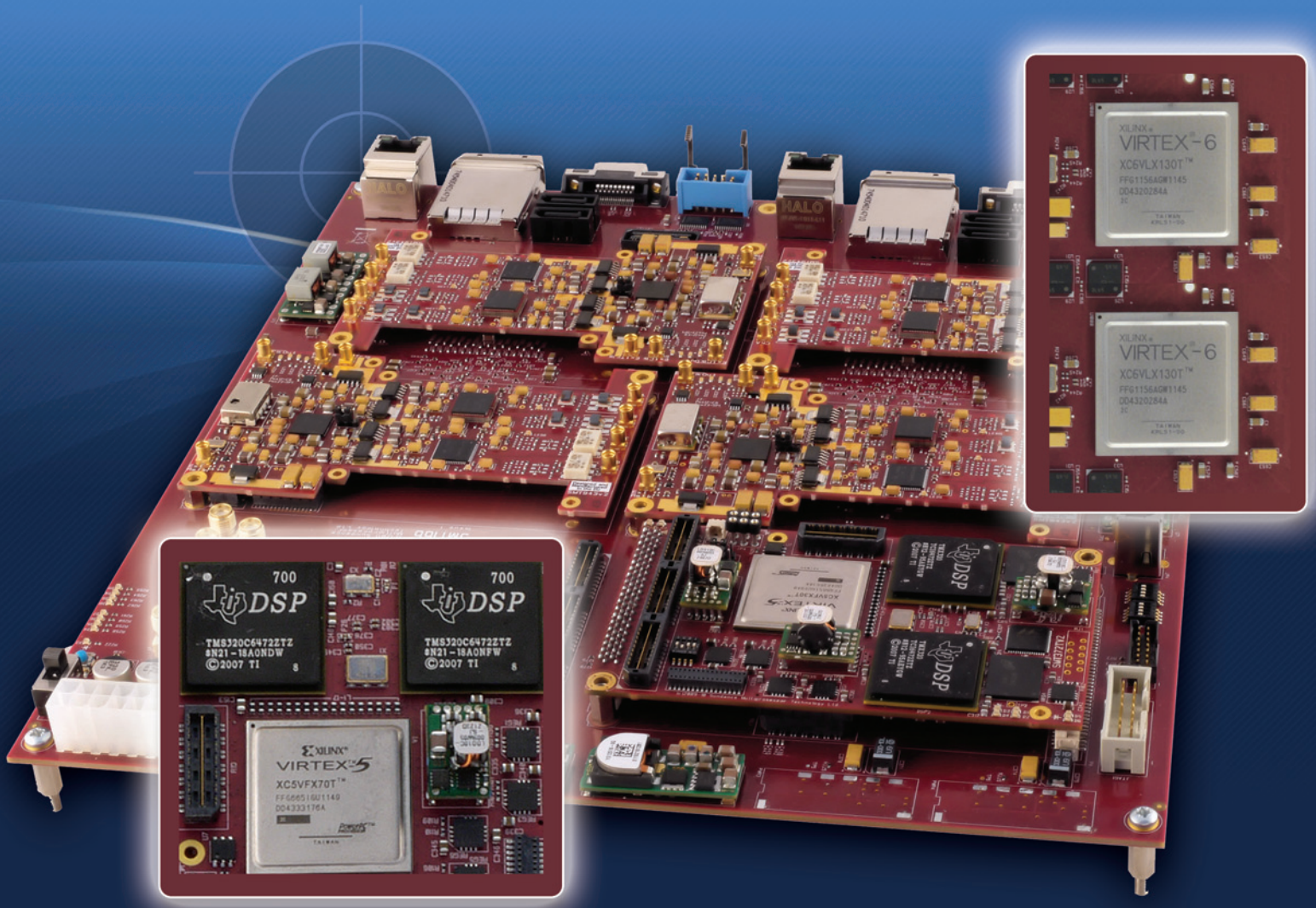
Sundance Multiprocessor Technology

embedded signal processing solutions

SUNDANCE

FlexTiles Development Platform

Self-Adaptive Heterogeneous MultiCore Solutions based on DSPs and FPGAs



www.FlexTiles.biz

Test & Measurement instruments • Communications, MIMO & Satellite receivers • RADAR ECM systems
Space, Land, Sea, Air equipments • Research, Science & Technology • Industrial control & video solutions

Xcell journal

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

EDITOR Jacqueline Damian

ART DIRECTOR Scott Blair

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

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

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

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

Tomoko Suto, Japan
tomoko@xilinx.com

REPRINT ORDERS 1-800-493-5551

Xcell PUBLICATIONS
www.xilinx.com/xcell/

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124-3400
Phone: 408-559-7778
FAX: 408-879-4780
www.xilinx.com/xcell/

Getting Real with the Zynq-7000 All Programmable SoC

If you have been a regular reader of *Xcell Journal* over the last five years, you've followed the development and deployment of Xilinx's 28-nanometer line of All Programmable devices and our next-generation software suite, Vivado™. This massive effort, which was kicked off by CEO Moshe Gavrielov in 2008 and executed by the hardworking folks here at Xilinx, is now starting to pay dividends.

As you've read in past issues, back in 2008 Xilinx's management decided to make several breakout moves starting with the 28-nm node. Instead of just producing FPGAs with twice the capacity and faster and more transceivers, the company expanded its portfolio by adding the Zynq™-7000 All Programmable SoC, while also pioneering homogeneous and heterogeneous 3D ICs. Xilinx was the first to commercially release such 3D parts, a generation ahead of the competition.

Since the effort began, we here at Xilinx have been very excited about the prospects of the 28-nm lineup, especially the Zynq-7000 SoC. The device pairs a 28-nm ARM® dual-core Cortex™-A9 MPCore™ that runs at up to 1 GHz with programmable logic—all on the same chip. As someone who has covered the IC design space for many years, after hearing about the 28-nm architecture plans, I couldn't help but think the Zynq-7000 was a stroke of semiconductor business genius—truly the right device at the right time.

Today we are seeing how all this hard work is beginning to pay off in the real world. The Zynq-7000 All Programmable SoC (previously called the Extensible Processing Platform) has received top honors from several of the top trade publications and engineering organizations. It was named the SoC Product of the year in 2011 by CMP Media (now called UBM, the publisher of *EE Times* and *EDN*), and won the IET Innovation Award as well as the Embedded System Product of the Year Award from Elektra (the European Electronics Industry) that same year. In 2012, *Electronic Products Magazine* named the Zynq-7000 its Product of the Year and *The Microprocessor Report* gave the device its Analyst Choice Award.

But by far the most rewarding part of the rollout has been seeing how excited you—Xilinx customers—are about the Zynq All Programmable SoCs and how today you are actively designing with them. In this issue of *Xcell Journal*, you will read several articles about how to implement Zynq-7000 designs, starting with the cover story, a hands-on tutorial by EADS Astrium principal engineer Adam Taylor.

In addition to these feature articles, there are a growing number of other informative resources in which your colleagues are actively sharing their Zynq-7000 All Programmable SoC design methods and experiences. Two of these resources are Avnet's ZedBoard.org (*www.zedboard.org*) and UBM's All Programmable Planet (*www.programmableplanet.org*). I encourage you to not only read and learn from these sources, but to actively participate in the discussions at both sites. And of course, Xilinx's official training group (*http://www.xilinx.com/training/index.htm*) and Xilinx User Community Forums (*http://forums.xilinx.com/*) are fabulous resources as well.

If you don't have a Zynq-7000 yet, contact Avnet to get a ZedBoard (*http://www.em.avnet.com/en-us/design/drc/Pages/Zedboard.aspx*). And if you feel lucky, (and live in North America, excluding Quebec), you can try your hand at writing a witty caption for our caption contest on page 66. The person who writes the funniest caption, according to our judges and in compliance with the contest rules, will win a new ZedBoard, and his or her caption will run in the next issue of *Xcell Journal*.

Get interactive, get designing and have fun innovating. Oh, and if you feel like writing about your experiences for an upcoming *Xcell Journal*, please feel free to contact me.

Mike Santarini
Publisher

Get Published

Would you like to write for Xcell Publications? It's easier than you think.

Interested in adding "published author" to your resume and achieving a greater level of credibility and recognition in your peer community? Consider submitting an article for global publication in the highly respected, award-winning *Xcell Journal*.

The *Xcell* team regularly guides new and experienced authors from idea development to published article with an editorial process that includes planning, copy editing, graphics development, and page layout. Our author guidelines and article template provide ample direction to keep you on track and focused on how best to present your chosen topic, be it a new product, research breakthrough, or inventive solution to a common design challenge.

Our design staff can even help you turn artistic concepts into effective graphics or redraw graphics that need a professional polish.

We ensure the highest standards of technical accuracy by communicating with you throughout the editorial process and allowing you to review your article to make any necessary tweaks.

Submit final draft articles for publication and we will assign an editor and a graphics artist to work with you to make your papers clear, professional, and effective.

For more information on this exciting and highly rewarding opportunity, please contact:

Mike Santarini
Publisher, Xcell Publications
xcell@xilinx.com

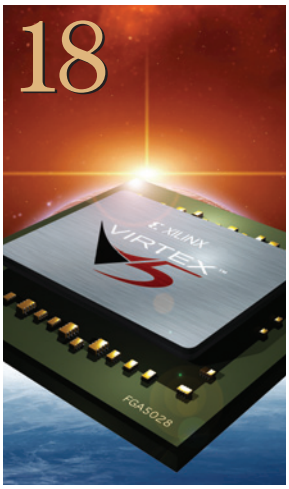
Xcell Publications

www.xilinx.com/xcell/

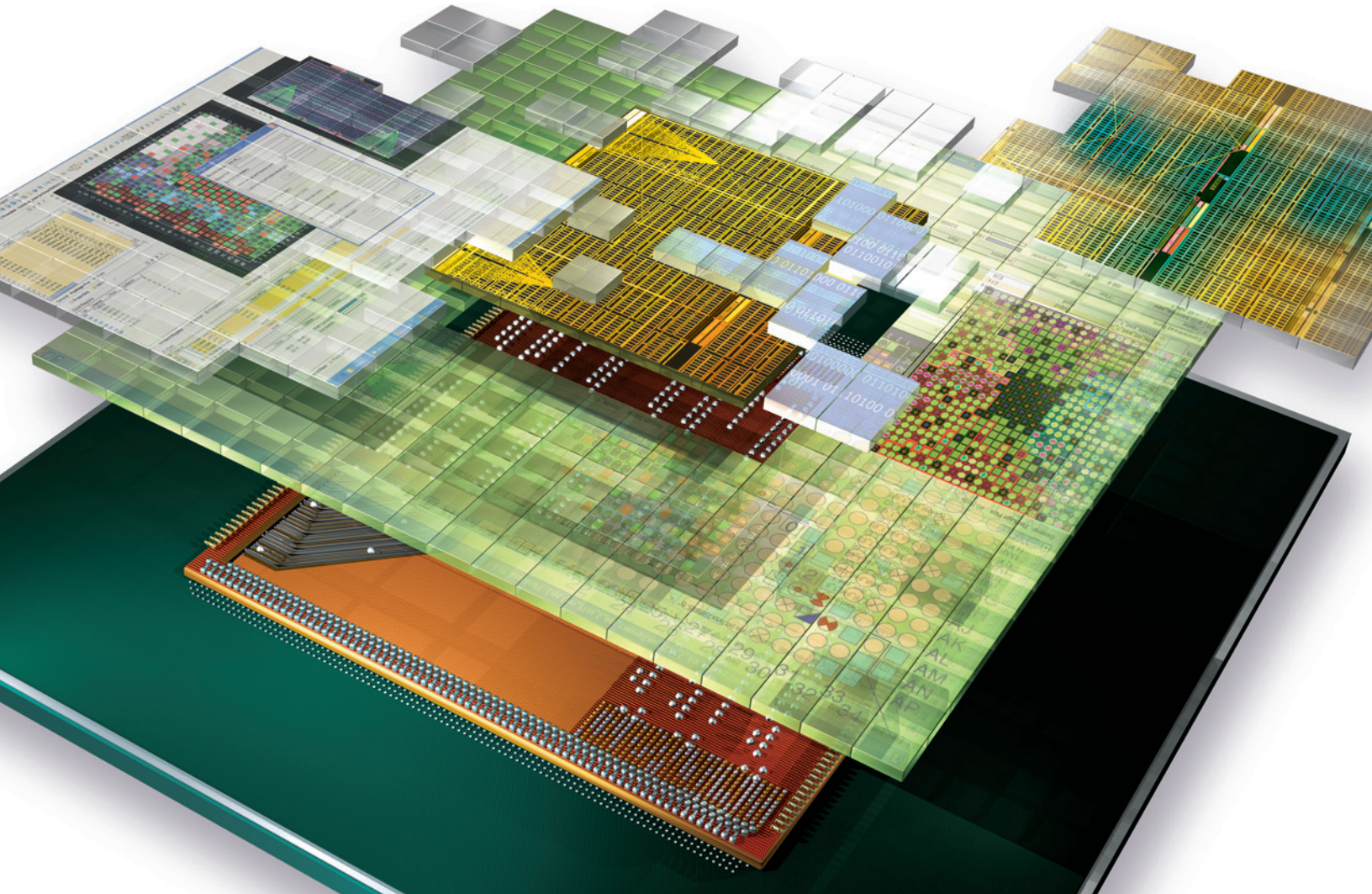
XILINX
ALL PROGRAMMABLE™

VIEWPOINTS

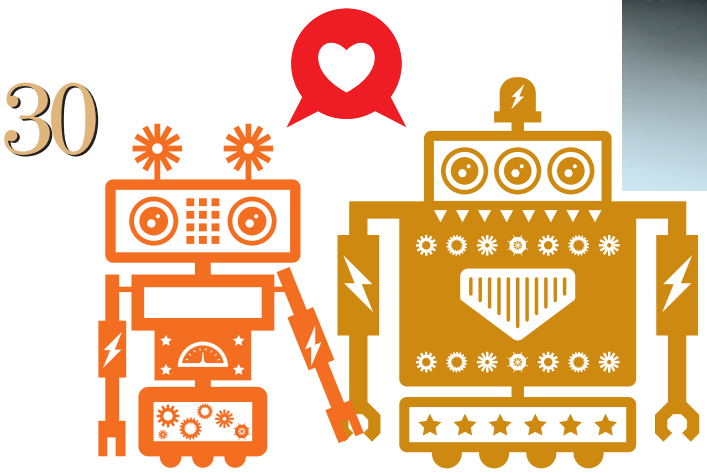
XCELLENCE BY DESIGN APPLICATION FEATURES

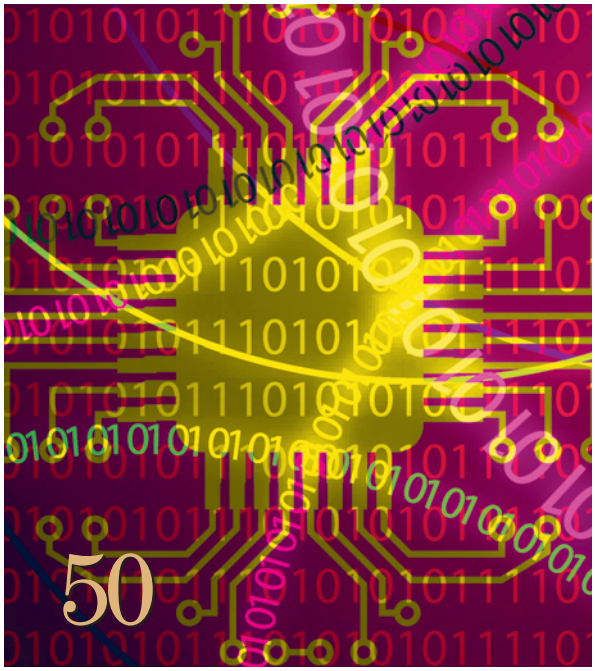


Cover Story



THE XILINX XPERIENCE FEATURES





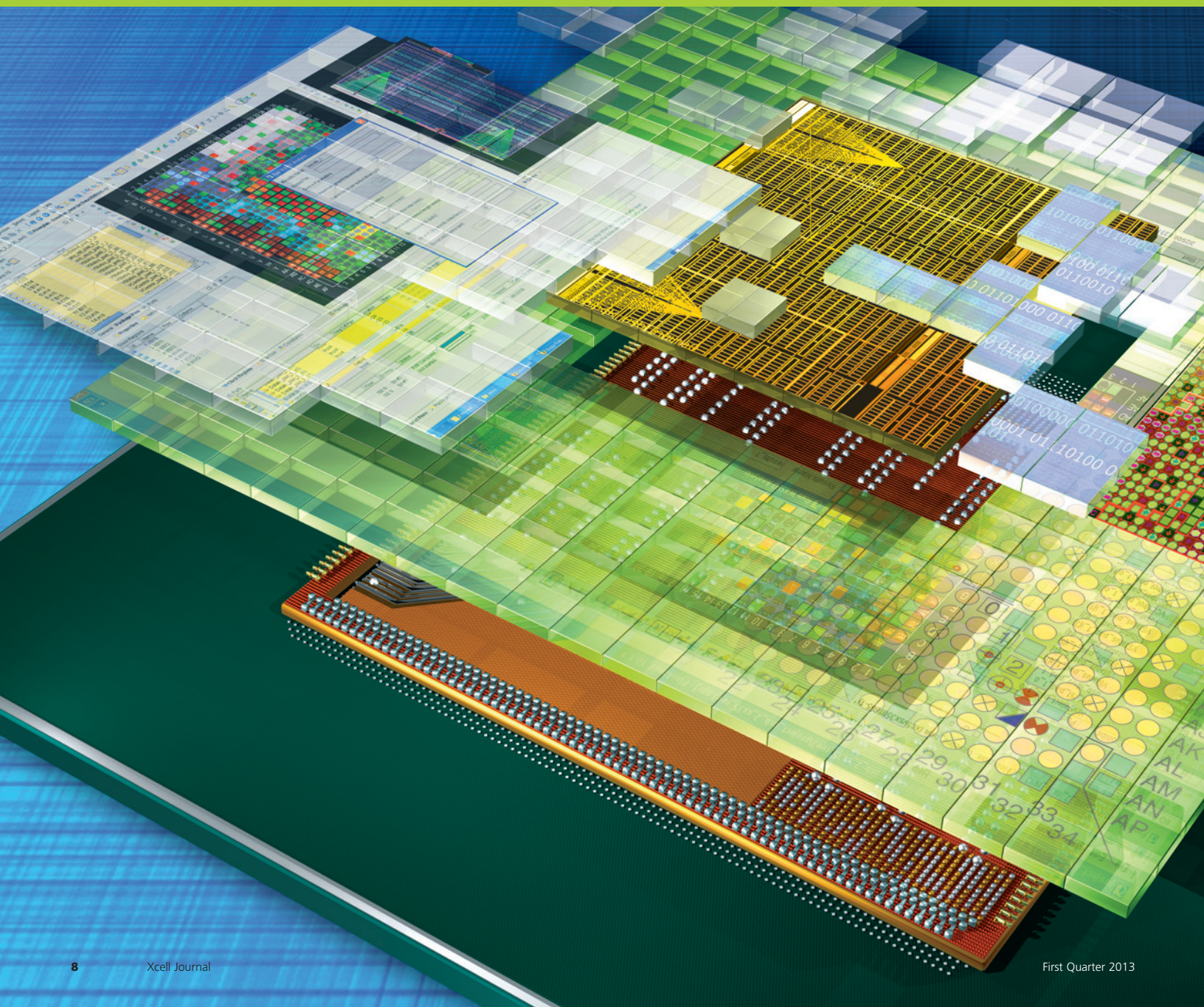
XTRA READING

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

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

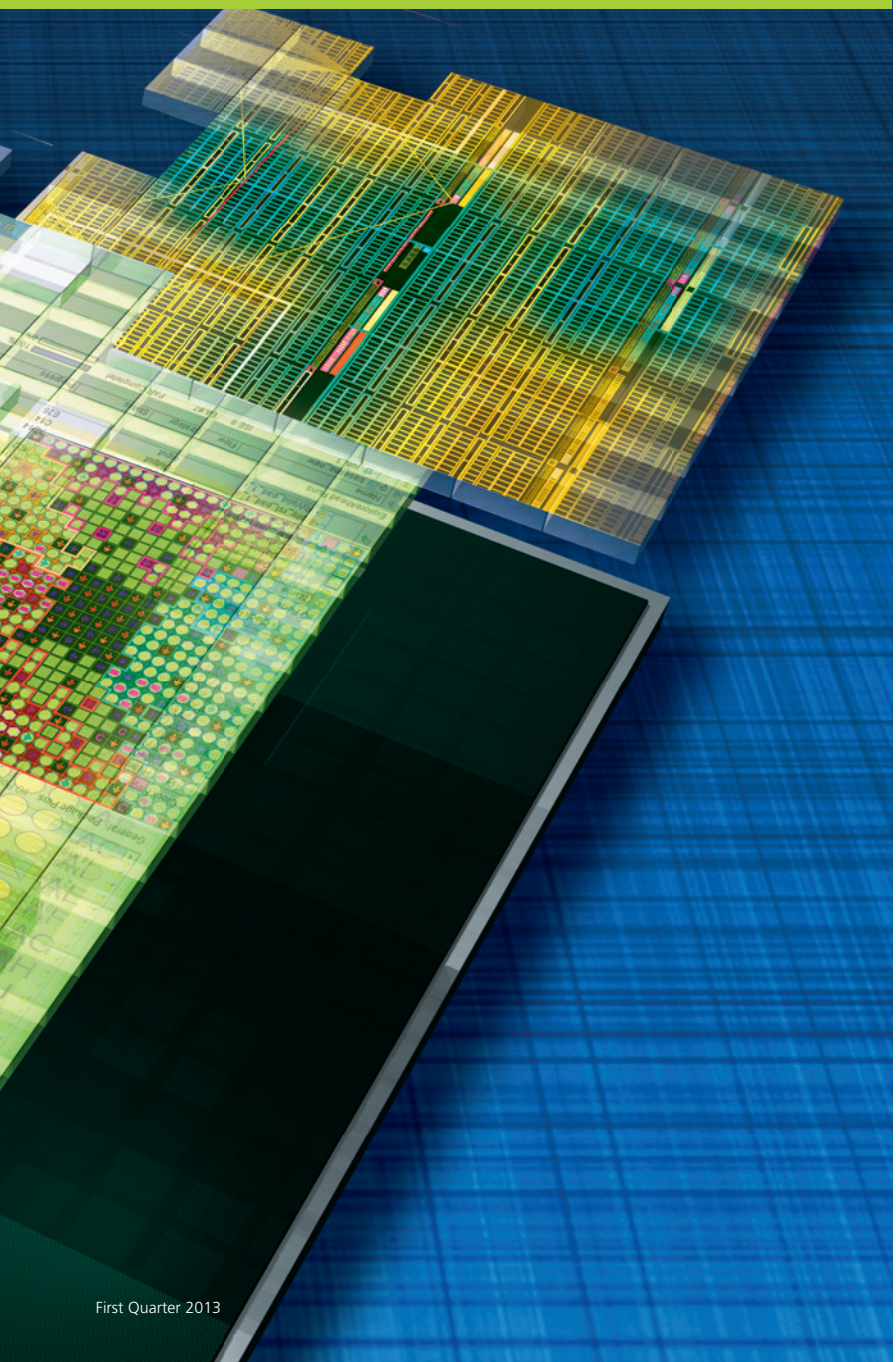
Getting Your Zynq SoC Design Up and Running Using PlanAhead

Xilinx's All Programmable SoC offers many advantages to the designer, and while development may seem complicated at first, in fact it is not.



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

This is the first in a planned series of hands-on Zynq-700 All Progammable SoC tutorials by Adam Taylor. A frequent contributor to Xcell Journal, *Adam wrote a second article in this issue on how to design an FPGA into your hardware (see page 42). He is also a blogger at All Progrmmable Planet (*http://www.programmableplanet.com/profile_content.asp?piddl_userid=450978*). – Ed.*



The Zynq™-7000 All Programmable SoC is the first of a new class of Xilinx® devices that marry a dual-core ARM® Cortex™-A9 processor with programmable logic on a single chip (see cover story, *Xcell Journal* issue 75). As such, the device offers a great leap forward in not only system flexibility, but also performance and integration. This system-on-chip platform does, however, require the FPGA engineer to consider a slightly different development path than is customary for logic-based FPGAs.

The good news is that development is not as difficult as you might think, thanks in large part to the availability of the Xilinx PlanAhead™ tool. Let's take a closer look at the steps involved in generating a Zynq-7000 system that you can load via JTAG.

LOTS OF FPGA RESOURCES

The Zynq-7000 family combines the dual-core ARM Cortex-A9 processor—the chip's processing system (PS)—with programmable logic (PL) resources equivalent to devices from the Xilinx Artix™ or Kintex™ families. Depending upon the device and speed grade you select, it is possible for the processing system to operate at frequencies up to 1 GHz.

The processing system side of the Zynq SoC provides supporting hardware for a number of memory and communications peripherals. This architecture allows the PS to be the master and to operate autonomously, without the need to even power up or configure the programmable

logic system. The key features the processing system supports are:

- Dynamic RAM interface—double-data-rate SDRAM, supporting both DDR2 and DDR 3 standards, including LPDDR2.
- Static RAM interfaces—support for SRAM devices along with NAND, NOR and QSPI flash memories.
- Communications interfaces—UART, CAN, I2C, SPI, USB, Gigabit Ethernet and SD/SDIO.
- General-purpose I/O—support for up to four 32-bit general-purpose I/Os.

With the exception of the DRAM link, the remaining interfaces utilize the Multiuse I/O, a bank of 54 I/O pins that you can assign to different functions as required. This does, of course, mean you cannot implement all peripherals at once. However, you can extend this interface into the programmable logic if you wish.

Communication between the processing system and programmable logic sides is comprehensive, and includes:

- Advanced Microcontroller Bus Architecture (AMBA®) and Advanced eXtensible Interface (AXI), with both master and slave interfaces, including direct links to external DDR and on-chip memory. In addition, the Accelerator Coherency Port (ACP) provides coherent access to the CPU memory space from the programmable logic side.
- Extended Multiuse I/O, which allows you to increase the number of processing system peripherals by using programmable logic I/O.
- Direct memory access and interrupts, including 16 interrupts from the PL to the PS, along with four DMA channels.
- Four clocks and resets from the processing system to the programmable logic.

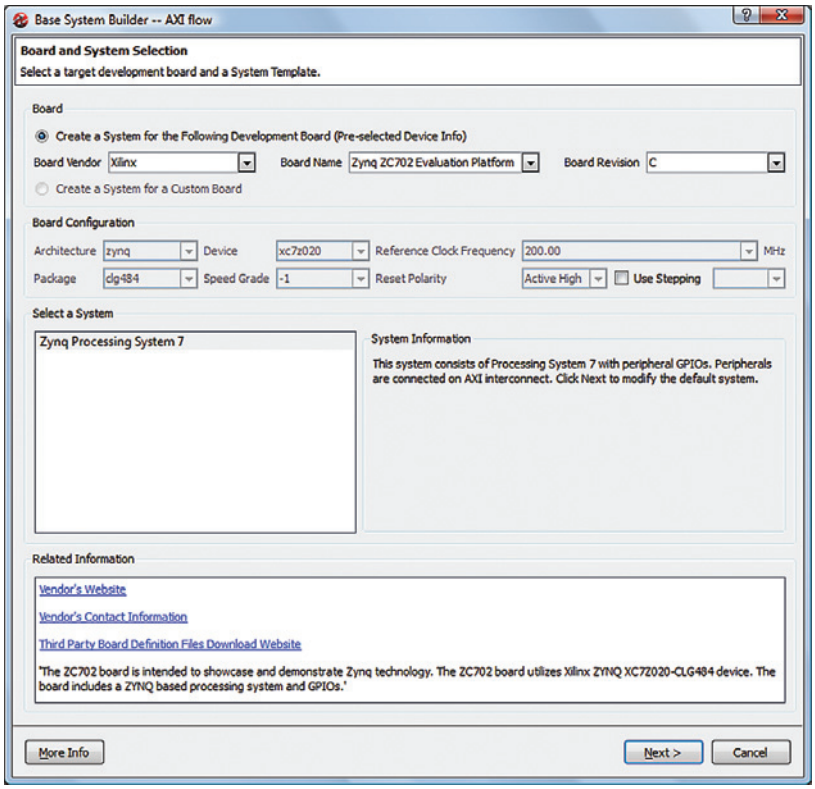


Figure 1 – The Base System Builder for the AXI bus

Zynq configuration differs from the process of configuring many of the previous families of Xilinx FPGAs, even those that contained hard macro processors like the Virtex®-2 Pro, Virtex-4 and Virtex-5 families. Within the Zynq system, the processing side is the master and therefore boots following a normal software boot process, loading in the application from nonvolatile memory and then either executing in place or cross-loading the application into faster DDR memory using a more complex boot loader. The programmable logic side of the Zynq loads via the Processor Configuration Access Port (PCAP), which allows both partial and full configuration. As always, you can also configure the device through the JTAG interface. Because of this added complexity, the Zynq device has more mode pins than you normally find on an FPGA to support all of these various configuration methods.

WHAT TOOLS DO WE NEED?

Creating a system-on-chip (SoC) requires a little more effort than developing a logic-based FPGA design. However, it is still pretty straightforward and the tool chain provides good guidance. To create a Zynq SoC design you will need four tools at a minimum: Xilinx Platform Studio, the ISE® Design Suite, Xilinx's Software Development Kit (SDK) and iMPACT.

Xilinx Platform Studio is the place where you create your processing system, be it PowerPC®, MicroBlaze™ or, in this case, the Zynq PS. Here, in XPS, you define the configuration, interfaces, timing and address ranges—everything needed to generate a processor system. The output from this process is an HDL netlist defining your system.

Most FPGA engineers are familiar with ISE. This Xilinx tool chain takes your HDL design, including the XPS netlist, and generates the required bit file. The Software Development Kit,

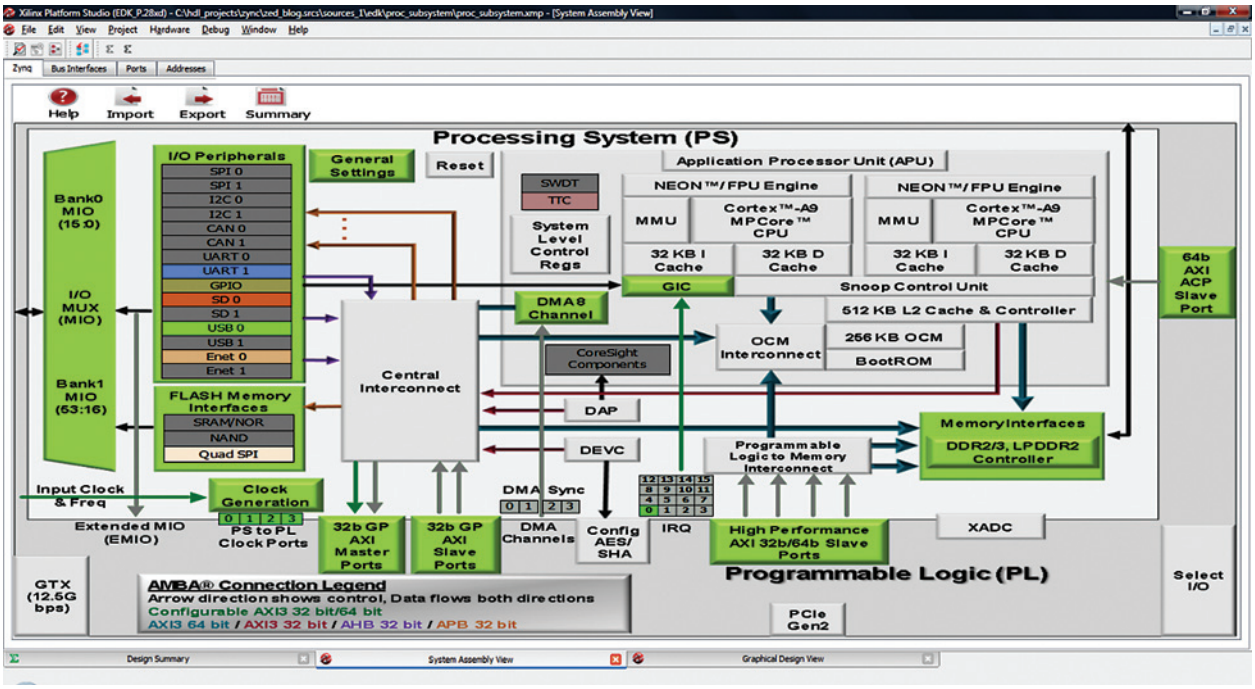


Figure 2 – The XPS view of the processing system

meanwhile, is where you will develop the software for the processing system. To correctly generate the software, the SDK needs to be aware of the hardware configuration of the system. Finally, the iMPACT tool allows you to program the bit file into the system and generate the bitstream format.

While you can use these four tools in isolation to create an All Programmable SoC, rather helpfully Xilinx's PlanAhead tool is capable of integrating them all, delivering a much simpler development process. Therefore, let's focus on this PlanAhead approach as we dig deeper into the Zynq flow.

CREATING THE HARDWARE

The first step in the development process is to open PlanAhead and create a new RTL project. This is a fairly straightforward matter. Since you will have no existing RTL, IP or constraints initially, keep selecting "next option" until you reach the device selection tab. If you are using a development board, then select the board you intend to target. Otherwise, select the device you wish to target for your system.

Once you have created the project, you will see the default screen within PlanAhead. At this point you will need to add a source file to the design. You can do this by selecting "add sources" under project management options on the flow options tab, which should be on the left-hand side of the screen. Initially we are interested in getting the processing system side of the Zynq up and running; therefore, select the "add or create embedded sources" option. This will open yet another tab from which you can select the "create subdesign" button, and then enter the selected name for the processing system.

The tool will now take you through a number of stages to create a base system using what is called the Base System Builder (BSB) Wizard, as shown in Figure 1. This step will open Xilinx Platform Studio, which will present you with a graphical representation of the processing system within the device (Figure 2).

If you have targeted a development board, the tool will automatically configure the peripheral interfaces for your board. However, if you are targeting a single device you will need to implement your own peripherals within this system—for example, DDR timings and the MIO connections. Doing so is very straightforward, and you can make sure you have made no errors by running the design rule checker before you leave XPS and return to PlanAhead.

Once back in PlanAhead, you will need to generate an RTL netlist for the processing system in either VHDL or Verilog, picking your choice of language via the project settings. Generating this HDL file is as simple as right-clicking on the name of your processing system and selecting the "generate HDL" option.

You can then create RTL designs for the programmable logic side of the device within PlanAhead as well, using the "add sources" option. Once you have created the RTL you require, you will need to define the top-level module by right-clicking on it and selecting the "set top" option. If your design has both a processing system and a programmable logic system, you will need to create a top-level RTL file to connect them together as you desire.

The board support package will be specific to your hardware implementation. It will contain drivers for peripherals along with a number of hardware-specific C header files.

The final remaining step before the SoC is complete and a bit file can be generated is to create a constraints file with pinout information for the PL side of the Zynq-7000 device. Do this within PlanAhead by right-clicking on the constraints option within the source window and selecting "add sources." This action will allow you to create the file. Now it is time to implement the design using the "run implementation" button.

CREATING THE SOFTWARE

Having defined your system using PlanAhead and Xilinx Platform Studio, you now need to create the software that is to run on the processing system. For the majority of this development you will be using the Software Development Kit. However, there's one more thing you need to do within PlanAhead—namely, export the hardware configuration to the SDK by means of the export options within PlanAhead.

Performing this export should result in the opening of the SDK and the importation of the hardware specification, as seen in Figure 3.

Once you have imported the hardware, you will need to create a board support package for your hardware, as shown in Figure 4. This BSP will be specific to your hardware implementation. It will contain drivers for peripherals along with a number of hardware-specific C header files such as xparameters.h, which defines the memory map of the system along with other system configuration parameters. When creating the BSP, be sure to select your target processing system and CPU0. (We are examining a simple system and will leave multicore operations to another day.)

The next step is to create the actual software project itself. At this point, you will get to choose a language, either C or C++, and a number of templates ranging from a simple "hello world" to a first-stage boot loader. You can also choose whether to implement the project on a bare-metal system—that is, one with no operating system—or within the Linux OS. Once you have completed this step and created your project, you are free to begin writing code for your application.

If you are new to developing embedded software, the task may at first seem a little daunting. However, the SDK provides plenty of support and assistance to both first-time and experienced developers. Should you be unsure of the peripheral drivers within your system or how to drive them, the system.mss file under your BSP within

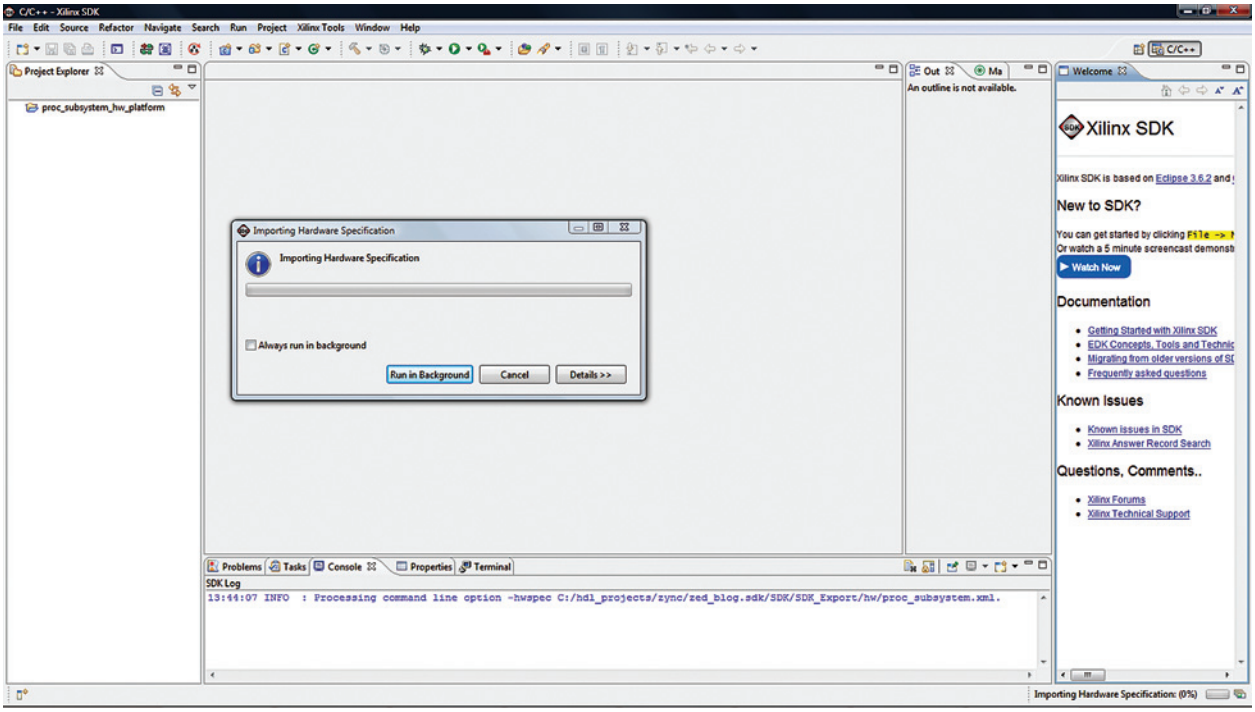


Figure 3 – Importing the hardware to the SDK

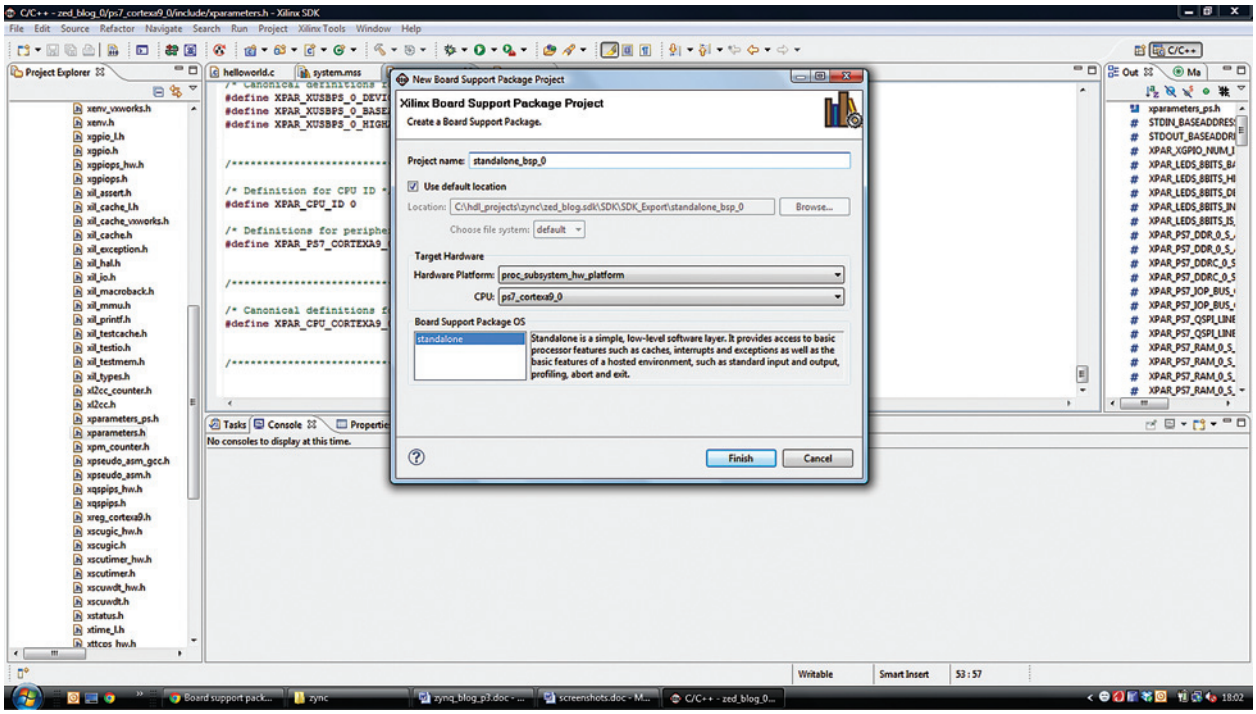


Figure 4 – Creating the board support package

the tool's Project Explorer will show the memory map. This view also offers a list of helpful links to documentation and examples.

There are a large number of C header files for you to call into a project. However, certain commonly used files contain parameters and functions to drive peripherals.

- **Stdio.h** defines the standard input and output. This file should be familiar to many people who have used C before.
- **Platform.h** defines basic functions for the implementation and platform initialization, and for cleanup.
- **Xil_types.h** defines a number of types needed for the Xilinx IP.
- **Xgpio.h** defines the drivers for the AXI-connected general-purpose I/O module.
- **Xparameters.h** defines the hardware architecture and configuration for this implementation.

Having written your code and successfully compiled it, you will then of course wish to download the application to your board. To do this, you first need to create a linker file script that will determine where the executable is placed within the system memory. While your first program may be small and would fit within the on-chip memory, it is often best to place it within the DDR memory (if used) to demonstrate that this interface was also correctly configured. You can create the linker script by right-clicking on your project and selecting "generate linker script."

The final thing to do is to set up the run configurations (project explorer -> project -> run -> run, as within the right-click menu). Here you can define your STDIO connection—the serial port you are using for the STDIO (if any). Once you are happy, you can apply the configuration and then close it (do not click "run," as we are not quite ready for that). The final stage is to attach the board to your PC, connecting the JTAG cable and all other cables you may need, including an RS232 serial cable, and programming the hardware via the Xilinx tools (use the "program FPGA" option). Check that the bit file is the correct one and then program the device.

Having completed the configuration, you can now download the ELF file and try out your program. This is as simple as selecting your project within PlanAhead's Project Explorer, right-clicking "run as" and then "launch on hardware." If you have not previously built or compiled your code, running this command will automatically build your design, provided it is error-free.

The Zynq-7000 device introduces a new type of all programmable system-on-chip development flow that may seem complicated to the new user. However, the tool chain is integrated within, and accessible from, Xilinx's PlanAhead tool, creating a seamless, stage-by-stage development process that ensures you will be able to create a Zynq system with relative ease.

Designing to Forestall Failure

Planning a safety-critical system demands close attention to hardware failure rates and redundancy.

by Austin Lesea
Principal Engineer
Xilinx Labs
austin.leasea@xilinx.com

When it comes to safety-critical systems (planes, trains, automobiles), or to safe industrial controls (windmills, factory steam plants), the designer must take a number of factors into account. First, failures are always an "option"—they can and will happen. [1] Second, when a failure occurs, it is your job to make sure it does not result in damage or loss of life.

As opposed to a commercial system, where a failure results in inconvenience or frustration, the consequences of a failure in a safety-critical application are dire. Therefore, the designer of a fail-safe system must differentiate and apply the proper design philosophy to eliminate or at least minimize failures.

In a server, router, telephone system, cell phone switch or other commercial system, it's extremely unlikely that failure will result in damage, destruction or loss of life. Depending on the service grade (number of customers affected) and the use of the equipment (business telephone switch or 911 service center), a more stringent set of rules may apply. But if the system is deemed noncritical and it fails, the result will be a loss of service, which is something that is not very pleasant, but is acceptable. Of course, a reputation for a low failure rate will always help in keeping customers happy (and paying for your products).

Let's take a closer look at systems in which failure is acceptable and simply means a loss of service, before discussing systems where failure is unacceptable.

SYSTEMS THAT ARE 'OK' TO FAIL

Typically, designers wish to make their designs as reliable as possible. The hard failure rates of Xilinx® FPGA devices by technology are listed in the Quarterly Reliability Report [2], in Tables 1-16. Take the hard failure rate for the 7 series devices, which is listed as 24 FIT (failures per billion hours), as of Aug. 22, 2012. That equates to one failure per 4,756 years. Imagine having 10,000 systems in the field, and you would expect to see one customer field failure return every 174 days on average. This is the baseline, "do nothing special," failure rate for the system if the system consisted of just one component: a Xilinx 28-nanometer 7 series FPGA device. Now, no system consists of just one component. There are the printed-circuit boards, power supplies, connectors, LEDs, switches and so forth.

A typical system, with all of its associated components, is going to have a lifetime much shorter than that of the Xilinx FPGA device alone. Typical numbers range from 1,000 to 10,000 FIT for systems in which the 7 series FPGA device is a small part. This equates to failures ranging from every 114 years down to 11.4 years, or from four days down to less than a day between customer returns. Again, in these non-safety-critical systems, a failure represents an inconvenience, but no one is harmed and no damage results.

Not all components have a constant failure rate. Most devices have what is known as a "bathtub" curve—one with steep sides and a flat bottom—for their mean time between failures. The initial period of life harbors more failures than the latter portion, a phenomenon known as "infant mortality." There is usually a long stretch after this initial period of a lower failure rate, known as the "normal operating life." Near the end of the component's lifetime, the failure rate increases, and this period is known as the "end of life."

A laptop computer, for example, has a typical service life of three years. Failures that occur immediately after purchase or after three years of ownership dominate the failures that the user will experience, with far fewer failures in between.

Xilinx designs its commercial and industrial products for a 15-year minimum life. Depending on junction temperature, this time period may be more or less. The reliability and user guides spell out the details for the various Xilinx FPGAs.

SYSTEMS THAT ARE NOT 'OK' TO FAIL

In a system where life or the environment is at stake, there is always an allowed failure rate. It may be a very stringent requirement, but since no system is perfect and no design exists that is completely fail-safe, there is a number, however tough it may be to achieve.

One example is for a power-control system in a windmill. The windmill, if it fails, may throw off a windmill blade, totally destroying the device and potentially injuring or killing a person nearby. Let us suppose for argument's sake that that number is 10 FIT, or one failure every 11,416 years. The first thing to recognize is that no single device is reliable enough to meet this requirement; the system will have to have some form of redundancy.

As the hardware failure rate of the components is too large to meet the requirement, the designer will have to provide duplication for all of the critical components, subsystems and power supplies, along with other means to verify that there are no failures. Such systems are often designed such that when they fail, they fail safely.

Imagine two completely separate assemblies, each with a failure rate of 10,000 FIT, each checking the other through a duplicated communications channel (in this case, perhaps just two UARTs on four wires in each system). The single point of failure is now the four wires or the UARTs, but any failure would be detectable; the subassembly would be unable to talk to its twin, and the system would be able to immediately shut down everything safely.

There is still the possibility that both systems might fail at the same time. That is the probability of failure of each, times itself, or one simultaneous failure every 130 million years. That performance is probably going to meet the requirements.

As another example, the space shuttle used five redundant systems, and as long as they all agreed with one another, they could launch. After launch, as long as three of the five worked (agreed), the systems were deemed safe. Similarly, commercial airplanes use three-way redundancy for their flight controls.

WHEN AM I DONE?

Often designers forget that designing for reliability is an endless task unless they have a clearly stated goal. In the above examples, if the soft failure rate (from atmospheric neutrons) was 1 FIT for an element in a Xilinx 7 series device (that's the actual rate for a PicoBlaze™ soft processor), what should a designer do? Well, the general rule that I have seen used for a very long time now is that once any failure rate is less than the hardware failure rate, you are done.

If the final system does not meet the requirement, then you have the wrong architecture (no redundancy, for example).

THERE ARE STANDARDS FOR THAT, NOW...

Fortunately, designers now have international standards for industrial, aeronautic, automotive, medical and other systems. [3] IEC 61508 is the umbrella standard that applies to all at the highest level, with each industry having its own subset standard. The one thing to remember about these standards is that none of the manufacturers of the components are certified (or certifiable), but the final system and its manufacturers must be certified.

Anyone who claims to be able to sell you a 26262 (IEC automotive standard) certified FPGA device is lying; there is no such thing. The same is true for DO-254, a standard for airplanes. It is the entire system and its design that must meet the standard, not the components. Of course, if there is one FPGA device in the system and nothing else, it is pretty easy to find the failure rate. Just look it up in UG116, if it is a Xilinx component. But that is never the case; all systems consist of more than just one device.

References

1. *"Failure is always an option," Adam Savage, from Myth Busters:* http://www.brainyquote.com/quotes/quotes/a/adamsavage207580.html

2. http://www.xilinx.com/support/documentation/user_guides/ug116.pdf

3. *For more information and some fun reading, visit* www.xilinx.com/applications/aerospace-and-defense/avionics/. *Here you will find white papers on DO-254 and SEU. Another helpful link is* http://en.wikipedia.org/wiki/ IEC_61508.

Austin Lesea graduated from UC Berkeley with his BS EECS in electromagnetic theory (1974) and MS EECS in communications and information theory (1975). He worked in the telecommunications field for 20 years designing optical, microwave and copper-based transmission systems. For 10 years Austin was in the IC Design department for the Virtex® product line at Xilinx. For the last three years he has worked for Xilinx Research Labs, where he is looking beyond the present technology issues. Austin holds 70 patents.

No Room for Error

Synopsys FPGA Design Tools are an Essential Element for Today's High Reliability Designs

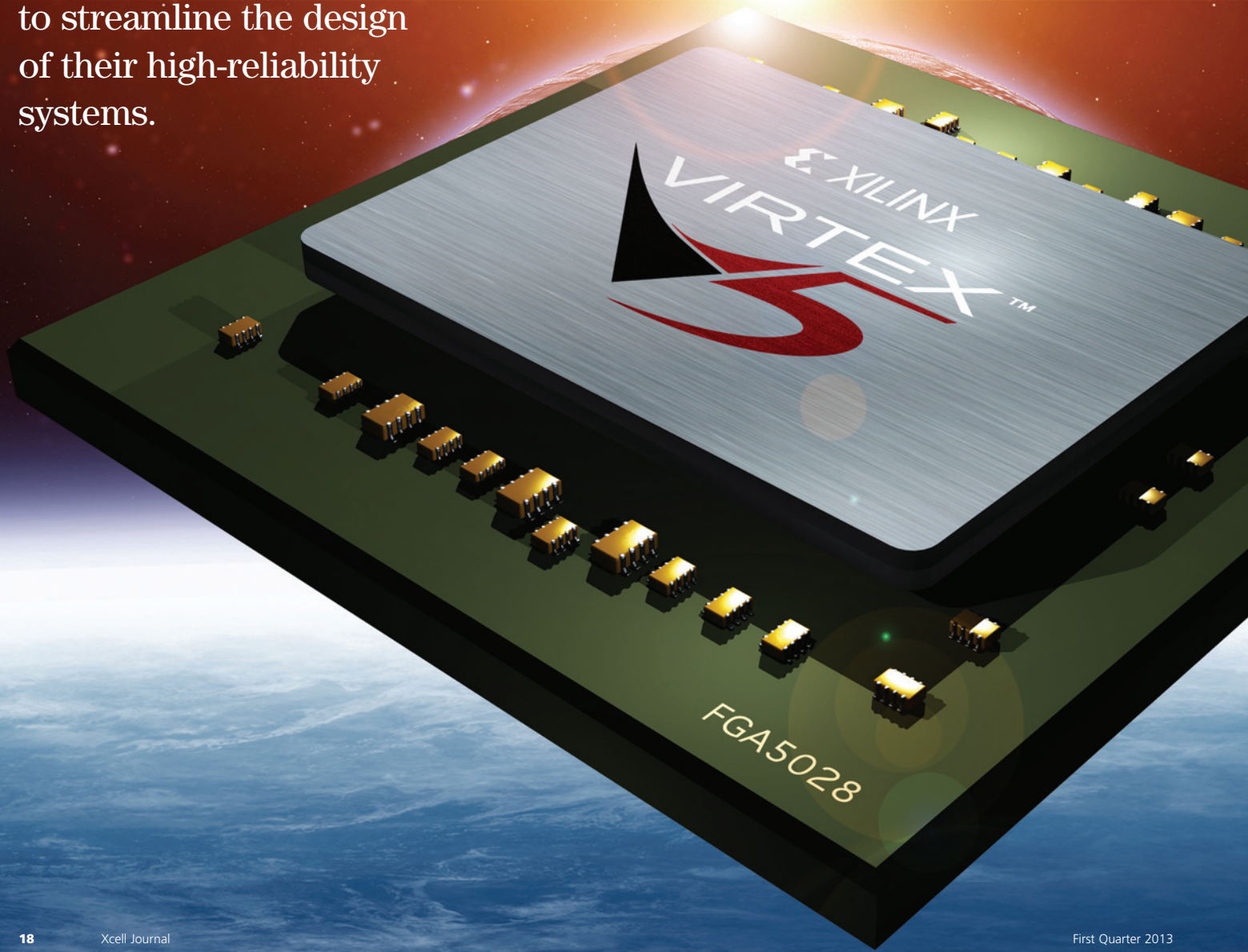
There is no room for error in today's communications infrastructure systems, medical and industrial applications, automotive, military/aerospace designs. These applications require highly reliable operation and high availability in the face of radiation-induced errors that could result in a satellite failing to broadcast, a telecom router shutting down or an automobile failing to respond to a command.

To learn more about Synopsys FPGA Design tools visit www.synopsys.com/fpga

SYNOPSYS®
Accelerating Innovation

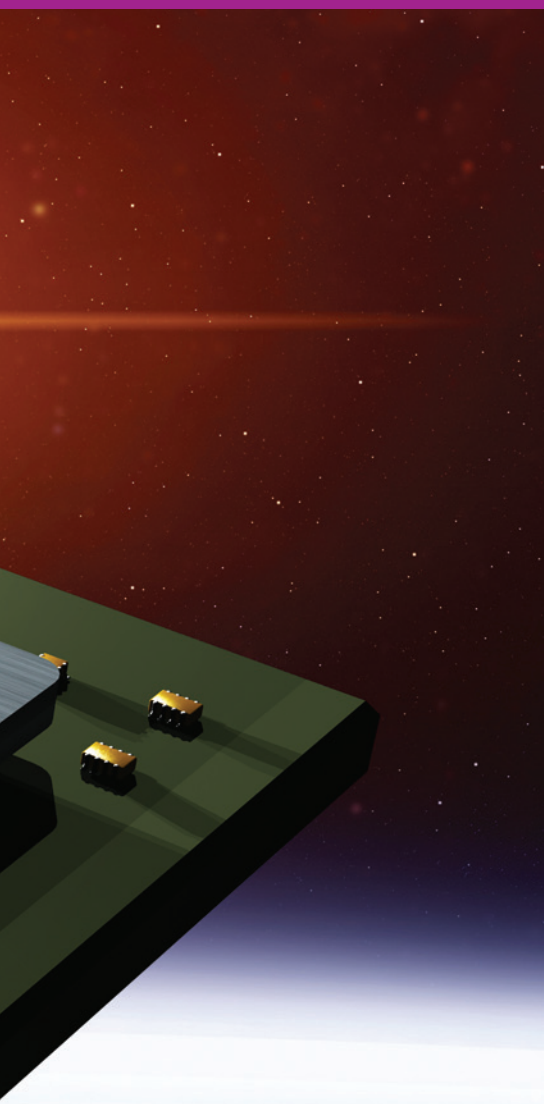
Implementing Analog Functions in Rugged, Rad-Hard FPGAs

Aerospace engineers are pulling converter and clocking functions onboard Xilinx FPGAs to streamline the design of their high-reliability systems.



by Allan Chin
CEO
Stellamar
allan.chin@stellamar.com

Luciano Zoso
CTO
Stellamar
luciano.zoso@stellamar.com



FPGAs have already changed the cost/reliability paradigm for embedded systems in high-reliability applications, thanks to advances in hardness and power reduction. But on many embedded applications for high-reliability markets, designers depend on a number of peripheral analog components such as analog-to-digital and digital-to-analog converters to talk to the real world. Other system components such as phase-locked loops (PLLs) and DC/DC converters are usually required to complete a system design. These peripherals impact overall cost, size and reliability. Peripheral analog parts can also be challenging to work with and to source for radiation environments, as an example.

To further leverage the power of FPGAs, military-and-aerospace engineers are actively looking for ways to integrate many of these analog functions onto the FPGA. Synthesizable, digital IP cores that replace some analog functions now exist, allowing mil/aero designers to implement ADC, DAC, DC/DC controller and clock-multiplier functions in fully digital processes such as FPGAs. Not only does this new ability leverage the advantages of FPGAs, it also helps mitigate many challenges of using analog components in high-reliability applications.

OVERCOMING HIGH-RELIABILITY DESIGN CHALLENGES

The engineering challenges of designing for military or high-reliability applications such as aerospace are numerous. Power and weight are usually under strict budgets because they can affect operating costs and insertion costs exponentially. Physical shock safeguards, force survival and protection from single-event upsets (SEUs) and latchup often mean that parts are larger, heavier and more power hungry than commercial devices. For instance, a commercial 12-bit, 10-MHz bandwidth ADC measures approximately .71 by .42 inches and consumes 280 milliwatts. The equivalent radiation-hardened part is .81 by .72 inches and consumes 335 mW. That's almost double the size at 20 percent more power.

A wide temperature range is another issue. Typically, temperatures of -40°C to +80°C are expected for many military embedded applications here on Earth. Temperature takes on another complexion in space. In satellite electronics design, for instance, the normal operating junction temperature might be -55°C to +125°C. Monitoring this onboard temperature is key to effective system maintenance, but installing a rad-hard ADC part to provide this function can add up to one square inch of board and require additional components and testing.

When a high-reliability design makes use of peripherals such as ADCs, DACs, DC/DC converters or PLLs, each one of those components represents a possible point of failure. Each must be qualified and tested, and each is most likely not optimally designed for the specific need. There is also always a risk that the manufacturer will discontinue the part, forcing requalification of the entire system.

These challenges to working with analog components in high-reliability environments can evaporate by using the FPGA for a unified, all-digital approach. Let's take a look at this new paradigm in military/aerospace design.

PULLING ANALOG FUNCTIONS ONTO THE FPGA

Regardless of how you define "analog" and "digital," significant differences and integration issues exist between the two. Because of these issues, it can be very advantageous to have digital designers pull analog functions onto an FPGA and test them. Herein, we will define "digital" as using standard digital library cells and passive components for a fully

A Digital ADC IP core on a radiation-hardened FPGA can get down to 0.5-Hz bandwidth per channel and can consume less than 6 mW, compared with 335 mW for an external part.

synthesizable and digitally testable design. Designers can create digital IP blocks of ADCs, DACs, DC/DC converter controllers and clock multipliers in RTL format and implement them in all-digital processes.

With these IP blocks, military designers can take advantage of rugged and radiation-hardened FPGAs to implement customized analog functions. Not only does this approach leverage the inherent protection properties of the FPGA, but these blocks are also a great way to utilize unused FPGA resources. Xilinx recognizes this advantage and now partners with Stellamar to provide these functions. Increasingly, aerospace companies are turning to these solutions to attack analog-integration problems.

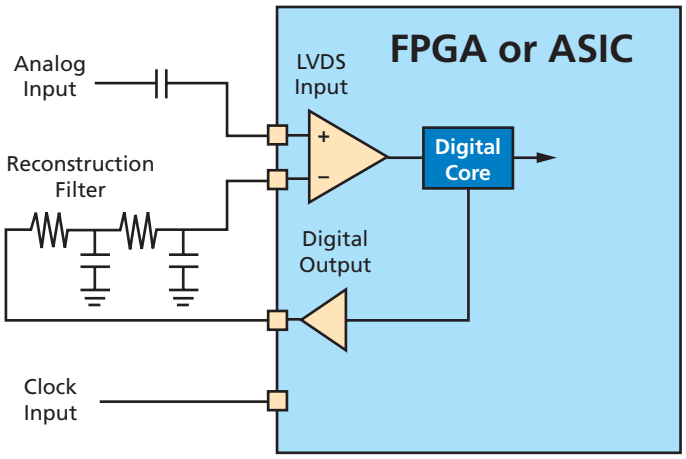


Figure 1 – An example of a fully digital ADC IP core interface

DIGITAL ADC CORE YIELDS BENEFITS

Figure 1 depicts an example block diagram of a Stellamar Digital ADC IP core. With the digital approach, the core requires only a few external passive components. The IP core is instantiated right inside the FPGA and is easy to implement through digital synthesis. On a Xilinx® Virtex®-5QV device, a scenario such as that pictured in Figure 1 utilizes less than 1 percent of FPGA resources.

Proprietary signal processing makes it possible to replicate analog sigma-delta ADC performance with all-digital library cells. Companies like SEAKR Engineering and the Finnish Meteorological Institute are using Digital ADC IP in their On Board Processor Program and Lunar Landing Missions, respectively. Some benefits are:

- 50 percent lower power than analog ADC parts
- 68 percent smaller area than analog ADC parts
- Process technology independence
- Reduced risk and cycle time
- Digital integration, synthesis and testing
- Easier radiation-hardened design

PERFORMANCE PLUS APPLICATIONS

Current performance is up to 15 bits of resolution and several hundred kilohertz of bandwidth. Bandwidth depends on the selected resolution. This level of performance is suitable for a host of applications including sensors (temperature, pressure, voltage, current and acceleration), touchscreen integration, high-quality voice and motor control.

As an example, many design teams use a radiation-hardened, 12-bit, 10-MHz bandwidth ADC part for monitoring onboard temperature and voltage. Some FPGAs, such as the Xilinx Virtex-5QV space-grade FPGA, even have embedded diodes highlighting the importance of the temperature-sensing function. However, normal bandwidths for these types of measurements are 0.5 Hz to 10 Hz, so using bandwidth in the megahertz is like driving the head of a pin with a sledgehammer. A Digital ADC IP core on a radiation-hardened FPGA can get down to 0.5-Hz bandwidth per channel and can consume less than 6 mW, compared with 335 mW for the external part. Why waste critical board space and power for such a low-level task?

CONTROLLING DC/DC POWER MANAGEMENT

Power management is becoming a larger part of overall system design. Sometimes a single design can include more than 30 power supplies. External radiation-hardened DC/DC converters retain the same difficulties as external ADCs. Thus, the use of these parts to control power complexity in high-reliability applications does not scale well.

All-digital DC/DC controller IP now exists to take advantage of radiation-hardened FPGAs' processes and to allow for simplification of control, redundant power supplies, infinite sequencing and infinite throttling (see Figure 2). You will still need an external power transistor, but this can be much easier to work with than a full DC/DC converter part.

DIGITAL CLOCKING SOLUTIONS

Phase-locked loops are some of the most widely used analog blocks for clock generation; thus, most FPGAs have incorporated PLL capability within the package. However, some FPGAs, including certain radiation-hardened FPGA families, do not include PLLs at all. Other radiation-hardened FPGAs generally do not include the PLLs in the rad-hard portion of the package.

Digital clock multiplier IP used on these FPGAs can provide the ability to generate any clock up to about 2 GHz with no lock time. Models show 50-picosecond peak and 35-ps RMS, with 5-ns to 1-ns rise/fall. As with Digital ADC IP, this solution requires very few off-the-shelf passive components.

PUTTING IT TOGETHER

Historically, FPGAs did not lend advantages to analog functions, forcing high-reliability design teams to use nonoptimal external analog parts. This is no longer the case, as mil/aero engineers now have robust options for integrating analog functions into any digital fabric, including radiation-hardened FPGAs. By using digital implementations of analog functions from Stellamar, engineers can add critical functionality such as thermal monitoring, redundant power supplies and clocking functions–all without adding weight, power or size to the design. The digital synthesis and test methodology ensures the operability and greatly increases reliability.

Further, designers can easily leverage these technologies across projects and across the whole organization. With budgets being slashed and performance more important than ever, these digital IP cores give mil/aero engineering teams the flexibility and productivity they need to meet critical mission objectives.

For more information about Stellamar cores, call (480) 664-9594 or go to *www.stellamar.com*.

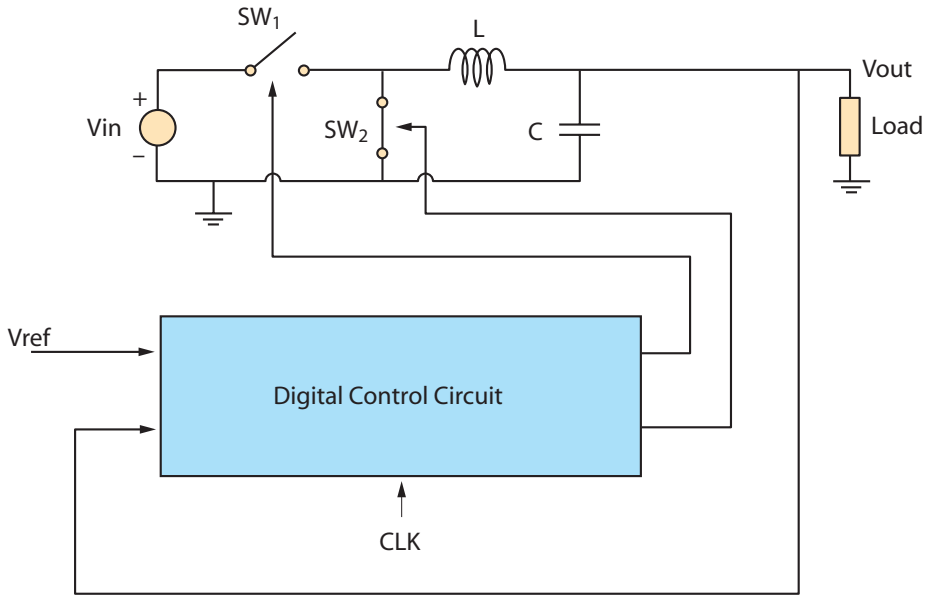


Figure 2 – Example DC/DC controller block diagram

Software-Programmable Digital Predistortion on the Zynq SoC

Xilinx's Zynq All Programmable SoC and Vivado HLS tool provide an efficient and flexible way to implement wireless digital front-end applications.

by Baris Ozgul
Research Scientist
Xilinx, Dublin, Ireland
baris.ozgul@xilinx.com

Jan Langer
Research Scientist
Xilinx, Dublin, Ireland
jan.langer@xilinx.com

Juanjo Noguera
Research Scientist
Xilinx, Dublin, Ireland
juanjo.noguera@xilinx.com

Kees Vissers
Distinguished Engineer
Xilinx, Inc.
kees.vissers@xilinx.com

Digital predistortion (DPD) is an advanced digital signal-processing technique that mitigates the effects of power amplifier (PA) nonlinearity in wireless transmitters. DPD plays a key role in providing efficient radio digital front-end (DFE) solutions for 3G/4G basestations and beyond. A generic DPD system consists of a predistorter that compensates for the nonlinearity effects prior to the input of the PA, and an estimator on the feedback path from the output of the PA, which updates the predistorter coefficients to reflect the possible changes in operational characteristics. Based on the modulation type, power amplifier technology and transmission bandwidth, the DPD solution can differ from system to system. Hence, it is worthwhile to provide a flexible design methodology for DFEs that facilitates the implementation and integration of new DPD coefficient estimation algorithms.

Modern FPGAs are a promising target platform for the implementation of flexible wireless DFE solutions, including DPD. The Xilinx® Zynq™-7000 All Programmable SoC, which integrates the programmable-logic fabric with a multicore ARM® processor system, provides an especially interesting new option. The Zynq-7000 All Programmable SoC enables the partitioning of functionality among hardware and software components to increase the overall system performance. As shown in Figure 1, this All Programmable SoC is capable of implementing all the required functions of a DFE in a single chip.

Our team created a software-programmable design flow to implement the DPD coefficient estimators on a Zynq-7000 device using a methodology that allows the flexible partitioning of the functionality among the hardware and software components, depending on the complexity of the estimation algorithm in use. We achieved a significant productivity increase thanks to the Vivado™ high-level synthesis (HLS) tool, which allowed the implementation and verification of our hardware design at the software level.

SOFTWARE-PROGRAMMABLE DESIGN FLOW ON ZYNQ

The Zynq-7000 SoC is a hybrid computing platform that consists of two major parts. First, there are two embedded ARM Cortex™-A9 processors operating at up to 1 GHz and their support infrastructure, including a cache hierarchy, memory controllers and I/O peripherals. This section in itself represents a complete programmable embedded platform that is fit for use without any FPGA programming. The two ARM processor cores come with a single-instruction, multiple-data (SIMD) extension called NEON that provides a 128-bit-wide data path.

Second, the Zynq-7000 devices also contain an area of programmable logic that represents a conventional FPGA. The major advantage of this device is the high bandwidth available between the FPGA and the embedded processors by means of a multitude of AXI4 communication ports. In this way, it is possible to develop software for the processor and, as necessary, offload compute-intensive tasks into the programmable logic.

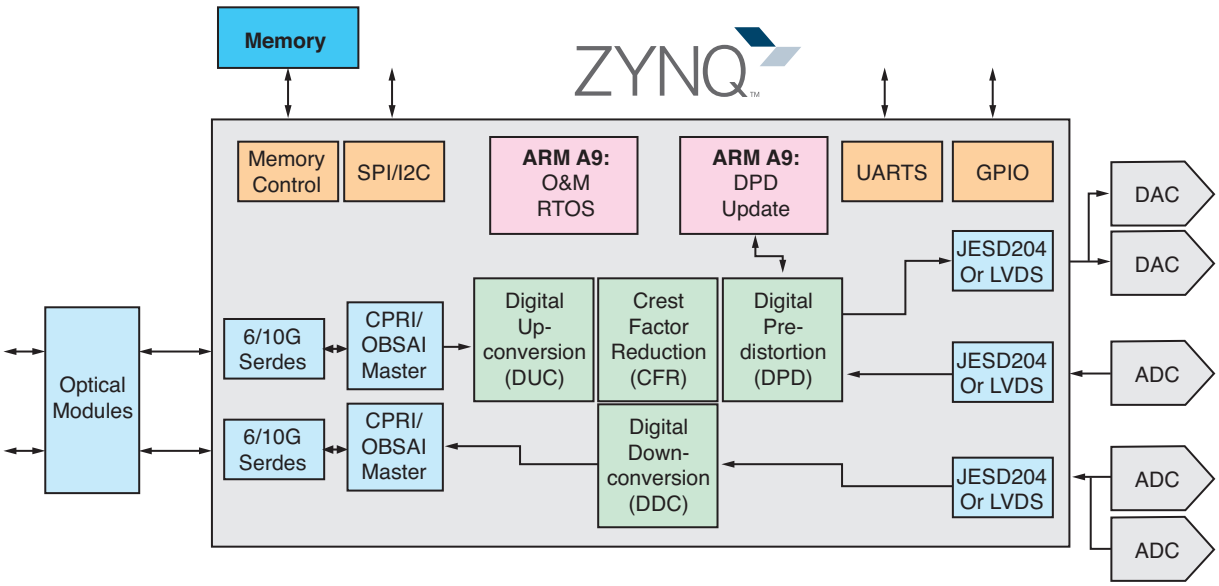


Figure 1 – Programmable wireless digital front end on the Zynq All Programmable SoC

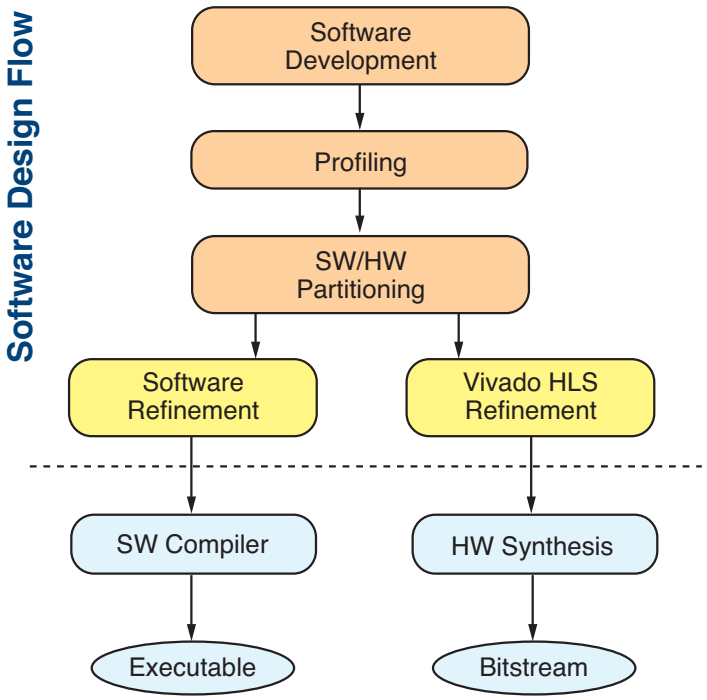


Figure 2 – Overview of the design flow, which begins in software development

Figure 2 illustrates a short overview of our proposed design flow. The first step is no different from a typical software design flow. It involves implementing the application in pure software, where the results can be tested and verified thoroughly. Next, a profiling step reveals the bottlenecks of the application, such as compute-intensive subfunctions that need hardware acceleration.

This so-called hardware/software partitioning involves not only the selection of functions to accelerate, but also the decision on an adequate communication infrastructure such as DMA transfers vs. memory mapping. Additionally, some changes to the software are necessary in order to call the hardware accelerator instead of the original C function.

In a traditional design flow, an experienced hardware designer implements the functions to be accelerated in a hardware description language like VHDL or Verilog. With the availability of the Xilinx Vivado HLS tool, we replaced the manual hardware implementation with an automatic step that uses the original software functions to generate corresponding hardware accelerators. The conversion requires several incremental manual refinement steps that include adding directives to the code or even restructuring the algorithm in order to obtain an efficient hardware implementation. During this process, the code is still executable as software, so that the designer can use the original test and verification environment. This is a big advantage over the traditional hardware design flow.

After the completion of the refinement step, the Vivado HLS tool generates a hardware accelerator implementation that meets the design constraints—for example, the required clock frequency and the amount of hardware resources. Next, the integration step requires the instantiation of communication components (for example, DMA) to enable the interaction between the hardware accelerator and the processor. Finally, system synthesis generates a bitstream to program the programmable logic.

HIGH-LEVEL SYNTHESIS FOR PROGRAMMABLE LOGIC

HLS tools raise the level of abstraction for designs in the programmable logic, and make the time-consuming and error-prone register-transfer-level (RTL) design tasks transparent. These tools take as their input a high-level description of the specific algorithm to implement and generate the RTL design for the target hardware accelerator.

Modern HLS tools accept as their input untimed C/C++ descriptions, from which they interpret the sequential semantics of the input/output behavior and the architecture specifications. Based on the C/C++ code, compiler directives and target throughput requirements, these tools generate high-performance pipelined architectures. Furthermore, they enable automatic pipeline stage insertion and resource sharing to reduce hardware resource utilization.

We have adopted the overall hardware design flow seen in Figure 3. The first step in this flow is restructuring a reference C/C++ code that the designer could have derived from a MATLAB® functional description. Here, restructuring means doing modifications in the original code (which is typically coded for clarity and ease of conceptual understanding rather than for optimized performance) to turn it into a format more suitable for the target processing engine. This is similar to rearranging an application's code to have more efficient performance on a DSP processor. When targeting FPGAs, this restructuring might involve, for example, rewriting the code so it represents an architecture specification that can achieve the desired throughput, or rewriting to make efficient use of specific FPGA features such as embedded DSP macros.

The functional verification of the implementation code uses traditional C/C++ compilers (gcc, for example) and reuses C/C++ level testbenches

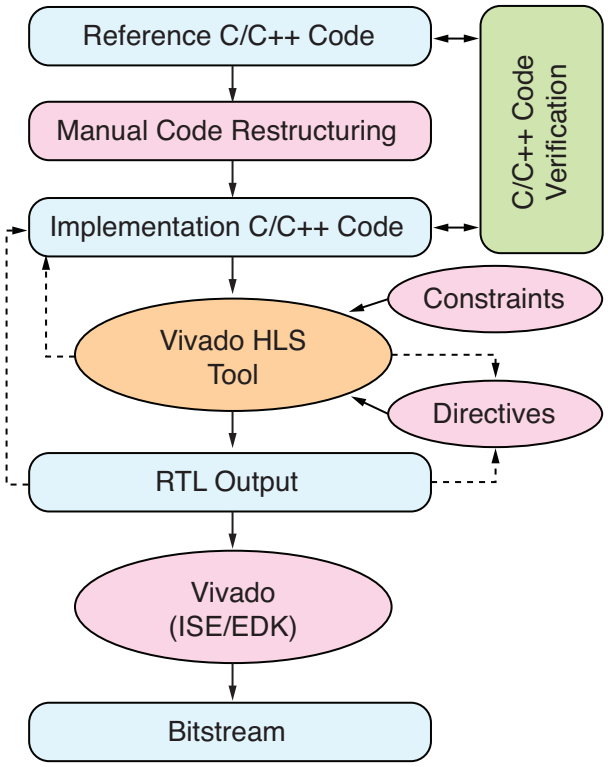


Figure 3 – High-level synthesis design flow for programmable logic

developed for the verification of the reference code. In addition to the implementation code, constraints and compiler directives (e.g., pragmas inserted in the code) are the other important input of the HLS tool. Two essential constraints are the target FPGA family (that is, the technology) and the target clock frequency. Naturally, both of these factors will have an effect on the number of pipeline stages in the generated architecture. The designer can apply different types of directives to different sections of the code. For example, there are directives for loop unrolling. As another example, there are directives to limit the instances of specific functions or operations in order to minimize the corresponding FPGA resource utilization.

The HLS tool takes all these inputs (the implementation C/C++ code, constraints and directives) to generate an RTL output and to report the throughput of the generated architecture. If the generated architecture does not meet the required throughput, you can modify the implementation C/C++ code or the directives, or both. If the architecture meets the required throughput, then you can use the RTL output as the input to the Xilinx Vivado or ISE®/EDK tools. The reporting of final achievable clock frequency and number of FPGA resources used occurs only after you have run logic synthesis and place-and-route. If the design does not meet timing or the FPGA resources are not as expected, you should modify the implementation C/C++ code or the compiler directives.

It is worth noting that this is an iterative design flow, so the implementation code can go through different types of restructuring until the design requirements are met. A key concept to keep in mind is that the C-level verification infrastructure is reused to verify any change to the implementation. In this way, you do not carry out the verification at the register-transfer level, avoiding time-consuming RTL simulation and hence, contributing to a reduction in the overall development time.

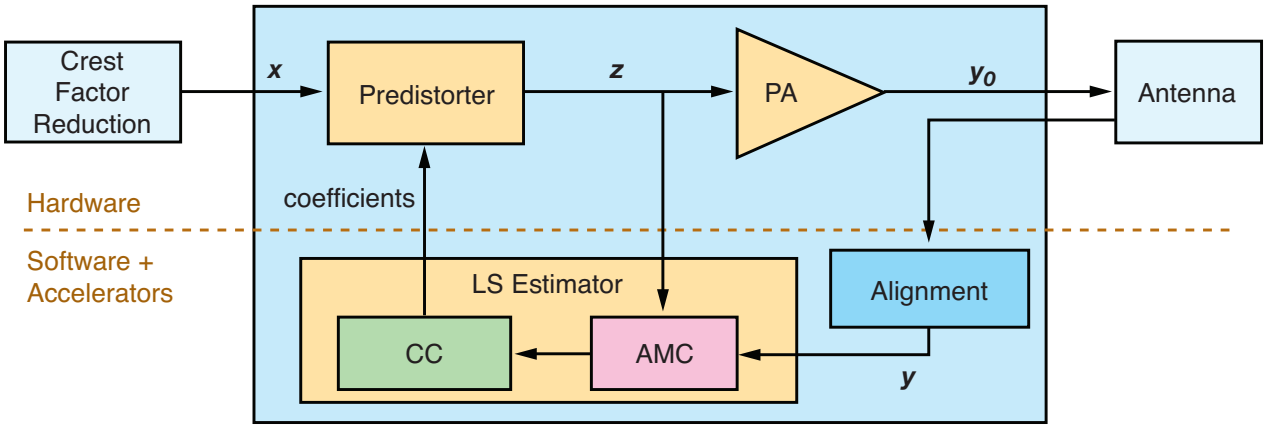


Figure 4 – An algorithmic view of the digital predistorter (DPD)

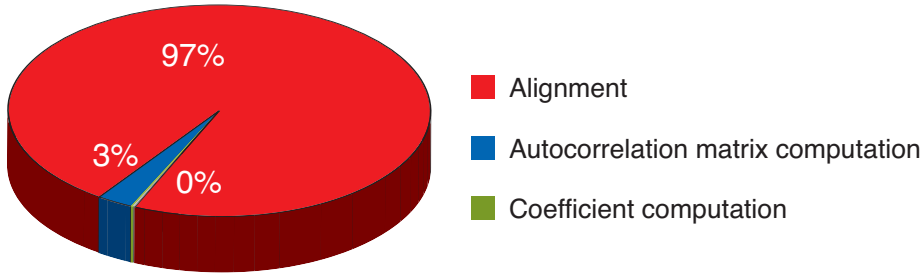


Figure 5 – Results of initial profiling

SYSTEM MODEL FOR DPD

High peak-to-average power ratio (PAPR) is a major problem of the non-constant envelope signals (for example, wideband code-division multiple access and orthogonal frequency-division multiple access signals) widely adopted in 3G/4G and emerging wireless systems. Due to high PAPR and PA nonlinearity, the transmitted signals get distorted during transmission. This distortion typically results in a growth of out-of-band spurious emissions. A straightforward solution to this problem is to back off the PA input so as to keep it in the linear operating range of the PA. However, the main disadvantage of this approach is the inefficient use of the PA, which results in a higher cost than required for the same output power. Another solution is to use digital predistortion. DPD negates the nonlinearity effects of the PA and increases efficiency.

As Figure 4 shows, a DPD system consists of a predistorter employed prior to the amplification and a parameter estimator on the feedback path from the output of the PA. (Please note that this illustration is an algorithmic view, which excludes the digital-to-analog and analog-to-digital converters at the PA input and output, respectively, as well as the RF circuitry in between.)

The parameter estimator computes the coefficients of the predistorter based on the samples of the PA input and output. To separate the PA behavior from the additional analog hardware effects, the PA output ***y0*** is aligned prior to the parameter estimation. The aligned PA output ***y*** matches the amplitude, delay and phase variations of ***z***.

The predistorter and parameter estimator rely on a memory model that is used to describe the nonlinearity effects of the PA. For wideband DPD applications, it is quite common to employ models based on the Volterra series, which is widely used for the approximation of nonlinear systems.

We consider the efficient implementation of alignment and a least-square (LS) estimator in Figure 4. The DPD system employs these blocks to update the predistorter coefficients when there are major changes in the signal characteristics or power dynamics. The autocorrelation matrix computation (AMC) block in Figure 4 is the most computationally complex function in our LS estimator design. It relies on a Volterra-series-based model and generates the inputs to the coefficient compute (CC) block, which estimates the predistorter coefficients.

Unlike the predistorter filter in hardware, the alignment and the LS estimator blocks do not operate at the sample rate, since they are not in use unless the DPD system updates the predistorter coefficients. Here, our main goal is to reduce the overall coefficient update time. In this way, the DPD solution reacts faster to changing conditions, leading to more-effective predistortion correction. Furthermore, faster updates enable the support of more-complex DPD solutions, using a larger number of active coefficients. With shorter update times, it is also possible to run the same design multiple times in a serial fashion, in order to update predistorter coefficients of different data paths. This approach makes it possible to implement efficient DPD solutions for multiantenna basestations.

Let's take a closer look at the details of our software-programmable design flow, which facilitates the implementation of efficient DPD coefficient update solutions for modern wireless transmitters.

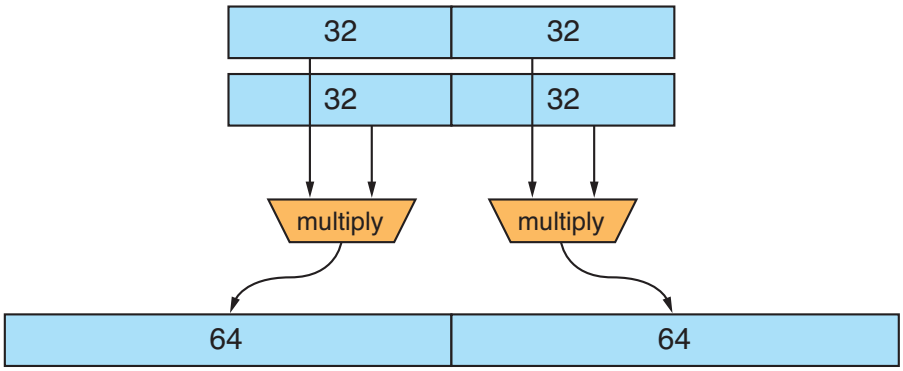


Figure 6 – Two parallel 32 x 32-bit multiplications using NEON

SOFTWARE IMPLEMENTATION FOR DIGITAL PREDISTORTION

During the software development process, we used a test environment that reads the ***z*** and ***y*** samples in

Figure 4 from a reference vector and writes a set of coefficients. Subsequently, we compared the coefficients to a reference implementation written in MATLAB that visualizes the difference between both sets of coefficients. We performed the software profiling in two levels. First, we ran the software on a standard x86 server and used the *gprof* software profiling tool, to get a first estimate of the expected bottlenecks. Second, we ran the software on the ARM processor and, depending on the *gprof* results, instrumented the subfunctions of interest with calls to the global CPU timer of the Zynq All Programmabale SoC. This timer runs at half the CPU frequency, hence giving excellent resolution with the overhead of only a few cycles. Figure 5 shows the profiling results of the three main function blocks running on the ARM processor, indicating that the AMC block is the bottleneck of the application. It consumes 97 percent of the overall update time, making it a prime candidate for hardware acceleration.

Prior to profiling, we expected the solver used for coefficient computation in Figure 4 would consume a larger part of the update time, because in contrast to the other functionality it was performing double-precision floating-point operations. However, the ARM's floating-point unit solved the task very efficiently.

Before actually implementing a hardware accelerator for the AMC block, we examined potential software optimization possibilities. The SIMD NEON engine of the ARM processor has a 128-bit-wide data path. Since the AMC algorithm works on 64-bit fixed-point data types, the NEON engine can carry out two parallel computations, as Figure 6 illustrates. Instead of using low-level assembly instructions to access the NEON engine, the compiler provides a set of functionlike wrappers for the instructions. These wrappers, which are called intrinsics, provide type-safe operations, while allowing the compiler to automatically schedule the C variables to NEON registers.

Applying the intrinsics in the C code results in a speed-up factor of two. Furthermore, during the NEON operations, the normal ARM processor is free and can continue processing simple non-NEON instructions like loop conditions and pointer increments, while the NEON engine runs in parallel.

```
1:  typedef struct {
2:      ap_int<32>   real;
3:      ap_int<32>   imag;
4:  } CINT32;
5:
6:  typedef struct {
7:      ap_int<64>   real;
8:      ap_int<64>   imag;
9:  } CINT64;
10:
11: CINT64 CMULT32(CINT32 x, CINT32 y){
12:       CINT64 res;
13:       ap_int<33> preAdd1, preAdd2, preAdd3;
14:       ap_int<65> sharedMul;
15:
16:       preAdd1 = (ap_int<33>)x.real + x.imag;
17:       preAdd2 = (ap_int<33>)x.imag - x.real;
18:       preAdd3 = (ap_int<33>)y.real + y.imag;
19:
20:       sharedMul = x.real * preAdd3;
21:       res.real = sharedMul - y.imag * preAdd1;
22:       res.imag = sharedMul + y.real * preAdd2;
23:       return res;
24: }
```

Non-standard bit-widths

3 pre-adders

3 multiplications instead of 4

Figure 7 – Optimized complex multiplication code

```
1:  void amc_accelerator_top(_)
2:  {
3:  #pragma HLS allocation instances=mul limit=3*UNROLL_FACTOR operation
4:
5:      <function body>
6:      CINT64 Marray[MSIZE];
7:
8:  #pragma HLS array_partition variable=Marray cyclic factor=UNROLL_FACTOR dim=1
9:
10:
11: #pragma HLS resource variable=Marray core=RAM_2P
12:     <function body>
13:     label_compute_M: for (int i=0; i < MSIZE; ++i)
14:     {
15: #pragma HLS pipeline
16: #pragma HLS unroll factor=UNROLL_FACTOR
17:             <loop body>
18:     }
19:     <function body>
20: }
```

Limit the number of multiplications

Partition based on the unrolling factor

Resource mapping

Loop pipelining and unrolling

Figure 8 – Code snippet for loop unrolling

HARDWARE IMPLEMENTATION FOR DIGITAL PREDISTORTION

To improve the overall parameter update time, we implemented an AMC accelerator using the Vivado HLS tool based on the design flow in Figure 3. Our accelerator's programmable configurations support a number of different predistorter coefficients and allow the flexible selection of nonlinear terms in the Volterra-series-based model. Hence, it is possible to support several DPD configurations using the same AMC accelerator. In addition, you can make new changes in the existing C++ code and in the compiler directives to generate a brand-new accelerator in a much shorter time than if you were doing a hand-coded RTL design. Let's take a closer look at some specific examples of code rewriting and compiler directives that we used for the AMC accelerator.

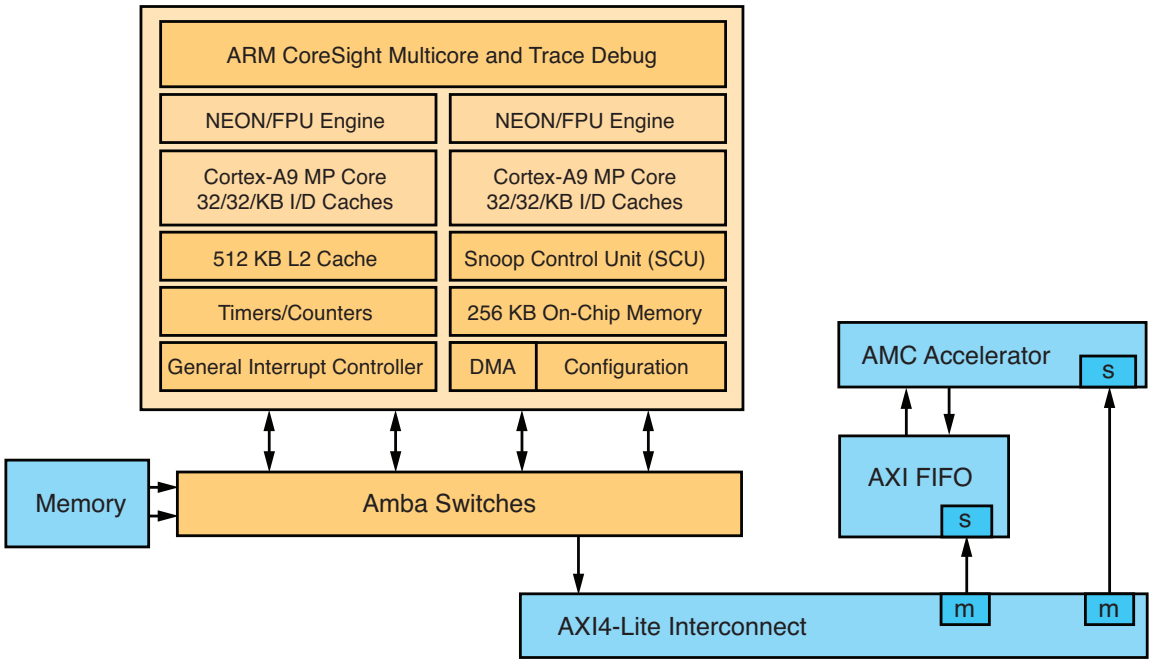


Figure 9 – Integration of the processor system with the hardware accelerator

It's possible to rewrite the C/C++ code to more efficiently utilize specific FPGA resources and, hence, improve timing and reduce area. There are two very specific examples of this type of optimization: bit-width optimizations and efficient use of embedded DSP blocks (DSP48s). For example, the standard approach uses built-in C/C++ data types (such as short, int) in the reference C/C++ code, whereas the actual design may require fixed-point data types having word lengths that are not integer multiples of the byte size. Here, the Vivado HLS tool supports C++ template classes that can represent integer data types with an arbitrary bit width. For our AMC accelerator, we leveraged these template classes, hence reducing FPGA resources and minimizing the impact on timing.

The snippet of C++ code in Figure 7 is a good example of code rewriting, bit-width optimization and the efficient use of DSP48s. The example focuses on complex multiplication, which we widely used in our AMC accelerator. A standard complex multiplication carries out four real multiplications, and requires the use of four different multipliers in a fully pipelined implementation. However, we show in Figure 7 that we can achieve the equivalent functionality by rewriting this code to use three multipliers (employing fewer DSP48 blocks), at the expense of three additional pre-adders and a 1-bit increase in the multiplier word length. In this example, the Vivado HLS tool generates three 33 x 32-bit multipliers, each giving a 65-bit result.

The original reference C++ code for the complex multiplication uses four multipliers, each multiplying two 32-bit numbers. Please note that the original code uses built-in C/C++ data types and the 32-bit inputs should be cast to 64-bit integer to avoid loss of information (because the result is a 64-bit integer). However, based on this code, the HLS tool generates four 64 x 64-bit multipliers, which are clearly much more expensive in terms of DSP48s. On the other hand, by using the C++ template classes in Figure 7 for the data types, the C++ code functionality works fine without any casting, while the Vivado HLS tool generates only three 33 x 32-bit multipliers, each using four DSP48s.

Our main reason for implementing an AMC accelerator was to reduce the time that it took to run the AMC algorithm, and hence to improve the total time to update predistorter coefficients. For this purpose, we pipelined the loops in the C++ code and also applied loop unrolling when possible. Using the Vivado HLS tool, we verified that the most time-consuming loop in our implementation was the matrix computation loop. We unrolled this loop by a configurable factor that we predefined as a C macro in the compiler options. Depending on the unrolling factor, the designer can choose between better resource sharing and shorter computation time.

Figure 8 shows how we employed the configurable unrolling factor (UNROLL_FACTOR) and the Vivado HLS directives for loop unrolling at the software level. In the matrix computation loop, we called the CMULT32 function in Figure 7, using three multipliers for complex multiplication. In Figure 8, the HLS directive on line 3 enables the

Performance results corroborate that our software-programmable design flow allows the implementation of several DPD designs, trading off faster predistorter coefficient update times.

sharing of 3xUNROLL_FACTOR multipliers in the case of loop parallelization. The directive on line 8 partitions the array that stores the matrix values by the unrolling factor, to avoid parallel access problems. Lines 15 and 16 show how we applied the loop pipelining and unrolling directives, respectively. Furthermore, Vivado HLS allows you to specify the resource to implement a variable in RTL. For example, we mapped the Marray variable to dual-port RAM on line 11.

INTEGRATING HARDWARE AND SOFTWARE COMPONENTS

The communication protocol between the processor system and hardware accelerator is the Advanced eXtensible Interface (AXI). The second and most-recent version of AXI is AXI4. Our design uses the Xilinx AXI interconnect to transfer data from the Zynq processing system (ARMs) to the hardware accelerator. Given the bandwidth requirements of this application, we don't need DMA transfers. As Figure 9 shows, we used the AXI interconnect core to link AXI4-Lite masters to slaves. AXI4-Lite is a lightweight, single-transaction memory-mapped interface. When connected to AXI4-Lite slaves, the AXI interconnect core stores the transaction IDs and restores them in the response transfers. Furthermore, it controls the transactions and does not propagate any illegal transaction to the AXI4-Lite slave.

The AXI FIFOs in Figure 9 are for the input and output data samples of the accelerator, which are using AXI4-Stream interfaces of the accelerator. The AXI4-Lite slave on the accelerator is for the AMC configuration parameters. We generated all the streaming and AXI4-Lite interfaces for the accelerator at the software level, using Vivado HLS.

PERFORMANCE RESULTS ON ZYNQ-7000 SOC

Figure 10 illustrates the total DPD coefficient update time at different design stages. Our target was the Zynq-7000 SoC ZC706 board using a Zynq 7045 device. The target clock frequency that we used in Vivado HLS to generate the AMC accelerator was 250 MHz. However, it is possible to increase the target clock frequency constraint in Vivado HLS in order to generate faster accelerators. Our implementation with Vivado HLS is quite efficient, resulting in 3 percent area utilization.

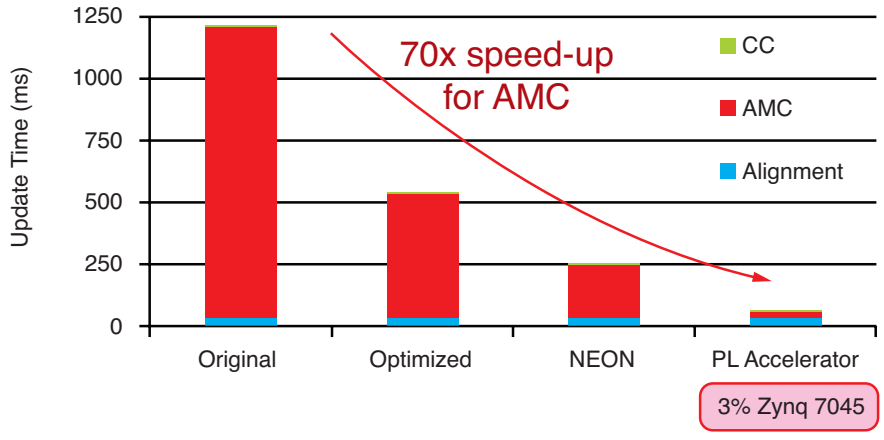


Figure 10 – Performance results on a ZC706 board

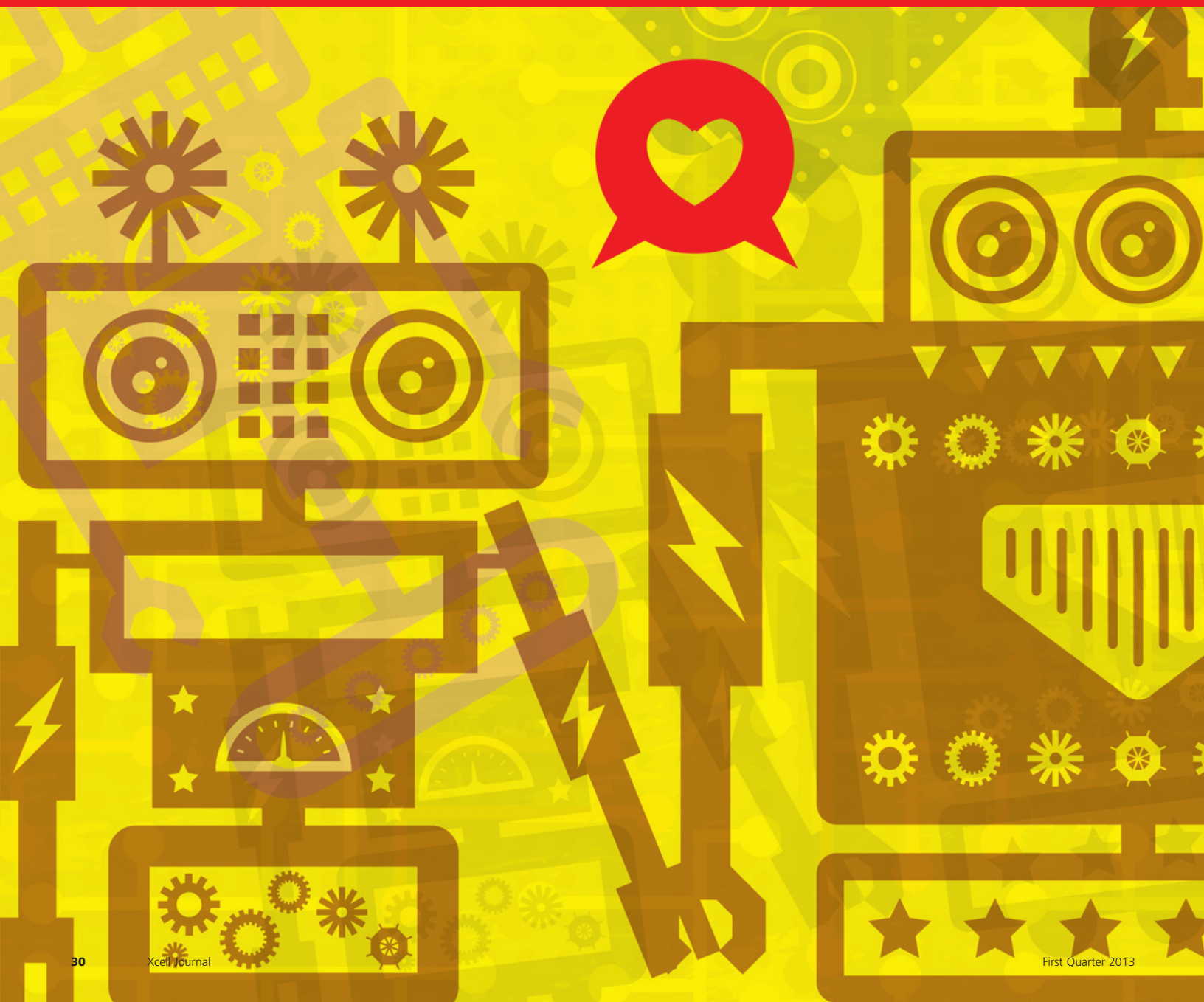
In Figure 10, the update time for the software-only solution using the original code is around 1.250 seconds, with the AMC as the main bottleneck for the application. After some code optimization for the AMC, we obtained a speed-up factor of about 2. NEON optimizations added another speed-up factor of 2. After accelerating the AMC in the programmable logic, we achieved a 70x speed-up for this block. We have tested our designs successfully on the target device.

Our accelerated design will be preferable in the high-complexity DPD applications going forward. For example, in a multiantenna basestation, it is possible to run an accelerated design more than once in a serial fashion (computing predistorter coefficients for a different antenna each time). The accelerated design becomes more feasible if the number of predistorter coefficients increases, as well. For low-complexity DPD applications, our software-only design using NEON optimizations can be sufficient.

Performance results corroborate that our software-programmable design flow allows the implementation of several DPD designs, trading off faster predistorter coefficient update times. Our design flow supports the flexible partitioning of functionality among hardware/software components and facilitates the implementation of efficient DPD solutions for modern wireless transmitters.

We would like to thank the Xilinx DPD Engineering Team for their valuable comments and cooperation.

How a MicroBlaze Can Peaceably Coexist with the Zynq SoC



by Bill Kafig
Senior Content Development Engineer
Xilinx, Inc.
bill.kafig@xilinx.com

Praveen Venugopal
Solutions Development Engineer
Xilinx, Inc.
pvenugo@xilinx.com

It's a fairly simple matter to add MicroBlaze processors to a design based on Xilinx's Zynq All Programmable SoC.

The Xilinx® Zynq™-7000 All Programmable SoC already has plenty of processing power onboard. But the presence of powerful twin Cortex™-A9 processors and associated peripherals in Zynq's application processing unit (APU) should not keep you from adding one or more MicroBlaze™ processors in the same package, if your application would benefit from them.

Why might you want to add a MicroBlaze to a solution already endowed with serious processing clout? First there is the issue of reliability. Single-threading dramatically improves reliability. You can cleanly place one thread per Cortex-A9 (for computationally intensive tasks), and instantiate as many MicroBlaze processors as you need for other threads. Second, you can farm out any housekeeping chores that don't require the power of a Cortex-A9 to a MicroBlaze, thus saving critical performance cycles for the jobs that need them most.

Here's an example that covers both of the above situations. Consider a task that requires long stretches of intense computing while monitoring user input. Here the MicroBlaze could manage the user input (lower frequency, non-computationally intensive) and write into the APU's memory space so that when the APU "comes up for air"—that is, completes its processing task—it can see what information it needs to process next.

Once you've made the decision to include a MicroBlaze processor in your Zynq-based design, several issues become immediately apparent. First and foremost is the question of how the APU will communicate with the MicroBlaze, and what processing system (PS) resources are available for the MicroBlaze. Many boards, such as the ZC702 and Zedboard, map many of the peripherals directly to the pins connected to the PS. These pins are not directly accessible to the MicroBlaze in the programmable logic (PL). The PS also contains a variety of timers and interrupt sources. Is there any way to access them from the domain of the MicroBlaze?

INTERFACES BETWEEN THE PS AND PL

The processor system and the programmable logic are well coupled. This means that there are multiple tightly integrated connections between the Cortex-A9s, snoop control unit (SCU), PS peripherals, clock management and other functions, and the programmable logic. In fact, there are six different types of interconnects between the PS and the PL, and you can use them in conjunction with one another. Additionally, many of

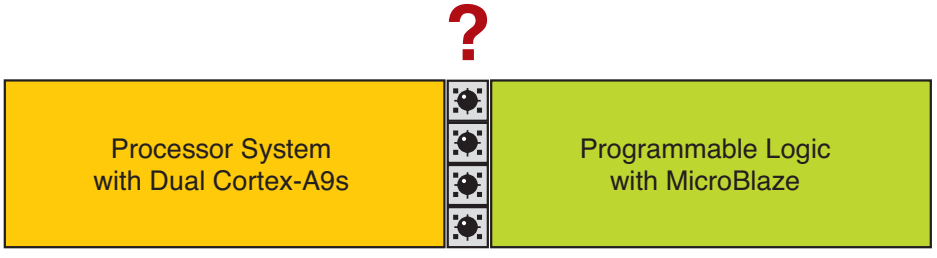


Figure 1 – Is the boundary between the PS and the MicroBlaze within the PL a minefield, or can the two share resources?

these paths are symmetric—that is, the PS can initiate or "master" connections to the PL and the PL can master connections to the PS.

Much of the information presently available from Xilinx, from app notes to user guides and white papers, illustrates how the Zynq-7000 APU, as the "center" of the design, can use the programmable logic to access memory, PL-based peripherals and hard silicon peripherals such as the PCIe® block, Block RAMs, DSP48s and multigigabit transceivers. In examining how the MicroBlaze can be the captain of its domain, the logical place to begin is by looking at the six interface varieties, starting with three types of AXI interfaces: general purpose, high performance and the Accelerator Coherency Port.

The PS is equipped with two master AXI channels to the PL and two slave channels mastered by the PL (Figure 2). "Master" in this context means that the AXI channel is the initiator and can begin data exchanges, whereas a "slave" can only respond to arriving data. The master AXI channels are typically used to communicate with peripherals located in the PL. The slave AXI channels respond to requests made from the PL, which can include transactions made by MicroBlaze processors. These AXI channels tie into the central interconnect of the PS and can be routed to many resources.

In addition, there are four channels of high-performance (64-bit-wide) AXI attachment points. All four of these channels are slaves from the PS' perspective and are connected to the memory interface subsystem within the PS (Figure 3). The purpose of these four channels is to allow masters in the PL to initiate double-data-rate (DDR) memory transactions.

This memory interconnect and DDR memory controller are the gateways to the DDR memory from all sources. While the Cortex-A9 processors usually have priority over the slave AXI connections, each one of the four slave AXI connections has a "service me now" signal that gives priority to the requesting channel. When this signal is not asserted, the architecture uses a round-robin scheme to determine which requestor can gain access to the specific type of memory.

The Accelerator Coherency Port (ACP) is another 32-bit AXI PS slave connection from the PL. What makes the ACP unique is that it is tied directly into the snoop control unit (SCU). The job of the SCU is to ensure coherency among the L1, L2 and DDR memories. Using the ACP, you can access the fast cache memory for each of the Cortex-A9 processors in the PS and not be concerned with synchronizing data with the main memory (as the hardware will automatically take care of this). This capability greatly reduces the burden of design and provides a significantly faster way of moving data between the processors and the PL.

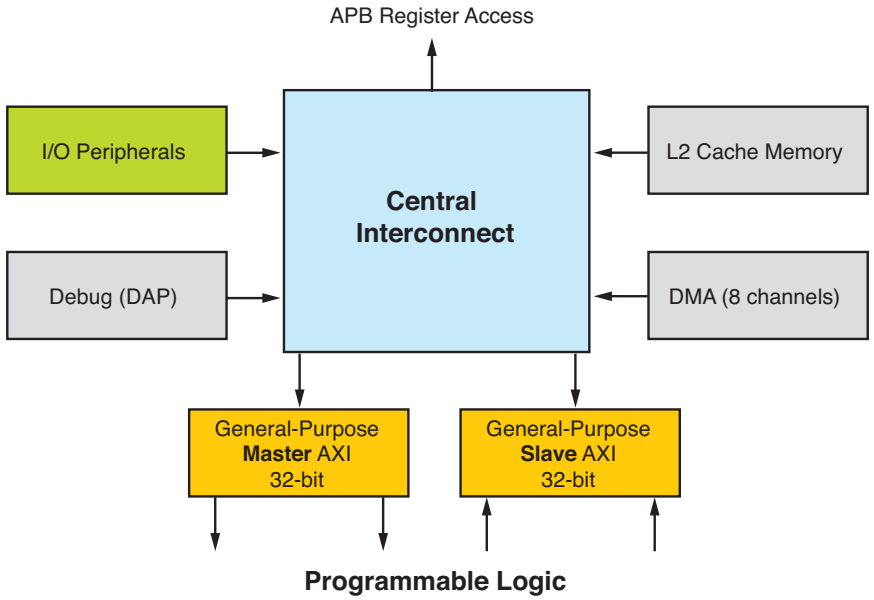


Figure 2 – Simplified connections to the processing system's central interconnect

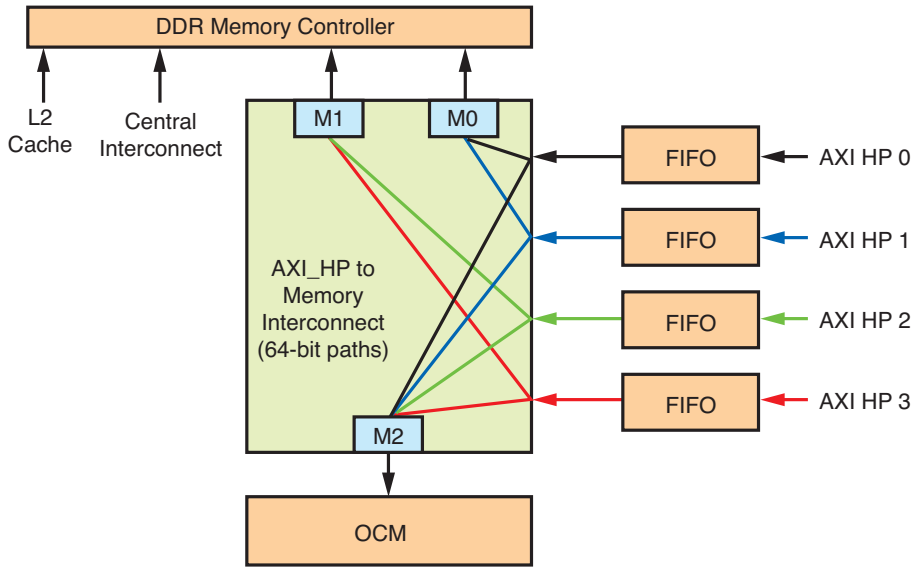


Figure 3 – Simplified connections to the DDR memory controller and on-chip memory (OCM)

Beyond AXI links, the Extended Multiplexed Input and Output (EMIO) signals are available for routing many of the PS' hard peripherals through the PL to access the package pins. There are only 54 package pins tied directly to the PS; however, the PS' hard peripherals can use considerably more than these 54 pins. The EMIO is the conduit between the PS' hard peripherals and the PL. These I/O signals can be routed directly to the package pins available to the PL. Alternatively, you may use them to communicate with a compatible peripheral located in the PL.

Another variety of miscellaneous signals between the PS and PL falls into five basic categories: clocks and resets; interrupt signals; event signals; idle AXI; DDR memory signals; and DMA signals.

- Clocks and resets: There are four independent programmable frequencies that the PS makes available to the PL. Typically one of these clocks is used for the AXI connections. Each of these clock domains has its own domain reset signals for resetting any device associated with that domain.
- Interrupt signals: The general interrupt controller (GIC) in the PS collects interrupts from all available sources, including all of the interrupt sources from the PS' peripherals and 16 "peripheral" type interrupts from the programmable logic. Additionally, there are four direct interrupts that tie to the CPUs (IRQ0, IRQ1, FIQ0 and FIQ1). A total of 28 interrupts (from the PS' peripherals) are available to the PL.
- Event signals: These "out-of-band" asynchronous signals indicate a special condition of the PS. The PS provides a number of signals that indicate which CPU has entered a standby mode and which CPU has executed a SEV ("send event") instruction. The PS can leverage an event signal to wake from a WFE ("wait for event") state.
- Idle AXI and DDR memory signals: The idle AXI signal to the PS is used to indicate that there are no outstanding AXI transactions in the PL. Driven by the PL, this signal is one of the conditions used to initiate a PS bus clock shutdown by ensuring that all PL bus devices are idle. The DDR urgent/arb signal is used to indicate a critical memory-starvation situation to the DDR arbitration for the four AXI ports of the PS DDR memory controller.
- DMA signals: The direct-memory-access module within the PS communicates with the PL slaves via a series of request-and-acknowledge signals.

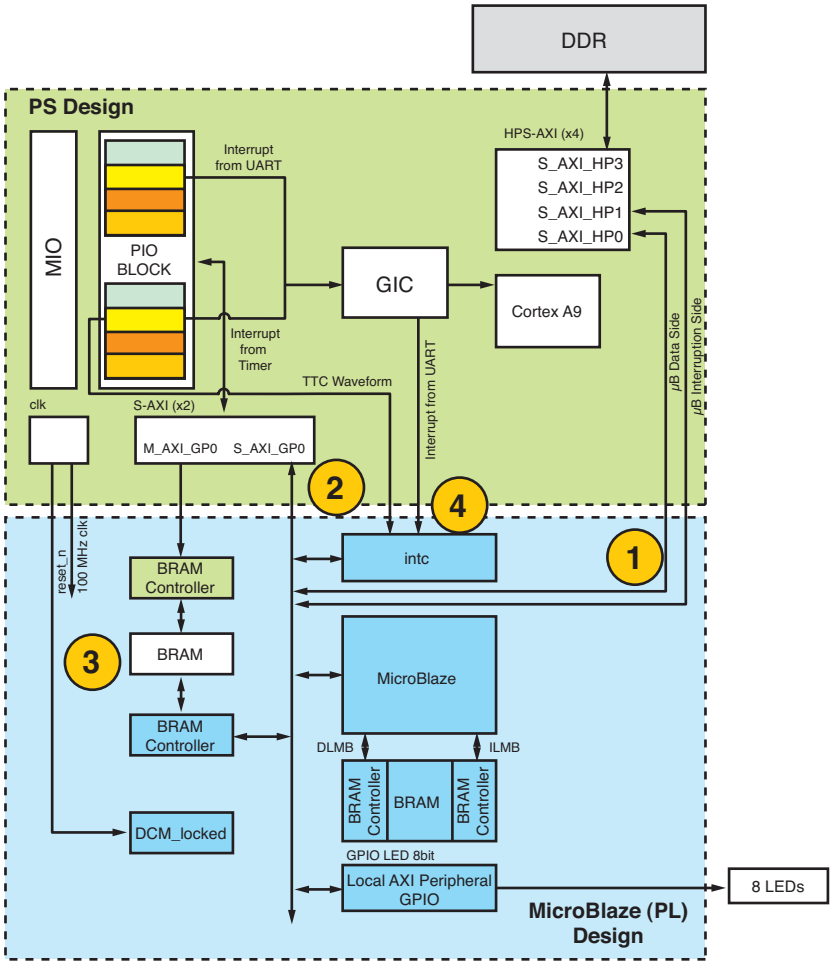


Figure 4 – Block diagram of example hardware design; the numbers represent (1) ways to access DDR memory; (2) use of the peripherals in the PS' IOP; (3) how to pass blocks of data between the MicroBlaze and the PS; and (4) how to synchronize events between the MicroBlaze and the PS.

ACCESSING DDR MEMORY

Let's take a look at an example design that covers several common needs the typical MicroBlaze user might have, including how to access DDR memory, how to use the peripherals in the PS' IOP, how to pass blocks of data between the MicroBlaze and the PS, and how to synchronize events between the MicroBlaze and the PS. Figure 4 shows specific techniques to address each of these issues (labeled 1-4, respectively).

The easiest way to access the DDR memory is to connect through one or more of the four high-performance (HP) AXI interfaces (top right section of the diagram). Four 64-bit-wide ports are accessible to the programmable logic. You must first enable a port; then you can connect an AXI to it. The

MicroBlaze utilizes the AXI4-Lite interface, whereas the HP ports are expecting full AXI4 connections. Fortunately, the Xilinx design tools automatically compensate to make a successful connection without any manual modification. This connection is shown in the block diagram at circle "1." The advantage in making this type of connection is that it is very easy, and the DDR memory appears to the design as regular memory. The disadvantage is that AXI-Lite doesn't support burst mode and is only 32 bits wide, so you lose quite a bit of performance.

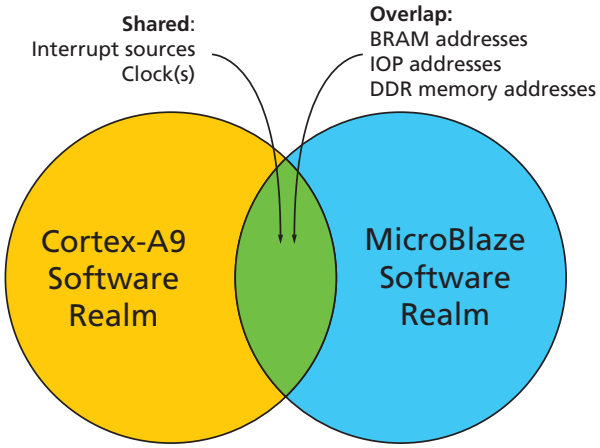


Figure 5 – Independent software realms and items of overlap

There are other ways to connect PL-based peripherals to the DDR memory as well, including the use of a DMA controller, likewise by means of the HP ports. While more complex, this mechanism offers better performance for transferring large blocks of data.

Within the PS, a UART and a triple timer/counter (TTC) are enabled. The M_AXI_GP0 connects to a BRAM controller that resides in the PL but is conceptually (and practically) part of the PS design.

The other portion of the design, the MicroBlaze, uses BRAMs to provide 64 Kbytes of combined code and data space for exclusive use by the MicroBlaze. It is certainly possible to connect these ports to the HP ports of the PS in order to access the DDR memory. However, doing so invites additional complexity in the implementation.

There are three standard peripherals attached to the MicroBlaze—an interrupt controller that receives the TTC's PWM waveform input and interprets it as an interrupt source and the UART-character received interrupt; a GPIO that drives the eight LEDs on the ZC702 board; and a BRAM controller that connects to the "B" side of the same BRAM that the PS connects to. Then there are two additional connections: one to the S_AXI_GP0 so that the peripherals internal to the PS can be addressed, and one to the high-performance ports so that the MicroBlaze has access to a portion of the DDR memory.

PERIPHERALS, DATA BLOCKS AND SYNCHRONIZATION

The key to accessing the programmable system's IOP block is the connection to the PS' S_AXI_GP0 port (circle "2" in the figure). The "S" denotes that this is a slave port—one that can accept transactions that elements in the PL initiate. When you make the connection between the MicroBlaze and the S_AXI_GP0 port, the IOP block spans the range of 0xE000_0000 to 0xE02F_FFFF. This means that no other peripherals in the MicroBlaze's address space can overlap this range of addresses, as they will be shared between the processors.

Meanwhile, a number of mechanisms support passing blocks of data between the MicroBlaze and the PS—DMA to and from DDR and on-chip memory (OCM) being but two possibilities. Another option, as implemented in the example design, uses a dual-port Block RAM (circle "3" in the figure). The PS masters one side of a BRAM controller while the MicroBlaze masters the other side. Software manages partitioning and use of the shared BRAM memory space, since both sides have full read/write access to the entire contents of this memory.

The final issue involves how to synchronize events between the MicroBlaze and the PS. The PS contains a plethora of timers, and you can instantiate virtually any number of timers in the PL. The example design uses one channel within one of the triple-timer/counter peripherals to generate a 100-millisecond pulse (circle "4" in the figure). The trick here is to realize that the TTC generates a waveform that is accessible outside the PS, but the TTC interrupt is only available inside the PS. You can get around this issue by using the waveform itself as an interrupt to the MicroBlaze processor. Alternatively, you may employ a hardware timer (whether an AXI timer or one that's user-coded) to provide an interrupt to both the PS and the MicroBlaze.

SOFTWARE SUPPORT

As with any embedded design, you must consider not only the hardware implementation, but the software support as well. For this example, consider the areas of overlap between the two processors, as outlined in Figure 5.

In terms of the hardware, think of the system as two intersecting embedded designs: the PS and the MicroBlaze. Our reference design has the PS generating the clocks for both itself and the MicroBlaze as well as sourcing the interrupts. The MicroBlaze can use

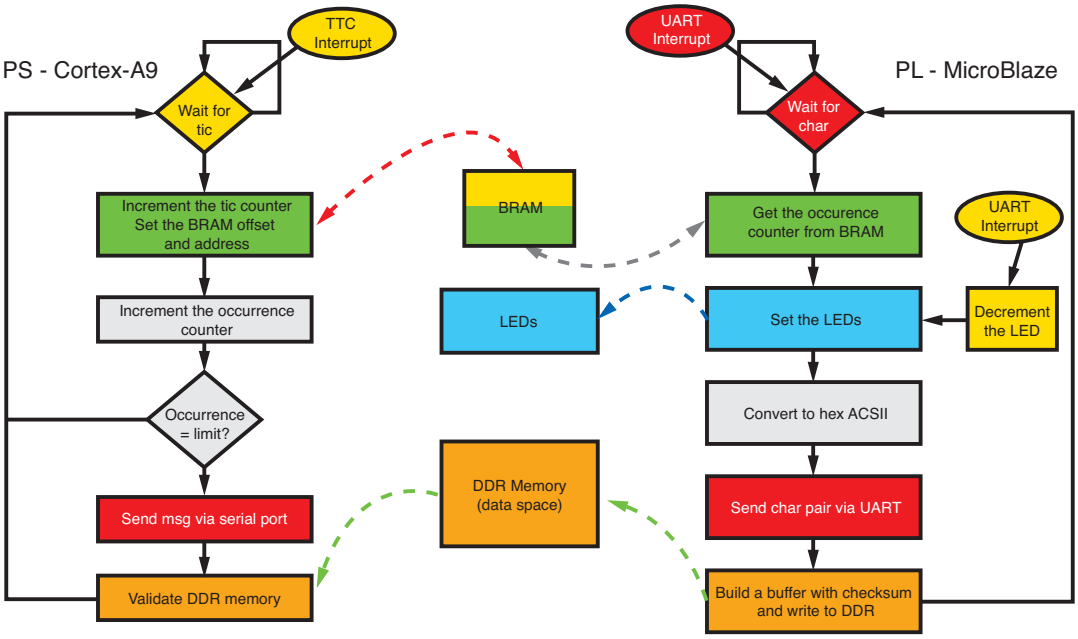


Figure 6 – Flow diagram for both the Cortex-A9 application and the MicroBlaze application

these signals, but it can't influence or alter them. Both the PS and the PL share the PS' IOP block, the DDR memory and controller, and the common dual-port BRAM.

From an address map standpoint, the IOP addresses are fixed in hardware and cannot be modified. Both the MicroBlaze and the Cortex-A9 must use the same addresses for any of these peripherals. The Technical Reference Manual is especially helpful in this regard. There are two portions of the TRM that are useful for dealing with peripherals in this design: the sections on the specific peripheral (such as Section 8 – Timers and Section 19 – UART Controller), and Appendix "B," which lists all of the registers for each of the peripherals. Each register in this section is listed by its address along with a brief description. It is left as an exercise to the user to manage the resources between the two processors to avoid data collisions, starvation and so on.

Use of the DDR memory and its controller is a bit easier, since the DDR memory controller is capable of managing collisions and has appropriate heuristics to avoid data starvation for any of the requestors. Since this is not under user control, the latency may become a bit unpredictable.

The common BRAM will have identical lower address bits; however, the location in each processor's memory space for this block is selectable using the address tab in Xilinx Platform Studio. As before, data collisions, synchronization and memory allocation are left as an exercise to the reader.

Our example software design is more about demonstrating the process of connecting a Cortex-A9 with a MicroBlaze than doing anything practical. A couple of techniques will illustrate some of the possibilities in this regard.

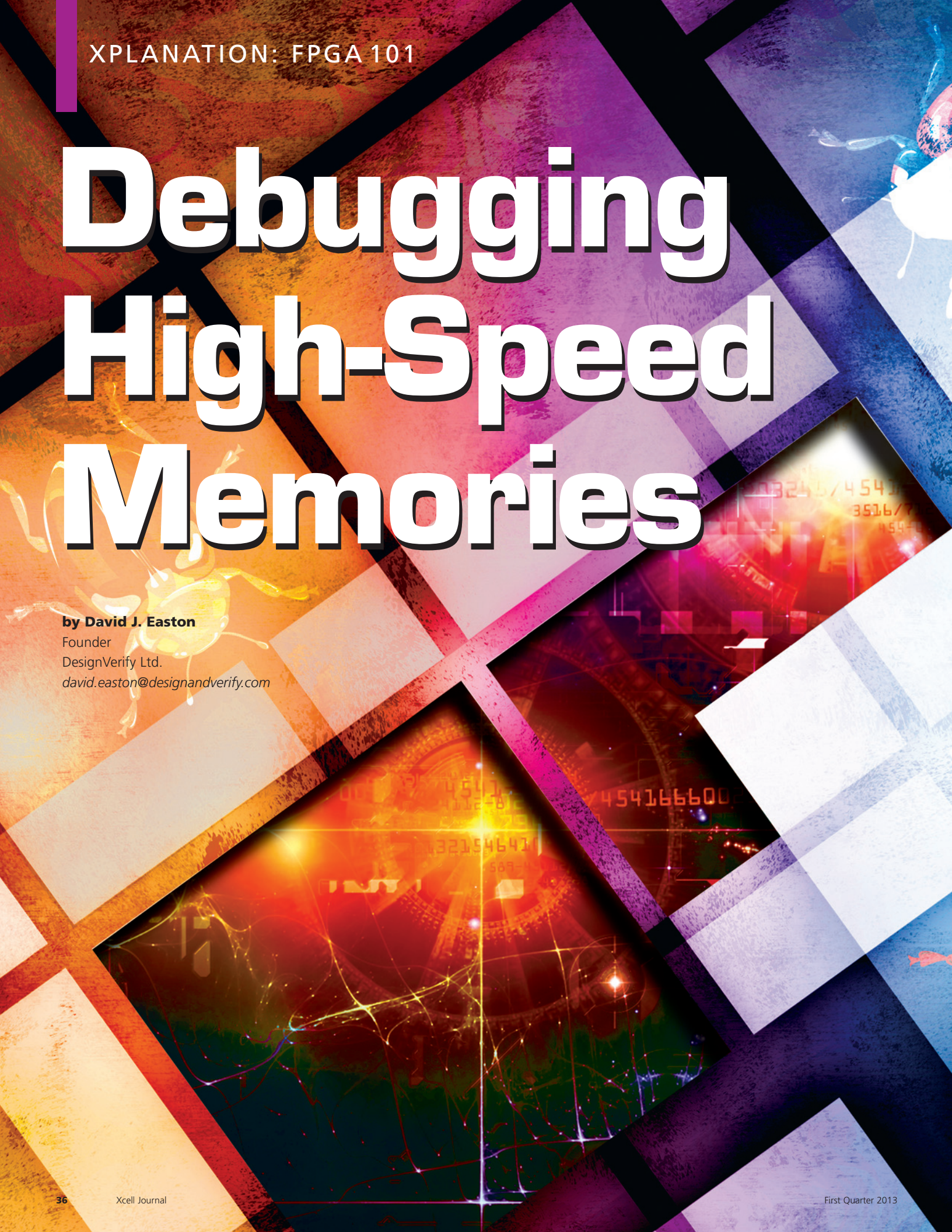
The Zynq-7000 application shown in the left side of Figure 6 is a simple design that uses the Xilinx Standalone board support package. It runs a program that, for the most part, keeps the processor idling and periodically issues a "T" character (for "tic") via the serial port to indicate that it is still alive. Additionally, it counts the number of elapsed tics and places this count in the Block RAM that is shared with the MicroBlaze.

The MicroBlaze application also runs a Standalone board support package. This approach supports interrupts, and each time a character appears on the serial port the MicroBlaze buffers that character and echoes its Hex-ASCII equivalent back to the serial port in the PS' IOP block. Additionally, it builds a string of received characters and places it in the DDR memory for each character received. When a carriage return is received, a checksum is computed and added to the DDR memory immediately following the string. A flag is set in a predetermined location in BRAM and when the Cortex code "sees" this flag, it reads the string from the DDR memory and verifies the checksum. If the checksum is correct, the MicroBlaze emits a "+." Otherwise, it transmits an "X."

In short, our example design proves that the MicroBlaze and Zynq-7000 PS can coexist peacefully. The copious number of AXI connection points provide the MicroBlaze (or anything in the PL) easy access to the peripherals in the IOP block, the OCM and the DDR. The programmable logic can easily access interrupts from the IOP block, just as a number of interrupts can be generated in the PL and supplied to the PS. Software coordination and well-defined behavior are key to avoiding race conditions and addressing conflicts.

Debugging High-Speed Memories

by David J. Easton
Founder
DesignVerify Ltd.
david.easton@designandverify.com



Many FPGA designs have high-speed memory interfaces that can be difficult to debug, but the right methodology will ensure success.

Modern FPGAs often have high-speed SRAM and SDRAM memories connected to them. These devices can be difficult to debug to ensure error-free operation. There are a number of factors that all have to be correct to guarantee a successful working memory design, including the board layout, the power supplies and the memory interface circuitry within the FPGA.

You may encounter several problems while debugging SRAM and SDRAM memories that might leave you scratching your head for days. It is essential to have a good debug methodology in place when designing and commissioning high-speed memory interfaces. The designs that I will be describing here as models for an efficient flow used Xilinx® Virtex®-4 and Virtex-5 FPGAs, but the problems and solutions are equally valid for Virtex-6 and 7 series devices too. My colleagues and I used the Xilinx Memory Interface Generator (MIG) tool to generate all the memory IP cores, and the Xilinx ISE® tools to synthesize, place and route the designs.

Before examining specific design features in detail, let's take a look at the Xilinx MIG tool, the memory calibration process specific to Xilinx FPGAs and the built-in self-test (BIST) circuitry that we used to validate the memory interface designs.

XILINX MEMORY INTERFACE GENERATOR

Xilinx provides a Memory Interface Generator tool to generate memory interface cores for its FPGA devices. The MIG tool, which is invoked from the Xilinx CORE Generator™, can produce interfaces for many types of memories including DDR-II SRAM, DDR2 SDRAM and QDR-II SRAM, to name a few. Figure 1 shows a Xilinx CORE Generator screenshot detailing the Xilinx families and memory technologies supported in MIG3.6.1. There is also a 7 series MIG tool for the newest Xilinx devices.

The MIG relies on a graphical user interface (GUI) to input the details of the memory and FPGA types. The tool outputs register-transfer-level (RTL) files in either VHDL or Verilog, along with a User Constraints File (UCF) for the memory interface. You can then integrate both of these types of files with the rest of your design. The MIG also generates an infrastructure block that provides the clocks and resets required for the memory interface. In addition, the tool can provide testbench circuitry with which to generate writes and reads to the memory device to verify correct operation. You can download a full description of the MIG tool, and details of the FPGAs and memories it supports, from the Xilinx website.

CALIBRATION CONCERNS

To ensure that the data is correctly captured from the memory device into the FPGA, the memory interface core needs to calibrate the read datapath before it can be used. The calibration process takes place automatically on power-up. Essentially, a data training pattern is written to the memory device and then continually read back. This is a three-stage process in which

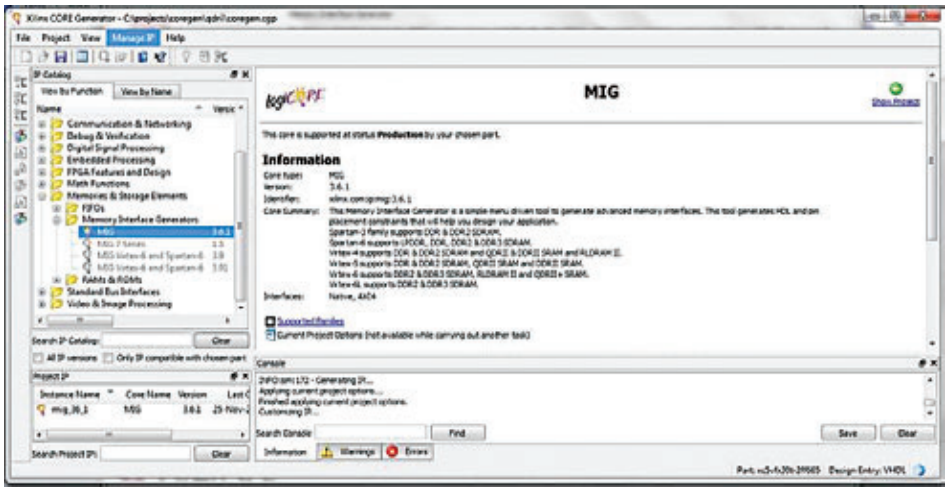


Figure 1 – The CORE Generator MIG tool serves many generations of Xilinx FPGAs.

the first stage centers the read-data window with respect to the data strobe signal, clocking the data into the FPGA. The second stage ensures that the centered data and clock are synchronized to the FPGA clock domain so that the data can be transferred between the input flip-flops and the flip-flops in the FPGA fabric. The third stage then provides a read-valid signal. The Xilinx MIG documentation offers full descriptions of the calibration process for the different types of memories.

CLOCK GENERATOR AND BIST

We generated all of our memory interface designs using the Xilinx MIG tools. But we diverted from the MIG flow in a couple of instances. The FPGAs that we were using resided on boards that are designed for a variety of users and for a variety of applications. For this reason, we designed a custom clock generator and reset module to provide all the clocks and resets required for user applications. This meant removing the MIG-generated infrastructure module and supplying the clocks and resets to the memory interface from our custom module.

We also designed our own custom BIST module to fully test the memories by writing and reading every memory location using various data patterns, including 0s, Fs, As, 5s, walking 0s, walking 1s, sequential data and pseudorandom binary sequence (PRBS) data. The BIST circuitry runs under control of a host interface, and reports back any errors found. The BIST circuitry can store the address of any data errors found, and also signals which data bit or bits are in error. Figure 2 shows a block diagram of the generic design, with the MIG core being specific to the memory technology on the board—that is, DDR-II SRAM, DDR2 SDRAM or QDR-II SRAM.

It bears repeating that for any FPGA memory interface to operate correctly, it is essential that you follow the PCB layout recommendations for both the FPGA and the memory device. It is important to match the track lengths between the devices to minimize skew, and also necessary to minimize crosstalk between signals and to provide adequate decoupling for all devices. The FPGAs and memories have a number of power supplies, which are typically powered using a mixture of switching and linear voltage regulators. It is important to follow the manufacturers' guidelines to ensure correct operation of all of these devices. You will recoup any time you spend studying the details of the schematics and layout of the power supplies to the FPGAs and memories in reduced debug time.

Now, let's take a closer look at three examples of memories that exhibited bit errors when we tested them using the BIST circuit previously described. All the examples exhibited similar symptoms, which were bit errors that occurred after a successful calibration of the memory device. However, in each example, the error had a different cause. We found these bit errors by repeatedly running the BIST tester. It should also be mentioned that all the boards had successfully passed JTAG testing.

EXAMPLE 1: VIRTEX-4 BOARD WITH DDR-II SRAM

This board used a Virtex-4 FPGA with a DDR-II SRAM memory operating at 200 MHz. The memory interface calibrated successfully, and then when the BIST started running, single bit errors occurred. These errors were

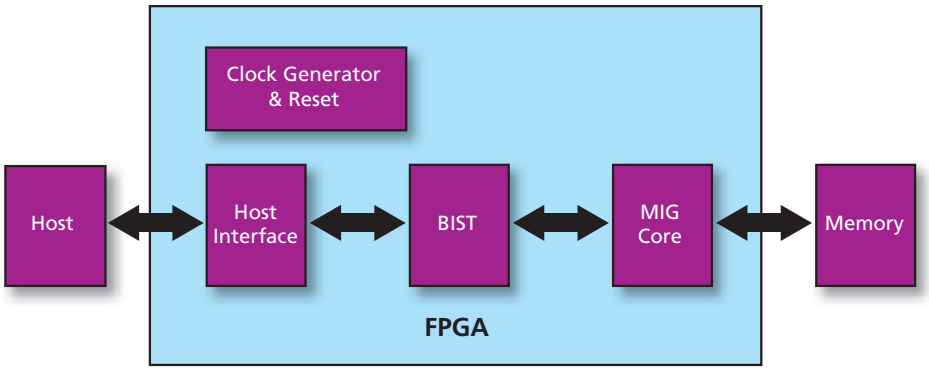


Figure 2 – The MIG core in this memory interface block diagram will be specific to the memory technology on the board.

cropping up at different memory locations and in different bit positions within the data word.

The first thing we did was to measure all the power supplies to the FPGA and memory, and we found them all to be within the correct voltage specification. We then checked that the oscillator on the board was the correct frequency, and that there was no excessive jitter on the signal. Next, we decided to check the voltages again, using an oscilloscope when the BIST was running.

The memory power supply VDD is supplied by a 1.8-volt regulator, and we discovered that the output voltage from the regulator had excessive ripple that was going out with the specification of the memory device. We checked the regulator circuit and found that it had an incorrect value for the output-smoothing capacitor, being only 10 percent of what was required. When we replaced it with the correct value, the ripple dropped to acceptable levels. As a result, the voltage came within the specification and the bit errors disappeared.

EXAMPLE 2: VIRTEX-5 BOARD WITH DDR2 SDRAM

Our second example board used a Virtex-5 FPGA with DDR2 SDRAM memory operating at 250 MHz. The memory interface calibrated successfully, but when the BIST started running, single bit errors occurred. These were taking place at different memory locations and in different bit positions within the data word, but then gradually, every bit within the data word failed.

Just as in the first example, we checked all the power supplies and clocks on the board and found these to be within specification. Again, we decided to check all the power supply voltages when the BIST was running. The memory devices have a voltage reference input VREF, supplied by a voltage regulator. This regulator was also supplying some other circuitry. When all circuits were active, the regulator was slightly overloaded and the voltage on VREF was falling below the specification. We modified the regulator circuitry to power it using a different supply, and that change kept the VREF voltage within specification. Once again the bit errors disappeared.

EXAMPLE 3: VIRTEX-5 BOARD WITH QDR-II SRAM

Our third board used a Virtex-5 FPGA with a QDR-II SRAM memory operating at 250 MHz. In this case, there were two failure mechanisms. The first was that occasionally the memory interface would fail to calibrate. The memory interface outputs a cal_done status bit that is asserted when calibration completes. The user needs to monitor this process to determine that the memory interface is ready to use.

The second failure that we were seeing was that even though the memory interface completed the calibration stage, repeated BIST testing was reporting bit errors. Once again, we checked the voltages and clocks on the board, and then we started looking at the FPGA RTL design. Because the calibration was failing, this pointed toward something going awry shortly after the power-up or upon reset of the FPGA, as the calibration happens immediately afterward.

The QDR-II memory interface that the Xilinx MIG tool generates has a SIM_ONLY generic, which is set at the top-level module when it is instantiated in the design. Its purpose is to bypass a 200-microsecond power-on delay during simulation, so as to speed up your simulations. When the memory interface is used in hardware, you must set the SIM generic at 0 to enable the 200-µs delay, and when simulating you should set it to 1 to bypass it.

The QDR-II memory device has input clocks K/K#, which are provided by the memory interface, and output clocks CQ/CQ#, which are generated by an internal phase-locked loop (PLL) in the memory device and are also referenced to the K/K# clocks. The CQ/CQ# output clocks are used to clock the data into the FPGA. The PLL needs to be locked correctly, and this typically requires around 20 µs of stable K/K# clocks.

What we discovered was that we had incorrectly set the SIM_ONLY generic so that the calibration process was running immediately after the system reset, at which point we were also just starting to output the K/K# clocks. The calibration process was running before the PLL was locked, and the CQ/CQ# output clocks would not be stable. Sometimes the calibration would pass

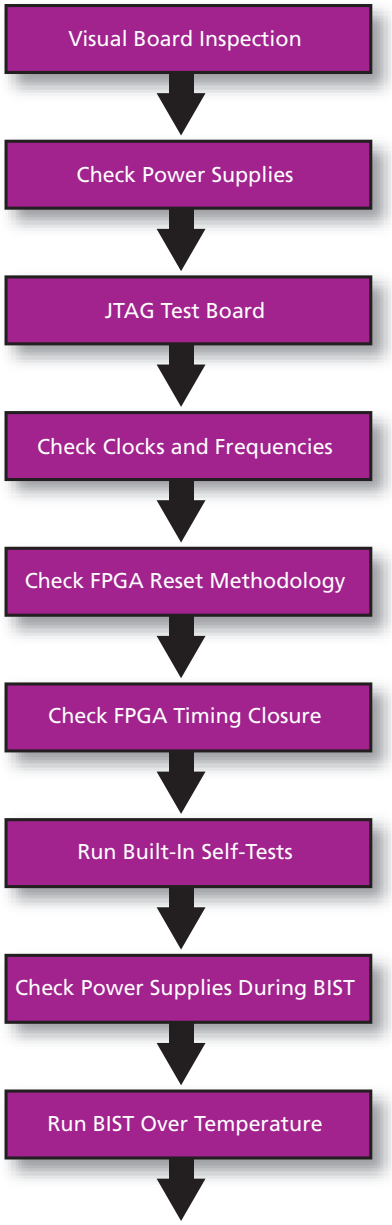


Figure 3 – A typical debug flow like this one will uncover problems in both hardware and firmware.

and sometimes it would fail, but even when it passed it would not have been completed optimally; hence the bit errors during subsequent BIST runs. Correcting the SIM_ONLY generic allowed 200 µs for the K/K# clocks to stabilize. Once we had corrected the SIM_ONLY generic, our calibration and bit-error problems disappeared.

I am embarrassed to admit that this SIM_ONLY issue has caught me out more than once, much to the amusement of colleagues. To avoid the same problem, make sure that the top-level design has this generic set to always enable the power-on delay, and that the simulation testbench always has the delay skipped while simulating. Do not design it so that an engineer has to remember to modify the SIM_ONLY generic before either simulating or building a design for hardware testing. He or she will forget.

TIPS ON METHODOLOGY

A good methodology—in our case, using PCBs with the board layout done correctly—is mandatory when designing and commissioning high-speed memory interfaces. We also used FPGA designs with timing constraints correctly applied to the complete design, and ensured that we achieved timing closure using the ISE tools. It cannot be stressed enough that if you have not achieved timing closure, then all manner of timing errors may surface and obfuscate any underlying problems.

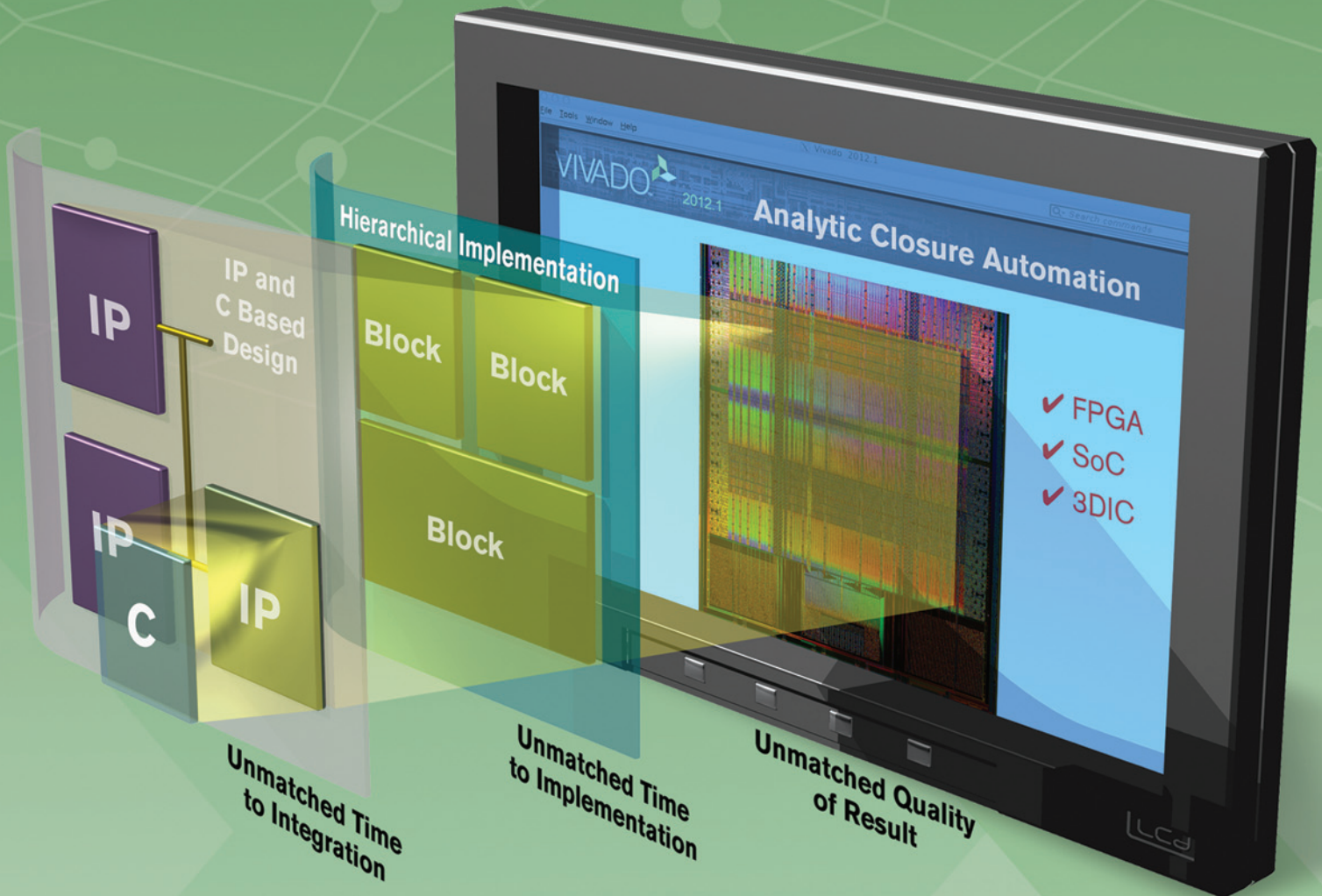
The problems we found in our three examples were relatively straightforward to unearth and correct, but this is not always the case. Both hardware and firmware can be the source of problems in FPGA designs involving high-speed memory.

Figure 3 outlines a typical debug methodology to use. Unfortunately, sometimes things go awry when the pressure of commissioning new hardware kicks in, and a few things may fall through the cracks—especially the SIM_ONLY generic!

DesignVerify Ltd. is a Scottish-based consultancy specializing in the design and verification of FPGA and CPLD devices. For more information, see *www.designandverify.com*.

A Generation Ahead

Vivado Design Suite WebPACK™ Edition Delivers Unmatched Time to Integration and Implementation



LEARN MORE

XILINX
ALL PROGRAMMABLE™

Nuts and Bolts of Designing an FPGA into Your Hardware

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

Many engineers feel the job is done once they've defined the functionality of their FPGA. However, inserting that FPGA into your PCB can provide its own set of challenges.

To many engineers and project managers, implementing the functionality within an FPGA and achieving timing closure are the main areas of focus. However, actually designing the FPGA onto the printed-circuit board at the hardware level can provide a number of interesting challenges that you must surmount for a successful design.

It all begins with the architecture. The initial stage of hardware development involves defining the architecture of the solution. This architecture should be a response to the system requirements, detailing how they are to be implemented within the hardware. Although this architecture will vary from system to system—and each system will vary from application to application—many will contain similar architectural blocks. You can and should reuse frequently needed hardware modules, in just the same way that you reuse common HDL modules.

Figures 1 and 2 show examples of an overall architecture and a power architecture, respectively, while the sidebar lists common considerations to take into account when designing an FPGA-based system.

DEVICE SELECTION

The most important choice you will face at the outset is selecting the correct FPGA from the large number available. There are many factors that will drive the choice of the device. The first and most important factor is the resources available within the FPGA and whether they are sufficient to implement the functionality you desire at the operating frequency required.

These parameters will quickly thin down the field to a manageable number of suitable devices, in a process that allows you to apply further selection criteria to these correctly sized FPGAs. Another driving factor will be additional resources your system might require, such as DSP slices or multipliers, embedded processors or high-speed serial links. The availability of these resources should further narrow the field in terms of device selection. In many cases, the list of resources will lead you to subfamilies of devices, for example, the Xilinx® Spartan®-6 LX or Spartan-6 LXT if logic or high-speed serial links are required.

The number of inputs and outputs required from the device will drive both the device selection and the packaging selection, as it is possible to get the same device in several different packages, each providing a different number of user I/Os. Here, it's often worth your while to consider an upgrade path and choose a footprint that is common across a number of devices within the selected family.

You must also consider the operating environment when selecting a device. For example, is a commercial component sufficient, or do you need to consider an industrial, medical or automotive part? In some cases, your system may require a defense- or space-grade component.

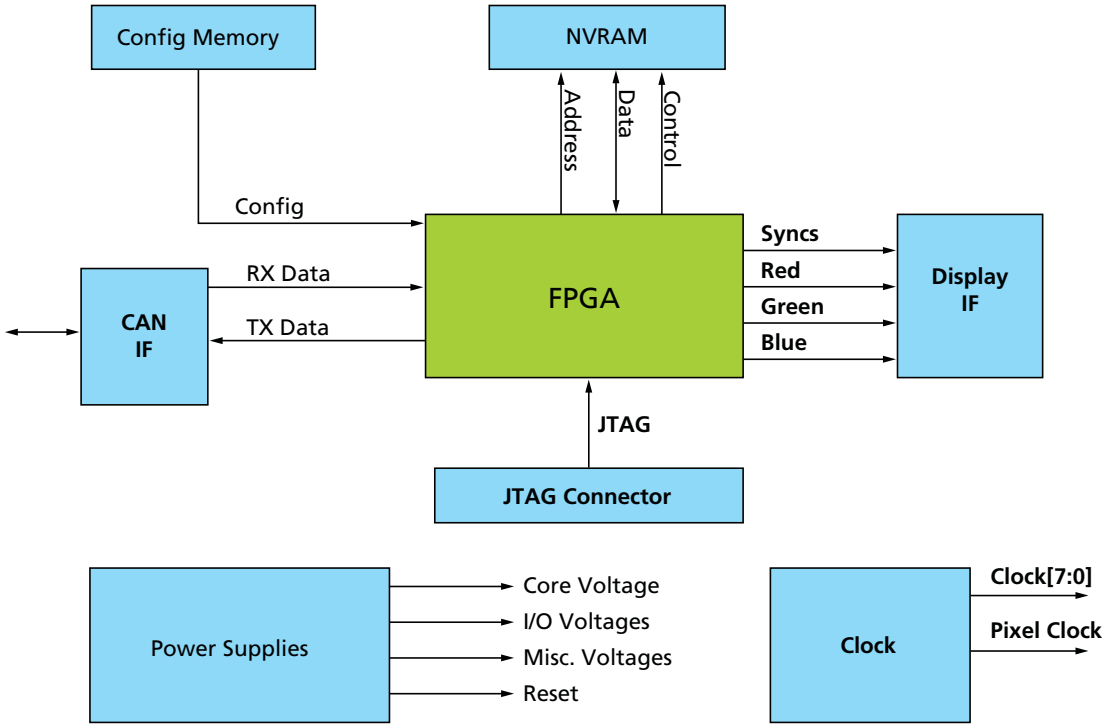


Figure 1 – A typical overall system architecture

Then too, you must also take the configuration architecture into account. Are SRAM-based FPGAs acceptable for the application or is a nonvolatile solution like the Xilinx Spartan-3AN series a better fit? Security of the design should also factor into your deliberations. If your application needs it, you should consider preventing readback and encrypting the data stream.

As always, the cost of the component will also be a driver, regardless of whether the application is to be produced in volume or as a one-off, bespoke spaceflight design. You should have a target cost budget and strive to ensure you achieve it.

A number of the above parameters are also important when considering other devices for use within the system. In particular, one important criterion is selecting components that operate off the same voltage rails, thus simplifying the power architecture.

You should also attempt where possible to derate components within the design. All manufacturers' component data sheets provide absolute-maximum and operating electrical stresses for the device in question. If we designed a system such that a device was operating with an electrical stress just below the absolute maximum, the reliability of the design would plummet because the system would be functioning outside the recommended operating conditions. Depending upon the end application, this failure could result in anything from the loss of life/mission to your company's gaining a bad reputation for quality. To produce reliable equipment, you must reduce the electrical stress upon the design. You should therefore consider the electrical stresses that devices will be subjected to when selecting one for your application.

POWER ARCHITECTURE

After the selection of the device, the power architecture is the next most critical area for a successful project, and one that is often overlooked. The FPGA will have a core voltage that will vary from 0.9 to 1.5 V for modern FPGAs. Often these devices will require a quiescent current that can be pretty large for a high-performance FPGA, along with the dynamic current once the device is configured and clocking.

Power estimation of the design is thus very important and is required early in the project to correctly size the power architecture. Xilinx provides power estimation spreadsheets, which you can download from *http://www.origin.xilinx.com/products/design_tools/logic_design/xpe.htm*. Within these spreadsheets, you can select the FPGA resources' clock rates and toggle rates along with environmental parameters for the chosen device. If you are not sure, err on the side of caution—be pessimistic rather than optimistic in your estimates. Once you have determined the power requirements for the FPGA, you can merge them into a larger, overall power budget for the system and use it to determine the power architecture.

In systems that require large currents, it is advisable to utilize switching DC/DC converters to maintain overall efficiency and ensure that the thermal design of the unit is not complicated. For lower-current requirements and

systems that must be particularly noise free—namely, those supplying power rails for high-speed serial links or sensitive ADC and DAC components—you may need to implement linear voltage regulators and additional filtering. In all cases, you should thoroughly read the data sheet of the device in question.

Once you have completed the FPGA design, you can obtain a more-detailed power estimation by using the XPower Analyzer within the Xilinx ISE® design suite. This step should help you close the loop in terms of the power architecture.

In addition, you should also consider the supply decoupling of devices on the board. Modern devices are capable of switching many times faster than the power supplies regulating the voltage. Without decoupling capacitors, this situation could lead to a dip in the power rail while the regulator attempts to keep up. Ideally, select the decoupling capacitors' values such that the impedance profile they present when combined with inter-PCB plane capacitance is below 0.1 ohm, between 100 kHz and 1 GHz, if possible. To achieve this performance, you will need to use a range of decoupling-capacitor values, each with a different self-resonant frequency. This technique enables devices with a poor power supply rejection ratio (PSRR) to achieve the best possible performance, as PSRR tends to decrease with operating frequency on some devices.

Finally, try to minimize the number of additional voltage rails. Doing so will reduce the complexity of the solution. You also need to be aware of any power rail sequencing or ramp rates, and ensure that your solution meets them.

CLOCKS AND RESET TREE

Your system will require at least one clock to operate. It is normal to use a logic-level oscillator at the desired frequency. When considering an oscillator, there are a number of factors to take into account. The desired output frequency and stability of the oscillator are key parameters. Oscillator stability is usually given in parts per million; typical PPMs are +/-50 PPM or +/-100 PPM.

A low phase noise and jitter are also often required for oscillators driving high-speed serial links or providing clocks to ADCs or DACs. Keeping noise and jitter low is important, as low noise reduces the bit-error rate of the high-speed link, while high noise and jitter will increase the noise floor on the ADC and DAC, reducing the signal-to-noise ratio.

You should also consider the output signal standard. Differential signaling such as LVDS or LVPECL provides for a better noise immunity than single-ended LVCMOS or LVTTL outputs. Differential signaling also lessens EMI concerns and delivers faster rise and fall times. However, differential signaling can have a worse phase-noise performance than a single-ended output. High-performance systems will use sine wave oscillators as the primary clock source to reduce the effects of phase noise and jitter; this is especially true when clocking ADCs and DACs.

Since FPGAs can support several clock domains internally, it is therefore common these days for systems to have more than one clock domain. Often, designers will use a high-speed clock and a lower-rate clock if they cannot achieve the necessary division via internal clocking resources such as

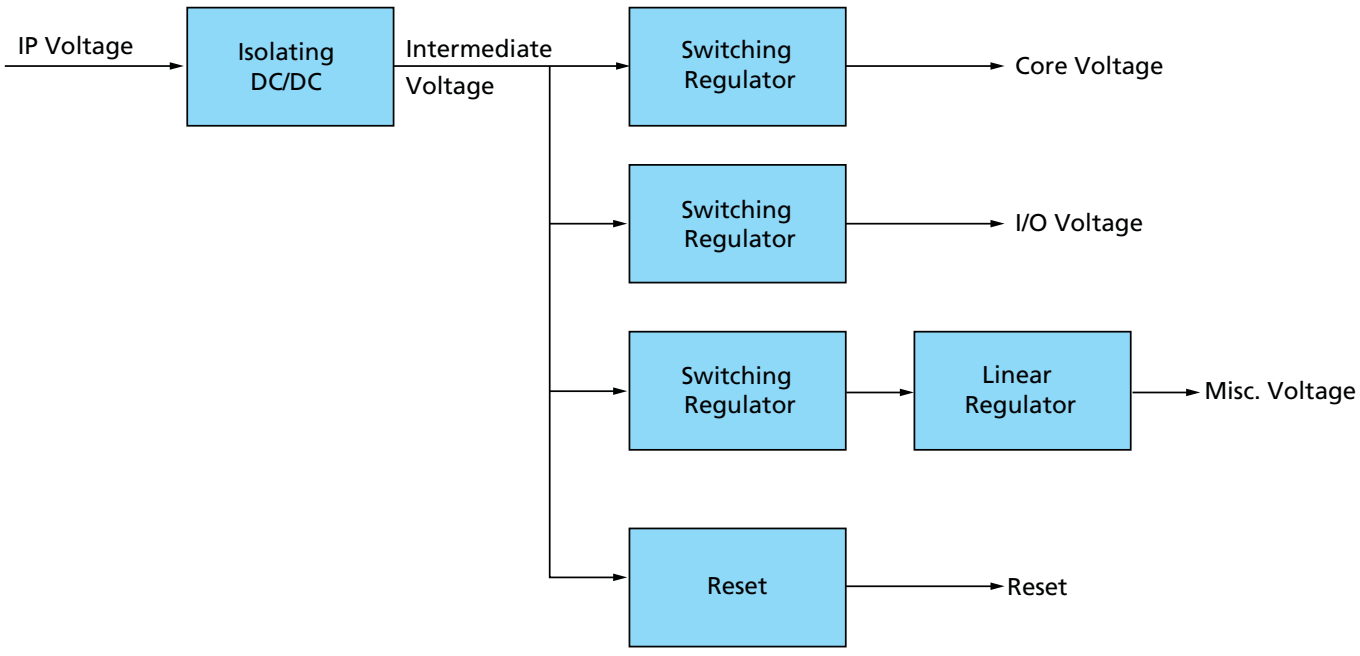


Figure 2 – Typical power architecture

Many FPGA designs will also require a reset that is connected to a number of devices to ensure they initialize properly. Resets can cause issues that can be tough to get a grip on, since they occur only randomly.

DLLs, DCMs or PLLs. Alternatively, varying protocols or algorithms could require different clock frequencies, creating multiple clocks and domains within the design. Figure 3 shows a typical clock tree.

Many FPGA designs will also require a reset that is connected to a number of devices to ensure they are initialized properly following the power-up of the system or when commanded, for example via a pushbutton. Resets can cause many issues in a design that can be tough to get a grip on, since they occur only randomly. Once the power rails have reached their operating range, the oscillators on the board will begin to function, though it will take time for them to become stable. It is therefore a good idea to ensure the reset on the board is asserted until the voltage rails have risen and the oscillator is stable.

Because this signal will be subjected to a potentially high fan-out within the FPGA, it's best to filter the signal heavily to ensure that glitches and EMI cannot result in accidental resets of the system.

You must also take care within the FPGA design to ensure that the reset is connected only where it is really required (remember, with SRAM, FPGA registers and RAMs can be initialized as part of the configuration). Also, take care within the FPGA to ensure that the reset removal cannot lead to a metastable event—that is, that its removal near a clock edge causes metastability. It is therefore important to correctly synchronize the reset to all clock domains.

Be sure to allocate the clocks and resets to the correct input style. Clocks can be allocated to global or local clock resources, and the hardware engineer will have to work closely with the FPGA engineers to ensure the correct global resources are used.

I/O STANDARDS AND SIGNAL INTEGRITY

FPGAs are capable of supporting a number of single-ended and differential I/O standards. Planning the I/Os of a device and ensuring that the banking rules are maintained can be a major challenge. This stage of the design requires close cooperation between the FPGA engineer and the hardware designer.

This is where the benefits of minimizing the voltage rails and interface types prove themselves. Once you have completed the schematic design or used a tool like Mentor Graphics' I/O Designer to successfully plan the device's I/Os, you can use tools such as Xilinx's PlanAhead™ to pull in the pin allocation and check for I/O banking rule and placement violations. This is especially helpful when using high-speed serial links, as you need to ensure that user I/O around the supply pins for these HSSLs is not used. Otherwise, inadvertent noise could couple onto the supply rail and affect the performance of the link.

You should also consider the signal integrity (remember, it is not the signal frequency but the edge rate that determines the length of trace before it will need termination). Most FPGAs support on-chip termination, reducing the need for discrete components. Of course, this approach works correctly only if you have defined the printed-circuit-board stack-up as controlled impedance.

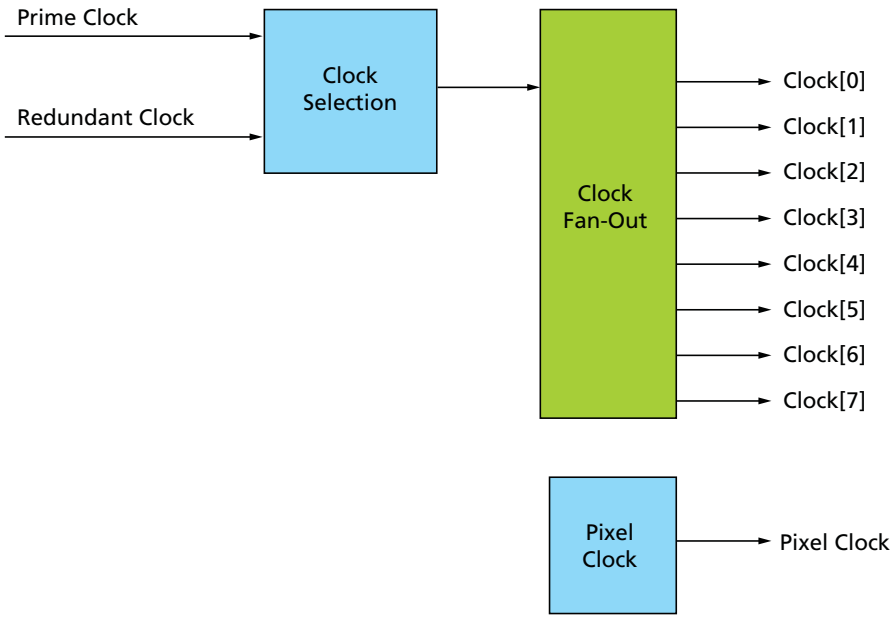


Figure 3 – Typical clock tree

MECHANICAL AND THERMAL CONSIDERATIONS

The hardware engineer will also have to work closely with the mechanical engineers involved in the project to determine the physical board outline, positions of connectors and the like. Modern FPGAs can get pretty hot, and require thermal considerations. This will impact the mechanical design, maybe requiring forced airflow (fans) or thermal straps and heat sinks to achieve the desired operating environment.

If the application is to be based in a harsh or rugged environment, the mechanical design will add more constraints to the PCB placement, as you will need to take shock and vibration levels into account. Depending upon the tool sets used, the mechanical and PCB engineers should be able to transfer information as DXF files between CAD systems to check that board outlines, keep-out zones and skylines do not clash.

PCB LAYOUT

The printed-circuit-board layout is one of the most critical areas in achieving the design performance required of the system. For any high-performance system, the board must be a controlled-impedance, multilayer PCB. Achieving this requires discussion between the PCB layout engineer and the PCB supplier to determine the stack-up of the board. The stack-up and corresponding trace widths and separations are critical to implementing a controlled-impedance board. Within most high-performance systems, you must calculate both microstrip and stripline correctly to provide a controlled impedance of 50 Ω for single-ended signals and 100 Ω for differential signals. This enables the hardware engineer to correctly terminate any transmission lines that occur using either on-chip or discrete termination.

It is also important within the stack-up to ensure that the voltage planes and the return planes are close to each other in order to allow capacitive coupling. This setup enables the substrate of the PCB between the planes to act like a decoupling capacitor, adding the decoupling performance. Figure 4 shows a typical impedance-control multilayer PCB stack-up.

You must also specify a series of constraints to the layout engineer at this stage as well. These will guide the placement of components, separating analog from digital. The list should also specify constraints on the grouping and length matching of buses and signals within the design. To aid with this task, you might find it helpful to also specify a time of flight for the signal.

In addition, don't forget to consider crosstalk between signals within

9 THINGS TO TAKE INTO ACCOUNT WHEN DESIGNING YOUR HARDWARE

1. The FPGA itself: The system requirements and functions will dictate the size, technology and speed grade, while other parameters (high-speed serial links, embedded processors, etc.) will dictate the FPGA choice.

2. Power architecture: In evaluating the power architecture, be sure to consider the numerous voltage rails needed, the efficiency of the overall solution and—particularly in more-sensitive applications—the noise present on these rails.

3. Configuration: Modern FPGAs require a large nonvolatile memory to store the configuration data. As these devices have several methods of configuration, you need to choose the one that's most optimal for your design.

4. JTAG port and chain: You will need these to configure both the FPGA and the configuration memory, as well as for boundary-scan testing of the hardware following manufacture to identify production issues.

5. Clocks and reset tree: Providing the different clocks required for the FPGA design and the reset tree will ensure the design is reset correctly.

6. Application-specific interfaces. Depending on your design, these might include communications links such as Ethernet, PCI Express™, CAN and RS422 or RS232, allowing the solution to talk to other components of the system.

7. Application-specific devices: Typically, most designs use discrete devices interfacing to the FPGA. Commonly used devices are A/D and D/A converters.

8. Application-specific memories. Use these devices to store large quantities of data during processing or application-configuration data, such as image overlays for an imaging system. Examples include high-capacity memories (double-data-rate or quad-data-rate SRAMs) and nonvolatile memory—such as flash or FRAM devices—for storing data that must be retained when the system loses power.

9. Operating environment. Attention to this parameter will ensure you pick the correct grade of components.

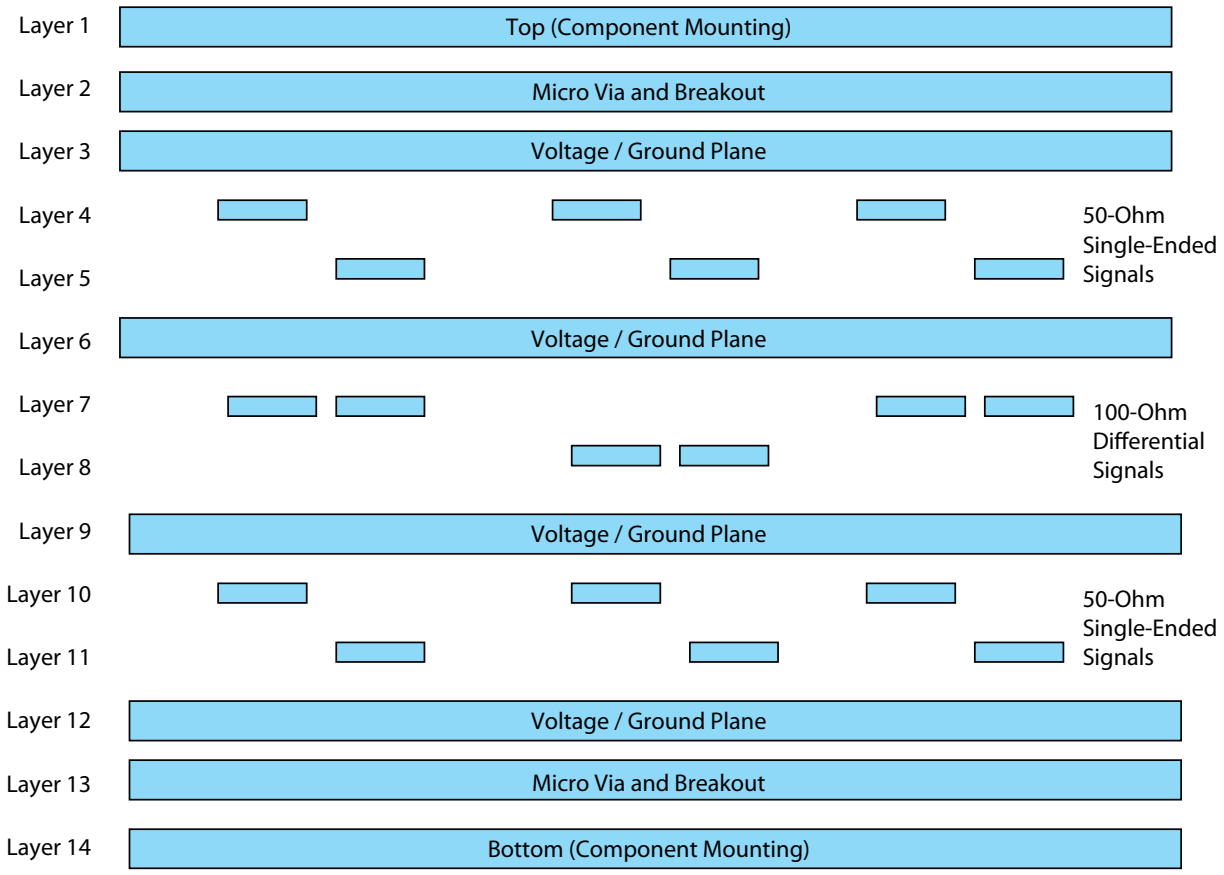


Figure 4 – Typical 14-layer impedance-controlled PCB stack-up

the PCB. It's important to separate signal traces correctly so as to reduce signals on the same layer and ensure that signals on the layers above or below within stripline structures do not have long parallel runs, as this too will be a source of crosstalk. It is a good idea if possible to declare impedance-controlled layers between planes as vertical or horizontal routing to ensure that tracks on different layers do not have long parallel runs. Increasing the space between two traces on a PCB will reduce the crosstalk; a good rule of thumb is to keep traces separated by at least three track widths. However, for more-critical signals, increase the distance to five or even seven track widths.

Be sure to keep analog and digital signals as far apart as possible, and whenever you can, use separate analog returns and power supplies. Neither digital nor analog signals

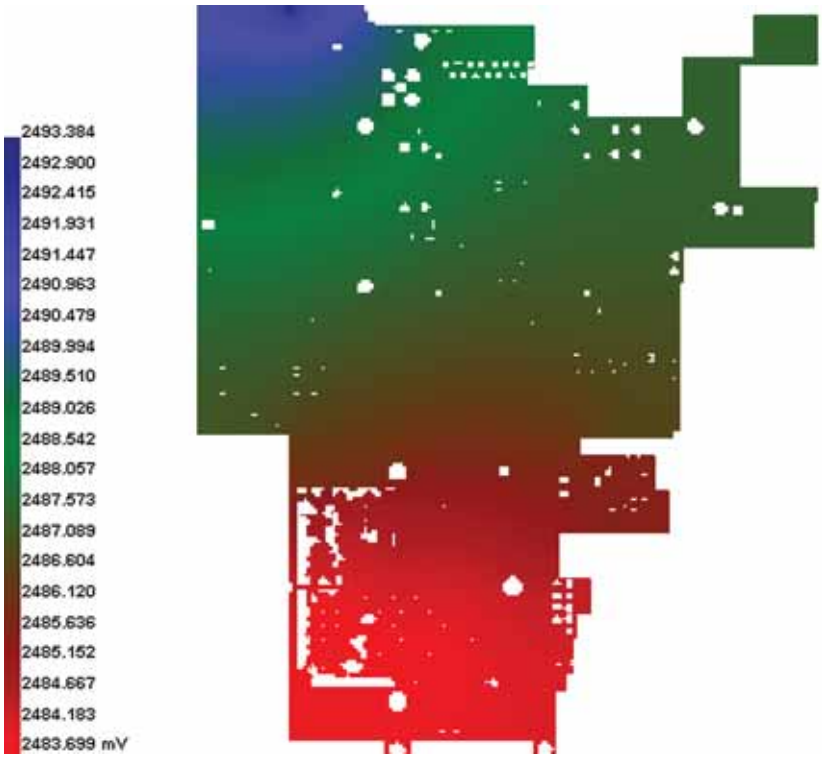


Figure 5 -- Results of a DC power drop analysis for a space-grade Virtex-5 device

One of the most important stages toward the end of the layout, once the planes have been entered, is the validation of both the signal integrity of the completed design and the power integrity, using tools like Mentor Graphics' HyperLynx SI and PI. This validation my be an iterative process.

should be routed referencing the other's plane, as it complicates the return-current path and will degrade the system performance.

One of the most important stages toward the end of the layout, once the planes have been entered, is the validation of both the signal integrity of the completed design and the power integrity, using tools like Mentor Graphics' HyperLynx SI and PI. This validation my be an iterative process—more so for the power integrity, as the high currents the FPGA core requires may necessitate adjusting more planes to achieve the required voltage performance of the planes. Figure 5 shows the results of a PI simulation for a large, high-performance FPGA.

TEST ISSUES

The engineer must also give some thought to how the board and its performance will be verified as achieving all requirements. You can use the JTAG controller and boundary-scan testing to test the infrastructure and interconnectivity of most digital devices and drivers. However, you must also consider the performance of power, clock and reset circuits. Test headers or points connected to voltage rail outputs allow quick and easy checking of the power architecture. (Make sure you use a series resistor to limit the current should the test point accidentally become shorted to something.) You can use very low-value resistors (in the milliohm range) to determine the currents downstream by the FPGA and supporting devices. Similarly, if your design uses clock buffers and there is a spare output, it is wise to correctly terminate this output to allow for testing of the clocks.

Debug LEDs to show the status of the FPGA done line, and others that the application will use, can be of great assistance in terms of debugging as the board is commissioned.

It may also be necessary to add logic-analyzer or pattern-generator headers to the board. Make sure these are easily accessible not only at open-layer testing but also at system testing, when the unit is within its enclosure.

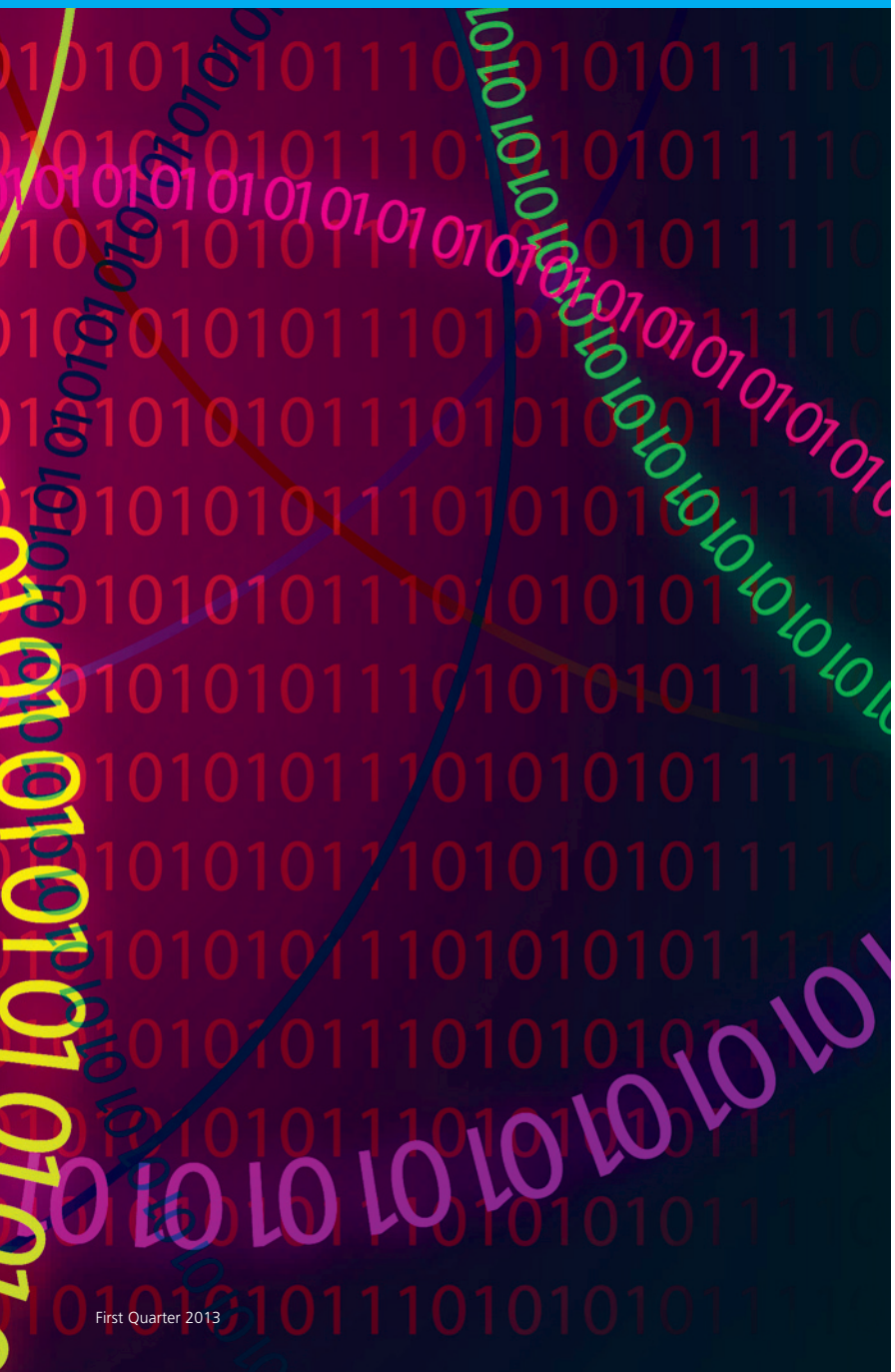
Tools like Xilinx's ChipScope™ will help you debug the FPGA functionality and interfaces, and can determine the bit-error rate on high-speed serial links.

Designing FPGAs into the hardware of the system can present a number of interesting challenges for an engineer. A number of tools are available to assist you in this task, while numerous data sheets and user guides can also provide guidance. But the best resource is a solid understanding of the basic hardware elements of your system.

Using the Parallel FFT for Multigigahertz FPGA Signal Processing

by Chris Eddington
Senior Technical Marketing Manager
Synopsys Inc.
chrised@synopsys.com

Baijayanta Ray
Corporate Application Engineer
for Synphony Model Compiler
Synopsys Inc.
baijayan@synopsys.com



Very high-speed fast Fourier transform (FFT) cores are an essential requirement for any real-time spectral-monitoring system. As the demand for monitoring bandwidth grows in pace with the proliferation of wireless devices in different parts of the spectrum, these systems must convert time domain to spectrum ever more rapidly, necessitating faster FFT operations. Indeed, in most modern monitoring systems, it is often necessary to use parallel FFTs to run at sample throughputs of multiple times the pace of the highest clock rate achievable in state-of-the-art FPGAs like the Xilinx® Virtex®-7, taking advantage of wideband A/D converters that can easily attain sample rates of 12.5 Gigasamples/second and more. [1]

At the same time, as communications protocols become increasingly packetized, the duty cycles of signals that need to be monitored are decreasing. This phenomenon requires a dramatic decrease in scan repeat time, which necessitates low-latency FFT cores. Parallel FFTs can help in this regard as well, since the latency scales down almost proportionally to the ratio of sample rate to clock speed.

For all of these reasons, let's delve into the design of a parallel FFT (PFFT) with runtime-configurable transform length, taking note of the throughput and utilization numbers that are achievable when using parallel FFT.

HARDWARE PARALLELISM FOR FFTS

Due to the complexity of implementing FFTs directly in logic, many hardware designers use off-the-shelf FFT cores from various vendors. [2] However, most off-the-shelf FFT cores use "streaming" or "block" architectures that process only one or fewer samples per clock, which limits the throughput to the maximum clock speed achievable by the FPGA or ASIC device. A PFFT offers a faster alternative. A PFFT can accept multiple samples per clock and process them in parallel, to deliver multiple output samples per clock. This architecture multiplies the throughput beyond the achievable device clock speed, but comes at an additional cost in area and complexity. Thus, to use a PFFT you will have to make trade-offs in throughput vs. area. The trade-offs for a typical Virtex-7 FPGA design are outlined in Figure 1 and Table 1.

Typical performance and area trade-offs for parallel FFTs on Virtex-7 class devices

FFT Architecture Length 1024 16-bit	Number of Complex Input Samples	Max System Clock Achieved	FFT Throughput (Samples/s)	Hardware Multiplier Utilization	Latency (System Cycles)
Streaming	1	500 MHz	500 MS/s	32	1260
Parallel x2	2	500 MHz	1 GS/s	64	630
Parallel x4	4	490 MHz	1.968 GS/s	128	360
Parallel x8	8	490 MHz	3.92 GS/s	260	220
Parallel x16	16	440 MHz	7.088 GS/s	408	145

Table 1 – Area scalability is generalized by hardware multiplier utilization. Throughput scalability vs. area is slightly better than linear and generally very usable for increasing throughput to multigigahertz sample rates.

Looking at the table, a few general features can be seen in the trade-off curve:

1. As parallel throughput increases, multiplier (area) utilization increases, with a slightly lower multiple (better than linear).
2. Slower system clocks and timing closure yield sublinear throughput growth as parallelism increases. However, on modern FPGAs this degradation is diminishing.
3. Overall better-than-linear throughput/area growth is realized due to No. 1 and No. 2 above.
4. Latency decreases as parallelism increases.

Note that the specific numbers measured in Table 1 are valid only for a given target and configuration of the FFT. In this case, that is a length of 1024, with 16-bit input, dynamic length programmability (4 through 1024) and flow control. Flow control is very important for applications such as spectral monitoring, where side-channel information is often utilized to change the FFT size (in order to change the resolution bandwidth) or to temporarily stall the FFT while other operations, such as acquisition, are going on. In theory, you can accomplish flow control by inserting buffers before the transform operation. But for acquisition-driven operations like spectral monitoring, it's not easy to precompute the size of the buffer required, resulting in the need to maintain large, fast and expensive memory banks.

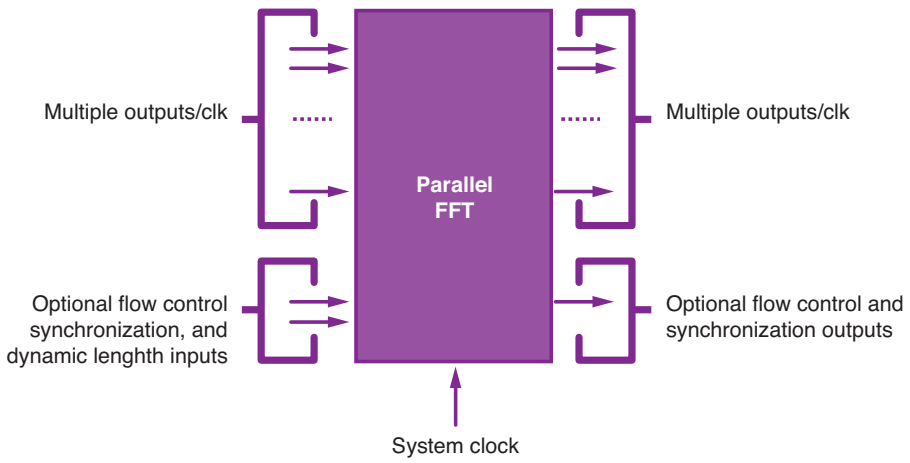


Figure 1 – A parallel FFT processes multiple samples at a time to scale throughput beyond achievable system clocks of the target device. Optional features include flow control, synchronization and dynamic length programmability.

IMPLEMENTATION ARCHITECTURE

While there are a number of ways to implement FFTs, a parallelized version of the Radix2 Multi-Path Delay Commutator kernel (Radix2-MDC) [3] works very well as a modular method to create configurable parallel-FFT cores that scale well in advanced FPGA devices. The Radix2-MDC is a classical approach to building pipelined FFTs of varying lengths, as shown in Figure 2a for a 16-length FFT. It breaks the input sequence into two parallel data streams flowing forward with the correct "distance" between data elements that are entering the butterfly (a subelement of FFT algorithms) and that are scheduled by proper delays. The Radix2-MDC is relatively easy to parallelize using a wider data path and vector operations, as shown in Figure 2b. MDC structures also lend themselves easily to flow control and dynamic length reconfigu-

ration, as opposed to single-path delay feedback (SDF) structures, where the incorporation of flow control (stall) signals typically reduces maximum throughput considerably.

Another choice that can affect scalability is the complex-multiplier implementation—that is, either 4multiply (4M) or 3multiply (3M) structures. Choosing a 3M complex multiply often leads to lower area usage in your design, but at the expense of slower clock speeds. [4] This trade-off is also very dependent on the DSP hardware of the FPGA device. Below are the most important parameters and the choices we made in the case study that we are about to present:

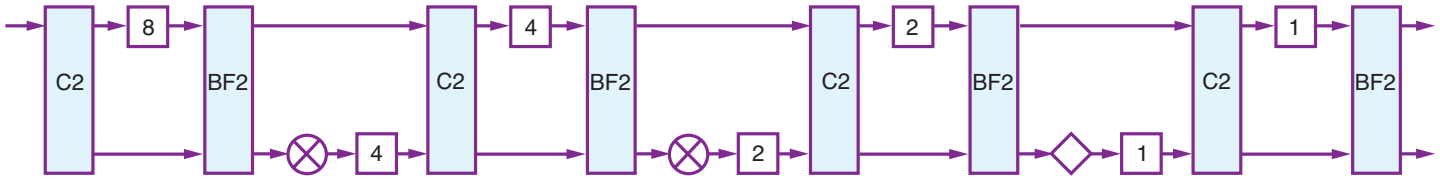
- Length = 1024
- Input precision = 16 bits
- Radix2-MDC architecture using 4Mult-5add complex multipliers
- Data path precision = 1-bit growth per stage (10 stages / bits for L=1024)
- Dynamic length programmability included
- Optional flow control and synchronization turned on

CASE STUDY: A 1024-LENGTH PARALLEL FFT

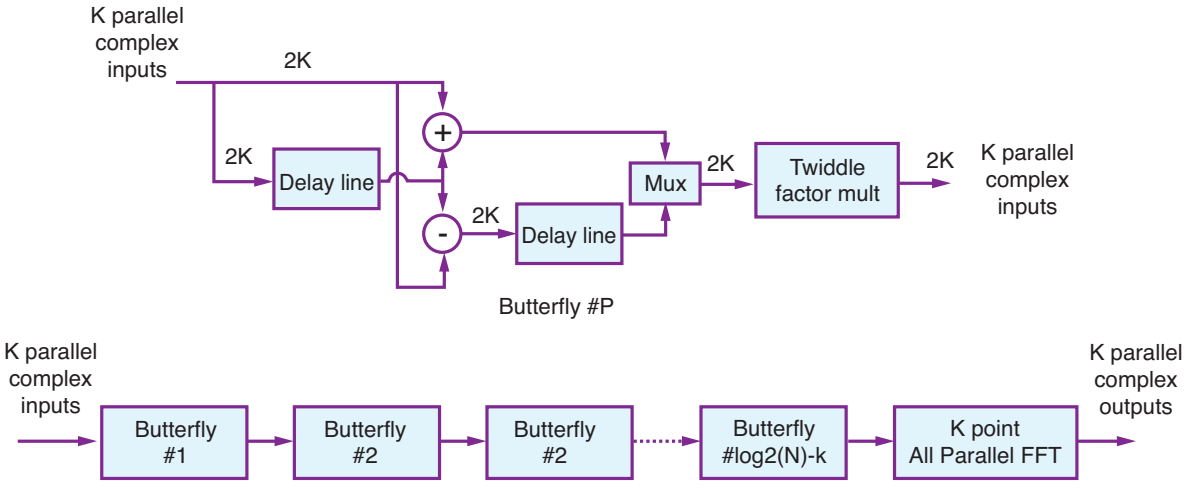
For our case study, we used Synopsys' Synphony Model Compiler (Synphony MC or SMC) high-level synthesis tool [5] to explore a set of parallel-FFT results for Xilinx Virtex-7 and Virtex-5 FPGAs.

Synphony MC has a parallel-FFT IP block that is designed specifically to achieve high-quality-of-results (QoR) DSP mapping for advanced FPGA devices like the Virtex-7. This custom IP block instantiates arithmetic primitives in the subsystem underneath to create an architecture based on user-specified options such as length, precision, flow control and dynamic programmability.

Using the vector support and a custom block methodology in Synphony MC, we were able to create a concise, parameterizable parallel Radix2-MDC block that enabled quick instantiation of parallel FFTs across different configurations. Figure 3 shows a subset of a 1024x16 PFFT data path using these custom blocks (called PR2MDC). Each block inputs and outputs a length-32 vector representing 16 complex samples of real and imaginary values, and a length and twiddle-factor address parameter. The fixed-point word length grows 1 bit in each stage, but is parameterized for easy adjustment. Underneath this block is the implementation of Figure 2b using Synphony MC arithmetic operators (multiply, add, shift registers, etc.).



a) Radix2-MDC for 16x1 streaming FFT



b) Parallel Radix2-MDC for NxK parallel FFT

Figure 2 – The Radix2-MDC kernel (a, at top) can effectively be parallelized and used in a modular way to create parallel-FFT implementations (b).

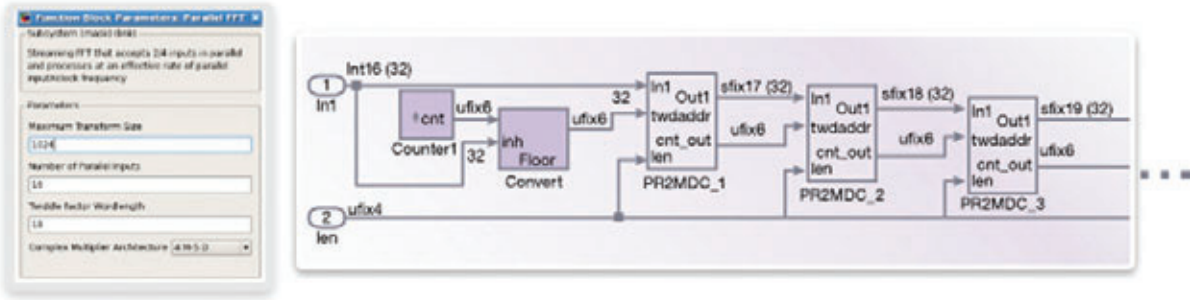


Figure 3 – The parallel-FFT IP block user interface (left) delivers a microarchitecture based on user-specified configuration.

Although SMC is a high-level design environment, you can use it to achieve specific mapping for a variety of target technologies. In this case, we chose a 4M complex multiply that maps into one DSP48E unit using 18x25 multiply mode. The block is designed to daisy-chain the twiddle-factor lookup for performance and flow control capability (note that Figure 3 does not show flow control). The SMC parallel-FFT core does not show any significant timing degradation between turning flow control on or off, or using dynamic length programmability; the throughput scalability it offers on advanced FPGA devices is a useful feature when building real-time spectral-monitoring applications.

We used Synphony MC to generate RTL optimized for Virtex-5 and Virtex-7 targets, and then used the 2012.09 release of Synphony Model Compiler and Synplify Pro with Xilinx's ISE® 14.2 tool suite for place-and-route. Table 2 shows the post-place-and-route area and timing results. The observed scalability matches very well to theoretical expectations and the FPGA family's capabilities, achieving better than 7 Gsamples/s using an X16 parallel configuration with only approximately 6.3 times the resources of the 1X configuration. The better-than-linear area growth is due to multiply operations that become fixed-coefficient as parallelism increases, and thus get implemented more efficiently in logic than if you were using a full multiply in a DSP unit. This happens mostly in the last K-point FFT (see Figure 2b).

HIGHER THROUGHPUT

Parallel FFTs are required for high-throughput gigasample applications such as spectral monitoring. Our PFFT hardware implementation offers some

Parallel FFT[1]	Max System Clock Achieved (Virtex-7)	FFT Throughput[2] (Samples/s) (Virtex-7)	DSP48E Utilization[2] (Virtex-7)	BRAM Utilization[2] (Virtex-7)	Latency (System Cycles)
Parallel x2	500 MHz	1 GS/s	64	8	630
Parallel x4	492 MHz	1.968 GS/s	128	11	360
Parallel x8	490 MHz	3.92 GS/s	260	16	220
Parallel x16	443 MHz	7.088 GS/s	408	8	145
Virtex-5	**Virtex-5**	**Virtex-5**	**Virtex-5**	**Virtex-5**	**Virtex-5**
Parallel x2	349 MHz	0.698 GS/s	64	8	630
Parallel x4	312 MHz	1.248 GS/s	128	11	360
Parallel x8	280 MHz	2.24 GS/s	260	16	220
Parallel x16	251 MHz	4.016 GS/s	408	8	145

1. Using Synphony Model Compiler, parallel Radix2-MDC architecture and configuration.
2. Implementation flow with Synphony Model Compiler 2012.09., Synplify Pro 2012.09 and ISE 14.2

Table 2 – The PFFT block's portability is illustrated by the excellent results achieved on both the Virtex-7 485T: -3 and Virtex-5 SX95T: -2.

insight and guidance on the throughput scalability for advanced Xilinx FPGA devices. A case study demonstrates that the parallel FFTs using the parallel Radix2-MDC architecture can effectively achieve throughput of more than 7 GHz on FPGA devices in an X16 parallel configuration. We developed the PFFT block in a high-level design flow using Synphony Model Compiler, with the ability to fine-tune the implementation for DSP hardware mapping across different Xilinx devices. [5] Synopsys is making this block freely available to all Synphony Model Compiler users.

For more information on Synphony Model Compiler, please visit *http://www.synopsys.com/cgi-bin/sld/pdfdla/pdfr1.cgi?file=synphony_model_comp_ds.pdf?cmp=FPGA-SMC-xcell.*

References

1. Tektronix Component Solutions Web Page, http://component-solutions.tek.com/services/data-converter/

2. Xilinx FFT Core Datasheet, http://www.xilinx.com/support/documentation/ip_documentation/ds808_xfft.pdf

3. Shousheng He; Torkelson, M.; "A new approach to pipeline FFT processor," Parallel Processing Symposium, 1996, Proceedings of IPPS '96, The 10th International, April 1996

4. Xilinx Complex Multiplier v5.0 Block Datasheet, http://www.xilinx.com/support/documentation/ip_documentation/cmpy/v5_0/ds793_cmpy.pdf

5. Synphony Model Compiler Datasheet, http://www.synopsys.com/cgi-bin/sld/pdfdla/pdfr1.cgi?file=synphony

OmniTek, Xilinx Roll Zynq-7000 Video Development Kit

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

The new OZ745 platform eases the design of real-time professional broadcast and other apps.

Xilinx and Xilinx Alliance member OmniTek this month will make available a real-time video-processing platform targeted at companies creating high-bandwidth equipment for a wide range of motion-picture industries.

The new platform, called the OmniTek OZ745 Zynq™-7000 SoC Video Development Kit, includes all the key components to help OEMs improve system performance, reduce BOM costs and get to market quickly with new and innovative monitors, studio cameras, production switchers and an expanding list of other applications in the medical, automotive and aerospace industries that require leading-edge video technologies.

The platform—which includes hardware, design tools, IP, an operating system and preverified reference designs—is built around the Xilinx® Zynq-7045 All Programmable SoC. This device integrates high-speed transceivers capable of supporting uncompressed HD video (for example, SD/HD 3G-SDI). "With this platform, the broadcast and professional A/V industry has, for the first time, a completely programmable device that bridges software and hardware domains," said Roger Fawcett, managing director at OmniTek.

Fawcett said that the Zynq-7000 SoC enables system architects to implement video- and audio-processing algorithms in two ways: either in software on the dual-core ARM processors onboard, or offloaded into FPGA hardware if the application needs acceleration to meet real-time demands. An FPGA solution is particularly useful for multichannel HD, 4K, faster frame rates and other sophisticated video demands. Further, the tight coupling between software processor and FPGA hardware makes it possible to develop codec algorithms in both domains.

"It is also worth noting that each ARM® Cortex™-A9 processor includes a NEON single-/double-precision floating-point unit. Added to the DSP-rich FPGA fabric, this FPU gives the DSP designer a highly flexible platform on which to design all kinds of signal-processing algorithms," said Fawcett. "Additionally, the ability to stream and process uncompressed video from transceivers through the FPGA hardware, while monitoring the video stream in software as well as implementing overall system control, means that the innovation possibilities are endless." Then too, he said, a Zynq-based solution "also means saving power and cost through integration of multiple ASSPs."

The OmniTek board (see Figure 1) extends the capabilities of the Zynq device further by offering support for a wide range of signal types through five SD/HD 3G-SDI input/outputs; an

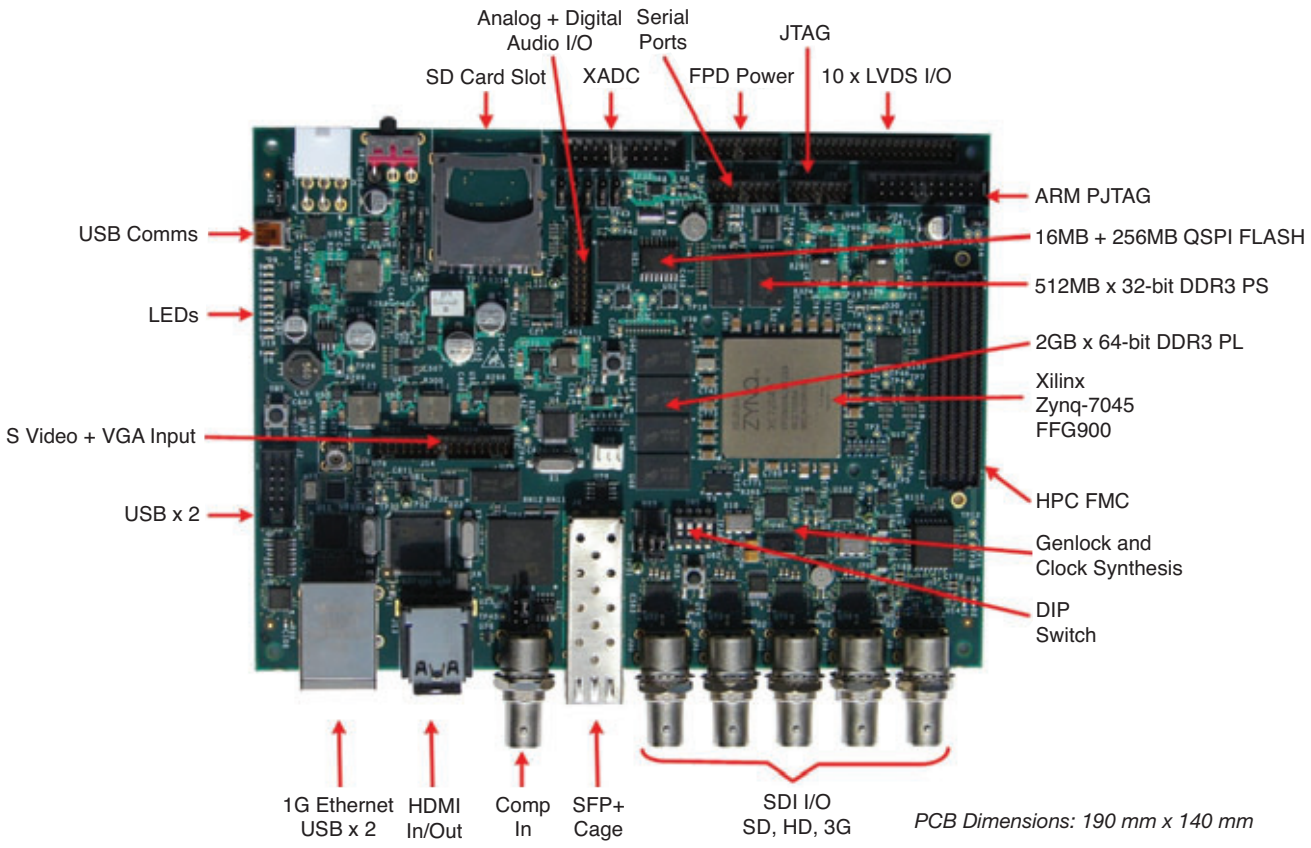


Figure 1 – The board for OmniTek's OZ745 Zynq-based Video Development Kit

A second reference design implements a range of video IP blocks within the FPGA fabric of the Zynq-7045 SoC, delivering a working FPGA design to use in evaluating image quality.

HDMI input/output pair; and composite, VGA and S-Video inputs.

The board also provides analog and digital audio I/O; up to four USB ports; a serial port; a 1-Gbps Ethernet port; and 10 bidirectional LVDS ports to use, for example, with an LCD flat-panel display. Also included are LCD flat-panel power, an HPC FMC expansion connector and an SD Card slot, along with FPGA and ARM JTAG debug ports. There is also an SFP+ cage, which designers can use, for example, to add 10-Gbps Ethernet capabilities for supporting new and emerging standards for video-over-Internet Protocol networks.

TWO REFERENCE DESIGNS

"Just as important as the hardware is the board support package that is provided alongside the OZ745 board," said Fawcett. In addition to a Linux build with Qt graphics and board built-in self-test (BIST), "this package also offers two reference designs for use as the basis for the user's own designs," he said. The first reference design provides design teams with the firmware, drivers and other elements they need to enable the ARM processor to access and control all the I/O and peripherals on the board. A simple control application allows designers to exercise the I/O on the board as well as peek at and poke the I/O and the onboard SDRAM.

Meanwhile, the second reference design is a version of the latest Xilinx multichannel Real-Time Video Engine (RTVE 2.1). This reference design implements a range of video IP blocks within the FPGA fabric of the Zynq-7045 SoC to deliver a working FPGA design that design teams can use to evaluate image quality (see Figure 2).

READY-MADE VIDEO PIPELINE

Fawcett said that a key element of RTVE 2.1 is OmniTek's new OSVP IP block, which provides all the firmware needed to handle multiformat conversion and compositing in a single IP block. Among the functions the OSVP block handles are video deinterlacing and resizing of up to 10 SDI channels; chroma resampling; and multichannel compositing of video inputs onto as many as 10 video outputs, facilitating picture-in-picture, quad-split display and other effects. The IP block also supports video graphics overlay; 3:2

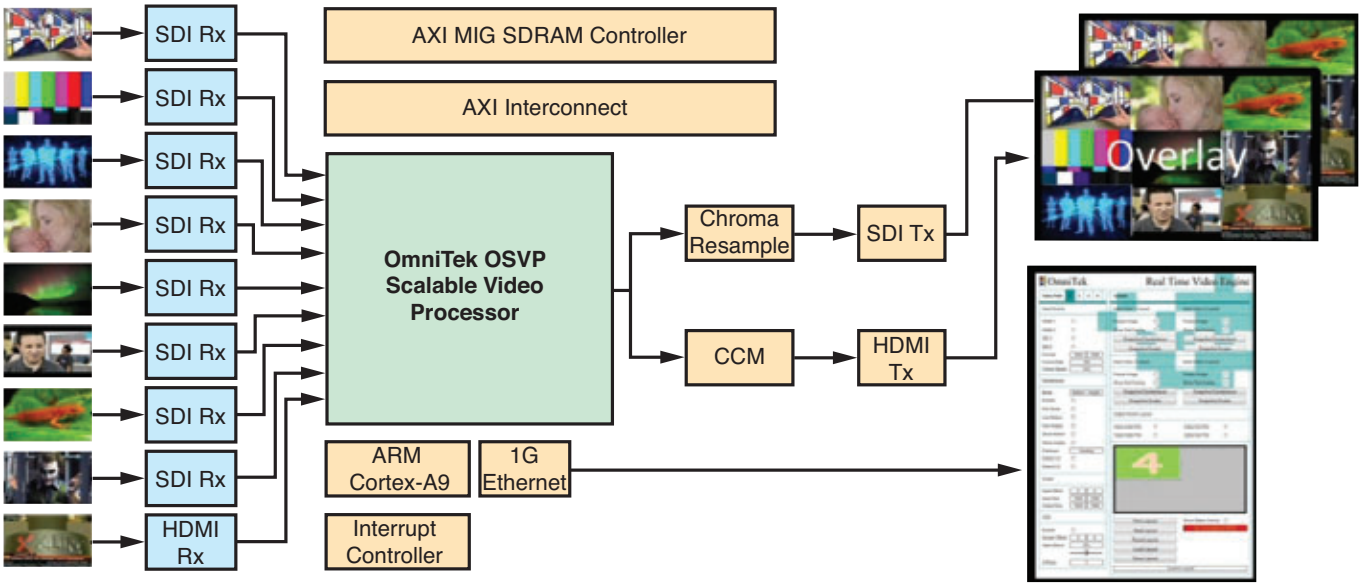


Figure 2 – Outline structure of the RTVE 2.1 reference design

and 2:2 film-cadence detection and processing; detail enhancement; and frame synchronization, even across changes in video standards.

Fawcett said the company specifically optimized the OSVP block for Xilinx FPGA technology. "All the interfaces to the block conform to the AXI4 protocol, and Xilinx CORE Generator™ system and EDK integration are also supported, making the block easy to interface to other IP intended for implementation in Xilinx technology," he said.

Applications include flat-panel display controllers, video-format converters, multiviewer displays and video pattern generators. The platform supports all SD, HD and 3-Gbit/second SDI formats, and can also interface with flat-panel displays with resolutions up to 4K and beyond.

RANGE OF IP BLOCKS

In addition to providing development platforms like the new Zynq board, OmniTek also supplies an expanding range of video-processing-related IP blocks, many of which are available optimized for Xilinx technology. The functions covered include video deinterlace and resize; SDI audio embed and extract; video and audio monitoring; video and audio test-pattern generation; audio codecs; 608, 708, OP-47 and PAL closed-caption and teletext decode; SDI eye pattern and jitter monitoring; and digital effects.

OmniTek also operates an IP and design consultancy service that specializes in FPGA, electronics and software design for video- and image-processing applications in the broadcast, medical-imaging, industrial and defense markets.

The OmniTek OZ745 Video Development Kit featuring Xilinx's Zynq-7000 All Programmable SoC is available now. For more information, please visit the Xilinx Broadcast page (*http://www.xilinx.com/applications/broadcast/index.htm*) or the OmniTek website, at *http://www.omnitek.tv*.

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.

XAPP744: HARDWARE-IN-THE-LOOP (HIL) SIMULATION FOR THE ZYNQ-7000 ALL PROGRAMMABLE SOC

http://www.xilinx.com/support/documentation/application_notes/xapp744-HIL-Zynq-7000.pdf

The fact that the Xilinx® Zynq™-7000 All Programmable SoC does not deliver a simulation model poses a problem for designers. This application note describes a way to bring the MPCore processor subsystem into the ISE® Design Suite Simulator (ISim) simulation environment with the help of Zynq-7000 platform hardware-in-the-loop (HIL) simulation technology. Author Umang Parekh explains how to bring the processing system in a Zynq-7000 AP SoC design into the simulation and guides the user in setting up a system environment for HIL simulation.

The Zynq-7000 All Programmable SoC is a new class of product from Xilinx that combines an industry-standard ARM® dual-core Cortex™-A9 MPCore processor subsystem (PS) with Xilinx 28-nanometer programmable logic (PL). Traditionally, Xilinx has offered the MicroBlaze™ embedded processor. Designers created peripheral IP in the FPGA logic in RTL and simulated the entire system using an RTL simulator for which Xilinx provided the MicroBlaze simulation RTL model.

HIL technology simulates, debugs and tests both the PS and the PL portions of a Zynq-7000 SoC design. All of the AXI-based IP connected to the PS through the master/slave general-purpose, high-performance or Accelerator Coherency Port (ACP) interfaces can be simulated in ISim, while the PS is simulated in the hardware (the ZC702 board). Clocking the AXI-based PL interfaces using the testbench clock allows cycle-accurate simulation of IP in the PL. The PS and the DDR memory operate in free-running mode.

This approach is very useful for developing IP with the processor system, and also for IP driver development and software debugging. Software support for the Zynq-7000 HIL is available in the 14.2 version of the Xilinx ISE design tools. HIL simulation is much faster than RTL simulation, ensuring quick design turns for both hardware and software.

XAPP892: IMPLEMENTING SMPTE SDI INTERFACES WITH VIRTEX-7 GTX TRANSCEIVERS

http://www.xilinx.com/support/documentation/application_notes/xapp892-smpte-sdi-if-v7-gtx-transceivers.pdf

The Society of Motion Picture and Television Engineers' (SMPTE) serial digital interface family of standards forms the foundation for much professional broadcast video equipment. Broadcast studios and video production centers use these SDI interfaces to carry uncompressed digital video, along with embedded ancillary data such as multiple audio channels. The Xilinx SMPTE SD/HD/3G-SDI LogiCORE™ IP is a generic SDI receive/transmit datapath that does not have any device-specific control functions. This application note provides a module containing control logic that couples the SMPTE SDI LogiCORE IP with the Virtex®-7 GTX transceivers to form a complete SDI interface. Author John Snow also provides several example SD/HD/3G-SDI designs that run on the Xilinx Virtex-7 FPGA VC707 evaluation board.

Designers can connect the Xilinx SDI core to a Virtex-7 GTX transceiver to implement an SDI interface capable of supporting the SMPTE SD-SDI, HD-SDI and 3G-SDI standards. Some additional logic is necessary to connect the SDI core and GTX transceiver together to implement a fully functional SDI interface. The application note describes this extra control and interface logic, and provides the necessary control and interface modules in both Verilog and VHDL source code.

Also included is a wrapper file that contains an instance of the control module for the GTX transceiver and an instance of the SMPTE SDI core with the necessary connections between them. This wrapper file simplifies the process of creating an SDI interface. Two SDI demonstration applications provide detailed examples of SDI implementations in a Virtex-7 FPGA design.

XAPP794: 1080P60 CAMERA IMAGE PROCESSING REFERENCE DESIGN

http://www.xilinx.com/support/documentation/application_notes/xapp794-1080p60-camera.pdf

The Xilinx Zynq-7000 All Programmable SoC Video and Imaging Kit (ZVIK) builds on the ZC702 evaluation kit by adding hardware, software and IP components for the development of custom video applications. The included video reference designs, WUXGA color image sensor and video I/O FPGA mezzanine card (FMC) with HDMI input and output enable users to immediately start development of video system software, firmware and hardware designs.

This application note by Mario Bergeron (Avnet, Inc.) and Steve Elzinga, Gabor Szedo, Greg Jewett and Tom Hill of Xilinx describes how to set up and run the 1080p60 camera image-processing reference design using the ZVIK. The authors also provide instructions on how to build the hardware and software components and how to create the SD card boot image.

The VITA-2000 image sensor from ON Semiconductor, which is configured for 1080p60 resolution, generates the video input. An image-processing pipeline implemented using LogiCORE IP video cores converts the raw Bayer subsampled image to an RGB image; the IP cores remove defective pixels, de-mosaic and color-correct the image. A video frame buffer is implemented in the processing system's DDR3 memory, making images accessible to the ARM processor cores via the AXI Video Direct Memory Access (VDMA). The video frame buffer is not required for the operation of the image-processing pipeline, but is included in the design to enable the capture of input video images for analysis.

XAPP524: SERIAL LVDS HIGH-SPEED ADC INTERFACE

http://www.xilinx.com/support/documentation/application_notes/xapp524-serial-lvds-adc-interface.pdf

This application note by Marc Defossez describes a method of utilizing dedicated SelectIO™ deserializer components (ISERDESE2 primitives) in 7 series FPGAs to interface with analog-to-digital converters (ADCs) that have serial low-voltage, differential-signaling (LVDS) outputs. The associated reference design illustrates a basic LVDS interface connecting a Kintex™-7 FPGA to an ADC with high-speed, serial LVDS outputs. The high-speed ADCs used today have a resolution of 12, 14 or 16 bits, with possible multiple converters in a single package. Each converter in the package can function in standalone mode or can be combined in an interleaved mode to double or quadruple the conversion (sample) speed.

In both standalone and interleaved modes, designers can use one or two physical serial outputs as a connection to the interfacing device. One set of differential outputs is called a data lane. Using one data lane means that the converter is operating in one-wire mode, while a two-data-lane design works in two-wire mode. For every possible data output combination there is always one high-speed bit clock and one sample rate frame clock available. The one-wire mode is used in single- and double-data-rate (SDR and DDR) configurations, while two-wire mode is used only in DDR mode.

The FPGA's SelectIO technology deserializer components are configured as ISERDESE2 primitives. The application note uses two ISERDESE2s in SDR mode to capture a DDR signal. One ISERDESE2 is clocked at the rising edge and the second at the falling edge of the bit clock. This method allows capturing up to 16 bits, since each ISERDESE2 can capture 8 bits.

XAPP743: EYE SCAN WITH MICROBLAZE PROCESSOR MCS

http://www.xilinx.com/support/documentation/application_notes/xapp743-eye-scan-mb-mcs.pdf

This application note by Mike Jenkins and David Mahashin describes code that executes on an internal MicroBlaze processor, implementing the algorithm to measure a statistical eye (bit-error ratio vs. time and voltage offset) at the post-equalization, data-sampling point within the receiver of a 7 series FPGA GTX transceiver. Point-by-point measured data is stored in Block RAM to be burst-read by an external host PC. This software-centric approach for implementing a statistical eye measurement provides a flexible and extensible method for incorporating eye scan capability into existing designs.

As line rates and channel attenuation increase, receiver equalizers are more often enabled to overcome channel attenuation. This situation poses a challenge to system bring-up, because it's impossible to determine the quality of the line by measuring the far-end eye opening at the receiver pins. At high line rates, the received eye measured on the printed-circuit board can appear to be completely closed even though the internal eye, which follows the receiver equalizer, is open. The RX Eye Scan in GTH, GTX and GTP transceivers of 7 series FPGAs provides a mechanism to measure and visualize the receiver eye margin after the equalizer step. Additional use modes enable several other methods to determine and diagnose the effects of equalization settings.

XAPP792: DESIGNING HIGH-PERFORMANCE VIDEO SYSTEMS WITH THE ZYNQ-7000 ALL PROGRAMMABLE SOC

http://www.xilinx.com/support/documentation/application_notes/xapp792-high-performance-video-zynq.pdf

With high-end processing systems like the Xilinx Zynq-7000 All Programmable SoC, customers want to fully utilize the processing system (PS) and custom peripherals within the device. An example is multiple video pipelines in which live video streams are written into memory (input) and memory is read to send out live video streams (output) in which the processor is accessing memory. This application note by James Lucero and Ygal Arbel covers design principles that target customers needing high performance from the Zynq-7000 SoC's memory interfaces, from AXI master interfaces implemented in the programmable logic (PL) and from the ARM Cortex-A9 processors.

Guaranteed worst-case latency is required for the video streams in such designs, to ensure that no frames are dropped or corrupted. Providing high-speed AXI master interfaces in the PL with lower latency and direct access to the device's memory interfaces demands high-performance (HP) interfaces. The Zynq-7000 AP SoC contains four HP links that are 64-bit or 32-bit AXI3 slave interfaces designed for high throughput.

The design uses three AXI video direct-memory access (VDMA) engines to simultaneously move six streams (three transmit and three receive), each in 1,920 x 1,080p format, with a 60-Hz refresh rate and up to 32 data bits per pixel. Each VDMA is driven from a video test pattern generator (TPG) with a video timing controller (VTC) core to set up the necessary video timing signals. Data read by each AXI VDMA goes to a common on-screen display (OSD) core capable of multiplexing or overlaying multiple video streams to a single output video stream. The output of the OSD core drives the HDMI video display interface on the board.

Performance-monitor cores are added to capture performance data. The three AXI VDMAs are connected to three separate HP interfaces by means of the AXI interconnect and are controlled by the Cortex-A9 processor. This design uses 70 percent of the memory controller bandwidth. The reference system is targeted for the Zynq-7000 ZC702 evaluation board.

XAPP586: USING SPI FLASH WITH 7 SERIES FPGAS

http://www.xilinx.com/support/documentation/application_notes/xapp586-spi-flash.pdf

This application note by Arthur Yang describes the advantages of selecting a serial peripheral interface (SPI) flash as the configuration memory storage for the Xilinx 7 series FPGAs. The document includes the required connections between the FPGA and the SPI flash memory, and the details necessary to select the proper SPI flash.

The SPI flash is a simple, low-pin-count solution for configuring 7 series FPGAs. Support of indirect programming enhances ease of use by allowing in-system programming updates through reuse of connections already required for the configuration solution. Although some other configuration options permit faster configuration times or higher density, the SPI flash solution offers a good balance of speed and simplicity.

XAPP733: APPLYING MULTIBOOT AND THE LOGICORE IP SOFT-ERROR MITIGATION CONTROLLER

http://www.xilinx.com/support/documentation/application_notes/xapp733-multiboot-sem-controller.pdf

This document and six supporting reference designs detail the implementation of the MultiBoot feature and the LogiCORE IP Soft Error Mitigation (SEM) controller for Spartan®-6, Virtex-6 and 7 series FPGAs. The MultiBoot feature supports robust design management and field-upgrade capabilities. Designers initiate a MultiBoot event by commanding the FPGA to fully reconfigure itself with an alternate bitstream, usually over the Internal Configuration Access Port (ICAP).

Typically, says author Eric Crabill, the SEM controller has exclusive control of the ICAP to meet its functional and performance specifications. So, to apply MultiBoot and the SEM controller, you must coordinate ICAP sharing and might also need to organize multiple sets of SEM controller data in addition to multiple sets of FPGA configuration data. This application note demonstrates how to do so, while also illustrating how to organize multiple sets of SEM controller data. The supporting reference designs yield implementations for hardware evaluation on the SP605, ML605 and KC705 evaluation kits.

XAPP523: LVDS 4X ASYNCHRONOUS OVERSAMPLING USING 7 SERIES FPGAS

http://www.xilinx.com/support/documentation/application_notes/xapp523-lvds-4x-asynchronous-oversampling.pdf

Xilinx 7 series FPGAs can implement asynchronous communication using SelectIO™ interface resources, making it possible to reserve GT transceivers for other uses. This application note by Marc Defossez describes a method of capturing asynchronous communication using LVDS with SelectIO interface primitives. The method consists of oversampling the data with a clock of similar frequency (±100 ppm) by taking multiple samples of the data at different clock phases to get a sample of the data at the most ideal point.

The SelectIO interface in 7 series FPGAs can perform 4x asynchronous oversampling at 1.25 Gbits/second by using ISERDESE2 primitives. A mixed-mode clock manager (MMCME2_ADV) generates the clocks through dedicated high-performance paths between the components. The technique also helps to reduce cost by making it possible to select a smaller FPGA for your design.

XAPP538: SOFT ERROR MITIGATION USING PRIORITIZED ESSENTIAL BITS

http://www.xilinx.com/support/documentation/application_notes/xapp538-soft-error-mitigation-essential-bits.pdf

A soft error in configuration memory can corrupt a design. However, the proper operation of a typical design loaded into a Xilinx FPGA involves only a fraction of the total number of configuration memory cells. This application note describes a method for defining the hierarchical regions of interest in a design and identifying the prioritized essential bits associated with the defined user logic, using the ISE design tools, version 13.4 and later. Author Robert Le also explains how to use the LogiCORE IP Soft Error Mitigation (SEM) controller with the prioritized essential bits to detect and correct soft errors in the configuration memory of Xilinx 7 series and Virtex-6 FPGAs. The method reduces the effective failures in time (FIT) and increases design availability.

What's New in the Vivado 2012.4 Release?

The Vivado™ Design Suite 2012.4 is now available, at no additional cost, to all Xilinx® ISE® Design Suite customers that are currently in warranty. The Vivado Design Suite provides a highly integrated design environment with a completely new generation of system-to-IC-level features, including high-level synthesis, analytical place-and-route and an advanced timing engine. These tools enable developers to increase design integration and implementation productivity.

INTRODUCING VIVADO WEBPACK EDITION

The Vivado WebPACK™ tool enables you to simulate, synthesize and implement your design at no cost when targeting select devices. The Vivado Design Suite WebPACK Edition supports the Artix™-7 (7A100T, 7A200T) and Kintex™-7 (7K70T, 7K160T) devices.

You can download the WebPACK today at *www.xilinx.com/download*.

VIVADO 2012.4 DEVICE SUPPORT

The following devices are production ready:

- Virtex®-7 2000T (including Low Voltage)
- Artix-7 100T and 200T

The Virtex-7 X1140T device is in general engineering sampling.

LOGIC SIMULATION

Vivado now offers faster simulation models for GTHE2 primitives, featuring a 5x to 6x speedup over current models. The simulator now supports Aldec Riviera-PRO in compile_simlib.

VIVADO HIGH-LEVEL SYNTHESIS (HLS)

Xilinx has expanded register-transfer-level (RTL) co-simulation support to include four new simulators: Cadence's Incisive and Aldec's Riviera-PRO, as well as the Xilinx Vivado and ISE simulators.

Support has been added for floating-point single- and double-precision tan, atan, sinh, cosh and exponent cmath.h functions.

In addition, intellectual property (IP) exported in Pcore format now supports a fully or partially synthesized netlist, reducing the overall synthesis time when the IP is used in Xilinx Platform Studio. Vivado now supports Xilinx development boards as targets for synthesis in the graphical user interface (GUI). The Vivado HLS library contains 31 beta functions for video and OpenCV I/O interfaces.

SYSTEM GENERATOR FOR DSP

A new model-upgrade feature provides seamless migration when using the latest version of superseded blocks. Also, System Generator for DSP has several new floating-point blocks, including multiply add, accumulator and exponential. The latest version of the tool offers a 5x simulation time improvement for the DSP48 macro block. Models using multiple DSP48 macro blocks will see significant simulation-time improvement.

XILINX INTELLECTUAL PROPERTY

For a detailed list of Xilinx IP cores, see the IP Release Notes Guide (XTP025).

For more information regarding the latest version of Vivado, please see the Vivado 2012.4 Release Notes, at *http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_4/irn.pdf*.

Q: *WHAT IS THE VIVADO DESIGN SUITE?*

A: It's all about improving designer productivity. This entirely new tool suite was architected to increase your overall productivity in designing, integrating and implementing with the 28-nanometer family of Xilinx All Programmable devices. With 28-nm manufacturing, Xilinx devices are now much larger and come with a variety of new technologies including stacked-silicon interconnect, high-speed I/O interfaces operating at up to 28 Gbps, hardened microprocessors and peripherals, and analog/mixed signal. These larger and more complex devices present developers with multi-dimensional design challenges that can prevent them from hitting market windows and increasing productivity.

The Vivado Design Suite is a complete replacement for the existing Xilinx ISE Design Suite of tools. All of the capabilities of the ISE Design Suite point tools are now built directly into the Vivado integrated development environment (IDE), leveraging a shared scalable data model. With the Vivado Design Suite, developers are able to accelerate design creation with high-level synthesis and implementation by using place-and-route to analytically optimize for multiple and concurrent design metrics, such as timing, congestion, total wire length, utilization and power. Thanks to Vivado's shared scalable data model, the entire design process can be executed in memory without the need to write or translate any intermediate file formats, accelerating runtimes, debug and implementation while reducing memory requirements.

Vivado provides users with upfront metrics that allow for design and tool-setting modifications earlier in the design process, when they have less overall impact on the schedule. This capability reduces design iterations and accelerates productivity. Users can manage the entire design process in a pushbutton manner by using the Flow Navigator feature in the Vivado IDE, or control it manually by using Tcl scripting.

Q: *SHOULD I CONTINUE TO USE THE ISE DESIGN SUITE OR MOVE TO VIVADO?*

A: The ISE Design Suite is an industry-proven solution for all generations of Xilinx's All Programmable devices. The Xilinx ISE Design Suite continues to bring innovations to a broad base of developers, and extends the familiar design flow for 7 series and Xilinx Zynq™-7000 All Programmable SoC projects. ISE 14.4, which brings new innovations and contains updated device support, is available for immediate download.

The Vivado Design Suite 2012.4, Xilinx's next-generation design environment, supports 7 series devices including Virtex-7, Kintex-7 and Artix-7 FPGAs. It offers enhanced tool performance, especially on large or congested designs.

Q: *IS VIVADO DESIGN SUITE TRAINING AVAILABLE?*

A: Vivado is new and takes full advantage of industry standards such as powerful interactive Tcl scripting, Synopsys Design Constraints, SystemVerilog and more. To reduce your learning curve, Xilinx has rolled out new instructor-led classes to show you how to use the Vivado tools. For more information, please visit *www.xilinx.com/training*. We also encourage you to view the Vivado Quick Take videos found at *www.xilinx.com/training/vivado*.

Q: *ARE THERE DIFFERENT EDITIONS OF THE VIVADO DESIGN SUITE?*

A: The Vivado Design Suite comes in three editions: Design, System and now WebPACK, the no-cost, device-limited version of the tool suite. In-warranty ISE Design Suite Logic and Embedded Edition customers will receive the new Vivado Design Edition. ISE Design Suite DSP and System Edition customers will receive the new Vivado System Edition. To download today, please visit *www.xilinx.com/download*.

Xilinx recommends that customers starting a new design contact their local FAE to determine if Vivado is right for the design. Xilinx does not recommend transitioning during the middle of a current ISE Design Suite project, as design constraints and scripts are not compatible between the environments.

For more information, please read the ISE 14.4 and Vivado 2012.4 release notes.

Q: *WHAT ARE THE LICENSING TERMS FOR VIVADO?*

A: The Vivado Design Suite WebPACK Edition is a free download that provides support for Artix-7 100T and 200T and Kintex-7 70T and 160T devices. The Vivado Design Suite Design Edition is available at no additional cost to in-warranty ISE Design Suite Logic Edition and Embedded Edition customers, while the Vivado Design Suite System Edition with high-level synthesis is available at no additional cost to ISE Design Suite DSP and System Edition customers.

For customers who generated an ISE Design Suite license for versions 13 or 14 after Feb. 2, 2012, your current license will also work for Vivado. Customers who are still in warranty but who generated licenses prior to February 2 of last year will need to regenerate their licenses in order to use Vivado.

For license generation, please visit *www.xilinx.com/getlicense*.

Xpress Yourself in Our Caption Contest

DANIEL GUIDERA

If you're looking to Xercise your funny bone, here's your opportunity. We invite readers to swing over to our verbal challenge and submit an engineering- or technology-related caption for this cartoon showing a Tarzengineer in the jungle of his workplace. The image might inspire a caption like "Eddie made a New Year's resolution to get more exercise, but his colleagues disapproved of the way he retrofitted the lab."

Send your entries to *xcell@xilinx.com*. Include your name, job title, company affiliation and location, and indicate that you have read the contest rules at *www.xilinx.com/xcellcontest*. After due deliberation, we will print the submissions we like the best in the next issue of *Xcell Journal*. The winner will receive an Avnet ZedBoard, featuring the Zynq™-7000 All Programmable SoC (*http://www.zedboard.org/*). Two runners-up will gain notoriety, fame and a cool, Xilinx-branded gift from our swag closet.

The contest begins at 12:01 a.m. Pacific Time on Feb. 6, 2013. All entries must be received by sponsor by 5 p.m. PT on April 1, 2013.

So, grab a vine and get swinging!

ROBERT TAYLOR, senior engineer at MacAulay Brown, Inc. (Dayton, Ohio), won a shiny new Avnet ZedBoard with this caption for the cartoon of lab cats in Issue 81 of *Xcell Journal*:

"I asked for anti-static mats, not cats!"

Congratulations as well to our two runners-up:

Bob had his hopes of designing with Vivado dashed when he discovered his new abbreviated job title actually stood for "Feline Play and Grooming Accessory Design Engineer."

– John Blessing, senior design engineer, Garmin International (Olathe, Kans.)

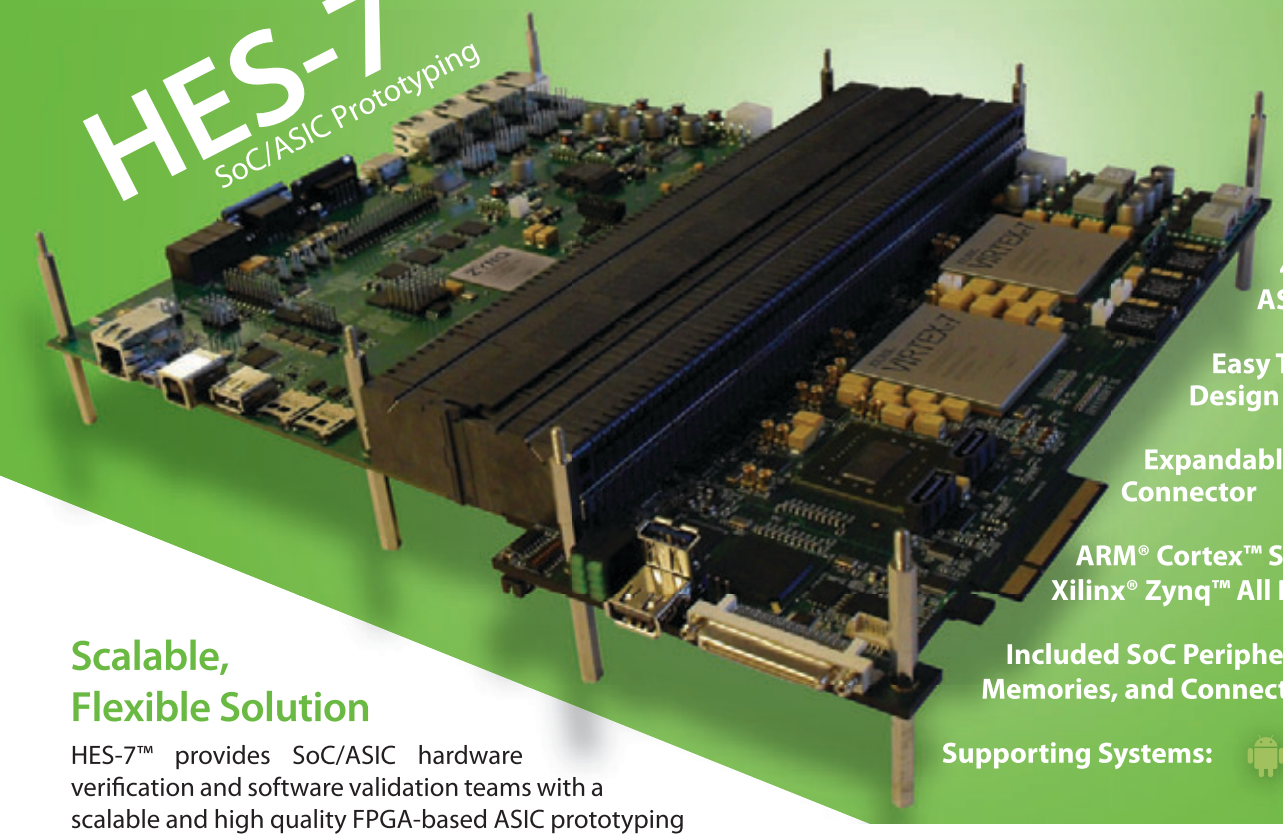
"I thought you said you had a mouse problem."

– Hector Herrera, lead technologist, Pier Programming Services Ltd. (Vancouver, British Columbia)

member XILINX ALLIANCE PROGRAM

HES-7™

SoC/ASIC Prototyping



4 to 96 million Scalable ASIC Gate Capacity

Easy To Use with Reduced Design Partitioning

Expandable via Non-Proprietary Connector

ARM® Cortex™ Support with Xilinx® Zynq™ All Programmable SoC

Included SoC Peripherals: Media Interfaces, Memories, and Connectors

Supporting Systems:

Scalable, Flexible Solution

HES-7™ provides SoC/ASIC hardware verification and software validation teams with a scalable and high quality FPGA-based ASIC prototyping solution backed with an industry leading 1-year limited warranty. Each HES-7 board with dual Xilinx® Virtex®-7 2000T has 4 million FPGA logic cells or up to 24 million ASIC gates of capacity, not including the DSP and memory resources.

The HES-7 prototyping solution was architected to allow for easy implementation and expansion. Using only one or two large FPGAs, rather than multiple low density FPGAs, HES-7 does not require as much labor-intensive partitioning or tool expense. Using a non-proprietary HES-7 backplane connector, HES-7 can easily expand prototype capacity up to 96 million ASIC gates and can include the expansion of daughter boards.

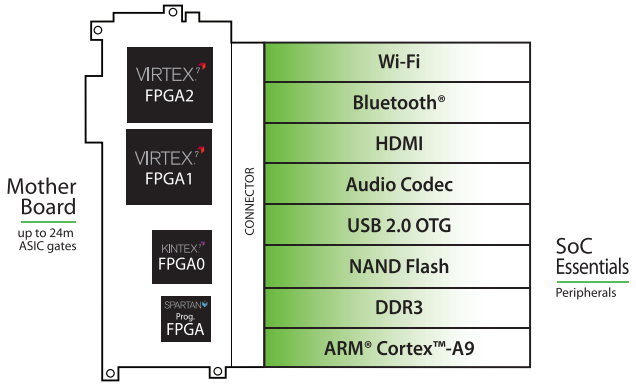
ARM Cortex Support

HES-7 supports ARM® dual-core Cortex™-A9 MPCore™ with Xilinx® Zynq™-7000 All Programmable SoC, allowing designers to leverage the serial processing capabilities of the Cortex-A9 processor for applications that require intensive computations and operating systems with the parallel processing capabilities of HES-7 ASIC prototyping platform to create applications across a diverse range of markets including: Video, Communications, Control Systems and Bridging.

The 4 to 96 million ASIC gate scalable capacity of HES-7, coupled with open-source Linux, Android, and FreeRTOS solutions available from Xilinx, delivers a powerful verification platform for HW/SW design teams.

HES-7™

SoC/ASIC Prototyping Platform



Essential SoC Peripherals

HES-7 provides SoC Peripherals, allowing designers the ability to interface real-world stimuli to the design-under-test (DUT). Users can utilize gigabit Ethernet transceivers to develop networking applications, WLAN 802.11 b/g/n and Bluetooth® v2.1 to develop wireless systems, or High Performance HDMI Transmitter to develop home entertainment products. Memories and connectors provide additional data storage for read/write capability of today's popular memory interfaces, and the ability to connect external hardware with the HES-7.

Aldec, Inc.
Corporate - **N. America**
Ph +1.702.990.4400
sales@aldec.com

www.aldec.com

Israel
Ph. +972.072.2288316
sales-il@aldec.com

China
Ph. +86.21.6875.2030
info@aldec.com.cn

Europe
Ph. +44.1295. 20.1240
sales-eu@aldec.com

India
Ph. +91.80.3255.1030
sales-in@aldec.com

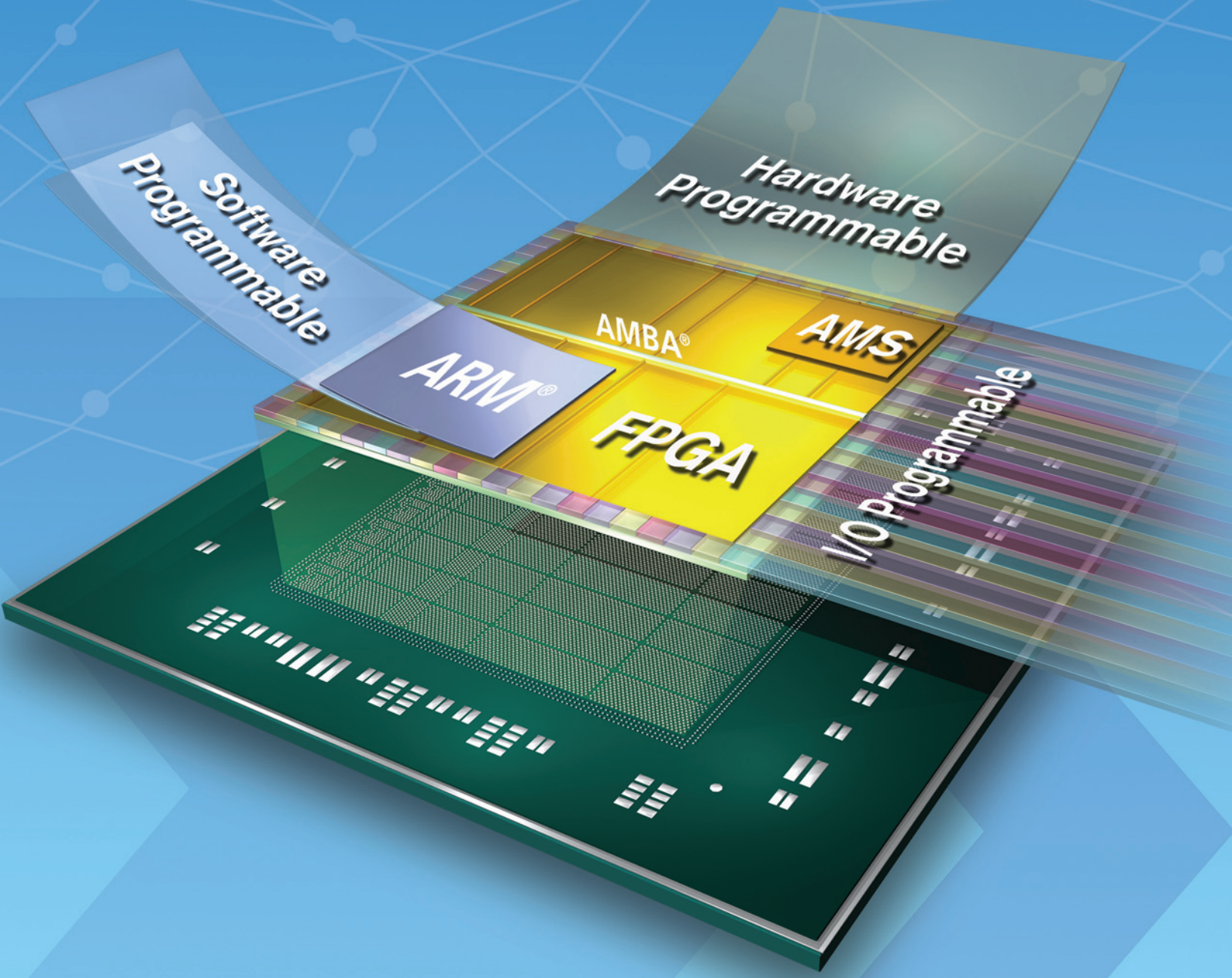
Japan
Ph. +81.3.5312.1791
sales-jp@aldec.com

Taiwan
Ph. +886.3.6587712
sales-tw@aldec.com

ALDEC®
THE DESIGN VERIFICATION COMPANY

A Generation Ahead

Hardware, Software and I/O Programmable SoC



LEARN MORE

XILINX
ALL PROGRAMMABLE™

PN 2535