ISSUE 85, FOURTH QUARTER 2013

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

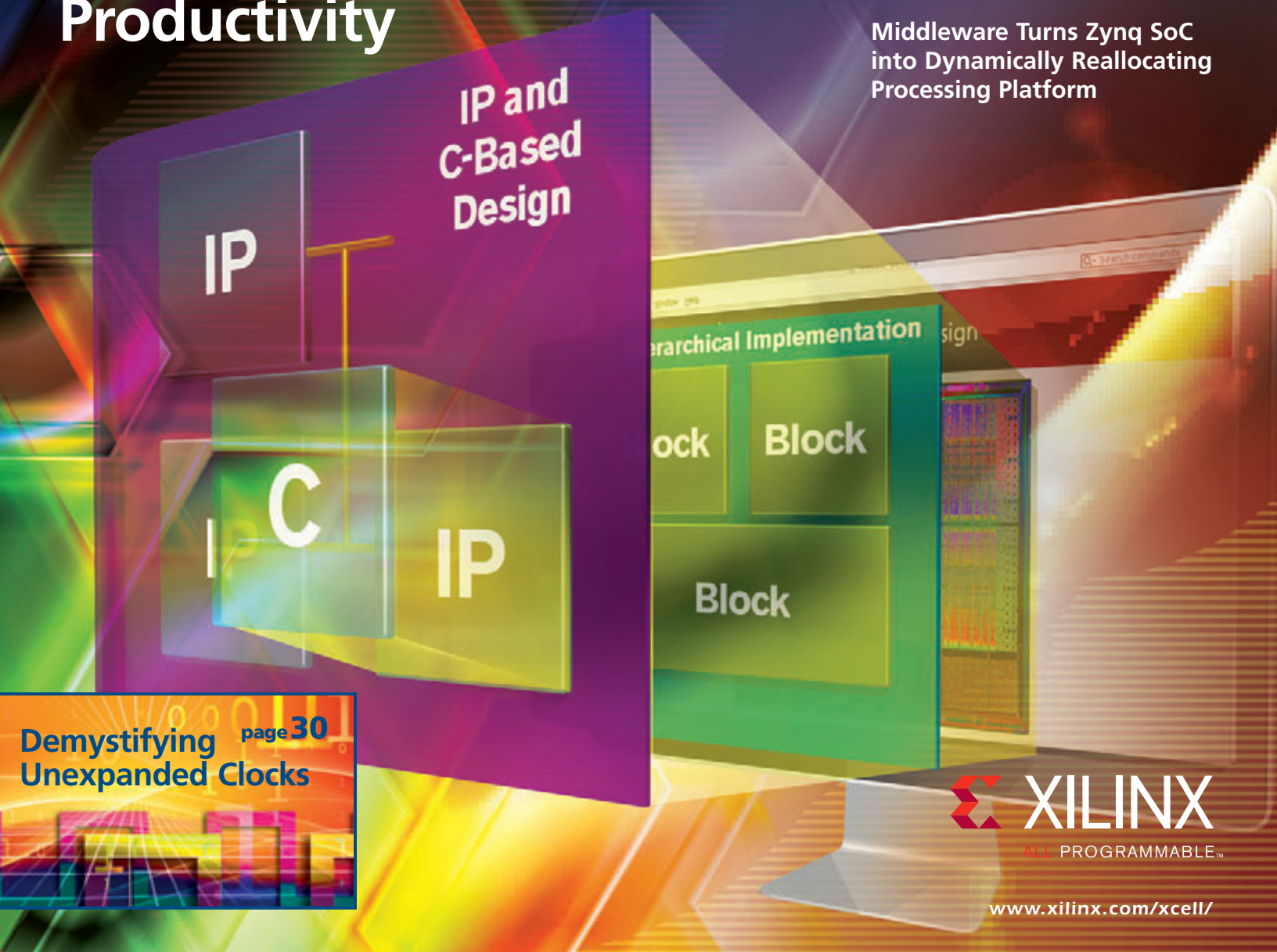
Xilinx's UltraFast Methodology: A Formula for Generation-Ahead Productivity

Accelerate Cloud Computing with the Zynq SoC

Zynq SoC Enables Red Pitaya Open-Source Instruments

How to Design IIR Filters for Interpolation and Decimation

Middleware Turns Zynq SoC into Dynamically Reallocating Processing Platform



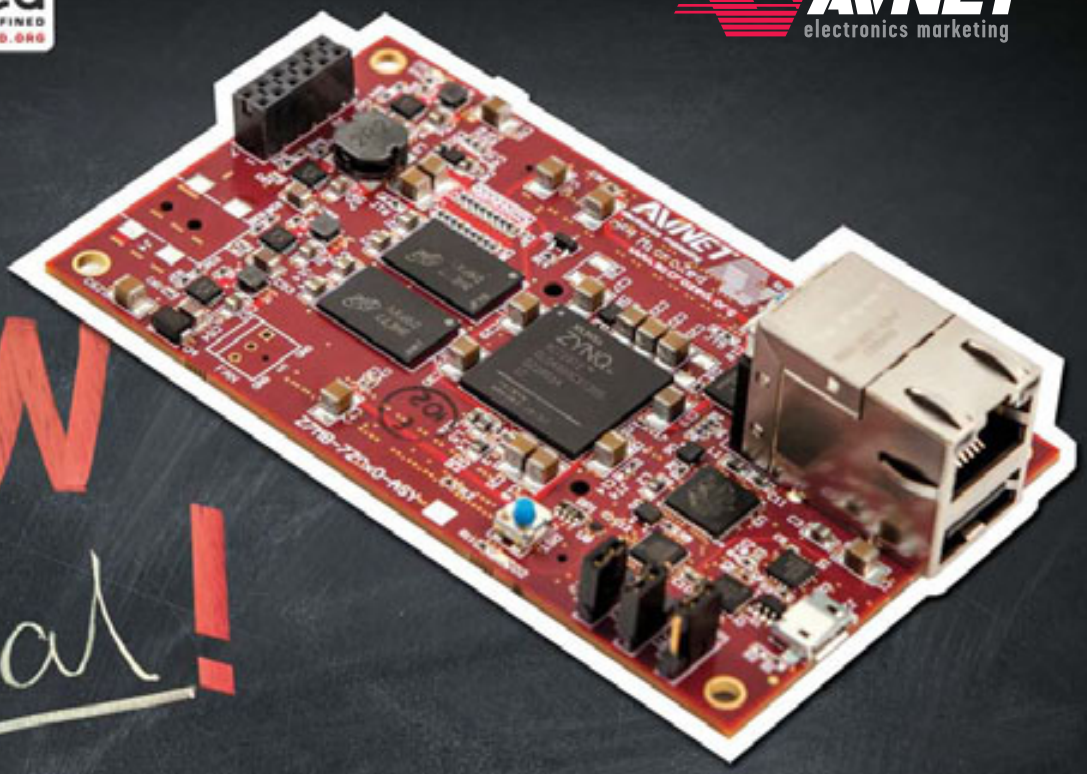
Demystifying Unexpanded Clocks page 30

XILINX
ALL PROGRAMMABLE™

www.xilinx.com/xcell/

MicroZed
FLEXIBILITY REDEFINED
MICROZED.ORG

AVNET
electronics marketing



NEW Arrival!

Welcome to the Family!

- √ Low-cost development tool
- √ Off-the-shelf SOM solution
- √ Out-of-the-box Linux support
- √ Easy migration from prototype to production

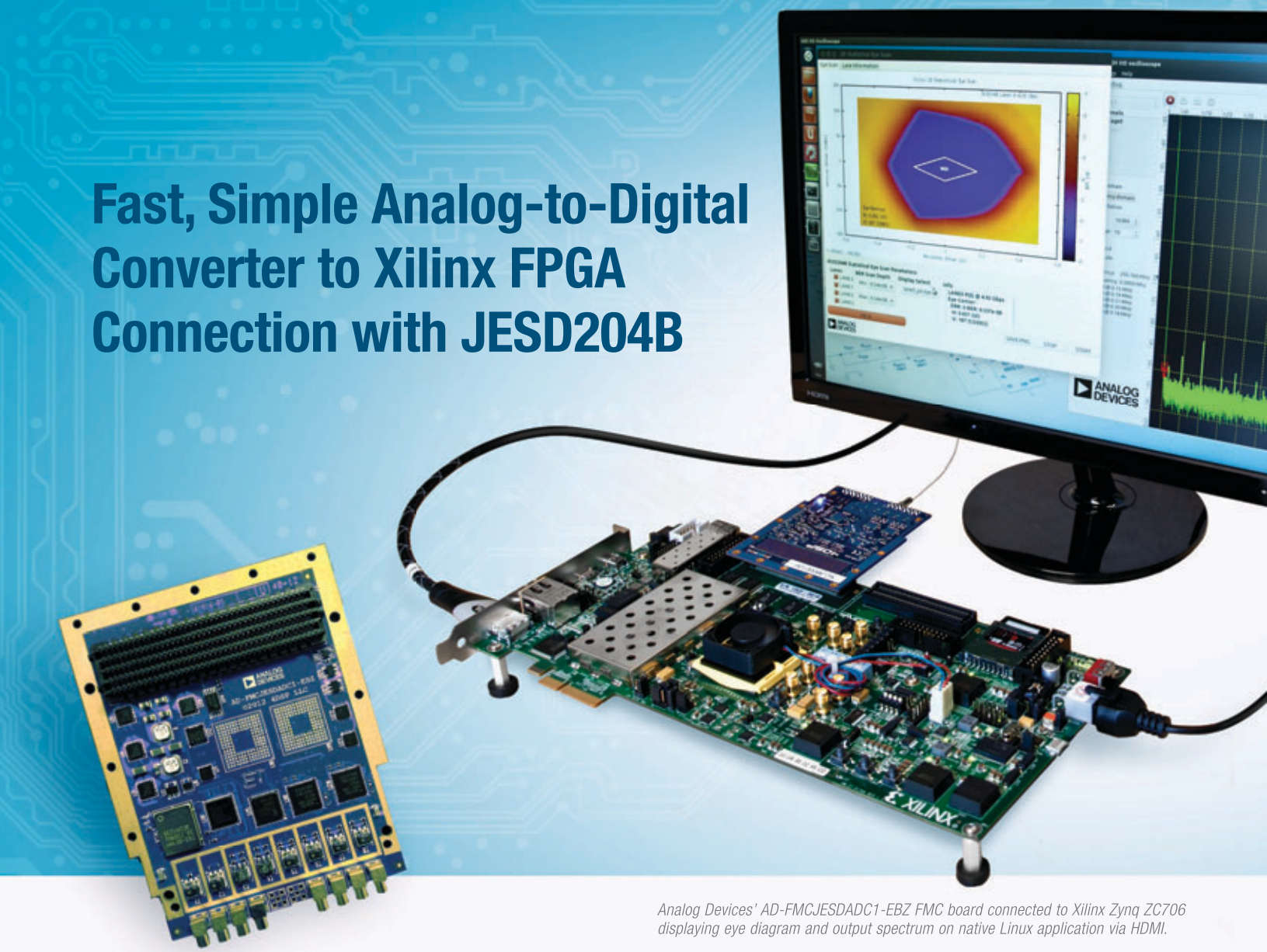
ZYNQ®

www.microzed.org

MicroZed™ is a low-cost development board based on the Xilinx Zynq®-7000 All Programmable SoC. Its unique design allows it to be used as both a stand-alone evaluation board for basic SoC experimentation, or combined with a carrier card as an embeddable system-on-module (SOM). This combined stand-alone/SOM approach can quickly move a design idea from concept to production, making MicroZed the ideal platform for SoC based applications. MicroZed is supported by a community website where users can download kit documentation and reference designs as well as collaborate with other engineers also working on Zynq designs.

DESIGNED BY AVNET

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

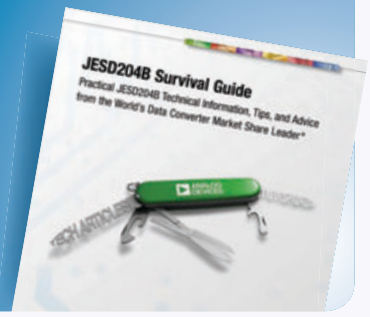


Analog Devices' AD-FMCJESDADC1-EBZ FMC board connected to Xilinx Zynq ZC706 displaying eye diagram and output spectrum on native Linux application via HDMI.

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

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

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



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

ANALOG DEVICES

Xcell journal

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

EDITOR Jacqueline Damian

ART DIRECTOR Scott Blair

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

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

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

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

Tomoko Suto, Japan
tomoko@xilinx.com

REPRINT ORDERS 1-800-493-5551

Xcell PUBLICATIONS
www.xilinx.com/xcell/

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

Xcell Journal Marks 25th Anniversary with Launch of *Xcell Daily*

This quarter marks the anniversary of *Xcell Journal's* 25th year of publication. Since its humble beginnings in 1988 as an eight-page newsletter published by the late, great Peter Alfke, *Xcell Journal* has grown and evolved in tandem with Xilinx's growth from a tiny startup offering a radical new device called an FPGA to a dynamic supplier of billion-transistor All Programmable devices. As Xilinx's customer base has expanded to more than 20,000 brands worldwide over the last 30 years, *Xcell Journal* now claims upwards of 50,000 subscribers around the world and is published in three languages: English, Chinese and Japanese. The English version of last quarter's issue was downloaded more than 100,000 times, and its articles were syndicated in dozens of tier-one trade publications globally.

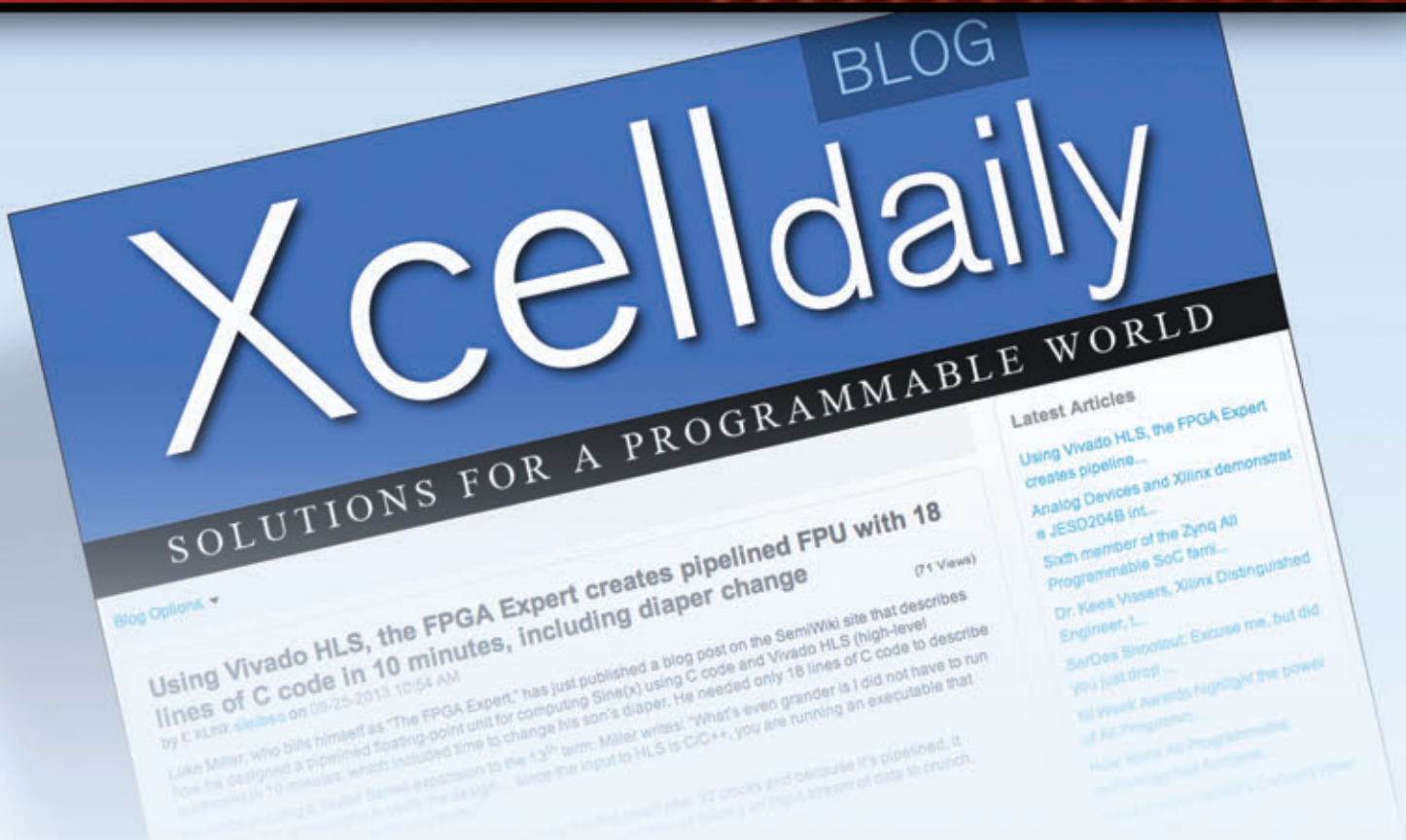
Xcell Journal's wide reach is a testament to the growing popularity of Xilinx's technology. It also attests to the talents of countless contributors from Xilinx's customer base and among Xilinx employees, and their willingness to share their experiences about how they use our technology to create remarkable, life-enhancing products. To build on this success and keep customers even more informed of the latest developments, we at Xilinx are happy to announce an Xpansion of the Xcell brand with our launch of a new online news channel called *Xcell Daily* (*http://forums.xilinx.com/t5/Xcell-Daily-Blog/bg-p/Xcell*).

Xcell Daily is run by my longtime friend and colleague Steve Leibson, now the director of strategic marketing and business planning at Xilinx. If the name sounds familiar to you, it's because Steve was the editor-in-chief of *EDN* magazine during the 1990s and later became the EIC of *Microprocessor Report*. He left the trade publishing world to do content marketing for processor IP startup Tensilica and, later, Denali Software and Cadence. Steve was one of the first in the IC design business to embrace blogging, and over the years he has written several popular blogs such as Leibson's Law, which *EDN* syndicated for many years.

With *Xcell Daily*, Steve and his contributors will keep readers informed on a daily basis of the latest goings on at Xilinx and in the user community. I encourage you to check the site often during your work week. You may even want to hit up Steve to become a contributor, whether regular or occasional. And of course, please keep reading and sending great contributed articles to us at *Xcell Journal*. We haven't been printing *Xcell Journal* on paper for a couple of years now, but we keep growing in readership, depth and breadth thanks in large part to the premier quality of technical contributions from you. You make *Xcell* great.

Mike Santarini
Publisher

Xcell Journal Adds New Daily Blog



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

Recent Posts Include:

- *Zynq-based Red Pitaya Open Instrumentation Platform Blows Past $50K Kinstarter Funding Goal by 5x*
- *Mars Curiosity Rover's MAHLI Images of Dusty Penny on Mars with 14nm Resolution*
- *100Gbps Ethernet over CFP2 Optical Modules Crosses 10km of Fiber with Zero Errors Using Virtex-7 H580T FPGA*
- *Xilinx Virtex-7 VC709 Connectivity Kit Serves as a Ready-to-Go, 40Gbps Development Platform*
- *Analog Devices and Xilinx Demonstrate JESD204B Interoperability for High-speed ADCs*

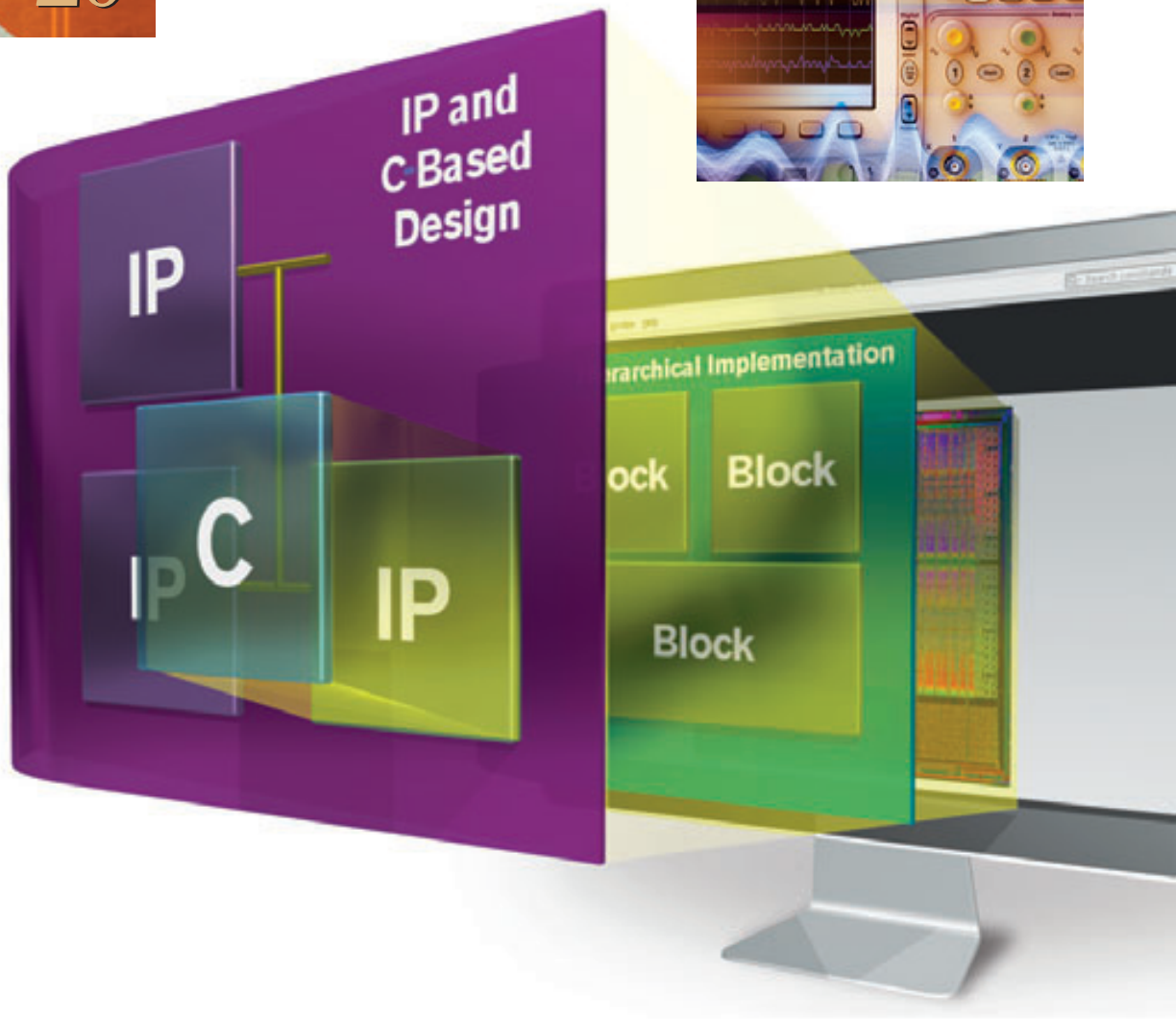
Visit Blog: www.forums.xilinx.com/t5/Xcell-Daily/bg-p/Xcell

XILINX
ALL PROGRAMMABLE.

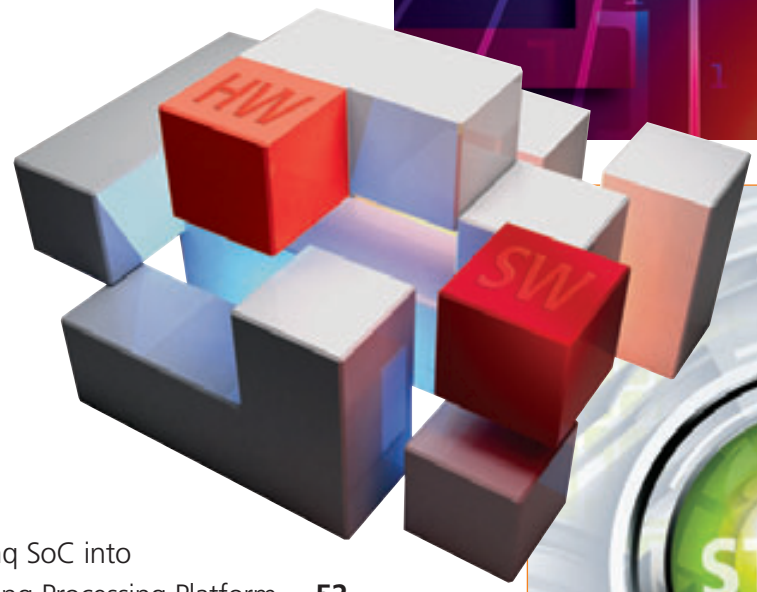
VIEWPOINTS

XCELLENCE BY DESIGN APPLICATION FEATURES

Cover Story



THE XILINX XPERIENCE FEATURES



52

XTRA READING

APEX® AWARDS FOR PUBLICATION EXCELLENCE

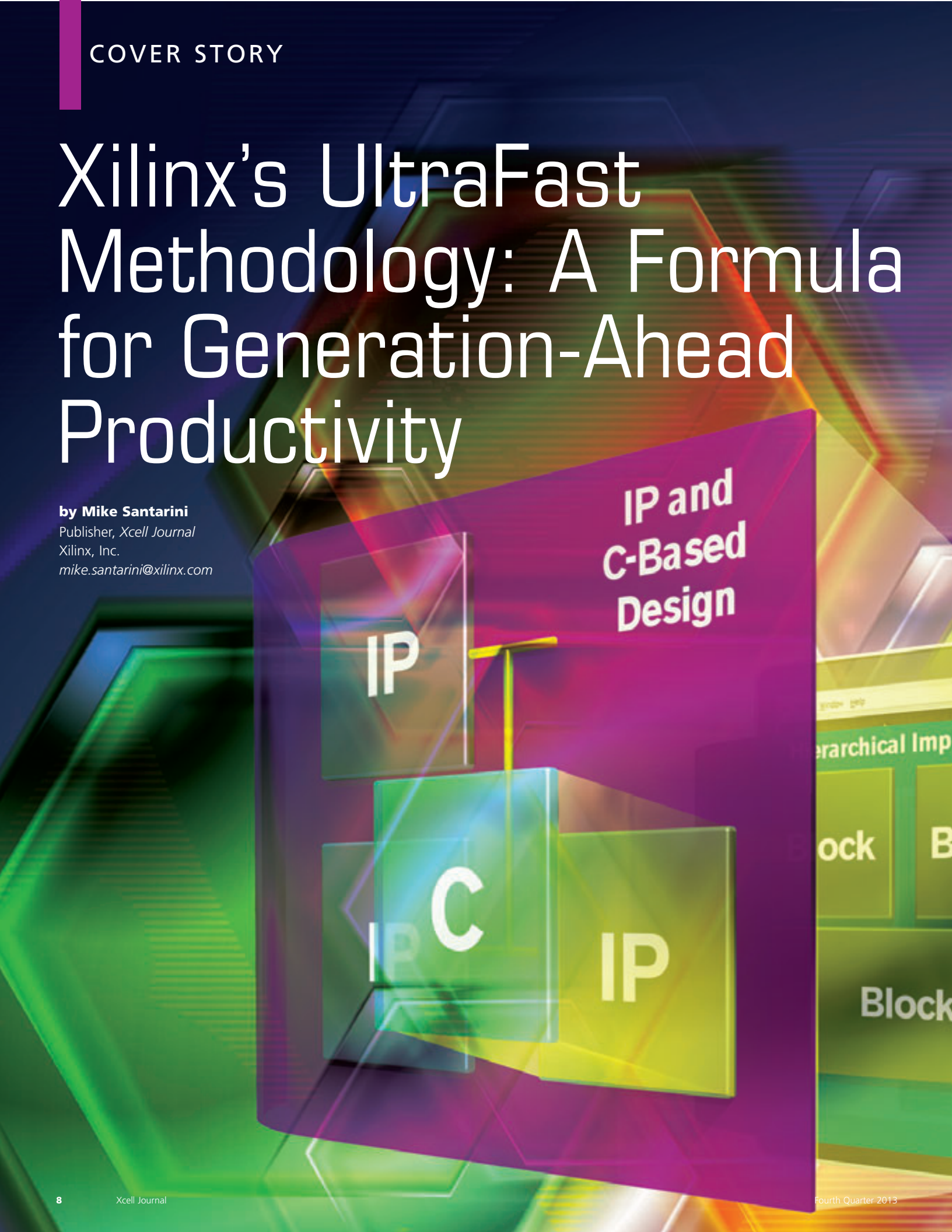
Excellence in Magazine & Journal Writing
2010, 2011

APEX® AWARDS FOR PUBLICATION EXCELLENCE

Excellence in Magazine & Journal Design and Layout
2010, 2011, 2012

Xilinx's UltraFast Methodology: A Formula for Generation-Ahead Productivity

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



Hands-on techniques geared to tools in the Vivado Design Suite ensure predictable, repeatable results.

The relentless progression of IC process technology over the last 40 years has enabled electronics companies to create the boundless array of products we all enjoy today. But while this advancement of silicon process technology has been critically important to electronic innovation, it would not have been possible without the simultaneous and rapid evolution of tools from academia and the EDA industry. The progression from transistor-level SPICE simulation in the 1970s to today's highly advanced billion-gate system-level integrated design environments is truly remarkable. What's equally remarkable but often overlooked for its essential contribution to the electronics revolution is design methodology.

A design team can have access to the most advanced silicon in the world and the greatest tools in the world, but if the group doesn't establish a solid methodology it's difficult to deliver products at the right time and turn that rollout into business success. A good design methodology not only cuts down design time to allow teams to deliver quality products on deadline, but allows them to do so in a predictable and repeatable manner—a key to long-term business success. That said, methodologies must constantly evolve to take advantage of advances in both silicon and design tools.

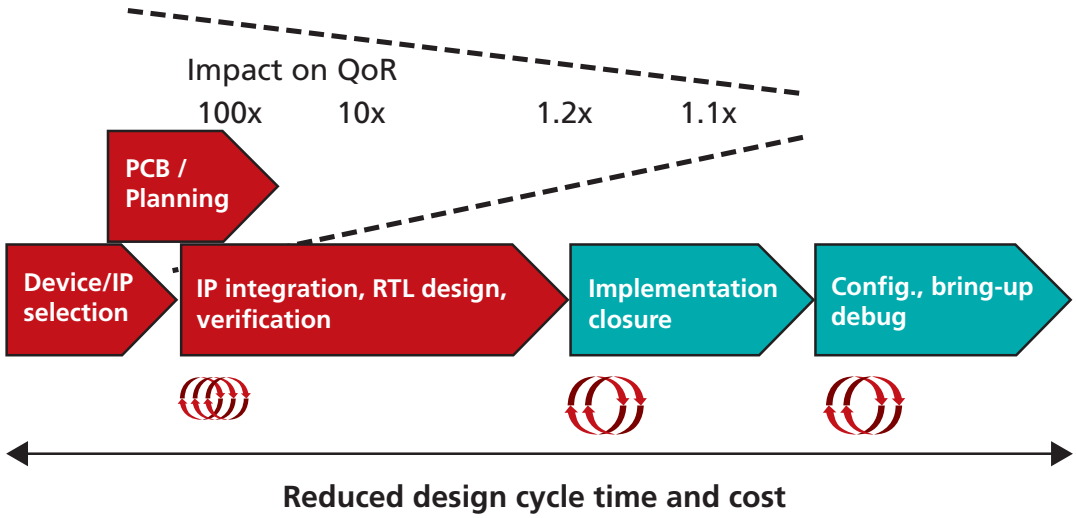


Figure 1 – Achieving closure at the beginning of the flow drastically improves the quality of results and cuts design time.

To help customers become even more productive and successful in rolling out next-generation innovations, Xilinx has just launched its UltraFast™ Design Methodology for design teams using the award-winning Vivado® Design Suite to implement designs in Xilinx®'s 28-nanometer 7 series and upcoming 20/16-nm UltraScale™ product families.

Ramine Roane, senior director of product marketing for the Vivado Design Suite, said this new methodology does not represent a radical shift; rather, with UltraFast, the company has hand-picked the best practices from industry experts and distilled these practices into a potent methodology that will speed design teams to predictable success. These best practices cover all aspects of board planning, design creation, design implementation and closure, programming and hardware debug.

"The UltraFast Design Methodology will enable design teams to take full advantage of the state-of-the-art technologies in the Vivado Design Suite and the Xilinx All Programmable devices, to accelerate productivity and repeatedly deliver designs to shorter, predictable schedules and thus deliver products to market sooner," said Roane.

The UltraFast methodology is yet another example of how Xilinx is maintaining its Generation Ahead lead over the competition. Xilinx not only has the best devices and most modern tool suite, but also the most comprehensive methodology in the industry.

To speed the adoption of the UtraFast Design Methodology, Xilinx has published a free methodology manual. *The UltraFast Methodology Guide for the Vivado Design Suite* (UG949) walks readers through an entire methodology, from board selection and RTL design to implementation and final debug. The document includes a comprehensive checklist designed to guide engineers throughout the design flow. Additionally, the 2013.3 release of the Vivado Design Suite automates many aspects of the methodology, including linting, and adds new DRC rule decks entitled "Methodology" and "Timing." The new version of the Vivado Design Suite also includes HDL and XDC templates, enabling an optimal-by-construction quality of results (QoR) for synthesis and implementation. Xilinx is also offering a series of free self-training videos online, as well as official training courses at various locations worldwide.

DESIGN METHODOLOGY FOR RAPID CONVERGENCE

Roane said that the UltraFast Methodology's main theme is to bring design closure to the front end of the design flow, where the impact on quality of results is greater (see Figure 1). And in this way, design teams can rapidly converge on a correct-by-construction design. "If you make informed decisions earlier in the flow levels, you will effectively eliminate much longer cycles in the implementation phases," said Roane.

Roane said this process was enabled by the Vivado Design Suite, which is the only design suite in the programmable industry to offer interactive design analysis and cross-probing to the sources at each step of the

flow, starting at design entry and continuing though IP integration, RTL synthesis, implementation (optimization, placement, physical optimization, routing) and bring-up. "With traditional tools, designers can only discover issues at the end of the process, on the fully implemented floorplan," said Roane. "If their design doesn't function as expected, their only choice is to backtrack to their initial design steps with little clue of what caused the problem, and incur many long iterative loops."

The key enabler of this cross probing and analysis capability is the Vivado Design Suite's unified data model. "The unified data model enables design teams to use the same analysis and closure routines throughout the entire flow," said Roane. "It's a huge advantage over older design suites, as it enables engineers to modify their design at multiple steps, even in-memory, and cross-probe any trouble area back to the sources or any other design view. Xilinx designed this unified data model to scale to high-end devices with many millions of logic cells, while competing tools have already started to break down on midrange devices."

Perhaps the best example to illustrate the UltraFast methodology's rapid design convergence is the concept of "baselining."

BASELINING TO RAPIDLY CONVERGE ON TIMING CLOSURE

"Baselining is a technique used to accelerate design convergence by focusing on internal F_{max}, which is the biggest problem nine times out of 10," said Roane. "This avoids wasting time with complex and extremely error-prone I/O constraints and timing exceptions, which can lead users and the tools in the wrong direction. With baselining, teams start the convergence process with the simplest of constraints, focused on flip-flop paths. Then—depending on whether the problem is in the clock path or a datapath, in an interconnect delay or logic delay—you apply the documented fixes and rerun the analysis."

Once design teams have closed timing with the baseline XDC, they're almost done. They then need to add I/O interface constraints. "It is important to ensure that these constraints are correct, in order to not create 'false' timing issues," said Roane. "This is why we provide XDC templates for source-synchronous, center-aligned DDR I/O constraints, for instance. If need be, constraints can be fine-tuned with timing exceptions or do a bit of floorplanning. But it is important to note that exceptions are not helpful, unless the corresponding paths show as critical. Similarly, over-floorplanning a design can be more harmful than helpful."

Roane said that baselining is not, however a substitute for sign-off constraints: "You still need to validate your design against complete constraints."

The UltraFast methodology also describes detailed steps to get to pristine sign-off constraints. Vivado automates this task with numerous batch and graphical routines to analyze timing paths, clock networks, interactions between clocks and more. The new Timing DRC rule deck can also be used for linting the constraints and clock networks in the design.

THE ULTRAFAST METHODOLOGY GUIDE FOR THE VIVADO DESIGN SUITE

To become familiar with the UltraFast Methodology, the first place to start is by

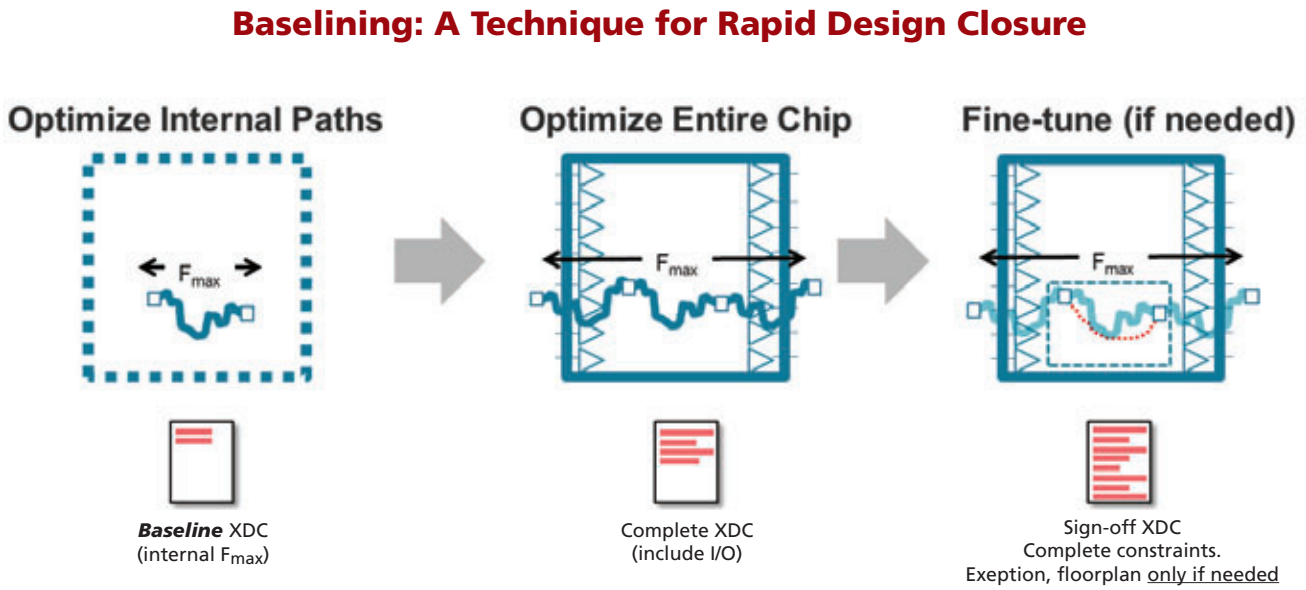


Figure 2 – Baselining allows design teams to quickly achieve timing closure.

reading *The UltraFast Methodology Guide for the Vivado Design Suite*. The guide is organized into six chapters, with the first two chapters introducing the content of the guide and recommendations for design suite flows. Chapters 3 through 6 dive deep into the best practices of the UltraFast methodology.

Chapter 3 addresses board and device planning and offers wise recommendations for PCB layout, clock resource planning and assignment, I/O planning design flows, FPGA power aspects and system dependencies. In order to avoid board respins, the methodology outlines the use of the Xilinx Power Estimator to explore and find architectures that meet the allocated power budget.

The chapter also emphasizes the importance of kicking off design projects with good I/O planning and recommends performing it in conjunction—or at least aligned—with board planning. Failing to align I/O and board planning can create timing and power distribution problems at a system level late in the process. The chapter also covers the various power modes, recommendations for power and thermal analysis, as well as cooling considerations for PCBs. It includes I/O recommendations when implementing Xilinx All Programmable 3D ICs in design projects, as the interposer connecting these multidice devices has unique requirements.

Chapter 4, "Design Creation," starts out with strategies and tips for creating a solid design hierarchy, selecting IP appropriate for your design, and then goes into several sections that offer practical RTL coding guidelines. It includes sections on control signals and sets, inferring RAM and ROM, coding for proper DSP and arithmetic inferences, coding shift registers and delay lines, and initialization of all inferred registers, SRLs and memories. The chapter includes a section on parameter attributes and constraints, as well as a section on clocking and whether to instantiate or infer.

'The way you write your HDL can make a big difference in how synthesis infers the logic. Good coding style will enable your design to use the hard blocks in the architecture and thus run at a higher frequency.'

"The way you write your HDL can make a big difference in how synthesis infers the logic," said Roane. "Good coding style will enable your design to use the hard blocks in the architecture and thus run at a higher frequency. To help customers leverage these resources to the fullest and thus speed their overall designs, we include templates to guide the inference of these components, particularly for RAM, shift registers and DSP resources. These templates are built into the Vivado Design Suite 2013.3"

Another highlight of the chapter are three sections dedicated to individual coding techniques for high reliability, improved performance and power optimization. Each section offers constraint recommendations to take advantage of the Vivado Design Suite's unified data model.

The section titled "Coding Styles for High Reliability" includes recommendations for clock domain crossings (synchronous and asynchronous), untimed resets and avoidance of combinatorial loops. The section on "Improved Performance" includes recommendations for high fan-outs in critical paths, register replication and considerations for implementing pipelining in designs. The "Coding Styles to Improve Power" section reviews various power-savings techniques you can employ in your design, from the tried-and-true datapath and clock gating to more subtle recommendations such as maximizing gating elements and keeping control sets in check.

Chapter 5 focuses on the implementation flow, from synthesis to routing. It kicks off with an overview of implementation, synthesis attributes and the advantages of a bottom-up flow. As discussed previously, the chapter includes a very deep look at timing analysis and introduces the concept of baselining for timing closure. Above and beyond that, the timing-closure section offers several great recommendations for what to do and what not to do for various timing problems you may encounter. It also covers considerations timing may have on power.

The last chapter in the guide, Chapter 6, covers configuration and debug. The first half walks readers through the best methods for generating a bitstream and programming the bitstream into your targeted Xilinx All Programmable device. The second half mainly presents best practices for debugging your design at multiple stages of the flow. This section discusses how to implement an HDL instantiation debug probing flow, or alternatively, how to use a netlist insertion debug probing flow. It also includes strategies for debugging the design once you've loaded it into your targeted device.

The guide's appendix provides a wealth of additional resources, perhaps the most significant being the UltraFast Design Methodology checklist, which highlights things a design team should consider at each stage in the cycle, from design planning to hardware debug. "It includes a lengthy list of questions highlighting the typical areas in which design decisions are likely to have downstream ramifications," Roane said. The checklist links readers to the areas in the guide or to external links that best describe that particular design concern. Xilinx also provides the checklist as a downloadable spreadsheet.

SUPPORT FOR THE ULTRAFAST METHODOLOGY

In addition to compiling all these best practices into the very useful *UltraFast Methodology Guide for the Vivado Design Suite*, Xilinx has incorporated many of the recommendations in the UltraFast methodology into its 2013.3 release of the Vivado Design Suite. With the 2013.3 release, the Vivado Design Suite now supports Methodology and Timing DRC rule decks to better guide users through design cycles, and includes very handy HDL and constraints templates for correct-by-construction designs.

In addition, Xilinx's worldwide training staff and Alliance Member ecosystem are actively supporting the UltraFast methodology.

For example, Blue Pearl Software has added Xilinx UltraFast design rules to its Analyze RTL linting tool. "Blue Pearl automates the RTL guidelines outlined in the UltraFast methodology," said Roane. "In addition to performing language linting, it enforces coding styles that give the optimal QoR for Xilinx devices, such as use of the right type of reset, or coding a RAM or a MAC in a way to infer our built-in blocks in the most optimal way."

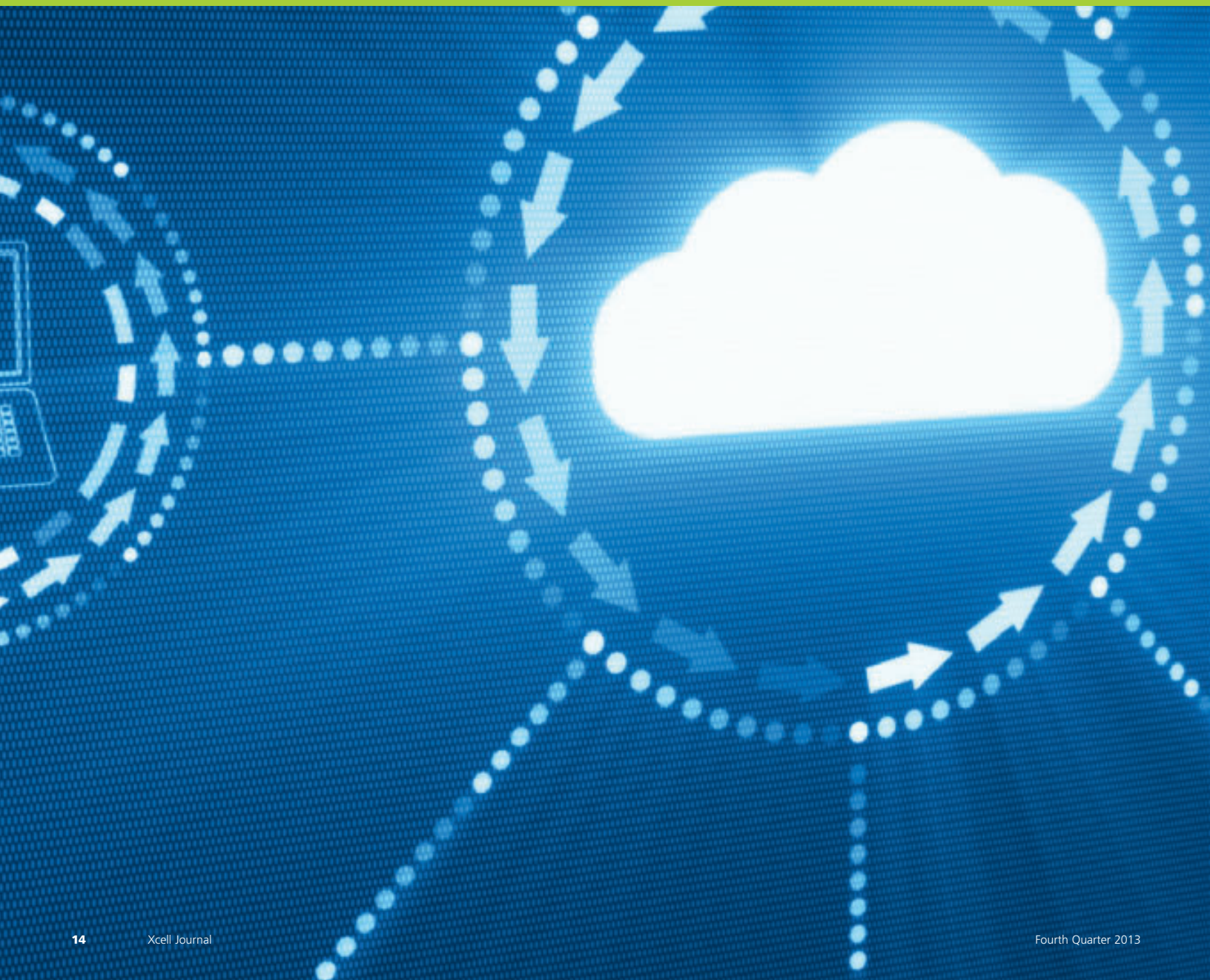
In addition to third-party EDA support, Xilinx has been actively testing its IP to ensure it conforms to the UltraFast methodology and DRCs and is actively encouraging Alliance Member IP vendors to ensure they comply to these same guidelines as well.

Last but not least, the company is launching a series of training classes conducted by Xilinx and partners worldwide. It is also releasing a new UltraFast methodology QuickTake video. In addition, all new Xilinx videos will also incorporate the UltraFast guidelines.

To download a PDF version of the *UltraFast Design Methodology Guide* for the Vivado Design Suite (UG949) and for further information on the UltraFast Methodology, visit *www.xilinx.com/ultrafast*.

Accelerate Cloud Computing with the Xilinx Zynq SoC

A novel reconfigurable hardware accelerator speeds the processing of applications based on the MapReduce programming framework.



by Christoforos Kachris
Researcher
Democritus University of Thrace
ckachris@ee.duth.gr

Georgios Sirakoulis
Professor
Democritus University of Thrace
gsirak@ee.duth.gr

Dimitrios Soudris
Professor
National Technical University of Athens (NTUA)
dsoudris@microlab.ntua.gr

Emerging Web applications like streaming video, social networks and cloud computing have created the need for warehouse-scale data centers hosting thousands of servers. One of the main programming frameworks for processing large data sets in the data centers and other clusters of computers is the MapReduce framework [1]. MapReduce is a programming model for processing large data sets using a high number of nodes. The user specifies the "Map" and the "Reduce" functions, and the MapReduce scheduler distributes the tasks to the processors.

One of the main advantages of the MapReduce framework is that it can be hosted in heterogeneous clusters consisting of different types of processors. The majority of data centers are based on high-performance, general-purpose devices such as the Intel Xeon, AMD Opteron and IBM Power processors. However, these processors consume a lot of power even in cases when the applications are not so computationally intensive but rather, are I/O-intensive.

To reduce the power consumption of the data centers, microservers have recently gained attention as an alternative platform. These low-cost servers are generally based on energy-efficient processors such as the ones used in embedded systems (for example, ARM® processors). Microservers mainly target lightweight or parallel applications that benefit most from individual servers with sufficient I/O between nodes rather than high-performance processors. Among the many advantages of the microserver approach are reduced acquisition cost, reduced footprint and high energy efficiency for specific types of applications.

In the last few years several vendors, such as SeaMicro and Calxeda, have developed microservers based on embedded processors. However, the MapReduce framework allocates several resources from the embedded processors, reducing the overall performance of the cloud-computing application running on these platforms.

To overcome this problem, our team has developed a hardware acceleration unit for the MapReduce framework that can be combined efficiently with ARM cores in fully programmable platforms. To develop and evaluate the proposed scheme, we selected the Xilinx® Zynq®-7000 All Programmable SoC, which comes hardwired with a dual-core Cortex-A9 processor onboard.

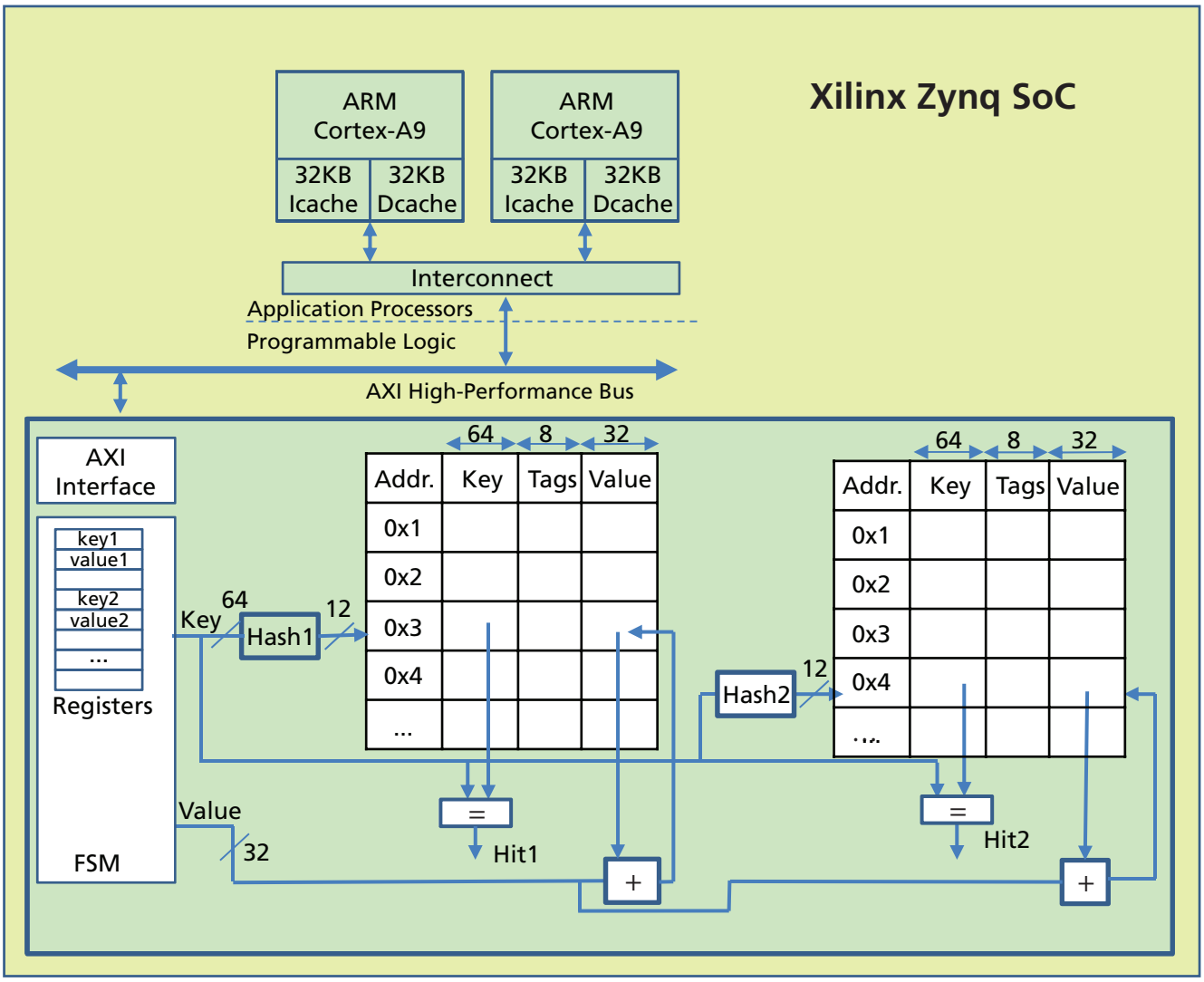


Figure 1 – Block diagram of the MapReduce hardware accelerator

MAPREDUCE HARDWARE ACCELERATOR UNIT

The MapReduce acceleration unit handles the efficient implementation of the Reduce tasks. Its main job is to merge the intermediate key/value pair from all the processors and to provide a fast access for the insertion of new keys and the updates (accumulation) of key/value pairs. We implemented the MapReduce accelerator as a coprocessor that can be augmented to multicore processors through a shared bus. The block diagram of the accelerator inside multicore SoCs is shown in Figure 1.

As the figure shows, we have incorporated the hardware accelerator unit into a Zynq SoC equipped with a pair of ARM Cortex™-A9 cores. Each of these cores has its own instruction- and data-level cache, each of which communicates with the peripheral using a shared interconnection network. The accelerator communicates with the processors through a high-performance bus that is attached to the interconnection network. The processors emit the key and the value that have to be updated to the MapReduce accelerator by accessing specific registers of the accelerator. After the end of the Map tasks, the accelerator has already accumulated the values for all the keys. The processors retrieve the final value of a key by sending only the key to the accelerator and reading the final value from the register. In this way, the proposed architecture can accelerate the MapReduce processing by sending a nonblocking transaction to the shared bus that includes the key/value pair that needs updating.

PROGRAMMING FRAMEWORK

Figure 2 shows the programming framework of the MapReduce applications using the hardware accelerator. In the original code, the Map stage emits the key/value pairs and the Reduce stage searches for this key and updates (accumulates) the new value by consuming several CPU clock cycles. By contrast, using the MapReduce accelerator, the Map stage

The hash function can accelerate the indexing of the keys, but it may create a collision if two keys have the same hash value. We selected cuckoo hashing as the best way to resolve hash collisions.

just emits the key/value pair; the MapReduce accelerator merges all the key/value pairs and updates the relevant entries, thus eliminating the Reduce function.

Communication from the application level running under Linux to the hardware accelerator takes place using the memory map (mmap) system call. The mmap system call is used to map a specified kernel memory area to the user layer, so that the user can read or write on it depending on the attribute provided during the memory mapping.

We use a control unit to access these registers and serialize the updates of the key/value elements. The key/value pairs are stored in a memory unit that you can configure based on the application requirements. The memory block contains the key, the value and some bits that are used as tags. These tags indicate if the memory line is empty and whether it is valid. To accelerate the indexing of the keys, a hash module translates the initial key to the address of the memory block.

In the current configuration, we have designed the memory structure to host 2K key/value pairs. Each key can be 64 bits long (eight characters) and the value can be 32 bits long. The total size of the memory structure is 2K x 104 bits. The first 64 bits store the key in order to compare whether we have a hit or a miss using the hash function. The next 8 bits are used for tags and the next 32 bits store the value. In the current configuration, the maximum value of a key is 64 bits and a hash function is used to map the key (64 bits) into the memory address (12 bits).

```
Original Code:
Map{
   Emit_Intermediate(key, value);
}
Reduce(key, value){
  search(key);
  update(key, value);
}

Accelerator code:
Map{
   Emit_Intermediate_Accel(key,value);
}
…
Emit_Intermediate_Accel(key,value)
{
   mmapped_addr = mmap(MapReduce_Accel);
   send(mmapped_addr + 0x4, key);
   send(mmapped_addr + 0x8, value);
}
```

Figure 2 – The programming framework

CUCKOO HASHING

The hash function can accelerate the indexing of the keys but it may create a collision in case two different keys have the same hash value. To address this problem, we selected cuckoo hashing as the best way to resolve hash collisions. Cuckoo hashing [2] uses two hash functions instead of only one. When a new entry is inserted, then it is stored in the location of the first hash key. If that location is occupied, the old entry is moved to its second hash address and the procedure is repeated until an empty slot is found. This algorithm provides a constant lookup time O(1) (lookup requires just inspection of two locations in the hash table), while the insert time depends on the cache size O(n). If the procedure should enter an infinite loop, the hash table will be rebuilt.

The cuckoo hashing algorithm can be implemented using two tables, T1 and T2, for each hash function, each of size r. For each of these tables, a different hash function is used, h1 and h2 respectively, to create the addresses of T1 and T2. Every element x is stored either in T1 or in T2 using hash function h1 or h2 respectively—that is, T1[h1(x)] or T2[h2(x)]. Lookups are therefore straightforward. For each of the element x that we need to look up, we just check the two possible locations in tables T1 and T2 using the hash functions h1 and h2, respectively.

To insert an element x, we check to see if T1[h1(x)] is empty. If it is empty, then we store the element in this loca-

Resources	Number	Percentage
Slice Registers	843	< 1%
Slice LUTs	903	< 1%
Block RAMs	29	21%

Table 1 – Programmable logic resource allocation

tion. If not, we replace the element y that is already there in T1[h1(x)] with x. We then check whether T2[h2(y)] is empty. If it is empty, we store the element in this location. If not, we replace the element z in T2[h2(y)] with y. We then try to place z in T1[h1(z)], and so on, until we find an empty location.

According to the original cuckoo hashing paper [2], if an empty location is not found within a certain number of tries, the suggested solution is to rehash all of the elements in the table. In the current implementation of our software, whenever the operation enters such a loop, it stops and returns zero to the function call. The function call then may initiate a rehashing or it may choose to add the specific key in the software memory structure as in the original code.

We implemented cuckoo hashing for the MapReduce accelerators as depicted in Figure 1. We used two Block RAMs to store the entries for the two tables, T1 and T2. These BRAMs store the key, the value and the tags. In the tag field, one bit is used to indicate whether a specific row is valid or not. Two hash functions are used based on simple XOR functions that map the key to an address for the BRAMs. Every time an access is required to the BRAMs, the hash tables are used to create the address and then two comparators indicate whether there is a hit on the BRAMs (i.e., that the key is the same as the key in the RAM and the valid bit is 1). A control unit coordinates access to the memories. We implemented the control unit as a finite state machine (FSM) that executes the cuckoo hashing.

PERFORMANCE EVALUATION

We have implemented the proposed architecture in a Zynq SoC. Specifically, we mapped the Phoenix MapReduce framework to the embedded ARM cores under Linux 3. Whenever the processors need to update the key/value pairs, they send the information through specific function calls. For the performance evaluation of the system, we used three applications from the Phoenix framework, modified to run utilizing the hardware accelerator: WordCount, Linear Regression and Histogram.

The proposed scheme is configurable and it can be tuned based on the application requirements. For the performance evaluation of the Phoenix MapReduce framework application, we have configured the accelerator to include a 4K memory unit (4,096 key/value pairs can be stored: 2K in each BRAM). The maximum size of each key is 8 bytes.

Table 1 shows the programmable logic resources of the MapReduce accelerator. As you can see, the accelerator is basically memory-intensive while the control unit that is used for the finite state machine and the hash function occupy only a small portion of the device.

Figure 3 compares the execution time of the original applications and the execution time of the applications using the MapReduce accelerator. Both of these measurements are based on the Xilinx Zynq SoC design.

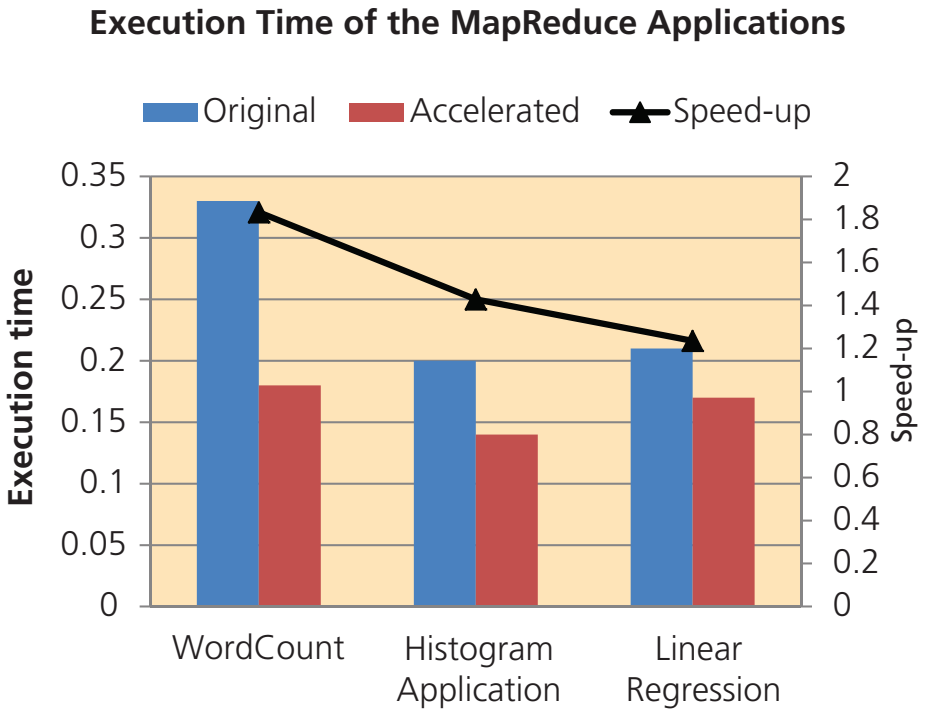


Figure 3 – The overall system speed-up was 1.23x to 1.8x, depending on the application.

In the case of the WordCount, in the original application the Map task identifies the words and forwards them to the Reduce task. This task gathers all the key/value pairs and accumulates the value for each key. In the accelerator case, the Map task identifies the words and forwards the data to the MapReduce accelerator unit through the high-performance AXI bus. The key/value pairs are stored in the registers (which are different for each processor), and then the accelerator accumulates the values for each key by accessing the memory structure.

PROCESSOR OFFLOADED

The reduction in execution time is due to the fact that in the original code, the Reduce task has to first load the key/value table, then search through the table for the required key. Then, after the accumulation of the value, the Reduce task must store the key back on the memory. By utilizing the MapReduce accelerator, we offload the processor from this task and thus reduce the total execution time of the MapReduce applications. Cuckoo hashing (O(1)) keeps the searching time of the key in the accelerator, while the processor is not blocked during the update of the key/value pair.

As Figure 3 shows, the overall speed-up of the system is from 1.23x to 1.8x. The speed-up depends on the characteristics of each application. In cases where the mapping functions are more complex, the MapReduce accelerator provides a lesser speed-up. In applications with simpler mapping functions that are allocated less of the overall execution time, the speed-up is greater, since a high portion of the total execution time is used for communication between the Map and the Reduce functions. Therefore, in these cases the MapReduce accelerator can provide much more acceleration. Furthermore, the MapReduce accelerator results in the creation of fewer new threads in the processors, which translates into less context switches and therefore reduced execution time. For example, in the case of WordCount, the average number of context switches dropped from 88 to 60.

The MapReduce framework can be widely used as a programming framework both for multicore SoCs and for cloud-computing applications. Our proposed hardware accelerator can be used to reduce the total execution time for the multicore SoC platforms such as the Xilinx Zynq SoC and the cloud-computing applications based on the MapReduce framework by accelerating the Reduce task of these applications. For more information on accelerating cloud computing with the Zynq SoC platform, contact the lead author, Dr. Christoforos Kachris, or visit *www.green-center.weebly.com*.

References

1. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Commununications of the ACM, vol. 51, no. 1, pp. 107–113, January 2008

2. R. Pagh and F. F. Rodler, "Cuckoo Hashing," Proceedings of ESA 2001, Lecture Notes in Computer Science, vol. 2161, 2001

3. C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski and C. Kozyrakis, "Evaluating MapReduce for Multicore and Multiprocessor Systems," Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, HPCA '07, 2007, pp. 13–24

Acknowledgments

The authors would like to thank the Xilinx University Program for the kind donation of the Xilinx EDA tools. The research project is implemented within the framework of the action "Supporting Postdoctoral Researchers" of the operational program "Education and Lifelong Learning" (Action's Beneficiary: GSRT) and is co-financed by the European Social Fund (ESF) and the Greek State.

Zynq SoC Forms Foundation for LTE Small Cell Basestations

Femtocell, picocell and other small cell designs can attain new levels of integration, flexibility and low power by using Xilinx's Zynq-7000 All Programmable SoC.

by Wai Shun Wong
Senior Product Marketing Manager
Xilinx, Inc.
waishun@xilinx.com

George Wang
Staff DSP Specialist
Xilinx, Inc.
hwang@xilinx.com

Small cells are low-power wireless basestations that operate in a licensed spectrum and are managed by mobile-network operators. Types of small cells range from femtocells (which cover less than 100 meters) to metrocells (which cover up to a few hundred meters), in between which are picocells and microcells. Today, the electronic brains of these small cell systems consist of a combination of FPGAs, DSP devices, and network and host processors that conform to wireless standards, which are continuously evolving. But the next-generation small cell systems are fast becoming highly integrated by incorporating those elements into a system-on-chip (SoC).

Carriers can use small cells to improve cellular coverage and capacity for homes and enterprises as well as metropolitan and rural public spaces. More and more customers want to use mobile phones wherever they are, at home or at work, even when there is a fixed line available. They also expect ubiquitous coverage on the go and innovative mobile data services with sufficient bandwidth to enjoy them.

A vast majority and growing amount of data usage occurs indoors, but macrocell coverage can be very poor indoors. Thus, small cells can play a big role in complementing both coverage and capacity needs. Small cells can also improve the quality of experience for end users by ensuring fewer dropped calls and better data rates. Meanwhile, they also reduce operator capital and operating expenditures, lowering site rental, maintenance and power costs.

The Xilinx® Zynq®-7000 All Programmable SoC combines the software programmability of a processor with the hardware programmability of an FPGA, resulting in unrivaled levels of system performance, flexibility and scalability along with significant system benefits in terms of power reduction, lower cost and fast time-to-market. Because the Zynq SoC is hardware programmable as well as software programmable, it offers customers an unmatched degree of programmability and differentiation compared with traditional SoC processing solutions. Standard SoCs are only software programmable, vastly limiting design flexibility and differentiation, and curbing their ability to effectively adapt to evolving wireless standards.

ZYNQ SOC IS IDEAL FOR LTE SMALL CELLS

The Zynq SoC's high-performance programmable logic and ARM processors allow customers to create designs that accommodate the ever-increasing transmission bandwidths seen in current and next-generation wireless equipment. Design teams can

Most small cells come with thermal requirements, as many must run without cooling fans. The Zynq SoC is very low in power, which not only eliminates the need for a fan but also makes expensive heat sinks unnecessary.

implement their protocol stacks in the Zynq SoC's processing system and see how they run. If it is too slow, they can use Vivado® Design Suite high-level synthesis (HLS) to translate the code into Verilog or VHDL to run in the Zynq SoC's programmable logic. Doing so can speed some functions by as much as 700x while freeing up the processor to run other tasks faster—thus increasing overall system performance.

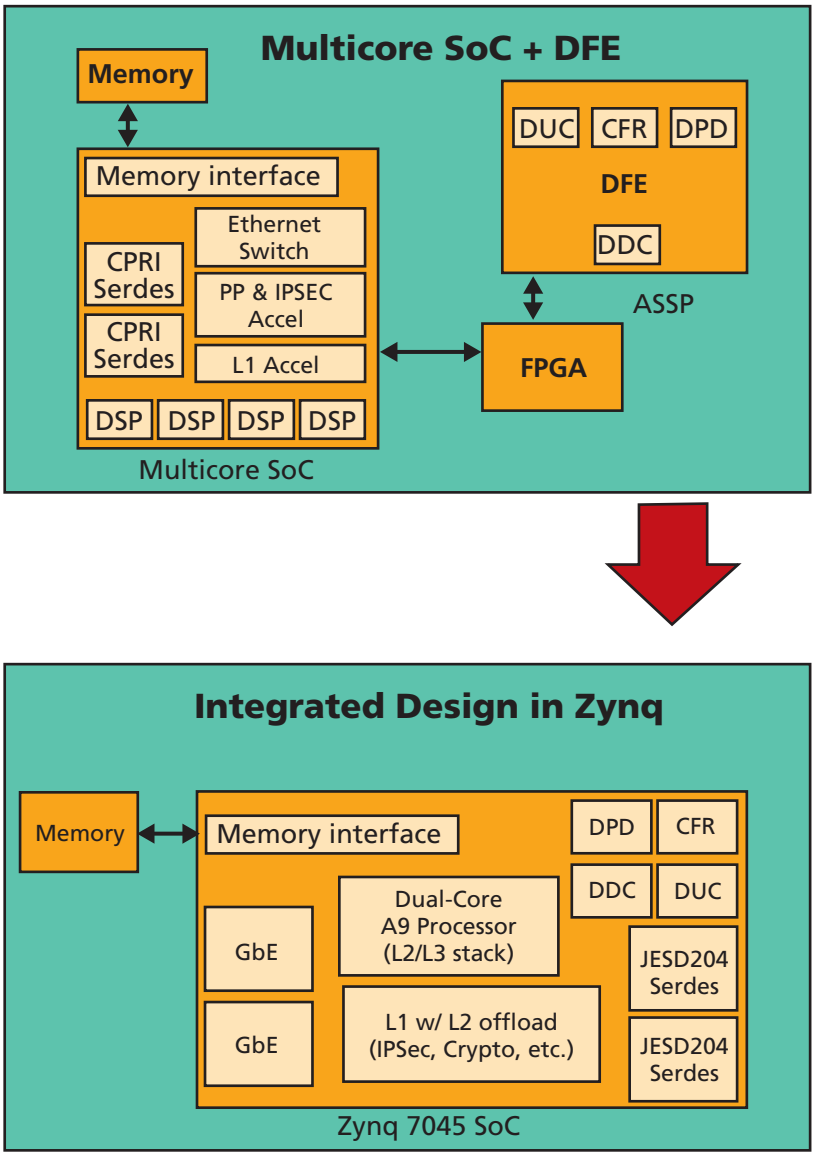


Figure 1 – A small cell basestation design that formerly took three chips (a multicore SoC, an ASSP and a small FPGA; top diagram) can be done in a single Zynq-7000 SoC.

Most small cells come with thermal requirements, as many must run without cooling fans. The Zynq SoC is very low power, which not only eliminates the need for a fan but also makes expensive heat sinks unnecessary. Xilinx also offers radio IP for power amplifier efficiency; this IP can help design teams further reduce the power their small cell implementations consume.

To meet the needs of heterogeneous networks (HetNets) connecting a broad number of computers and devices that operate with different OSes and protocols, Xilinx offers a compelling portfolio of All Programmable devices. These products offer maximum design flexibility and can solve HetNet multistandard and multiband needs with a single reconfigurable device. Xilinx silicon allows customers to choose different devices depending on whether they are designing a small cell or conventional macrocell architecture. Xilinx's Zynq SoC devices are particularly well suited for small cell HetNet systems, as they offer even greater degrees of system integration than any other device. The dual ARM® Cortex™-A9 processors provide the power needed for real-time processing tasks. One Cortex-A9 can run the real-time operating system and take care of the L1 control, while the other Cortex-A9 can run part (or all) of the higher-layer functions.

The heavy-lifting user-plane processing is best done in hardware. Coupled with Xilinx's extensive, optimized suite of LTE baseband functions, the embedded 7 series FPGA

fabric provides superior real-time performance at reduced power dissipation compared with multicore SoCs. The fabric can also be used to accelerate L2 user-plane functions, offloading the ARM processors and thus giving designers the flexibility to accelerate hardware in the fabric for improving higher-layer performance. The fabric also allows the integration of digital front-end (DFE) functions including digital up- and downconversion (DUC/DDC) and optional crest-factor reduction (CFR) and digital predistortion (DPD). This integration further reduces power dissipation and helps to reduce overall end-to-end latency, which is crucial for LTE basestation design.

FLEXIBLE FOR SMALL CELLS

Xilinx offers a Zynq SoC-based LTE small cell reference design that can be supplied with a Xilinx LTE baseband. Alternatively, customers can integrate their own LTE baseband if they so choose. Xilinx also offers comprehensive software solutions including LTE protocol stacks, RAN security, backhaul IPSec, timing and synchronization, and Robust Header Compression (RoHC).

Support for PCIe® makes it easy to integrate Wi-Fi in your LTE small cell SoC design. The high-performance processing capability and I/O also suit the Zynq SoC to high-performance small cell design with integrated wireless backhaul. Moreover, support for JESD204B enables an LTE small cell SoC to easily connect with any industry-leading RF transceivers.

For low- and medium-capacity small cells, the Zynq SoC enables a highly integrated solution with integrated PHY: radio plus L1 baseband implemented in programmable logic (PL), along with higher-layer protocol stacks, backhaul transport management, timing and control for the PHY implemented in the processing system (PS). The PL also provides hardware acceleration for some higher-layer functionality in order to maximize capacity (potentially a key differentiator over competing offerings). The Zynq SoC family can potentially address several deployment scenarios with a single-chip implementation.

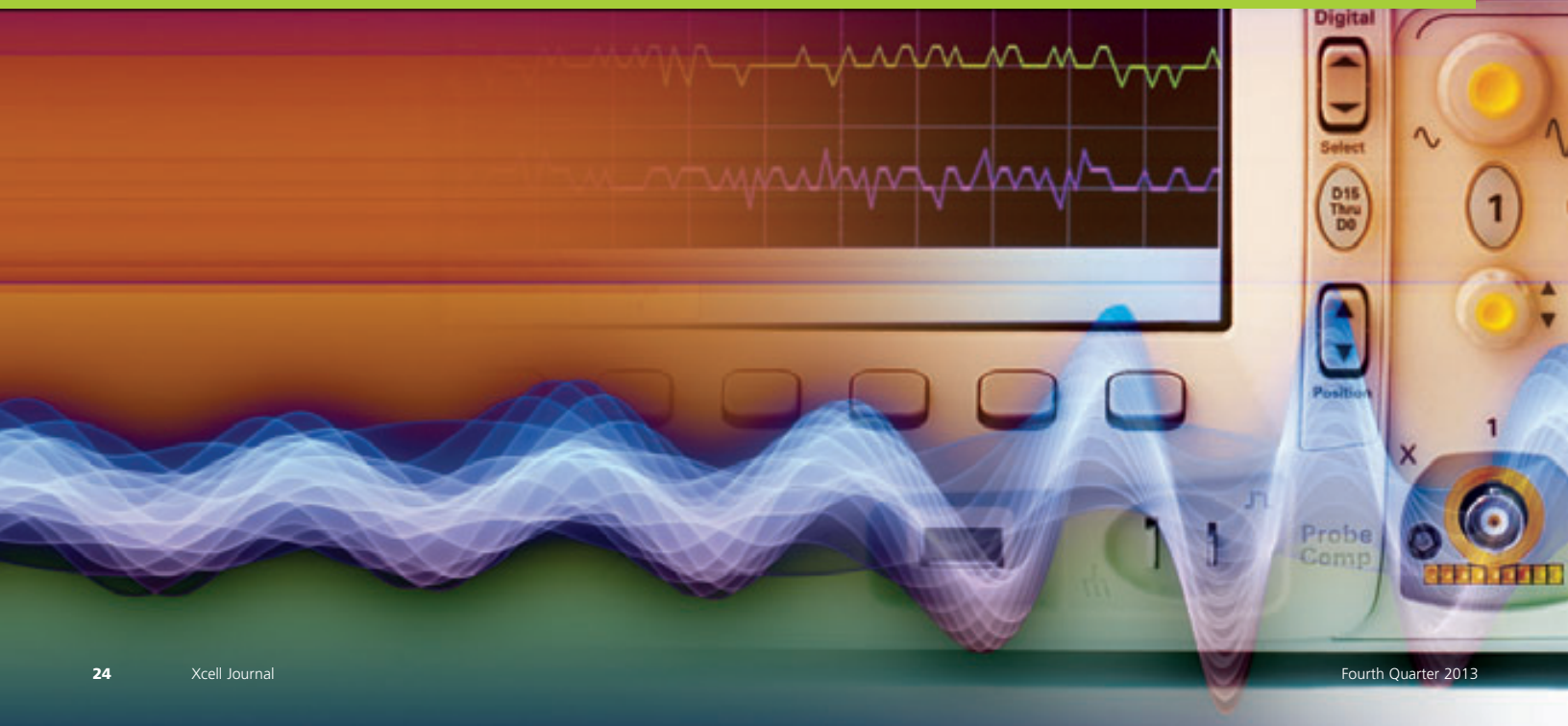
For very high-capacity small cells, given the processing capability of the current Zynq SoC series, Xilinx proposes implementing the radio plus L1 baseband in PL, and associated timing and control in PS. A companion processor could handle the higher-layer protocol stacks and backhaul transport management.

The top diagram in Figure 1 shows a typical 2x2 (2T2R) LTE small cell design. A typical bill of materials (BOM) for such a design includes three chips: a multicore SoC, an ASSP (for the DFE) and a small FPGA for the remaining logic and memory. In the bottom diagram, everything fits into a single Zynq SoC device (such as the Z-7045), with the dual ARM Cortex-A9 running the LTE L2/L3 stack. LTE L1 is integrated on the FPGA fabric, with hardware acceleration for L2 offloading such as IPSec, crypto and RoHC. The FPGA fabric also contains the digital front end (DPD, DUC, DDC, CFR). Having a single-chip design reduces chip-to-chip latencies for optimized packet processing. The BOM cost can be reduced 25 percent while doubling system performance as well as achieving a reduction of 35 percent in total power dissipation.

LTE small cell basestation design demands low cost, low power dissipation, high integration and high reliability. Full integration, scalability and flexibility are very important to achieve these goals. With its dual-core ARM processor subsystem, high system performance and low-power programmable logic, the Zynq-7000 All Programmable SoC is a cost-effective solution to meet current and future small cell requirements.

Zynq SoC Enables Red Pitaya Open-Source Instruments

The new Red Pitaya initiative makes it easy to develop many instrumentation variants inexpensively from one platform.

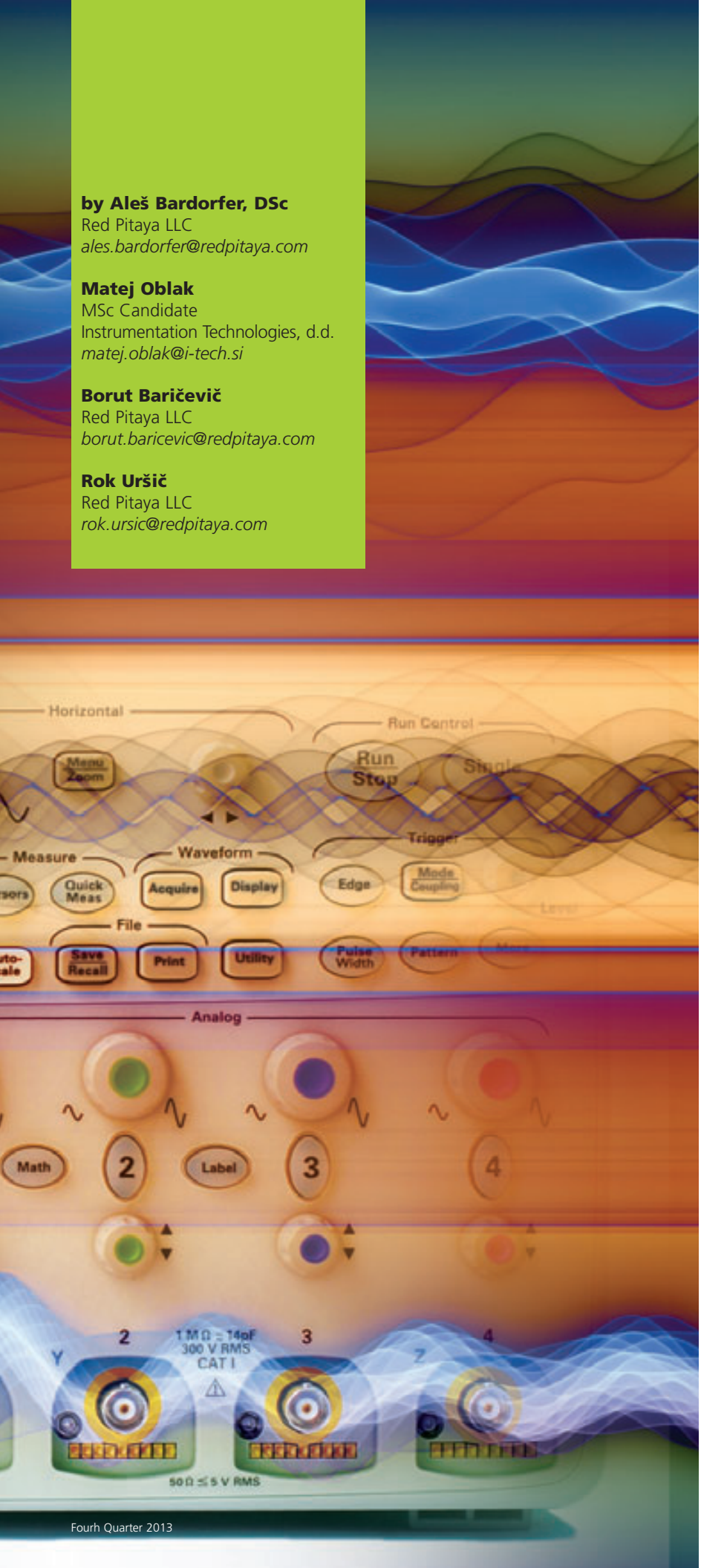


by Aleš Bardorfer, DSc
Red Pitaya LLC
ales.bardorfer@redpitaya.com

Matej Oblak
MSc Candidate
Instrumentation Technologies, d.d.
matej.oblak@i-tech.si

Borut Baričevič
Red Pitaya LLC
borut.baricevic@redpitaya.com

Rok Uršič
Red Pitaya LLC
rok.ursic@redpitaya.com



In the early days of the electronics industry, test-and-measurement equipment such as oscilloscopes, signal generators and the like came with fixed functionality and performed a small and rigid set of tasks. Over the last few decades, manufacturers have leveraged the programmability of DSPs to offer broader feature sets tailored for specific applications. However, the ability to program these instruments and determine their final feature set was restricted to whatever the manufacturers did before the products left the factory.

Even with broader feature sets, most instruments on the market today don't have the precise functionality that customers may require for their specific projects. This is especially true for interdisciplinary research groups at institutes and universities worldwide, which focus on building large, complex systems based on smaller subsystems, with instruments being one of the latter. In such cases, it is often essential to use the instruments at much finer levels than the vendors who designed the equipment enabled. Typically, integration problems arise in fast-feedback systems involving two or more instrument subsystems that require a hard-real-time response with very little jitter. Often, the interfaces to the instruments prove inadequate to achieve the required overall integrated system performance. Closed-source, DSP-oriented instruments prevent such access.

Our engineers at Instrumentation Technologies were wondering what would happen if we created instrumentation—or let us say a signal-processing platform—that was user-programmable, user-customizable and open source. To this end, we developed a compact board called Red Pitaya, which is backed by an open-source development ecosystem. The Red Pitaya system has the potential to replace many of the standard test-and-measurement instruments in a single reconfigurable device, at a significantly lower price. Figure 1 illustrates the system architecture of Red Pitaya, a modern instrument and signal-processing system with multiple analog and digital inputs and outputs.

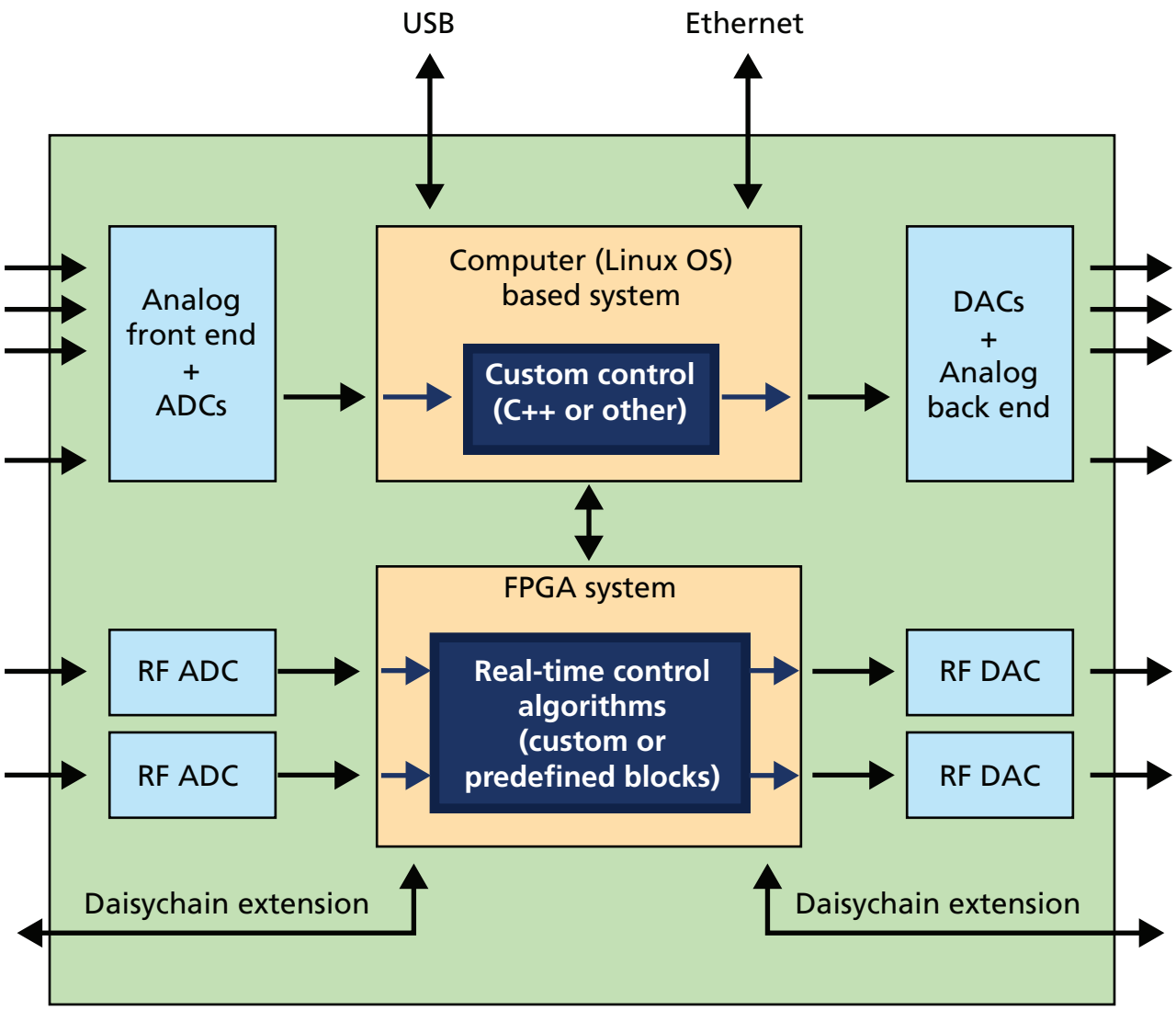


Figure 1 – The Red Pitaya system architecture

SYSTEM ARCHITECTURE

Good instrumentation in general requires excellence in two domains. The analog domain, which is moving to ever higher frequencies, still dictates the noise floor and measurement performance of an instrument. But it is the digital domain that is gaining momentum due to its stability, noise immunity, time invariance and ability to perform very complex processing, combined with standard connectivity. Thus, the signal processing is increasingly moving from the analog world to the digital.

The basic approach in modern signal-processing systems is to be as generic as possible at the analog front-end electronics, and to sample the signals with fast ADCs as soon as possible in the processing chain. On the digital side, the trend is to process the signals and send them to the DACs as late as possible in the processing chain, and again to be as generic as possible in terms of the analog back-end electronics. With such an architecture, all the specifics of the processing/measurement system are concentrated in the digital domain, while the analog electronics are kept simple and generic. This mostly digital modern architecture creates a great opportunity for customization of hardware to perform a number of tasks across several applications spaces. The possibilities are limited only by the bandwidth of the analog front and back ends, and the computational resources of the digital domain—primarily the FPGA and the CPU resources.

There are two main types of processing chains present on Red Pitaya in terms of speed. The first type is the ~50-MHz bandwidth signal-processing chain achieved by leveraging the extremely fast and low-jitter hard-real-time processing capabilities of FPGAs. The other Red Pitaya processing chain, at a bandwidth of ~50 kHz, is achieved through the CPU, possibly running a hard-real-time operating system.

The reduction of data from an instrument's inputs to its meaningful outputs is the main job of the digital signal processing embedded in the instrument.

Based on our experience, making an FPGA and CPU the basic blocks of a signal-processing system has always been a winning combination in reconfigurable instrumentation. This combination allows for a lot of freedom in partitioning the parts of signal processing between the high-performance FPGA and the easily programmable CPU. In particular, almost any instrument faces the problem of reducing huge amounts of input data from the raw sampling point to the point of the instrument's outputs, such as a scope diagram holding 1,000 points only, for example.

Typical examples of measurement results include a signal plot of an oscilloscope, a frequency-domain plot of a spectrum analyzer or a stream of position coordinates of a basketball player, being tracked by processing the output of a ceiling-mounted camera. All these measurement outputs represent a lot less data compared with the raw instrument input signals, the latter being sampled at very high frequency. This reduction of data from the instrument's inputs to its meaningful outputs is therefore the main job of the digital signal processing embedded in the instrument.

Having both the FPGA and the CPU available on the signal-processing system enables the developer to freely decide which parts of DSP processing to implement on the FPGA and which parts on the CPU. There are slight differences between them in terms of processing suitability, but both can perform digital signal processing. In general, the FPGA handles ultrafast, yet simple DSP operations, but is a bit less suitable for complex procedural operations. The CPU, on the other hand, excels at slower, but arbitrarily complex procedural operations; CPUs are also

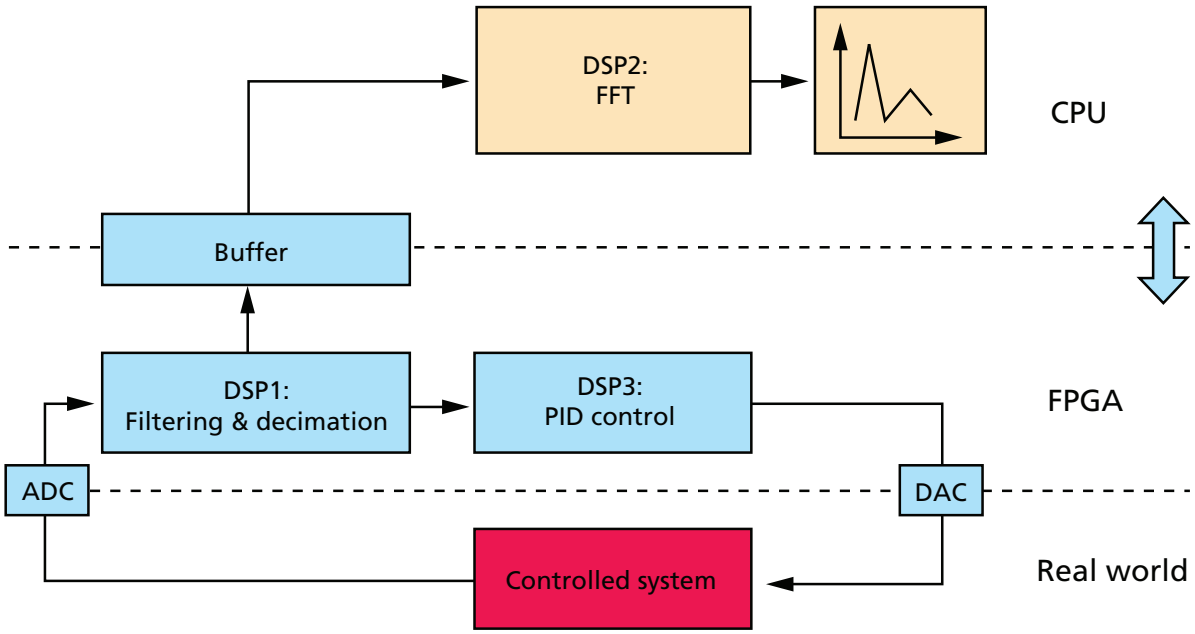


Figure 2 – DSP algorithm partitioning between the FPGA and CPU

good at running standard interactive interfaces, such as Web servers, for example. Despite the large improvements in FPGA development tools in recent years, it is still generally easier to write procedural software to run on the CPU, compared with RTL coding and synthesis of digital structures in the FPGA.

The freedom of partitioning the DSP processing between the FPGA and the CPU brings another advantage—namely, the ability to rapidly prototype a performance-limited, but fully functional system. Implementing most of the DSP on the CPU allows prototype demonstrations at early stages of the development project, suitable for marketing, while at its later stages, this strategy allows for a smooth transition of the performance-critical part of the DSP to the FPGA for a final product with the same functionality, but full performance.

Figure 2 shows an example of partitioning the DSP algorithms between the FPGA and the CPU. During the development cycle, the border between the two can vary. Since the DSP processing is split, the partially processed data must be transferred between the FPGA and the CPU or vice versa. The bus speed between them is therefore very important in order to keep from introducing additional processing latency.

RED PITAYA

The Red Pitaya system, as shown in Figure 3, is based on the Xilinx® Zynq®-7010 All Programmable SoC. Red Pitaya boasts fast two-channel, 125-Msample/second signal-acquisition and signal-generation capabilities, which can be combined with FPGA-based DSP processing in between to form hard-real-time feedback loops. In addition to fast signal processing, the system includes several slower (~100-kHz) I/O channels, leveraging Xilinx Analog Mixed Signal (AMS) technology (Figure 4), along with several digi-

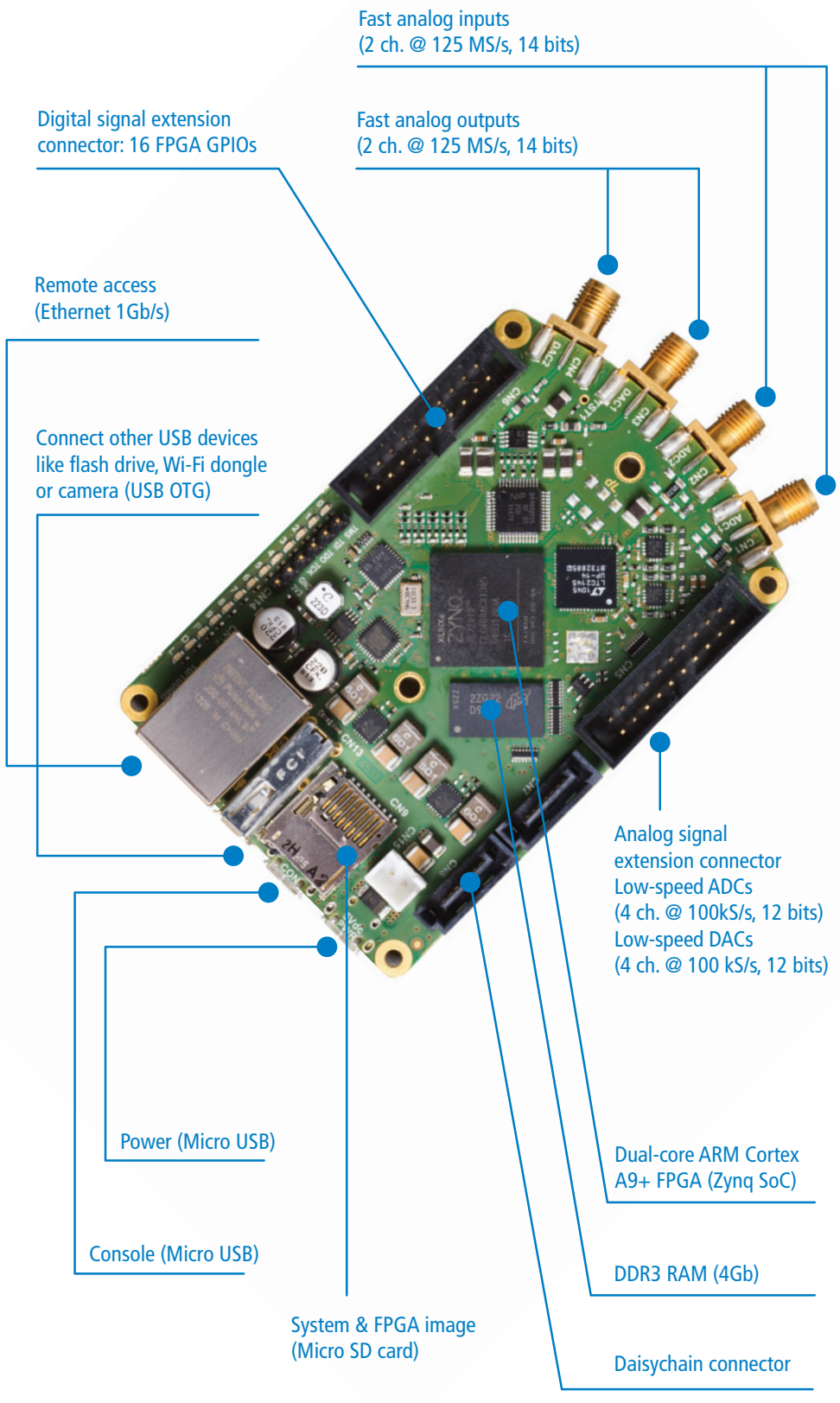


Figure 3 – The versatile Red Pitaya instrument platform

tal I/Os. Distributed processing is possible if you daisychain several Red Pitaya modules via fast serial daisychain connectors. In this way, you can build a complex system, requiring more inputs and outputs, using several Red Pitaya subsystems interconnected with one another. The CPU runs the Linux operating system and supports the standard peripherals, such as 1000Base-T Ethernet, USB OTG, Micro SD storage and USB serial console.

Initial applications include a two-channel, 125-Msample/second oscilloscope, a spectrum analyzer, a signal generator and a PID controller. The Red Pitaya instrument with applications and ecosystem will launch in December 2013. The beauty of being open-source is that motivated enthusiasts can develop custom applications to fit their specific needs. As with every open-source customization, chances are that others will be able to use the modified or new instrument or application as well. Additional applications are expected to evolve over time within the Red Pitaya ecosystem.

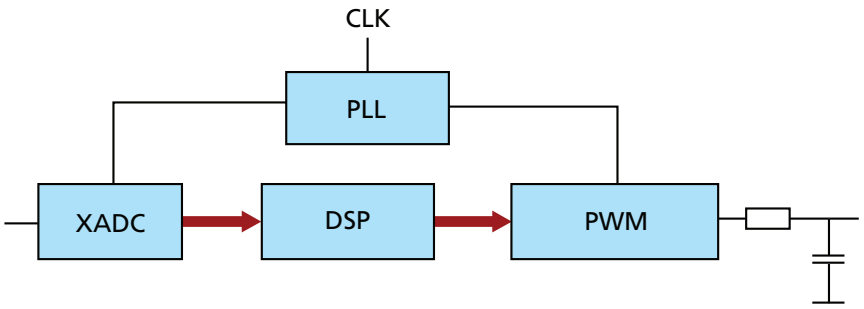
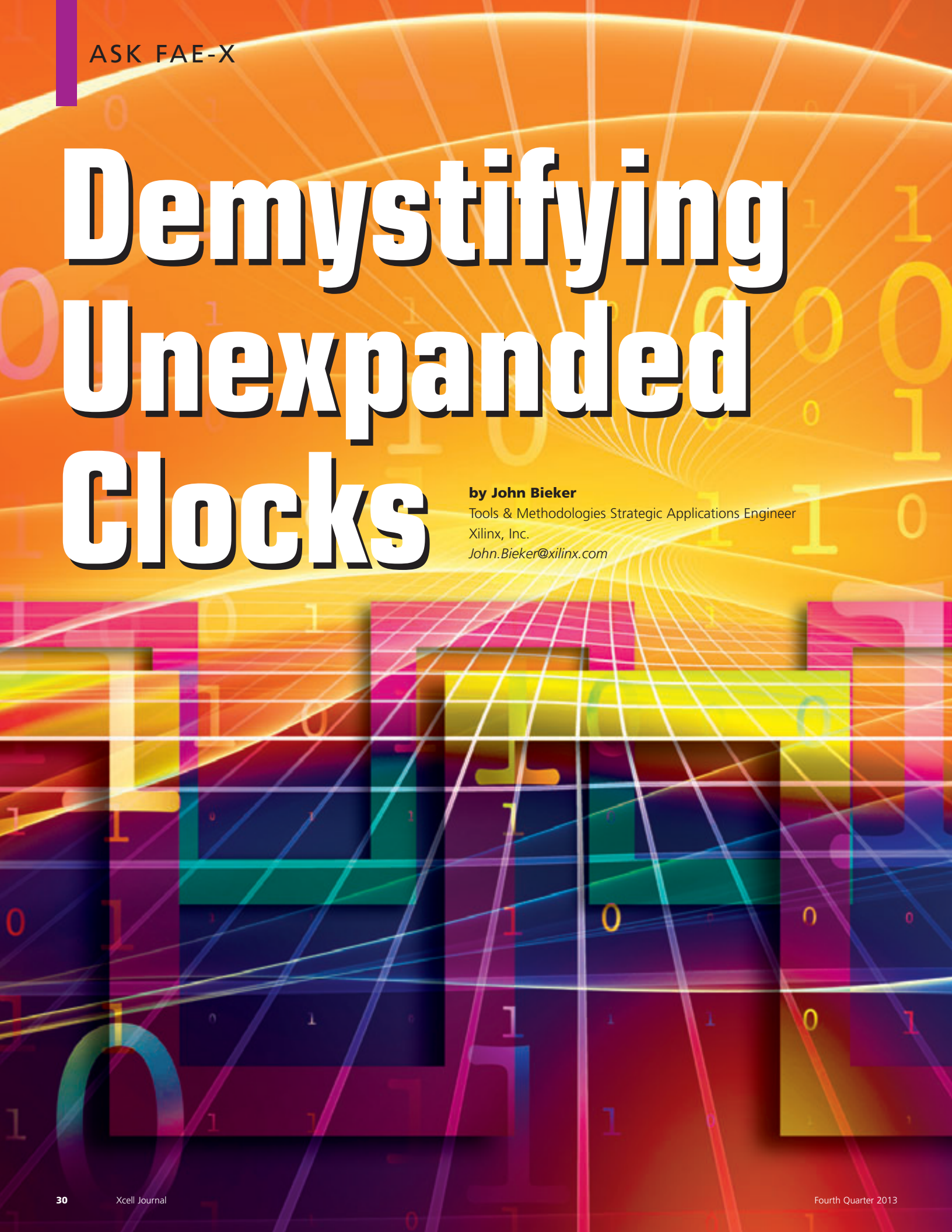


Figure 4 – Xilinx AMS technology (XADC) combined with PWM-based outputs

Demystifying Unexpanded Clocks

by John Bieker
Tools & Methodologies Strategic Applications Engineer
Xilinx, Inc.
John.Bieker@xilinx.com



Clock expansion becomes an issue for engineers using Xilinx's Vivado Design Suite, but it need not be a stumbling block.

As more and more Xilinx® users embrace the Vivado® Design Suite, some have expressed confusion over unexpanded clocks. What are they? How are they related and timed? What is the prescribed method for dealing with unexpanded clocks? Let's take a detailed look at the issue, focusing on ways that designers can ensure they have used proper synchronization techniques to safely traverse clock domain crossings between unexpanded clocks.

With the advent of the Vivado tools, Xilinx introduced support for timing constraints in the industry-standard Synopsys Design Constraints (SDC) format. This was a significant shift from the way the ISE® Design Suite tools handled timing. The most fundamental difference in the Vivado suite is that all clocks are related by default. As a result, timing paths that were unconstrained in ISE are constrained in Vivado Design Suite. This constraint can cause very tight timing requirements when two clocks do not have a periodic relationship.

Unexpanded clocks are clocks that do not exhibit a periodic relationship with one another within 1,000 cycles. The Vivado BFT Core example design offers a good way to illustrate how unexpanded clocks work.

LAUNCH AND CAPTURE

Timing paths are initiated by a launching event and terminated by a capture event. The launch event occurs when the clock of a synchronous cell transitions from the inactive state to the active state. The capture event occurs when the clock of the downstream, or capturing, synchronous cell transitions from the inactive state to the active state. The time from the launch event to the capture event is called the "path requirement," because it represents the required time for data to be valid at the pin of the capture cell after the launch event.

When the launch and capture clocks are the same physical clock net, the path requirement is simply the period of the clock. For example, in the case of a 250-MHz clock, there are precisely 4 nanoseconds between the launch event and the capture event, assuming that the active edges of the launch and capture cells are the same (rising edge to rising edge or falling edge to falling edge). When the launch and capture elements have different active edges, the path must be timed at half the frequency of the clock. This is the case for rising-edge-to-falling-edge or falling-edge-to-rising-edge transitions of the clock and corresponds to a 2-ns path requirement for a 250-MHz clock.

When the source and destination clocks are not the same, the timing picture becomes more complex. Timing paths that have different source and destination clocks are called clock-domain crossing, or CDC, paths.

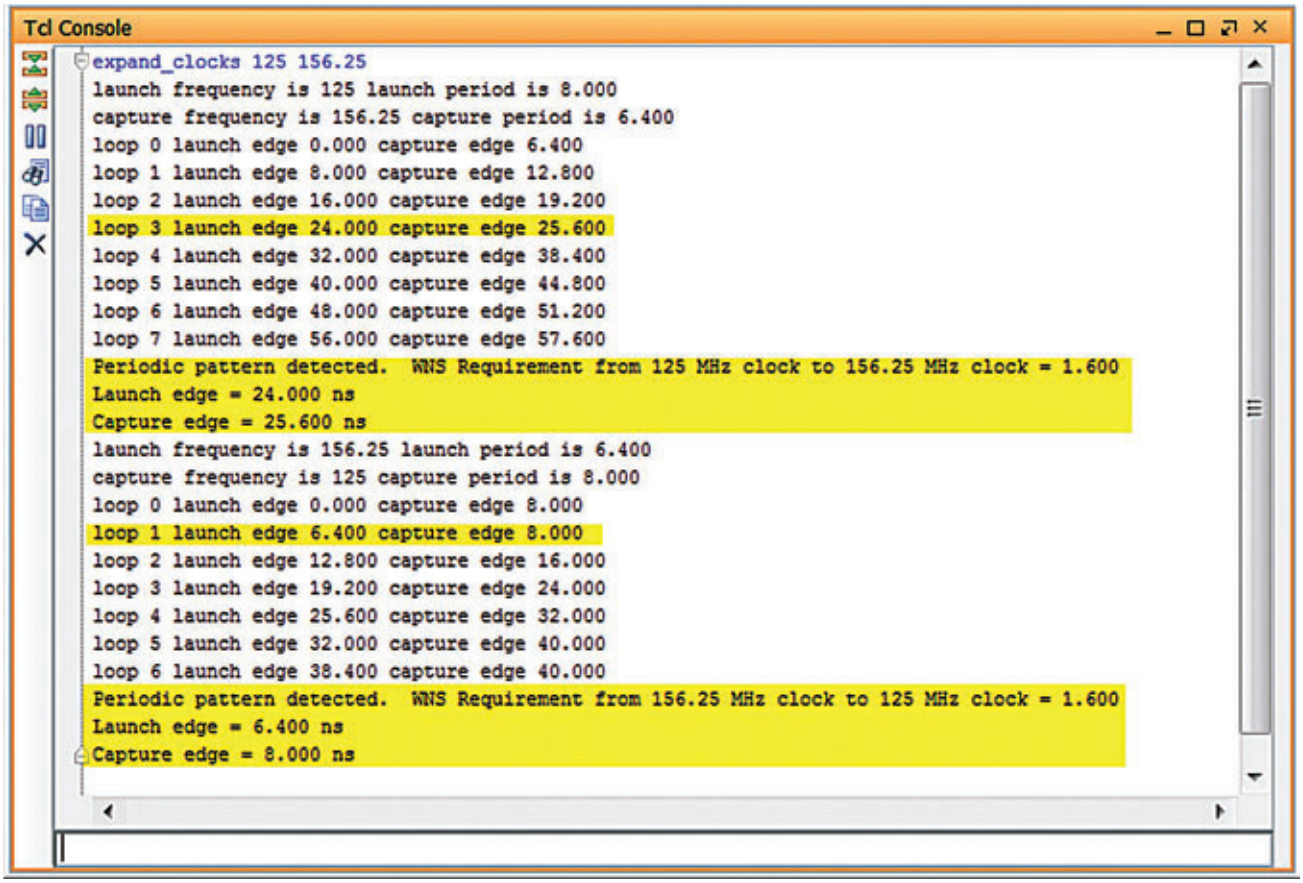


Figure 1 – Tcl script for expanded clocks

When the phase relationship between the launch and capture clocks is not known, it's impossible to calculate the path requirement. Timing such paths is unsafe precisely because no known phase relationship is defined. Unsafe clocking is highlighted in the clock interaction report that Vivado Design Suite produces. When the phase relationship between the launch and capture clocks is known, the path requirement can be deduced mathematically so long as the two clocks have a periodic relationship. The process used to determine the minimum path requirement for a given launch and capture clock is called clock expansion. The search for a periodic relationship between two clocks, however, needs to be bounded by practical limits because a common, periodic pattern between two different clocks does not always exist. The practical limit that the design suite uses is 1,000 clock cycles, measured by the faster of the two clocks. If, after expanding the clocks out 1,000 cycles, no periodic relationship is found, the two clocks are called "unexpanded."

The path requirement that Vivado Design Suite uses between two unexpanded clocks is the smallest amount of time between adjacent active edges of the launch and capture clocks. Note, however, that a tighter path requirement may exist beyond 1,000 cycles, which is why the Vivado Design Suite calls the relationship between the two clocks unexpanded.

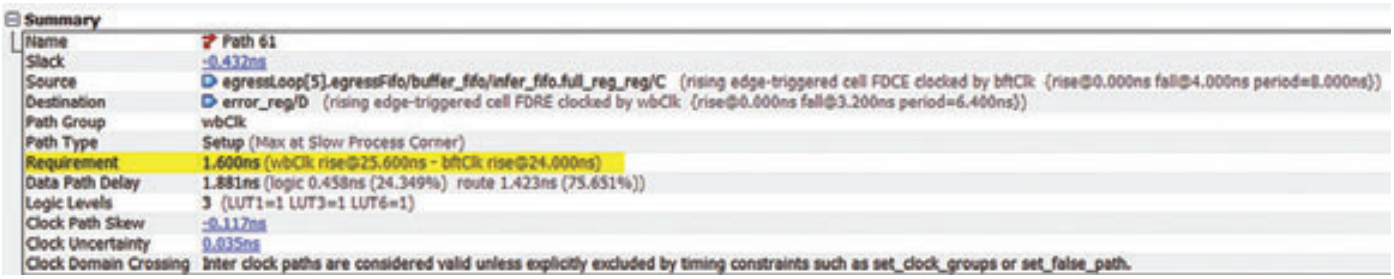


Figure 2 – Clock expansion of 125-MHz and 156.25-MHz clocks

The Vivado BFT Core provides the ideal setting to test different clock rates to determine whether two clocks are expandable or not.

To better understand how the software calculates unexpanded clocks, the Tcl script provided in Figure 1 shows the expansion details for the rising-edge-to-rising-edge path requirement for any two clock rates. The output of the script is provided in Figure 2 for the expansion of two clocks running at 125 MHz and 156.25 MHz.

To confirm the results given by the Tcl script, let's use the Vivado BFT Core example design. The BFT Core provides the ideal setting to test different clock rates to determine whether two clocks are expandable or not. This core has two independent clock sources named wbClk and bftClk. The timing constraints set the frequencies of wbClk and bftClk to 100 MHz and 200 MHz, respectively. You can change these constraints to any two clock rates in order to discover the expansion relationship determined by the Vivado timing engine. Figure 3 illustrates the result of setting bftClk to 125 MHz and wbClk to 156.25 MHz, and reporting the associated timing (report_timing -from [get_clocks bftClk] -to [get_clocks wbClk]). Note that the launch edge, capture edge and timing requirement confirm the results that the Tcl script has predicted.

Table 1 provides the launch edge, capture edge and path requirements for several clock rates obtained using the Tcl script in Figure 4.

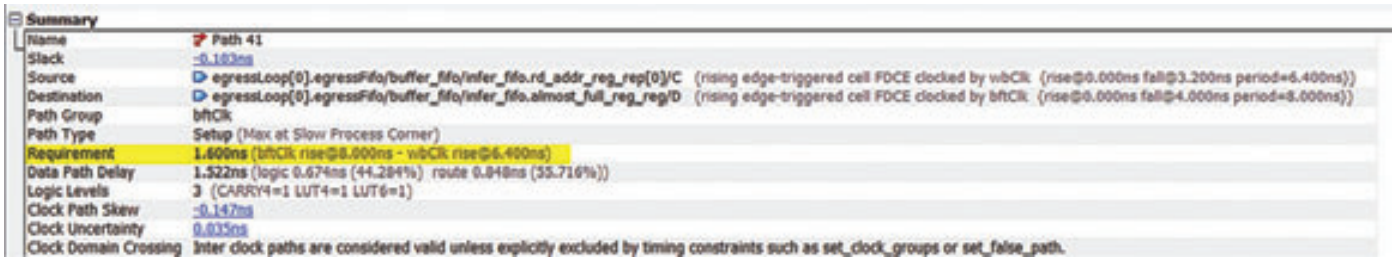


Figure 3 – Clock expansion of 125-MHz launch clock to 156.25-MHz capture clock

Launch Clock		Capture Clock		Launch Clock Edge (ns)	Capture Clock Edge (ns)	Path Requirement (ns)
Frequency (MHz)	Period (ns)	Frequency (MHz)	Period (ns)			
100.000 200.000	10.000 5.000	200.000 100.000	5.000 10.000	0.000 5.000	5.000 10.000	5.000 5.000
100.000 155.520	10.000 6.430	155.520 100.000	6.430 10.000	3260.000 3169.990	3260.010 3170.000	0.010 0.010
156.250 200.000	6.400 5.000	200.000 156.250	5.000 6.400	44.800 115.000	45.000 115.200	0.200 0.200
200.000 250.000	5.000 4.000	250.000 200.000	4.000 5.000	15.000 4.000	16.000 5.000	1.000 1.000
200.000 333.333	5.000 3.000	333.333 200.000	3.000 5.000	5.000 9.000	6.000 10.000	1.000 1.000

Table 1 – Clock expansion examples

```
# ------------------------------------------------------------------------------------------------------
# Target device : Any family
# Description :
#   Determine rising-edge to rising-edge setup path requirement for two clocks
# Assumptions :
#   - Clock frequency specified in MHz.
# ------------------------------------------------------------------------------------------------------
# Calling syntax:
#   source <your_Tcl_script_location>/expand_clocks.tcl
# ------------------------------------------------------------------------------------------------------
# Author  : John Bieker, Xilinx
# Revison : 1.0 - initial release
# ------------------------------------------------------------------------------------------------------

proc expand_clocks {clock_freq1 clock_freq2} {

  get_edge_relationship $clock_freq1 $clock_freq2
  get_edge_relationship $clock_freq2 $clock_freq1

}

proc get_edge_relationship {launch capture} {

  set launch_period  [expr [expr round(1000000.0/$launch)]/1000.0]
  set capture_period [expr [expr round(1000000.0/$capture)]/1000.0]

  puts "launch frequency is $launch launch period is [format %.3f $launch_period]"
  puts "capture frequency is $capture capture period is [format %.3f $capture_period]"
  set closest_edge $capture_period
  set capture_edge $capture_period
  set closest_launch_edge 0.000
  set closest_capture_edge $capture_edge
  set terminal_edge [expr 1000 * [expr min($launch_period, $capture_period)]]
  set i 0
  set exit_for_loop 0
  for {set launch_edge 0.000} {$launch_edge < $terminal_edge} {set launch_edge [expr $launch_edge +
$launch_period]} {
    incr i
    if {[check_edges_aligned $launch_edge $capture_edge]} {
      break
    } else {
      while {[set difference [format %.5f [expr $launch_edge - $capture_edge]]] >= 0.00000} {
        set capture_edge [expr $capture_edge + $capture_period]
        if {[check_edges_aligned $launch_edge $capture_edge]} {
         set exit_for_loop 1
         break
        }
      }
    }
    if {$exit_for_loop == 1} {break}
    puts "loop $i launch edge [format %.3f $launch_edge] capture edge [format %.3f $capture_edge]"
    if {[set tmp [format %.5f [expr $capture_edge - $launch_edge]]] < $closest_edge} {
      set closest_edge $tmp
      set closest_launch_edge $launch_edge
      set closest_capture_edge $capture_edge
    }
  }
          puts "Final loop launch edge [format %.3f $launch_edge] capture edge [format %.3f $capture_edge]"
          puts "WNS Requirement from $launch MHz clock to $capture MHz clock = [format %.3f $closest_edge]"
          puts "Launch edge = [format %.3f $closest_launch_edge] ns"
          puts "Capture edge = [format %.3f $closest_capture_edge] ns"
}

proc check_edges_aligned {edge1 edge2} {
  if {[set difference [format %.5f [expr abs($edge1 - $edge2)]]] <= 0.00100} {
    return 1
  } else {
    return 0
  }
}
```

Figure 4 – Clock expansion Tcl script

HANDLING UNEXPANDED CLOCKS

Always treat unexpanded clocks as asynchronous, and use appropriate synchronization strategies—such as FIFOs, edge detectors or synchronizer circuits—to cross the boundary. If you employ such techniques, it is appropriate to use timing exceptions in the constraints to deal with unexpanded clocks (typically, either set_false_path or set_max_delay exceptions).

It's important to understand and explain unexpanded clocks because the edge relationships between clocks will drive the path requirements used by the timing engine in the Vivado Design Suite. Xilinx encourages close examination of all unexpanded clocks in designs to ensure that proper synchronization techniques are in place to safely traverse clock-domain crossings between unexpanded clocks.

See the Vivado User Guides for further information on the tools and design methodologies.

Prior to coming to Xilinx, John Bieker spent four years as an ASIC designer at Tellabs. John transitioned toward FPGA development in the late 1990s, focusing on RTL design, simulation, synthesis and floorplanning. After seven years of FPGA design, John joined the SAE team at Xilinx and has spent the last seven years working on a wide variety of customer designs with a focus on timing closure and device fitting. About two years ago, John joined the Tools & Methodology Team at Xilinx to help facilitate customer adoption of the Vivado Design Suite. A graduate of the University of Illinois at Champaign-Urbana, John holds an MSEE in electrical engineering. He enjoys educating customers on how to use Xilinx tools in order to help them reach their design goals.

How to Design IIR Filters for Interpolation and Decimation

It's simple to create high-order polyphase IIR filters with very low resource requirements suitable for implementation in DSP48E1 slices or logic.

by Dr. David Wheeler
Technical Director
EnSilica
david.wheeler@ensilica.com

Design engineers will most often choose finite impulse response (FIR) filters for their applications because these filters are well understood and supported by good design and IP implementation tools. The Xilinx® FIR Compiler is an excellent tool for mapping coefficients generated by MATLAB® into DSP and FPGA logic resources. However, it's possible to design an infinite impulse response (IIR) filter to meet a particular filtering specification using far fewer FPGA resources. The main drawback of choosing an IIR filter is that it does require some specialist knowledge to drive the design tools, usually followed by hand-coded RTL. However, modern tools like Xilinx System Generator can autogenerate HDL for you once you have architected the filter design and expressed it as fixed point. Xilinx provides a good overview of traditional IIR filters in a white paper on the subject [1].

Let's take a closer look at how to implement high-performance polyphase IIR filters with very low FPGA resource requirements. These structures have the property that order $NK+1$ filters can be implemented with just K multiplications. The filters can be shown to have low sensitivity to coefficient quantization, leading to efficient implementation in fixed point. Furthermore, the maximum gain at any node in a filter stage is bounded, meaning that only 1 bit of headroom is required for intermediate calculations. Here, we will present a general architecture suitable for pipelining and mapping to Xilinx DSP48E1 slices found in 7 series devices. This architecture includes a number of multiplier-less fifth-order elliptic filter designs that are applicable to efficient polyphase interpolation and decimation in just a few FPGA logic slices.

MATLAB can decompose an IIR filter design into a cascade of lower-order sections. The cascade has better numerical properties than would be the case if you were to implement the high-order difference equation directly. Typically, second-order biquadratic sections (SOS) are chosen for the low-order filters and implemented using the Direct Form 1 structure. Each SOS requires four multiplies, three adds and some rounding to reduce the bit width for the next filter in the cascade. A fixed-point implementation usually requires a further multiplier to scale between sections.

However, the cascade decomposition is not an efficient architecture for interpolators or decimators because it doesn't take advantage of polyphase decomposition. For instance, during interpolation the input data is upsampled by inserting N-1 zeros between samples, raising the input rate to the output rate, before applying to the filter. The IIR filter cascade cannot take advantage of the zero inputs to reduce computation because of feedback paths in each SOS.

IIR FILTERS FOR POLYPHASE DECOMPOSITION

The filters presented here are based on the parallel filter decomposition architecture given in Figure 1. In this parallel form, the filter on each branch is successively delayed by one more than the previous branch. Furthermore the filters in each branch are constrained to be N-band. The delays in each branch operate at the input/output sample rate, but each $A_n(z^N)$ filter has terms involving only powers of z^{-N}, which means that the difference equation operates on every N^{th} input/output sample, ignoring all samples in between.

The transfer function is given by the sum of delayed N-band all-pass filters $A_n(z^N)$.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{N}\sum_{n=0}^{N-1} A_n(z^N)\, z^{-n}$$

Furthermore, let each all-pass filter be expressed as the cascade of elementary all-pass sections, as shown

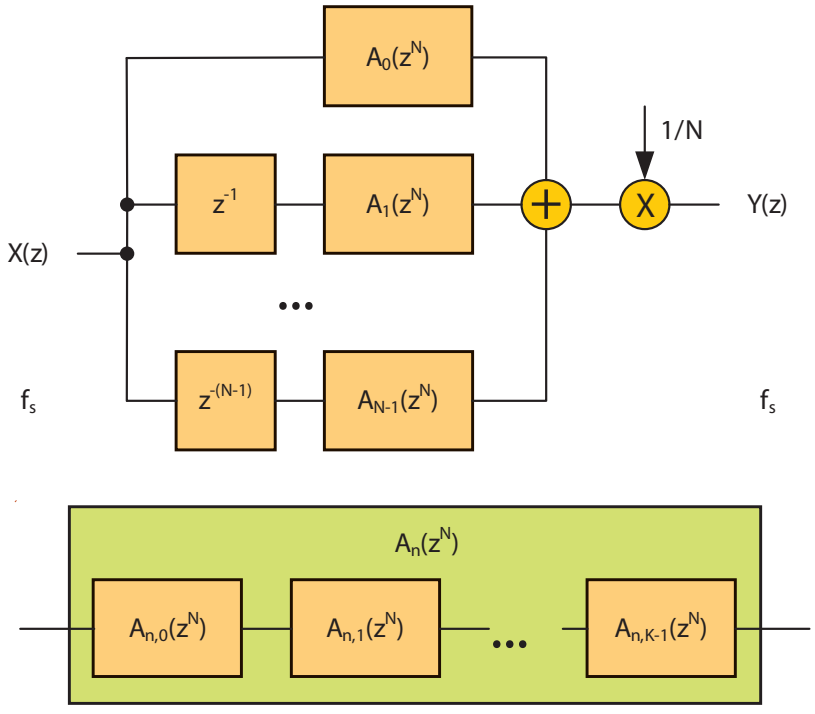


Figure 1 – At top, parallel form of an IIR filter; each branch is a cascade of elementary filters, as shown in the bottom box.

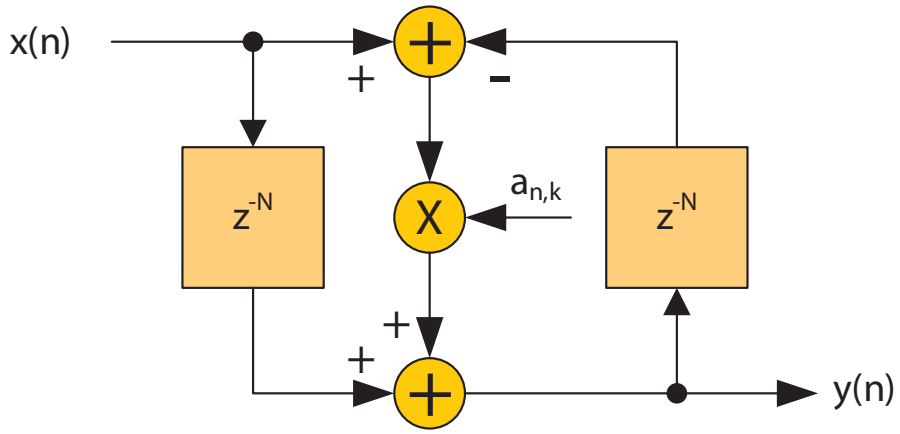


Figure 2 – Elementary all-pass section

in Figure 1. The total number of all-pass sections is K and the order of $H(z)$ is $NK+1$.

$$A_n(z^N) = \prod_{k=0}^{K_n-1} A_{n,k}(z^N)$$

$$A_{n,k}(z^N) = \frac{a_{n,k} + z^{-N}}{1 + a_{n,k}z^{-N}}$$

$$K = \sum_{n=0}^{N-1} K_n$$

Figure 2 shows an elementary all-pass section implemented in Direct Form 1. It consists of two N-sample delay elements and a single coefficient multiplier. Apart from being canonical in multipliers, there are other advantages to using Direct Form 1, which will map efficiently to DSP slices and share the delay resources when cascaded.

INTERPOLATION AND DECIMATION

This filter architecture naturally maps to the decimation and interpolation structures in Figure 3. The commutating switches, which move on every sample at the higher sampling rate, replace the delay elements in Figure 1.

For interpolation and decimation by N, we can either design around the optimum structure in Figure 3 or design a cascade of prime-factor rate conversions. Decomposing into prime factors [2] can ease the coefficient optimization problem because there are fewer free variables, and the resulting filter cascade is very close to the optimum anyway. In many applications, the required rate conversion is a power of two, which you can achieve by repeatedly interpolating or decimating by two. So without loss of generality, let's turn our attention to the special case of N=2.

It can be shown that the N=2 low-pass filter in Figure 3 has an overall order of 5, even though it's composed of two second-order all-pass sections and a delay element. Now it should be clear that, in an ideal case, only two multiplies and five adds are required to implement this fifth-order IIR filter, compared with 10 multiplies and eight adds if implemented in SOS. And there are even more savings if we consider decimation by 2, where only one branch is active for each input sample. This is equivalent to dividing up the input sequence into two half-rate sequences containing odd and even samples respectively. These are then applied to the all-pass branches and summed. Furthermore, the delay elements in each elementary section may then operate at the lower rate, halving the storage requirements. This is pipelined and mapped efficiently into a DSP slice because an external adder can be used to accumulate the two branch

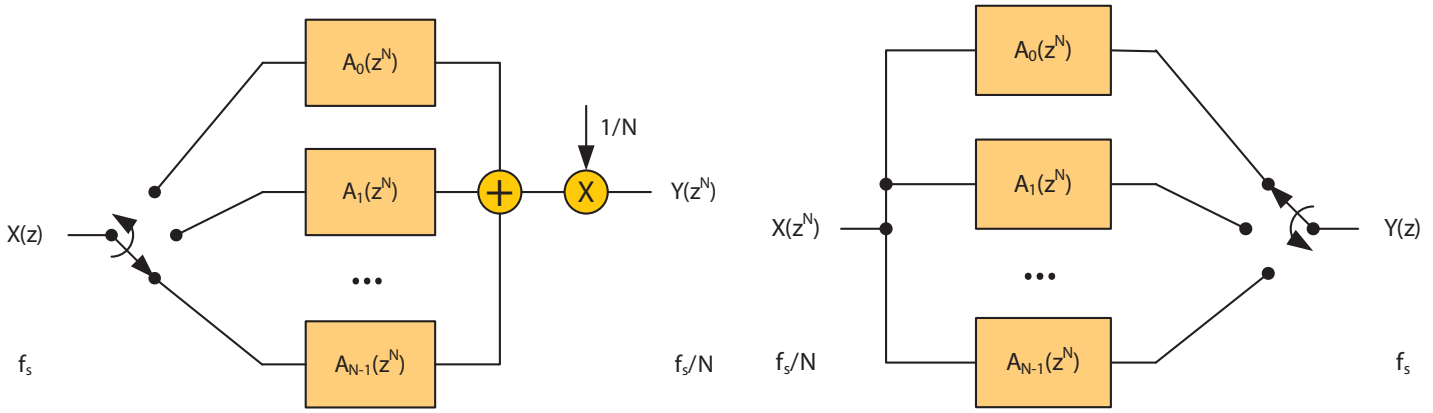


Figure 3 – Decimation (at left) and interpolation (right diagram)

filter outputs. So only one DSP48E1 slice is required to implement a fifth-order decimate-by-2 IIR filter, as shown in Figure 4. Similar observations apply to interpolation, where each branch filter provides one interleaved output sample in turn. Again, it takes only one DSP slice to support interpolation by 2.

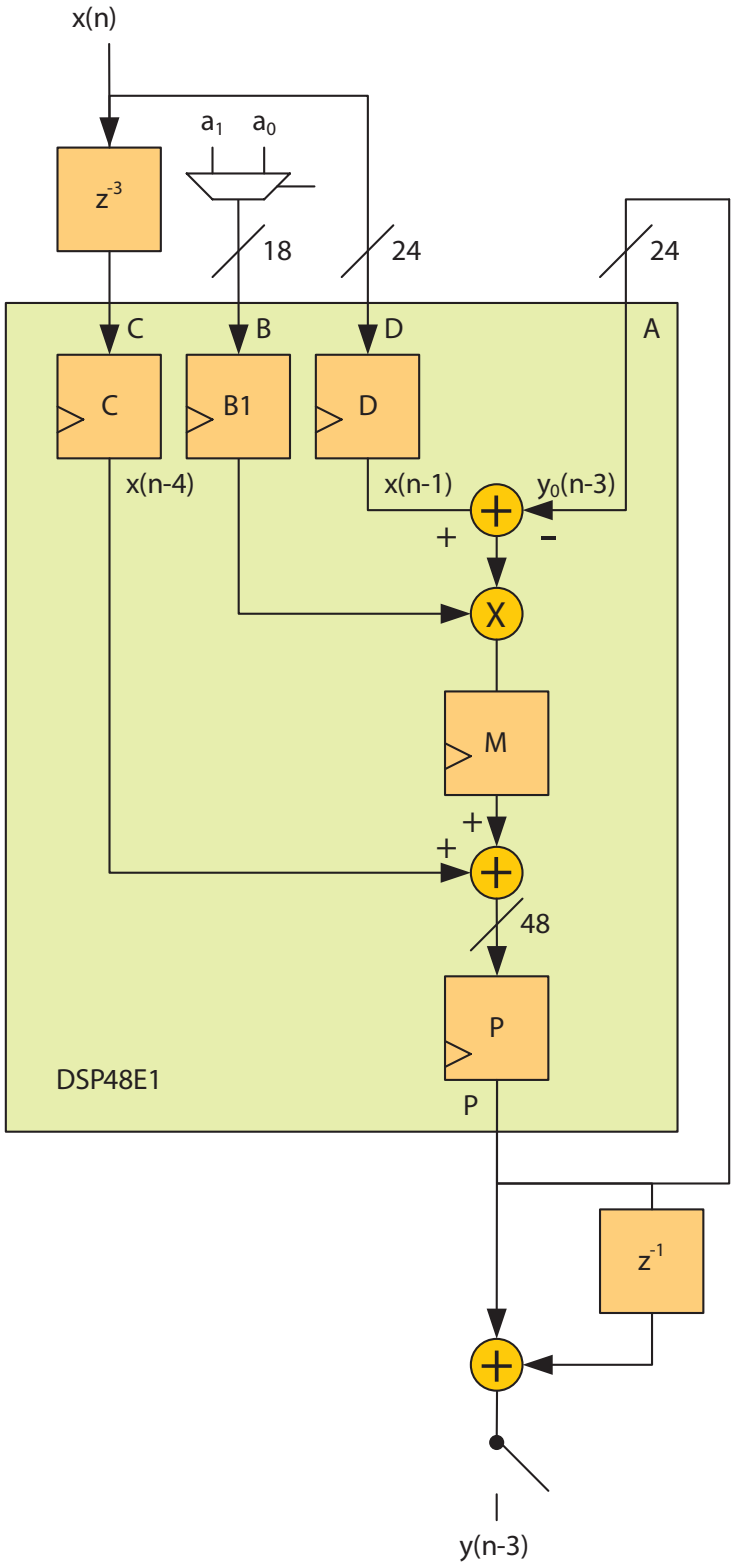


Figure 4 – Decimation by two, mapped to Xilinx DSP48E1

The A, B, C, D and P ports follow the Xilinx 7 series nomenclature [3]. You can cascade the DSP slices to implement higher-order filters, and because of the two-branch structure, the pipeline delays in each branch will be equal. Note that the DSP slice cannot be fully pipelined, which requires three internal registers, because the feedback path only has a delay of two. Using the M register achieves a higher frequency and lower power than registering at the inputs instead. For N=3 and higher then, it should be possible to get the maximum operating frequency from a DSP slice. It is also possible to get the maximum DSP slice pipelining by operating the filter in a two-channel TDM mode, where the delay elements double up; the A input feedback path would use registers A1 and A2 in the slice, and the delay before the C input would increase from 3 to 5.

QUANTIZATION AND HEADROOM

These N-band structures have been termed wave digital filters (WDF) because they emulate a doubly terminated lossless ladder network in classical analog filtering. The design of such filters having elliptic response is comprehensively covered in the literature. This work, together with the bilinear transform for translating between the analog and digital domains, provides a powerful way to design digital elliptic and Butterworth filters. Another advantage of being derived from a ladder filter is that these structures inherit low sensitivity to coefficient quantization. This means that the 18-bit word length for coefficients will be more than sufficient to implement filters with 100-dB stopband requirements without losing the equiripple properties. One analysis of the steady-state gain characteristics of an elementary section [4] showed that the maximum gain at the worst-case arithmetic node was 2.0, occurring at the pre-adder output. This has important consequences for fixed-point implementation because it means we only need one additional

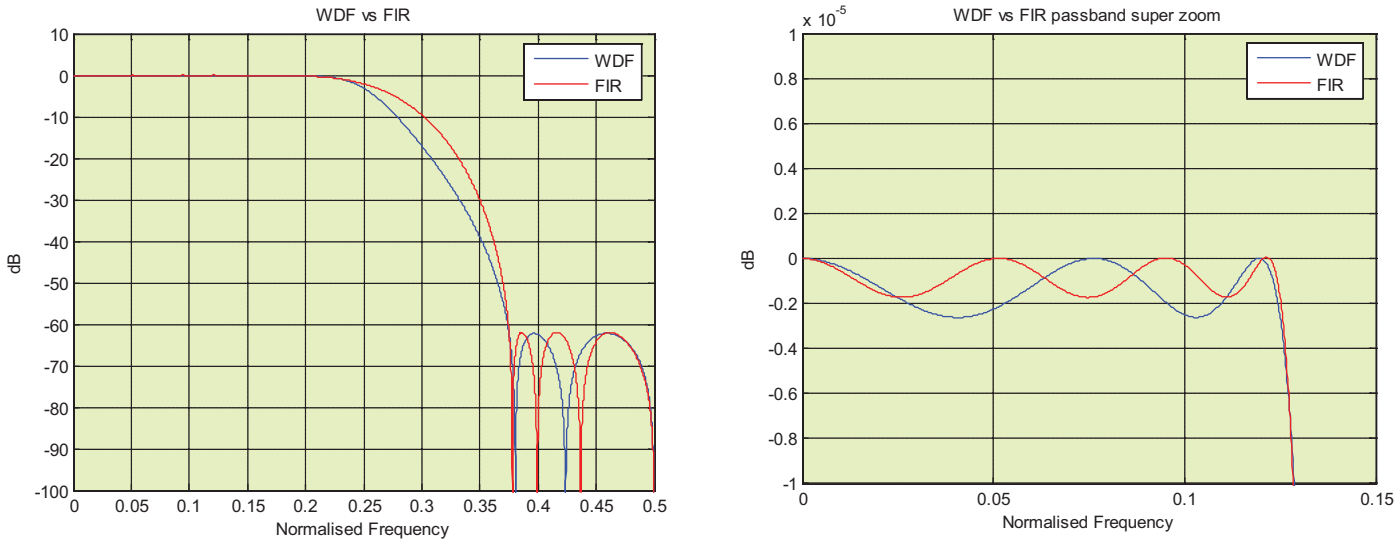


Figure 5 – 21st-order FIR filter and fifth-order WDF to meet the same stopband (left diagram) and passband (right diagram) specification

bit at the pre-adder output. In a Xilinx DSP48E1 slice there's no saturation logic at the pre-adder output, so limiting the A and D inputs to 24 bits will allow the filter to operate without numerical overflow but, more important, to the maximum internal precision of the DSP slice. Although the steady-state gain is limited to 2.0, transients in a step response can exceed 2.0, so for this reason it is recommended that you use only 23 bits and sign extension for safe operation. For some data sources, such as 24-bit music, you can apply the full dynamic range, taking advantage of prior knowledge of the signal characteristics.

If your input data is less than 24 bits then it should be left aligned in the word, thereby providing some fractional bits. For instance, with 16-bit input data a good choice would be one guard bit, 16 data bits and seven fractional bits. In general only three to four fractional bits are sufficient for good accuracy compared to floating point.

GENERATING FILTER COEFFICIENTS

Methods to generate coefficients for any order elliptic filter implemented by two-branch structures are adequately covered in a paper by R.A. Valenzuela and A.G. Constantinides [5]. Figure 5 shows a fifth-order (two coefficients: a0 = 0.1380, a1 = 0.5847) WDF filter with normalized passband 0.125, compared with a 22-tap FIR filter, designed using Parks-McClellan to meet the same passband ripple and stopband attenuation. The results show a typical four-to-one saving in filter order and the WDF requires just two multiplies compared with 11 for the FIR (taking the symmetry into account).

In a halfband WDF, the stopband and passband ripples are not independent. However, setting any reasonable stopband attenuation leads to negligible passband ripple; for example, it was 10^{-6} dB for the filter in Figure 5. The FIR has an advantage in being able to specify these two design parameters independently, so that you can specify a higher passband ripple to aid meeting a stopband requirement for a given order.

Next, let's look at what can be achieved without using DSP48E1, but quantizing the coefficients heavily to implement a filter in just a few LEs. Recall that since WDF filters are based

Normalized Passband	Coefficients	Stopband Attenuation (dB)
0.1	a_0 = 1/8 a_1 = 9/16 = 1/2+1/16	69
0.175	a_0 = 3/16 = 1/8+1/16 a_1 = 21/32 = 1/2+1/8+1/32	44
0.19	a_0 = 7/32 = 1/4-1/32 a_1 = 89/128 = 1/2+1/8+1/16+1/128	39

Table 1 – Quantized coefficients for three example passbands

```
module wdf5_deci #(parameter WIDTH = 16, FRAC = 3, GUARD = 2)
(
    input                           clk,
    input                           reset,
    input      signed [WIDTH-1:0]   data_in,
    input                           enable,
    output reg signed [WIDTH:0]     data_out,
    output reg                      valid_out
);
    reg                                state;
    reg  signed [GUARD+WIDTH+FRAC-1:0] x, x_1r, x_2r, sum_1r, sum_2r;
    wire signed [GUARD+WIDTH+FRAC-1:0] p, p0, p1, sum, diff, dec;

    always @(posedge clk)
    begin
        if (reset)
        begin
            x <= 0;
            x_1r <= 0;
            x_2r <= 0;
            sum_1r <= 0;
            sum_2r <= 0;
            state <= 0;
            data_out <= 0;
            valid_out <= 0;
        end
        else if (enable)
        begin
            x <= data_in <<< FRAC;
            x_1r <= x;
            x_2r <= x_1r;
            sum_1r <= sum;
            sum_2r <= sum_1r;
            state <= ~state;
            data_out <= (dec >>> FRAC+1);
            valid_out <= state;
        end
    end

    assign diff = x - sum_2r;
    assign p0 = diff >>> 3;                      // 0.125 = 1/8
    assign p1 = (diff >>> 1) + (diff >>> 4);     // 0.5625 = 1/2+1/16
    assign p = state ? p1 : p0;
    assign sum = p + x_2r;
    assign dec = sum_1r + sum_2r;
endmodule
```

Figure 6 – Verilog code to implement the first decimation filter in Table 1

on ladder analog prototypes, then we should be able to find low-bit-density coefficients for a range of specified passbands. Where possible, the bits are written in a canonical decomposition to minimize the number of adders, as seen in Table 1.

Although the stopband attenuation of a fifth-order filter with 0.19 passband is modest, simply cascading two of these filters produces a 10th-order filter with a stopband of -78 dB. For example, the first decimator in Table 1 can be implemented with the few lines of Verilog given in Figure 6.

RESOURCE EFFICIENT

We have presented an IIR filter architecture that naturally maps to interpolation and decimation. This structure is more resource efficient than IIR second-order sections or even a FIR. It maps well to the preadd/multiply/postadd of a DSP48E1 slice, and is robust to fixed-point quantization effects of coefficients to 18 bits to give a controlled 100-dB stopband. Some multiplier-less fifth-order designs are permissible for special pass bandwidths, and you can map them to a few registers and adders in logic to save DSP resources.

References

1. Xilinx white paper WP330, "Infinite Impulse Response Filter Structures in Xilinx FPGAs," August 2009

2. P.P. Vaidyanathan, Multirate Systems and Filter Banks. *Prentice-Hall, Englewood Cliffs, N.J., 1993*

3. Xilinx, 7 Series DSP48E1 Slice User Guide, v1.5, April 3, 2013

4. Artur Krukowski, Richard Morling, Izzet Kale, "Quantization Effects in Polyphase N-Path IIR Structure," IEEE Transactions on Instrumentation and Measurement, vol. 51, no. 6, December 2002

5. R.A. Valenzuela and A.G. Constantinides, "Digital Signal Processing Schemes for Efficient Interpolation and Decimation," Electronic Circuits and Systems, IEEE Proceedings Part G, vol. 130, no. 6, 1983

Implementing Analog Mixed Signal on the Zynq SoC

Using the XADC within the Zynq SoC greatly increases system integration. The good news is that implementation is quite straightforward.

by Adam Taylor
Chartered Engineer
aptaylor@theiet.org

This is the fourth in a planned series of hands-on Zynq-7000 All Programmable SoC tutorials by Adam Taylor. Earlier installments appeared in Xcell Journal *issues 82, 83 and 84. A frequent contributor to* Xcell Journal, *Adam has a second article in this issue, on commissioning your design. He is also a blogger at All Programmable Planet (*www.programmableplanet.com*).*

The Xilinx® Zynq®-7000 All Programmable SoC comes with an XADC block that contains two 12-bit analog-to-digital converters. These ADCs are capable of sampling at up to 1 Megasample per second (MSPS), providing an ideal effective input-signal bandwidth of 500 kHz (250 kHz on the auxiliary inputs). The XADC can multiplex among 17 inputs along with a number of internal voltages and temperatures. If your design is pin-limited in terms of available analog-capable inputs for external signals, you can configure the XADC to drive an external analog multiplexer and sequence through all the inputs in the desired order.

The XADC is capable of unipolar or bipolar measurements; hence, each input to the device is differential. The 17 differential inputs are split between one dedicated pair, referred to a VP/VN, and 16 auxiliary inputs that can be either analog or digital I/O, referred to as V_{AUXP} and V_{AUXN}. The effective input-signal bandwidth varies depending upon whether you are using the dedicated pair, in which case it is 500 kHz, or the auxiliary inputs, in which case the maximum bandwidth is 250 KHz.

The mixed-signal performance the XADC is very good, offering a minimum of 60-dB signal-to-noise ratio (SNR) and 70 dB of total harmonic distortion (THD), according to the data sheet. Depending upon the temperature range the device is exposed to (-55ºC to 125ºC or -40ºC to 100ºC), the resolution of the XADC is ether 10 bits or 12 bits. This gives the XADC an equivalent number of bits of 9.67 when using the equation

$$ENOB = (SNR–1.76/6.02)$$

(See *Xcell Journal* issue 80, "The FPGA Engineer's Guide to Using ADCs and DACs," for more detail on the theory behind these devices.)

The XADC provides for user-selectable averaging to reduce the noise present on the input, offering 16-, 64- or 256-sample averaging. The user can also program a series of minimum and maximum alarm levels for each internal device parameter that is measured.

WHAT IS THE XADC GOOD FOR?

Designers can use this ADC for a number of applications ranging from simple housekeeping telemetry of onboard parameters (voltage, current, temperature) to supporting touch sensors, motor control or simple wireless-communication protocols. You can also use it in military or other critical systems to detect tamper attempts on the unit.

One great advantage is that you can use the XADC to monitor a number of internal parameters of the system to verify the health of your design. In addition, to ease verification during the early stages, you can use the XADC within a Zynq-7000 All Programmable SoC-based system to measure the temperature as recorded by the on-chip temperature sensor, along with the following additional parameters:

- V_{CCINT}: The internal core voltage of the programmable logic
- V_{CCAUX}: The auxiliary voltage of the programmable logic
- V_{REFP}: The XADC positive reference voltage
- V_{REFN}: The XADC negative reference voltage
- V_{CCBRAM}: The BRAM voltage of the programmable logic
- V_{CCPINT}: The internal core voltage of the processing system
- V_{CCPAUX}: The auxiliary voltage of the processing system
- V_{CCDDR}: The double-data-rate RAM voltage of the processing system

HARDWARE IMPLEMENTATION

To get your XADC working, first open up the PlanAhead™ project that contains your Zynq SoC implementation (if you are unsure how to do this, see the cover story in *Xcell Journal* issue 82). Select the processor subsystem you created and double-click on it to open Xilinx Platform Studio. It is here, in XPS, where you will modify the processing system.

Once XPS opens, select the IP Catalog tab on the left-hand side of the project window and choose the AXI XADC from under the analog menu. A prompt will ask if you want to add the IP to your design. Click yes and an IP configuration window will be displayed, as shown in Figure 1. There is not much here to configure unless you want to get into the complexities of the AXI bus interface so as to optimize it.

The main area of manipulation is the user tab, which allows you to determine whether you wish to include an interrupt to the processor and to enable the temperature bus. You can connect the temperature bus, if enabled, to the Memory Interface Generator to provide temperature information that may

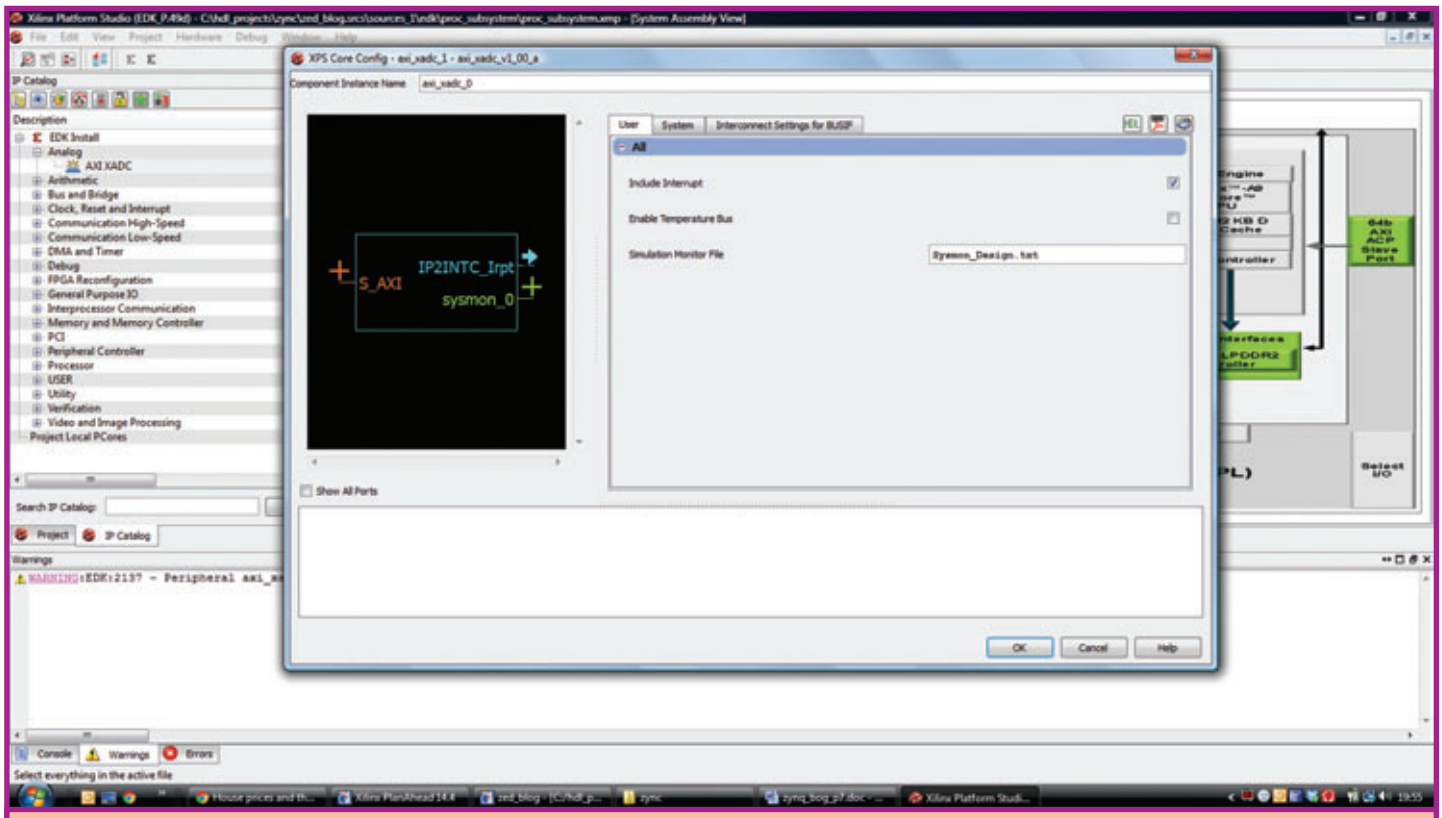


Figure 1 – Adding in the XADC within Xilinx Platform Studio

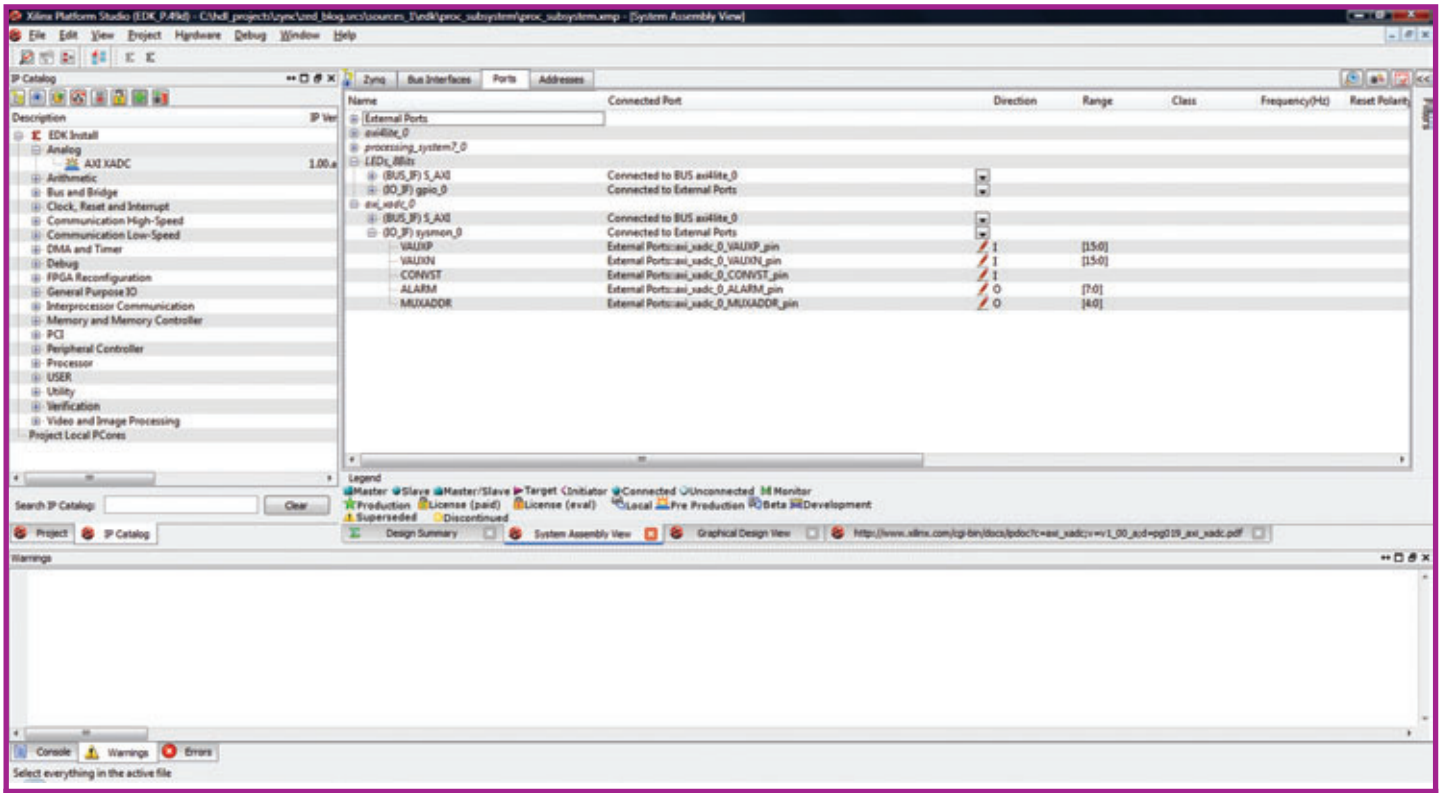


Figure 2 – The external connections within XPS

affect timings. The simulation monitor file is the file used when simulating the XADC within ISim.

Once you are happy with the configuration of your XADC, close the IP configuration window and the XADC will be added into your processing system design. If you click on the addresses tab within the System Assembly viewer, you will see the address range that has been allocated to the XADC. To lock the address range, select the tick box.

The next stage is to configure the number of external ports connected to the XADC. By default the XADC will be created with support for an external multiplexer, all 16 auxiliary analog inputs, eight alarm outputs and a conversion start, as shown in Figure 2.

You can remove these ports if necessary such that the ADC has only the dedicated analog inputs Vp and Vn and the internal channels to monitor. For example, the ZedBoard supports only two more auxiliary analog inputs on channels 0 and 8 while also providing four GPIOs dedicated to the XADC. You may connect these GPIOs to the alarm outputs or mux outputs as desired. Within XPS you can disconnect external I/O by selecting the "not connected to external ports" option in the drop-down menu.

If you wish to have some auxiliary inputs still connected—for example, to include the two additional ZedBoard inputs—you can leave the V_{AUXP} and V_{AUXN} channels within the system and connect only channels 0 and 8 at the top level of your RTL design. If you wanted to include these auxiliary inputs but remove the other ports from being external ports, you would need to right-click on a specific I/O and select the "no connection" option, as Figure 3 shows.

Once you are happy with the external connections, it's time to run the design rule check. If this DRC shows no errors, exit XPS and return to PlanAhead, where you can now generate the bitstream. If you take the time to check your synthesis results, you should see that an XADC is included in the resource list.

WRITING THE SOFTWARE

Having completed the implementation, you then need to re-export the hardware to the Software Development Kit (SDK) in order to update the board support package. Within PlanAhead, select file -> export -> export hardware for SDK. This will bring up a warning, as we will be overwriting an existing hardware definition. Click yes to the overwrite.

If you have the SDK open, you will also see a warning describing the changes in the hardware specification file. If you do not have SDK open, you should see this warning the next time you open it.

Once you click on yes within the SDK warning, the hardware definition will be updated and your project will

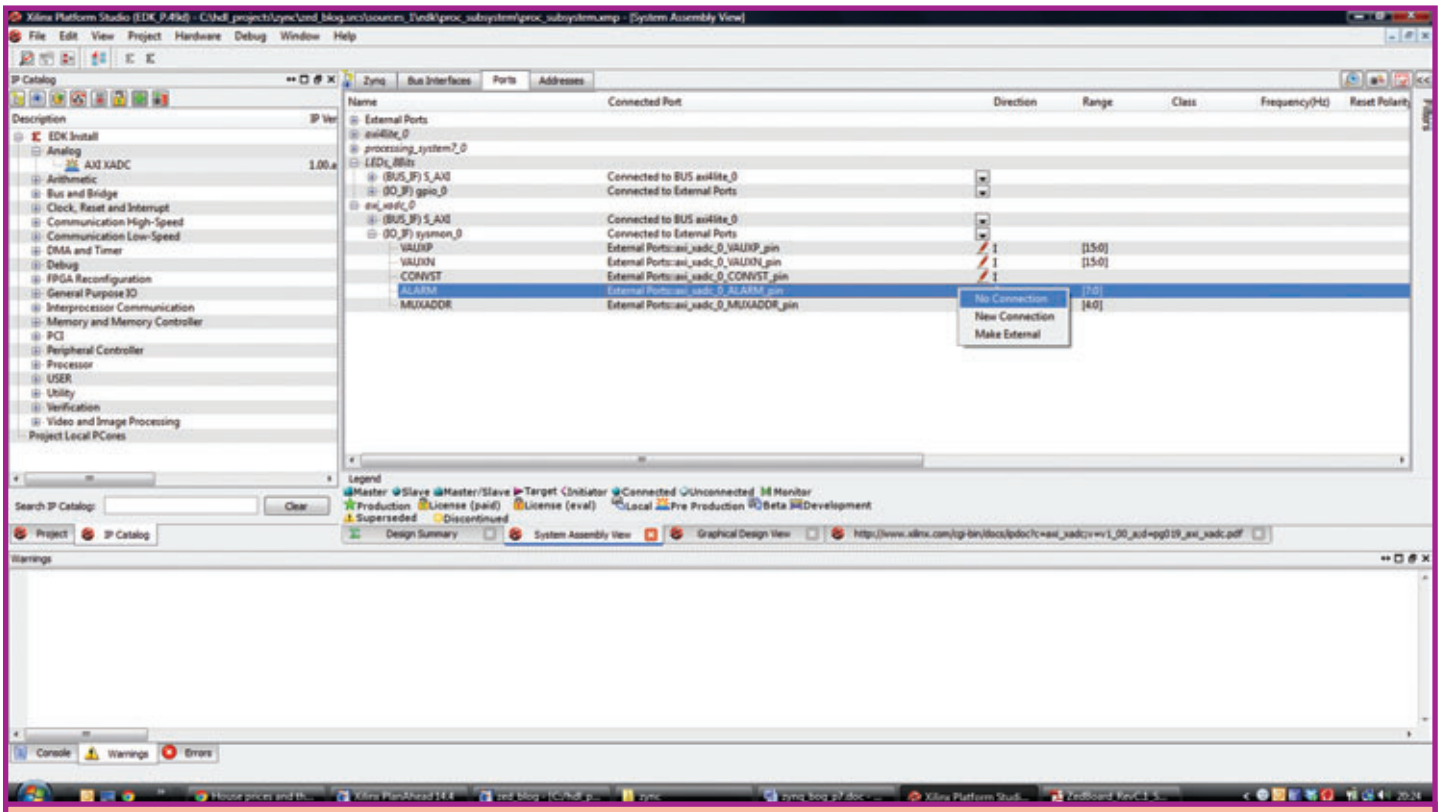


Figure 3 – Customizing the external connections

be rebuilt. Provided you have no errors with your design build, you will now be in a position to start modifying the code.

If you open the system.xml or the xparameters.h files, you will see not only the address range for any other peripherals within your design but also the address range of the XADC we just added in XPS.

Opening the system.mss file will reveal comprehensive details on the board support package (BSP). What we are really interested in here is the peripheral drivers, under which you will see the xai_adc_0 (or whatever you named your XADC within Xilinx Platform Studio). Next to this you should see hot links that will open up both the documentation and examples.

To reduce development time, Xilinx rather helpfully provides a number of drivers within header files to help us use these devices. Within your code you need to access these drivers. To do this you need to include the following header file, which was generated when the BSP was updated:

#include "xadcps.h"

This file contains definitions for the XADC registers, sampling averaging options, channel sequence options and power-down modes, among others. It also contains a series of type definitions, macros and functions that you can use within your system.

PUTTING IT ALL TOGETHER

For my simple example I am going to read the internal temperature and voltage parameters of my system and output them over an RS232 link.

The first thing to do within the code is to look up the configuration of the XADC to be initialized. This requires a pointer of the type XAdcPs_Config. Using the function call `XAdcPs_LookupConfig()` coupled with the device ID obtained from the updated xparameters.h file, the configuration (device ID and base address of the XADC being initialized) will be stored in the pointer. The next step in the initialization process is to use the information previously obtained and stored within the configuration pointer, a step that also requires a pointer of the type `XAdcPs`.

Status_ADC = XAdcPs_CfgInitialize (XADCInstPtr,ConfigPtr,ConfigPtr->BaseAddress)

I named my configuration pointer ConfigPtr and my instantiation pointer, XADCInstPtr. Having initialized the XADC, I now need to configure it as desired for my example. Here's how I set up the configuration process:

1. Run a self-test allowing me to check that there are no issues with the device using the function `XAdcPs_SelfTest()`

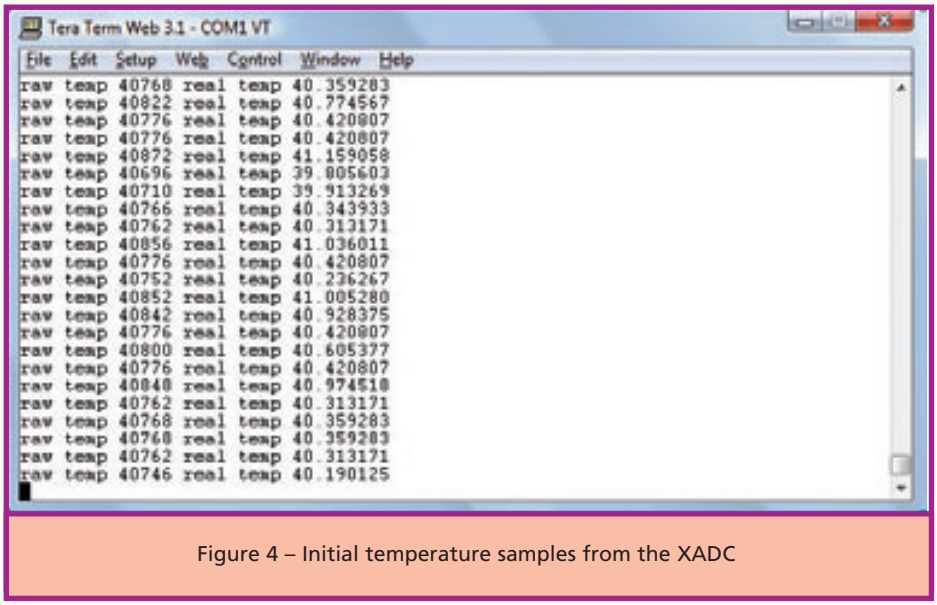


Figure 4 – Initial temperature samples from the XADC

2. Stop the sequencer from its current operation by setting it to a single channel using `XAdcPs_SetSequencerMode()`
3. Disable any alarms that may be set using `XAdcPs_SetAlarmEnables()`
4. Restart the sequencer with the sequence I desired using `XAdcPs_SetSeqInputMode()`
5. Configure the channel enables that I wish to sample using `XAdcPs_SetSeqChEnables()`

Having completed the above steps, I am ready to begin actually using the ADC and receiving the data from the XADC. Reading a sample from the XADC can be as simple as calling the function `XAdcPs_GetAdcData()`. For the internal temperature and voltage parameters, I then used two provided macros—`XAdcPs_RawToTemperature()` and `XAdcPs_RawToVoltage()`—to convert the raw XADC value into a real-world temperature or voltage.

Both of these real-world and raw values are then output over an RS232 link. Figure 4 shows the initial results when sampling the temperature of the device only.

When I expanded the sampling to cover the internal power rails and temperature, the following was returned:

- Raw Temp 40696, Real Temp 39.805603
- Raw VccInt 21677, Real VccInt 0.992294
- Raw VccAux 39431, Real VccAux 1.805008
- Raw VccBram 21691, Real VccBram 0.992935
- Raw VccPInt 21649, Real VccPInt 0.991013
- Raw VccPAux 39402, Real VccPAux 1.803680
- Raw VccDDR 32014, Real VccDDR 1.465485

All of these parameters are within the acceptable limits for operation of the Zynq SoC, but that is to be expected in laboratory conditions.

As you can see, implementing and using analog mixed-signal within the Zynq SoC is very simple and straightforward. The XADC will increase the integration and functionality of your All Programmable system.

A Pain-Free Way to Bring Up Your Hardware Design

Commissioning your digital hardware design can be a stressful and time-consuming process. But there are ways to ease the angst.

by Adam Taylor
Chartered Engineer
aptaylor@theiet.org

One of the most exciting moments in an engineering project is when the hardware arrives in the lab for the first time, ready for commissioning before integration testing. This stage in the development process typically can mean long hours and a certain amount of stress for all the engineers on the project. But tools and techniques are available to help ease the way and move the project along.

Let's take a look at how we can minimize any issues that may arise in getting a design to the next level, and how to get through the commissioning phase in a timely manner.

THINK OF HOW YOU WILL TEST FROM DAY ONE

All engineers know that the cost of implementing fixes increases as the development progresses. It is more expensive to fix a pinout error once the design has been finalized and manufactured than during an early design review, for example. This is also true for testing and integration in that the earlier you think about how you will test the hardware, FPGA, system, etc. and write the test specifications, the easier it becomes for the engineering team to include the necessary test points, hooks and functionality. The aim of the testing is to ensure you are delivering a safe system that meets the user's specified requirements. You must therefore ensure that each of these requirements is demonstrated by test. Thus, the functional test requirements should be flowed down and traceable to the design requirements (that is, each test should show which requirements it addresses).

It is also good practice to compile a design verification matrix that details how each functional requirement is to be tested—for example by test, by analysis or by read across (if the requirement was qualified or tested earlier, on another project). This document (Figure 1) may also show which tests are proof of design and which ones are to be used for the production run. Completing these documents early in the project life cycle will ensure that both the system design teams and the test equipment design teams will have an established baseline.

However, before the functional tests can be conducted, the design engineers must first be satisfied that the underlying hardware is correct. They will typically require a hardware-level test specification that will include such things as voltage rails, performance and basic verification of the hardware; the latter is performed prior to the functional testing.

It is worth determining the test equipment that you will need and what performance is required—for instance, do the pattern generator and logic analyzer have sufficient storage depth and operating frequency? You will also need to determine whether any more-specialized test equipment will be required, for instance arbitrary waveform generators, high-stability frequency references and the like.

DESIGN PHASE INCLUSIONS

During the design of the hardware, you may need to include several design features and functions to allow testing of the board with greater ease. These requirements can be simple or more in depth.

The simplest and most often implemented test provision is placing test points on all voltage rails (saving the need to damage solder joints by probing on them). However, it is also good practice to place a pad connected to the ground (0V) return close to the voltage test point, to ease the testing. Protecting this test point with a high-value resistor will limit the current that could flow if it were accidentally shorted during testing. You may also wish to think about adding test pins to these pads to enable them to connect to automatic test sys-

tems, which can record the results later on during production runs.

Being able to monitor the outputs of clocks and resets is also important. For this reason, it is good practice to place test points on the reset line. You should be sure also to correctly terminate an unused clock buffer and add test points, allowing the clock to be probed with ease. Then too, consider adding test ports to enable signal injection and extraction via a pattern generator, logic analyzer or other test instrument.

To aid with closure of the power budget on the prototypes, it is often good to place low-value resistors (10 milliohms, 100 milliohms, etc.) in series with the output of the voltage regulators, if possible. Doing so will enable you to accurately determine the current drawn by a rail.

For a first-of-type design—the first actual build of a new product—you might make a more in-depth design decision enabling you, for example, to decouple the power supplies from the downstream electronics. In this way, you can establish that the power supplies and sequencing are functioning correctly, reducing the danger of overstressing or damaging downstream components. Another example of a more-detailed upfront design stage you can pursue to aid testing is to ensure that the JTAG port can be used for more than just programming any FPGA or processor within the system. You might also press the port into service for initial verification of the hardware using boundary scan testing. Boundary scan test can be very useful in retiring hardware design risks early in the test phase. It does require that the design be optimized to ensure maximum coverage of boundary scan devices.

Boundary scan test can be very useful in retiring hardware design risks early in the test phase.

Many FPGA devices also provide a way to monitor the die temperature via the inclusion of a temperature diode. You will need to provide the means to supply this diode with a constant current. Determining the die temperature is a very useful capability when you are trying to achieve derated junction temperatures.

Requirement Number	Requirement Title	Analysis	Inspection	Test	Demonstration
100	FPGA Power Rails			X	
101	FPGA Power Rail Sequence			X	
102	FPGA Reset Release			X	
103	FPGA Reset Asserton			X	
104	FPGA Oscillator Tolerance +/-100 ppm	X			
300	Module Weight		X		
400	Front Panel LEDs				X

Figure 1 – A verification matrix detailing how each functional requirement is to be tested can be a helpful tool.

HARDWARE CHARACTERIZATION

When the system arrives in the lab for the first time, the first thing you will wish to do is determine that the module underlying the hardware is acceptable to progress to further testing. Checks will include the initial power-on testing of the module, which can be a tense experience. When the module first arrives, you will wish to ensure that it has been manufactured correctly before applying initial power. The first step is to ensure that all of the components are fitted in their correct positions, with pin 1 oriented in the proper position and any polarized components correctly placed. Often a design may include a number of components that don't need to be fitted (for example, they might be present for different variants or build options). Ensure that all of your components are fitted or not as required by your design, especially if only one pull-up or pull-down resistor is meant to be fitted to select a configuration mode.

Once you have checked the population of the printed-circuit board, the next step will be to power the board for the first time. For any engineer, this can be a nerve-wracking moment. However, the test provisions you've made during the design phase (test points, current-sense resistors, etc.) will be of great assistance at this time. The first step is to ensure that the power output of point-of-load and other regulators is not shorted to return. You may see low impedances on rails which contain devices that have a high-current demand. However, the impedances should be greater than 1 ohm.

Provided you are confident that none of the rails are shorted, your next task is to apply power. When doing the initial power-on, I favor a two-stage approach. The first stage applies low voltage (0.5 V) at a low current to ensure you haven't missed any shorts between planes or voltage rails. The next stage is to then power up the design at the correct working voltages and with the current limit set to account for the predicted current (do not forget to account for inrush current).

Having successfully applied power to the design, your next job is to determine that the power rail sequencing, reset and clocks are working as intended. Remember to ensure that the reset

duration extends past all of the clocks and becomes stable prior to its release.

The next stages of characterization are to ensure that you can see the hardware via the JTAG chain. This will enable you to not only program the FPGA but also to perform boundary scan testing. Boundary scan testing can help you quickly test the interconnections between devices; test memories to ensure they function correctly; and even loop back inputs and outputs, provided you develop loopback connectors. JTAG and boundary scan testing can remove much of the risk from a design before you progress to more-detailed testing.

TOWARD SIMPLER RTL

If you have a complicated design at both the hardware and FPGA level, it can be helpful to develop a simplified, pared-down version of the RTL to aid in the testing of the board and the interface between the FPGA and the peripheral (Figure 2). This is especially the case if you are designing high-speed interfaces. You could use this streamlined RTL in conjunction with the Xilinx® ChipScope™ tool to capture data, along with Block RAMs that have been preloaded with data patterns to act as stimulus. This tactic is especially helpful when using ADCs and DACs connected to an FPGA. In this case, you should utilize the reprogrammable nature of the FPGA to the maximum extent to develop designs that will allow parametric testing of the ADC and DAC, for instance noise/power ratio, spurious-free dynamic range and effective-number-of-bit calculations.

You should also aim to capitalize on the resources the FPGA provides, especially the Xilinx System Monitor and XADC, which can be very useful for monitoring the voltage rails on dice, hence helping to verify the power integrity analysis you performed during the design stage. These technologies can also easily report the die temperature as well; that can be of use for environmental testing and for correlating the power drawn against the die temperature.

Many times, crafting a simpler RTL design and using the resources provided by the FPGA can be of great assistance in pinpointing an area that is not functioning as intended.

```
OBUFDS_inst : OBUFDS
generic map (
IOSTANDARD => "DEFAULT")
port map (
O => idc(1), -- Diff_p output (connect directly to top-level port)
OB => idc(0), -- Diff_n output (connect directly to top-level port)
I =>icheck);

OBUFDS_inst1 : OBUFDS
generic map (
IOSTANDARD => "DEFAULT")
port map (
O => dac1_sync_p, -- Diff_p output (connect directly to top-level port)
OB => dac1_sync_n, -- Diff_n output (connect directly to top-level port)
I =>'0');

process(clk)
begin
  if rising_edge(clk) then
    if count = x"ff" then
      rs <= '0';
    else
      count <= count + 1;
      rs <= '1';
    end if;
  end if;
end process;

process(clk)
begin
  if rising_edge(clk) then
    addr <= addr + 1;
  end if;
end process;
```

Figure 2 – This streamlined code is a snippet from a simple DAC interface tying off the outputs to a known state to enable the generation of a sine wave at Fs/2. The standard version of such code would be hundreds of lines long.

WHAT IF I ENCOUNTER ISSUES?

As you are working through your test plan, you may come across one or two issues where things do not function as intended or fail to meet the functional performance desired. Don't panic—there are a number of avenues to investigate while determining the root cause and the required corrective action.

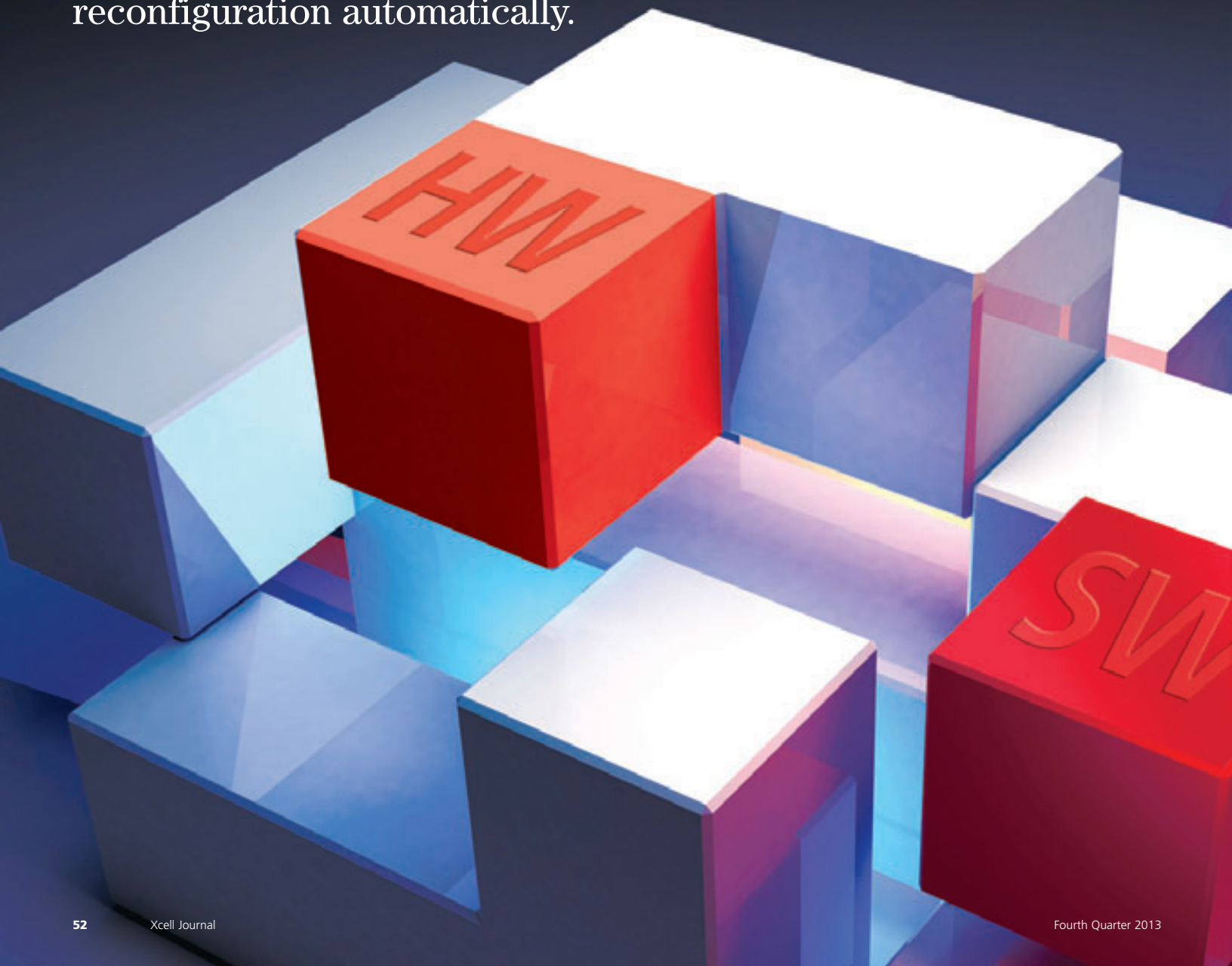
In the event, resist the temptation to immediately change anything. First, go and revisit the design, particularly the schematics and design information, including data sheets. If the issue is FPGA related, it is worth checking whether the pin constraints file correctly ties up with the design. It's possible the file may have gotten out of synchronization with the design.

If nothing jumps out at you as obviously wrong, it is worth utilizing the power of the Internet to see if other engineers have encountered the same problem you are seeing. There are many Web forums available as well where you can question other designers. Programmable Planet and Xilinx forums provide a wide range of support for FPGA-based designs.

In the end, commissioning hardware can be one of the more challenging but rewarding aspects of engineering. Thinking about the testing early in the design stage and including test facilities within the design can considerably ease the commissioning process. Using all of the available resources to debug the system—including ChipScope, System Monitor and XADC—coupled with wise use of traditional test equipment can lead to a successful development.

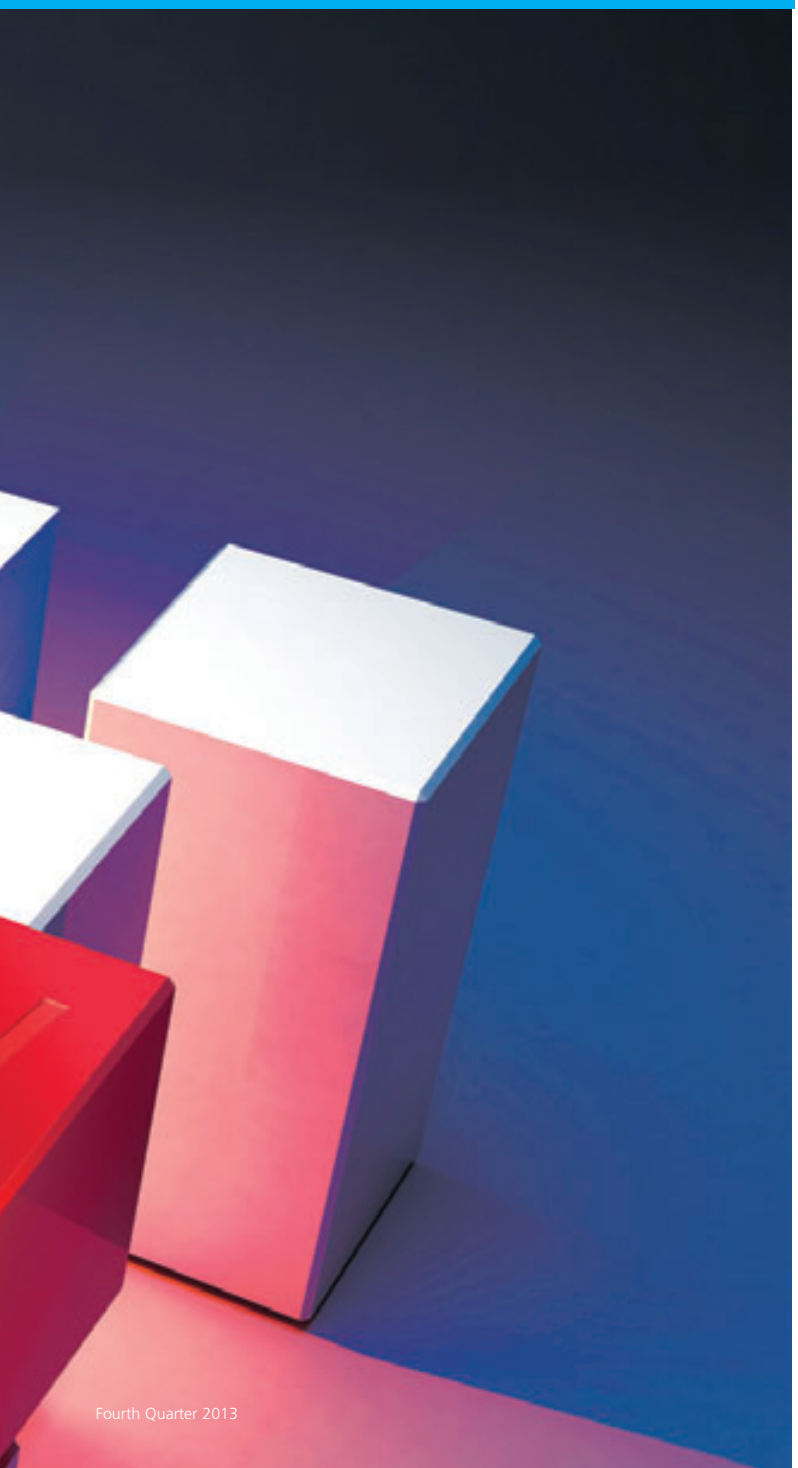
Middleware Turns Zynq SoC into Dynamically Reallocating Processing Platform

TOPIC's Dyplo IP bridges the gap between software and hardware design, and handles FPGA partial reconfiguration automatically.



by Dirk van den Heuvel
Hardware Architect
TOPIC Embedded Systems
dirk.van.den.heuvel@topic.nl

René Zenden
System Architect
TOPIC Embedded Systems
rene.zenden@topic.nl



Over the last three decades, FPGA technology has evolved from primarily programmable logic arrays with a few registers into the advanced, integrated system devices we know today. A little over a decade ago, vendors first began offering 8-, 16- and 32-bit soft processors on their FPGAs to achieve greater degrees of integration and functionality between processing and programmable logic. They then started implementing hardened versions of these processors, offering customers even more system benefits. Xilinx®'s Zynq®-7000 All Programmable SoC is the state of the art among such devices.

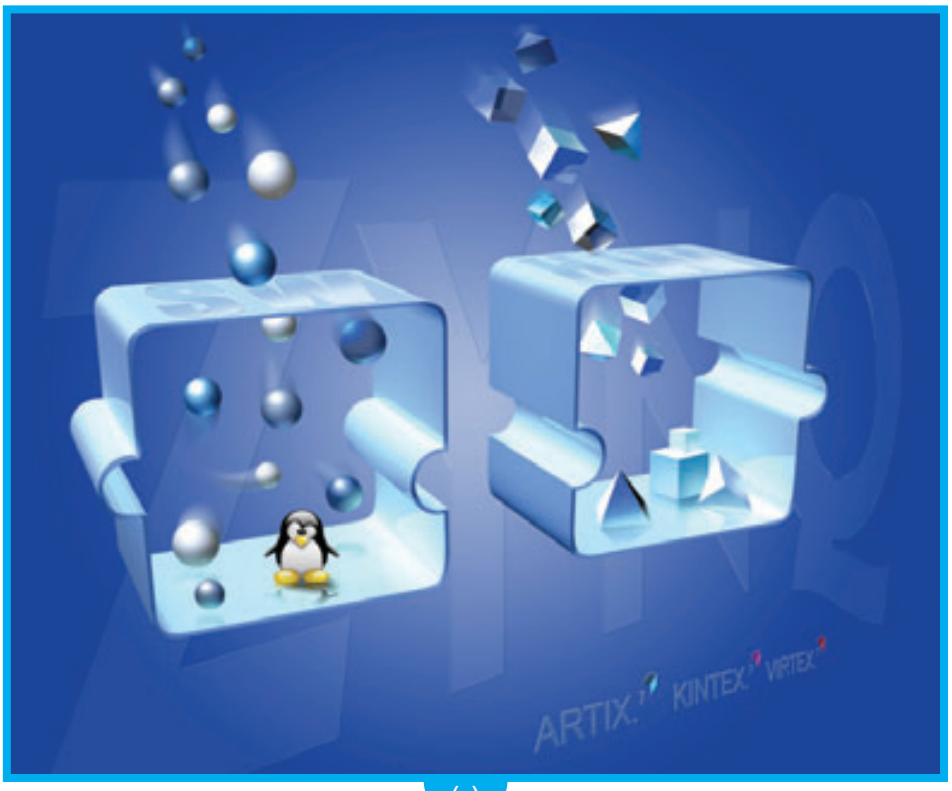
With its ARM® dual-core Cortex™-A9 MPCore processing system, FPGA fabric and peripherals all on a single device, the Zynq SoC has moved the hardware and software worlds closer together than ever before, enabling new degrees of system-level integration and functionality. But this tighter integration has, as a consequence, blurred the boundaries between software and hardware design, as it introduces FPGA design in the software domain and application software development in the FPGA domain.

Our company, TOPIC Embedded Systems, has developed an IP block that significantly reduces the development time and costs of creating systems on the Zynq SoC. Our Dyplo system's middleware functionality tackles the gap between hardware and software design, and provides a means to enable a fully software-driven development flow. Dyplo does automatic partial reconfiguration, an advanced design technique in which the FPGA layout can change its hardware configuration on the fly to best optimize performance for a given task. A graphical tool lets users drop functions into each partition and then figure out if they run best in logic or on the processor.

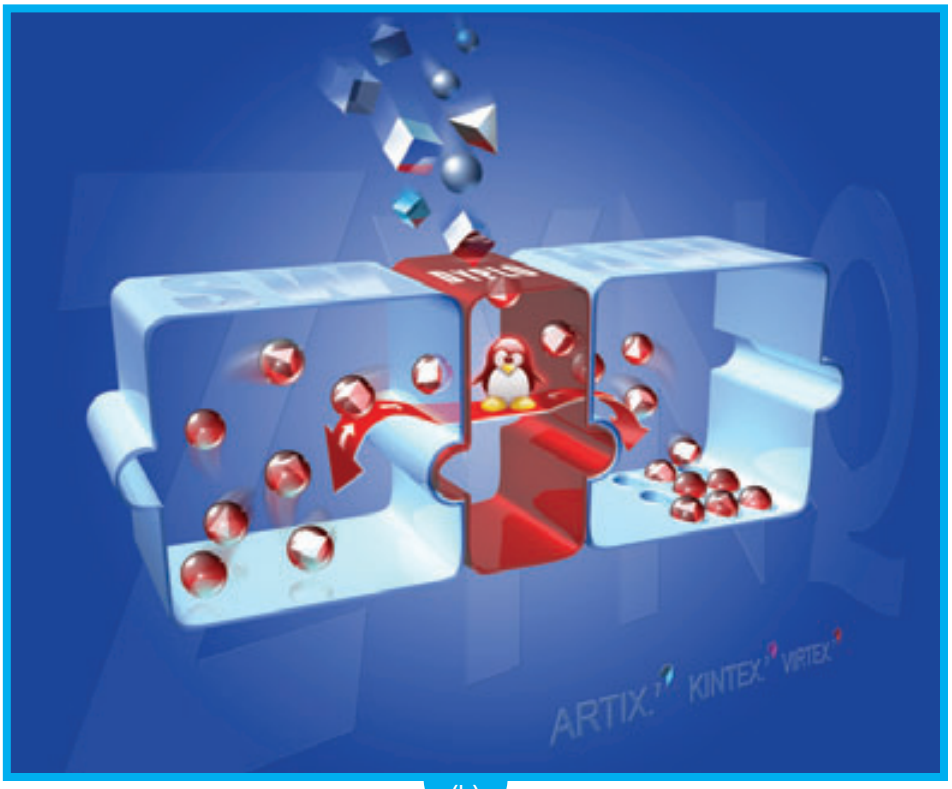
DYPLO: OUR DYNAMIC PROCESS LOADER

Dyplo is a middleware IP block that links scalable data streams via standard AXI interfaces between individual FPGA processes and processor-executed software processes.

The concept behind Dyplo can best be explained from a development perspective. When you start a design project, you must first capture the design requirements, defining the needs of your customer in abstract terms. Changes in requirements are relatively painless in this phase. In the next step, you translate the requirements into a design specification, where you develop a blueprint of the implementation architecture. In this architectural-definition step, you define what functions should go into hardware and software, and set your timing goals. In many cases there are multiple architectures possible, but budgets and time often limit the exploration to a single implementation architecture or, if you are lucky, two of them. Requirement changes can at this level start to become more expensive.



(a)



(b)

Figure 1 – Instead of having two separate processes (a) where hardware designers enter HDL into Zynq SOC's logic blocks and software designers enter software on the processor, Dyplo creates a single point of code entry for both hardware and software functionality (b). This facilitates system-level co-design and verification in a Linux environment.

When you've completed the architectural definition, you begin designing the various functions. This is typically the step in the design process where you discover that you missed some details, justifying an implementation swap from hardware to software or vice versa. Usually such a swap will cause pain in the development effort, as you start solving problems in domains where they should not be—for example, disruptive control functionality in DSP-type data flows. Such scenarios are very common, as the customer's insight evolves over time in tandem with their understanding of the complexities of the design. Therefore, designers need a mechanism to easily move functionality between hardware and software.

Figure 1 illustrates a common arbitrary scheduling scenario in which processes are mapped on FPGA hardware and in software without a common approach (a). A more structured and predictable infrastructure between software and hardware is necessary. Dyplo implements this infrastructure (b).

Dyplo provides users with streaming data interfaces between hardware and software processes. Process communication can be between software processes individually, hardware processes individually and combined hardware/software processes. With Dyplo, you can define the hardware processes in reconfigurable partitions. To enable a given hardware function, you have to activate it on one of those reconfigurable partitions. Then, you initialize the input and output streams to and from that partition and connect those streams to your software processes or other hardware processes. Apart from data stream synchronization, it is also possible to synchronize processes on events such as framing boundaries, discrete sample moments and software task alignments.

The number of software processes is limited to the operating system you are targeting for your design and how the process communication is organized. The number of hardware processes is limited to the number of reconfigurable

partitions reserved on the FPGA. The amount, size and locations of the reconfigurable partitions can be user configurable or automatically chosen.

Dyplo handles the process communication and synchronization by means of a hardware and software communication backplane with scalable performance. For example, the default configuration allows four HDMI video streams to be active simultaneously while maintaining full data-exchange capabilities with the processor.

The Dyplo infrastructure, the functionality mapped on the reconfigurable partitions as well as the generation of a full Linux distribution are realized using a graphical tool flow that is compatible with the standard Vivado® Design Suite project flow.

On the fly, you can stop the software and hardware processes from executing, replace the functionality of the processes and continue executing. In the case of software processes, this is trivial and designers can augment their code limited only by the abilities of the operating system they have chosen for their project. In the case of hardware processes, the partial reconfiguration procedures are executed to replace functionality of a part of the FPGA without compromising the execution of other programmable logic or of the external physical interfaces. The correct loading and verification of the logic module occur automatically.

In this way, Dyplo integrates hardware process execution in a software environment while making the benefits of FPGA functionality seamlessly available to software programmers. The Zynq SoC alone brings a lot of benefits in terms of hardware system-level integration. All features are available for an all-software programmable platform. However, from an application point of view, a middleware layer is required to benefit from this architecture. That is what Dyplo makes possible.

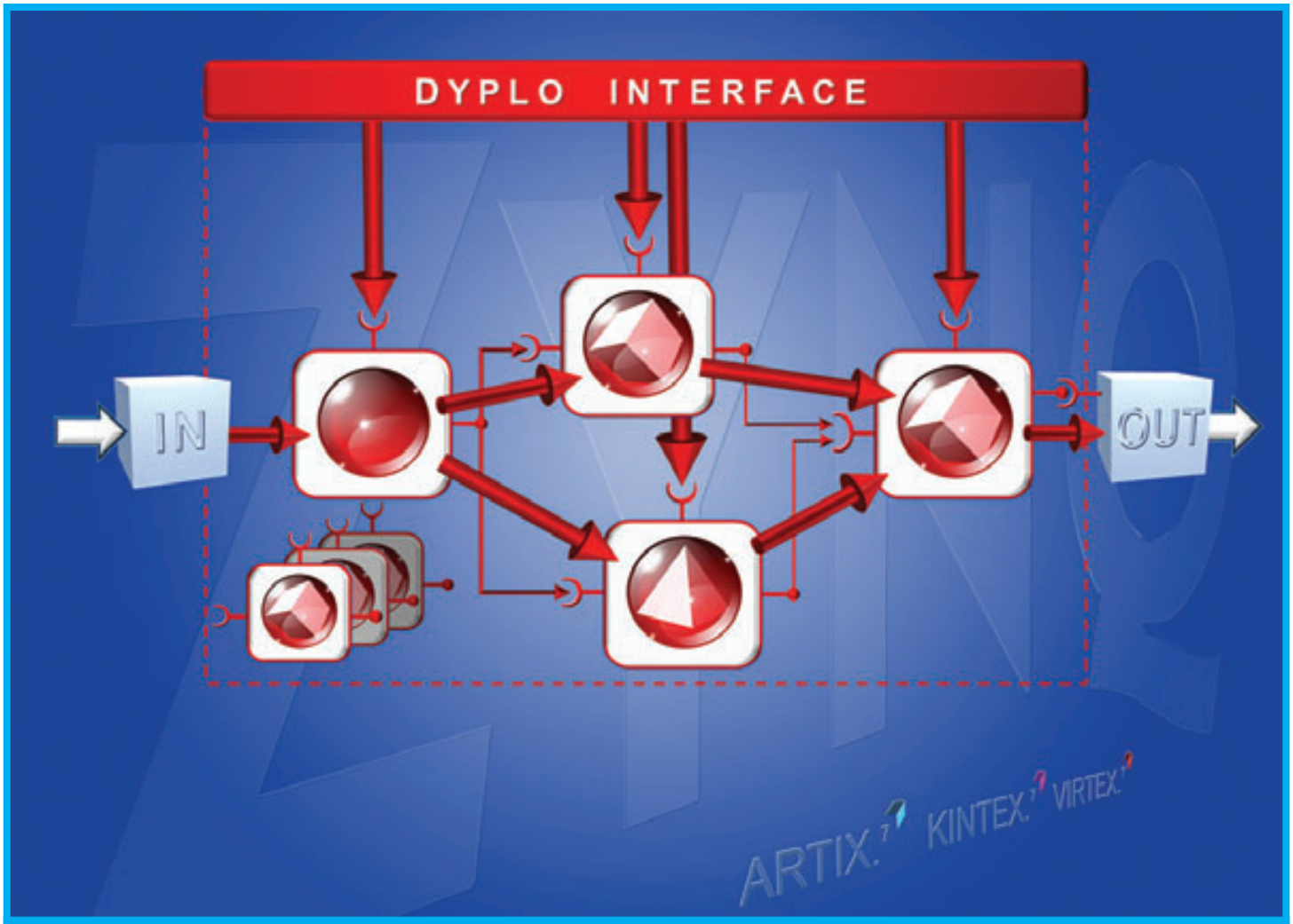


Figure 2 – The balls, cubes and pyramids represent functions or processes that can run on FPGA hardware or in software. The processes are encapsulated in a Dyplo canvas (the squares around the figures). The arrows represent data flowing from the input through the different processes to the output. The Dyplo interface manages this whole process.

PARTIAL RECONFIGURATION AS A COMMODITY

Dyplo makes extensive use of the partial reconfiguration capabilities of the FPGA. For a designer, partial reconfiguration is not a trivial task due to the complexities of loading and managing partial bitstream files and the need to maintain consistency in your design when changing the parts of the FPGA. Also, the placement and routing with respect to physical dependencies require fairly advanced experience in FPGA design.

Dyplo provides out-of-the-box, integrated partial reconfiguration support. Default areas for partially reconfigurable partitions are reserved based on configuration parameters such as the number of required partitions and the logic complexity of a given reconfigurable partition. By providing the HDL descriptions of functionality of such a reconfigurable block, Dyplo automatically generates the partial bit files.

As part of a software application, the reconfigurable block gets its function by issuing a simple software instruction. Dyplo takes care of all required steps to execute the hardware function, such as graceful termination of the currently running function, programming of the new functionality, connection of data streams, configuration of the block and activation of the function.

BENEFITS FOR THE HARDWARE ENGINEER

What does the integration of a Dyplo framework mean for your job as an FPGA design engineer? It makes life easier by handling the integration with the processor environment while minimizing the need to support low-level software. This means that a vast majority of your design effort can be spent on the implementation of functionality. The benefits of FPGA technology are especially effective with high-speed interface design, high-data-bandwidth circuits and accelerated algorithmic processing. Interface and high-bandwidth circuit design will

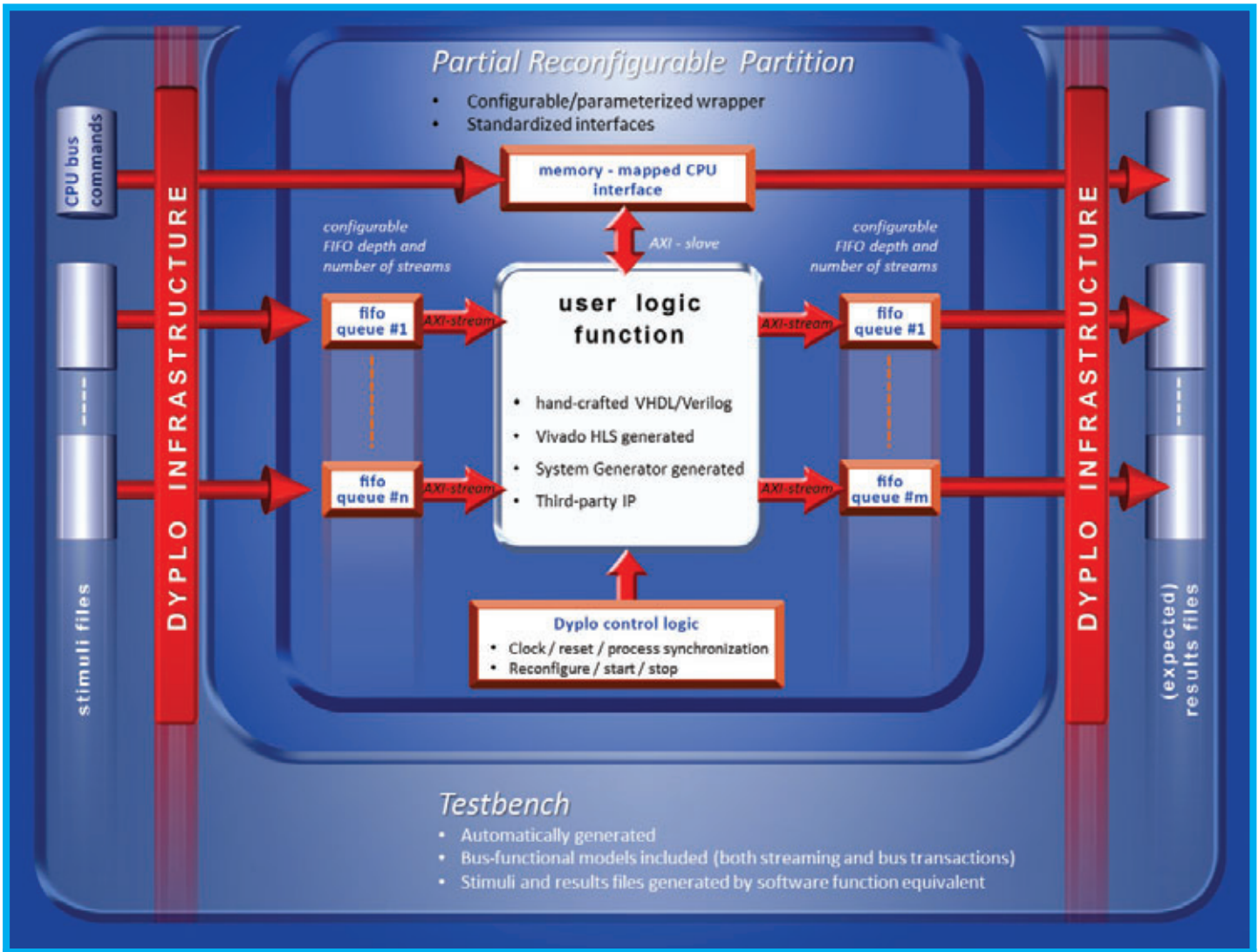


Figure 3 – A functional testbench template implements streaming inputs and outputs while a bus-functional model implements the CPU interface.

always require manual labor and technical insight into the technology benefits of FPGA devices. In the case of accelerated algorithmic processing, system-level synthesis tools like Vivado HLS will start playing an increasingly important role. However, even the use of high-level synthesis requires some design experience to judge the implementation effectiveness.

The Dyplo infrastructure makes it easy to embed algorithmic processing blocks, providing a high-speed configurable interprocess communication network among these algorithmic processing blocks, I/O blocks and the processor-implemented functions. The processing blocks are architected so as to make them suitable for partial reconfiguration. There are clear interfaces to the blocks: AXI4-compliant streaming data interfaces with the high-speed interconnection fabric and a simplified processor bus interface directly with the processing system. Links to other FPGA logic or to the device I/O are implemented using a dedicated, scalable interface. Dyplo has sufficient performance to support, for instance, multiple 1080p input and output video streams and to attach them with little configuration effort.

FPGAS MADE ACCESSIBLE FOR SOFTWARE ENGINEERS

Using Dyplo, design teams can streamline their HDL coding effort significantly, but even more benefits can be found on the software side of system design. The Dyplo method allows teams to isolate software functions for hardware implementation, prototype them in software and then transform them into hardware, maintaining the same interface behavior. Conversely, different functions can share the same FPGA fabric under software control. This approach, which is comparable to software thread behavior, can reduce FPGA cost, as smaller devices are required.

From a software system perspective, the coherence between the individual processes and their connectivity can be considered as a process network with both data and event synchronization mechanisms. Figure 2 illustrates how software and hardware processes are linked with the Dyplo infrastructure.

This method also allows dynamic performance scaling, as multiple instances of the same functionality are possible depending on the system execution context. The result is flexibility of GPU-accelerated process execution, but with much more performance. The technique also makes FPGAs much more accessible to software engineers, as the interface protocol is straightforward and the complexities of the design can be separated into blocks of manageable complexity. More important, Dyplo matches perfectly with Vivado HLS, because the interfaces are compatible—fitting nicely in a software-driven development flow.

Testing is also simpler with Dyplo. The data streamed to or received from an HDL module can be stored in files that the HDL testbench can use for pattern generation. This means that during verification, the team can focus solely on the required functionality.

TEST AND DEBUG ENVIRONMENT

Complex system integration can be quite time-consuming and verification even more difficult. We developed the Dyplo infrastructure bearing in mind that test and integration of designs take a large portion of the development time and are the least appealing part of a development cycle. As it is being configured, the basic Dyplo environment generates testbench templates for verification. For example, for every reconfigurable block in the infrastructure, the Dyplo environment generates a functional testbench template, implementing streaming inputs and outputs from files or regular expressions as well as bus-functional models that implement the CPU interface (see Figure 3). This allows designers to adopt a test-driven design methodology. The infrastructure allows design teams to single-step a design to find issues and perform backplane performance analyses so as to visualize all relevant state information on the fabric. The concept is so flexible that stopping the HDL logic of the Dyplo infrastructure automatically will synchronize software execution. This functionality is critical to identify and solve interface and dynamic problems.

BRIDGING THE GAP

Based in the Netherlands, TOPIC Embedded Systems has been developing embedded systems for many years, often in projects that integrate various types of FPGAs and processors. Over time, we've become experts in advanced FPGA design techniques including partial reconfiguration. The Zynq SoC, as a multiprocessor platform with embedded FPGA technology, increases the effectiveness of embedded system design significantly. But the challenge for many design teams is how to best take advantage of the Zynq SoC's rich feature set given the expertise of designers in either software or hardware engineering.

Our Dyplo middleware bridges this gap. Applications targeted for GPU implementation can be implemented instead on the FPGA fabric, with the same effort and much more flexibility. Using reconfigurability to its full extent makes reuse of FPGA fabric very simple from a software application perspective and contributes to a lower product cost. Automating many of the low-level design, integration and test tasks at the FPGA-design and software-design levels streamlines the design process. In short, the productivity, flexibility and performance optimization enabled by our Dyplo middleware system allow design teams to get the most out of the Zynq SoC and get innovative system designs to market faster. A beta release will be available starting Dec. 1, 2013 for a restricted group of customers. If you want to participate in this program, please contact us. The commercial version will be launched on March 1, 2014.

For more information about using Dyplo in combination with the Zynq SoC, visit *www.topic.nl/en/dyplo* or contact us by phone at +31 (0)499 33 69 79 or by e-mail at *dyplo@topic.nl*.

Easier FPGA Design Using the Power of the Cloud

by J.P. Naude
Hardware and FPGA Designer
CSIR
scineric@csir.co.za

Harnhua Ng
Engineering Team Lead
Plunify
harnhua@plunify.com

Engineers at the CSIR and Plunify tell how they used cloud computing to analyze implementation options and user constraints for a big Virtex-6 design.

As FPGA devices increase in size and the designs living in them increase in density, timing closure is more challenging than ever before. As implementation tools struggle to keep up with the increased complexity, turnaround times between different implementation runs get longer and longer [1]. To speed things up, some designers are looking to use the power of the cloud to help compare and analyze the effects of different implementation options and user-constraint decisions.

At the Council for Scientific and Industrial Research (CSIR) in South Africa, we recently went through such an analysis process. We were able to find the parameters that provided the shortest implementation run-times as well as the best timing scores for our design by using the Plunify cloud-accelerated chip design methodology. Before delving into details of our analysis and the tools we used, let's begin with a bit of background.

BOARD UPGRADE

A recent project required us to upgrade an existing board by replacing the resident Xilinx® Virtex®-5 processing FPGA with a next-generation Virtex-6 device. The first board had several issues, the most important being that the pinout was not optimal for the target application and the device was too small for the requirements of the system.

We were in a privileged position since we had most of the RTL code that needed to run on the new board ready. Thus, it was possible to analyze decisions made during the pinout assignment phase of the PCB schematics design. Due to the long implementation run-times of the FPGA design, we had to limit the number of builds we performed during the PCB design stage in order to make our decisions.

During the hardware-testing phase of the design, we added the missing functionality that did not fit into the original Virtex-5 board. As expected, this created more work for the place-and-route tools, and we went through a detailed floorplanning exercise as well as a large number of runs in order to find the best placement seed. We decided to use the cloud implementation services provided by Plunify in order to manage the large amount of runs we wanted to do.

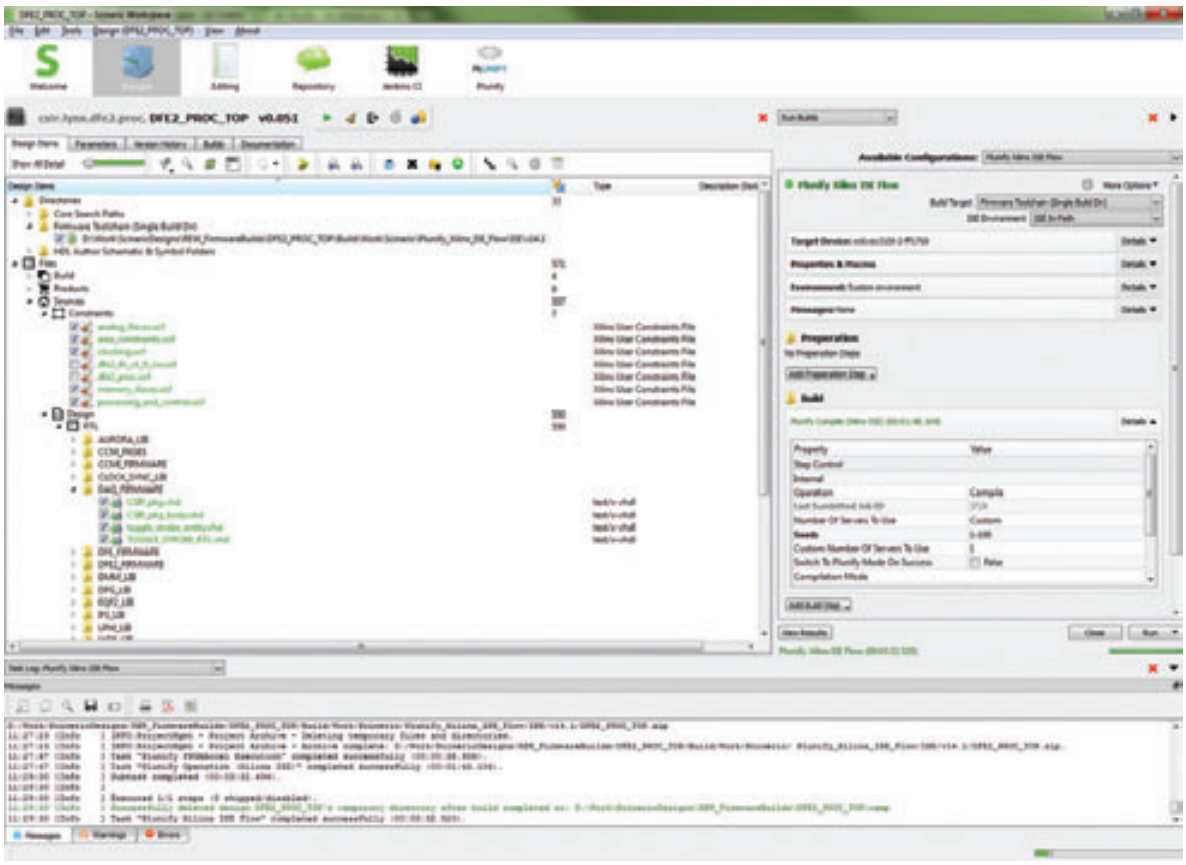


Figure 1 – Scineric Workspace after uploading our project

FPGAAccel is a software API that lets you define, allocate and execute FPGA synthesis and place-and-route builds in parallel on remote servers.

FOUR CONSTRAINT SETS

Because of the processing power that is available when using Plunify, we decided to analyze the final design with four different constraint sets.

The first two sets shared the same pinout, where the first set included no floorplan and the second included a detailed floorplan. We chose the pinout in these first two sets in such a way as to replicate the nonoptimal pinout effects that we experienced in the first revision of the board. The last two sets of constraints shared the chosen pinout of the device on the new version of our board and again, one set included no floorplan and the second included a detailed floorplan.

These sets were required to verify that the changes made to the FPGA's pinouts really made a difference in timing closure, and that a different placement seed on the first-version board alone would not be enough to meet timing specifications. For each set of constraints we ran 100 builds, each with a different placement seed between one and 100.

TOOLS USED

For FPGA design software, we used Xilinx ISE® Design Suite Logic Edition version 14.4 in the designer's local environment for tasks in the existing workflow, and in the cloud for synthesis and place-and-route.

On the front end, CSIR's Scineric Workspace (see Figure 1) is a lightweight and fast new IP-XACT-based integrated development environment for FPGA design management. This tool takes a fresh new "file manager" approach toward managing designs, giving the designer a complete overview of every aspect of a design at all times. Design-merging capabilities make it possible to integrate Scineric into existing workflows where users have a specific design environment they prefer to use. We used Scineric Workspace's graphical interface with our ISE projects to configure, submit and retrieve builds to and from the cloud. This front end is, however, not a hard requirement; the Plunify client also integrates smoothly into both ISE and Vivado® flows.

On the back end, Plunify's FPGAAccel client is a software API that enables engineers to define, allocate and execute FPGA synthesis and place-and-route builds in parallel on remote servers, and to analyze results when all the builds have been completed. In this case, we used a cloud-computing server farm to process the 400 builds. FPGAAccel client is one of the configurations that Scineric Workspace supports.

Figure 2 shows how a designer works with all of these tools. The procedure is as follows:

1. Start Scineric Workspace.
2. Import the ISE project.
3. For each ISE project, define the 100 experiments.

Scineric Workspace allows the user to import designs from numerous sources, including Xilinx ISE and Vivado projects and IP-XACT component-definition files. Each design can have multiple build configurations, including configurations that allow you to define

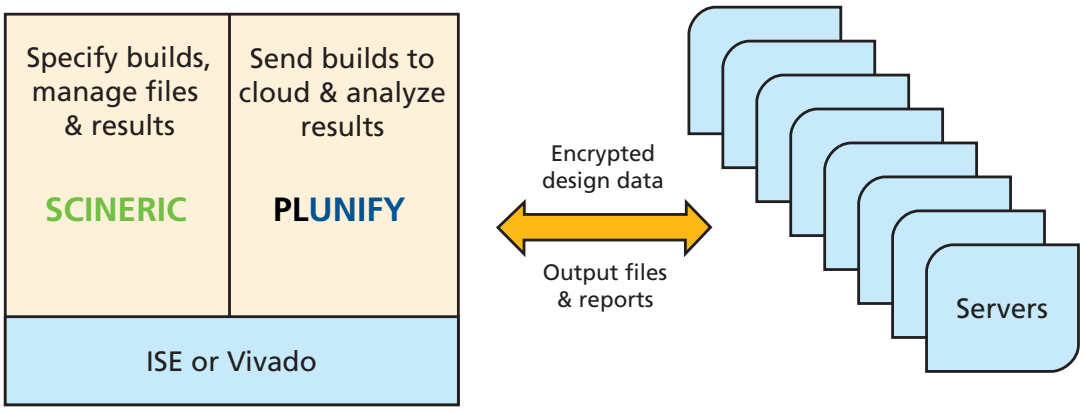


Figure 2 – Flow of data from the user to the cloud and back

Plunify experiments that you can then upload to Plunify.

4. Upload the build request.

The Plunify plug-in first authenticates the designer's identity before encrypting all design files in the ISE project. Next, it securely transmits files along with the build parameters to the remote server farm.

In the cloud, Plunify software generates the required builds, allocates 400 servers and runs ISE to synthesize and place-and-route each of the different experiments on each server.

5. You will receive an e-mail notification when builds are done.
6. Log on to *http://www.plunify.com* to view reports using your job ID.
7. Repeat for each ISE project.

400 BUILDS DONE IN PARALLEL

We used the results of these 400 builds for numerous benchmarks, including the average runtimes, the number of solutions that were unroutable and, most important, the timing scores of the routable solutions. Figure 3 shows the results of the timing-score analysis for the four sets.

Using our four constraint sets, we were again able to show that an inefficient pinout for our target application had severe effects on the timing scores for the design. Furthermore, we were able to determine the best placement seed for the design, and we realized that the implementation tools performed best when not guided by a floorplan for our specific case.

Figure 4 shows the runtimes for the four sets.

Previously, such an effort would have taken more than 30 days to complete, but with the described approach, we obtained the results and analysis in just a day. Another advantage—which we realized only when we saw how much data was generated by the 400 runs (200 Gbytes' worth)—was that Plunify manages this data. All we needed to do was to download the results for the specific runs that we were interested in.

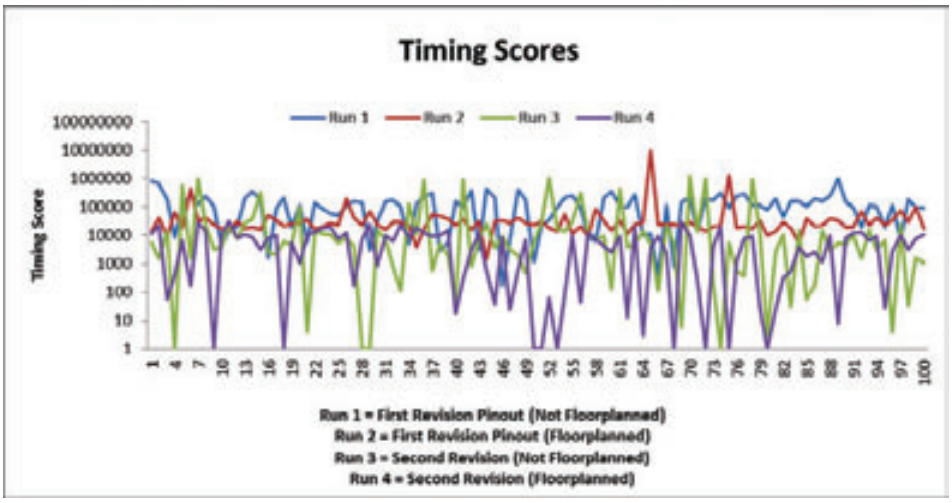


Figure 3 – Timing score vs. placement seed

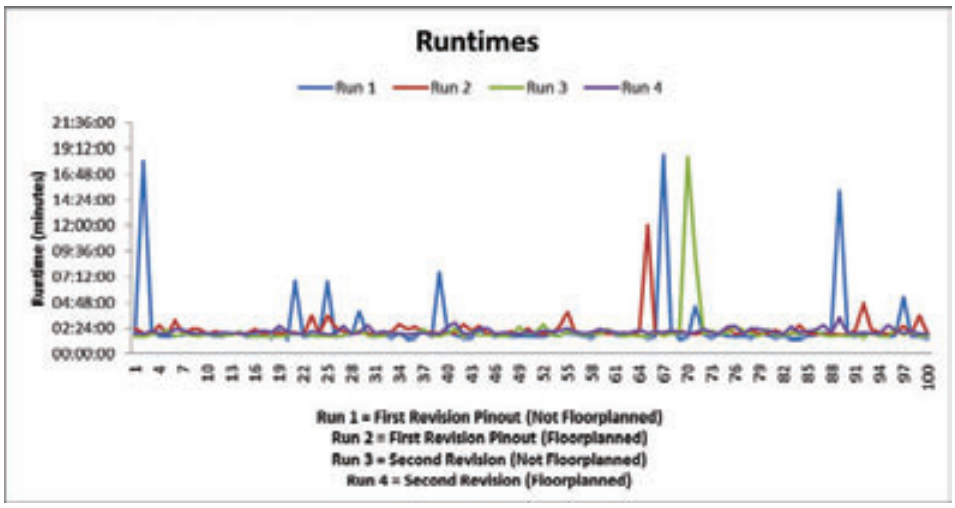


Figure 4 – Runtime vs. placement seed

BEST TIMING SCORE

Timing closure is an important phase of any FPGA design, and we have shown that there are numerous parameters that affect your timing score. Using new tools and the power of cloud computing, we were able to analyze these effects significantly faster than previously possible.

We were able to pin down the constraint set and placement seed that resulted in the best timing score for our design. Using the optimal parameters, we were able to proceed to the next development phase of our system knowing that our builds met their timing specifications.

Reference

1. P. J. Hazewindus, "Managing FPGA Design Complexity: A Software Perspective". Synopsys, Keynote at FPGA World, 2012

What's New in the Vivado 2013.3 Release?

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

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

The Vivado Design Suite 2013.3 is available from the Xilinx Download Center at www.xilinx.com/download.

VIVADO DESIGN SUITE 2013.2 RELEASE HIGHLIGHTS

- IP flow enhancements
- Introducing the UltraFast Design Methodology for the Vivado Design Suite
- Partial Reconfiguration Support
- Hardware Debug Enhancements
- Implementation Tool Enhancements

Device Support

- New device support: Zynq®-7000 7Z015
- The following devices are production ready:
  - Artix®-7 XC7A75T
  - Zynq-7000 XC7Z030 in the SBG485 package, XC7Z015

THE ULTRAFAST DESIGN METHODOLOGY FOR THE VIVADO DESIGN SUITE

In conjunction with the release of Vivado Design Suite 2013.3, Xilinx has launched its UltraFast Design Methodology, which is described in detail in a new best-practices manual called *The UltraFast Methodology Guide for the Vivado Design Suite* (see cover story). Xilinx has integrated many of the processes outlined in the guide into the 2013.3 release. For example, this release includes two new rule decks called Methodology and Timing to better guide users through design cycles. It also includes very handy HDL and XDC constraint templates for correct-by-construction design. *The UltraFast Methodology Guide for the Vivado Design Suite* (UG949) is a free download at *www.xilinx.com/ultrafast*.

VIVADO DESIGN SUITE: DESIGN EDITION UPDATES

Interactive Design Environment

Vivado now includes an easy-to-use dialog for setting, viewing and editing device properties for use with bitstream generation. For example, the Constraint Editor can now detect and report invalid timing constraints, while the Text Editor now includes a file diffing capability that allows users to select two files and visually see the differences between them.

Power

Vivado 2013.3 provides a simpler way to enter designwide power constraints for vectorless power estimation, and export hierarchical design information from Vivado to XPE.

Vivado IP Flows

"Bottom-up" synthesis is now the default flow. A synthesized Design Checkpoint (.dcp) is created by default for all IP except: 7 series MIG, IBERT, ILA, VIO, PCI32, PCI64, Image Stabilization, Object Segmentation, AXI BFM, Zynq BFM.

VIVADO DESIGN SUITE: SYSTEM EDITION UPDATES

Vivado High-Level Synthesis

Vivado HLS now includes new C libraries for FFT and FIR functions to improve system integration and provide best-in-class implementation. These libraries allow users to quickly develop systems with C functions and implement them in high-performance Xilinx IP. Vivado HLS includes improved support for C math libraries with new fixed-point implementations for the popular sin, cos and sqrt functions. Other improvements include full support for AXI4 interfaces in IP for System Generator for DSP. Software integration is made easier with generated software drivers now included in the packaged IP. And Vivado Design CheckPoint format (.dsp) is now supported for packaged IP. Users can direct the tool to synthesize AXI4-Stream interfaces using a single optimization directive.

System Generator for DSP

Xilinx has improved the simulation speed for designs using the DDS and Complex multiplier blocks in System Generator for DSP. The tool now fully supports AXI4-Lite interfaces, which System Generator for DSP can automatically generate with single- or dual-clock support. AXI4-Lite address offsets are generated automatically or can be explicitly defined. The tool automatically creates software driver files for AXI4-Lite interfaces. Users can now incorporate Vivado HLS IP that has AXI4 interfaces directly into System Generator for DSP. Vivado Design CheckPoint format (.dsp) is also now supported as an output format, while interface documentation is now supported for all gateway in and out interfaces.

The customization of compilation targets is made easy through a new MATLAB® API framework. Meanwhile, verification and debug is made easier with the ability to view signals within the current hierarchy and preserve the waveform viewer settings across simulations. Ease-of-use improvements include code generation that now supports user-defined VHDL libraries.

IP FLOW ENHANCEMENTS

This releases introduces the next generation of plug-and-play IP with the following major enhancements.

Bottom-up Synthesis

IP is now synthesized bottom-up, resulting in faster design cycle times.

Third-party simulators IES version 12.20.016 and VCS version H-2013.06-3 are now fully supported through the flow and IP. Single-language simulators are also supported for all IP. Revision control-friendly command options have been introduced (see get_files); all IP comes with a detailed change log. All IP can be upgraded to the new version, and the upgrade process generates an upgrade log.

IP Enhancements

As part of the new IP configuration wizards, designers can share clocking reset resources between multiple instances of an IP core. GT debug ports can also be optionally brought up to the IP interface for ease of GT debug.

Note that these interface-level changes may require a one-time rework when upgrading IP in 2013.3.

PARTIAL RECONFIGURATION

Partial reconfiguration is now available in production status as an add-on option to the Vivado Design Suite. This version supports nonproject Tcl-based flows only for specific 7 series FPGA devices, such as the Kintex®-7, Virtex®-7 T and XT (including 7V2000T and 7VX1140T) and Zynq 7Z045 and 7Z030 SoC. Most standard implementation and bitstream features are now in place, including PR Verify, Reset After Reconfiguration, bitstream compression and encryption, black box bitstreams and more (per-frame CRC checks are not yet available.) The partial reconfiguration flow is enabled with the same license code as the ISE® Design Suite. For more information, see the *Partial Reconfiguration User Guide* (UG909) and the Partial Reconfiguration Tutorial (UG947).

VIVADO PHYSICAL IMPLEMENTATION TOOLS

With the release of Vivado Design Suite 2013.3, Xilinx has improved placement and routing runtime 11 percent compared with the previous release. Other enhancements to the implementation tools in 2013.3 include optional post-placement optimization to improve critical-path timing after placement or after routing, shift register optimization to improve critical-path timing involving SRL primitives, Block RAM enable optimization to improve timing on power-optimized BRAMs and quality-of-results improvements for the phys_opt_design -directive Explore.

TANDEM CONFIGURATION FOR XILINX PCI EXPRESS IP

Tandem Configuration is the Xilinx solution for fast configuration of PCIe® designs to meet enumeration needs within open PCIe systems. New features in 2013.3 include the XC7K160T moving to status, joining the XC7K325T, XC7VX485T and XC7VX690T. The suite supports all packages and PCIe block locations in the IP Catalog for these devices.

Vivado QuickTake Tutorials

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

Vivado Training

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

Latest and Greatest from the Xilinx Alliance Program Partners

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

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

SOC-E HSR-PRP SWITCH IP CORE

http://soc-e.com/products/hsr-prp-switch-ip-core-all-hardware-low-latency-switch-for-fpgas/

The HSR-PRP Switch from Xilinx Alliance member SoC-e (Bilbao, Spain) is an IP core for the implementation of the High-availability Seamless Redundancy and Parallel Redundancy Protocols (HSR and PRP, IEC 62439-3-Clause 5 and 4, respectively) for reliable Ethernet communications. The HSR-PRP Switch is a full hardware solution that can be implemented on a low-cost FPGA. It is an excellent solution for energy market equipment that will be connected to HSR rings or PRP LANs, or will work as network bridges. The core supports gigabit Ethernet switches, frames and high switching speeds. The processing architecture has been designed specifically for HSR/PRP and results in reduced latency. SoC-e has optimized the core in programmable logic to minimize footprint and remove the need for an on-chip microprocessor or software stack.

A2E H.264 MICRO FOOTPRINT CODEC IP FOR SPARTAN-6 AND XILINX 7 SERIES DEVICES

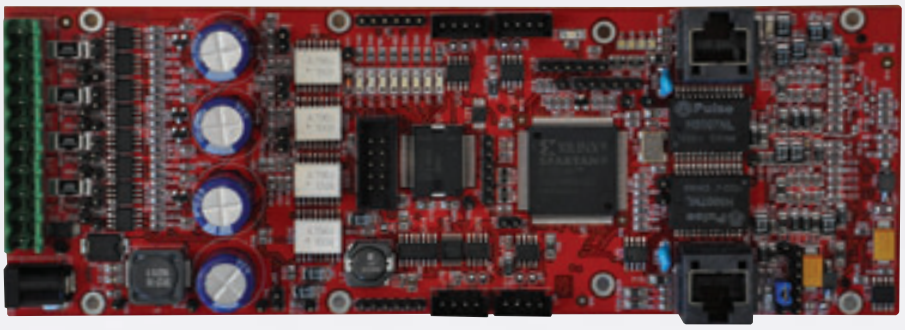
http://www.a2etechnologies.com/products_CODEC.html

The new H.264-15 core from Xilinx Alliance member A2e Technologies (San Diego) represents a huge leap in performance with just a small increase in size over the company's previous-generation core. Encoding and decoding performance have been doubled with just a 30 percent increase in gate count. A single H.264-15 core is capable of compressing or decompressing 1080p30 video in a Spartan®-6 device. Additionally, a more flexible bus interface allows simultaneous and asynchronous access to reference frame memory, video input data and compressed data (in or out). The company has added support for the latest Xilinx devices, such as the Zynq®-7000 All Programmable SoC and Kintex®-7 FPGA, as well as AXI bus support. A complete IP camera can now be created in a single, low-cost Zynq SoC device.

QDESYS MOTION AND MOTOR CONTROL FMC COMPATIBLE WITH ZC702 (NETMOT)

http://www.qdesys.com/netmot-v2.php

The NETMOT from Xilinx Alliance member QdeSys (Nürnberg, Germany) is a networking and motor control daughterboard that plugs into the FMC connector of Xilinx kits. It is intended as a compact and effective interface for advanced networked motor and motion control systems based on a Xilinx SoC or FPGA. The kit includes two Ethernet 10/100/1000 physical-layer interfaces; two power bridges to drive 24-volt permanent magnet motors (PMSM, BLDC, stepper); a sigma-delta analog-to-digital converter with high common mode; and legacy communication such as Profibus or RS485 and CAN. This board has been successfully employed in the new Xilinx Targeted Design Platform concept to realize networked variable-frequency drive and dual-axle networked VFD. Main usage of this board is in creating the infrastructure and proving the concepts for industrial communication and control. The NETMOT accelerates prototyping with Xilinx FPGAs, allowing



The QdeSys NETMOT board

the deployment of all the concepts of the final applications before entering into production, thus reducing the risk of respin.

LOGICIRCUIT OFFERS COMPREHENSIVE PORTFOLIO OF DO-254 CERTIFIABLE IP FOR 7 SERIES

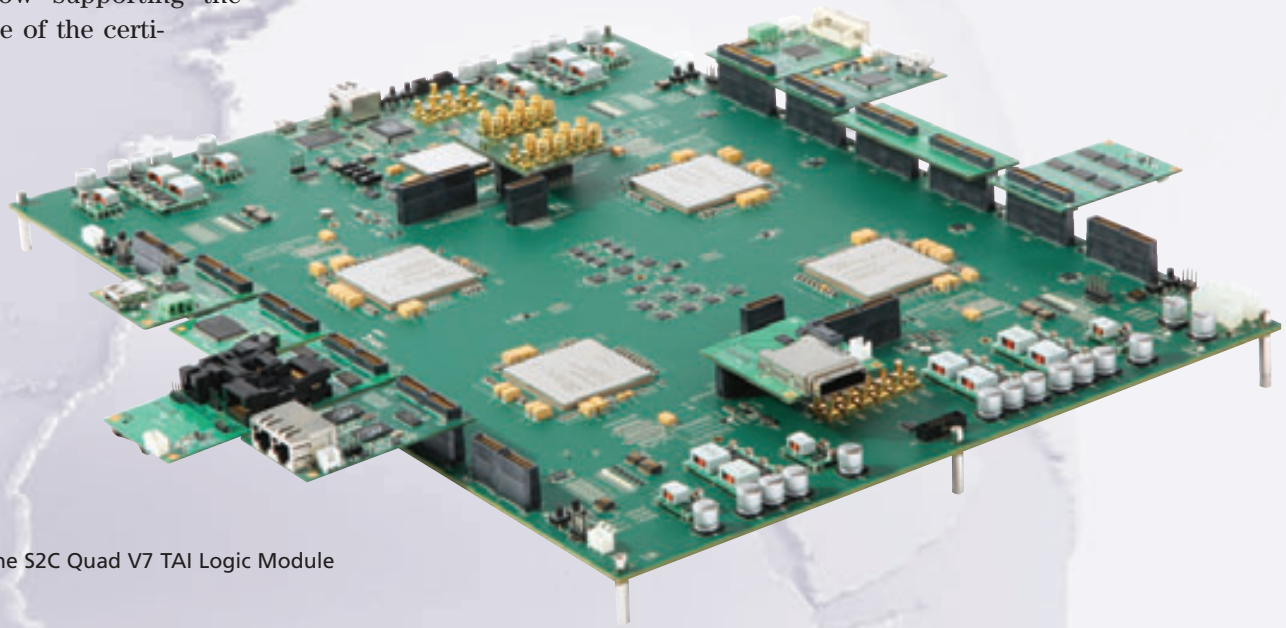
http://logicircuit.com/products

Logicircuit (Alpharetta, Ga.), an expert company focusing exclusively on DO-254 and DO-178B compliance, worked closely with Xilinx to fill a gap in the process of obtaining the required certification on multiple pieces of IP. The company is now supporting the resale of the certifiable IP data packages for the wider avionics customer base. The IP is verified extensively and modified only as necessary to achieve compliance, ensuring full requirements traceability and verification with no unused functionality within the IP cores. Documentation of the compliance process is captured as part of the Certification Data Packet (CDP) that Logicircuit offers to its customers. The end user (that is, the DO-254 applicant) then receives the encrypted IP along with the CDP, which offers proof of compliance to the DO-254 standard.

S2C VIRTEX-7 TAI LOGIC MODULE NOW SHIPPING

http://www.s2cinc.com/product/HardWare/V7TAILogicModule.htm

The Quad V7 TAI Logic Module from Xilinx Alliance member S2C (San Jose, Calif.) can hold designs of up to 80M ASIC gates with four Xilinx Vitex®-7 2000T FPGA devices. The module has two DDR3 SO-DIMM sockets and two DDR2 SO-DIMM sockets onboard to meet a variety of high-speed memory applications. The Quad V7 TAI Logic Module supports 48 channels of high-speed transceivers capable of running at up to 10 Gbps for a variety of high-speed interfaces such as PCIe®, SATA and XAUI. The company's existing USB 2.0 port enables its popular run-time software features such as FPGA download, programmable clock generation and self-test. In addition, the new Quad V7 TAI Logic Module now supports these run-time features through Ethernet cable so that customers can control FPGA hardware remotely. S2C has also added a number of new run-time features such as I/O voltage setting and clock frequency readback through software control.



The S2C Quad V7 TAI Logic Module

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.

XAPP1086: DEVELOPING SECURE AND RELIABLE SINGLE-FPGA DESIGNS WITH XILINX 7 SERIES FPGAS USING THE ISOLATION DESIGN FLOW

http://www.xilinx.com/support/documentation/application_notes/xapp1086-secure-single-fpga-using-7s-idf.pdf

This application note targets FPGA designers working on security or safety-critical designs—specifically, single-chip cryptography, avionics, automotive and industrial applications—using the Xilinx® Isolation Design Flow (IDF). Author Ed Hallett explains how to use IDF to implement isolated functions in a single Xilinx 7 series or 7 series Q (defense-grade) FPGA or in a Zynq®-7000 All Programmable SoC. Functions might include red/black logic, redundant Type-I encryption modules or logic that processes multiple levels of security. Designers will also learn how to verify the isolation using the Xilinx Isolation Verification Tool (IVT).

Xilinx developed the Isolation Design Flow to allow independent functions to operate on a single chip. The ability to develop a safe and secure single-chip solution containing multiple isolated functions is made possible through Xilinx partition technology. Special attributes such as SCC_ISOLATED provide controls to achieve the isolation needed for meeting certifying-agency requirements. With this application note, designers can develop a fail-safe single-chip solution that meets all security requirements for high-grade, high-assurance applications.

The Kintex®-7 FPGA family is currently supported for the IDF. For an in-depth look at the Isolation Design Flow, see also XAPP1085 (*http://www.xilinx.com/support/documentation/application_notes/xapp1085-7s-isolation-design-flow-ise-14-4.pdf*).

XAPP1170: ZYNQ-7000 ALL PROGRAMMABLE SOC ACCELERATOR FOR FLOATING-POINT MATRIX MULTIPLICATION USING VIVADO HLS

http://www.xilinx.com/support/documentation/application_notes/xapp1170-zynq-hls.pdf

Matrix multiplication is used in nearly every branch of applied mathematics. For example, it is essential in beamforming, which is the process of phasing a receiving antenna digitally by computer calculation in modern radar systems. This application note describes how to use Vivado® high-level synthesis (HLS) to develop a floating-point matrix-multiplication accelerator with an AXI4-Stream interface and connect it to the Accelerator Coherency Port (ACP) of the ARM® CPU in the Zynq-7000 All Programmable SoC. Vivado HLS can quickly implement the accelerator, which is modeled in the C/C++ code, and optimize it into an RTL design. The solution is then exported as a pcore connected with an automatically created AXI4-Stream interface to the ACP.

Authors Daniele Bagni, Juanjo Noguera and Fernando Martinez Vallina used Xilinx Platform Studio (XPS) to design the programmable logic hardware, including the matrix-multiplier peripheral, the DMA engine and an AXI timer. They used the Software Development Kit (SDK) to design the processing system software to manage the peripherals. The driver application is a 32x32 matrix-multiplication core optimized for 32-bit floating-point accuracy using Vivado HLS.

XAPP1160: AXI CHIP2CHIP REFERENCE DESIGN FOR REAL-TIME VIDEO APPLICATION

http://www.xilinx.com/support/documentation/application_notes/xapp1160-c2c-real-time-video.pdf

The LogiCORE™ IP AXI Chip2Chip is a soft Xilinx core that bridges between AXI systems for multidevice system-on-chip solutions. This application note by Saambhavi Vajjiravelu Baskaran and Vamsi Krishna provides a setup demonstrating real-time video traffic across Kintex-7 FPGA and Zynq-7000 All Programmable SoC boards. The AXI Chip2Chip core provides connectivity across two Xilinx boards using FMC connector cables.

The reference design includes two embedded systems created with the Xilinx Platform Studio (XPS) v14.5 tool, which is part of the ISE® Design Suite: System Edition. The axi_chip2chip_v3_00_a core implements a video system in which the Test Pattern Generator (TPG) creates test patterns. Two instances of the core are instantiated, one as a master and one as a slave. Complete XPS and SDK project files are provided.

XAPP1164: 7 SERIES FPGA AXI MULTIPORT MEMORY CONTROLLER USING THE VIVADO IP INTEGRATOR TOOL

http://www.xilinx.com/support/documentation/application_notes/xapp1164.pdf

A Multiport Memory Controller (MPMC) is a common requirement in many video, embedded and communications applications where data from multiple sources moves through a common memory device, typically DDR3 SDRAM. In this application note, authors Khang Dao, Sikta Pany and Ricky Su demonstrate how to create a basic DDR3 MPMC design using a Xilinx 7 series FPGA and the IP Integrator tool within the Vivado Design Suite by combining the Memory Interface Generator (MIG) core and the AXI interconnect IP, both of which are provided in the Vivado tools.

The accompanying reference design for a basic video system uses the AXI4, AXI4-Lite and AXI4-Stream interfaces. These interfaces provide a common IP interface protocol framework for building the system. The example design is a full, working hardware system on the KC705 evaluation board in the Kintex-7 FPGA KC705 Evaluation Kit. AXI VDMA IP cores move video frames through DDR3 memory from a video test pattern generator IP block to an HDMI display IP block.

XAPP1178: DISPLAYPORT TRANSMIT REFERENCE DESIGN

http://www.xilinx.com/support/documentation/application_notes/xapp1178-displayport-transmit.pdf

This application note by Vamsi Krishna and Saambhavi Baskaran shows how to implement a LogiCORE IP DisplayPort system that includes policy maker features and a DisplayPort controller. The reference design, created and built using the Vivado Design Suite 2013.2, uses audio and video pattern generators to generate the traffic for testing. The document includes instructions for building the hardware and testing the design on the board with the provided C source code. Complete Vivado and SDK project files are provided.

The design showcases a system transporting video and audio data from the Xilinx DisplayPort transmit core to a DisplayPort-capable monitor. A DisplayPort source policy maker is implemented on the KC705 Evaluation Board, which includes the MicroBlaze™ processor and DisplayPort core, as well as the video and audio pattern generators.

XAPP1175: SECURE BOOT OF ZYNQ-7000 ALL PROGRAMMABLE SOC

http://www.xilinx.com/support/documentation/application_notes/xapp1175_zynq_secure_boot.pdf

The Zynq-7000 All Programmable SoC integrates both a processor-based system-on-chip (SoC) and programmable logic. Unlike in previous Xilinx devices, the boot mechanism is processor driven. This application note provides the concepts, tools and methods to implement a secure boot. The document shows how to create a secure embedded system and how to generate, program and manage the public and private cryptographic keys. Author Lester Sanders demonstrates how to boot the Zynq SoC device securely using QSPI and SD modes. He also describes the optimal use of RSA authentication and AES encryption for different security requirements, and provides a method of handling RSA keys securely. Multiboot examples show how to boot a golden image if the boot of an image fails. Examples also explain how to generate and program keys.

XAPP1165: USING VIVADO DESIGN SUITE WITH VERSION CONTROL SYSTEMS

http://www.xilinx.com/support/documentation/application_notes/xapp1165.pdf

This application note provides recommendations for using version control systems with the Vivado Design Suite in both Non-Project Mode and Project Mode. Author Jim Wu recommends using Vivado's Manage IP feature for IP management, while also keeping design sources as remote files. Wu favors using Tool Command Language (Tcl) scripts to manage design sources, design runs and project creation. An accompanying example design follows this methodology from register-transfer level (RTL) to bitstream. Tcl scripts for creating an initial design repository are included, using Git in both Non-Project Mode and Project Mode.

XAPP1093: SIMPLE AMP: ZYNQ SOC CORTEX-A9 BARE-METAL SYSTEM WITH MICROBLAZE PROCESSOR

http://www.xilinx.com/support/documentation/application_notes/xapp1093-amp-bare-metal-microblaze.pdf

The Zynq-7000 All Programmable SoC contains two Cortex™-A9 processors that can be configured to concurrently run independent software stacks or executables. The programmable logic within the Zynq SoC can also contain MicroBlaze embedded processors. This application note by John McDougall describes a method of starting up one of the Cortex-A9 processors and a MicroBlaze processor, each running its own bare-metal software application, and allowing them to communicate through shared memory by means of asymmetric multiprocessing.

AMP is a mechanism that allows multiple processors to run their own operating systems or bare-metal applications, with the possibility of loosely coupling those applications via shared resources. The reference design includes the hardware and software necessary to run one Cortex-A9 processor and one MicroBlaze in an AMP configuration. The second Cortex-A9 processor (CPU1) is not used in this design. Each CPU runs a bare-metal application within its own standalone environment. The design takes steps to keep the CPUs from conflicting on shared hardware resources. This application note also describes how to create a bootable solution and how to debug both CPUs. The design is built using Xilinx Platform Studio (XPS) 14.5.

XAPP1172: USING THE ZYNQ-7000 PROCESSING SYSTEM (PS)-TO-XILINX ANALOG-TO-DIGITAL CONVERTER (XADC) DEDICATED INTERFACE TO IMPLEMENT SYSTEM MONITORING AND EXTERNAL CHANNEL MEASUREMENTS

http://www.xilinx.com/support/documentation/application_notes/xapp1172_zynq_ps_xadc.pdf

The XADC-to-PS interface is a dedicated link present in Zynq-7000 All Programmable SoCs that enables connectivity between the device's processing system and XADC block without requiring building of a hardware bitstream. Designers can use the interface for system-monitoring applications where a CPU needs to monitor the on-chip voltage and temperature along with external voltage or current channels. The XADC provides alarm events that are set based on user-preconfigured values. The alarms can trigger system-level actions such as shutdown in case of overtemperature.

In this application note, authors Pallav Joshi, Srinivasa Attili and Mrinal J. Sarmah explore the system-monitoring aspects of the dedicated XADC-to-PS interface along with event programming and monitoring. Their design also explores the possibility of using an external auxiliary channel through the XADC-to-PS interface, and characterizes the maximum signal frequency that can be monitored using the interface. Designers can use the external auxiliary channel to monitor external voltage and current that is not captured by internal sensor. The latency number provided in this application note is the best that can be achieved through this interface; it might vary depending on CPU load conditions.

XAPP 1081: QUICKBOOT METHOD FOR FPGA DESIGN REMOTE UPDATE

http://www.xilinx.com/support/documentation/application_notes/xapp1081-quickboot-remote-update.pdf

A primary advantage of an All Programmable FPGA is its remote-update capability, making it possible to update deployed systems with design patches or enhanced functionality. This application note by Randal Kuramoto provides a reliable field update through a combination of a fast, robust configuration method and an efficient HDL-based, in-system programming reference design in a solution known as the QuickBoot method.

The QuickBoot method places the responsibility for programming error/interrupt recovery on the programming operation via a simple adjustment to the programming algorithm for the bitstream update process. QuickBoot integrates the programming method with a configuration method based on a special configuration header to form the remote update solution. This solution is robust, compatible with many configuration setup variations and quick to configure in all cases.

XAAPP1167: ACCELERATING OPENCV APPLICATIONS WITH THE ZYNQ-7000 ALL PROGRAMMABLE SOC USING VIVADO HLS VIDEO LIBRARIES

http://www.xilinx.com/support/documentation/application_notes/xapp1167.pdf

Computer vision is a field that broadly includes many interesting applications, from industrial monitoring systems that detect improperly manufactured items to automotive systems that can drive cars. Many of these systems are built or prototyped using OpenCV, a library that contains optimized implementations of common computer vision functions targeting desktop processors and GPUs. OpenCV applications can be used in embedded systems by recompiling them for the ARM architecture and executing them in Zynq-7000 All Programmable SoCs, as Stephen Neuendorffer, Thomas Li and Devin Wang explain in this application note.

Designers can use OpenCV at many points in the design process, from algorithm prototyping to in-system execution. OpenCV code can also migrate to synthesizable C++ code using video libraries that are delivered with Vivado high-level synthesis (HLS). When integrated into a Zynq SoC design, the synthesized blocks make it possible to implement high-resolution and high-frame-rate computer vision algorithms.

XAPP1082: PS AND PL ETHERNET PERFORMANCE AND JUMBO FRAME SUPPORT WITH PL ETHERNET IN THE ZYNQ-7000 ALL PROGRAMMABLE SOC

http://www.xilinx.com/support/documentation/application_notes/xapp1082-zynq-eth.pdf

The focus of this application note is on Ethernet peripherals in the Zynq-7000 All Programmable SoC. Authors Srinivasa Attili, Sunita Jain and Sumanranjan Mitra explain how to use the processing system (PS)-based gigabit Ethernet MAC through the extended multiplexed I/O (EMIO) interface with the 1000BASE-X physical interface using high-speed serial transceivers in programmable logic (PL). The document also describes how to implement PL-based Ethernet supporting jumbo frames. The accompanying reference designs enable the use of multiple Ethernet ports, and provide kernel-mode Linux device drivers. In addition, the application note includes Ethernet performance measurements with and without checksum offload support enabled. The test results show a trend of throughput improvement with increasing packet size.

XAPP1176: EXECUTE-IN-PLACE (XIP) WITH AXI QUAD SPI USING THE VIVADO IP INTEGRATOR

http://www.xilinx.com/support/documentation/application_notes/xapp1176-xip-axi-quad-spi-ipi.pdf

This application note by Sanjay Kulkarni and Prasad Gutti describes the execute-in-place (XIP) feature introduced in the AXI Quad SPI v3.0 IP core, released in the Vivado

Design Suite v2013.2. The document provides information about the required connections to configure the FPGA from an SPI serial flash device, as well as the configuration flow for the SPI mode.

XIP is a method of executing programs directly from long-term storage rather than copying them into Block RAM. In embedded systems, flash memory is preferred over DDR for this purpose. Low power and a smaller pin interface count are additional benefits of flash memory. For "boot from flash" operation, SPI-based flash memory is the best choice. The executable code residing in the SPI flash is loaded into DDR through an XIP-configured Quad SPI IP core to demonstrate the store-and-load feature of the XIP mode implemented.

XAPP1092: IMPLEMENTING SMPTE SDI INTERFACES WITH ZYNQ-7000 ALL PROGRAMMABLE SOC GTX TRANSCEIVERS

http://www.xilinx.com/support/documentation/application_notes/xapp1092-smpte-sdi-zynq-gtx-transceivers.pdf

The Society of Motion Picture and Television Engineers (SMPTE) serial digital interface (SDI) family of standards is widely used in professional broadcast video equipment. In broadcast studios and video production centers, SDI interfaces 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 by John Snow provides a module containing control logic to couple the SMPTE SDI LogiCORE IP with the Zynq-7000 All Programmable SoC GTX transceivers, to form a complete SDI interface. The GTX device-specific control logic necessary to use the transceivers in SDI applications is included, along with two detailed SDI demonstration application examples in a Zynq SoC design. The example SD/HD/3G-SDI designs run on the ZC706 evaluation board.

XAPP1169: VIDEO-OVER-IP WITH JPEG2000 REFERENCE DESIGN

http://www.xilinx.com/support/documentation/application_notes/xapp1169-VOIP-JPEG2000-ref-design.pdf

This application note describes a video-over-IP reference design that integrates the Xilinx SMPTE 2022-5/6 LogiCORE IP cores and Barco-Silex JPEG2000 encoder and decoder IP cores. The design, which supports up to four SD/HD-SDI streams, is built on two platforms: transmitter and receiver. According to authors Jean-François Marbehant and Virginie Brodeoux, the transmitter platform design uses four LogiCORE IP triple-rate SDI cores to receive the incoming SDI video streams. The SMPTE 2022-5/6 video-over-IP transmitter core multiplexes three of those SDI streams, encapsulates them into fixed-sized datagrams and then sends them out through the LogiCORE IP 10-Gbit/second Ethernet MAC (10GEMAC). The JPEG2000 encoder compresses the fourth SDI stream, encapsulates it into fixed-sized datagrams and sends it out through the LogiCORE IP trimode Ethernet MAC (TEMAC) to a standard Cat.5e cable.

On the receiver platform, the Ethernet datagrams of the uncompressed streams are collected at the 10GEMAC. The video-over-IP receiver core filters the datagrams, de-encapsulates and demultiplexes them into individual streams, and outputs the SDI video through the triple-rate SDI cores. The Ethernet datagrams of the compressed streams are collected at the TEMAC, de-encapsulated and fed to the JPEG2000 decoder. Its output video is converted to SDI and sent to the triple-rate SDI cores. All the Ethernet datagrams are buffered in a DDR3 SDRAM for both the transmitter and receiver. A MicroBlaze processor initializes the cores and reads the status.

XAPP1096: DC COUPLING WITH 7 SERIES FPGAS' GTX TRANSCEIVERS

http://www.xilinx.com/support/documentation/application_notes/xapp1096-dc-coupling-gtx-transceivers.pdf

Transceiver usage in electronics design generally involves AC-coupled links between the transmitter and receiver. But there are drawbacks to AC-coupled links, including discontinuities and routing congestion created by connections to the AC coupling capacitors; an increase in board area due to AC capacitor placement; and the loss of AC swing across the capacitor. In addition, the run length is limited with AC coupling and the technique may degrade low-frequency signal content.

Designers can mitigate these drawbacks by using a DC link between the transmitter and receiver. In this application note, authors Vaibhav Kamdar, Mustansir Fanaswalla and Santiago Asuncion explain how to use the Xilinx 7 series FPGAs' GTX transceivers for this purpose. The document explores utilizing the GTX transceiver as a receiver and as a transmitter. It also describes communication between two GTX transceivers. In an AC-coupled system, for a typical current-mode logic (CML) transceiver with on-die termination, the common mode at the RX input is dictated by the RX termination voltage, while the common mode of the TX is dictated by the TX termination voltage and the output swing. In a DC-coupled system, the common-mode voltage of the link is typically determined by the TX termination voltage, the output swing and the RX termination voltage.

Xpress Yourself in Our Caption Contest

DANIEL GUIDERA

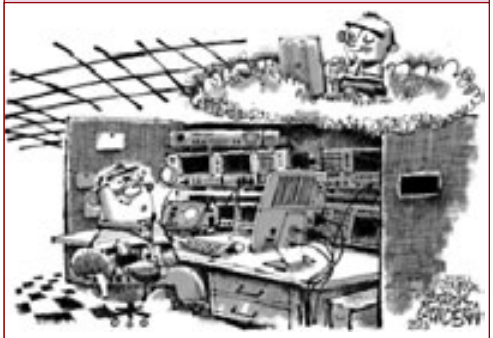
If you've ever volleyed ideas back-and-forth at team meetings, you'll appreciate this peek behind the scenes at one top-level confab. Xercise your funny bone as well as your paddle arm by submitting an engineering- or technology-related caption for this cartoon showing a couple of engineers using table tennis to inspire new concepts. The image might inspire a caption like "The new VP of marketing didn't have a technical background, but his reputation as head of development at Dave's Sporting Goods preceded him."

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

The contest begins at 12:01 a.m. Pacific Time on Oct. 28, 2013. All entries must be received by the sponsor by 5 p.m. PT on Jan. 6, 2014.

So, put down your paddle and get writing!

MASON DONAHUE, implementation engineer at GEOcommand Inc. (Ingleside, Ill.), won a shiny new Avnet ZedBoard with this caption for the engineer-on-a-cloud cartoon in Issue 84 of *Xcell Journal*:



"You see, timing is an illusion. Our design must simply find peace within itself."

Congratulations as well to our two runners-up:

"Hardware? Who needs hardware? We do everything in the cloud, now!"

– Tom Burke, project engineer, Oceaneering (Pasadena, Md.)

"After many years of believing the whole digital world was really analog, Joe came to the realization that everything is now based on software in the clouds."

– Boris Shajenko, principal engineer, FPGA Works, LLC (Littleton, Mass.)

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

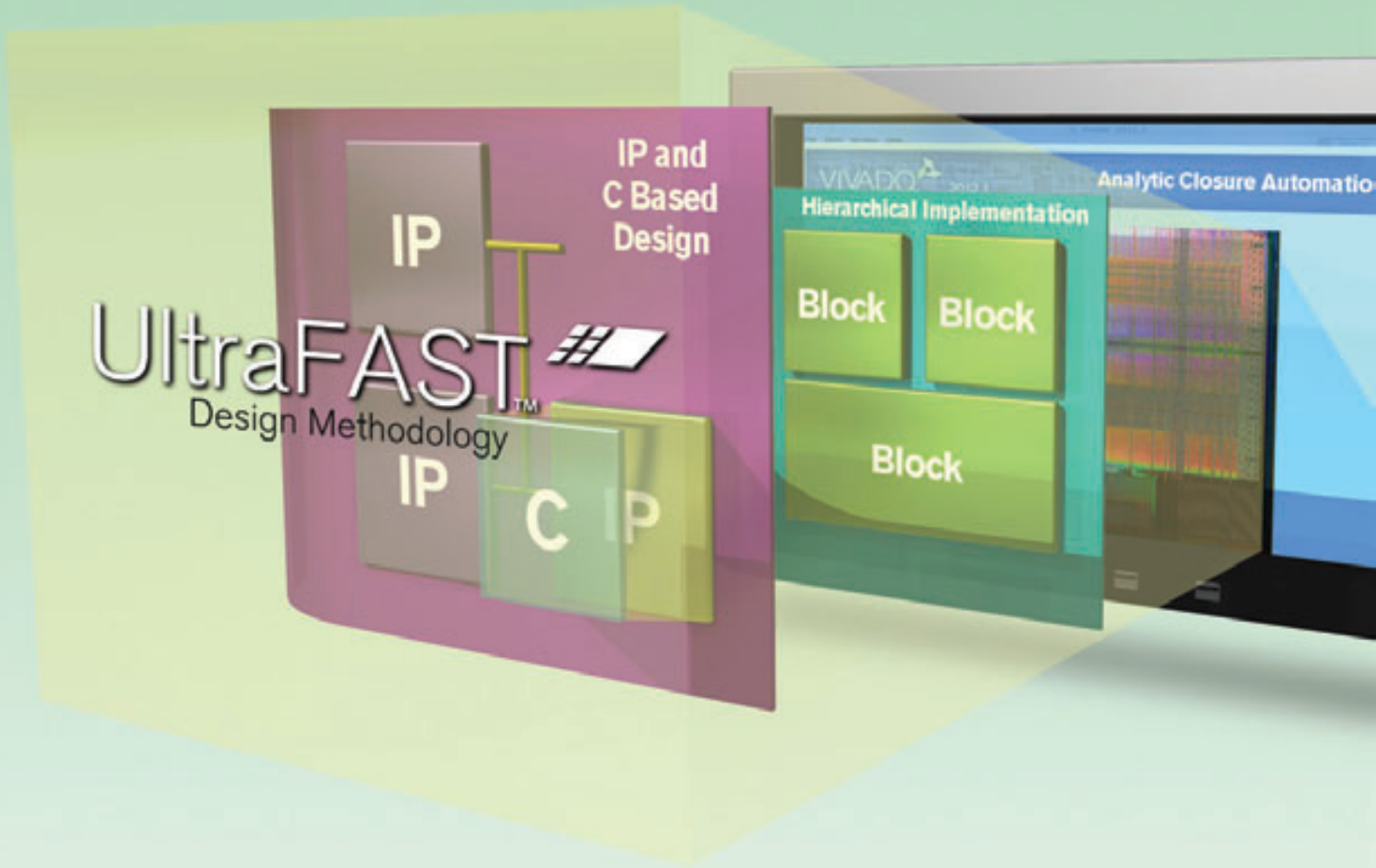
FPGAs keep getting bigger, but your schedule is not. There is no time to waste on numerous design iterations and long tool runtimes. Use the hierarchical and incremental techniques available in Synopsys' Synplify® software to bring in your schedule and meet aggressive performance goals.

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

SYNOPSYS®
Accelerating Innovation

Xilinx Introduces

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



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

VIVADO

XILINX
ALL PROGRAMMABLE.

Learn More: www.xilinx.com/ultrafast