SOLUTIONS FOR A PROGRAMMABLE WORLD

Profits Proliferate on Zynq SoC Product Platforms

XILINX

ALL PROGRAMMABLE™

www.xilinx.com/xcell

Xilinx® SpeedWay Design Workshops™

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

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

em.avnet.com/xilinxspeedways

DESIGNED BY **AVNET**

All Programmable Platforms: Foundation for Profitability

When I first started covering the IC design industry as a trade journalist, the merchant ASIC market had already had its heyday and the custom digital IC business was rapidly turning to ASSP SoCs as a way to improve profit margins. Merchant ASICs came to dominance when differentiation of an end product's performance and feature set relied mostly on its hardware—performance, power as well as unique functionality hardwired into the gates of a device. But the merchant ASIC business was short-lived. Why?

Tom Hart, chairman and former CEO of Quicklogic, summarized it best at meeting we had years ago. "The key to making money in the semiconductor business is selling a lot of one type of chip to a lot of different customers," said Hart. "The ASIC business is a crummy business. It is a bet on a bet. You have to bet that the customer you are building an ASIC for has built the right product for the right market."

By the early 2000s, silicon process technology and gate counts had advanced to the point where companies could embed microprocessors and other IP into their custom digital designs, creating what quickly became known as systems-on-chip. These SoCs allowed semiconductor companies to build a single device and sell it to a broader number of customers—a business we call today merchant ASSPs. With ASSPs, the on-chip hardware typically meets a minimum hardware requirement and any and all customer differentiation occurs in software. While still popular, the ASSP business model also has flaws—but mainly for the customer.

The biggest problem is that semiconductor vendors generally do not build an off-the-shelf ASSP until standards are nailed down and a market is already established. So if you as a customer want to be first to market, get the highest product ASP and maximize profitability, you still need to create custom hardware as well as software for your own differentiated chips. Further, to maximize the time you dominate the market and maintain your first-to-market price point, you need end-product differentiation. This fact points to yet another flaw in the off-the-shelf ASSP model: If you can buy it from a vendor, someone else can too. Arguably, it is relatively easy and fast to differentiate the software functionality once your ASSP is available. But it is also relatively easy for your competitors to create lower-cost knockoffs of your design or even improve it using the exact same hardware. As such, ASSPs have proven great for companies that want to build "me-too" products and get the leftovers of a given market opportunity's carcass before it is completely picked clean.

In the face of daunting silicon costs, many companies are turning to a platform business model to maximize profitability. That is, you create a first, custom chip at a given silicon process node and then build less-expensive derivatives leveraging IP and design reuse. Companies can build platforms with ASICs, their own ASSPs or merchant ASSPs, but those options are still weighed down by issues I pointed out above. So a growing number of customers are taking the next step in the semiconductor evolution and building platforms with Xilinx®'s award-winning Zynq®-7000 All Programmable SoC. As you will read in the cover story, the Zynq SoC is by far the wisest business as well technological choice available today for building a differentiated product platform and maximizing bottom-line profitability.

Mike Santarini
Publisher

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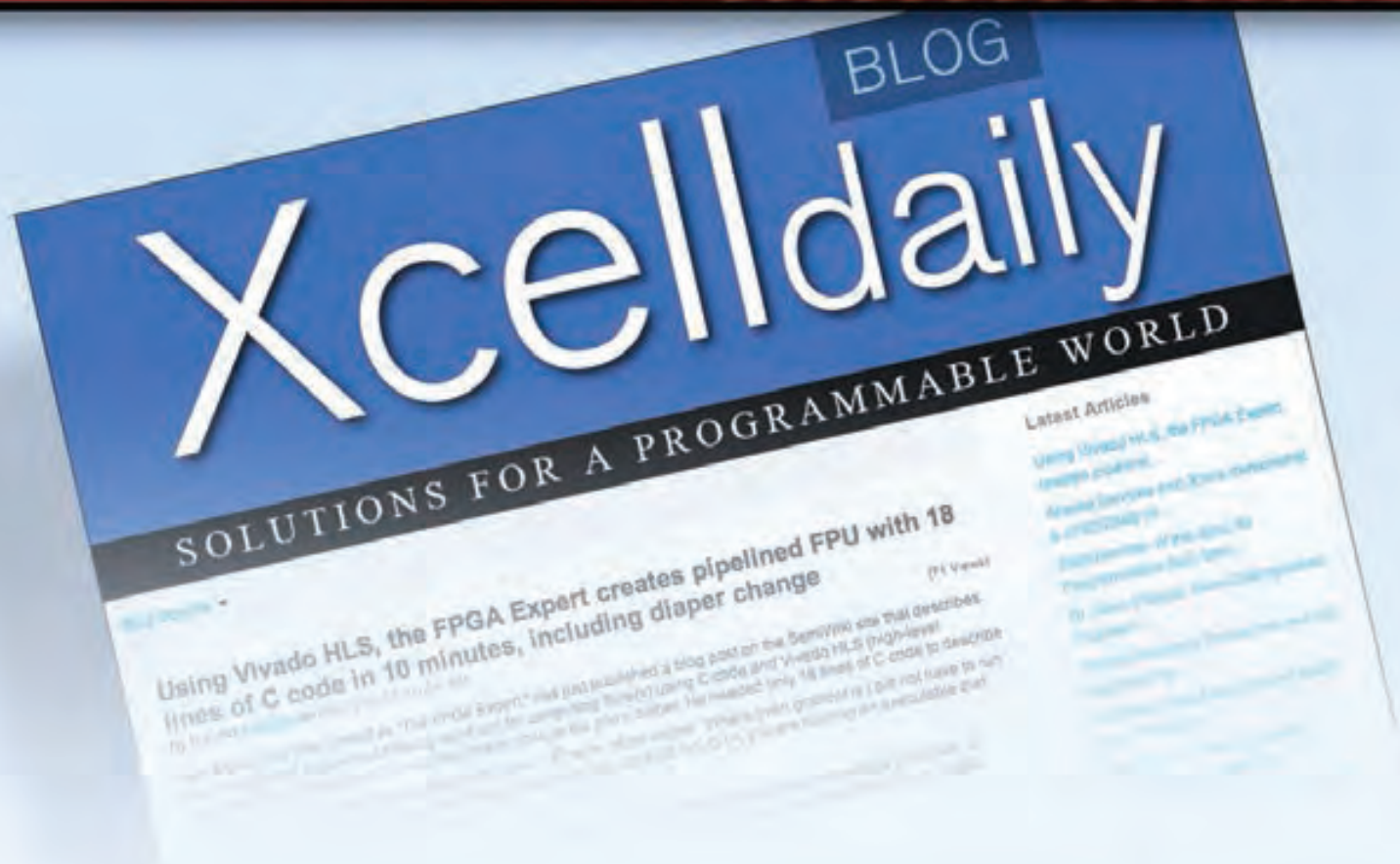
Xilinx has extended the Award Winning Journal and added an exciting new *Xcell Daily Blog*. The new site provides dedicated readers with a frequent flow of content to help engineers leverage the flexibility and extensive capabilities of Xilinx products, ecosystem, and customers to create All Programmable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

Recent

■ *How to Use High Speed ADCs and DACs with FPGAs*

■ *Low-cost Artix-7 50T FPGA Eval Kit now ready to order from Avnet*

■ *New Zynq-based RazerCam is one smart industrial machine vision camera with three different sensor options*

■ *Adapteva's Zynq-based parallel supercomputer: an update*

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

XILINX
ALL PROGRAMMABLE.

CONTENTS

18

24

30

Cover Story

THE XILINX XPERIENCE FEATURES

38

52

44

XTRA READING

Products, Profits Proliferate on Zynq SoC Platforms

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

Customers find they can maximize revenue by building their product portfolios around Xilinx's Zynq SoC.

E

Ever since Xilinx® shipped the Zynq®-7000 All Programmable SoC in late 2011, a bounty of products has been arriving. Today the Zynq SoC is at the heart of many of the world's newest and most innovative automotive, medical and security vision products, as well as advanced motor-control systems that make factories safer, greener and more efficient. The Zynq SoC has also won sockets in next-generation wired and wireless communications infrastructure equipment as well as a wealth of emerging Internet of Things applications.

Having experienced firsthand the unmatched versatility of a device that integrates a dual-core ARM® Cortex™-A9 MPCore processor with programmable logic and key peripherals all on the same chip, a growing number of customers are expanding their use of the Zynq SoC from the processor of choice for one socket to the platform choice for entire product lines. By deploying a platform strategy leveraging the Zynq SoC and hardware/software reuse, they are able to quickly create many derivatives or variations of their products. The result is higher levels of design productivity and an improvement in the bottom line.

Let's look at what practices top platform-electronics companies employ to improve profitability; why the Zynq SoC is far superior to ASIC, standalone ASSP and even two-chip ASSP+FPGA platform implementations; and how you can put the Zynq SoC to good use to drive prolific profitability at your company.

To many, the word "platform" has become an overused marketing term. But in the electronics industry, many companies such as Apple, Intel and Cisco Systems have effectively executed platform business strategies to become highly profitable electronics leaders. In deploying platform strategies, companies make a relatively substantial upfront investment in creating and documenting the blocks they designed for the initial version of their electronics product platform. They then turn those design blocks into intellectual-property (IP) blocks, which they reuse to quickly and easily expand into derivative product lines and

models along with next-generation products, delivering each of those derivative products faster and with less effort, less design cost and fewer resources.

CHALLENGES TO ACHIEVING PROFITABILITY

Research firm International Business Strategies (IBS) in its 2013 report "Fac-tors for Success in System IC Business" concludes that as the cost of producing an ASIC or ASSP device using the latest silicon processes continues to rise from the 28-nanometer manufacturing node to 20 nm, 16 nm and 10 nm, companies producing their own chips will increasingly struggle to achieve the traditional end-product revenue goal: revenue 10 times larger than their initial R&D investment. Many make great strides toward achieving this 10x goal by creating multiple derivative products on each node.

"Derivative designs can cost 20 percent of the initial design cost, which means that if a commitment is made to a new product family that has very

Rising IC Design Costs

Initial design cost at 28nm averages $130 million.
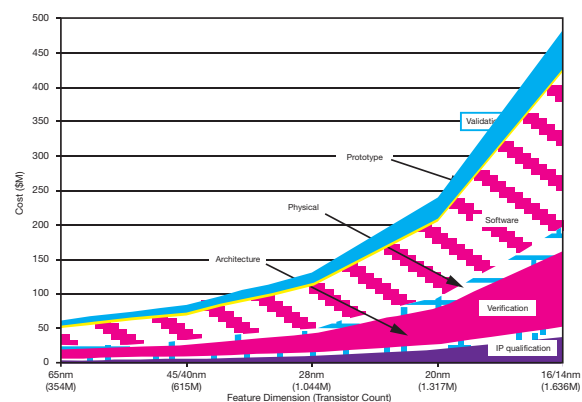
Derivative design cost at 28nm averages $35.6 million.
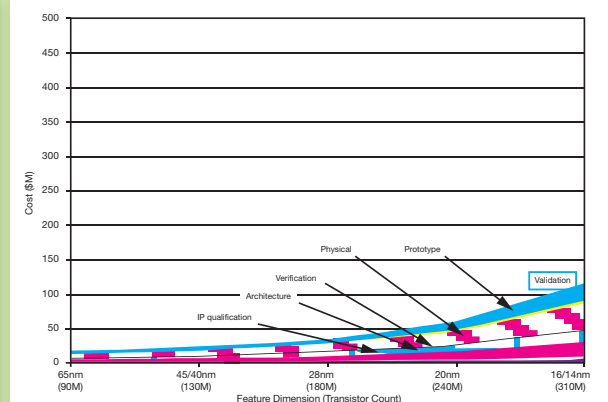
Rising Revenue Requirements to Support Rising IC Design Costs

Revenue target of 10x initial design cost at 28nm is $1.3 billion.
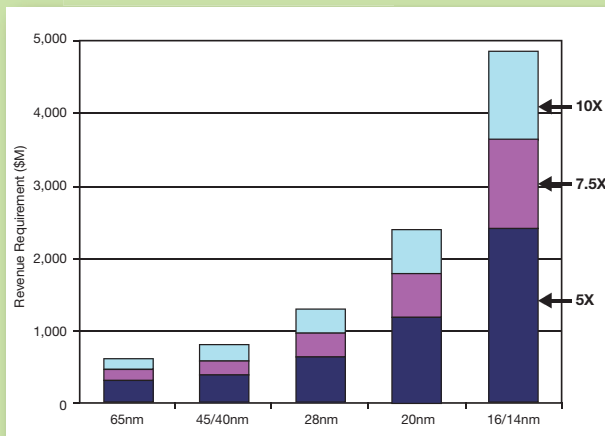
Revenue target of 10x derivative design cost at 28nm is $356 million.
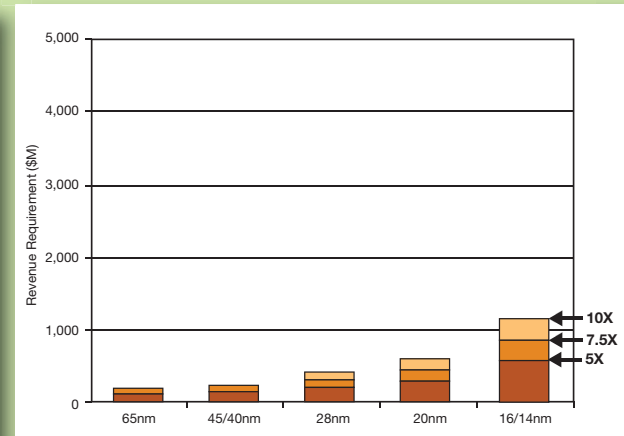
Source: International Business Strategies, Inc. (IBS) (2013/2014)

Figure 1 – The initial cost of developing an IC rises with the introduction of each new silicon process technology. In comparison, the cost of developing subsequent derivative products on the same node is much lower, making it far easier to achieve the end-product revenue target of 10x design cost. Platform design allows companies to rapidly develop derivative designs and increase profitability.

'New design concepts that reduce the cost of implementing new products have the potential to change the structure of the semiconductor industry dramatically.'

high development costs, then derivative designs can be implemented at a much lower cost. To optimize revenues and profits, it is advantageous for companies to implement multiple [derivative] designs in a technology node," the report said. "Implementing only one or two designs in a technology node can result in very high up-front costs and high risks associated with getting good financial returns.

"New design concepts that reduce the cost of implementing new products have the potential to change the structure of the semiconductor industry dramatically," the report went on. "However, until a new design methodology emerges, semiconductor companies need to adapt their business models to the reality of the changing financial metrics in the semiconductor industry as feature dimensions are reduced" [*Source: International Business Strategies, Inc. (IBS) (2013/2014)*].

In the study, IBS shows that the design cost of a 28-nm ASIC or ASSP (the first or initial product) is $130 million (Figure 1). Meanwhile, the design cost of a derivative is significantly lower: $35.6 million. Thus, to achieve the 10x revenue goal for both types of devices requires an investment of $1.3 billion for complex devices but only $356 million for derivatives [*Source: International Business Strategies, Inc. (IBS) (2013/2014)*].

The IBS study shows that companies must spend 650 engineering years to design a complex ASIC at 28 nm. In com-
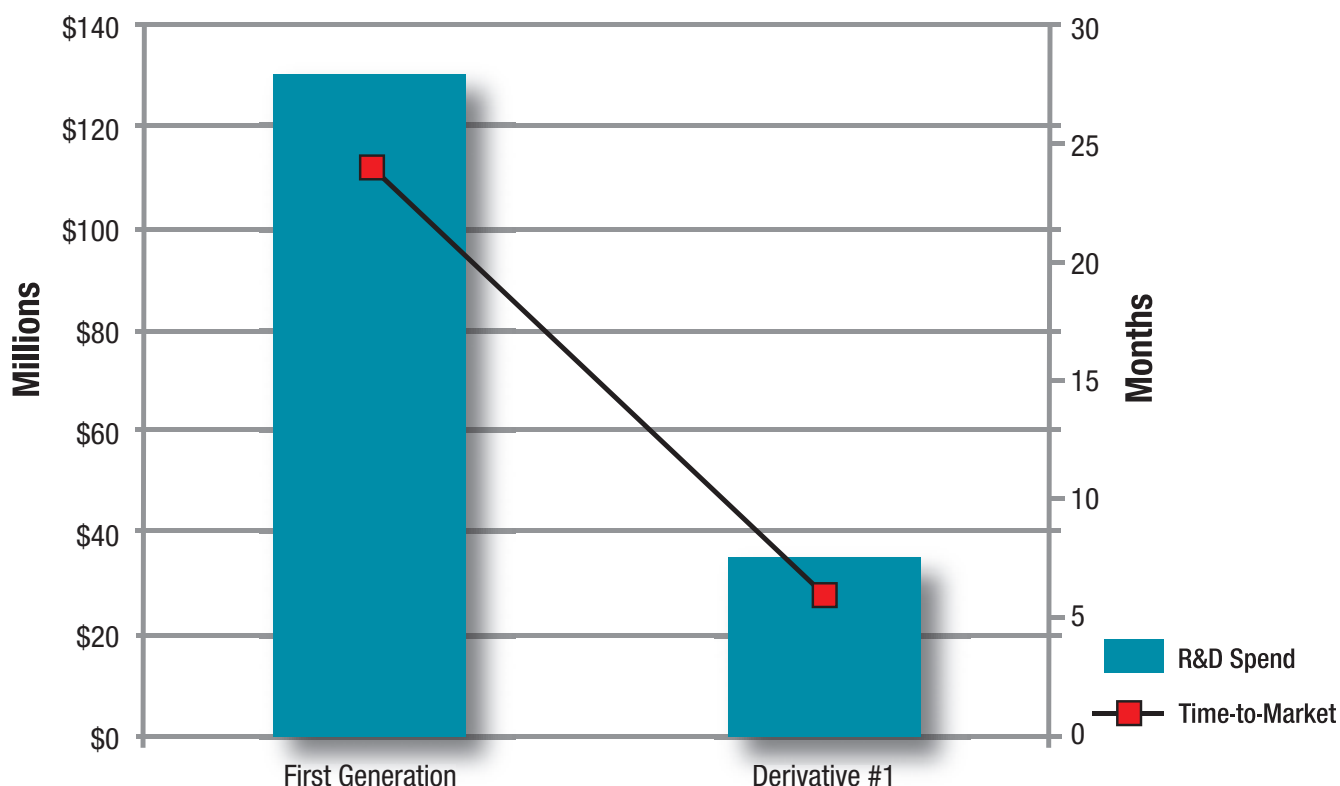
Figure 2 – Derivative designs reduce time-to-market, development time and costs, making profitability goals easier to achieve.

Developing multiple derivatives on the same node using a platform strategy allows a company to optimize revenue and profit.

parison, a derivative 28-nm ASIC design requires only 169 engineering years to develop, a 3.8x reduction.

Assuming ASIC teams are developing new designs in step with Moore's Law and are working on a two-year development cycle, it would take 325 engineers to complete a complex 28-nm ASIC in those two years. However, it would take only 85 engineers to complete a derivative 28-nm ASIC in two years. Or if a company were to use all 325 of engineers to develop the derivative as well, they could complete the job in six months (Figure 2).

Further, as illustrated in Table 1, if we assume that the initial complex design achieved its 10x revenue payback of $1.3 billion using 325 engineers, a derivative design with a smaller addressable market that is only 80 percent ($1.04 billion) the revenue size of the initial ASIC's market would require just 85 engineers over two years to develop a product that would garner a net present value (NPV) that is much better than the NPV of the initial ASIC design. (NPV is defined as the difference between the present value of cash inflows and cash outflows. The concept is used in capital budgeting to analyze the profitability of an investment or project.)

What's more, the derivative would have a much more favorable "profitability index" or PI (NPV divided by R&D money spent) than the initial ASIC. Even if that derivative addressed a market half the size ($650 million) of the initial design, it would have an NPV better than the initial ASIC, with essentially the same PI.

PLATFORMS: THE BEST STRATEGY FOR PROFITABLE DERIVATIVES

Increasingly semiconductor companies, as well as electronics system companies, are turning to platform strategies as a way to quickly create derivative products and maximize profitability in the face of rising R&D costs, increased competition and customer demand for better everything. Platform strategies further reduce product development time, time-to-market and engineering-hour costs while simultaneously increasing the profitability of each derivative or next-gen product.

As the IBS study shows, developing derivative designs is a way for companies to "optimize revenues and profits." And developing multiple derivatives on the same node (in other words, derivatives of derivatives) using a platform approach allows companies to further optimize revenue and profit, as each subsequent design can benefit from lessons learned in the prior design, reuse and a more precise understanding of customer requirements.

PROCESSING CHOICE IS KEY TO SUCCESS OF THE PLATFORM

Two of the biggest business decisions a company can make when deploying a platform strategy are actually vital technical decisions: Which one of the many processing systems will be at the heart of your product platform? And which silicon implementation of that processing system is the best for improving profitability?

In a platform strategy, a processing system must meet or exceed application software and system requirements. It must be scalable and easily extendable; must have a large, established and growing ecosystem; and

	Lifetime Revenue* ($M)	R&D Spend ($M)	Lifetime Net Profit** ($M)	NPV*** ($M)	Profitability Index
Initial Complex ASIC	$1300	$130	$260	$12.85	0.1
Derivative #1 (80% size market)	$1040	$35	$208	$74.78	2.14
Derivative #2 (50% size market)	$650	$35	$130	$34.47	0.98

Assumes 7 years
**Assumes profit margin 20%*
***Assumes 15% discount rate*

Table 1 – Creating derivative designs has an impressive net present value (NPV) but even more impressive profitability index.

must allow architects and engineers to leverage prior design work. Finally, it must come from an established, stable supplier with a road map and a track record of not deviating from that road map or of issuing endless errata. While there are candidates that fit some of these qualifications, the one that meets or exceeds all of them is the ARM microprocessor architecture.

ARM has become the de-facto-standard embedded processing architecture for just about anything that isn't a PC. A vast majority of electronics systems today that use advanced embedded processing, from mobile phones to cars to medical equipment, employ ARM processor cores. In particular, ARM's Cortex-A9 processor architecture is at the heart of many types of systems-on-chip (SoCs). It can be found in ASIC designs typically created for highest-volume, value-added products like bleeding-edge smartphones and tablets, as well as in many ASSP designs for companies wishing to enter established low- to moderate-volume markets that typically compete on pricing for lack of feature differentiation.

To add differentiation to their products, many companies create product platforms that pair an FPGA with an off-the-shelf ASSP based on an ARM processing system. In this configuration, they can differentiate in hardware as well as in software, creating a broader feature set or a higher-performing end product that's flexible and upgradable—one that helps them outshine competitors offering me-too software-programmable-only ASSP implementations. Adding Xilinx FPGAs to these ASSPs has helped a plethora of companies differentiate their products in the marketplace.

THE IDEAL PLATFORM SOLUTION: ZYNQ SOC

With the Zynq-7000 All Programmable SoC, Xilinx is fielding a platform implementation of the stalwart ARM Cortex-A9 that suits the vast majority of embedded applications. As illustrated in Table 2, the Zynq SoC offers many advantages over ASIC, ASSP and even ASSP+FPGA combos as a silicon platform. In comparison to other hardware implementations of the ARM processing

system, the Zynq SoC has the best feature set in terms of NRE, flexibility, differentiation, productivity/time-to-market, lowest cost of derivatives and best overall risk mitigation (Table 3).

What's more, the Zynq-SoC has vast cost advantages over other platform implementations. Let's look at the numbers.

The average cost of designing a 28-nm ASIC is $130 million, and thus the 10x revenue goal amounts to $1.3 billion for ASIC designs, said Barrie Mullins, director of All Programmable SoC product marketing and management at Xilinx. But typical design projects based on the Zynq SoC inherently have a much lower overall design cost and faster time-to-market than ASIC implementations, he said. That's because Zynq SoCs supply a predesigned, tested, characterized, verified and manufactured SoC that provides software, hardware, I/O performance and flexibility for differentiation. What's more, the Zynq SoC benefits from the fact that Xilinx hardware and software design tools are inexpensive and are highly integrated, whereas ASIC tool flows are

	Total System Cost	Flexibility	Differentiation	Time-to-Market	Cost of Derivatives	Risk
Zynq SoC	Low + best value	Most flexible: HW and SW programmable + programmable I/O	Highest degree of programmability, HW/SW co-design	Fastest for integrated HW & SW differentiation	Lowest due to HW & SW programmability	Predictably low risk
ASSP + FPGA	Higher than Zynq SoC (system dependent)	Highly flexible but ASSP I/O limited compared to Zynq SoC	HW and SW programmable, ASSP-dependent	Fastest if ASSP requires HW differentiation	Low to high depending on FPGA vendor	Low to high depending on FPGA vendor
ASSP	Lowest if SW-only programmability is sufficient	Good but SW-programmable only	Limilted to SW programmable only - easy cloning	Fastest if SW-only differentiation required	Lowest if SW-only derivatives needed	Can be Lowest if SW-only programmability is sufficient
ASIC	High to prohibitive	Once manufactured only limited SW flexibility	Best HW differentiation but limited SW differentiation	Lowest & riskiest	Highest	Terrible (respins)

Legend:
- Best platform attributes
- Good platform attributes
- Passable platform attributes
- Worst platform attributes

Table 2 – The Zynq-7000 All Programmable SoC offers an ideal mix of attributes for customers looking to implement a platform strategy.

Lowest NRE, Best Risk Mitigation	Greatest Flexibility & Differentiation	Streamlined Productivity & Fast TTM	Lowest Cost of Derivatives & Highest Profitability
✔ Already manufactured silicon ✔ Negligible development & design tool costs ✔ Xilinx IP library + third-party IP ✔ Extensive development boards	✔ All Programmable HW, SW & I/O ✔ Anytime field programmable ✔ Partial reconfiguration ✔ System Secure (encryption)	✔ Instant HW/SW co-development ✔ All Programmable Abstractions (C, C++, OpenCV, OpenCL, HDL, model-based entry) ✔ Vivado Design Suite, Vivado HLS, IP Integrator & UltraFast Methodology ✔ Broad and open OS & IDE support (Open-source Linux & Android, FreeRTOS, Windows Embedded, Wind River, Green Hills, & many others)	✔ IP standardized on ARM AMBA AXI4 ✔ Reuse precertified code (ISO, FCC, etc.) ✔ Reuse & refine code & testbenches ✔ Volume silicon, power circuity, PCBs & IP licensing

Table 3 – Factors like low NRE charges and flexibility make the Zynq SoC the ideal processing choice for a platform strategy.

complex, have significant interoperability and compatibility issues, and entail complex licensing with costs running in the millions. Xilinx's design flow is especially streamlined when designers use Xilinx's recommended UltraFast™ methodology. In addition, said Mullins, IP qualification costs are low because the Xilinx ecosystem IP is already designed and preverified, while Xilinx tools generate middleware.

As a result, Mullins said, a typical Zynq SoC project runs $23 million. Thus, to achieve the standard 10x revenue goal for design projects requires lifetime revenue of $230 million—a 10x goal that is far more achievable and feasible than the $1.3 billion required to achieve 10x for an ASIC implementation (Table 4).

Using the method described above while analyzing the IBS data, if we as-

sume that an initial complex design implemented in a Zynq SoC was able to capture 100 percent of the same $1.3 billion` targeted market, it would require only a $23 million investment using 57 engineers for two years to bring the product to completion.

If we assume that the initial Zynq SoC design has the same 20 percent profit margin as the initial ASIC design, the initial Zynq SoC design would have an

	28nm ASIC (IBS Data)			Zynq SoC (Xilinx Estimates)		
	%	Approximate Engineering Months	Total Cost ($M)	%	Approximate Engineering Months	Total Cost ($M)
Hardware						
IP qualification	26	704	11.8	20	240	4.0
Architecture	8	209	4.2	45	100	2.1
Verification	53	1431	28.9	35	160	3.0
Physical design	13	350	6.9	0	0	0
Subtotal hardware (design engineering resources)	100	2694	51.8	100	500	9.1
Software		4296	59.8		720	10.0
Prototype cost ($M)	2.1			1.0		
Validation of prototypes		815	16.6		140	2.8
Total		7805	130.3		1360	22.9

Table 4 – The cost of a Zynq SoC project is considerably lower than that of an equivalent ASIC project.

Zynq SoC Platform at 15% Profit Margin	Lifetime Revenue* ($M)	R&D Spend ($M)	Lifetime Net Profit** ($M)	NPV*** ($M)	Profitability Index
Initial Complex Zynq SoC design	$1300	$23	$195.00	$73.67	2.54
Derivative #1 (80% size market)	$1040	$9.2	$156.00	$69.78	6.02
Derivative #2 (50% size market)	$650	$9.2	$97.50	$39.55	3.41

Assumes 7 years
**Assumes profit margin 15%*
***Assumes 15% discount rate*

Zynq SoC Platform at 20% Profit Margin	Lifetime Revenue* ($M)	R&D Spend ($M)	Lifetime Net Profit** ($M)	NPV*** ($M)	Profitability Index
Initial Complex Zynq SoC design	$1300	$23	$260.00	$107.27	3.70
Derivative #1 (80% size market)	$1040	$9.2	$208.00	$96.66	8.33
Derivative #2 (50% size market)	$650	$9.2	$130.00	$56.34	4.86

Assumes 7 years
**Assumes profit margin 20%*
***Assumes 15% discount rate*

Table 5 – The NPV and profitability index show the Zynq SoC to be a far superior platform choice than ASIC-based platforms.

NPV of $107.27 million, with a PI of 3.7, which is dramatically better than the initial ASIC's NPV of $12.85 million and its PI of 0.1. The NPV and PI for Zynq SoC derivatives at that same 20 percent profit margin are even more impressive (Table 5).

Xilinx customers have shown that the cost of a derivative in a Zynq SoC platform strategy is typically 60 percent less than their initial design (see sidebar).

Comparing the Zynq SoC platform derivative at the same 20 percent profit margin as the ASIC platform derivative addressing a market 80 percent the size of the initial design, the Zynq SoC platform's NPV is $96.66 million with a PI of 8.33. This is considerably better than the ASIC derivative, which has an NPV of $74.78 and a PI of 2.14. Similarly, the NPV for the derivative Zynq SoC design addressing a market half the size of the initial Zynq SoC design's targeted mar-

ket would be $56.34 million with a PI of 4.86. This is far superior to the ASIC platform derivative's numbers.

Even if we leave the ASIC platform's profit margin at 20 percent and compare the results to a Zynq SoC platform assuming a lower, 15 percent profit margin (accounting for perhaps higher unit costs for the Zynq SoC), the Zynq SoC presents a far superior path to maximizing profitability. The initial Zynq SoC design at a 15 percent profit margin would have an NPV of $73.67 million, yielding a PI of 2.45. This is a vast improvement over the initial ASIC's NPV of $12.85 million and its PI of 0.1 even when the ASIC has 20 percent profit margin.

For a Zynq SoC platform design that targets a market of 80 percent ($1.04 billion), the revenue size of the initial Zynq SoC's targeted market, it would take 23 engineers two years to develop a derivative Zynq SoC-based product. In the end, the product would garner an NPV

of $69.78 million with a PI of 6.02. This compares with the ASIC derivative's NPV of $74.78 million, which is slightly better than the Zynq SoC derivative's NPV. However, the PI for the Zynq SoC derivative at a 15 percent profit margin is considerably better than the ASIC derivative's PI of 2.14, even when the ASIC has a higher (20 percent) profit margin.

Further, a derivative Zynq SoC design (again at a 15 percent profit margin) addressing a market half the size of the initial Zynq SoC design's targeted market would garner an NPV of $39.55 million and a PI of 3.41. That is not only better than the ASIC derivative's PI of 0.98 but also better than the PI of the initial Zynq SoC.

It should be noted that while profit margins will vary depending on the volume needs of a given market, the data shows that the Zynq SoC is a superior platform choice even for high-volume applications. Even when comparing an

CASE STUDY:

National Instruments Achieves New Efficiencies with Zynq SoC

by Mike Santarini

National Instruments was an early adopter of the Zynq SoC and is already showing how leveraging the device as a platform can increase efficiencies and raise profitability.

"What we are doing with the Zynq platform is creating our own platform," said James Smith, director of embedded systems product marketing at NI (Austin, Texas). "We are a tools provider for scientists and engineers. We are creating a development platform that customers can design on top of."

Xilinx and National Instruments have long worked closely together on Xilinx product road maps. This was especially the case as Xilinx was developing the Zynq SoC. NI was one of the first customers to receive shipments of the new device in November 2011 and has already put it to good use as a platform.

In the summer of 2013, NI announced not one but three new products based on the Zynq SoC: the high-end CompactRIO-9068 software-designed controller; a low-cost version targeted at students called myRIO; and another product called roboRIO for the First Robotics Competition.

Smith said that NI has been using a platform approach for the last decade, typically pairing a Xilinx FPGA with an off-the-shelf microprocessor. He termed the Zynq SoC the ideal platform for NI's RIO line.

"We get a lot of unique benefits out of using Zynq SoC over previous processing platforms," said Chris Rake, senior group manager for CompactRIO hardware at NI. "Zynq is a high-value product in that it integrates a processor that gives us about four times the performance of our previous-generation comparable product, along with a very rich Xilinx 7 series logic fabric plus additional DSP resources—and all at a very competitive price point."

Because the processor and logic are on the same chip, Rake said, "we not only see the processor performance improvement but a dramatic increase in DMA [direct memory access] performance. We have been able to more than double our DMA throughput and dramatically increase the number of DMA channels going between the processor and the programmable logic. None of that was possible with the platform architectures we were previously using. The price point to create an equivalent product would have been prohibitive."

Rake said that having an integrated processor and FPGA on the same chip also enables smaller form factors. "Instead of having two or three separate packages, we now have one package that represents the heart of the architecture and we can dramatically reduce size," said Rake. "Zynq allowed us to develop a range of products at the right cost of goods that we needed for the marketplace."

Rake noted that moving to a new platform always carries higher initial costs, and moving to the Zynq SoC was no exception. "We started the project by porting our software stack to the ARM dual-core A9 processor when Zynq was still in development," said Rake. "We used Xilinx's early development platform to work toward that, as well as using an ASSP with dual-core A9s that was on the market."

Then, when the silicon became available, NI began using the Zynq-7020, he said, porting the entire LabVIEW RTOS to NI Linux real-time. "So we had to do all those things upfront, and it was a significant effort. But now we have a core, standard architecture that multiple development teams throughout NI can use for new designs," said Rake.

NI stores all project schematics and layouts in a central repository. "For teams that are interested in using this processor and programmable logic technology, there are

well-established technical leads that act as points of contact for internal design teams, who assist with design and validation," Rake said. "The team that pioneered the initial Zynq SoC-based platform is now a team of experts who are a great resource in answering questions related to Zynq. They facilitate the development of the variants. At NI, it is a collaborative effort with the pioneering team."

Subsequent teams may leverage and reuse the platform schematic or the schematic plus layout and actual hardware components in the new products they are developing. "So after the initial work, now we get to enjoy the benefits of that investment for the coming years as we roll out the products on our road map as well as quickly create any other derivatives that we may add to the road map in the future," Rake said.

By January 2014, the company had already rolled out two derivative products built on its initial Zynq SoC-based platform (a just-released derivative is highlighted on page 58). "We have those products out already and other derivative products will now be spinning off each of those," Smith said. "It's really a tree that starts with this reference design as the trunk, sprouting branches, which in turn sprout other branches."

In addition to achieving greater time-to-market efficiencies, the Zynq SoC platform is also directly impacting the bottom line. In its 2013 Q3 earnings call, NI reported that its cRIO-9068 and sbRIO (single-board) analyzer lines drove record third-quarter revenue, while the myRIO helped lead to a new Q3 revenue record for NI's academic division. Smith estimates that each derivative design costs approximately 60 percent less than the original architecture, and time-to-market is reduced by approximately 30 percent.

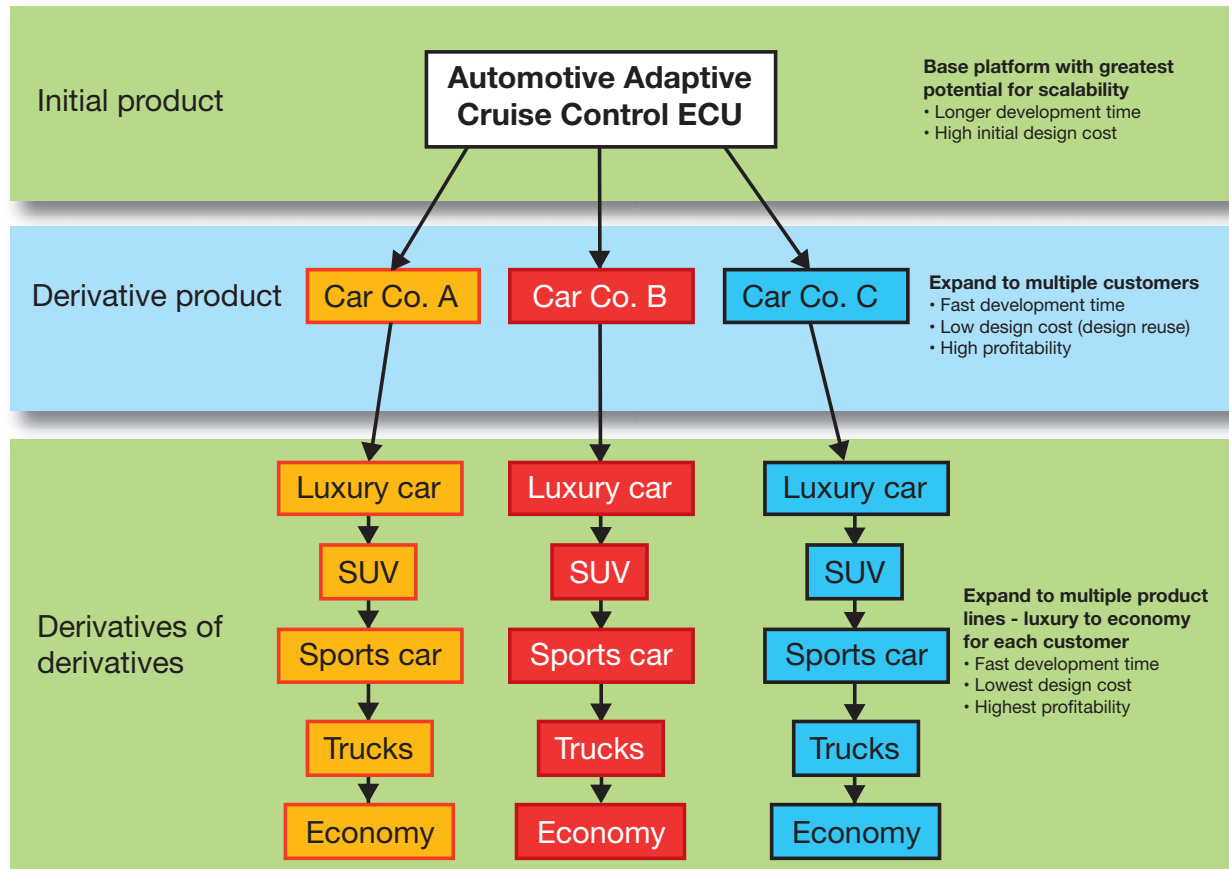The costs of doing the design in an ASIC would have been far higher.

Figure 3 – A Xilinx customer has maximized its initial design investment by using a Zynq SoC platform to serve multiple carmakers and multiple lines and models for each manufacturer, improving profitability.

ASIC platform at a higher profit margin of 20 percent to a Zynq SoC platform at a 15 profit margin, the Zynq SoC is a far better platform solution financially as well as technically. At lower volumes, the Zynq SoC platform is of course even more convincingly the best platform choice to maximize profitability

PROVEN PLATFORM SUCCESS WITH THE ZYNQ SOC

Today, a number of customers in a broad range of application areas are achieving dramatically greater scales of economy by leveraging the Zynq SoC as the heart of their platform strategies. A prime example is a world-renowned maker of high-end electronic control units (ECUs) for the automotive industry. This customer is standardizing on the Zynq SoC as a platform solution.

Wielding the Zynq SoC and heavily leveraging the reuse of tightly coupled hardware and software IP, the company has created a highly flexible ECU platform that it can quickly customize for the specific needs of multiple automakers and their different lines, models/configurations and accessory bundles (Figure 3). By using the Zynq SoC as a central platform, the company has achieved maximum economy of scale, reducing budgets while increasing the number of products it delivers to a growing number of customers. The upshot is delivering tailored ECUs to customers faster.

For a more detailed examination of another company using the Zynq SoC as a

profitability platform, see sidebar ("Case Study: National Instruments Achieves New Efficiencies with Zynq SoC").

The Zynq-7000 All Programmable SoC is the best device for implementing a platform strategy for most embedded applications. With its unmatched integration between ARM processing and FPGA logic and I/O programmability, the Zynq SoC allows every level of an enterprise to harmonize their development efforts and bring highly differentiated product lines to market faster than the competition. The Zynq SoC platform is enabling these customers to achieve prolific profitability.

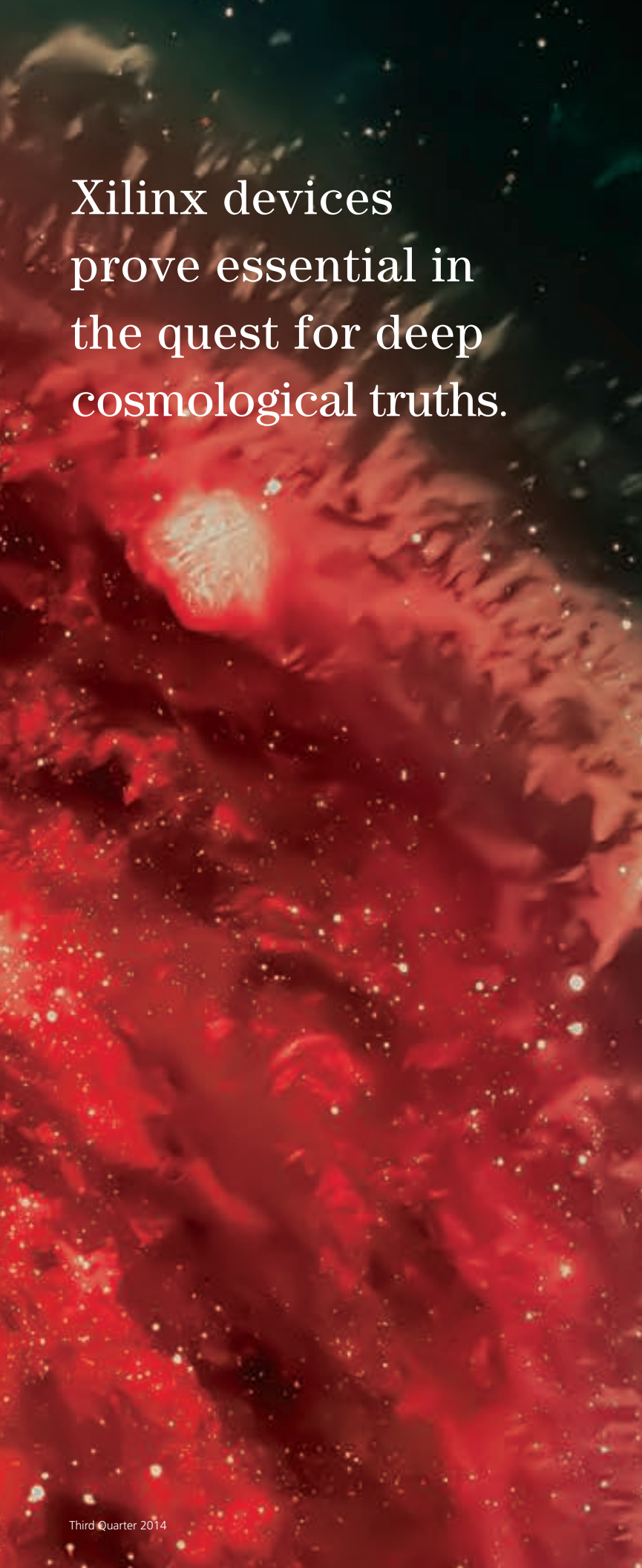For more information on the Zynq SoC platform, visit *http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm*.

The Search for Gravity Waves and Dark Energy Gets Help from FPGAs

by **Steve Leibson**
Editor in Chief, Xcell Daily
Xilinx, Inc.
sleibso@xilinx.com

Xilinx devices prove essential in the quest for deep cosmological truths.

A multidisciplinary team of scientists at the South Pole recently stared into the afterglow of the Big Bang. The team announced on March 17 that the BICEP2 experiment had collected the first evidence of gravity waves in the B-mode polarization of the cosmic microwave background (CMB).

Now scientists are looking for yet another fingerprint: evidence of gravity waves as recorded in the faint polarization spirals of CMB microwave photons. Finding these spirals would seem to confirm the inflation aspect of Big Bang theory—the idea that the universe expanded much faster than the speed of light long before the universe was even a picosecond old. In theory, this superluminal (faster than light speed) cosmic inflation created gravity waves that were embossed into the polarization of photons from the Big Bang.

The special camera the team is relying on to search for gravity waves uses transition-edge sensor (TES) bolometers to measure both E-mode (curl-free) and B-mode (gradient-free) microwave radiation. The camera is built around a second-generation McGill University DFMUX board based on Xilinx® Virtex®-4 FPGAs.

Astrophysicists elsewhere are using the same Xilinx board in their own experiments, while other researchers are trying out a new version of the camera updated with Kintex®-7 devices. The Kintex version is also part of a mammoth telescope that Canadian scientists will use to investigate dark energy.

ECHOES OF THE BIG BANG

The polarization variation in the CMB microwave photons is called the B-mode signal and the signature is extremely faint. While the overall CMB black-body temperature is 2.73 Kelvin, the B-mode signal is roughly one 10 millionth of a Kelvin.

The B-mode signal is generated at small angular scales by the gravitational lensing of the much larger, primordial "E-mode" polarization signal, and at large angular scales by the interaction of the CMB with a background of gravitational waves produced during the Big Bang's inflationary period.

B-mode polarization caused by gravitational lensing of the CMB was first detected in 2013 by the SPT polarimeter (SPTpol) camera installed on the 10-meter South Pole Telescope (SPT), which is operated by an international scientific team (Figure 1). The SPT is co-located at the Amundsen-Scott South Pole Station with the BICEP2 (soon to be BICEP3) and Keck Array CMB experiments.

The South Pole Telescope camera's helium-cooled, superconducting focal-plane microwave sensor is an array of 1,536 antenna-coupled TES bolometers paired as 768 polarization-sensitive pixels.

CMB radiation is the last echo of the immense energy burst that accompanied the Big Bang. Arno Penzias and Robert Wilson accidentally discovered it in 1964 as they experimented with cryogenic receivers to investigate sources of radio noise for Bell Telephone Laboratories in Holmdel, NJ. The CMB was the one noise source that the two scientists could not eliminate from their experimental data. Discovery of CMB radiation confirmed the cosmological Big Bang theory and earned Penzias and Wilson the Nobel Prize in Physics in 1978.

Based on the resolving power of that early-1960s experimental apparatus, the CMB appeared to be uniform in every direction and at all times of the day and night. That characteristic supported the theory that the CMB was a remnant of the Big Bang. More sensitive measurements, notably those performed by the Cosmic Background Explorer satellite, mapped the entire sky's worth of CMB to a very high resolution and showed that there were minute variations (anisotropy) in the CMB, which further reinforced the theory that the CMB was a fingerprint of the Big Bang. This discovery earned George Smoot and John Mather the Nobel Prize in Physics in 2006.

HELIUM-COOLED POLARIZATION SENSORS

Electrothermal equilibrium in superconductors and its ability to measure incident electromagnetic energy was discovered in the 1940s, but TES detectors only came into widespread use in the 1990s. They're now wide-ly employed for CMB experiments. The SPTpol camera's helium-cooled, superconducting focal-plane microwave sensor is an array of 1,536 antenna-coupled TES bolometers paired as 768 polarization-sensitive pixels; 180 pixels are sensitive to 90-GHz microwave radiation and 588 pixels are sensitive to 150-GHz radiation.

The 150-GHz CMB sensor module consists of corrugated feedhorn-coupled TES bolometers fabricated at the National Institute of Standards and Technology (NIST) in Boulder, Colo. Each 150-GHz TES bolometer module contains a detector array with 84 dual-polarization pixels operated at a temperature of a few hundred milli-Kelvin. Incident microwave energy travels down a coplanar waveguide to a microstrip transition and that feeds a lossy gold meander (a heating resistor). The incident microwave energy flowing into the meander causes heating. The meander is thermally connected to the TES sensors, which are made of an aluminum manganese alloy. These TES devices operate in the middle of their superconducting transitions, so they are extremely sensitive to small changes in incoming optical power.

The 90-GHz CMB sensor modules consist of individually packaged, dual-polarization polarimeters that were developed at Argonne National Laboratory. Each 90-GHz pixel couples to the telescope through a machined, contoured feedhorn, which channels the CMB radiation to a resistive PdAu absorbing bar. The resistive absorbing bar is thermally coupled to a Mo/Au bilayer TES (see Figure 2).

BRADFORD BENSON

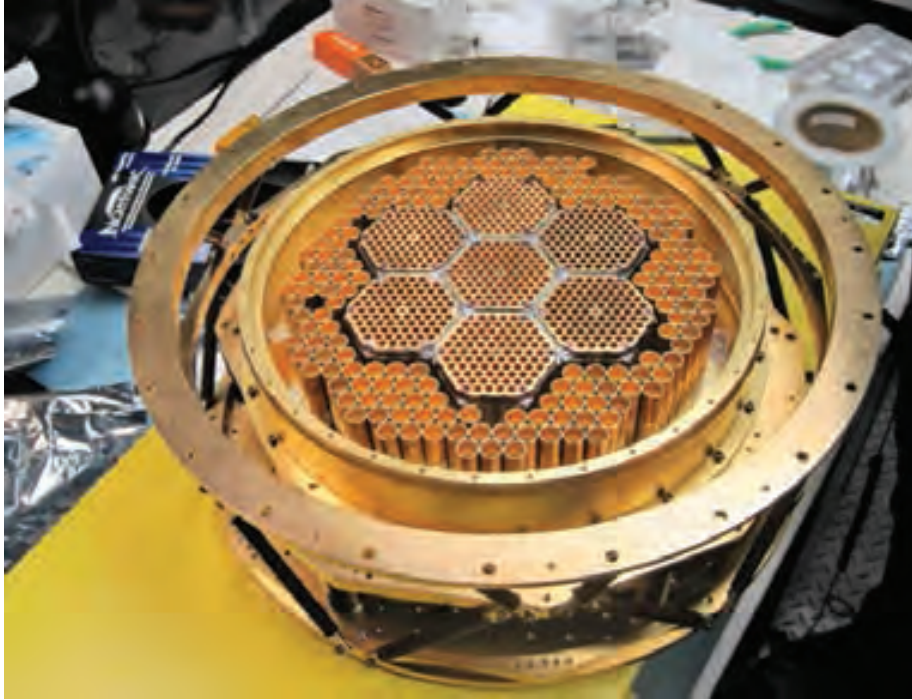Figure 1 – The South Pole Telescope at Amundsen-Scott South Pole Station

Figure 2 – The South Pole Telescope's microwave focal-plane array. The inner seven hex-shaped modules are a 150-GHz array and the outer ring is a 90-GHz array. Every pixel has its own individual horn, which couples light to each pixel's two TES bolometers.

For both the 150-GHz and 90-GHz sensors, thermal variations caused by microwave energy absorption create slowly varying changes in the resistance of each TES on the order of a few hertz. The resistance changes modulate a carrier current running through each of the 1,536 TES bolometers. These currents are then amplified by cryogenic superconducting quantum interference devices (SQUIDs). The need to bring all 1,536 measurements from the super-cold environments of the focal-plane sensor and SQUID arrays to the relative warmth of the South Pole required the development of an innovative digital frequency-multiplexing (DFMUX) scheme implemented with Xilinx Virtex-4 FPGAs.

SQUIDs have high bandwidth, so a frequency-multiplexed arrangement is easily used in this application. This multiplexing scheme permits the sharing of SQUIDs and minimizes the number of wires crossing into the cryostat that cools the focal-plane sensor array without degrading each bolometer's

noise performance. The DFMUX was developed at McGill University in Montreal, one of the institutions operating the South Pole Telescope. The others include the University of Chicago; University of California, Berkeley; Case Western Reserve University; Harvard/Smithsonian Astrophysical Observatory; University of Colorado, Boulder; University of California, Davis; Ludwig Maximilian University of Munich, Germany; Argonne National Laboratory, and NIST.

MAKING SENSE OF THE SENSOR DATA

The SPTpol camera uses a second-generation McGill DFMUX based on a Xilinx Virtex-4 FPGA. The FPGA digitally synthesizes a carrier comb that combines 12 carrier frequencies using direct digital synthesis (DDS). The carrier comb enters the focal-plane cryostat on a single wire and drives a set of 12 TES bolometers. Individual analog LC filters tune each of these 12 TES bolometers to a narrow frequency band. Each bolometer responds to the time-varying incident CMB radiation with a varying resistance over fre-

quencies ranging from 0.1 Hz to 20 Hz. The varying resistance of the TES bolometer modulates the carrier current flowing through the bolometer. The 12 TES bolometer currents are then summed together to form a modulated "sky signal."

A second DDS frequency comb called a "nuller" comb drives the summing node at the input to the SQUID amplifier. The nuller comb's phase and amplitude are set to cancel the carrier comb through destructive interference, leaving just the signals detected by the bolometers plus a small residual amount of carrier power. One SQUID amplifies this signal, converts it to a voltage and passes it back to room-temperature electronics for filtering, A/D conversion and demodulation by the FPGA. Figure 3 shows a system block diagram.

The digital output of the ADC directly enters the Virtex-4 FPGA for demodulation. The demodulation scheme resembles the digital up- and downconversion (DUC, DDC) algorithms used for GSM mobile telephony, with some exceptions. First, the bandwidth for each TES bolometer channel is very narrow—on the order of tens of hertz. Second, the carrier combs are made from synthesized sinusoidal carriers generated by the Virtex-4 FPGA. Carrier modulation occurs within the TES bolometers in the cryostat.

One Virtex-4 FPGA operates four of the SPTpol camera's 12-bolometer multiplex sets. The DFMUX design uses the Virtex-4 FPGA's on-chip logic, memory and DSP facilities for digital frequency synthesis, demodulation (downconversion, filtering and decimation), time-stamping and buffering. Using one FPGA to generate both the carrier frequency comb and the nuller frequency comb, as well as to demodulate the sky signal, means that all signals operate in lockstep. It's not possible for the comb generation and demodulation to drift relative to each other because they originate from the same master clock on the FPGA. Consequently, clock jitter is not a significant noise source, which measurements confirm.
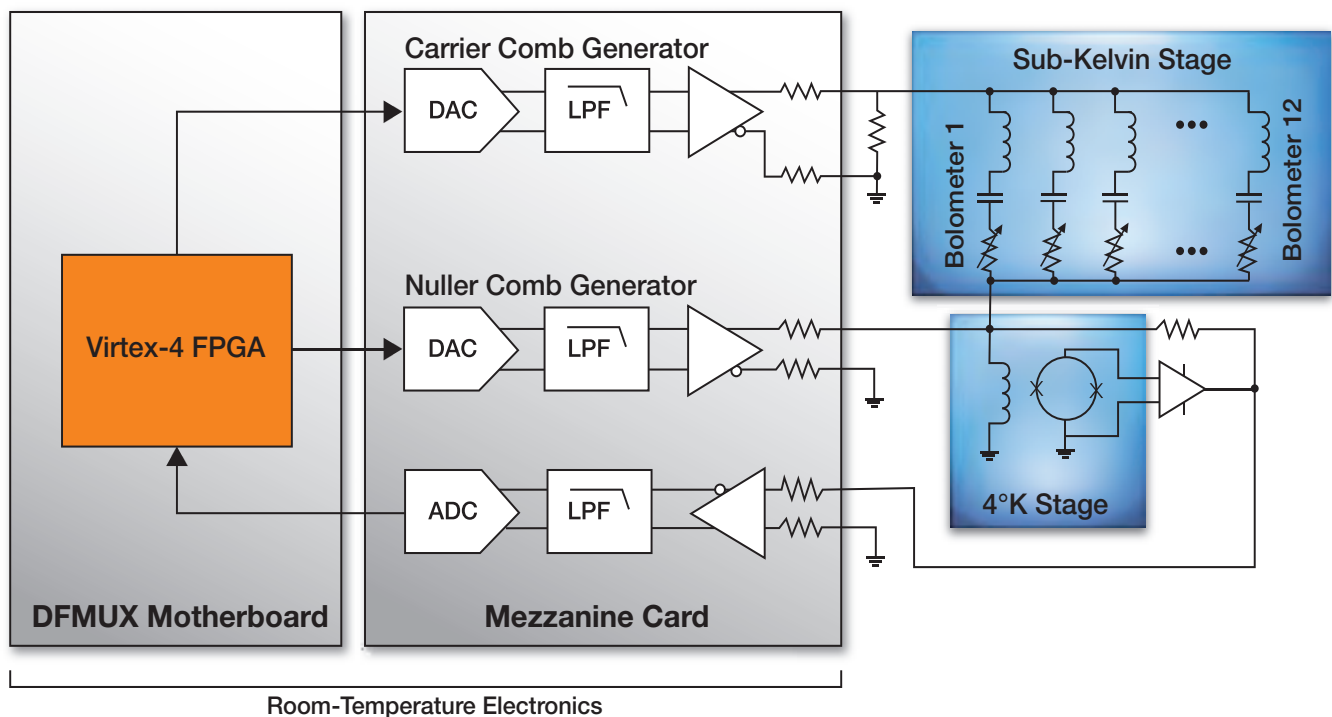
Figure 3 – Block diagram of a DFMUX-based TES bolometer system for measuring CMB radiation

A FULLY UTILIZED FPGA

There are two major blocks implemented within the FPGA: the digital multifrequency synthesizer (DMFS) and the digital multifrequency demodulator (DMFD). The system design uses two identical DMFS blocks for frequency synthesis. One generates the carrier frequency comb and the other generates the nulling signal. The frequency synthesizers operate at 200 MHz and employ 16-bit DACs running at 25 Msamples/second. The synthesizers are based on 11-bit, two's-complement direct digital synthesizers created by the Xilinx DDS compiler. Per-channel frequency resolution is 0.006 Hz.

Demodulation of the sky signal starts with digital downconversion. The incoming signal is mixed with reference waveforms to produce individual baseband signals. The frequency and phase of the reference waveforms are independent of each other.

The demodulated sky signal has been sampled with 14-bit resolution at 25 Msamples/s but the bandwidth of interest is much smaller than the Nyquist bandwidth of this sample rate. There-

fore, the demodulated baseband signals pass through cascaded integrator comb (CIC) decimation filters constructed from adders and accumulators in the FPGA. The first-stage CIC filter decimates the baseband signal by a factor of 128 using 28-bit precision. The output of this filter is then truncated to 17 bits.

The DFMUX time-domain-multiplexes eight bolometer channels (25 Msamples/s) into a CIC1 operating at 200 MHz. The CIC1 filter has an internal 28-bit data width and a 24-bit output. After CIC1 filtering, all bolometer channels are multiplexed together and feed a single CIC2, which has six variable decimation rates (by 16, 32, 64, 128, 256 and 512). The CIC2 filter is followed by a 152-tap FIR filter.

A channel identifier and time stamp are added to the FIR filter's output and then sent to dual-ported buffer storage with a rotating buffer list. The SDRAM's large buffer capacity eases latency requirements on the FPGA-based MicroBlaze™ soft processor, which runs Linux and supervises data flow through the system. Eased latency permits activation of the processor's MMU and

greatly improves Linux OS operation.

External control of the DFMUX board occurs over an Ethernet connection through an HTTP interface using two Web servers running on the MicroBlaze processor. All that's needed to control the DFMUX board is a Web browser. A Python scripting environment provides direct access to board-level control registers for more detailed work such as instrument tuning.

FUTURE WORK FOR THE DFMUX

The SPTpol camera is one of several such experiments looking at CMB radiation. The same DFMUX board used in the camera is also part of the EBEX balloon-borne "E and B Experiment" and the POLARBEAR CMB polarization experiment mounted on the Huan Tran Telescope at the James Ax Observatory in Chile. A more advanced version of the DFMUX board called the ICE-board, based on Xilinx Kintex-7 FPGAs, is just starting to deploy on new CMB experiments and in the Canadian Hydrogen Intensity Mapping Experiment (CHIME) radio telescope.

CHIME is a novel radio telescope located in a secluded valley near Penticton, British Columbia. The telescope consists of five large, 100 x 20-meter partial-cylinder reflectors roughly the size and shape of snowboarding half-pipes packed with arrays of radio receivers located along the focus of each partial cylinder. There are no moving parts (other than the Earth). When finished, CHIME will measure more than half of the sky each day as the Earth turns.

However, CHIME won't be studying the CMB. It will be looking for evidence of dark energy by surveying 21-cm (400- to 800-MHz) radio emissions in a large 3D volume of space at distances ranging from 7 billion to 11 billion light-years. CHIME will be measuring "baryon acoustic oscillations," or BAOs, which are periodic density fluctuations in enormous cosmic structures consisting of hydrogen gas. BAO matter clustering provides cosmologists with a "standard ruler" of approximately 490 million light-years, used for measuring immense distances. BAO signal variations could prove to be the signs of dark energy at work. At least, that's the hope.

CHIME is essentially a phased-array radio telescope. It synthesizes an image by recording the electromagnetic signal across a stationary antenna array and then reconstructing the overhead sky from the data using 2D correlation and interferometry. CHIME will require 160 interconnected Kintex-7 FPGAs to process BAO signal data being received at several terabytes per second.

FASTER THAN THE SPEED OF LIGHT

The inflation theory of cosmology posits that the universe underwent a violent expansion 10 to 35 seconds after the Big Bang—a physical expansion that exceeded the speed of light. That's pretty hard to accept if you think that the speed of light is absolute, and most of us do. Part of that Big Bang theory suggests that inflation left behind a cosmic gravitational-wave background (CGB) in addition to the CMB, and that the CGB impressed a polarization sig-

nature on the CMB. Results from the BICEP2 experiment are the first to confirm this theory.

Additional results from the SPTpol camera, EBEX, POLARBEAR, the Keck Array and the BICEP3 experiment are expected to reinforce that finding. For its part, CHIME will add yet another dimension to our quest for cosmological knowledge when it starts searching for dark energy.

Further Reading

For more technical information about the SPTpol camera, TES bolometers and the FPGA-based DFMUX readout board, check out the following references:

J. E. Austermann, et al., "SPTpol: an instrument for CMB polarization measurements with the South Pole Telescope," arXiv:1210.4970v1 [astro-ph.IM]

Ron Cowen, "Telescope captures view of gravitational waves," Nature, March 17, 2014

Matt Dobbs, et al., "Digital Frequency Domain Multiplexer for mm-Wavelength Telescopes," arXiv:0708.2762v1 [physics.ins-det]

M.A. Dobbs, et al., "Frequency multiplexed superconducting quantum interference device readout of large bolometer arrays for cosmic microwave background measurements," arXiv:1112.4215v2 [astro-ph.IM]

J. W. Henning, et al., "Feedhorn-Coupled TES Polarimeter Camera Modules at 150 GHz for CMB Polarization Measurements with SPTpol," arXiv:1210.4969v1 [astro-ph.IM]

J. T. Sayre, et al., "Design and characterization of 90-GHz feedhorn-coupled TES polarimeter pixels in the SPTpol camera," arXiv:1210.4968v1 [astro-ph.IM]

Graeme Smecher, et al., "An Automatic Control Interface for Network-Accessible Embedded Instruments," ACM SIGBED Review, Second Workshop on Embed With Linux (EWiLi 2012), Vol. 9 Issue 2, June 2012

Graeme Smecher, et al., "A Biasing and Demodulation System for Kilopixel TES Bolometer Arrays," arXiv:1008.4587 [astro-ph.IM]

K. Story, et al., "South Pole Telescope Software Systems: Control, Monitoring, and Data Acquisition," arXiv:1210.4966v1 [astro-ph.IM]

Virtex UltraScale FPGA Enables Terabit Systems
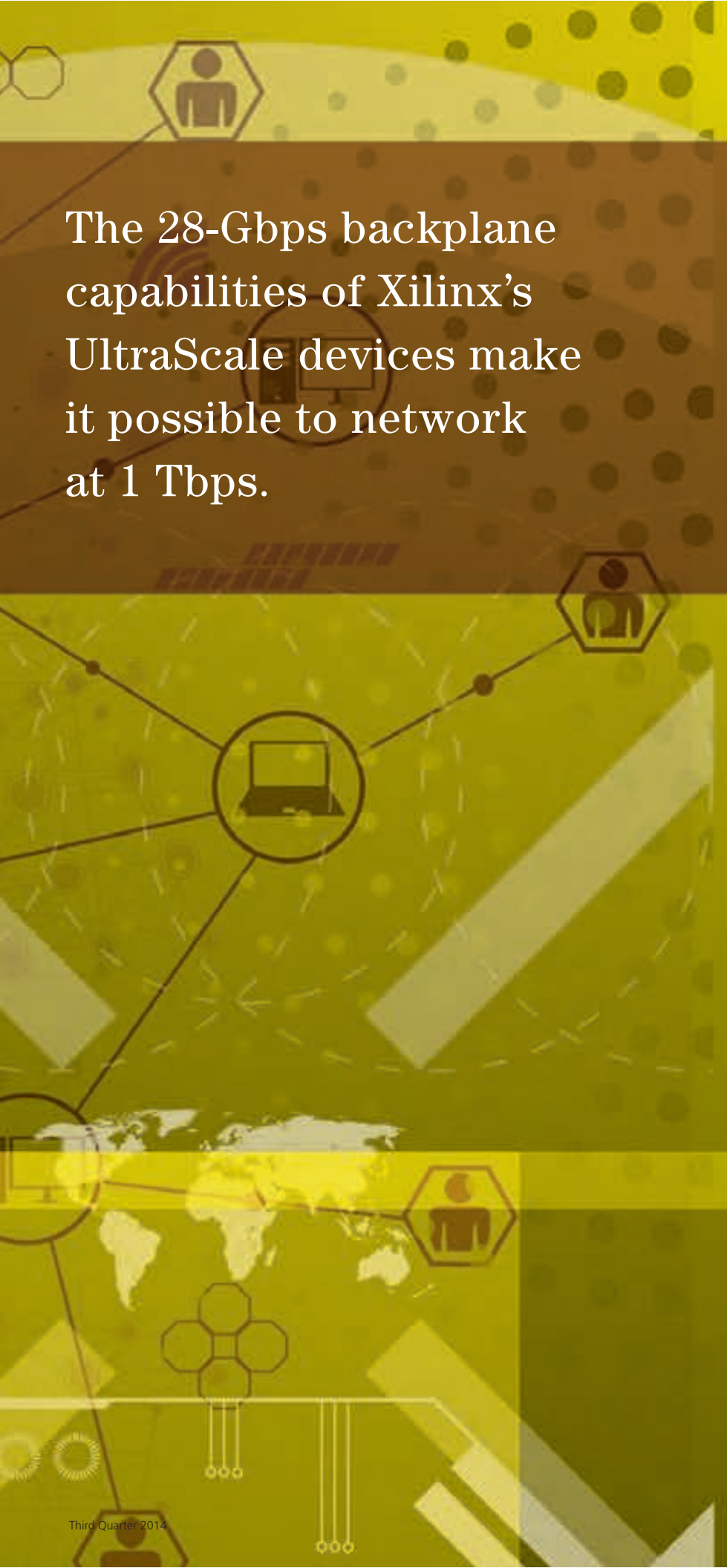
by Romi Mayder
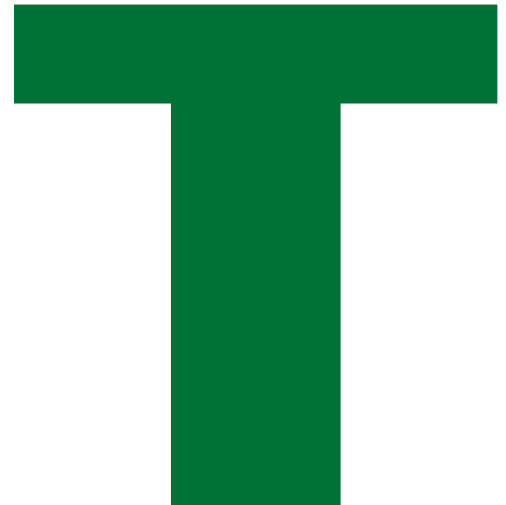Director of Technical Marketing
Xilinx, Inc.
RomiM@Xilinx.com

Frank Melinn
Distinguished Engineer
Xilinx, Inc.
FrankM@Xilinx.com

The 28-Gbps backplane capabilities of Xilinx's UltraScale devices make it possible to network at 1 Tbps.

T

Two years ago, a report by IEEE concluded that if current trends continue, communications networks will need to support capacity requirements of 1 terabit per second by 2015 and 10 Tbps by 2020. By next year, there will be nearly 15 billion fixed and mobile networked devices and machine-to-machine connections, according to the July 2012 report. For optical transport network (OTN) applications, the bandwidth per wavelength in a core node is expected to be in the range of 100G to 400G in 2015, rising to approximately 400G to 1T in 2020.

The Xilinx® Virtex® UltraScale™ All Programmable FPGA is an expansion of a line of high-end FPGAs that enable the implementation of 1-Tbps systems. The Virtex UltraScale line provides unprecedented levels of performance, system integration and bandwidth for a wide range of applications, such as wired communications, test and measurement, aerospace and defense, and data centers.

Many companies have expressed the need for 1-Tbit applications for networking. These applications require transceivers that can directly drive 25G/28G backplanes due to reasons of routability, crosstalk, differential insertion loss and impedance matching, among others. The Virtex UltraScale devices surmount these challenges and support 1-Tbps applications by enabling 25G/28G backplane operation without a retimer.

External retimers would create reliability concerns as well as require a significant amount of printed-circuit-board real estate, which is quite costly.

EXAMPLE 1-TBPS DESIGNS

Figure 1 is a high-level block diagram of a potential generic 1-Tbps system. As you can see, multiple tributary cards (here, two) with bandwidths of less than 1 Tbit are interfacing with a 1T line card over a backplane operating at 25G/28G.

Shown in Figures 2, 3 and 4 are more-detailed block diagrams of three possible 1-Tbps Ethernet and OTN applications, all built around Virtex UltraScale FPGAs. The 33-Gbps GTY transceivers may interface with CFP2/4 LR4 optical modules as well as the 25G/28G backplane. The 16-Gbps GTH transceivers are shown interfacing with high-speed memory.

None of these 1T applications would be feasible without a 25G/28G-capable backplane transceiver. A backplane with 10G links would be challenged in routability, crosstalk, differential insertion loss and impedance matching.

These 1T applications are only possible with FPGAs that can directly operate at 25G/28G over a backplane. Using external retimers would create reliability concerns as well as require a significant amount of printed-circuit-board real estate, which is quite costly. Backplane retimers represent additional components that must be placed on the line cards and tributary cards. Such PCB real estate is needed for voltage regulator modules, power distribution networks, DC blocking caps and reference clocks required for the backplane retimers to operate.

A MATTER OF ROUTABILITY

In these examples, we show 40 channels operating over a backplane at 25G/28G for a total bandwidth of 40 x 25 Gbps = 1 Tbps. A typical backplane is roughly 0.25 inch thick due to two factors: the mechanical requirements of press-fitting the connectors and the need to support the routing of numerous channels. If the backplane interface supported only 10-Gbps operation, the number of channels would need to increase by 25/10 or 2.5x. This would increase the number of backplane channels from 40 to 100 for each 1-Tbps line card and tributary card. In a system requiring 25 tributary cards and line cards, the result would be a channel count of 100 x 25 = 2,500. This would be very challenging to route.

Using a backplane dielectric material such as Panasonic's Megtron-6 with a dielectric constant of approximately 3.65 and a typical trace width of 7 mils, we find the stackup height per differential stripline pair to be 16 mils for 100-ohm differential odd-mode impedance.

Assuming a typical 2-mm pitch for the backplane connector, we can route roughly one channel between the connector pins per layer. Thus,
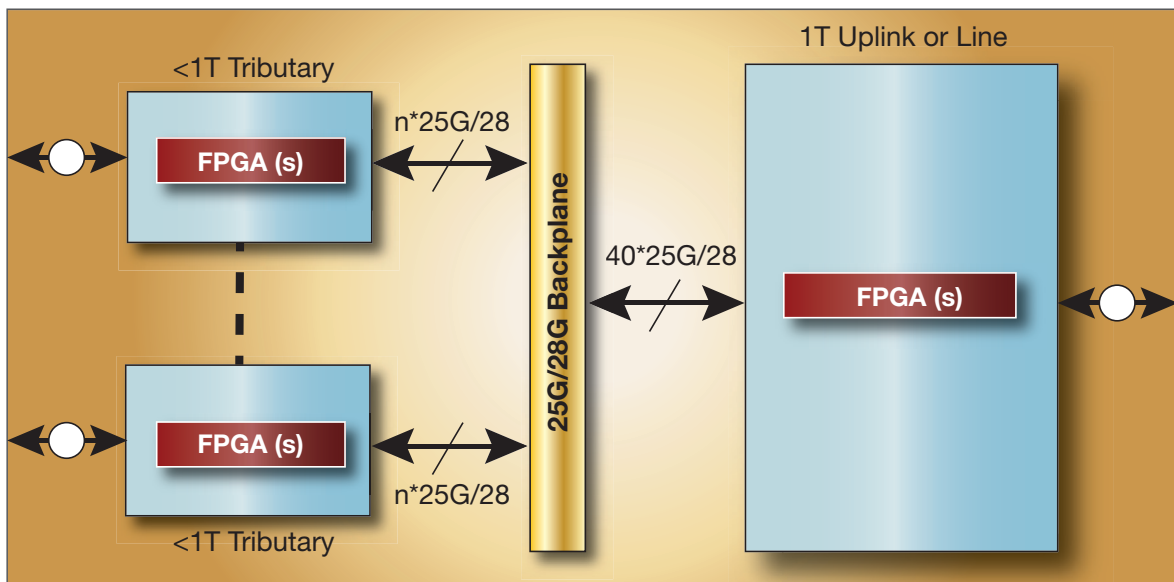

Figure 1 – In this high-level concept of a terabit system, multiple sub-1-Tbps tributary cards feed a 1-Tbps line card.

each layer of the backplane can support 10 channels (Tx + Rx) per connector per layer. For 25 Gbps, this typically requires 16 routing layers for a total board thickness of 16 mils x 16 routing layers = 0.256 inches. For 10 Gbps, it works out to 0.640 inches (16 x 2.5 = 40 routing layers for a total board thickness of 16 mils x 40 layers = 0.640 inches). However, the maximum thickness of a backplane is typically governed by the aspect ratio of the via holes. A typical drilled via hole is 15 mils in diameter and a typical aspect ratio is 25:1. This means the upper limit of the backplane thickness is roughly 14 x 25 = 350 mils. Thus, a 10G backplane cannot be fabricated to support a 1T application.

CROSSTALK CONCERNS

Another key advantage of a 25G backplane as compared with a 10G backplane for 1T applications is crosstalk. Crosstalk is a function of proximity of channels. A backplane with more traces has a higher probability of having higher crosstalk. Thus, a backplane with a thousand 25G channels will have less crosstalk than a backplane with 2,500 10G channels.

However, most dielectric materials have a component of forward-propagating crosstalk (FEXT) due to the fact that most dielectrics are not entirely homogenous. Additionally, crosstalk in the via hole area around the backplane connector is typically forward-propagating crosstalk.

Because crosstalk in a backplane system has both NEXT and FEXT components, designers must take great care to reduce the crosstalk noise contribution relative to the total noise budget. Both mutual capacitance Cm (electric field) and mutual inductance Lm (magnetic field) exist between victim and aggressor lines. The mutual inductance will induce current on the victim line opposite of the aggressor line (Lenz's Law). The mutual capacitance will pass current through the mutual capacitance that flows in both directions on the victim line. Currents of the near-end and far-end victim lines sum to produce the NEXT and FEXT components.
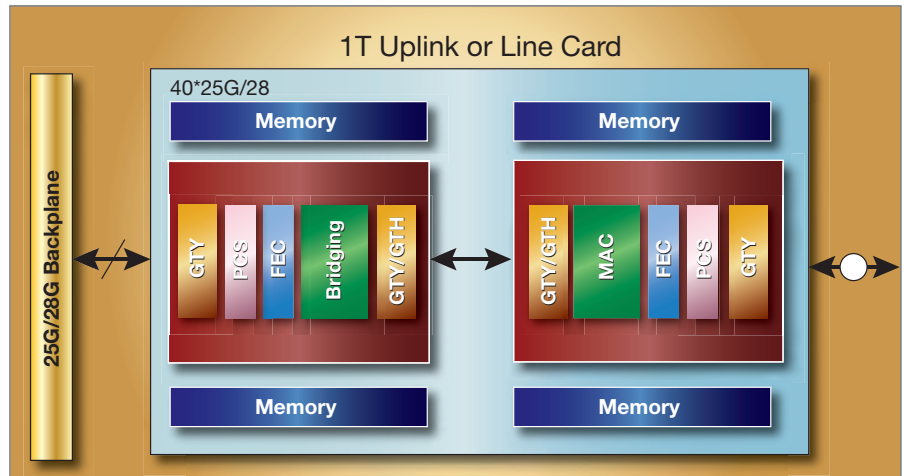

Figure 2 – A 1-Tbit Ethernet uplink module is fed by multiple sub-1TbE tributary cards via 25G backplane links.

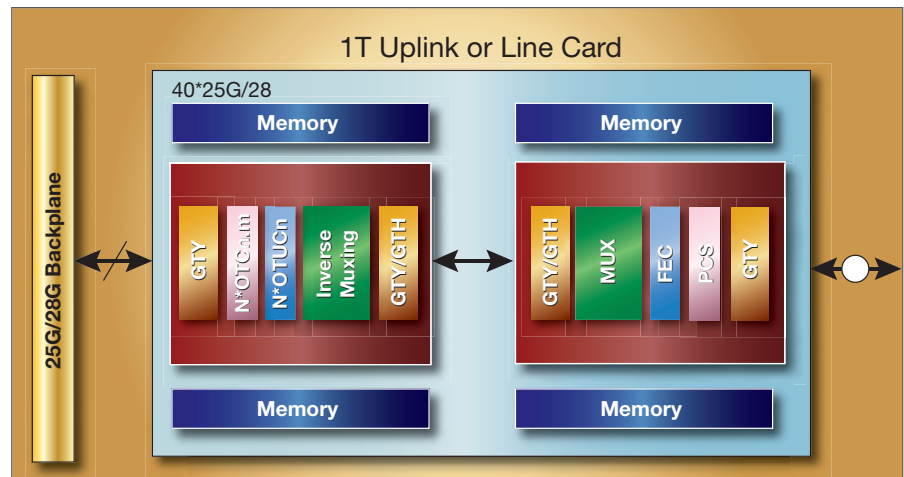Figure 3 – A 1TbE uplink module distributes its payload over OTN links for reuse of existing OTN line cards (<1Tb) via 25G/28G backplane links.
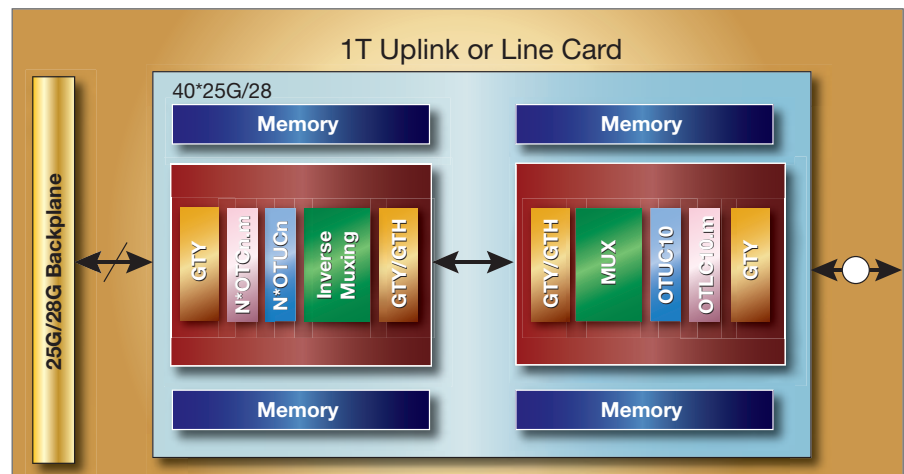

Figure 4 – A 1-Tbit OTN line card multiplexes feeds from sub-1-Tbit OTN tributary modules via 28G backplane links.

$I(Next) = I(Cm) + I(Lm)$ while $I(Fext) = I(Cm) - I(Lm)$.

NEXT is always positive. FEXT can be either positive or negative.

DIFFERENTIAL INSERTION LOSS
The insertion loss of the channel is an important parameter in establishing a reliable link. The IL is dominated by two factors: conductor loss and dielectric losses. When using a dielectric material such as Panasonic's Megtron-6, the tangent loss is 0.004. Shown in Figure 5 is the insertion of a 1-meter-long trace. The surface roughness is 1 micron (very low profile). Since the maximum thickness of the backplane can be only 0.350 inches, the trace width of the 10G backplane has been decreased to only 3 mils wide to allow for the 40 layers to fit into a backplane 0.350 inches thick and still maintain a differential odd-mode impedance of 100 ohms. For the 25G backplane, the trace width is 7 mils, which allows the required 16 routing layers to fit within the 0.350-inch maximum thickness.

Figure 5 shows that the differential insertion loss of the 10G backplane is higher than that of the 25G backplane. This is due to the narrower trace widths allowed for 10G backplanes because of the increased number of routing layers within the maximum backplane thickness.

We have shown that the Virtex UltraScale devices support 1-Tbps applications by enabling 25G/28G backplane operation without a retimer. These upcoming 1-Tbps applications will require transceivers that can directly drive 25G/28G backplanes due to reasons of routability, crosstalk, differential insertion loss and impedance matching. An excessive layer count and larger connectors play a role as well, along with manufacturing reliability issues related to via hole aspect ratio.

Differential Loss Comparison

m4
freq = 12.50GHz
dB (Backplane_25G_Loss = – 24.573

m5
freq = 5.000GHz
dB (Backplane_10G_Loss = – 26.070
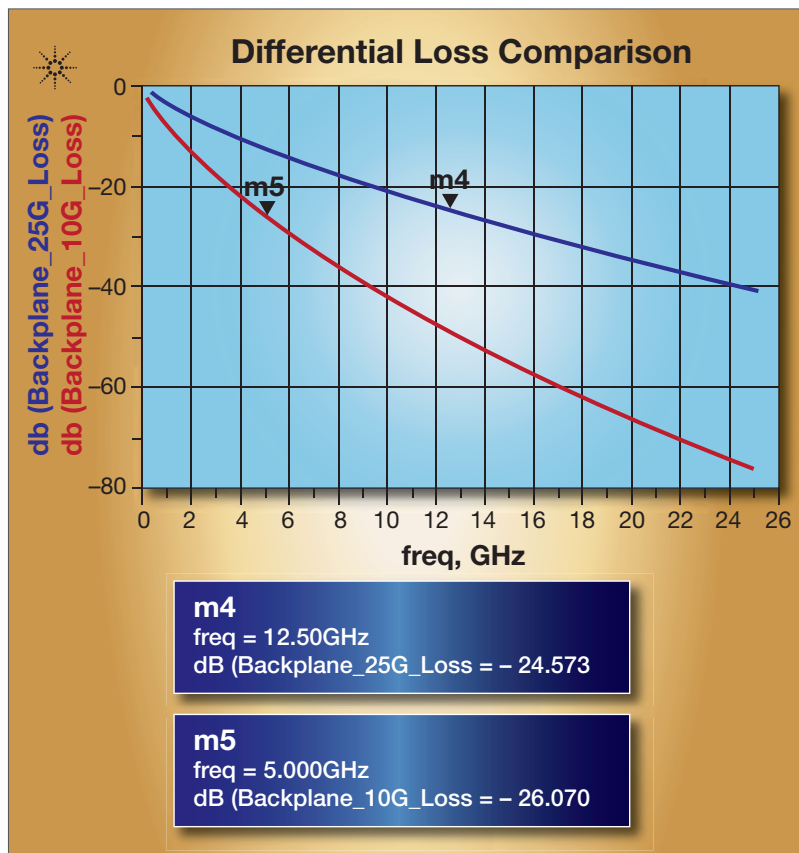
Figure 5 – The insertion loss of 25G and 10G backplanes is shown at the respective Nyquist frequencies.

Goodbye DDR, Hello Serial Memory

by Tamara I. Schmitz
Director of Memory and
Power, Technical Marketing
Xilinx, Inc.
tschmit@xilinx.com

DDR4 is the last of the the popular DDR line of memories that 90 percent of Xilinx customers use. Multiple contenders are vying for a chunk of that market share.

A seismic shift is shaking up the memory landscape. The cause for this shift is the fact that the line of incredibly popular DDR memories, a fundamental buffer used by 90 percent of Xilinx customers (Figure 1), will end with DDR4. This is not cause for immediate panic— DDR3 has a comfortable address on the majority of system boards and DDR4, though ramping slowly, will replace some of those sockets and serve them for years to come. Still, with the knowledge that DDR4 has no natural successor, customers are eyeing the next crop of memories and mulling over trade-offs such as bandwidth, capacity or power reductions. The likely successor is LPDDR3/4, with certain application spaces preferring serial DRAM solutions such as Hybrid Memory Cube (HMC).

To get a handle on these important changes in memories, let's look first at market trends that are affecting these devices and the limitations that are forcing the end of the DDR empire. Then let's consider the new class of DDR alternatives, from LPDDR to serial memory, a new concept about which designers will want to stay informed.

CHANGING MARKET TRENDS

Usually when customers are designing for their next generation of products, they look to the next generation of the same memory to give more capacity, speed and throughput. In this regard, Figure 2 shows the current and projected DRAM market share trends. DDR3 enjoys almost 70 percent of the total DRAM market today. Its rise to dominance was assured in the steep 40 percent uptick in adoption between 2009 and 2010. DDR4 has been slower in adoption, partly because of the incursions of Mobile DRAM, also known as LPDDR. DDR4 simply doesn't have as many sockets to claim if LPDDR is meeting the needs of the wireless market.

Looking at the graph, DDR4 is indeed picking up momentum, because it does have advantages—namely, a lower supply voltage, which saves power, and higher speed. So it will eventually take over for DDR3 in almost every market, eventually driven by the PC space. Despite the fact that PCs no longer drive >70 percent of DRAM consumption, they

are still the largest commodity-device segment. For now, according to memory vendors, DDR4 usage is localized more to the server space rather than in personal electronics segments. Still, DDR4 is an excellent choice for many designs. It is a well-known memory type and will be available for a very long time—in particular because there is no successor.

WHY IS DDR4 THE LAST?

So why is there no DDR5? When end customers want a new device, they want more memory. Consumers have an insatiable demand for memory bandwidth. MP3 players need to hold 10,000 songs rather than the couple dozen songs that a cassette tape used to hold. This applies equally well to the number of pictures or videos smartphones are expected to store. These expectations typically mean more components and more board space. Ironically, consumers don't always want their electronics devices to grow to a size proportional to their capacity or performance. There is an expectation that technology gets better, so there should be more in the same space—or maybe even in less space.
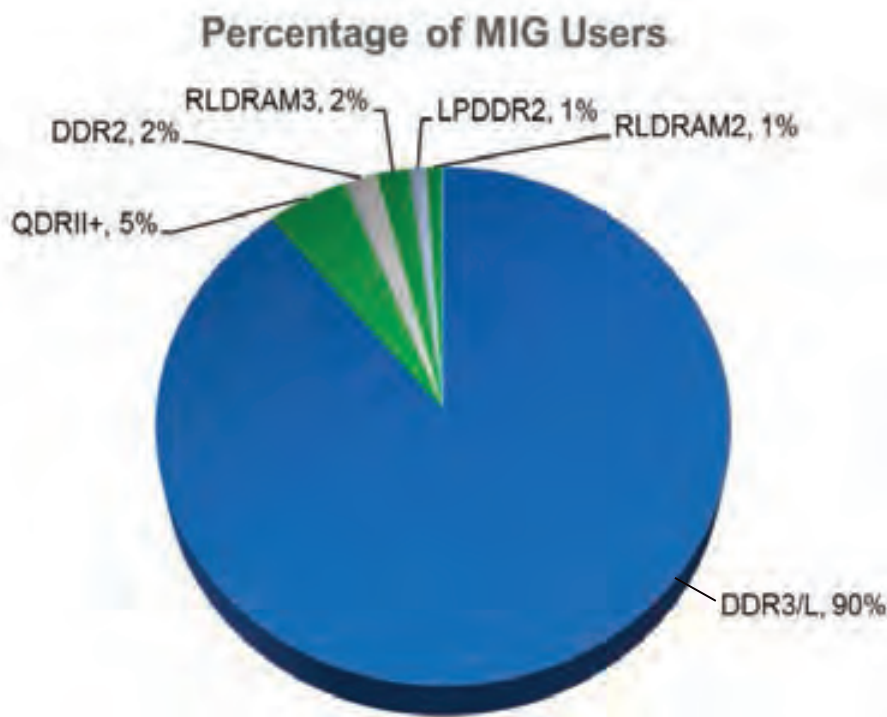

Figure 1 – This pie chart shows the memory usage of Xilinx customers as measured through the Vivado Memory Interface Generator (MIG) GUI in 2013.
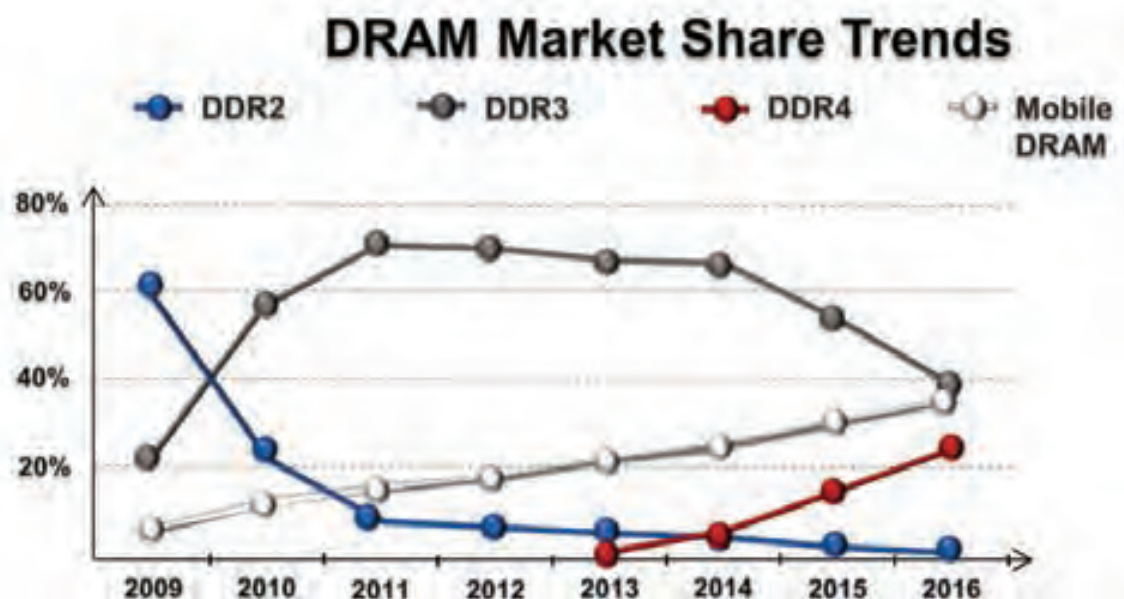

Figure 2 – Market trends for DRAM memory show big gains for LPDDR (Mobile DRAM).

When memory is used with a Xilinx® FPGA, there are specific guidelines about how to lay out the board to ensure proper margin and overall system success. Examples include trace lengths, termination resistors and routing layers. These rules limit how much the design can be compacted, or how close together the parts can be placed.

The alternative to the smallest board design would be some bleeding-edge type of packaging. Unfortunately, a new packaging technology that would include die stacking with through-silicon vias (TSVs) would translate into significantly more cost. DDR memory is not a high-cost device simply based on the economies of scale of the industry infrastructure and would not be able to adopt a radical departure in packaging or absorb an increased price point. Therefore, these improvements will most likely not be aiding any DDR3 or DDR4 system in the near future.

HOW XILINX TAILORED ULTRASCALE MEMORY PERFORMANCE

Xilinx® UltraScale™ FPGAs are designed for the higher performance and extended flexibility that any memory needs. DDR4 has already been demonstrated at 2,400 Mbps. This world's-first speed was confirmed by Agilent when that company designed an interposer to be inserted below the memory device, measuring the eye diagram of the system while in operation. Since DDR4 uses a new type of I/O structure called pseudo open drain (POD), Xilinx has added POD to UltraScale. This structure, in conjunction with the DDR4 protocol calling for an I/O voltage of 1.2 V, allows the memory interface I/O system to save up to 35 percent compared with a similar-speed DDR3 system.

UltraScale also supports a wide range of parallel memories: LPDDR3, RLDRAM3, QDRII+ and QDRIV, in addition to DDR3 and DDR4. In the serial memory space, UltraScale supports HMC and MoSys' Bandwidth Engine with up to 120 transceivers—plenty for most applications.

In addition, UltraScale has internal improvements to increase memory interface performance and FPGA I/O bank utilization. To improve utilization, Xilinx increased the number of I/Os per bank, with two PLLs for each I/O bank. In addition, there is a finer tap-delay capability of 5 picoseconds. Also, 4-byte lanes are supported per I/O bank, 13 pins per lane. Xilinx has also added circuitry for pre-emphasis and equalization of I/Os.

Quad-ranked DIMM modules and x4 devices are now supported in this generation, quadrupling the possible memory access depth. Improvements in physical-layer (PHY) latency give faster access to data. This grand list of improvements means that the memory architecture will be optimized to meet the performance demands of your market.

UltraScale FPGAs support every major memory standard. The Vivado® Design Suite with MIG IP wizard enables faster implementations to give customers the advantage they need to focus on solutions instead of problems.

— *Tamara I. Schmitz*
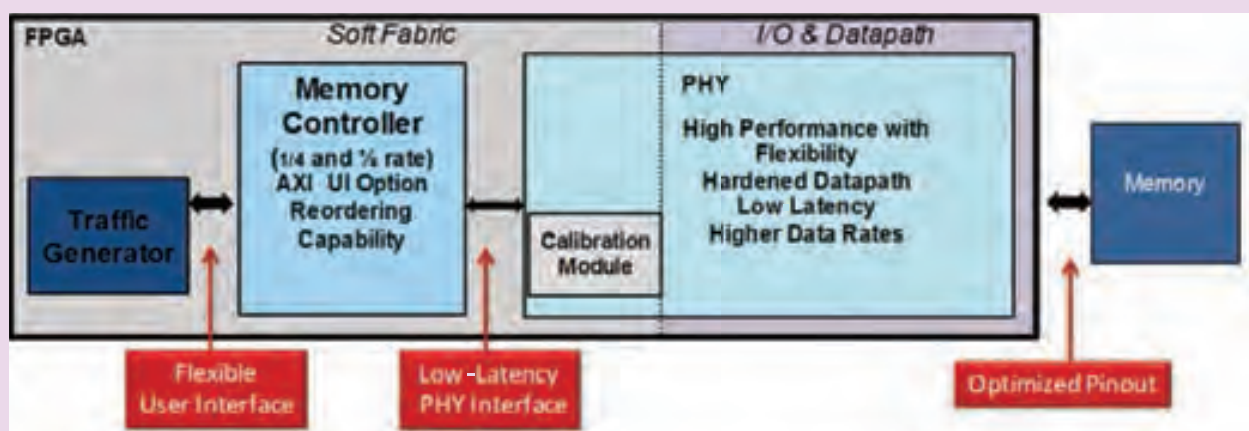
The UltraScale memory interface showcases a number of improvements, aiding both system design and system performance.

DDR3 / DDR4

LPDDR4

Serial Memory

- ✚ Well known, popular in wireless
- ✚ Incrementally more expensive than DDR4
- ➖ Short lifetime (6 – 9 months)
- ➖ Consumes I/O pins
- ➖ 3,200 Mb/s max

- ✚ Longer lifetime
- ✚ Leverages serial technology
- ✚ 15 Gb/s max
- ➖ New and unfamiliar
- ➖ Expensive

Figure 3 – Designers must consider the trade-offs of LPDDR4 vs. a serial memory like HMC.

Consumers also want more speed. Running a system at a higher speed has implications on the board design. DDR is a memory with a single-ended signal that needs proper termination. The faster you operate the system, the shorter the traces need to be from the memory to the FPGA to ensure proper functionality. This means that the devices themselves need to be placed in closer proximity to the FPGA. The limited distance from the FPGA will limit the number of memory devices that you can use in a design. Many DDR4 designs will have approached their limit, packing as many devices around the FPGA as possible.

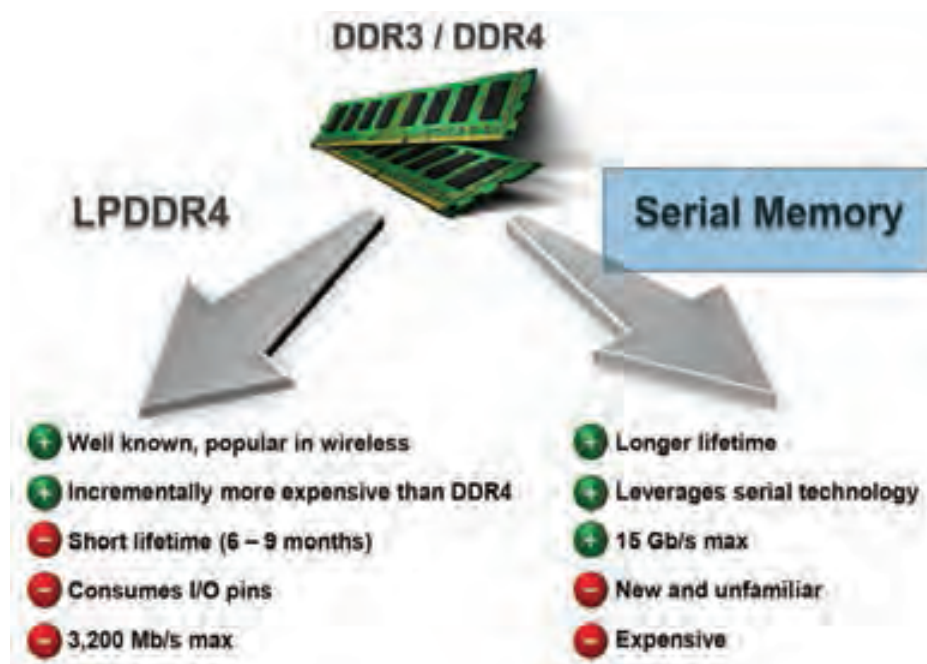If you want more memory, you need more devices. If you want to go faster, you have to bring things closer. There's a limit to how many memory devices you can cram into a fixed amount of space. Any speed improvements in DDR5 would reduce the area available for the memory devices, reducing the available capacity.

WHAT SHOULD SUCCEED DDR3?
Will DDR4 completely replace DDR3? Probably not in all cases. Trends show that the server market is adopting DDR4 while the lower cost of DDR3 continues for now to make it the predominant choice in the personal-computing segment. There is no doubt that consumer appetite will continue to

grow for more speed as well as more memory capacity, and eventually PCs will migrate to DDR4 down the road.

If there is no DDR5, then what other options are available? The most likely choice to replace DDR3 and DDR4 is LPDDR4. The LP stands for "low power." Low-Power DDR4 is actually a type of double-data-rate memory that has been optimized for the wireless market. Advantages of LPDDR are that it's popular, it's well known, the specs are defined and it's available. The low-power optimization makes LPDDR4 only a little more expensive than DDR, and it still uses the I/O pins that DDR uses. That makes for ease of migration, because LPDDR4 runs in the same frequency range that DDR runs in.

However, the biggest trade-off is its lifetime. Since the wireless market turns over its products every six to nine months, LPDDR memories change fast, too. If a big company sells products for 10 to 15 years, it is difficult to accommodate a memory that changes every six to nine months. Possibly a manufacturer could guarantee to deliver one version of those devices for that company for 10 to 15 years under a special agreement. Currently, that business model doesn't exist and special arrangements would have to be made. Of course, these arrangements could include preserving a process flow, an expensive endeavor that might be worth it only for the largest of opportunities.

IF NOT LPDDR, THEN WHAT?
There are other memory options besides LPDDR vying for the opportunity to be the next memory of choice. Serial memory is emerging as a viable alternative and is a completely different way of looking at the memory space (Figure 3).

As far as an FPGA goes, memory is the last frontier, the last section to go serial. The reason for that is latency. The time it takes to turn the data from a parallel stream into serial, sending it down the serial link to then turn it back from serial to parallel, always took too long. Now, the trade-offs from using the serial link are tolerable in some applications (such as those where there are multiple writes and few reads, like a test-and-measurement system for a CT scanner or a set of telescopes scanning the sky). On the other hand, if the measurement of quality is to write data and immediately read that same data, then serial memory will not perform as well as parallel data in any form. However, if

the measure of good memory is high bandwidth, storing lots of videos or sending loads of information over the Internet, then serial memory is tempting.

Latency aside, the same trade-offs deserve investigation. Lifespan is not a problem; these products will be made as long as there is an appetite for them, in comparison with the shorter availability of LPDDR. In fact, if the desire for serial memory grows, multiple vendors will likely join in the business of making it.

Instead of using I/O pins, serial memory leverages serdes technology. In FPGAs, it is possible to use serial interfacing (transceivers) to run at high rates. More recently, based on the need to reduce latency, vendors have addressed those performance concerns as well. This well-developed serial technology can support very high throughput of 15 Gbits per second. The next generation (in the case of HMC) is planned to reach 30 Gbps. People like "new" but at the same time they are scared of the unfamiliar. On the other hand, with newness come limited production rates and higher initial prices.

HYBRID MEMORY CUBE (HMC)

The strongest serial-memory candidate to replace DDR DRAM, the Hybrid Memory Cube (HMC), is being promoted by the HMC Consortium and spearheaded by Micron (see Figure 4). The backers have done a fantastic job of advertising HMC. People have even begun using the acronym to stand for "serial memory" in general. But it, in fact, HMC is just one type of serial memory.

In addition to HMC, MoSys is developing its Bandwidth Engine, a sort of serial SRAM, and Broadcom offers a range of serial-interface TCAMs. At the other end of the future spectrum, Samsung and SK Hynix are promoting High-Bandwidth Memory (HBM), which is a TSV-based DRAM stack with a massively wide parallel interface. This choice might seem lower risk, since it uses a parallel interface.

At this point, however, HMC is the strongest contender to take market share from DDR3 and DDR4. HMC has four or eight stacks of DRAM connected together with TSV technology on top of a logic layer to create the 2G or 4G package. The logic layer creates a convenient interface.

You can daisychain up to eight devices if you need more capacity. There is 256-bit access and enormous throughput considering the one- to four-link capability (in steps of half a link). Each link comprises 16 transceivers (eight for a half link)—all capable of handling 15 Gbps. That is an extraordinary amount of bandwidth previously unavailable to memory designers.

To see the improvement in bandwidth over the DDR solution, see Table 1, which presents three designs. Each of the three (DDR3, DDR4 and HMC) is sized to support 60 Gbps. Notice that pin count is lowered in the HMC solution by at least a factor of eight, greatly reducing board complexity and routing (illustrated in Figure 5). The high bandwidth of the serdes links allows fewer devices, one in the case noted. This single device and an FPGA deliver almost a factor of 20 in reduced board space. Finally, the HMC solution consumes one third of
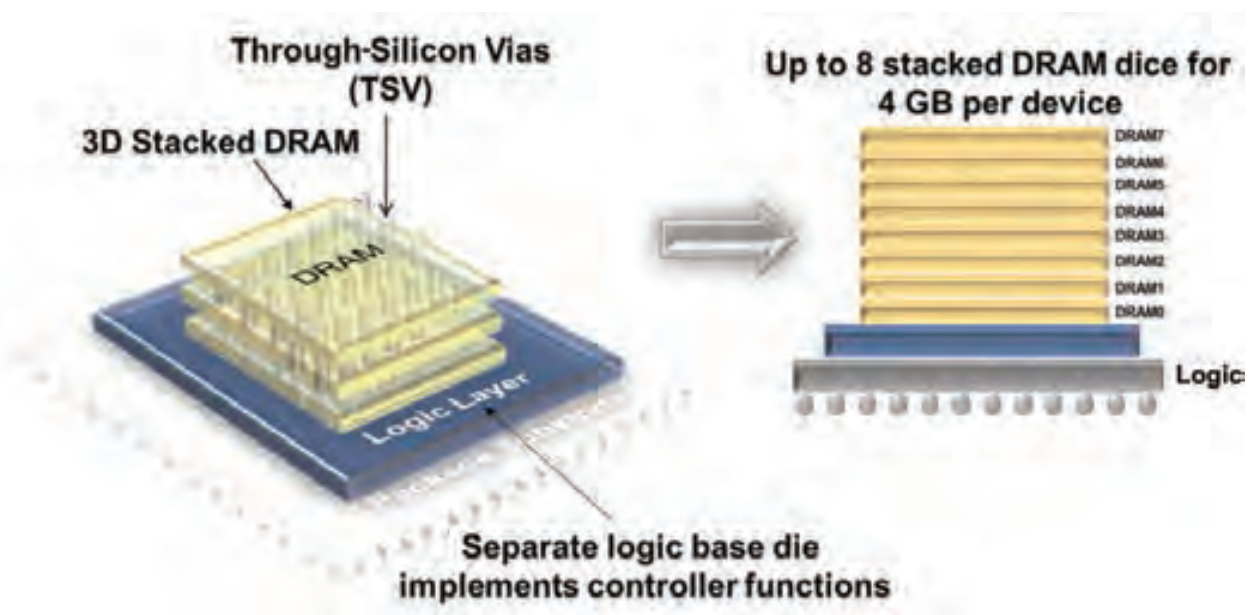
Figure 4 –The Hybrid Memory Cube is based on a through-silicon via (TSV) structure.

	DDR3	DDR4	HMC
Pin counts (power and ground not included)	715	592	70
Board area	8,250 mm^2	6,600 mm^2	378 mm^2
Power (memory + FPGA)	49 pJ/bit	34 pJ/bit	36 pJ/bit
Bandwidth	18 MB/pin	29 MB/pin	857 MB/pin

Table 1 – Comparison of resources needed in three memory types to support 60 Gbps

the power per bit. These are compelling numbers that have observers envisioning HMC capturing a portion of the market previously earmarked for DDR4.

OTHER SERIAL MEMORIES
Because "HMC" and "serial memory" are often mistakenly used interchangeably and sometimes even to represent any new high-bandwidth memory, it is useful to explore some of the other new memories that are coming down the pike. The three top contenders in this category are the Bandwidth Engine by MoSys, TCAM by Broadcom and HBM promoted by Samsung, SK Hynix and Intel.

Bandwidth Engine (BE2) by MoSys is like a serial SRAM, not a serial DRAM, using the transceivers to achieve 16 Gbps. However, BE2 is not a likely replacement for DDR. Instead, with its 72-bit access and lower latency, the technology targets QDR or RLDRAM. The application would be storage for packet headers or a lookup table instead of a packet buffer as in the case of DDR.

TCAM stands for ternary content-addressable memory. This special high-speed memory performs broad searches of pattern matching found in high-performance routers and switches. The high performance is paid for with expense, power and heat. Besides being high speed, TCAM is parallel in nature—it doesn't use serdes to reach those speeds. However, Broadcom is

Figure 5 – Mockup of a 2x100GE design with DDR3 on the left and HMC on the right.
The board-area savings and simplification in routing are attractive.

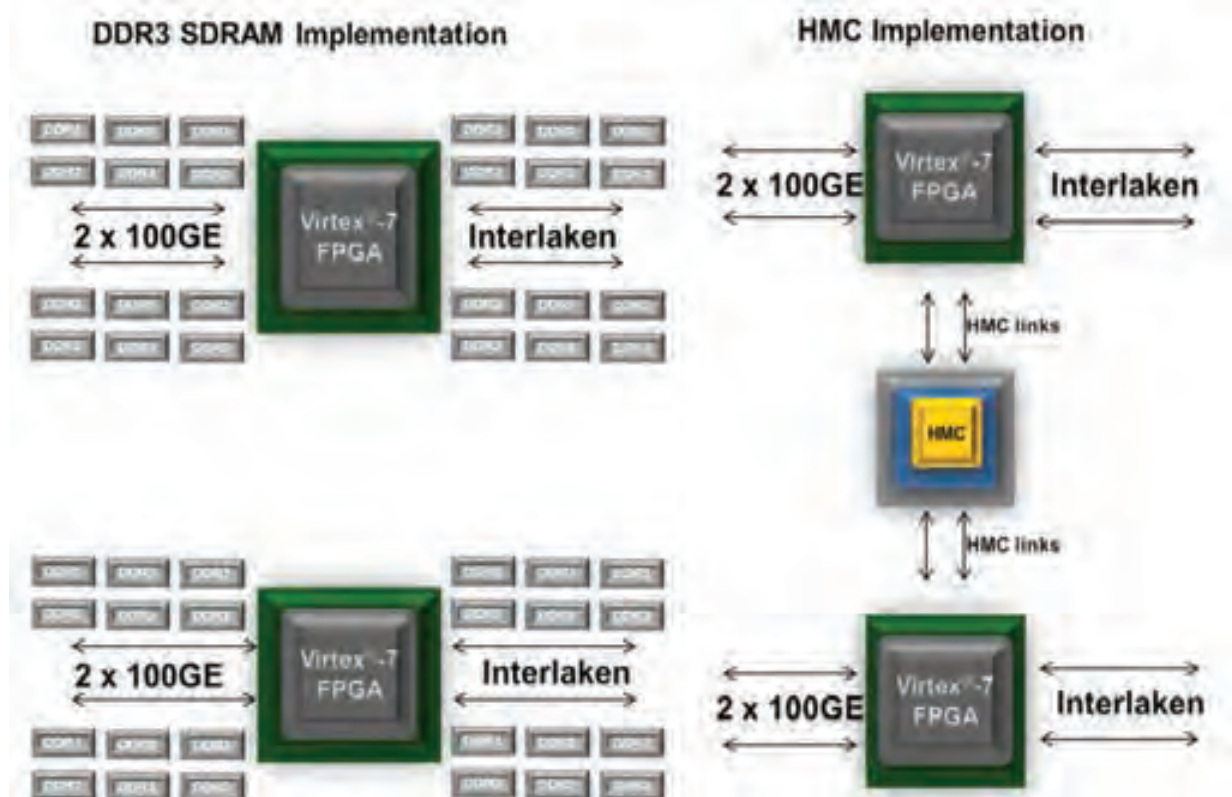offering serial versions of this memory. This way the advantages found in serial memories of low pin count and high speeds can still be associated with a TCAM solution.

The third memory type is HBM. Don't be fooled by the occasional debate between HMC and HBM. What people don't realize is that you can't buy an HBM device. If you want to adopt HBM, you'd in fact be buying a die from, for example, SK Hynix, and have to mount that die inside your package on an interposer, or silicon substrate. Connections between your device and the memory would need to be included in the interposer design to enable this high-bandwidth, parallel memory.

For this memory type to take over the market, companies would need to decide what they want to share in terms of trade secrets and would also have to agree on standards adoptions (interposer design, heights, interfaces, tolerances, etc.). Those details can be worked out, but have not been as yet. On the other hand, HBM's latency will be small. That's because the electrons will travel ridiculously short distances since they're within the package. It's a fantastic idea, but further out into the future.

MOVING INTO PRODUCTION

Success for any or all of these solutions would ensure that more suppliers will join these pioneers to serve the industry. The one option that is in production now is MoSys' Bandwidth Engine, BE2. HMC is sampling, and will be in full production by the end of the year. LPDDR4 will be sampling by the middle of this year. HBM is not available as a standalone package, though there are talks about the possibility of serializing HBM into its own package. If you want to buy a die and integrate HBM into your package, you can always talk to Samsung or Hynix or other, smaller vendors, which customers are doing right now.

The takeaway is that DDR3 is here and is very strong, while DDR4 is still in its growth-and-adoption phase. DDR4 will also experience its own staying power that will probably stretch longer than the popular DDR3 simply because it is the last offering in a line of extremely successful memories. LPDDR4 is the most likely candidate to fill the gap, but won't replace DDR4 in all areas unless there are very rapid read/write iterations.

Otherwise, serial memory is the newcomer to watch. HMC is poised to replace DDR while Bandwidth Engine is the serial solution replacing QDR and RLDRAM.
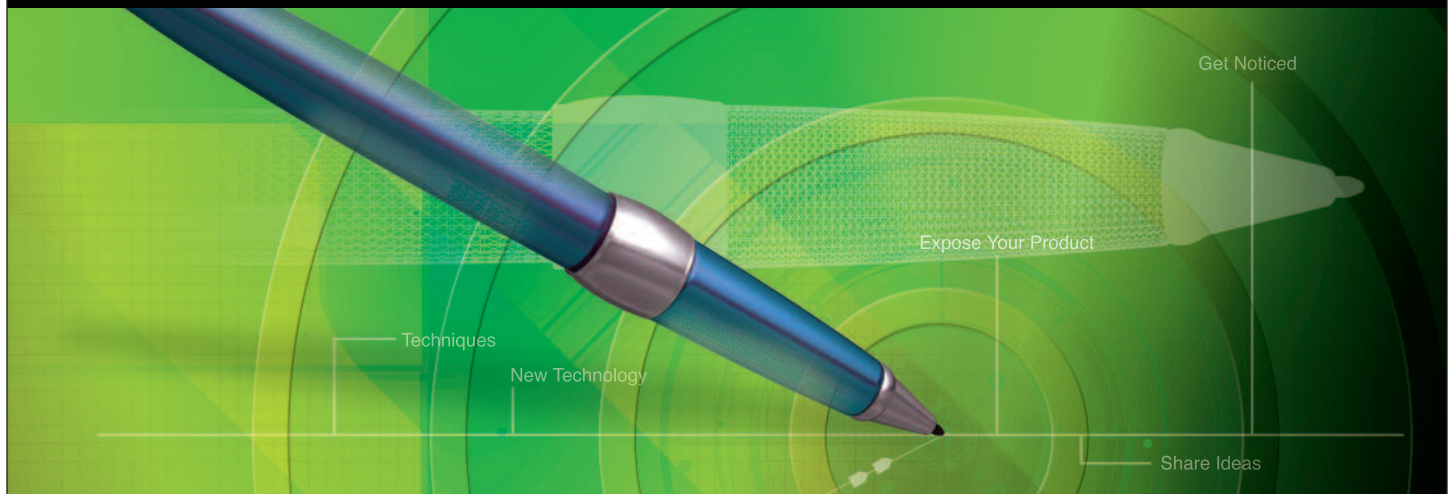
Design Reliability:
MTBF Is Just the Beginning

Reliability will be one of your major design concerns, no matter what your end application is. There are many ways to achieve it.
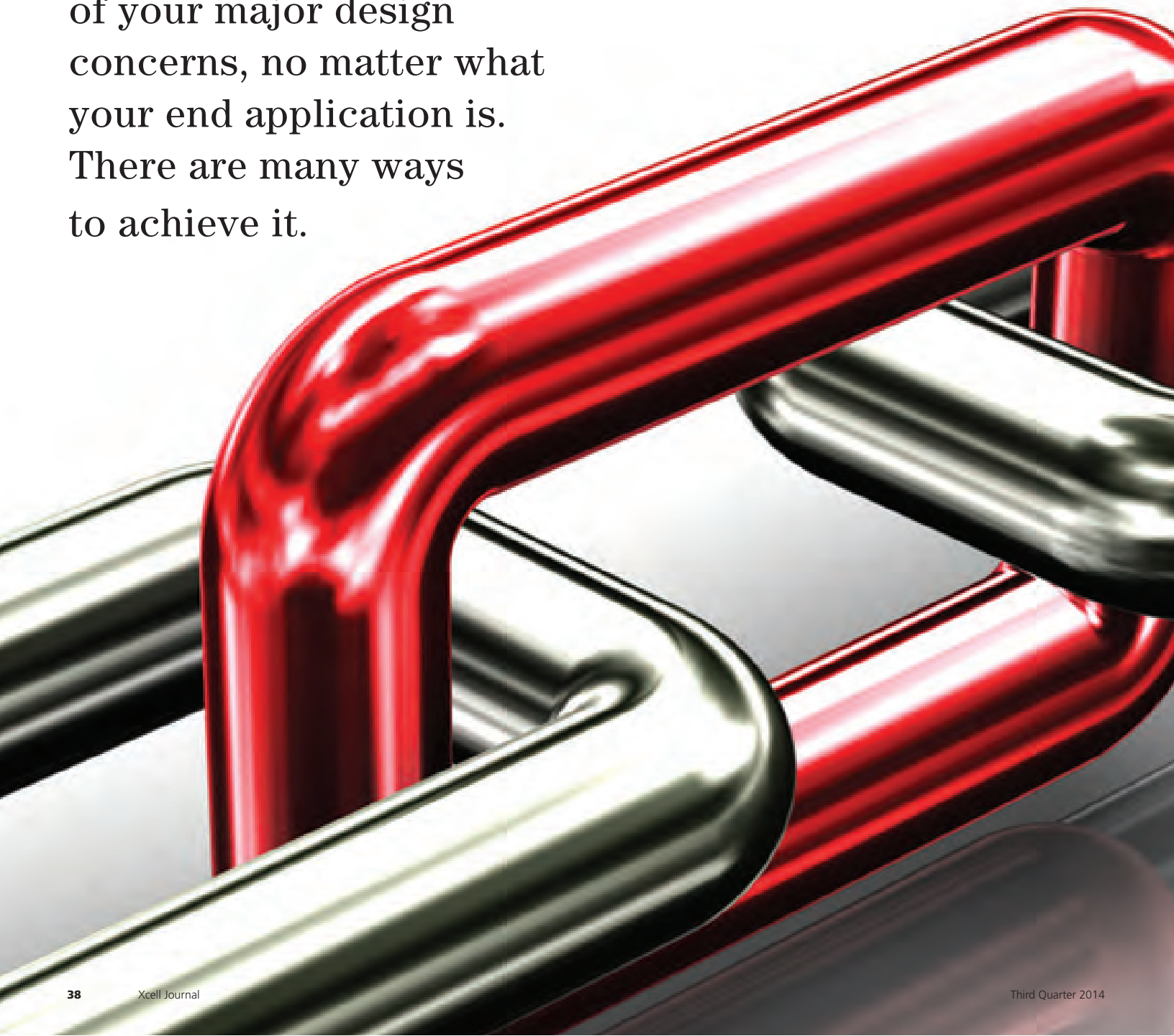
by Adam P. Taylor
Head of Engineering – Systems
e2v
aptaylor@theiet.org

W

When most engineers think about design reliability, their minds turn to a single, central metric: mean time between failures. MTBF is, in fact, an important parameter in assessing how dependable your design will be. But another factor, probability of success, is just as crucial, and you would do well to take note of other considerations as well to ensure an accurate reliability analysis and, ultimately, a reliable solution.

The need for reliability exists in every product you will design for different reasons, depending upon the end application. Aerospace and military designers must ensure both the safety of the operators/passengers and the success of the mission. In telecommunications, the goal is to ensure no outage of service, which impacts revenue flow and reputation. The job for industrial and process-control engineers is minimal downtime, safety and fail-safe operation in the event of a failure. And in commercial applications, designers must make sure their products achieve the prescribed warranty periods.

The use of FPGAs allows for a more-integrated solution, which can as a result increase the MTBF of the system. This is especially true when the device's manufacturer makes available a regular quarterly reliability report, as Xilinx does, published as UG116.

At the highest level, there are two ways to think about reliability. First is the confidence the system will operate for the required lifetime. Here is where the MTBF, probability of success and the familiar bathtub curve are of use. The second consideration is ensuring that, should an erroneous event occur, your design will either continue to

function and remain fail-safe or issue a report on an impending problem. The way we engineers perform the design and analysis can affect both of these aspects of reliability.

To ensure a reliable solution, your development environment must contain the correct engineering governance with review gates, design rules and guidelines, along with independent peer reviews at appropriate points in the life cycle.

MTBF AND THE BATHTUB

The definition of MTBF is the statistical prediction of the time between failures while a system is operating. Manufacturers calculate the MTBF by taking the reciprocal of the sum of the individual component failure rates. These failure rates are generally referred to as the FIT rate, where a failure in time (FIT) equals $1e^{-9}$ hours^{-1}. You can obtain these failure rates from the component supplier or by using a standard such as the Military Handbook MIL-HDBK-217F or the Bellcore/Telcordia SR332. The relationship between MTBF and failure-in-time rate is shown below.

$$MTBF = \frac{1}{Failure\ in\ Time\ rate}$$

However, these failure rates are only valid for the constant-failure-rate period of the bathtub curve, as shown in Figure 1.

The bathtub curve maps early ("infant mortality") failures at a product's introduction, failures that occur during its normal lifetime ("constant failure rate") and failures at the end of a product's design lifetime. It is therefore common to perform some form of "burn-in" during manufacturing to screen out infant-mortality failures. The temperatures experienced during burn-in accelerate latent defects within the device, ensuring the device fails before delivery and inclusion in a system.

You can determine where your product or system is located within the bathtub by performing a Weibull, or life data, analysis, which is easy to

To ensure an acceptable probability of success, many products require an MTBF significantly higher than the intended operating life.

do within Excel. The shape parameter β indicates whether the failure rate is stable, increasing or decreasing. A shape parameter (β) which is less than 1.0 indicates a decreasing failure rate experienced during the infant-mortality period, while a shape parameter of greater than one indicates an increasing failure rate, as would be seen within the wear-out phase.

Having determined where you are on the bathtub curve, you might be forgiven for feeling assured that the system will therefore continue to operate successfully for at least the MTBF. However, this is not the case. The MTBF is a statistical representation of the failure rate which can be expected during the useful life of the product; it does not represent the predicted operating life

of the product. To obtain the predicted operating life, we need to consider the probability of success as demonstrated by the equation below, where t is the desired operating time in hours.

$$P(s) = e^{\frac{-t}{MTBF}}$$

Plotting the probability of success shows that as the desired operating time approaches the MTBF, the probability of success is approximately 0.37, as demonstrated in Figure 2. This means that the probability a single module will still be working at an elapsed time equal to the MTBF is 0.37. If you are considering a batch of units, then 37 percent of them will still be functioning.

Therefore, to ensure acceptable probability of success for the mission life,

many systems/products require an MTBF significantly higher than the intended operating life. For example, assuming a five-year operating life with a 0.99 probability of success, a product would require an MTBF of 4,361,048 hours or 497 years, as the equation shows.

$$MTBF = \frac{P(s)}{-\ln(t)}$$

Clearly, that's considerably longer than the operating life.

CALCULATING RELIABILITY
You can calculate reliability and MTBF by one of two methods—either a parts-count analysis or a part stress analysis. The parts-count analysis is the simpler of the two and is sometimes performed early in the development cycle as an in-
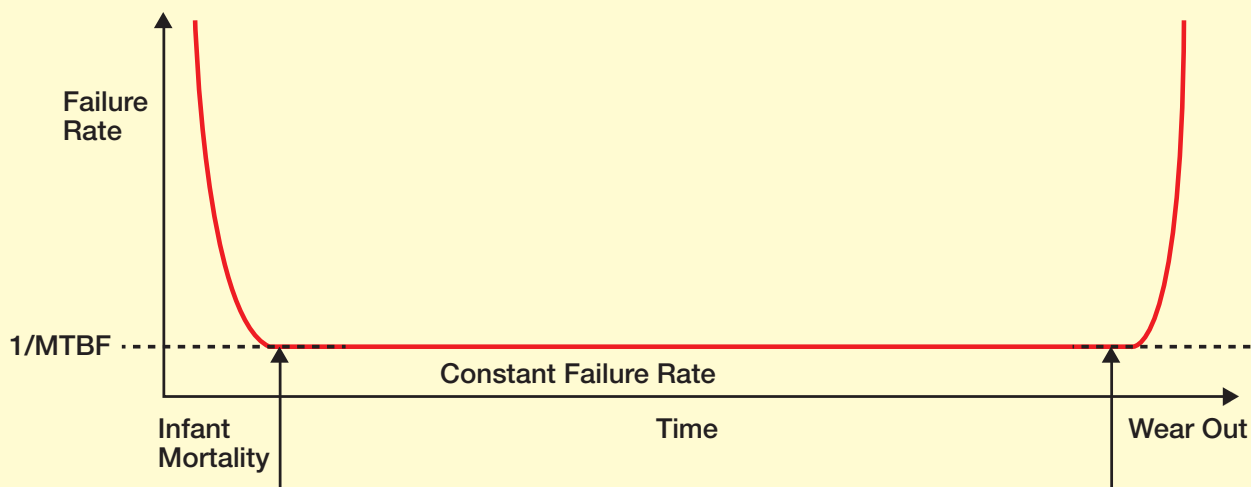

Figure 1 – The bathtub curve tracks early ("infant mortality") failures at product introduction, those during its useful life and "wear-out" failures at the end of life.
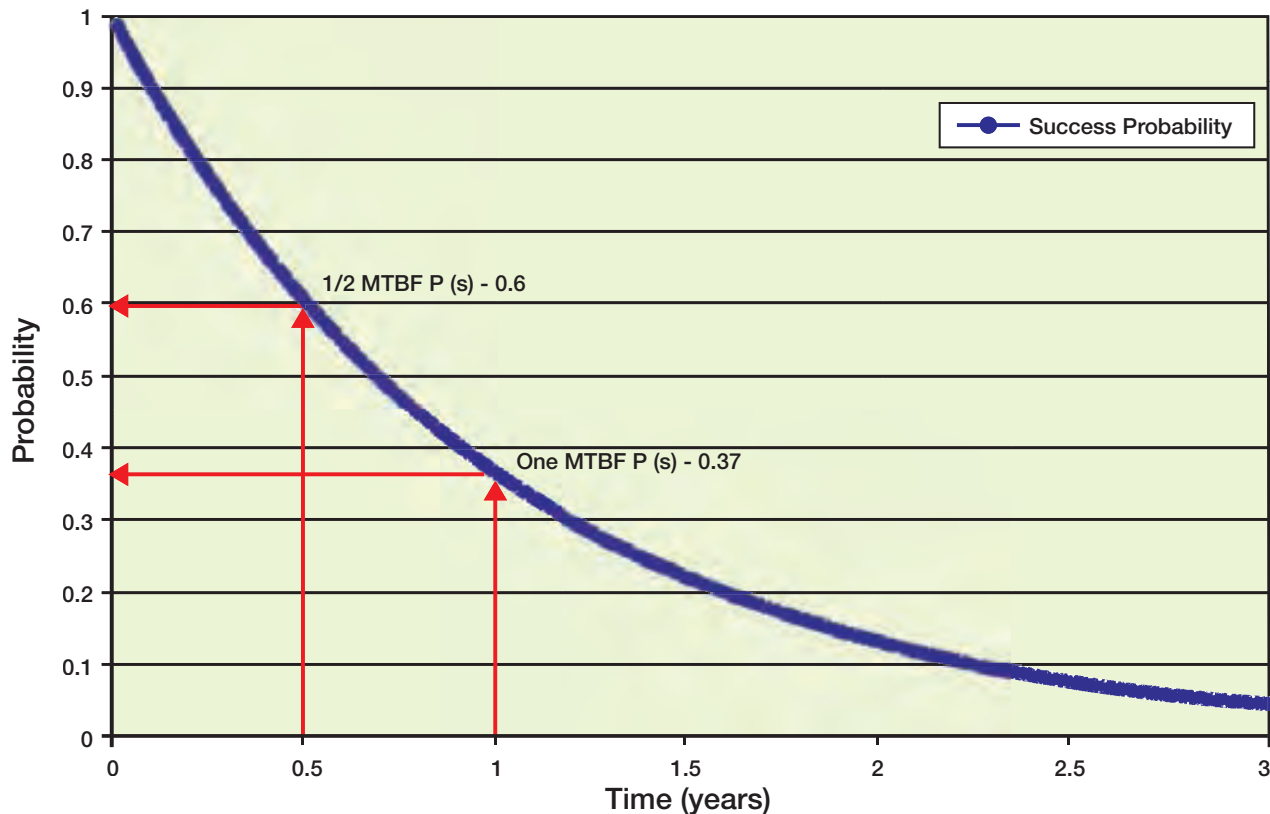
Figure 2 – As the desired operating time approaches the MTBF, the probability of success is 0.37.

dication the product will achieve its reliability requirements. This type of analysis takes into account the part's quality level, quantities and the environment of use. You can perform a parts-count analysis fast. However, the results tend to be on the pessimistic side, resulting in a higher failure rate and lower MTBF.

A part stress analysis will take into account a much wider number of parameters and as a result will take far longer to perform, but this type of analysis results in a more accurate figure. A stress analysis will consider multiple factors including temperature, electrical stress, quality, construction, operating environment and many more, depending upon the type of component you are analyzing. This analysis will result in much more accurate failure rate for the application at hand.

INCREASING THE RELIABILITY
There are many methods and techniques available that will help you increase the MTBF and hence the prob-

ability of success for your product or system. The most commonly used tactic is to derate the electrical and thermal stresses upon the components. This derating allows you to take the device stresses into account when performing the part stress analysis described above. Often companies have their own derating rules. However, you can use industry-standard rules as references if in-house regulations are not available. Two examples are the European Space Agency's ECSS-Q-30-11A and the U.S. Navy's NAVSEA TE000-AB-GTP-010.

While performing part stress analysis increases the nonrecurring-engineering cost, there are other choices available to the engineering team that will have an impact upon the recurring cost.

The first option is to increase the quality of the components while applying similar derating rules. This could mean an increase from standard commercial parts to military (QML Q for ICs) or even space (QML V for ICs) qualified compo-

nents. Be warned, however—as the quality level of the component increases, so can the price. Table 1 shows the different standards required for integrated circuits, hybrids and discrete components.

The second option is to introduce redundancy either intramodule or intermodule. Redundancy increases the size, weight and cost of the solution, but the payoff is a dramatic effect upon the reliability of the system and, hence, the system availability. It is often best to make a decision on redundancy at the system level and introduce the extra components in areas of the system where the failure rate is high. This targeted approach to redundancy will result in an optimized solution.

When considering redundancy, you can go either hot or cold. In a "hot" redundant solution, the redundant system is powered, configured and can be seamlessly switched in to replace a failing module with no impact on system performance. The downside is that in such a scenar-

Type	Standard	Military	Space
Integrated Circuits	MIL-PRF-38535	QML Q (Class B)	QML V (Class S)
Hybirds	MIL-PRF-38536	Class H	Class K
Discrete	MIL-PRF-19500	JAN TXV	JAN S

Table 1 – Standard, military and space versions exist for ICs, hybrids and discrete components.

Parameter	Comment
Temperature	Units and critical component
Current	Drawn from main supply
Voltages	The health of subregulated voltages within the system
Redundancy Switching	Reports position of any switches used to route prime/redundant signals
Processing Status	Results of CRC, over- and underflow of calculations, signals out of range, etc.

Table 2 – Chart details aspects of health status monitoring.

io, the redundant equipment is experiencing stress.

In a "cold" redundant scheme, the redundant system is not normally powered and will only spring to life following the failure of the prime module. The system will halt its activity until the redundant side can be reconfigured to continue the work of the now-failed module. While there is an interruption in service, the advantage is that a cold redundant solution does not age, since it is not subject to electrical stress while it is unpowered.

With the introduction of redundancy, you must take care to ensure there is no propagation of faults that will affect the redundant module performance should the prime side fail.

SYSTEM-LEVEL CONSIDERATIONS

Having considered parts quality and the significant effects of redundancy upon the system, there are other options you might want to implement to ensure correct performance under erroneous or failure events. They include:

• Propagation of dangerous failure modes

• Built-in test, telemetry and event logs to monitor and record the health of the system

• Equipment interfaces, whether single connector or prime and redundant

• Critical command sequences (for example, separating the system's "arm" and "fire" commands)

• Acceptable error rates (BER, ECC) on memories and data links

As part the regulatory or certification standards, you must perform a hazard analysis to determine the potential perils that could occur should the unit fail. It is therefore your responsibility to ensure

the design at a system level takes appropriate actions to prevent these hazards by means of interlocks and the like. If necessary, you should flow these mitigating actions down to subsystems as defined requirements so as to ensure these failure modes are correctly addressed.

Knowing the health status of the equipment and reporting or recording this status is a technique you can use for prognostics, ensuring fail-safe operation and determining why the equipment failed to help hasten its repair. More complicated systems may include a comprehensive self-test functionality that can run on power-up or continuously during operation. Table 2 shows a more detailed interruptive test on demand aspects that you may want to consider monitoring.

These results can be transmitted as health status over a communications link, stored within a nonvolatile memory—for instance, flash or FRAM—or both. Often, you might choose to use a real-time clock or elapsed-time counter to tag these events with a time occurrence so as to provide a frame of reference.

Another area of concern facing engineers in harsh environments is connectors. These components are a common point of failure, since individual cables within can break or the connector itself can fall off due to environmental effects such as vibration or shock. Therefore, you can build in greater reliability by introducing a redundant connector
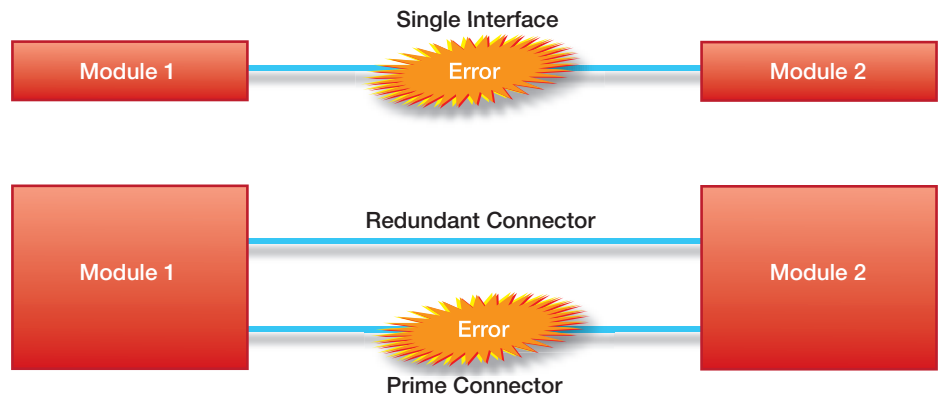

Figure 3 – A redundant connector takes over if the original connector fails— but at the cost of increased complexity.
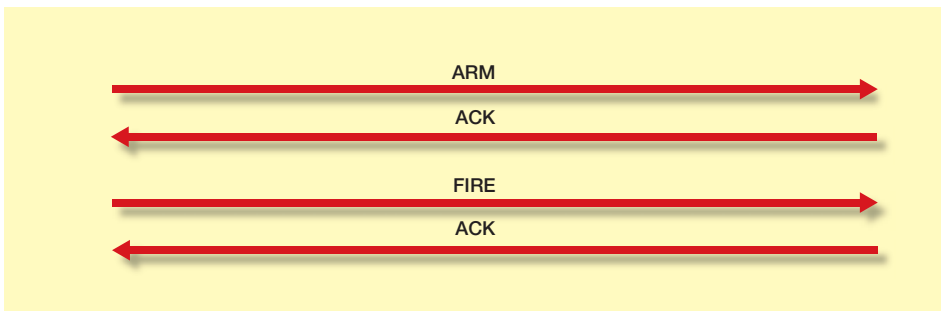
Figure 4 – An "arm and fire" sequence is useful in electrically noisy environments.

and cable. In the event of failure of the first connector, the redundant one takes over communications as shown in Figure 3. However, this redundancy comes at the cost of increased complexity, especially when you need to interconnect a number of modules. One alternative approach is to utilize a connector designed especially for harsh environments such as the MIL-STD 38999 series connectors.

If the system or product is to be used within a harsh environment—for instance, one that's electrically noisy—it may be prudent to consider an arm/fire approach on commands transmitted on buses within the system. In this scheme (see Figure 4), an initial command is transmitted to the receiver, which then acknowledges the command and starts a timeout. The receiver will issue a NACK (negative acknowledge character) command if it does not get the fire command, in response to which the receiver issues an ACK before the timeout occurs. Likewise, if the receiver receives any other command it will issue a NACK and the process starts again. This scheme ensures the corruption of one command due to electromagnetic interference (EMI) cannot inadvertently cause critical commands to occur.

In a similar manner to the arm-and-fire approach, you may also want to make sure all communication links and memories have error-correcting and detection codes to ensure reliable communications and storage of data. The choice of error detecting alone, vs. error detecting and correcting, will depend upon the end application. However, there are a number of codes you can use scaling from the very simple to the more complicated (Table 3). The level of protection also scales with the code's complexity.

All engineers, regardless of the end application they are working on, need to consider the reliability of their end system. There are many techniques at the engineer's disposal that will help in the quest to implement a reliable product.

Code	Error Correction	Error Detection	Comment
Parity	X		Errors can be masked
N of M	X		Not suitable to multiple bit
CRC	X		Good for burst errors
BCH	X	X	Easy to Decode
Hamming	X	X	One of First EDC
Reed Solomon	X	X	Special case of BCH

Table 3 – EDAC codes scale from simple to complex.

Protocol-Processing Systems Thrive with Vivado HLS

by Kimon Karras
Research Engineer
Xilinx, Inc.
kimonk@xilinx.com

James Hrica
Senior Software Applications Engineer
Xilinx, Inc.
jhrica@xilinx.com

Designers can use Xilinx's high-level synthesis tool to describe packet-processing systems in a software-like manner with high-level programming constructs—something that's difficult in RTL.
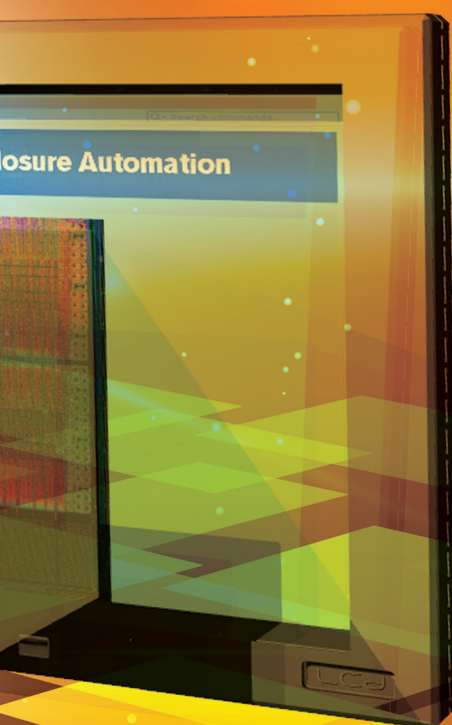
Protocol processing on different levels is present in any modern communication system, since any exchange of information requires the use of some communication protocol. This protocol typically contains packets, which have to be created by the sender and reassembled at the receiver, all while adhering to the protocol specifications. This makes protocol processing ubiquitous and thus of special interest to FPGA designers. Hence, implementing protocol-processing functionality efficiently is of profound importance in FPGAs.

Designers have used high-level synthesis in the video- and signal-processing space with considerable success. HLS lets you use a high-level programming language to express hardware functionality. To test how the technology would work with packet processing, we built a prototype system completely with the Xilinx® Vivado® HLS tool, producing really exciting results. Vivado HLS halved our development time, reduced the resources we used and lowered latency. Our example system was a simple ARP/ICMP server that replies to ping and Address Resolution Protocol (ARP) requests, and resolves IP address queries.

Let's take a closer look at how Vivado HLS can solve some of the key problems designers encounter when processing protocols. To understand the benefits of this technology, it's helpful to first examine Vivado HLS in detail to understand how it works.

RAISING THE ABSTRACTION LEVEL

Vivado HLS provides tangible benefits for designers by raising the level of abstraction in system design. It does this in two main ways.

- Utilizing C/C++ as a programming language and leveraging the high-level constructs available in such a language

- Providing additional data primitives, which allow the designer to easily use basic hardware building blocks (bit vectors, queues, etc.)

These characteristics allow a designer using Vivado HLS to tackle common protocol system design hurdles much more easily than when using RTL. The result is to ease system assembly, simplify FIFO and memory access, and enable abstraction of the control flow.

Easy architectural exploration and simulation are other major benefits of this technology.

Vivado HLS treats C++ functions as modules, with the function definition being equivalent to an RTL description of the module and a function call being equivalent to a module instantiation. This scheme dramatically simplifies the structural code describing the system by reducing the amount of code that you must write, thus speeding system assembly.

You can access a memory or a FIFO in Vivado HLS either through methods of an appropriate object (for example, the read and write methods of a stream object) or simply by accessing a standard C array, which synthesis then implements either as Block RAM or distributed RAM. The synthesis tool takes care of the additional signaling, synchronization or addressing as required.

In terms of control flow, Vivado HLS provides a set of flow-control-aware interfaces ranging from simple FIFO interfaces to full AXI4-Stream. In all of these interfaces the designer simply accesses the data without having to check for back pressure or data availability. Vivado HLS will schedule execution appropriately to take care of all contingencies, while ensuring correct execution.

Designers will also appreciate the easy architectural exploration that Vivado HLS makes possible. You can insert pragma directives in the code (or Tcl commands when using the GUI or batch mode) in order to communicate the desired features of the design to the synthesis tool. In this way, you are able to explore a great swath of architectural alternatives without requiring any changes to the implementation code itself. The options can range from fundamental issues, like the pipelining of a module, to more mundane ones such as the depth of a FIFO queue.

Finally, C and RTL simulation is another area where Vivado HLS shines. Designs are verified using a two-step process. The first step is the C simulation, in which the C/C++ is compiled and executed like a normal C/C++ program. The second step in verification is the C/RTL co-simulation. Here, Vivado HLS automatically generates an RTL testbench from the C/C++ testbench, and instruments and executes an RTL simulation that you can use to check the correctness of the implementation.

By leveraging all of these advantages, you will reap considerable benefits for your system design, not only in development time and productivity but also in code maintainability and readability thanks to the more compact nature of the Vivado HLS code. Furthermore, with high-level synthesis you still maintain control over the architecture and its features. Correct understanding and use of Vivado HLS pragmas is fundamental to realizing this control.

High-level synthesis thus occupies an intermediate slot in the hierarchy of Xilinx-offered packet-processing solutions. It is complemented by Vivado SDNet (see cover story, _Xcell Journal_ Issue 87), which uses a domain-specific language to offer a much simpler, albeit more constrained, way of expressing protocol-processing systems; and RTL, which allows for the implementation of a considerably wider breadth of systems that Vivado HLS is not able to express (e.g., systems requiring detailed clock management using DCMs or differential signaling). These caveats notwithstanding, Vivado HLS is an efficient way to implement the vast majority of protocol-processing solutions, without compromises or concessions in the quality of results or in designer flexibility.

SETTING UP A SIMPLE SYSTEM

The most basic tasks to be accomplished when starting a new design are, first, determining its structure and next, implementing it in Vivado HLS. In Vivado HLS, the basic building block of a system is a C/C++ function. Building a system consisting of modules and submodules essentially means having a top-level function call lower-level functions. Figure 1 illustrates a very simple three-stage pipeline, which we will use as an example to introduce the basic ideas behind system building in Vivado HLS. Protocol processing is typically performed in pipelined designs, with each stage taking care of a specific part of the processing.
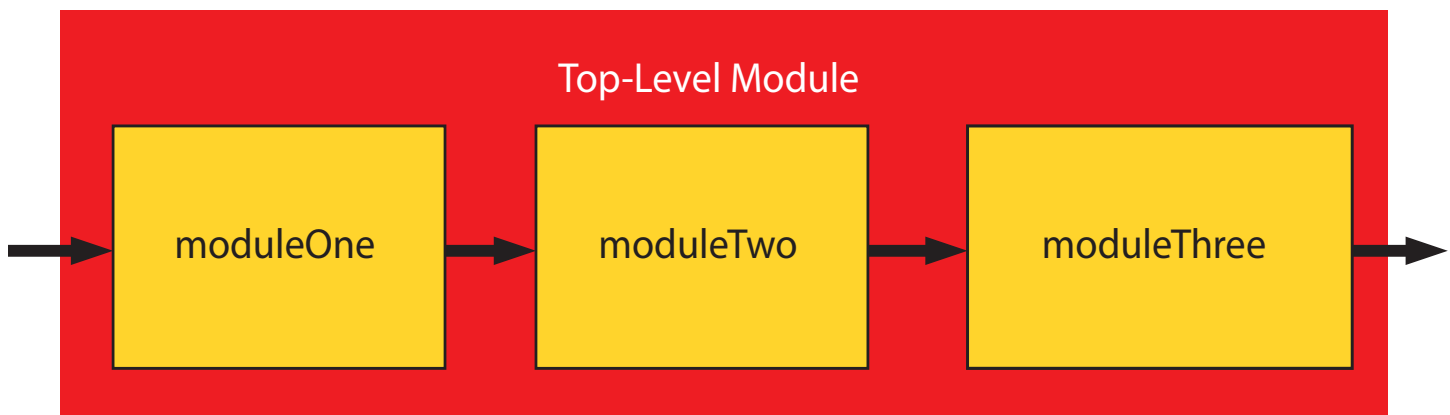

Figure 1 – A simple three-stage pipeline

Building a system consisting of modules and submodules essentially means having a top-level function call lower-level functions.

Example 1 – Creating a simple system in Vivado HLS

```
1  void topLevelModule(stream<axiWord> &inData,
stream<axiWord> &outData) {
2  #pragma HLS dataflow interval=1
3
4    #pragma INTERFACE axis port=inData
5    #pragma INTERFACE axis port=outData
6
7  static stream<ap_uint<64> > modOne2modTwo;
8  static stream<ap_uint<64> > modTwo2modThree;
9
10 moduleOne(inData, modOne2modTwo);
11 moduleTwo(modOne2modTwo, modTwo2modThree);
12 moduleThree(modTwo2modThree, outData);
13 }
```

The code in Example 1 creates the top-module function, which calls all the other subfunctions. The top-module function uses two parameters, both of which are of class *stream* (one of the template classes provided by the Vivado HLS libraries). A stream is an HLS modeling construct that represents an interface over which data is to be exchanged in a streaming manner. A stream can be implemented as a FIFO queue or memory. A stream is a template class that can be used with any C++ construct. In this case, we have defined a data structure (struct) called axiWord, which is shown in Example 2.

Example 2 – Definition of a C++ struct for use in a stream interface

```
struct axiWord {
    ap_uint<64>      data;
    ap_uint<8>       strb;
    ap_uint<1>       last;
};
```

We used this struct to define part of the fields for an AXI4-Stream interface. Vivado HLS automatically supports this kind of interface; you can specify it using a pragma statement. Pragmas are directives to the high-level synthesis tool that help steer the tool so as to reach the required results. The pragmas in lines 4 and 5 of Example 1 tell Vivado HLS that both parameters (essentially, the input and output ports of the top module) are to use AXI4-Stream interfaces. The AXI4-Stream I/F includes two mandatory signals, the valid and ready signals,

which were not included in the declared struct. This is because the Vivado HLS AXI4 I/F will take care of these signals internally, which means that they are transparent to the user logic. As mentioned before, Vivado HLS completely abstracts flow control from the user when using AXI4-Stream I/Fs.

Of course, an interface doesn't have to use AXI4-Stream. Vivado HLS provides a rich set of bus interfaces. We chose AXI4-Stream here as an example of a popular, standardized interface that you can use for packet processing.

The next task in implementing our design is to ensure that our three modules are connected with one another. This task is also done through streams, which are however internal to the top module this time. Lines 7 and 8 declare two streams for this purpose. These streams make use of another Vivado HLS construct, *ap_uint*. This is essentially an unsigned, one-dimensional array of bits that can then be manipulated as such. Again this is a template class; thus, the width of this array also has to be specified. In this case, we used 64 bits, matching the width of the data members of the input and output I/Fs of the top module. A further element that needs to be elucidated here is that these streams are declared as static variables. A static variable maintains its value over multiple function calls. The top-level module (and thus all its submodules) will be called once in every clock cycle when executed as a sequential C/C++ program, so any variables that need to keep their values intact from one cycle to the next need to be declared as static.

CREATING PIPELINED DESIGNS

The last pragma left to discuss is perhaps the most important one. The dataflow pragma in line 2 instructs Vivado HLS to attempt to schedule the execution of all of this function's subfunctions in parallel. The parameter '*interval*' sets the initiation interval (II) for this module. II defines the throughput of the design by telling Vivado HLS how often this module has to be able to process a new input data word. This does not preclude the module being internally pipelined and having a latency > 1. An II = 2 means that the module has two cycles to complete the processing of a data word before having to read in a new one. In this manner, Vivado HLS can simplify the resulting RTL for a module. That being said, in a typical protocol-processing application

The initiation interval defines the throughput of the design by telling Vivado HLS how often this module has to be able to process a new input data word.

the design has to be able to process one data word in each clock cycle, so from now on we will use an II = 1.

Finally, the last piece of the puzzle is the calling of the functions themselves. In Vivado HLS this process corresponds with the instantiation of the modules. The parameters that are passed to each module essentially define that module's communication ports. In this case, we create a chain of the three modules by connecting the input to the first module, then the first module to the second over stream modOne2modTwo and so on.

SETTING UP A SIMPLE SYSTEM

Protocol processing is typically a stateful affair. You must read in successive packet words arriving onto a bus over many clock cycles and decide on further operations according to some field of the packet. The common way to handle this type of processing is by using a state machine, which iterates over the packet and performs the necessary processing. Example 3 shows a simple state machine that either drops or forwards a packet depending on an input from a previous stage. The function receives three arguments: the input packet data over the *inData* stream; a 1-bit flag showing whether a packet is valid or not over the *validBuffer* stream; and the output packet data stream, called *outData*. Notice that the parameters in the Vivado HLS functions are passed by reference. This is necessary when using Vivado HLS streams, which are complex classes. Simpler data types like the ap_uint can also be passed by value.

The pipeline pragma used in line 2 instructs Vivado HLS to pipeline this function to achieve an initiation interval of 1 (II = 1), meaning that it will be able to process one new input data word every clock cycle. Vivado HLS will examine the design and will determine how many pipeline stages it needs to introduce to the design to meet the required scheduling restrictions.

Example 3 – Finite state machine using Vivado HLS

```
1 void dropper(stream<axiWord>& inData,
  stream<ap_uint<1> >& validBuffer,
  stream<axiWord>& outData) {
2 #pragma HLS pipeline II=1 enable_flush
```

```
3
4 static enum dState {D_IDLE = 0, D_STREAM, D_
  DROP} dropState;
5 axiWord currWord = {0, 0, 0, 0};
6
7 switch(dropState) {
8 case D_IDLE:
9    if (!validBuffer.empty() && !inData.empty()) {
10     ap_uint<1> valid = validBuffer.read();
11     inData.read(currWord);
12     if (valid) {
13       outData.write(currWord);
14         dropState = D_STREAM;
15     }
16   }
17   else
18     dropState = D_DROP;
19   break;
20  case D_STREAM:
21   if (!inData.empty()) {
22      inData.read(currWord);
23      outData.write(currWord);
24      if (currWord.last)
25         dropState = D_IDLE;
26   }
27   break;
28  case D_DROP:
29      if (!inData.empty()) {
30        inData.read(currWord);
31      if (currWord.last)
32         dropState = D_IDLE;
33      }
34      break;
35   }
36 }
```

Line 4 declares a static enumeration variable that will be used to express state in this FSM. Using an enumeration is optional but allows for more legible code, because states can be given proper names. However, any integer or ap_uint variable can also be used with similar results. Line 5 declares a variable of type axiWord, in which packet data to be read from the input will be stored.

The switch statement in line 7 represents the actual state machine. Using a switch is recommended but not mandatory. An if-else decision tree would also perform the same functionality. The switch statement allows the tool to more efficiently enumerate all the states and optimize the resulting state machine RTL code.

Execution starts at the D_IDLE state where the FSM reads from the two input streams in lines 10 and 11. These two lines demonstrate both uses of the stream object's read method. Both methods read from the specified stream and store the result into the given variable. This method performs a blocking read, which means that if the method call isn't successfully executed, then the execution of the remaining code in this function call is stalled. This happens when trying to read from an empty stream.

STREAM SPLITTING AND MERGING

Being able to forward packets to different modules according to some field in the protocol stack, and then recombining these different streams again before transmission, is a critical functionality in protocol processing. Vivado HLS allows for the use of high-level constructs to facilitate this forwarding process, as the code in Example 4 illustrates for the case of a stream merging.

Example 4 – Simple stream merge situation

```
1 void merge(stream<axiWord> inData[NUM_MERGE_
  STREAMS], stream<axiWord> &outData) {
2 #pragma HLS INLINE off
3 #pragma HLS pipeline II=1 enable_flush
4
5 static enum mState{M_IDLE = 0, M_STREAM}
  mergeState;
6 static ap_uint<LOG2CEIL_NUM_MERGE_STREAMS>
  rrCtr = 0;
7 static ap_uint<LOG2CEIL_NUM_MERGE_STREAMS>
  streamSource = 0;
8 axiWord inputWord = {0, 0, 0, 0};
9
10 switch(mergeState) {
11   case M_IDLE:
12     bool streamEmpty[NUM_MERGE_STREAMS];
13 #pragma HLS ARRAY_PARTITION variable=stream-
  Empty complete
14     for (uint8_t i=0;i<NUM_MERGE_STREAMS;++i)
15       streamEmpty[i] = inData[i].empty();
16     for (uint8_t i=0;i<NUM_MERGE_STREAMS;++i) {
17       uint8_t tempCtr = streamSource + 1 + i;
18       if (tempCtr >= NUM_MERGE_STREAMS)
19         tempCtr -= NUM_MERGE_STREAMS;
20       if(!streamEmpty[tempCtr]) {
21         streamSource = tempCtr;
22         inputWord = inData[streamSource].
          read();
23         outData.write(inputWord);
24         if (inputWord.last == 0)
25           mergeState = M_STREAM;
26       break;
27       }
28     }
29     break;
30   case M_STREAM:
31     if (!inData[streamSource].empty()) {
32       inData[streamSource].read(inputWord);
33       outData.write(inputWord);
34       if (inputWord.last == 1)
35         mergeState = M_IDLE;
36     }
37     break;
38 }
39   }
```

This example shows the use of a module merge, which has a stream array as input (inData) and a single stream (outData) as output. The purpose of this module is to read from the input streams in a fair manner and output the read data to the output stream. The module is implemented as a two-state FSM, which is described using the same constructs that were previously introduced.

The first state in the FSM ensures fairness when choosing the input stream. We do this by using a round-robin algorithm to go over the queues. The algorithm starts looking for new data from the next queue after the one that was accessed previously. The code in lines 17-19 implements the round-robin algorithm. The constant NUM_MERGE_STREAMS specifies the number of streams that are to be merged. Subsequently, line 20 tests the current stream, which is identified by the tempCntr variable for content. If it is not empty, then this is set to be the active stream (line 21). Data is read from that stream (line 22) and if this data word is not the last (checked in line 24), then the state machine moves to the M_STREAM state, where it then outputs the remaining data words from that stream. When the last data word is processed, then the FSM reverts to state M_IDLE, where it repeats the previous process.

This module introduces a new pragma directive called *array_partition*. This pragma lets Vivado HLS know if an array is to be split into multiple subarrays in order to improve throughput. If it is not specified, Vivado HLS uses a two-port BRAM to access an array. If the array is to be accessed more than twice in a clock cycle, the tool will not be able to schedule those accesses without raising the II value appropriately. In this example, omitting the array_partition pragma and having a NUN_MERGE_STREAMS value of 8 will result in an II = 4. But since we want to be able to access all elements of the streamEmpty array in each clock cycle to achieve the target II = 1, we need to partition the array fully. In this case, then, the array will be implemented into a flip-flop-based set of registers.

Splitting an incoming stream would be a very similar process in which data words coming from one stream would be routed appropriately to a stream array.

EXTRACTING AND REALIGNING FIELDS

Extracting and realigning fields is one of the most fundamental operations in packet processing. As packets typically arrive in a module through a bus over multiple clock cycles, it is a common occurrence that fields of interest either are misaligned in the data word in which they arrive or else spawn multiple data words (or more often, both).

Thus, in order to process these fields, you have to extricate them from the data stream, buffer them and realign them for processing.

Example 5 – Source MAC address extraction example

```
1  if (!inData.empty()) {
2      inData.read(currWord);
3      switch(wordCount) {
4          case 0:
5              MAC_DST = currWord.data.range(47, 0);
6              MAC_SRC.range(15, 0) = currWord.data.
                   range(63, 48);
7              break;
8          case 1:
9              MAC_SRC.range(47 ,16) = currWord.
                   data.range(31, 0);
10             break;
11         case 2:
12……
```

Example 5 illustrates a very simple field-extraction and re-alignment case, in which the source MAC address is extracted from an Ethernet header. The data arrives over a 64-bit stream called inData. In each clock the data is read in (line 2). Then, depending on the data word that's read, the appropriate statement is executed. Thus in line 5, the first 16 bits of the source MAC address are extracted and shifted to the beginning of the MAC_SRC variable. In the next clock cycle, the remaining 32 bits of the MAC address arrive on the bus and are placed in the 32 higher bits of the MAC_SRC variable.

CREATING SYSTEMS WITH MULTIPLE LEVELS OF HIERARCHY

We've just taken a look at how to implement a simple three-stage pipeline using Vivado HLS. Typical pack-et-processing systems might, however, encompass a multitude of modules distributed into several layers of hierarchy. Figure 2 shows an example of such a system. In this case, the first level of hierarchy consists of two modules, one of which then includes three submodules of its own. The top-level module in this case will look like the one described in the simple system assembly introduction. The lower-level module containing the three submodules will, however, use the INLINE pragma to dissolve this function and raise its submodules to the top level, as shown in Example 6.

Example 6 – Intermediate module in Vivado HLS

```
1  void module2(stream<axiWord> &inData,
   stream<axiWord> &outData) {
2  #pragma HLS INLINE
3
4  ………
```

Thus, after Vivado HLS synthesis the system essentially looks like Figure 3. As a result, Vivado HLS is able to correctly create a dataflow architecture out of the modules, pipelining all of them and executing them concurrently. Module and signal names are maintained as they were after the inlining of the function.

USING HIGH-LEVEL LANGUAGE CONSTRUCTS

One of the major advantages of high-level synthesis is that it allows the use of high-level language constructs to express complex objects, thus raising the level of abstraction considerably over traditional RTL design. One example is the description of a small lookup table.
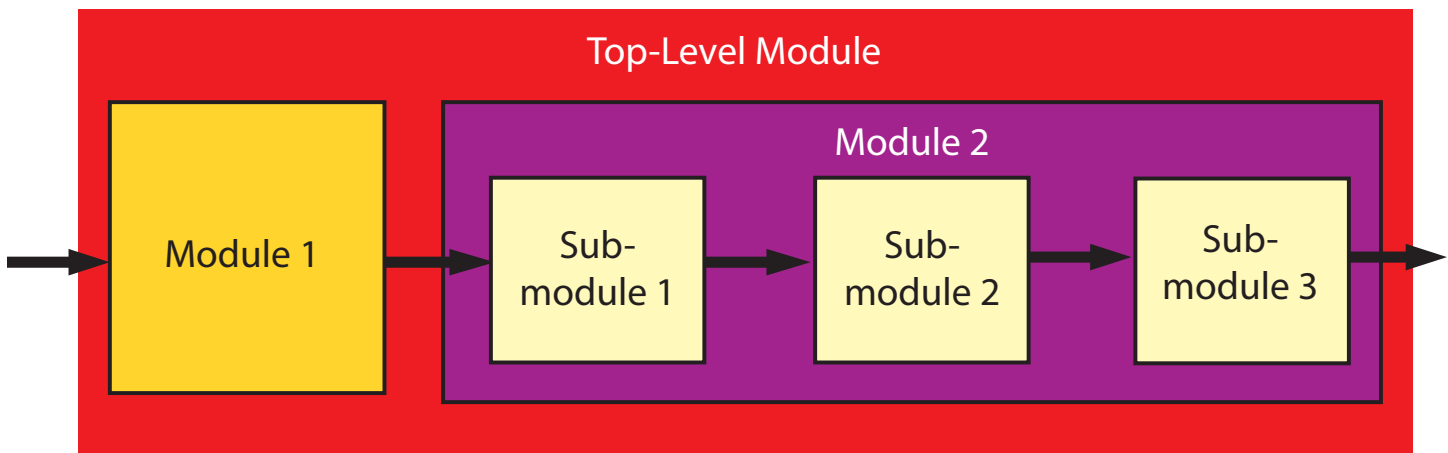
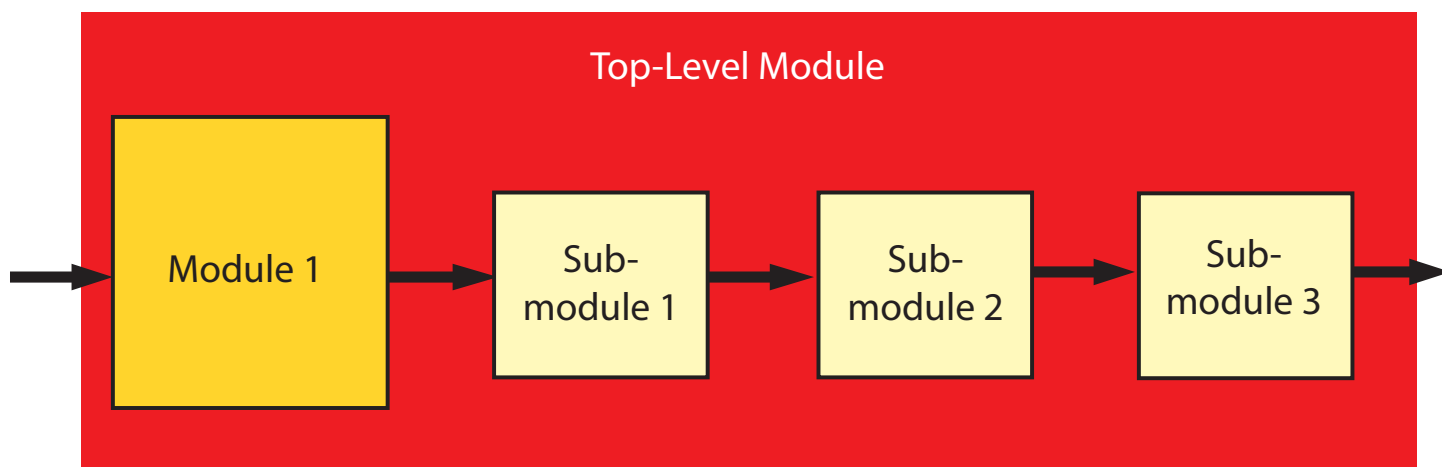Figure 2 – Example design with two levels of hierarchy

Figure 3 – Intermediate hierarchy level dissolved in a pipelined Vivado HLS design

The code shown in Example 7, for a content-addressable memory (CAM) class declaration, uses a class object to create a table, which is used to store and retrieve our prototype system's ARP data. The class has one private member, which is an array of *noOfArpTableEntries* number of entries of *arpTableEntry* type. This type is a struct, which consists of the MAC address, the corresponding IP address and a bit that indicates whether this entry contains valid data or not.

Example 7 – CAM class declaration

```
1 class cam {
2 private:
3 arpTableEntry filterEntries[noOfArpTableEn-
  tries];
4 public:
5    cam();
6    bool write(arpTableEntry writeEntry);
7    bool clear(ap_uint<32> clearAddress);
8    arpTableEntry compare(ap_uint<32>
     searchAddress);
9 };
```

The class also includes four methods (with one of them being the constructor) that operate on this table. One of them, the compare method, implements the actual lookup functionality. In this case an IP address is provided, for which the corresponding MAC address has to be returned. This is done by going through all the entries in the table with a for loop and searching for a valid entry with the same IP address. This entry is then returned in its entirety. An invalid entry is returned if nothing is found. In order for the design to reach the target II = 1, the loop has to be unrolled completely.

Example 8 – Compare method for the CAM class

```
1 arpTableEntry cam::compare(ap_uint<32> searchAd-
  dress) {
2   for (uint8_t i=0;i<noOfArpTableEntries;++i) {
3     if (this->filterEntries[i].valid == 1 &&
      searchAddress == this->filterEntries[i].ipAd-
      dress)
4        return this->filterEntries[i];
5   }
6   arpTableEntry temp = {0, 0, 0};
7   return temp;
8 }
```

Our experience and example design make it clear that you can use Vivado HLS to leverage high-level programming constructs and describe packet-processing systems in a software-like manner. That's not easy to accomplish in RTL.

PROTOCOL PROCESSING AT 10 GBPS

Vivado HLS allows for the quick and easy implementation of protocol-processing designs on FPGAs by using C/C++ and leveraging the productivity increases offered by higher-level languages in comparison to traditional RTL. Additional advantages include an effortless system buildup using C functions; data exchange over streams that offer standardized FIFO-like interfaces; and free flow control and HLS pragmas to direct the tool toward the desired architecture. You can use all of these functions to quickly explore different design trade-offs without rewriting the source code.

As a vehicle for explaining the basic concepts of such designs, we've discussed a simple ARP server that replies to ping and ARP request and resolves IP address queries. The result proves that Vivado HLS-designed modules can perform protocol processing at line rates of 10 Gbps and higher.

In FPGA Design, Timing Is Everything

Tools and techniques exist to help you efficiently achieve your timing performance goals.
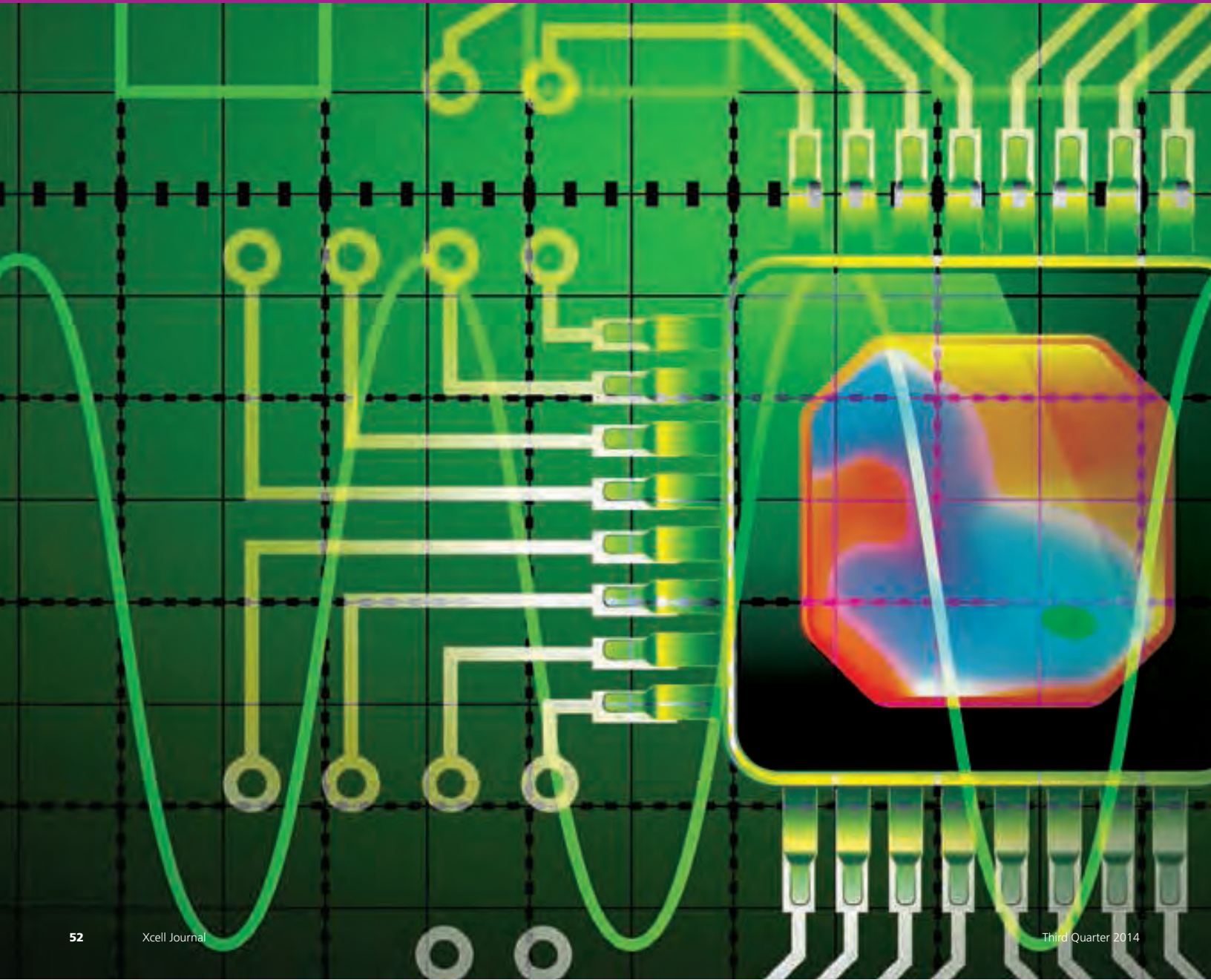
by Angela Sutton
Staff Product Marketing Manager, FPGA
Synopsys
sutton@synopsys.com

Paul Owens
Corporate Applications Engineer, FPGA
Synopsys
powens@synopsys.com

W

When your FPGA design fails to meet timing performance objectives, the cause may not be obvious. The solution lies not only in the FPGA implementation tools' talent in optimizing the design to meet timing, but also in the designer's ability to specify goals upfront and diagnose and isolate timing problems downstream. Designers now have access to certain tips and tricks that will help you set up clocks; correctly set timing constraints using tools like Synopsys Synplify Premier; and then tune parameters to meet the performance goals of your Xilinx® FPGA design.

There are multiple angles of attack, including:

- Better design setup, such as complete and accurate timing constraints and clock specifications;

- Time-saving design techniques such as careful RTL coding for better performance results and grouping together the parts of the design that pose the greatest performance challenge, to reduce iteration run-times when you later tune the design;

- Correlation of synthesis and place-and-route timing to deliver better timing quality of results (QoR) and timing closure.

Let's take a closer look at some of these techniques in all three categories, and examine how to use them to achieve your timing goals.

STEP 1: BETTER DESIGN SETUP

The biggest bang for your buck will come from specifying correct and complete design constraints. The constraints communicate your design intent and the design's performance goals to the synthesis tool. Once the design has been synthesized, these constraints and the critical-path information

You will want to check that you have adequately and completely constrained your design without having overconstrained it.

will automatically be forward-annotated to the Vivado® Design Suite place-and-route (P&R) tools to further ensure that timing will be met.

The synthesis tool can assist you with the challenging task of setting up presynthesis constraints. On your to-do list will be to:

1. Identify clocks

2. Identify and create clock groupings and clock relationships

3. Constrain clocks

4. Constrain design inputs and outputs

5. Define multicycle paths and false paths

You will want to check that you have adequately and completely constrained your design without having overconstrained it. Overconstraining will result in longer run-times and potentially, the reporting of false critical paths. Be sure to specify multicycle and false paths, and to set constraints on derived clocks (define_path_delay, define_false_path).

Setting up an initial constraints file for Vivado flows

Since constraints setup can be daunting, the synthesis software can help by providing an initial constraints template with basic constraints and syntax that can act as a starting point. For example, in the Synplify synthesis software, run the TCL utility to create an initial FDC file for a specific design:

TCL: create_fdc_template

Figure 1 shows an example of the constraints (.fdc) file that this process will generate. In this example, you can see that key items, such as declaring clocks, clock group (relationship between clocks) and input/output delays, have been taken care of.

Best practices for constraints setup in Vivado Design Suite flows

When setting up constraints in Vivado Design Suite flows, be sure to do the following:

• Define all primary clocks on input ports or nets connected to input ports.

• Define clocks on the black-box output pins.

• Define generated clocks on nets.

• Don't define gated clocks.

• Provide correct clock constraints: Don't overconstrain, and be sure to place unrelated (aka asynchronous) clocks in separate clock groups.

• Define timing exceptions such as false paths and multicycle paths.

Hint: In Vivado Design Suite, clock constraints should be applied as close to the source clock as possible, not on the BUFG, as was the case in Xilinx ISE® Design Suite flows.

Ensuring that your constraints are correct

We recommend four constraints verification techniques during the design setup phase. To give you an idea of the types of constraints checks that are worthwhile, let's look at the checks that Synplify software performs.

```
###==== BEGIN Clocks — {Populated from tab in SCOPE, do not edit)
create_clock —name {clock} [get_ports {p:clock}] —period 10 —waveform {0 5.0}
###==== END Clocks - {Populated from tab in SCOPE, do not edit)
###==== BEGIN Inputs/Outputs - {Populated from tab in SCOPE, do not edit)
set_input_delay {p:porta[7:0]} 1 —clock {c:clock} —add_delay
set_input_delay {p:portb[7:0]} 1 —clock {c:clock} —add_delay
…
set_output_delay {p:porto[7:0]} .5 —clock {c:clock} —add_delay
…
###==== END Inputs/Outputs - {Populated from tab in SCOPE, do not edit)
###==== BEGIN Registers - {Populated from tab in SCOPE, do not edit)
…
set_clock_groups —disable —asynchronous —name {clock_group} —group {clock} —comment
  {Source clock clock group}
```

Figure 1 – An initial Synplify synthesis input constraints file takes care of basic clock setup and I/O constraints requirements. The constraints will be forward-annotated to the Vivado place-and-route tool.
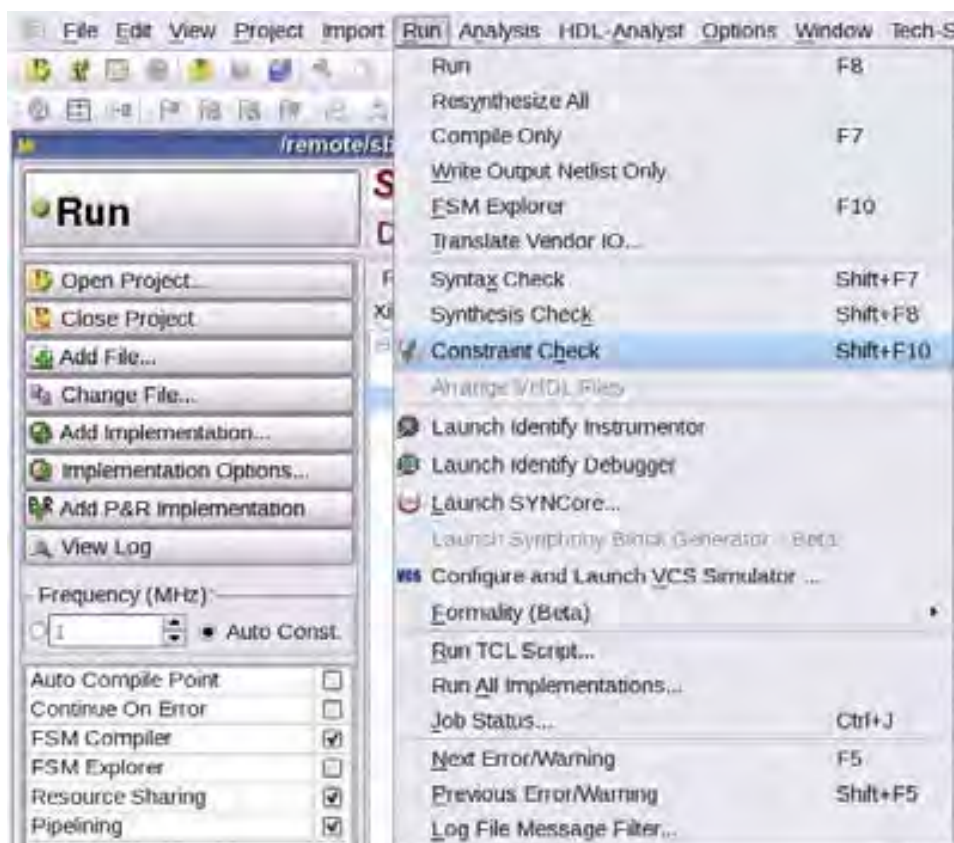
Figure 2 – Running a syntax, synthesis and then constraints check is a way to find constraints and clock-setup pilot errors, helping you achieve timing QoR quickly.

cable constraints and the instances not found so that the constraints file can be corrected. Synplify synthesis tools will run these kinds of checks automatically during the synthesis premap stage, or you can run the constraints check at the beginning of synthesis using the following TCL command:

TCL: project -run constraint_check

Running these basic types of checks identifies possible errors early in the synthesis cycle and improves the quality of results (Figure 2).

Once synthesis has been run, be sure to analyze the post-synthesis timing report, as it can provide important information. For example, when using Synplify software, a "System Clock" under the starting-clock section of the timing report indicates that some of your I/Os may not be constrained. The interface information in this report will confirm whether or not this is the case.

STEP 2: RTL CODING STYLES AND CRITICAL-PATH TUNING

To converge on better timing, we recommend that you use certain coding styles for finite state machines, RAMS, math/DSP functions, clock trees and shift registers. The result will be improved timing QoR, because the synthesis tool is better able to infer an implementation using FPGA primitive building blocks.

Additionally, these coding styles keep you from creating unnecessary logic such as inferred latches, read/write check logic for RAMS and logic that could have been packed into a DSP primitive. While much has been written on this topic, using core-generator capabilities within your synthesis tool is a key point to consider. For example, Synplify software includes a SynCore IP wizard that autogenerates the required RTL coding style for byte-enabled RAMs. Other IP generators, such as the Xilinx IP Catalog, Synopsys' Synphony Model Compiler or Synopsys' DesignWare coreTools and DesignWare Building Blocks,

First run a "syntax check"—that is, a quick check of constraints, including their embedded "get_XX" and "all_XX" commands, to uncover and clean up any constraint syntax errors. The errors will appear in a log file that can be hyperlinked back to the error manual that explains the error and suggests a fix. Use the Tcl command *check_fdc_query*.

Second, run a "synthesis check" to detect hardware-related errors such as incorrectly coded flip-flops. These errors are reported in a separate log file.

Third, run basic "fast synthesis" that checks for clock-setup issues, including declared, derived and inferred clocks. Fast synthesis allows you to perform a clock-setup check, because it generates both a clock report and a timing report that make clock-setup issues apparent.

Certain synthesis tools enable you to run synthesis in a "fast" mode, which disables some synthesis opti-

mizations in the interest of rapid runtime. In the Synplify Premier synthesis software, you can do this using the following command:

set_option –fast_synthesis 1

The synthesis compiler will create a synthesis report clock summary with information on inferred clocks that you can use for identifying, defining and constraining clocks.

Fourth, run a full "constraints check." This check looks for constraint setup issues with clock relationships, unconstrained start/end points, nonlocked I/Os and I/Os with no constraints.

A full constraints check will also look for the correct application of constraints and instance names. For example, it will flag timing constraints applied to nonexistent or invalid types of arguments and objects. The tool then generates a detailed explanatory report of inappli-
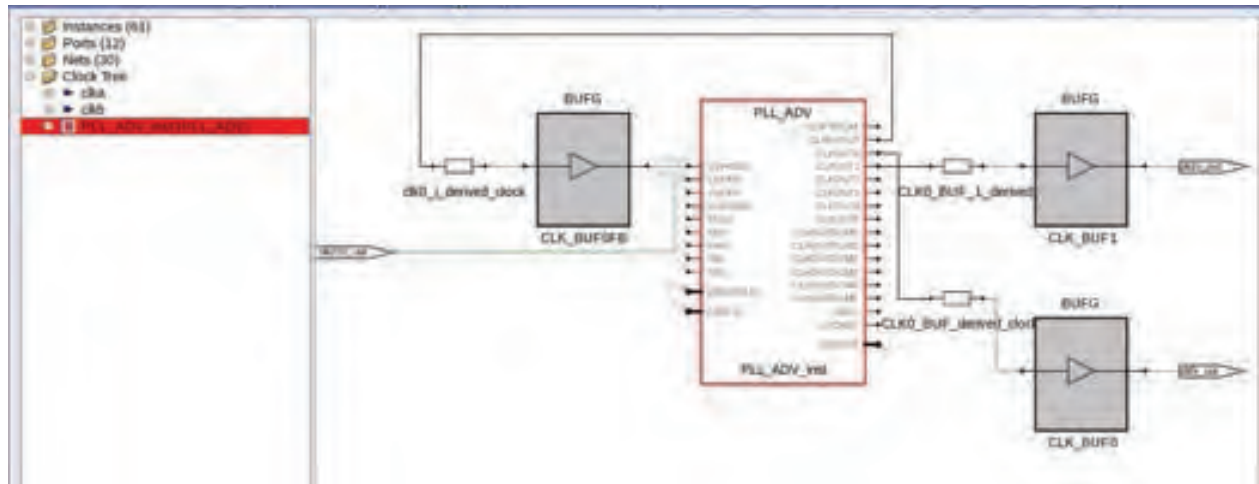
Figure 3 — Debug clock trees and clock constraints using a schematic viewer that pinpoints the clock trees.

can also help you configure the IP, perform many DSP and mathematical functions, and create good RTL coding styles. If hand-coding, keep in mind the following:

For finite state machines
- For Xilinx flows, use a synchronous reset to set hardware to a valid state after power-up, or reset hardware during operation.

- Separate the sequential blocks from combinational-always blocks.

- Assign a next-state variable for all possible (present) states.

For Block RAMs
- Code synchronous RAMs wherever possible since these will generally run at higher clock frequencies.

- Place RAM code in a separate module for easier debug at the netlist level.

- Before deciding to use a RAM with a specific reset condition, dual-port or byte-enabled RAMs, or asymmetric RAMS, look at the recommended coding style and check whether inference is supported. If not, the netlist may end up creating more control logic.

- Don't read from the same address that you write to in the same clock cycle.

- When all else fails, use an attribute (syn_ramstyle for Synplify software) to force implementation of small

RAMs into registers, freeing up RAM resources for use in more timing-critical or larger RAMs.

For DSP blocks
You can use these primitives to implement filters and mathematical functions such as counters, adders, multipliers, subtractors and the like.

- Place DSP code in a separate module for easier debug at the netlist level.

- When all else fails, use an attribute (syn_dspstyle for Synplify in Vivado Design Suite flows) to force implementation.

For SRLs
You can pack shift registers into a select_SRL Xilinx SRL primitive or implement them in registers.

- Packing occurs automatically. For shift register chains, Synplify software will always leave the last register in the chain out of the packed select_srl in order to optimize timing QoR.

- When all else fails, use the syn_srlstyle attribute to control how your SRLs are implemented.

For clock trees
- Since these will not automatically be inferred during synthesis, it is recommended that you instantiate phase-locked loops (PLLs), clock generators or clock muxes in your RTL.

- The timing constraints on the input clock of the PLL will automatically generate derived clock constraints on the PLL output pins.

To better review clock constraints, examine them in a schematic viewer. For example, Synplify's HDL Analyst tool can run a filter on the clock tree that lets you observe and debug clock trees and clock constraints (Figure 3).

Using modularity to improve critical-path performance
Certain parts of the design may be more timing-critical than others, and you will need to tune and incrementally improve those that are not performing. One of the techniques that you can apply to allow quicker tuning at the RTL and netlist phases involves isolating critical paths inside a single block or subproject that you can then iterate on incrementally to improve it. In addition, you can force the Vivado place-and-route tools to place the elements in close proximity to further ensure timing QoR. Capabilities at your disposal during synthesis that allow for this modularity include:

- Prior to synthesis, specify an RTL partition (in Synplify software, this would be called a "compile point") or create a hierarchical subproject.

- Post synthesis, isolate (export) as a subproject just that portion of the de-
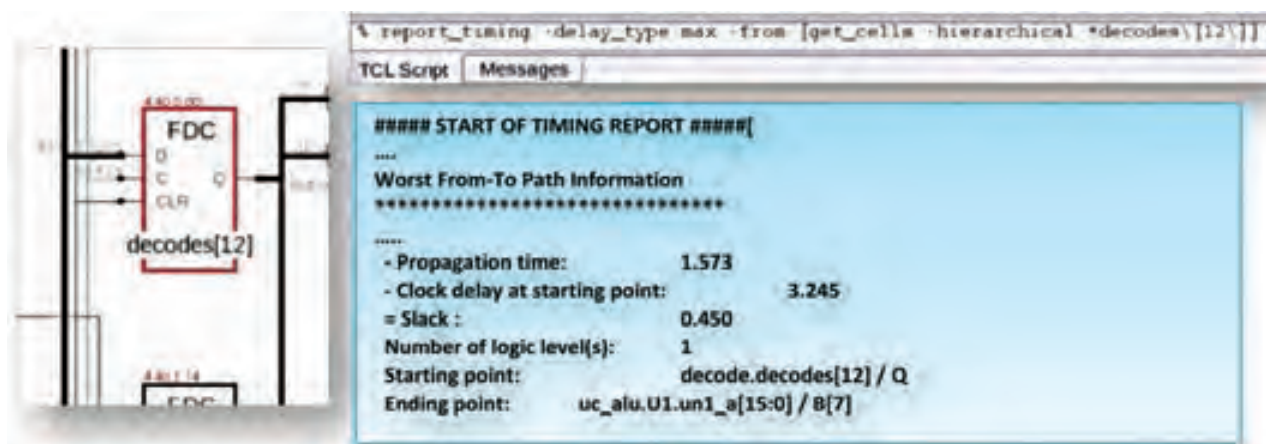
Figure 4 – You can generate timing reports to debug specific nodes in the design.

sign that is not performing using a hierarchical program-management flow. Iterate, fix and merge back the results.

• Have the synthesis software communicate to the Vivado place-and-route tool that it is to place the critical path on the same die of a multi-SLR device such as the Virtex®-7 2000T FPGA to avoid cross-SLR delays.

STEP 3: GAINING FINAL TIMING CLOSURE

General timing can be reported post-synthesis and after placement and routing (Figure 4). For example, Synplify software allows you to report upon specific parts of the design of interest using a TCL command *(report_timing)*.

To improve timing QoR further, we recommend that you correlate post-synthesis and post-P&R timing results, specifically the slack margins for given start points and endpoints on timing-critical paths. In Synplify Premier

synthesis software, for example, you can display the post-synthesis and P&R timing reports side-by-side to read the timing results.

The correlation tool does a side-by-side comparison of the status of endpoints, start points and required periods. Paths are reported against the end clock. Paths for which pre- and post-P&R timing do not correlate well to within your specified "slack margin" criteria will be flagged as "correlation mismatches" so that you can take action on them. A typical action would be to specify so-called "-route" constraints to the synthesis tool that tighten timing-path constraints only during the synthesis phase. Here's an example:

FDC Constraints input file to synthesis: *set_clock_route_delay {c:clka} 1.4*

These constraints make synthesis "work harder" to meet timing performance on those paths, resulting in better correlation and QoR.

The nice thing about the timing-correlation capability is that you can drill down to view the exact paths that are causing problems, for example, changing the number of paths that you want to be displayed per endpoint. You can search for specific clocks or instances of interest and have their timing paths displayed. Clocks are also compared and displayed to further assist in timing correlation (see Figure 5).

As you can see, there are certain steps you need to take in order to achieve better timing performance in a reasonable amount of time in your Vivado Design Suite flow. The methodology we have outlined will intercept clock and constraints setup issues early, while also offering a variety of techniques to tune and correlate timing in your design and its RTL to get fast timing closure.

For more information and examples, please visit *http://www.synopsys.com/fpga*.
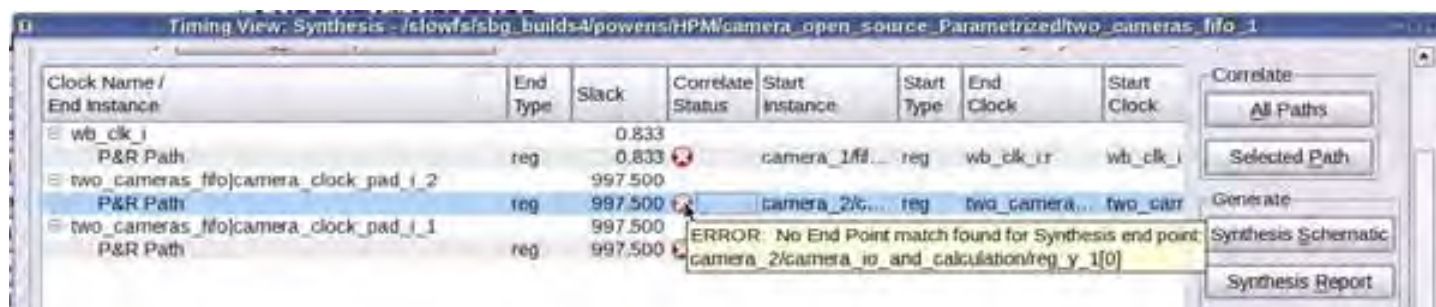

Figure 5 – A timing correlation reports allow you to compare timing mismatches post-synthesis and post-P&R, displaying and filtering path timing results and clocks side-by-side.

NI System-on-Module Brings Innovative Products to Market Fast

by Eric Myers
Product Manager
National Instruments
eric.myers@ni.com

Based on the Zynq SoC,
the NI SOM is extensively
tested and validated, and ships
with a complete software stack.

E

Embedded design teams are tasked with many demanding challenges. They are expected to maintain a competitive advantage by staying up to date with the latest technology and delivering many new features, and they are expected to do this faster and with fewer resources with each new project. This places a lot of pressure on these teams to develop quickly with traditional methods.

However, traditional design methods make rapid innovation difficult. In fact, these methods often lead to missed deadlines. According the 2013 Embedded Market Survey by UBM, 57 percent of embedded projects are either behind schedule or canceled.

It is possible for these teams to add resources to help manage high-investment tasks, such as software. However, managers are also charged with maintaining and improving the profitability of projects. Because of the need to balance innovation with profitability, many design teams have been transitioning to a new way of developing embedded systems.

Rather than developing a product from the ground up, more teams are beginning to use off-the-shelf components that will help accelerate their design process. The most notable of these components is the system-on-module (SOM). According to the 2012 edition of the World Market for Embedded Computer Boards and Modules by IHS, SOMs are expected to experience a 17.5 percent compound annual growth rate from 2010 to 2016, with the next closest category being standalone boards at 9.3 percent.

Figure 1 – The NI sbRIO-9651 system-on-module, built around Xilinx's Zynq SoC, provides a customizable form factor and a complete software stack.

A SOM provides many of the elements required for an embedded design, such as processing elements like the Xilinx® Zynq®-7000 All Programmable SoC, along with common components such as memory. Furthermore, some systems-on-module ship with a complete software stack, eliminating the need for costly driver, middleware and OS development. Embedded designers can then take advantage of the flexibility of the SOM and customize the system for a particular application, adding specific I/O, peripherals and packaging. Maintaining flexibility with an off-the-shelf hardware design gives design teams a head start on their application and reduces overall development time and risk.

The recently announced NI sbRIO-9651 system-on-module provides the customizability of the SOM form factor (Figure 1). The product offers completely validated software components that will save additional design time and reduce risk even further, while also providing a simple, alternative approach to hardware description languages (HDLs) for programming the FPGA.

A CLOSER LOOK AT THE HARDWARE

The NI SOM combines the Xilinx Zynq-7020 SoC with supporting components such as RAM and storage on a small PCB about the size of a business card (Figure 2). The Zynq SoC is equipped with a 667-MHz dual-core ARM® Cortex™-A9 processor along with Artix®-7 FPGA fabric. Together, these ingredients provide the base components needed to solve many of today's embedded challenges in an off-the-shelf product.

In the early stages of product development, embedded teams are tasked with selecting and integrating all of these components. While these specifications are important for the design, they offer very little differentiation for the end product. Instead, these tasks typically add many risks, such as multiple board spins. Embedded teams can save design time as well as reduce their project risk by using an off-the-shelf hardware product extensively tested for reliability.

Requirements for reliability come in many shapes and sizes and are unique for each application (see "De-

Specifications
Processor SoC
Xilinx Zynq-7020 667-MHz Dual-Core ARM Cortex-A9 Artix-7 FPGA Fabric
Size and Power
50.8 mm x 78.2 mm (2 in. x 3 in.) Typical Power: 3 W to 5 W
Dedicated Processor I/O
Gigabit Ethernet, USB 2.0 Host, USB 2.0 Host/Device, SDHC, RS232
Memory
Nonvolatile: 512 MB DRAM: 512 MB
Operating Temperature
-40 ˚C to 85 ˚C Local Ambient
FPGA I/O
160 FPGA I/O Pins Configurable Peripherals: Gigabit Ethernet, RS232 x3, RS485 x2, CAN x2

Figure 2 – The NI SOM combines the Zynq SoC with supporting components in a module roughly the size of a business card.

sign Reliability: MTBF Is Just the Beginning," in this issue). Reliability could mean anything from the uptime of a long-term deployment to the ability to operate in a specific environment. NI has a long history of putting an intense focus on verification and validation to provide quality products. As a result, our embedded controller platform, CompactRIO, is deployed in critical applications such as medical devices, harsh environments such as oil and gas fields, and long-term deployments such as smart-grid applications. The NI SOM continues this trend with validation work such as simulation and testing for electrical and shock and vibration, along with thermal testing for mechanical.

The development kit for the new product includes a reference carrier board with multiple peripherals (Gigabit Ethernet x2, USB Host, USB Device, SD, RS-232 x2, RS-485, CAN) and provides the design files to integrate them into a custom carrier-board design. The reference carrier board also provides a digital prototyping area to communicate with specific chip sets as well as four PMOD connectors to help accelerate I/O selection and integration. Many chip vendors offer PMOD modules that range from simple analog I/O to stereo power amplifiers.

VALIDATED, COMPLETE MIDDLEWARE SOLUTION

The NI SOM also integrates a validated board support package (BSP) and device drivers with NI Linux Real-Time (Figure 3). These software components provide out-of-the-box support for peripherals such as Ethernet or USB, interfaces to components like memory and the communication interface between the processor and FPGA. NI Linux Real-Time combines the performance of a real-time OS with the approachability and openness of Linux. Software developers can take advantage of the large Linux community to augment a real-time application while maintaining deterministic operation. NI Linux Real-Time also allows greater flexibility in programming the processor by providing a path for C/C++ and LabVIEW Real-Time applications to communicate with the programmable FPGA.

According to the 2013 Embedded Market Survey by UBM, software development takes up more than 60 percent of resources in an embedded project. The developers often must provide components like the middleware, firmware, embedded OS and application software that require a large investment for developing, testing and debugging. In much the same way as the hardware, the software for the NI SOM goes through an extensive verification-and-validation process, such as stress-testing for all peripherals. By providing a validated BSP, device drivers and real-time OS, the NI SOM helps design teams minimize their development time and risk. Instead, design teams can focus on key features, such as integrating specific I/O or developing custom algorithms and application software.

FPGA APPLICATION SOFTWARE

With reconfigurable FPGA technology, it is possible to perform high-speed signal processing, high-speed or deterministic control, inline signal processing and custom timing and triggering. For control systems, you can also run advanced control algorithms directly in the Zynq SoC's programmable logic to minimize latency and maximize loop rates.

NI LabVIEW system-design software provides a graphical development environment with thousands of functions and IP blocks for both processors and FPGAs. LabVIEW FPGA, which extends the LabVIEW graphical development platform, provides an alternative to HDL programming that simplifies the task of interfacing to I/O and communicating data, greatly improving embedded system design productivity and reducing the time-to-market.

LabVIEW FPGA provides IP developed by NI and Xilinx for basic functions such as counters or more advanced algorithms such as video decoding and complex motion control. Experienced HDL developers can import and reuse existing code with an IP Integration Node. The software also integrates a DMA engine for data transfer between the processor and FPGA fabric.

ACCELERATE THE PROTOTYPE PHASE WITH COMPACTRIO

Building a prototype based on a custom design often requires months of initial development work to integrate components and I/O before being able to validate the application software. While a standard off-the-shelf product may allow for the development of a proof of concept more quickly, teams then have to start over from scratch because they cannot reuse any of the code for their final deployment. With a CompactRIO controller and the NI SOM, design teams can quickly prototype and reuse the majority of their code for the final deployment.

CompactRIO controllers and the NI SOM are both based on the LabVIEW RIO architecture, which includes three components that can all be programmed with a single software tool chain: an embedded

controller for communication and processing; an FPGA for advanced control, digital communication protocols, timing, signal processing and filtering; and I/O for connectivity to any sensor or device. In addition to these components, CompactRIO controllers provide a mechanical package and offer more than 100 C Series I/O modules. As a result, teams can begin developing their application software immediately without having to develop custom hardware, which significantly reduces the time needed to prototype.

After you have created the prototype, the NI SOM can reuse the majority of the code, allowing design teams to focus on integrating the I/O from their custom carrier board rather than starting application software development again.

REDUCING DEVELOPMENT TIME AND RISK

Combining a fully tested and validated hardware design with a complete middleware solution and embedded OS, the NI SOM saves design teams significant development time and allows them to get innovative products to market faster. The NI SOM entirely manages many common tasks that are required for every embedded project, such as the BSP to support peripherals, connections to memory and RAM, and a communication line between the processor and FPGA. Likewise, the NI SOM simplifies many of the other common design tasks. As an example, it provides an integrated heat spreader that simplifies the mechanical design of a thermal solution and provides a single touch point for thermal validation.

With the NI SOM, design teams can have the confidence they will be able to deliver embedded projects on time while maintaining or improving profitability. Optimedica, a medical-device company based in California, is developing its next-generation precision laser system for cataract surgery using the NI SOM. "The NI SOM will greatly improve the profitability of our project," said Mike Wiltberger, founder of Optimedica. "It will save us six months of development effort over alternatives, and I can't even build one of these for the price I was quoted."

To learn more, visit *ni.com/som*.

LabVIEW and Middleware
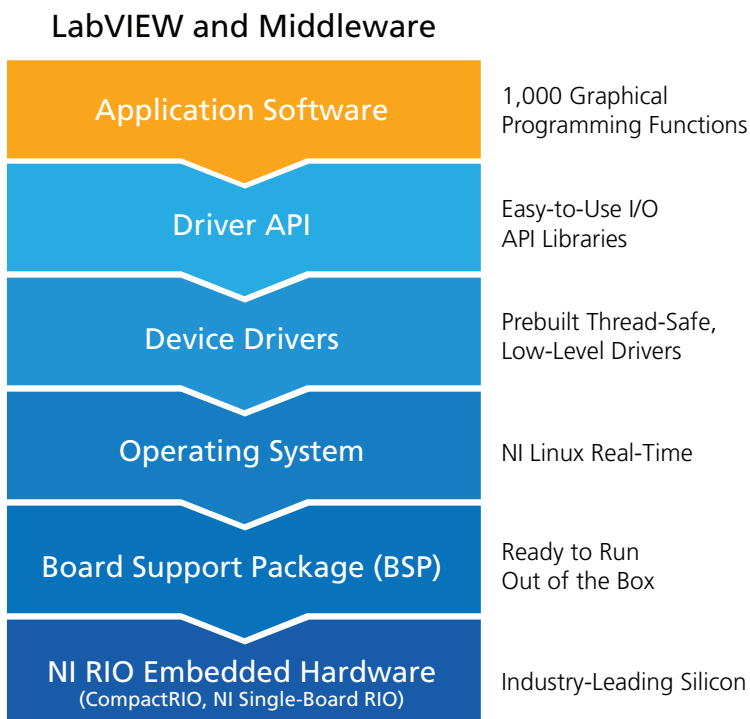

Application Software	1,000 Graphical Programming Functions
Driver API	Easy-to-Use I/O API Libraries
Device Drivers	Prebuilt Thread-Safe, Low-Level Drivers
Operating System	NI Linux Real-Time
Board Support Package (BSP)	Ready to Run Out of the Box
NI RIO Embedded Hardware (CompactRIO, NI Single-Board RIO)	Industry-Leading Silicon

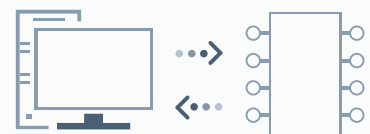Figure 3 – The NI SOM integrates a validated board support package (BSP) and device drivers with NI Linux Real-Time.

What's New in the Vivado 2014.2 Release?

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

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

The Vivado Design Suite 2014.2 is available from the Xilinx Download Center at www.xilinx.com/download.

VIVADO DESIGN SUITE 2014.2 RELEASE HIGHLIGHTS

The Vivado Design Suite 2014.2 features expanded support for Virtex® UltraScale™ devices and new optimizations to improve performance.

DEVICE SUPPORT

Production Ready

- Defense-Grade Artix®-7Q: XQ7A50T
- Defense-Grade Zynq®-7000 SoC: XQ7Z045 and RF900 Package
- XA Zynq®-7000 SoC: XA7Z030 and FBG484 Package

General Access

- Virtex UltraScale: XCVU065, XCVU080, XCVU095 and XCVU125

Early Access

(Contact your local Xilinx sales representative)

- Kintex® UltraScale SSI devices: XCKU100 and XCKU115
- Virtex UltraScale devices: XCVU160 and XCVU440

VIVADO DESIGN SUITE: DESIGN EDITION UPDATES

Partial Reconfiguration

Device support for partial reconfiguration has been extended to include the two smallest Artix-7 devices: the 7A50T and 7A35T.

Tandem Configuration for Xilinx PCIe IP

Support for the Zynq SoC 7Z100 device has been added. For more information, see the PCI Express® IP Product Guides PG054 (for Gen2 PCIe® IP) or PG023 (for Gen3 PCIe IP).

Vivado IP Flows and Vivado IP Integrator

A number of DRCs that were run during generation have been moved into the `validate_bd_design` step to catch these issues earlier in the flow.

Vivado Physical Implementation Tools

To improve run-time, the default behavior of the place-and-route timing summary has changed. In version 2013.4, both the placer and router reported a timing summary in the log with WNS based on signoff timing from the static timing engine. Beginning with the 2014.1 release, the placer and router no longer report signoff timing by default.

VIVADO DESIGN SUITE: SYSTEM EDITION UPDATES

System Generator for DSP

This tool now offers improved simulation performance. Waveform Viewer conversion times have been shortened

up to 90 percent, while System Generator block bring-up offers a 50 percent improvement. In addition, Xilinx has decreased simulation initialization times by up to 80 percent for models with multiple FFTs and other complex IP.

A new fast MCode model improves performance of MultAdd by over 90 percent. Xilinx has also updated WinPCap to 4.1.3 for Ethernet hardware co-simulation support in Windows 8.1. Finally, the tool has updated blocksets.

See the Vivado Design Suite 2014.2 Release Notes for more information.

ULTRAFAST DESIGN METHODOLOGY

With the UltraFast™ offering, Xilinx delivers the first comprehensive design methodology in the programmable industry. Xilinx has hand-picked the best practices from experts and distilled them into this authoritative set of methodology guidelines for the Vivado Design Suite.

Now in its second edition, the "UltraFast Design Methodology Guide" extends support to the UltraScale architecture, adds a new Timing Constraint Wizard for rapid timing closure and includes new best practices, such as:

- Design methodology DRCs
- Revision control
- IP and IP Integrator methodology
- Simulation (including third-party flows)
- Verification
- Vivado HLS (high-level synthesis)
- Partial reconfiguration

XILINX TCL STORE

Xilinx is taking another large step forward in designer productivity by hosting an open-source repository for sharing Tool Command Language (Tcl) code. This repository, called the Xilinx Tcl Store, will make it a lot easier to find and share Tcl scripts that other engineers have developed. With the power of Tcl, these scripts can extend the considerable core functionality of the Vivado Design Suite, enhancing productivity and ease of use. The Tcl Store is open to the user community to contribute to the greater good of all designers by publishing Tcl code that others might find useful.

The Xilinx Tcl Store provides examples of how to write custom reports, control specific tool behavior, make custom netlist changes and integrate with third-party electronic design automation (EDA) tools such as simulation, synthesis, timing and power analysis, and linting tools. Natively accessed from the Vivado integrated design environment (IDE), the Tcl Store enables users to select and install collections of Tcl scripts called "apps" directly from within the tool. Once installed, these apps have commands that appear just like built-in Vivado Design Suite commands.

To learn more about the Xilinx Tcl Store, watch the QuickTake video.

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. New topics include: Design Flow Overview, Using the Timing Constraint Wizard, Xilinx Tcl Store, Using Vivado with Xilinx Evaluation Boards and Packaging Custom IP for use with IP Integrator.

See all Quick Take Videos here.

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 some highlights.

OMNITEK'S SCALABLE VIDEO PROCESSOR IP NOW INCLUDES 4K PROCESSING

The OSVP Suite from Certified Alliance Member OmniTek comprises a set of IP blocks optimized for Xilinx technology that together provide complete multi-video format conversion, image resizing, color correction and compositing for video formats up to 4K, 60-Hz UltraHD. The suite includes v1.0 for Full HD and v2.0, which adds 4K/60 support and more, together with combiner and interlacer blocks. The cores can be configured to support up to eight high-definition channels or two 4K channels, with independent selection of video standards on each channel.

Each input can be deinterlaced, color corrected, sharpened, softened and resized, with both compile-time and runtime configuration control.

The progressive output from the video processor can be composited onto the video output with the Combiner block. This output can comprise one or more video streams, delivering quad-split display. The Interlacer block, for its part, can produce interlaced formats. To aid application development, the OSVP Suite also includes drivers that enable the cores to be driven within the OmniTek FPGA Software Interface Framework, an intelligent architecture that eases the task of prototyping systems and evaluating options

by removing the need to write kernel-mode code.

OmniTek has optimized the blocks in the OSVP Suite for Xilinx 7 series FPGA technology (Kintex® and Virtex® FPGAs and the Zynq®-7000 All Programmable SoC), and implemented in the Vivado® IP Integrator design environment. Together, these IP blocks achieve the highest broadcast-quality video processing in an extremely small footprint. The company also provides reference designs that integrate the OSVP IP with cores of the Xilinx Video and Image Processing Pack in a complete working FPGA design targeted for a range of hardware platforms.

Watch the demo here.

XYLON BASES AUTO DRIVER-ASSISTANCE DEVELOPMENT KIT ON XILINX ZYNQ SOC

The logiADAK automotive driver-assistance kit from Xylon is a Zynq SoC-based development platform for advanced driver-assistance (DA) applications that require intensive real-time video processing, parallel execution of multiple complex algorithms and versatile interfacing with sensors and a vehicle's communication backbones. Users can leverage the logiADAK kit to quickly bring new DA innovations to market. It provides automotive driver-assistance system (ADAS) designers with all the resources they need to efficiently develop vision-based DA systems. ADAS designers will save months of development time and can focus their efforts on system-differentiating functions and performance.

The kit comes with a full set of DA demo applications, customizable reference SoC designs with evaluation IP cores, software drivers and libraries, calibration software and documentation. The included DA demo applications use multiple-configuration Zynq SoC designs that reprogram (reuse) the device's programmable logic (PL) to support different feature bundles suitable for particular driving conditions. This ultimate reprogrammability, which occurs under the continued supervision of the ARM® Cortex™-A9 processors in the processing system (PS), saves silicon resources and enables use of a smaller and more cost-efficient Zynq SoC.

The unique blend of hard-coded and programmable logic on a single Zynq SoC enables ADAS designers to create SoCs that outperform competing solutions. The resulting designs achieve a new level of system differentiation through a combination of hardware-accelerated functions implemented in the programmable logic and familiar software-based DA functions running on powerful ARM processors.

The logiADAK hardware platform is appropriate for test vehicle installations and rapid engagements in proof-of-concept or demonstration projects.

Find product details here.

DO-254 ARINC 818 XGA TRANSCEIVER CORE NOW AVAILABLE FROM LOGICIRCUIT

The ARINC 818 transceiver core from Alliance Program Member LogiCircuit provides an easy way to implement ARINC 818-compliant interfaces in Xilinx FPGAs. The core can achieve ARINC 818 interfaces up to 4.25. Great River Technologies offers evaluation kits and development packages that greatly simplify implementing ARINC 818.

Designers can use the LogiCircuit core for transmit-only, receive-only or transmit-and-receive applications. The core has many flexible compile-time settings allowing for various link speeds, line segmentations and line synchronization methods. Moreover, you can configure this core for various resolutions and pixel-packing methods. Ancillary data can use default values that are set at compile time. Alternatively, data can be updated in real time via the register interface.

Key features of the ARINC 818 transceiver core include complete header/ancillary data recovery; embedded ancillary data with real-time update; flexible video resolution/frame rates; low latency; many pixel-packing and input formats; and progressive and interlaced video.

In addition, the core provides receiver error and status detection along with a simple pixel bus transmitter interface. It supports line-synchronous transmission and link speeds up to 4.25 Gbps.

Find product details here.

FIDUS AND INREVIUM PARTNERSHIP EXTENDS CUSTOMER ACCESS TO XILINX-BASED PRODUCTS AND HIGH-SPEED DESIGN SERVICES

Both inrevium and Fidus are Xilinx Alliance Program Premier Members committed to supporting Xilinx technology. Now, a partnership between the inaugural Japanese Premier Member and the inaugural North American Premier Member provides customers worldwide with the combined breadth of system design expertise and deep competency on Xilinx All Programmable technology and devices, thus accelerating their time from concept to production.

Tokyo Electron Device's inrevium is well known for bringing compelling Xilinx All Programmable FPGA, 3D IC and Zynq SoC-based products and services to the Japanese and Asia-Pacific markets. These offerings include leading-edge video development and demonstration platforms, FPGA-based motherboards and FPGA mezzanine cards (FMCs). Fidus Systems is a North American-based electronic product development and consulting services company, focused on providing high-speed, high-complexity solutions to a wide-ranging customer base. Fidus offers customers excellence in hardware, FPGA, signal integrity and embedded software services.

With this partnership, inrevium and Fidus will collaborate on new-product design and product-customization services that will have Fidus tailoring inrevium's off-the-shelf offerings to meet the customer's unique needs. Customization enables clients to get to market faster with less upfront investment. Fidus will also promote, distribute and support inrevium's products in North America.

Find product details here.

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.

XAPP1206: BOOST SOFTWARE PERFORMANCE ON ZYNQ-7000 AP SOC WITH NEON

http://www.xilinx.com/support/documentation/application_notes/xapp1206-boost-sw-performance-zynq7soc-w-neon.pdf

Generally speaking, a CPU executes instructions and processes data one-by-one. Typically, designers achieve high performance by using high clock frequencies, but semiconductor technology imposes limits on this technique. Parallel computation is the next strategy typically employed to improve CPU data-processing capability. The single-instruction, multiple-data (SIMD) technique makes it possible to process multiple data in one or just a few CPU cycles. NEON is the advanced SIMD engine found in the dual-core ARM® Cortex™-A9 processor that's part of the Xilinx® Zynq®-7000 All Programmable SoC. Effective use of NEON—which is specialized for parallel data computation on large data sets—can boost software performance in your design.

In this application note, author Haoliang Qin introduces four methods for improving software performance and cache efficiency using NEON on a Cortex-A9 processor core. They include optimizing assembler code and using NEON intrinsics, NEON-optimized libraries and compiler-optimized automatic vectorization. He also details ways to improve data exchanges among the CPU, cache and main memory.

Software optimization is a complex topic. To realize optimal performance from hardware, you must apply all of these techniques together and properly balance them, Qin says.

XAPP1208: BITSLIP IN LOGIC

http://www.xilinx.com/support/documentation/application_notes/xapp1208-bitslip-logic.pdf

I/O logic in Xilinx UltraScale™ devices refers to the dedicated I/O-handling components located between the I/O buffers and the general interconnect. The I/O logic setup in UltraScale devices provides faster I/O handling, better jitter specifica-

tions and more functionality than in previous device families. However, it omits some functionality available in the I/O logic of the 7 series and Virtex®-6 FPGAs, such as Bitslip.

This application note by Marc Defossez describes a Bitslip solution implemented in general interconnect that you can use in UltraScale devices as well as in previous device architectures. The reference design implements the Bitslip function and extends the basic functionality with several extra options.

This Bitslip reference design performs the same functionality as the native Bitslip functionality embedded in the ISERDES of 7 series and Virtex-6 FPGAs, but goes beyond those implementations by offering extra options not available in solutions based on those two device families. The general interconnect must be used when the functionality available in this design is needed in a 7 series or Virtex-6 FPGA design. Therefore, this Bitslip reference design meets the requirements and goals of Bitslip offered in previous device families.

XAPP1203: IMPLEMENTATION OF SIGNAL PROCESSING IP ON ZYNQ-7000 AP SOC TO POST-PROCESS XADC SAMPLES

http://www.xilinx.com/support/documentation/application_notes/xapp1203-post-proc-ip-zynq-xadc.pdf

This application note is a follow-up and companion to the white paper "Efficient Implementation of Analog Signal Processing Functions in Xilinx All Programmable Devices" (WP442), which proposes a simple and easy design flow for implementing analog signal-processing functions in Xilinx FPGAs and All Programmable SoCs, leveraging Xilinx All Programmable Abstractions. Here, authors Mrinal J. Sarmah and Cathal Murphy describe in detail how to leverage the concepts outlined in the white paper to build signal-processing IP cores and a complete mixed-signal system easily on the Zynq-7000 All Programmable SoC.

The application note demonstrates how to post-process samples from an analog-to-digital converter to filter out environmental noise in a cost-effective way. The design blocks used are lightweight solutions based on DSP blocks complying with standard AXI interfaces. Readers can reuse the IP cores in their own designs as a means to post-process the XADC samples. A design flow based on Vivado® IP Integrator enables ease of reuse in a schematic-based environment in which the designer does not have to work with the underlying RTL.

XAPP1205: DESIGNING HIGH-PERFORMANCE VIDEO SYSTEMS WITH THE ZYNQ-7000 ALL PROGRAMMABLE SOC USING IP INTEGRATOR

http://www.xilinx.com/support/documentation/application_notes/xapp1205-high-performance-video-zynq.pdf

With high-end processing platforms such as the Xilinx Zynq-7000 All Programmable SoC, customers want to take full advantage of the processing system (PS) and custom peripherals available within the device. An example of this philosophy is a system containing multiple video pipelines in which live video streams are written into memory (input) and memory content is sent out to live video streams (output) while the processor is accessing memory. This application note by James Lucero and Bob Slous covers design principles for obtaining high performance from the Zynq SoC memory interfaces, from AXI master interfaces implemented in the programmable logic (PL) and from the ARM Cortex-A9 processors.

With video streams, guaranteed worst-case latency is required to ensure that frames are not dropped or corrupted. To provide high-speed AXI interface masters in the PL with lower latency and direct access to the Zynq-7000 SoC memory interfaces requires connections to the High Performance (HP) interfaces. The Zynq SoC contains four HP interfaces that are 64-bit or 32-bit AXI3 slave interfaces designed for high throughput.

This design uses four AXI Video Direct Memory Access (VDMA) cores to simultaneously move eight streams (four transmit video streams and four receive video streams), each in 1920 x 1080p format, with a 60-Hz refresh rate and up to 24 data bits per pixel. Each AXI Video DMA core is driven from a video test pattern generator (TPG) with a Video Timing Controller (VTC) core to set up the necessary video-timing signals. Data read by each AXI Video DMA core is sent to a common Video On-Screen Display (OSD) core capable of multiplexing or overlaying multiple video streams to a single output video stream. The onboard HDMI video display interface is driven by the output of the Video On-Screen Display core with additional IP cores.

The design uses an AXI Performance Monitor core to capture performance data. All four AXI Video DMA cores are connected to four separate HP interfaces using the AXI interconnect and are under the control of the Cortex-A9 processor. This system uses 70 percent of the memory controller bandwidth. The reference design is targeted for the Zynq SoC ZC702 evaluation board.

XAPP1091: REAL TIME VIDEO ENGINE 2.0 IMPLEMENTATION IN KINTEX-7 FPGAS

http://www.xilinx.com/support/documentation/application_notes/xapp1091-k7-RTV-Engine-2-0.pdf

In the broadcast video landscape, video content with various formats flows across acquisition, contribution, distribution and consumption sectors. To properly archive, distribute and display the content, the video signal often needs to be properly processed with the appropriate format conversion. For example, to correctly display NTSC/PAL signals on a full high-definition (FHD) LCD screen, a series of deinterlacing, scaling, chroma-upsampling and color-correction operations must be performed, as well as alpha blending.

This application note leverages the latest Xilinx Kintex®-7 FPGA architecture to provide a truly scalable video processor reference design that serves multistream/multipipeline video-processing needs. Authors Bob Feng and Kavoos Hedayati target applications like multiviewer display, video switches and multichannel video routers, as well as multistream up- and downconverters.

XAPP1095: REAL TIME VIDEO ENGINE 2.1 IMPLEMENTATION IN XILINX ZYNQ-7000 ALL PROGRAMMABLE SOCS

*http://www.xilinx.com/support/documentation/application_notes/xapp1095-zynq-rtve.pdf*

Another video-oriented application note makes use of the latest Zynq-7000 All Programmable SoC architecture to provide a truly scalable video processor reference design to serve multistream/multipipeline video-processing needs. It also provides graphics-rendering capability to allow differentiated content creation. The design targets applications such as multiviewer displays, video switches and multichannel video routers, as well as multistream up- and downconverters.

Author Bob Feng says that the objective is to provide a highly demonstrable, broadcast-quality video-processing reference design targeted to a wide range of video applications. The Real Time Video Engine Reference Design version 2.1 (RTVE 2.1) provides a graphics-rendering platform using APIs under Linux v3.3 with a Qt graphics environment, and performs scalable video-processing features.

Xpress Yourself
in Our Caption Contest
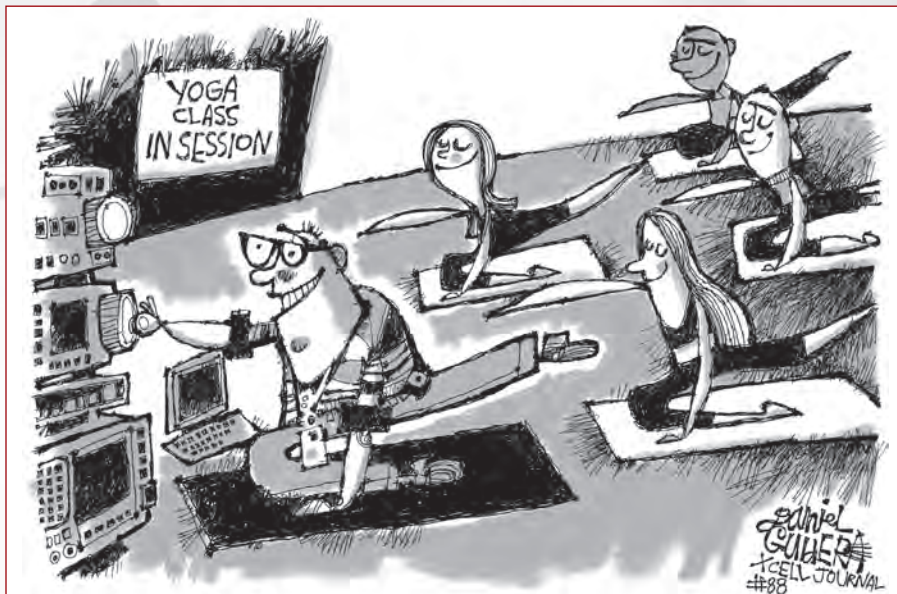
DANIEL GUIDERA

If you've ever had to stretch to meet a challenge, you'll appreciate this peek at one high-tech company's in-house yoga class. Unleash your downward-facing dog and submit an engineering- or technology-related caption for this cartoon showing an engineer who just can't let go of work even while on hands and knees in the balancing-table pose. The image might inspire a caption like "The yoga teacher spoke of opening your chakras, but Joe thought she was talking about the Chakra open-source operating system."

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

The contest begins at 12:01 a.m. Pacific Time on July 15, 2014. All entries must be received by the sponsor by 5 p.m. PT on Oct. 1, 2014.

Om!

STEVEN DICK, functional department manager (retired) at Northrop Grumman in Norwalk, Conn., won a shiny new Digilent Zynq Zybo board with this caption for the garden-in-the-lab cartoon in Issue 87 of *Xcell Journal*:

"I chose a garden-variety daisychain interrupt scheme."

Congratulations as well to our two runners-up:

"So is this supposed to happen with your random seed for the Bloom filter?"

— *Chris Lee, technical leader, Cisco Systems, San Jose, Calif.*

"… and the best thing is, they're nearly bug free!"

— *Don Ransbury, electronic design engineer, Tri-Z Electronic Design, San Diego*