Issue 70
First Quarter 2010

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

Broadcast Evolves to 'Broader-cast' Thanks to FPGA-Based Innovations

INSIDE

XILINX®

www.xilinx.com/xcell/

Design Platforms Accelerate Application Success

Development kits help ramp up new Spartan®-6 or Virtex®-6 FPGA designs

Avnet Electronics Marketing introduces three new development kits based on the Xilinx Targeted Design Platform (TDP) methodology. Designers now have access to the silicon, software tools and reference designs needed to quickly ramp up new designs. This approach accelerates time-to-market and allows you to focus on creating truly differentiated products.

Critical to the TDP methodology is the FPGA Mezzanine Card (FMC) from the VITA standards body. Avnet has collaborated with several industry-leading semiconductor manufacturers to create a host of FMC modules that add functionality and interfaces to the new baseboards, allowing for easy customization to meet design-specific requirements.

Learn more about the new Spartan-6 and Virtex-6 FPGA baseboards and FMC modules designed by Avnet at www.em.avnet.com/drc

DESIGNED BY AVNET

New baseboards for Spartan®-6 and Virtex®-6 FPGAs

- *Spartan-6 LX16 Evaluation Kit*
- *Spartan-6 LX150T Development Kit*
- *Virtex-6 LX130T Development Kit*

New FMC Modules for Baseboards

- *Dual Image Sensor FMC*
- *DVI I/O FMC*
- *Industrial Ethernet FMC*

More are soon to be released!

AVNET® electronics marketing

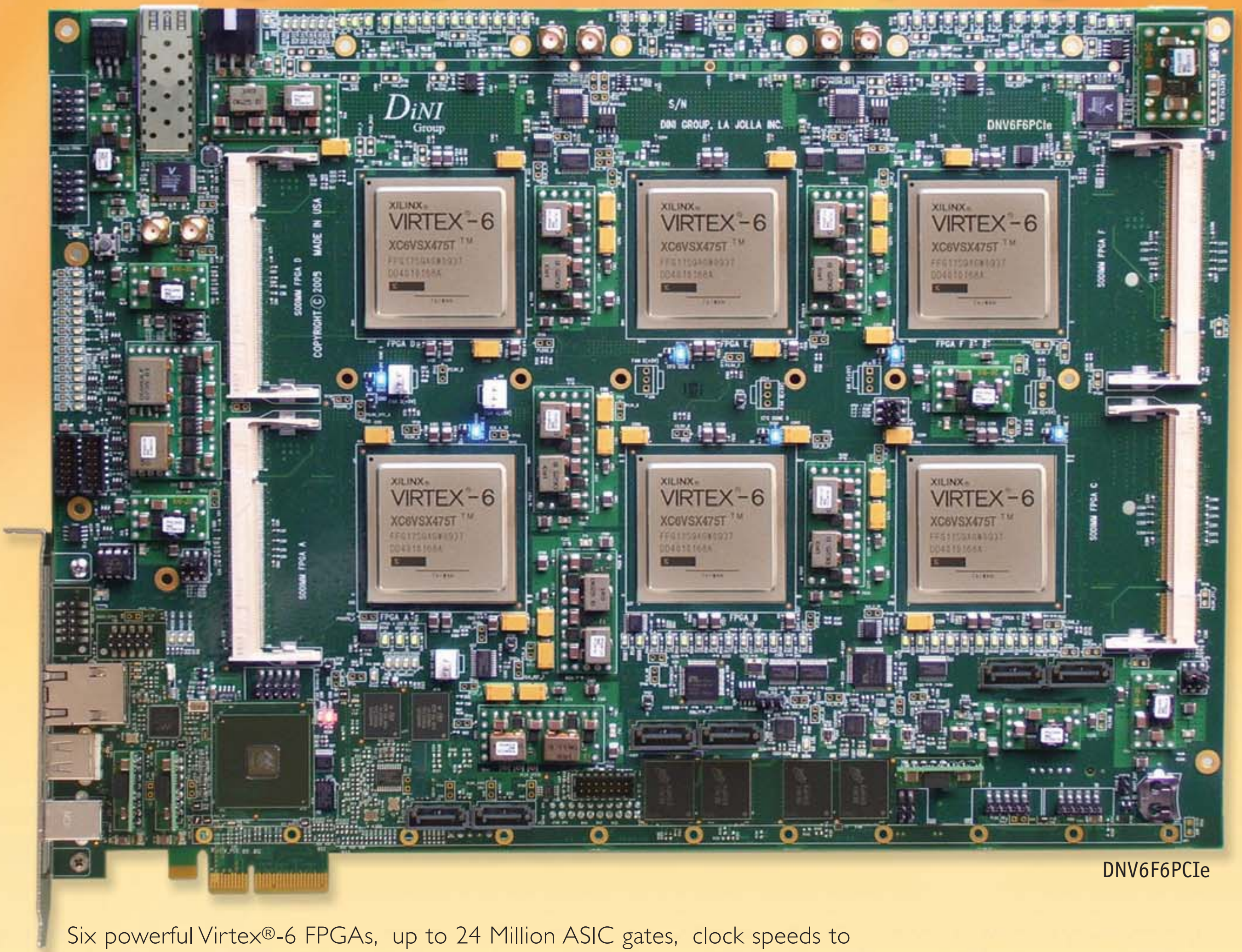
Accelerating Your Success™

Avnet Green Initiative

NEDA Supplier Authorized Distributor

1 800 332 8638
www.em.avnet.com

NEW Virtex-6x6 — More Power and Speed for ASIC Prototyping and High Performance Computing



DNV6F6PCIe

Six powerful Virtex®-6 FPGAs, up to 24 Million ASIC gates, clock speeds to 710 Mhz: this new board races ahead of last generation solutions. The Dini Group has implemented new Xilinx V6 technology in an easy to use PCIe hosted or stand alone board that features:

- 4 DDR3 SODIMMs, up to 4GB per socket
- Hosted in a 4-lane PCIe, Gen 1 slot
- 4 Serial-ATA ports for high speed data transfer
- Easy configuration via PCIe, USB, or GbE
- Three independent low-skew global clock networks

The higher gate count FPGAs, with 700 MHz LVDS chip to chip interconnects, provide easier logic partitioning. The on-board Marvell Dual Sheeva processor provides multiple high speed interfaces optimized for data throughput. Both CPUs are capable of 2 GFLOPS and can be dedicated to customer applications.

Order this board stuffed with 6 SX475Ts—that's 12,096 multipliers and more than 21 million ASIC logic gates—an ideal platform for your DSP based algorithmic acceleration and HPC applications.

Don't spin your wheels with last year's FPGAs, call Dini Group today and run your bigger designs even faster.

DINI Group

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

Xcell journal

PUBLISHER	Mike Santarini mike.santarini@xilinx.com 408-626-5981
EDITOR	Jacqueline Damian
ART DIRECTOR	Scott Blair
DESIGN/PRODUCTION	Teie, Gelwicks & Associates 1-800-493-5551
ADVERTISING SALES	Dan Teie 1-800-493-5551 xcelladsales@aol.com
INTERNATIONAL	Melissa Zhang, Asia Pacific melissa.zhang@xilinx.com Christelle Moraga, Europe/ Middle East/Africa christelle.moraga@xilinx.com Miyuki Takegoshi, Japan miyuki.takegoshi@xilinx.com
SUBSCRIPTIONS	All Inquiries www.xcellpublications.com
REPRINT ORDERS	1-800-493-5551

Xcell PUBLICATIONS

www.xilinx.com/xcell/

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

Xilinx Boards International Space Station

In 1988, only a few years after Xilinx was founded, the company established a group focusing solely on the aerospace and defense (A&D) market. In the two decades since then, a growing number of government contractors and agencies have discovered the advantages of Xilinx's A&D-grade FPGAs. Fast turnaround, reprogrammability and high performance have sent these devices into a slew of products ranging from satellites, weapons systems and radar to the two robot-geologist Mars Exploration Rovers, which are still at work on the red planet six years after the end of their expected operational lifetime. But while some have long seen that Xilinx's SRAM-based FPGAs offer unmatched advantages for this market, others have questioned whether they are resilient enough to stand up to the extremities of space, questioning in particular whether FPGAs can withstand bombardment by various cosmic rays, common in space and at high altitudes.

Roughly eight years ago, Xilinx's Carl Carmichael went on a mission to prove that FPGAs are tough enough for the job. He and Gary Swift, then an engineer at the Jet Propulsion Laboratory and now the single-event upset (SEU) expert in Xilinx's A&D Group, formed the SEU Consortium. Very recently Carmichael, Swift and the roughly 100 members of the consortium celebrated a milestone that could further increase the pace of adoption of Xilinx FPGAs for A&D applications.

In November of 2009, NASA astronauts brought two very important Xilinx-based experiments up to the International Space Station on space shuttle Atlantis' mission STS-129. The first, SEUXSE, is testing Xilinx® Virtex®-4 and Virtex-5 FPGAs to measure each product's intrinsic susceptibility and the efficacy of upset mitigation to shake off (via Xilinx Triple Module Redundancy software) the effects of various types of radiation strikes in real time. The second, SpaceCube, is testing the efficacy of an FPGA-based computing platform in space.

Created with the guidance of the SEU Consortium and co-funded by Xilinx, Sandia Labs and the Naval Research Lab, SEUXSE will not only show how each device while running stands up to radiation in space, but will also enable comparison of radiation hit rates with predictions from laboratory radiation-beam testing.

"This is the first time the consortium has run an experiment in space," said Swift. "Normally we use ground-based accelerators to simulate space radiation. The FPGAs in SEUXSE are fully powered and are recording and scrubbing radiation hits live. So far, we've taken a handful of radiation hits without any functional errors or system issues, so it is going fabulously."

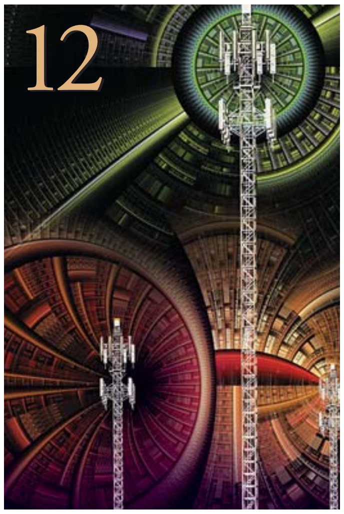
SpaceCube, for its part, is an FPGA-based reprogrammable general computing platform developed by space-avionics designer Gordon Seagrave for NASA/Goddard. Seagrave is another champion of Xilinx and of reconfigurability in space.

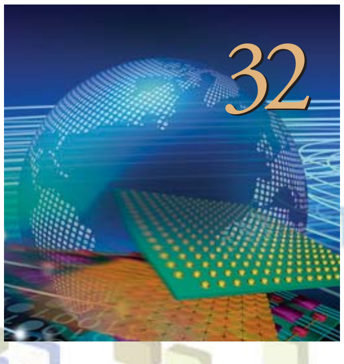
Last July, NASA first flew a Virtex-4 version of SpaceCube on the Hubble Servicing Mission (STS-125) to test its capability and, more so, its reliability in space. During that mission, SpaceCube recorded radiation strikes and successfully scrubbed all of them while executing an autonomous docking program for capturing the Hubble Space Telescope.

The latest version of the SpaceCube is now functioning in close proximity to SEUXSE. NASA engineers are running a series of video-based experiments on it to further test its reliability. With a successful demonstration, NASA could give the SpaceCube architecture a greater role in future missions. "The SpaceCube is the most sophisticated piece of electronics on the space shuttle," said Seagrave. It may be the most sophisticated piece of electronics in space today. In fact, NASA engineers recently named SpaceCube NASA's Invention of the Year for 2009.

Mike Santarini
Publisher

FIRST QUARTER 2010, ISSUE 70





From Broadcast to 'Broader-cast' with FPGA-Based Innovations

Xilinx devices propel television to HD, 3D and beyond.

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

Over the last 10 years, the electronics industry has delivered a dazzling array of flat-panel TVs and related technologies, pushing cathode-ray tubes, rear-projection television and analog standard-definition signal broadcasting into the dusty halls of antiquity. Today, companies are rolling out new TV technologies at a dramatic pace. But to bring the full value of this rapidly evolving technology to consumers requires an elaborate chain of equipment that allows broadcast companies and movie studios to send the latest advanced programming to the living room, the local cineplex and, soon, to your handheld device and your automobile.

Companies competing in the broadcast and cinema equipment markets have long used Xilinx® FPGAs as their primary logic ICs for most, if not all, broadcast equipment, from ultrahigh-end video cameras to just about every form of studio editing and signal-transport gear. Today, design teams in this market are well into the process of building out the FPGA-based equipment that will make 1080p broadcasting mainstream. They have even begun rolling out FPGA-based 3DTV equipment that will allow broadcasters to put *Avatar*-quality television onto the small screen. A closer look at the largest of these markets, the TV broadcast market, shows how designers are leveraging FPGAs across the entire broadcast equipment chain to bring ever-more-advanced programming to consumers.

Broadcast 101: From the Scene to Your Screen

Ben Runyan, marketing manager at Xilinx's Broadcast Segment Marketing Group, said that broadcast is one of those markets in which just about everyone sees the end result but few know how much goes into it. A great deal of complex technology is involved and a lot of work goes on behind the scenes to make it happen.

While out of the mainstream limelight, the broadcast and cinema markets are, in fact, highly competitive ones that involve high unit costs but relatively low volumes. Companies that play in these markets build just about every piece of equipment that brings a TV signal to your new digital TV, or a movie to the IMAX or screen at your local theater. Runyan notes that the cinema market is usually the early adopter of new broadcast technologies (see sidebar), but TV broadcasters are adopting new technology at an ever-more-rapid pace as the demand for high-quality broadcasts increases. "If you ever have the chance to tour a local TV studio or visit a network, you will be amazed by the massive amount of equipment and miles and miles of wiring that cover seemingly every inch of available space in the ceilings and walls," said Runyan. "What it takes—in terms of both effort and technology—to bring a program to a TV is remarkable."

For example, in a live-news broadcast, the reporter on your screen is speaking into a microphone while looking into a very advanced and expensive digital video camera. That camera sends—by wire or wirelessly—an uncompressed raw signal of voice and video to an on-the-scene news van that typically contains portable video viewing and editing equipment along with radio gear. All of the equipment in the van is able to instantly perform very light signal compression and send a live feed or related taped content directly to a transmission tower or to the broadcast studio via satellite.

In turn, the studio uses an elaborate array of equipment to receive the broadcast feed from the field, record a version of the content in a storage server and send the signal to mixing and editing equipment and multiscreen monitors in a control room. Program technicians, directors and producers then mix the content with other outside feeds and in-studio camera outputs (such as live-news anchor coverage) and add on-screen text or ticker tape graphics to coordinate the order of events and ultimately create a master program.

The broadcasters then send this master program to the transmission part of the studio, which will in turn compress it into multiple codec (coder/decoder) standards, each suited for transmission on particular media, such as cable, satellite, terrestrial and wired

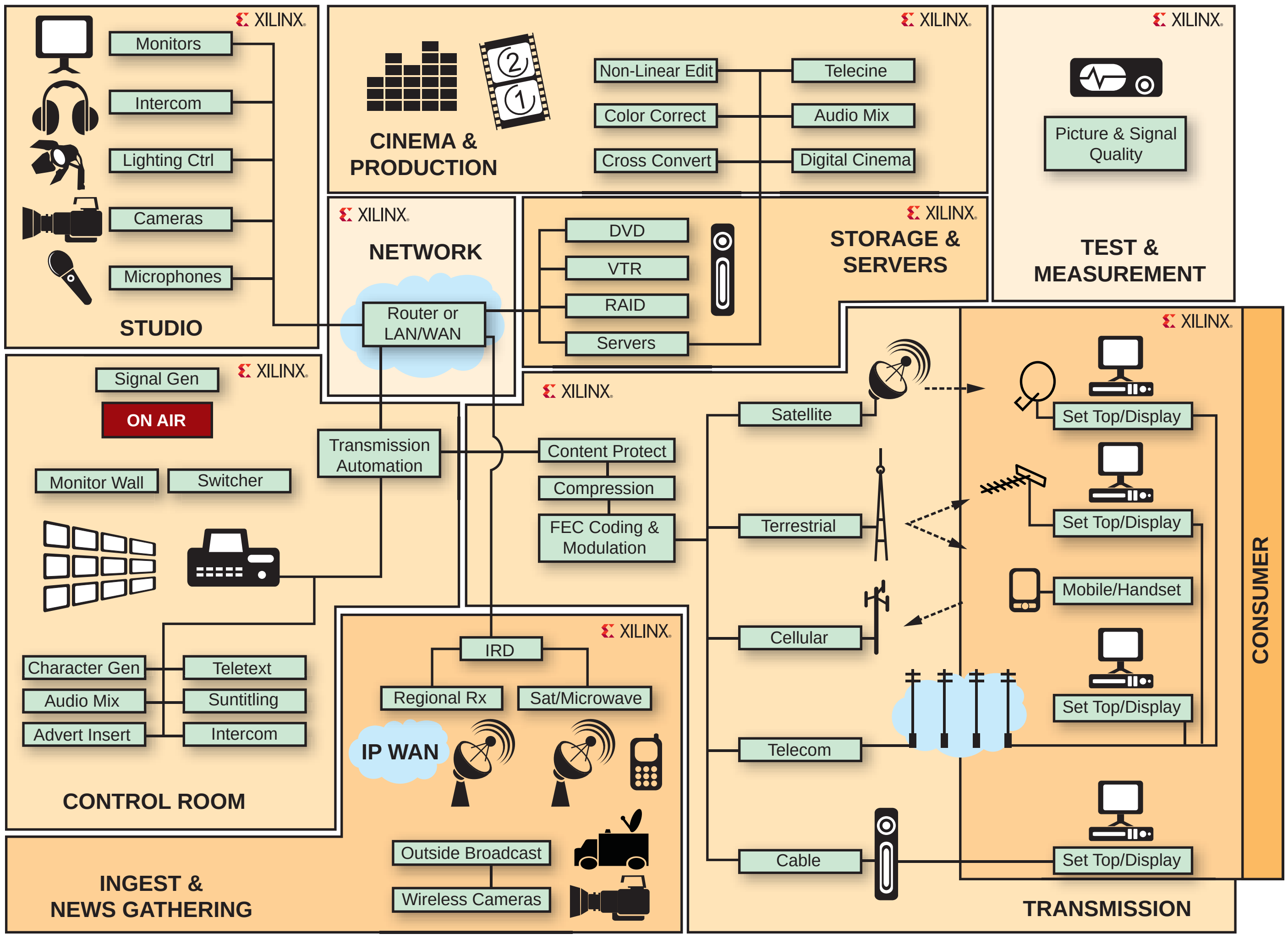


Figure 1 – Just about every piece of technology in the broadcast chain leverages Xilinx FPGAs.

broadcasting (for Internet Protocol TV) or wireless broadcasting (for cell phones). The studio then broadcasts these compressed bands through wired and wireless carrier networks (see cover story, *Xcell Journal* Issue 68) to get to your cable or satellite set-top box, digital TV, PC or handheld device. On the consumer end, these systems will decompress the signal and display the program.

Runyan said this entire broadcasting chain—especially live, on-the-fly broadcasting—requires a tremendous amount of high-bandwidth fiber and satellite resources along with high-performance video editing and vast arrays of storage. The equipment must also be extremely reliable, because it has to run 24 hours a day and seven days a week without losing one byte of video data. All of these requirements—high throughput, flexibility and reliability—have made FPGAs the ideal choice for broadcast equipment manufacturers (see Figure 1).

"From the cameras on the scene to the equipment in the news van, to the editing equipment (via switchers, routers and multiviewers) in the studio, to post-editing equipment and large video server storage farms, to the compression and transmission equipment—Xilinx FPGAs play a key role in all of it," said Runyan (see Figure 2). "In fact, broadcast is one market in which FPGAs long ago displaced ASICs and ASSPs and are now the dominant logic IC." The volumes of these highly advanced systems simply don't justify the costs of creating an ASIC, he said. Meanwhile, ASSPs "don't have the raw hardware performance to cut it in this market," Runyan said.

As the market moves to 1080p, 3D and mobile broadcasting over the course of the next few years, the specifications of TV broadcast equipment will become even more demanding, requiring greater raw performance, reliability and, particularly, flexibility. As such, the usage of FPGAs will also increase, said Runyan. To help further speed up innovation and productivity, Xilinx has launched its Targeted Design Platform strategy in support of its latest FPGAs, the Virtex®-6 and Spartan®-6 families.

Specifically, the soon-to-be released Virtex-6 FPGA Broadcast Connectivity Kit will give equipment makers a major leg up in this market.

"With Targeted Design Platforms, customers can spend more time on product differentiation and less time developing a system from the ground up," said Runyan. "We offer a number of broadcast-specific platforms that we've specifically tailored to help companies create equipment that requires advanced video, audio and network connectivity; real-time high-definition video processing; multichannel professional encoding and decoding, as well as high-speed digital signal processing for transmission and modulation."

Top TV Broadcasting Trend: 1080p

The reason the broadcast industry is advancing so rapidly today is because many people fought many standards battles, big and small, over the last 25 years to create the playing field for the digital-TV age, said Douglas I. Sheer, CEO and chief analyst with DIS Consulting Corp. (the broadcast-analyst partner of Strategy Analytics) and a 40-year veteran of the broadcast community. "This is allowing companies to create new types of broadcasts—audio and video—and deliver them to us in a growing number of ways," said Sheer. "What's been remarkable is the blending of IT with traditional broadcasting—a lot of people use their computers and, going forward, their handhelds to watch programming. It's what the National Association of Broadcasters calls 'Broader-casting.' "

Today, most TV broadcast companies are in the process of updating their studios or building new ones outfitted with 3-Gbps technology to support 1080p-resolution, full high-definition and wireless video broadcasting.

Roughly eight years ago, while most of the industry was still defining the high-def road map, some television networks, such as ESPN, made the bold move to build million-dollar transmission equipment and full digital 720p HDTV studios. Many other networks, however, held on to their legacy equipment as long as possible.

Due in part to government regulation, the North American and, now, worldwide markets are seeing a mass migration to digital broadcasting, said Xilinx's Runyan. In the United States, many of the companies that resisted going to 720p are bypassing 720p altogether to build or update studios that will support 1080p HD broadcasting.

"The move isn't a trivial matter, even for the early adopters of 720p," said Runyan. "The 720p progressive-scan format uses HD serial digital interface rates of roughly 1.5 Gbps, while 1080p at 60 frames per second has a data rate of about 3 Gbps. That's twice the bit rate and twice the bandwidth [of 720p], so you can get twice the amount of data."

A lot of people use the term "HD" broadly, he said, "and at the store perhaps they don't see a big difference in their TV resolutions. But in terms of broadcast equipment sophistication, there is a huge difference between 720p and 1080p."

Analyst Sheer said that over the last 10 years, most broadcast companies have had to take the first big step toward HD—and a hefty financial hit—by buying and installing

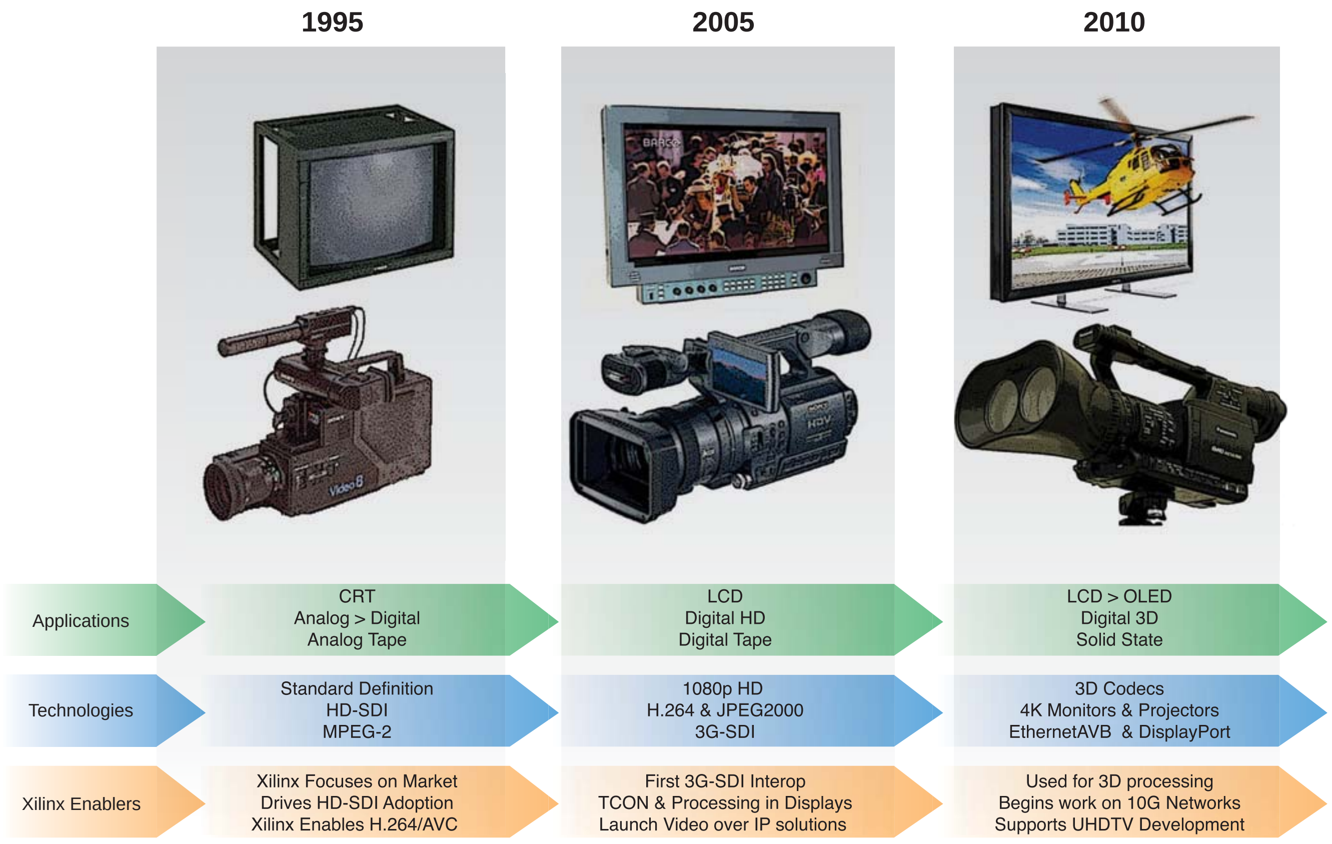


Figure 2 – Xilinx continues to deliver enabling technologies to broadcast innovators through the Targeted Design Platforms.

digital transmission equipment. They are now following up by starting to purchase 3-Gbps equipment to support 1080p as most consumers upgrade their sets to more reasonably priced 1080p HDTVs. "It's a push-and-pull as to what drives what," he said. "The 1080p consumer TV market was growing before the recession and declined during the recession, but has begun to rebound as the economy seems to be improving. Most broadcasters see that 1080p is where the consumer demand is headed, and so they are just now starting to buy equipment for it."

Runyan notes that at the data rate requirements for 1080p, every piece of equipment in the broadcast studio and throughout the entire TV chain needs greater bandwidth and some amount of hardware programmability to handle new functionality and any changes the industry may make to codec standards.

"For most companies, this move to 1080p is a huge capital investment," said Runyan. "Some of these companies have been using the same equipment for 20 years and now they have to change to new equipment. In some cases, they are being forced to do it simply because no one is building the equipment to support their legacy equipment."

For example, studios employ several broadcast quality gurus who monitor each video feed for quality. In the past these specialists have used ultrahigh-end CRT TVs, but today no one is making them, so broadcasters must move to new flat-panel formats. "There's a big debate among the folks who have trained eyes that spot every artifact, every color and shade, as to which format—LCD, plasma or OLED—and which brands make the best studio-quality monitors," said Runyan.

In the transition to 1080p, Runyan said, broadcast equipment will mainly use the new H.264/MPEG-4 AVC codec standard, but will also have to support MPEG-2 and other legacy standards. The H.264/MPEG-4 AVC codec is associated with Blu-ray and ultrahigh-definition streaming video. As is the case with most emerging video specifications, H.264/AVC is a multipart standard. Today the base features are fully defined, but the standard continues to evolve with new derivatives such as the Scalable Video Codec (SVC) and Multi-Viewer Codec (MVC) for 3D broadcasts. The standard will be more solidly defined as the broadcast companies put MPEG-4 to greater use and discover what additional functionality they will need for fast and reliable broadcasting in HD and 3D.

"These codecs are often works in progress as equipment vendors strive to

In the Footsteps of Digital Cinema

Where TV broadcasting today is ramping up to 1080p and 3D, digital cinema is light years ahead. It has a base resolution of 2K pixels (2,048 x 1,080 pixels), which requires double the bit rate of 1080p HDTV. The higher-resolution digital-cinema cameras operate at 4,520 x 2,540 pixels (4K x 2K).

But TV will get there one day.

"Cinema usually leads the way in terms of video technology," said Ben Runyan, marketing manager at Xilinx's Broadcast Segment Marketing Group. "These big-budget films like *Avatar* have the budgets to use the most bleeding-edge equipment, much of it custom in this early stage. Eventually, the technology makes its way to the TV broadcast market, but it usually takes some time because they have to build the infrastructure to support it."

Because these signals are so large, every piece of equipment—from super-high-end cameras to the cinematographic and editing equipment—has to be able to process massive signal feeds very rapidly to shoot movies natively in 3D and create stunning live images that mix seamlessly with special effects.

It's not only the movie studios that need more-advanced equipment. Theater equipment that projects these films also has to become more sophisticated. Runyan notes that the more-advanced movie theaters today have the ability to receive live content directly from the studios via satellite. Theaters can also now project or rebroadcast live performances and sporting events.

This capability "allows the theaters to focus on selling concessions, which is where they really make their money," said Runyan. "At the same time, it helps the studios in multiple ways."

Traditionally, movie studios would send theaters film canisters, tapes or discs of their films, to be shown by means of high-end projectors.

Every year, however, movie studios lose a boatload of money to pirates. To curb this theft of their intellectual property, they are directly downloading and even streaming movies right to theater projectors. Doing so cuts down on distribution costs, makes it faster for theaters to carry the latest movies and acquire known-good copies of those titles, while also making it harder for pirates to copy the content. This is also driving efforts in high-performance security, encryption and copy protection.

Of course, to support this trend, theaters must have equipment advanced enough to receive and ultimately project the movies.

– Mike Santarini

continuously improve video quality, and parts of the codecs may change, which is why FPGAs have been and will remain the best choice for this market," said Runyan. "With an FPGA-based system, companies can add new functionality as their needs change and as standards evolve. When a change to a codec occurs, it usually isn't one piece of equipment that needs to change, it's most of it." Xilinx and its intellectual-property (IP) partners offer a portfolio of codec IP to help customers quickly add new codec functions to their products, he said, "even after they've deployed them in the studio."

Sheer said that 1080p is just one of the many technologies that the new crop of 3-Gbps data-rate equipment will enable. "The timing of all these is a bit tricky," said Sheer. "The jury is still out as to exactly when 1080p will reach peak demand and, ultimately, which format will become the most popular." Thanks to 3-Gbps equipment, people will be able to watch broadcasts on their mobile devices and in their cars.

"Mobile broadcast isn't too good today, but at higher data rates, it probably will be," said Sheer. Seemingly, as the compute power and connectivity of handheld devices increases, wireless broadcasting will increase to keep pace with consumer demand to view broadcasts on laptops and netbooks.

The Next Big Market: 3DTV

In addition to building for 1080p broadcast TV, broadcast equipment manufacturers are also rolling out equipment that will allow broadcasters to bring sharper, 3DTV broadcasting to the home. Sheer and Runyan note that broadcast companies have been able to air 3D broadcasts for many decades, but not with truly sharp pictures. Now a new generation of high-definition 3D, which consumers have enjoyed in theaters in recent years, will soon be available in the home. This year at the Consumer Electronics Show in Las Vegas, several vendors introduced 3DTVs. Some require that viewers use 3D glasses, others don't.

The move to "3DTV will allow programmers to offer even greater premium services to subscribers," said Runyan. "They will be able to offer digital, HD or 3D cable or satellite TV packages and on-demand programming. You will not only be able to watch movies in 3D, but also live events like football and baseball games." In fact broadcasters in the U.K. (see *http://www.televisionbroadcast.com/article/93870*) have already begun to test live 3D broadcasting and plan to offer it more broadly for the 2010 World Cup.

For companies that have put 1080p studio equipment in place, the move to 3D broadcasting is a relatively small but fairly significant upgrade, Runyan said. To broadcast 3DTV signals the industry will need to deploy new H.264 codec technology and MVCs to handle the increased bandwidth from the multiple camera shots needed for 3DTV. That means the studio equipment will have to support the functionality and additional data throughput.

"The studio equipment has to be a bit more sophisticated," said Runyan. "At a minimum they will have to be able to simultaneously process two feeds: a 2D broadcast and a 3D broadcast. In some cases, broadcasters will use post-editing equipment to convert regular 2D programs into 3D. The more sophisticated way to do it is to shoot a program with 3D cameras the way that movie studios have done for many years—this is what James Cameron did for his blockbuster, *Avatar*—and then generate a 2D HD version from that content."

Most of this 3D technology hasn't been fully field-tested. Many still question whether 3DTV that doesn't require glasses is sharp enough, and whether viewers will want to wear 3D glasses every night.

Sheer believes that 3DTV will ramp fairly quickly once the market decides what does and doesn't work. "Everyone remembers sitting in theaters with cardboard glasses, but this generation of 3D technology, stereographic 3D, is substantially better," Sheer said. "It is being embraced by so many people in Hollywood, and it is even being used for live sports broadcasting. Perhaps the most important thing is that it is able to ride the back of HD." Indeed, where high-def "took a quarter of a century to come to fruition," Sheer said, "we believe 3D will do it in a fraction of that time, because much of the infrastructure is already in place with HD."

Leading the 3DTV market, in Sheer's view, will be 3D Blu-ray DVD players that can play the growing number of stereographic 3D movies coming out of the movie industry. Not long after their advent, 3DTV to the home will follow. "The draw of seeing live sports in 3D is huge," said Sheer.

Beyond 1080p and 3DTV

One would think that with 1080p TV broadcasting proliferating worldwide and 3D technology widespread in cinema and moving rapidly toward mainstream TV, the pace of broadcast technology advances would begin to slow as the industry paused to take a collective breather. Evidently that's not the case.

Runyan said companies are already demonstrating "ultra-HDTV" cameras that capture images at an astounding 7,680 x 4,320-pixel resolution, or 16 times the resolution of high-definition TV today. The bit rates required are so large that cameras can currently capture only 20 minutes' worth of images in a day. Sooner rather than later, with advances in electronics and the refinement of codecs, the ultra-HDTV-class video quality will undoubtedly move toward the mainstream. Or will it?

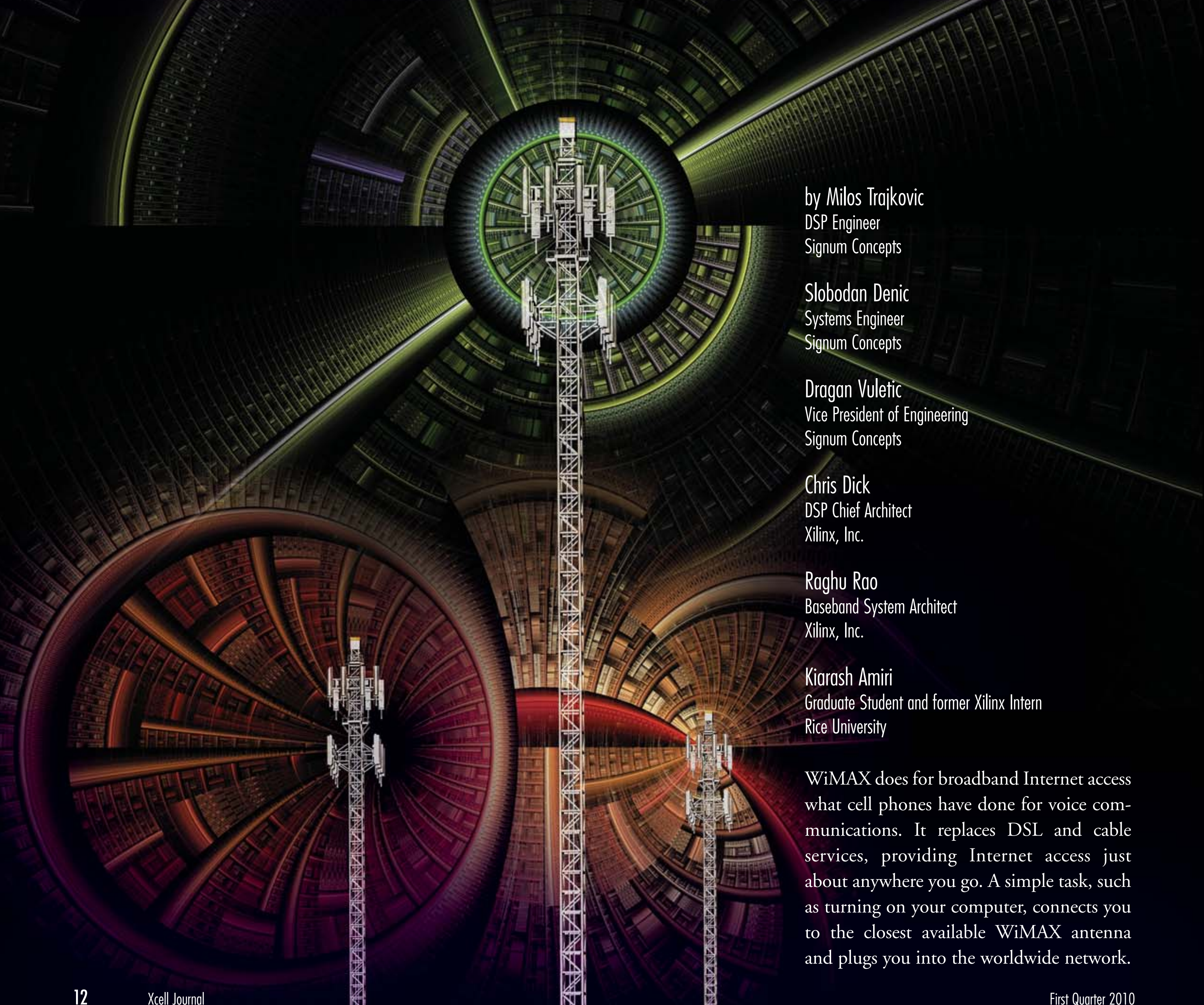
"If the industry achieves the bit rates they are talking about for ultra HDTV, we may not have TV or cinema as we know it today," said Runyan. "In fact, the bit stream will be so large it will be able to support true holographic projections. Imagine a live sporting event in holographic imagery, where you can see every angle. The zoom-in rate is a minimum 1080p, essentially the same as what we see on the sharpest commercial TVs today."

And at that point, we may not be watching broadcasts on TVs at all but a new type of holographic projection device, surely powered by Xilinx FPGAs.

To learn more about Xilinx's Targeted Design Platforms for the broadcast market, visit www.xilinx.com/broadcast. *To learn more about the Virtex-6 FPGA Broadcast Connectivity Kit, visit* www.xilinx.com/v6bck.

Implementing 4G Wireless Sphere Detector in FPGA

System Generator was the key to building a quasi-maximum-likelihood detector (4x4, 64-QAM) for spatial-multiplexing MIMO-OFDM systems.



by Milos Trajkovic
DSP Engineer
Signum Concepts

Slobodan Denic
Systems Engineer
Signum Concepts

Dragan Vuletic
Vice President of Engineering
Signum Concepts

Chris Dick
DSP Chief Architect
Xilinx, Inc.

Raghu Rao
Baseband System Architect
Xilinx, Inc.

Kiarash Amiri
Graduate Student and former Xilinx Intern
Rice University

WiMAX does for broadband Internet access what cell phones have done for voice communications. It replaces DSL and cable services, providing Internet access just about anywhere you go. A simple task, such as turning on your computer, connects you to the closest available WiMAX antenna and plugs you into the worldwide network.

One of the greatest challenges in broadband Internet access is mobility, and that is what the latest WiMAX standard defines. IEEE 802.16e-2005 introduces the usage of multiple antennas at transmission and reception in a concept known as MIMO, or multiple input, multiple output, a key feature in mobile WiMAX.

Spatial-division multiplexing (SDM) MIMO processing significantly increases the spectral efficiency, and hence capacity, of a wireless communication system. Spatial-multiplexing MIMO communication systems have recently drawn attention as a means to achieve tremendous gains in wireless-system capacity and link reliability.

The optimal hard-decision detection for MIMO wireless systems is the maximum-likelihood (ML) detector. ML detection is attractive due to its superior performance in terms of bit-error rate (BER). However, direct implementation grows exponentially with the number of antennas and the modulation scheme, making an ASIC or FPGA version unfeasible for all but low-density modulation schemes that use only a small number of antennas.

A prominent method for MIMO detection that maintains BER performance comparable to optimum ML detection while significantly reducing compute complexity is called sphere detection. This technique reduces detection complexity in both SDM and space-division multiple-access systems while maintaining BER performance comparable with that of optimum ML detection. There are several approaches for realizing sphere detectors, and the algorithmic landscape is rich with methods that enable the designer to make various tradeoffs among performance metrics—for example, throughput of the wireless channel, BER and implementation complexity.

While the algorithm (such as K-best or depth-first-search) and hardware architecture clearly have a strong influence on the resulting BER performance of the MIMO detector, the channel matrix preprocessing typically conducted prior to sphere detection likewise has a big impact. The channel matrix preprocessing can range from very simple processing that might, for example, compute a preferred order for processing the spatially multiplexed data streams, based on variance computations applied to the channel matrix, through to much more sophisticated matrix factorization techniques that determine the preferred order for processing the streams in a more-optimal (in the BER sense) manner.

Signum Concepts, a San Diego-based communication systems development company, working together with Xilinx and Rice University, designed an FPGA implementation of a MIMO detector for spatial-multiplexing MIMO in 802.16e broadband wireless systems. By utilizing a channel matrix preprocessor that realizes a type of successive-interference cancellation similar in concept to that employed in Bell Labs Layered Space Time, or BLAST, processing, the detector achieves close to maximum-likelihood performance.

System Considerations

Ideally, the detection process requires computing the ML solution across all possible combinations of the symbol vector. The goal of the sphere detector is to reduce the required compute complexity by using simple arithmetic operations, while simultaneously retaining the numerical integrity of the final result. In our approach, the first step decomposes the complex valued-channel matrix into an expression that involves real-valued numbers only. This operation increases the matrix dimension while at the same time simplifying the arithmetic required to manipulate the matrix elements. The second complexity reduction involves lowering the number of symbol candidates the detection scheme analyzes and processes. QR decomposition of the channel matrix is a key step here.

Figure 1 illustrates the mathematical transformations that lead to the final expression of computing the partial Euclidian distance metrics that are fundamental to the sphere detection process. The triangular form of the R matrix enables an iterative method for processing the candidate symbols starting with the matrix element rM,M where M is the dimension of the channel matrix expressed using real-valued quantities. The solution

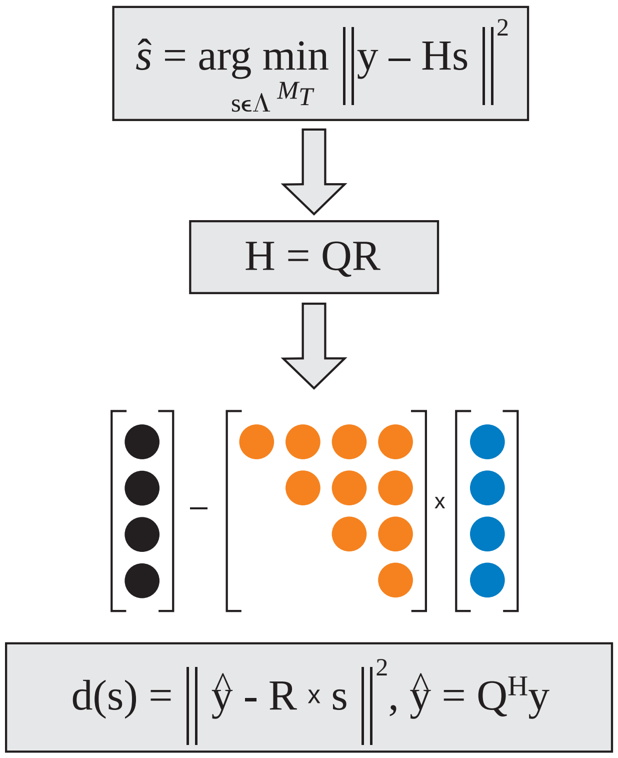

$\hat{s} = \arg\min_{s \in \Lambda^{M_T}} \left\| y - Hs \right\|^2$
$H = QR$
–
x
$d(s) = \left\| \hat{y} - R \times s \right\|^2, \hat{y} = Q^H y$

Figure 1 — Partial Euclidian distance metric equation for sphere detector-based MIMO detection

The order in which the sphere detector processes the antennas has a profound impact on the BER performance. Therefore, prior to sphere detection, our design applies channel reordering using a V-BLAST-like technique.

is delivered in M iterations defining the tree traversal structure, where each level of the tree i corresponds to processing symbols from the ith antenna.

There are several candidate methods for realizing the tree traversal. Our implementation employs a breadth-first search due to the attractive feed-forward structure (and hence hardware-friendly nature) of the approach. At each level, the implementation chooses only the K survivor nodes with the smallest distance for the expansion

The order in which the sphere detector processes the antennas has a profound impact on the BER performance. Therefore, prior to sphere detection, our design applies channel reordering using a V-BLAST-like technique.

The method determines the optimum detection order of columns of the channel matrix by calculating the norms of the rows of the pseudo-inverse of the channel matrix over several iterations. Depending on the iteration count, the technique will choose the row with the maximum or minimum norm. The row (of the inverse matrix) with the minimum Euclidian norm represents the influence of the strongest antenna, while the row with the maximum Euclidian norm represents the influence of the weakest antenna. This novel approach first processes the weakest stream. All subsequent iterations process the streams from highest to lowest power.

FPGA Hardware Implementation

To implement the described system, we used Xilinx® Virtex®-5 FPGA technology. The design flow employed Xilinx System Generator for design capture, simulation and verification. To support the different number of antenna/user and modulation orders, we designed the detector for the most demanding 4x4, 64-QAM case.

Our model assumes that the channel matrix is perfectly known to the receiver, which can be accomplished by classical means of channel estimation. After channel reordering and QR decomposition, we apply the sphere detector. In preparation for engaging a soft-input, soft-output channel decoder (for example, a turbo decoder), we produce soft outputs by computing the log-likelihood ratio (LLR) of the detected bits.

The main architectural elements of the system include the data subcarrier processing and features for managing the system submodules to process the desired number of subcarriers in real time while simultaneously minimizing processing latency. The channel matrix is estimated for every data subcarrier, which limits the available processing time for every channel matrix. For the selected FPGA, with a target clock frequency of 225 MHz and a communication bandwidth of 5 MHz (corresponding to 360 data subcarriers in a WiMAX system), the available number of processing clock cycles per channel matrix interval is 64.

We used careful pipeling and time-division multiplexing (TDM) of the hardware functional units to meet the real-time deadline for a WiMAX OFDM symbol.

Besides the high data rate, managing the latency of the submodules was also an important issue guiding the design architecture. We solved the latency issue by introducing TDM of the successive channel matrices. This approach provided more processing time between the matrix elements of the same channel while still sustaining high data throughput. The number of channels comprising a TDM grouping varies as a function of the specific submodule. The channel matrix inversion process employs five channels in the TDM scheme, while 15 channels are time-division multiplexed in the real-valued QR decomposition module. Figure 2 is the high-level diagram of the system.

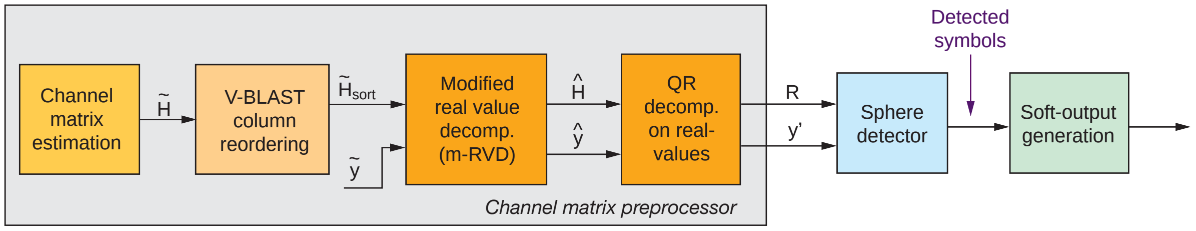

Channel matrix estimation
$\tilde{H}$
V-BLAST column reordering
$\tilde{H}_{sort}$
$\tilde{y}$
Modified real value decomp. (m-RVD)
$\hat{H}$
$\hat{y}$
QR decomp. on real-values
Channel matrix preprocessor
R
y'
Sphere detector
Detected symbols
Soft-output generation

Figure 2 – High-level diagram of the MIMO 802.16e broadband wireless receiver

Channel Matrix Preprocessing

The channel matrix preprocessor determines the optimum detection order for each layer in the spatially multiplexed composite signal. The preprocessor computes the norms of the channel matrix pseudo-inverse and, based on those norms, selects the next transmit stream that is to be processed. The row of the pseudo-inverse with the minimum norm corresponds to the strongest transmission stream (smallest post-detection noise amplification), while the row with the largest norm (largest post-detection noise amplification) represents the layer with the poorest quality. Our implementation detects the weakest layer first, with the following layers being ordered from lowest to highest noise amplification. For each step in the ordering process, the corresponding column in the channel matrix is then nulled, and this deflated matrix proceeds to the next stage of the antenna-ordering processing pipeline.

The calculation of the pseudo-inverse matrix is the most demanding component in the preprocessing algorithm. The heart of the process is matrix inversion, which is realized using QR decomposition (QRD) implemented by means of Givens rotations. The high system latency associated with the well-known algorithms for angle estimation and planar rotations, such as CORDIC, were unacceptable for our system. Hence, the goal was to find an alternative solution for vector rotation and phase estimation using the FPGA's embedded DSP resources (the DSP48E, in the case of Virtex-5 devices).

The systolic-array QRD structure consists of two types of processing cells, diagonal, or boundary, cells and off-diagonal or inner cells. The boundary cells perform a vectoring function and generate rotation angles that the inner cells of the array use. You can obtain the desired rotations by multiplying the value contained in an off-diagonal cell with the complex conjugate number in a diagonal cell and scaling by reciprocal value of the complex-number magnitude. The division is done as a multiplication, with a reciprocal value calculated using a polynomial approximation on the defined interval where the function is observed to be close to linear. Using the approximation, Figure 3 shows the final signal flow graph of the complex rotator in the diagonal systolic cell.

The data sent to the off-diagonal cells actually consists of in-phase and quadrature components of the rotated vector scaled by the corresponding approximated value. We obtained high data throughput using a pipelined architecture for the diagonal and off-diagonal cells, while managing the latency introduced by the approximation module and complex multiplier by means of time-division multiplexing of the hardware across five channels.

We implemented one diagonal and seven off-diagonal cells for the 4x4 matrix. The processing time to decompose a single matrix is 4 x 4 = 16 data cycles. The design delivers data at a rate of one sample every three clock cycles, so that total time to decompose a single matrix is 3 x 4 x 4 = 48 clock cycles (out of the available 64). We implemented back substitution of the decomposed matrix along with further reordering operations in the same TDM manner.

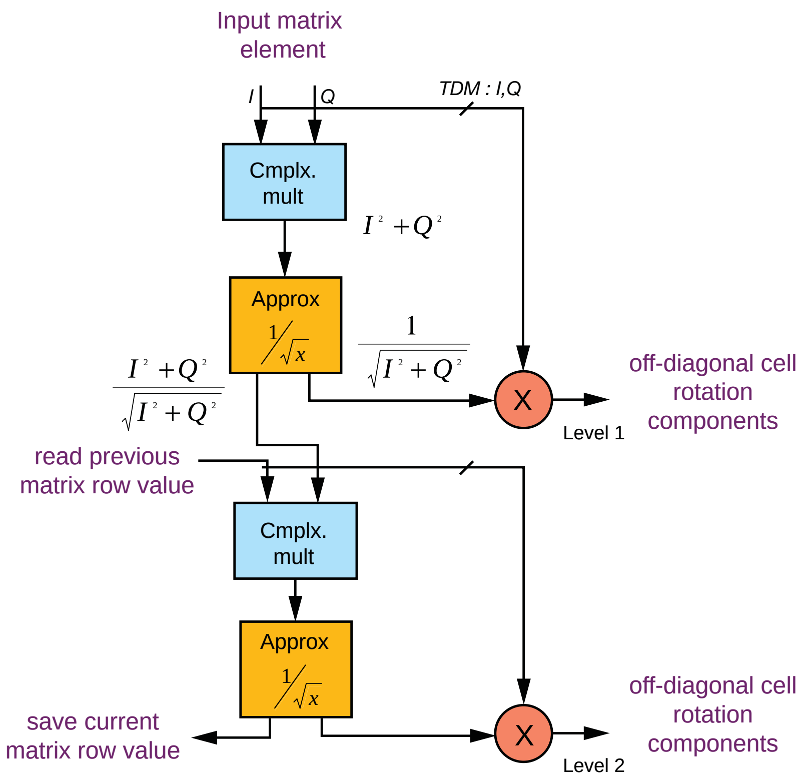

Input matrix element
I
Q
TDM : I,Q
Cmplx. mult
I^2+Q^2
Approx $1/\sqrt{x}$
$\frac{1}{\sqrt{I^2+Q^2}}$
$\frac{I^2+Q^2}{\sqrt{I^2+Q^2}}$
X
Level 1
off-diagonal cell rotation components
read previous matrix row value
Cmplx. mult
Approx $1/\sqrt{x}$
save current matrix row value
X
Level 2
off-diagonal cell rotation components

Figure 3 – Block diagram of the diagonal systolic-array cell

Sphere Detector

The sphere detector's partial Euclidian distance (PED) cells perform the norm computation. Depending on the level of the tree, we use three different types of PED cells. The root-node PED block calculates all possible PEDs. The second-level PED block computes eight possible PEDs for each of the eight survivor paths generated in the previous level. This will give us 64 generated PEDs for the next tree-level index. The third type of PED block is used for all other tree levels that compute the closest-node PED for all PEDs computed on the previous level.

The pipeline architecture of the sphere detector (SD) allows data processing on every clock cycle. As a result, only one PED block is necessary at every tree level. Thus, the total number of PED units is equal to the number of tree levels, which for a 4x4 64-QAM system is eight.

The SD can employ two types of decoding techniques: hard decoding, which determines the sequence with the minimum distance metric through all levels in the tree, and soft decoding, which represents each output bit as a log-likelihood ratio value. LLR values are typically supplied as an a priori input value to the channel decoder, e.g. turbo decoder.

FPGA Resource Footprint

The implementation and simulation included the detection process illustrated in Figure 2, with the exclusion of the soft-output generation module. The target chip was a Virtex-5 XC5VFX130T-2FF1738 FPGA. The design clock frequency was 225 MHz and the achievable data rate in that case was 83.965 Mbits/second.

Table 1 shows the resource consumption for each of the key functional units in the design. The percentage utilization values indicate FPGA area expressed relative to an XC5VFX130T device.

Function	Slices	LUTs/FFs	DSP48	Block RAM
Channel preprocessing	9,999 (48%)	20,339/29,954 (24%)	159 (49%)	105 (17%)
RVD QRD	1,715 (8%)	4,418/5,556 (5%)	30 (9%)	27 (4%)
Sphere detector	2,445 (11%)	3,113/6,525 (3%)	48 (15%)	12 (2%)

Table 1 – Resource footprint summary by subsystem

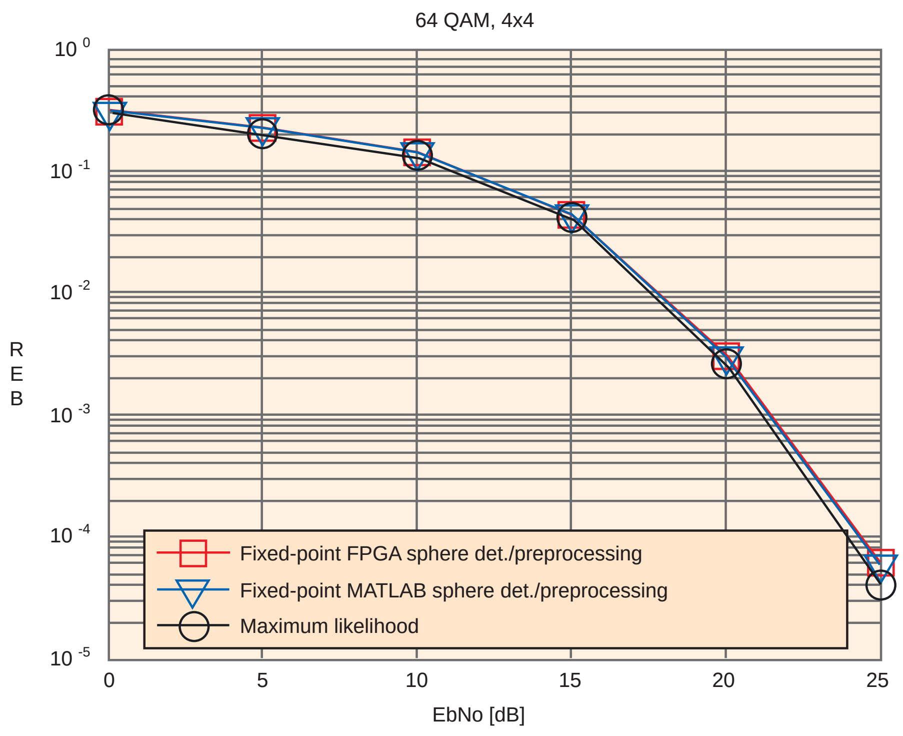


Figure 4 – BER curves comparing the 4x4 64-QAM system for the floating-point MATLAB simulation (hard decision), System Generator design (hard decision) and maximum-likelihood curve

System Generator and Model-Based Design

We realized the entire detection chain using the Xilinx System Generator for DSP design flow. Design validation employed not only the simulation semantics of the MATLAB®/Simulink® environment but also the co-simulation capabilities of System Generator. In-phase and quadrature components of the channel matrix coefficients are drawn from a normal distribution and delivered from MATLAB to the System Generator modeling environment. We likewise computed the bit-error rate using this simulation framework. Figure 4 contrasts the BER plots for our fixed-point hard-decision implementation, the floating-point version of the model and the golden ML reference curve. We developed a hardware demonstration of the design via Ethernet-based hardware co-simulation for the Xilinx ML510 development platform. The channel matrix coefficients are delivered to the sphere detector using the Xilinx AWGN IP core. We computed the BER by embedding the design in a self-synchronizing BER tester, which provided the input to the detector and captured the bit errors.

This paper has provided a brief overview of a sphere detector for communication systems employing spatial-multiplexing MIMO. We have presented the architectural details of the sphere detector and the channel matrix preprocessor. There are many ways to realize the preprocessing, and while our method is computationally complex, the resulting BER performance is close to maximum likelihood. While we have couched our discussion around WiMAX, designers can apply many of these methods to 3G LTE (long-term evolution) wireless systems.

The next task for our team is improving the BER plots by implementing iterative soft detection using convolutional turbo codes and a soft-output generation module.

For more information, contact Chris Dick at *chris.dick@xilinx.com*. For Signum Concepts FPGA-based design services, contact Dragan Vuletic at *dragan.vuletic@signumconcepts.com*.

Quickly see inside your FPGA

Get the most out of your test equipment and shave time out of your next debug cycle with Agilent oscilloscopes and logic analyzers. Our innovative FPGA dynamic probe application allows you to quickly connect to multiple banks of internal signals for rapid validation with real time measurements. Reduce errors with automatic transfer of signal names from the .cdc file from your Xilinx Core Inserter.

Toggle among banks of internal signals for incremental real-time internal measurements without:

- Stopping the FPGA
- Changing the design
- Modifying design timing

Shown with: 9000 Series oscilloscope and 16900 logic analyzer with FPGA dynamic probes

Also works with all InfiniiVision and Infiniium MSO models.

See how you can save time by downloading our free application note.

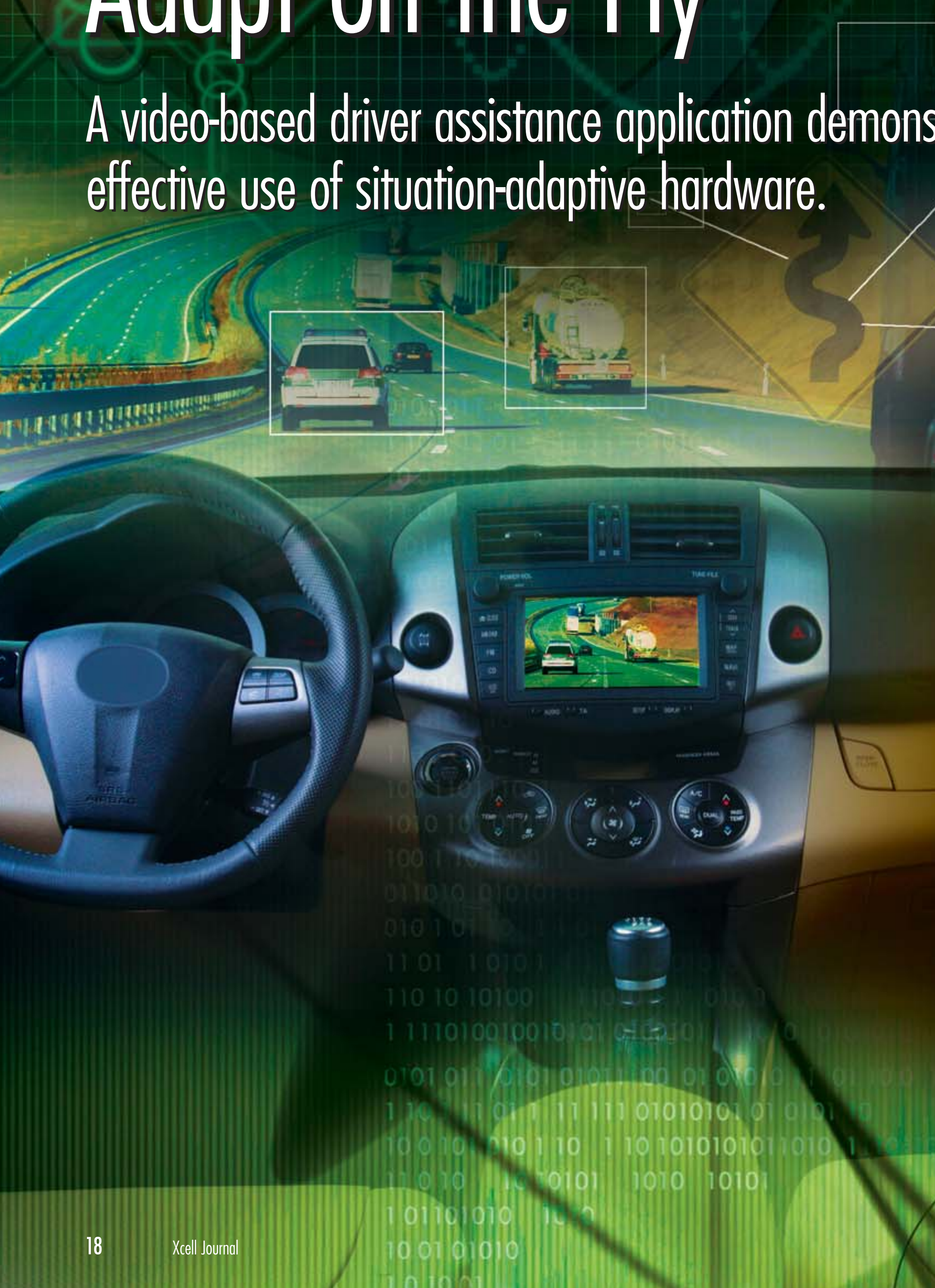
www.agilent.com/find/fpga_app_note

u.s. 1-800-829-4444 canada 1-877-894-4414

Agilent Technologies

Dynamic Partial Reconfiguration of Xilinx FPGAs Lets Systems Adapt on the Fly

A video-based driver assistance application demonstrates effective use of situation-adaptive hardware.



by Christopher Claus
PhD Candidate
Technical University of Munich, Germany
Christopher.Claus@tum.de

Florian Altenried
Master's Candidate
Technical University of Munich, Germany

Walter Stechele
Professor
Technical University of Munich, Germany
Walter.Stechele@tum.de

Video-based driver assistance systems demand real-time processing of complex algorithms in diverse situations. In many instances, based on available hardware in automotive environments, a pure software implementation does not offer this required real-time processing. Instead, such systems require hardware acceleration.

Dedicated hardware circuits such as application-specific integrated circuits (ASICs) or off-the-shelf application-specific standard products (ASSPs) can offer the required real-time processing, but they lack the necessary flexibility. In addition, the design times for ASICs and their development costs have risen over the past years. Because video algorithms for driver assistance are not standardized, design changes are quite frequent, which rarely makes an ASIC a suitable choice.

It's important to make as few design changes as possible for reconfigurability vs. a nonreconfigurable system. At the same time, reconfiguration has to be fast. If someone is using a video-based driver assistance system in a safety-critical environment, dropping a video frame is not acceptable.

FPGAs are an attractive alternative, since the programmable parts offer faster time-to-market, reduced costs and an opportunity for product differentiation. However, the amount of logical functionality that an FPGA design can realize differs greatly from that of an ASIC when using the same process technology.

The Institute for Integrated Systems at the Technical University of Munich, Germany, investigates multiprocessor system-on-chip architectures (MPSoC) for embedded systems. Another research focus is high-level simulation methods for early design space exploration of MPSoC, including networks-on-chip and memory hierarchies (*www.lis.ei.tum.de*). As part of that work, we undertook to develop a reconfigurable FPGA-based system-on-chip architecture for vision-based driver assistance that is capable of real-time processing and can rapidly adapt to any upcoming situation.

Architectural Concept

Our AutoVision architecture uses run-time reconfigurable hardware accelerator engines for future video-based driver assistance products. In the AutoVision system, only a portion of the device changes during run-time. A key to this research is the use of Xilinx® FPGAs, which offer the unique capability of dynamic partial reconfiguration (DPR), and embedded design tools such as Xilinx's ISE®, EDK and PlanAhead.™ Before we describe the architecture in detail, we clearly want to state that the approach is independent university research conducted with broad support from Xilinx.

Dynamically reconfigurable hardware is the best choice for video-based driver assistance systems that need a high degree of flexibility and inherent parallelism to achieve real-time constraints. Algorithms for video processing can be grouped into high-level application code and low-level pixel operations. High-level application code requires a great deal of flexibility and thus seems to be a good candidate for an implementation on an embedded CPU, such as a PowerPC® or MicroBlaze.™ Pixel manipulation, on the other hand, requires applying the same operation on many pixels in parallel and, thus, seems to be a good candidate for hardware acceleration.

Depending on various driving conditions, a system will have to use different algorithms for video processing. The pixel-level parts of these algorithms require different hardware accelerator engines, which we load into the AutoVision chip at run-time of the system. At the same time, we remove unused accelerators from the FPGA in order to save chip area.

Dynamic partial reconfiguration is available on certain commercial Virtex® families including the Virtex-5, which is what we used for this research project. DPR especially makes sense in mutually exclusive situations such as daytime-nighttime driving, forward-backward driving and driving at different velocities (urban environment vs. highway) and different weather conditions (rain, fog, snow and so on).

Logical-Resource Challenges

Prior to making a system reconfigurable, we first must make some investments in terms of logical resources by adding a reconfigurable interface, a controller for the Internal Configuration Access Port (ICAP) as well as a high-speed interconnect for the bitstream transfer. These are compensated for later in the design process with the help of DPR.

The ICAP allows read and write access to the configuration memory and thus can be used for on-chip self-reconfiguration. The reconfigurable interface consists of bus macros that guarantee a safe connection between the static and the reconfigurable part before, during and after the reconfiguration. Currently we use hardwired macros. In the future, these macros will no longer be necessary as the Xilinx development tools will one day guarantee a safe connection without requiring user intervention.

On the one hand, it's important to make design changes for reconfigurability as small as possible compared with a nonreconfigurable system. Nobody will accept DPR if the ICAP controller or other DPR-related logic occupies a huge number of Block RAMs (20 or even more) of a device. Thus, storing the partial bitstream in the on-chip BRAM is not a suitable solution.

On the other hand, the reconfiguration has to be as fast as possible. If someone is using a video-based driver assistance system in a safety-critical environment, dropping a video frame due to DPR is not acceptable. To guarantee a fast transfer from the bitstream data to the ICAP controller, we connected the ICAP controller directly to the Multi-Port Memory Controller (MPMC) via a Native Port Interface. Adding bus macros as a reconfigurable interface, an ICAP controller for self-reconfiguration and an additional port on the MPMC is considered DPR overhead. Depending on the amount of reconfigurations and when they have to be performed, we can divide the process into either inter- or intra-video frame reconfiguration, or InterVFR and IntraVFR respectively.

We define the InterVFR procedure as swapping reconfigurable modules between two consecutive video frames. In Figure 1(a), we denote the time to process an image with one engine as THW1 and that required by another engine as THW2. The reconfiguration time for swapping is denoted as TR, and includes clearing the reconfigurable region with a blank bit-

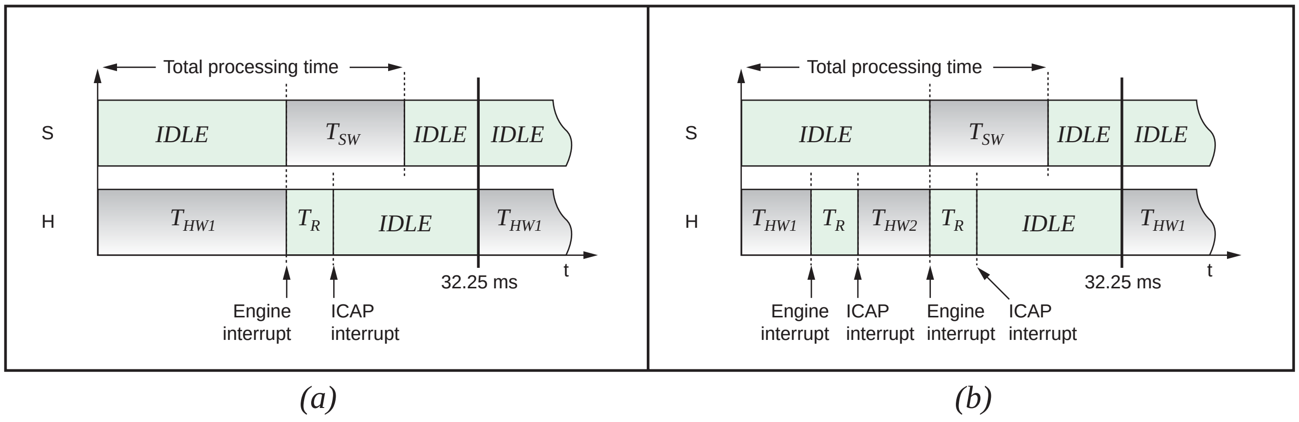


Figure 1 – Depending on the number of reconfigurations and when they are performed, the process is either inter- or intra-video frame reconfiguration, or InterVFR (a) and IntraVFR (b).

stream and loading the new module by using a second, partial bitstream. If THW1 + TR + TSW< 32.25ms (31 frames per second), then InterVFR is possible, which means that processing an image with an engine and reconfiguring the device can be done before THW2 starts.

The IntraVFR, for its part, involves swapping multiple reconfigurable modules within one video frame. The hardware accelerator for an FPGA-based optical flow calculation [1] can serve as an example here. It consists of two engines that execute sequentially and cannot be pipelined. The time to process an image with the first engine is denoted as THW1 and that required by the second engine as THW2. The partial bitstreams for both of the engines are considered to be the same size. Thus, the reconfiguration time for both engines is equal and is denoted as TR (blank and module bitstream) in Figure 1(b). If THW1 + THW2 + 2 _ TR < 32.25ms (31 fps), then IntraVFR is possible.

In the best case, the reconfiguration time TR overlaps with the time used to process the intermediate results on a CPU (denoted as TSW in the figure). As Figure 1(a) shows, no additional time need be reserved in InterVFR, because TR completely overlaps with TSW. This is also true for the second reconfiguration, when two sequentially running engines are swapped using IntraVFR. In that case the total processing time for one video frame will increase by the time for one reconfiguration TR.

Before an InterVFR or IntraVFR system could be implemented, the designers had to perform several steps in advance.

After creating a software (SW) prototype of the algorithm, we profiled the code in order to identify the performance-intensive parts, since these are the ones that should run in hardware (HW) later. We implemented hardware accelerators for the pixel operations by utilizing a modular, building-block-based pixel processing chain. Thus, we need implement from scratch only the operation on a single pixel and its neighborhood, as the modular AutoVision concept makes it possible to reuse the data-input, intermediate-data and data-output path.

By utilizing defined interfaces between hardware and software, we can implement a hardware accelerator in parallel to the high-level application code that's intended to run on embedded CPUs (HW/SW co-design).

After we simulated and tested the HW accelerator, we set up a nonreconfigurable system containing all necessary IP cores and interconnects. We use this system to verify the correctness of the overall system before generating partial bitstreams. If everything works well, we use the PlanAhead tool to floor-plan the design and perform some design rule checks.

Finally, we use an "early access" version of the reconfiguration tools (modified MAP, place and route) to generate partial bitstreams. These we store on a CompactFlash card on our development board. From there we copy the bitstreams into main memory (DDR or DDR2 SDRAM, which also serves as video frame buffer) during the initialization phase of the system in order to have faster access to this data. When a reconfiguration process is triggered, the first bytes of the partial bitstreams appear at the ICAP input around 20 cycles later.

Since a hardware accelerator engine has to finish its job before it is reconfigured, it's important to establish a reconfiguration schedule. All the IP cores in the AutoVision system communicate via interrupts. Once a video frame has been completely transferred to the main memory, the video input notifies the embedded CPU, such as the PowerPC or MicroBlaze, by using an interrupt signal.

Suddenly the CPU sends the start signal to the HW accelerator, along with the address for where to fetch the pixel data from memory and an interrupt to the CPU. If the system receives a reconfiguration trigger during the processing of the engine, it will execute the reconfiguration right after receiving the engine interrupt. If it does not receive a reconfiguration trigger (or immediately after the reconfiguration process has started), the embedded CPU will start executing the high-level application code.

Before starting the reconfiguration itself, the CPU sends a command to dis-

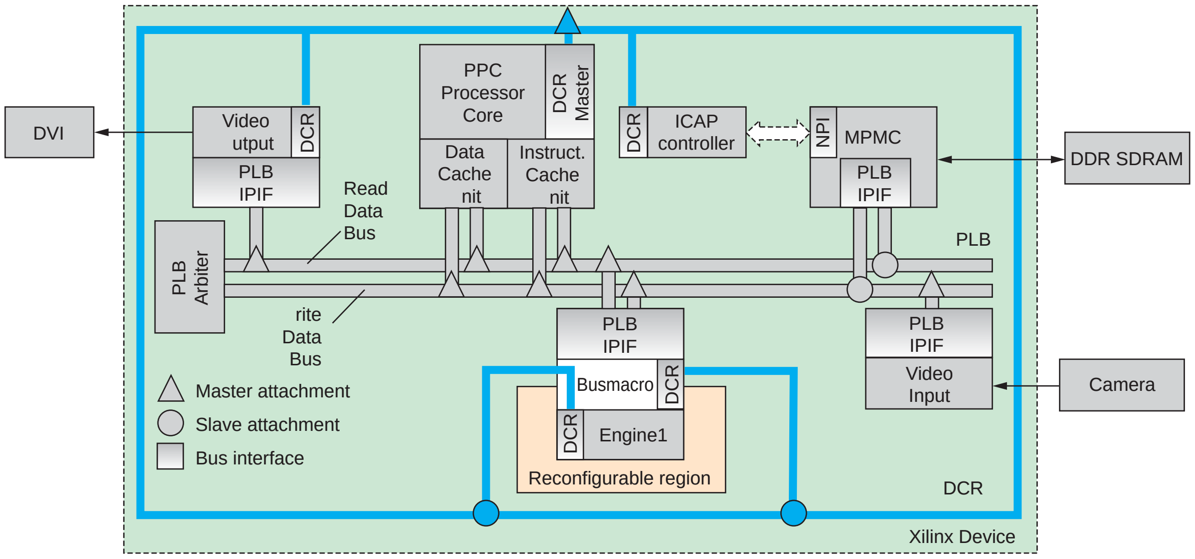


Figure 2 – Block diagram of the AutoVision system shows how the ICAP controller links to MPMC via a Native Port Interface.

connect the bus macros and the reconfigurable part from the rest of the system. The ICAP controller can initiate transfers using direct memory access and burst transfers without involving the CPU. The CPU just sends the memory address where the partial bitstream is stored along with the number of bursts to be transferred to the ICAP controller, and the reconfiguration process starts.

Because the high-level application code executes in parallel, the design does not need to allocate any additional time for the reconfiguration itself. Before a new engine is loaded, a blank bitstream clears out the partial reconfigurable region (PRR). The ICAP controller informs the CPU that the module has been removed via an interrupt and the final step can be launched.

To complete the reconfiguration, the ICAP controller fetches the partial bitstream, including the new HW accelerator, and partially reconfigures the device. Swapping two modules within one frame (IntraVFR) and clearing the reconfigurable region with a blank bitstream in each case results in four reconfigurations per video frame.

During the reconfiguration process, the PRR is held in a reset state so as to bring all the registers into a defined state. Immediately after the reconfiguration, the system disables the reset and initializes the configuration registers of the engine. Finally, the new engine can be started. All these steps can be seen in Figure 3.

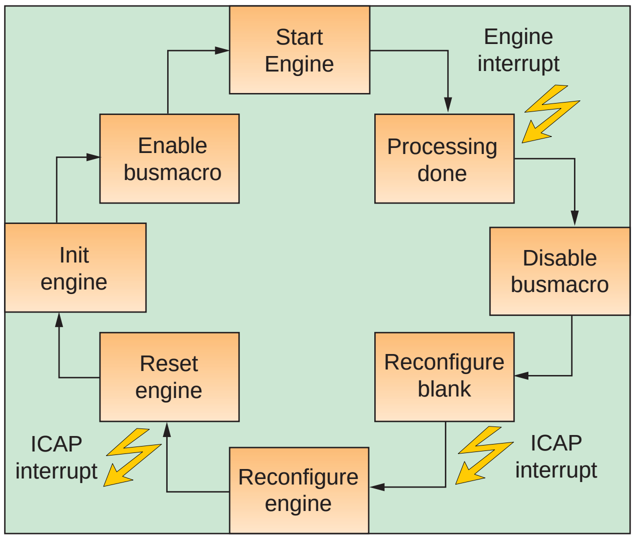


Figure 3 – The reconfiguration must take place at defined points in time.

To achieve the fastest reconfiguration times, we tried to discover the maximum frequency at which to feed the ICAP. On the Virtex-5 ML507 FPGA, we use a frequency of 300 MHz to push the configuration data into ICAP. The ICAP has an input width of 4 bytes and Xilinx specifies it up to a frequency of 100 MHz. Since we want to achieve the shortest reconfiguration times, we had to overclock the ICAP.

Our goal was to provide 4 bytes of data to the ICAP every clock cycle. Our current ICAP controller is clocked at a frequency of 300 MHz, so it requires some effort to get enough bitstream data from the main memory into one Virtex-5 FIFO primitive of our ICAP controller. Two techniques—use of a control finite state machine and connecting the ICAP controller to the MPMC via a Native Port Interface—ensure that the FIFO remains filled all the time.

Adding an extra port on the MPMC necessitates another FIFO to temporarily buffer the data. To verify that the reconfiguration was successful at nonspecified frequencies, we use an online verification approach. While the bitstream data is pushed into the ICAP, the system calculates a cyclic redundancy check (CRC) in parallel to ensure the bitstream's integrity. In addition, the ICAP produces distinct words at its output, indicating success or failure of the reconfiguration process. By merging the verification information from ICAP's output and our CRC IP, we can generate information about the reconfiguration process at run-time. We performed more than 7 million reconfigurations without observing one single configuration error.

Demo Application

We have implemented the InterVFR system on an XUPV2P board with a Virtex-II Pro device from Xilinx as a proof of concept. In this demonstrator, we considered the following scenario.

A car is driving on a highway on a sunny day. The system can detect other cars by feature points on their silhouettes. Then the car approaches a tunnel. Here it does not make sense to search for feature points on the silhouettes of other cars. It makes more sense to mark the dark tunnel entrance as a region of interest and enhance the contrast within this region. Inside the tunnel, meaningful features are the taillights of cars, which have to be separated from the tunnel lights. A more detailed description of this scenario can be found in our 2007 article describing the AutoVision system.[2]

We have implemented the current prototype for IntraVFR on a Xilinx ML507 development board as a proof of concept. You can find details on the HW accelerator for the optical flow in our 2009 paper.[1] The optical flow that we present in this article is intended for use in the detection of moving objects, such as pedestrians. It is used for illustration purposes only. Figure 4 shows sample outputs from the AutoVision system.

The HW accelerator for the optical flow consists of two separate engines. The first of these, the CensusEngine, trans-

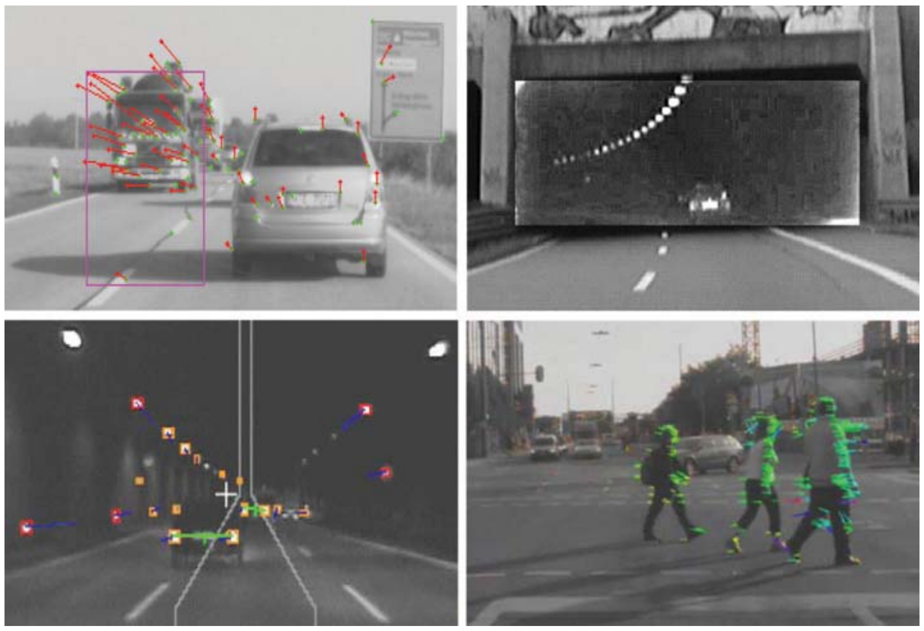


Figure 4 – Output of the AutoVision system in different environments: feature-point detection and motion segmentation on a sunny day (upper left), contrast enhancement on a tunnel entrance (upper right), taillight detection inside a tunnel (lower left) and motion detection in urban environments with the optical flow (lower right).

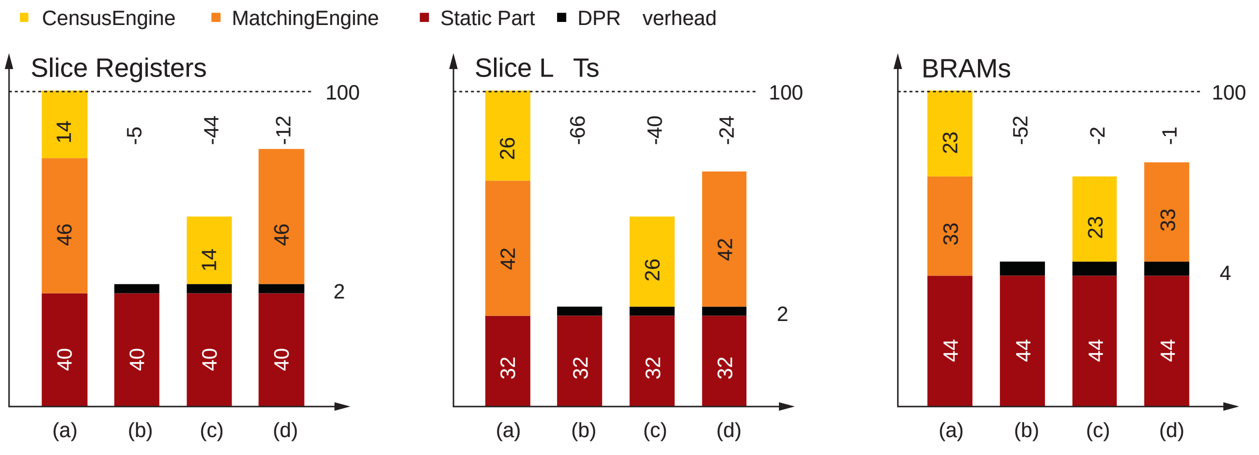


Figure 5 – Worst-case resource saving of a reconfigurable vs. a nonreconfigurable system

lates every pixel and its surrounding 15x15-pixel neighborhood in the image into a signature, which represents the luminance distribution within this neighborhood. The CensusEngine outputs an image with the same dimensions as the input image but with signatures instead of gray-scale pixel values. This is the so-called census image.

The system loads two consecutive census images into the second HW accelerator, namely the MatchingEngine. We use this engine to compare the signatures from two consecutive images to see if corresponding signatures can be found. A unique correspondence determines a movement of a pixel within a predefined (currently a 15x15-pixel) search region. If the system finds such a movement, it draws a motion vector into the output image. As these HW accelerators are running sequentially, they can be exchanged within one video frame by using IntraVFR.

Finally, we can show that Inter- as well as IntraVFR are possible by means of hardware demonstrators. In the demonstrator for IntraVFR, a frame rate of 31 fps results in a total of (31 x 4 =) 124 reconfigurations per second (rps). The demonstrators for InterVFR and IntraVFR can be combined in one single demonstrator.

Results and Resources

Figure 5 depicts the resource utilization of the IntraVFR system. We compare register, LUT and BRAM utilization of the nonreconfigurable system (a), the reconfigurable system with a blank module loaded (b), the reconfigurable system with the CensusEngine loaded (c) and the reconfigurable system with the MatchingEngine loaded (d). We consider the resource utilization of the static system including both engines as 100 percent.

As you can see, IntraVFR can save 12 percent of the registers, 24 percent of the LUTs and 19 percent of the BRAMs. This may not seem impressive at first glance, but always keep in mind that exchanging two modules is the worst case we can think of. Once we have three, four, five or more hardware accelerators, as in our AutoVision project (feature detection on the silhouette, contrast enhancement during nighttime or at tunnel entries, taillight or headlamp detection), significant reductions in terms of logical resources are possible.

In addition, video processing according to different weather conditions is imaginable, requiring new HW accelerators for mutually exclusive situations. By providing data to the ICAP with a 300-MHz frequency, we measured a throughput of bitstream data of 1.2 Gbytes/s. Reconfiguring a device with a partial bitstream with a size of 1.1 Mbytes (partial bitstream size of CensusEngine and MatchingEngine) takes 1.1 Mbytes/s to 1.2 Gbytes/s = 0.92 ms.

Future research will include additional measurements to determine if the reconfiguration will increase or decrease the power consumption compared with a nonreconfigurable system. If we perform only a few reconfigurations per hour, the impact of DPR on the overall power consumption is marginal. But since we are dealing with a rate of 124 rps, the effect on power consumption is not negligible. When no reconfiguration is performed, it's best to switch off the registers related to the reconfiguration via clock gating in order to lower the overall power consumption.

In addition, we have to perform more tests to make sure that DPR works reliably. Despite the fact that additional tests are needed in order to use DPR in an automotive environment, this technique offers great potential in maximizing the flexibility and reducing the overall costs in automotive and especially video-based driver assistance systems.

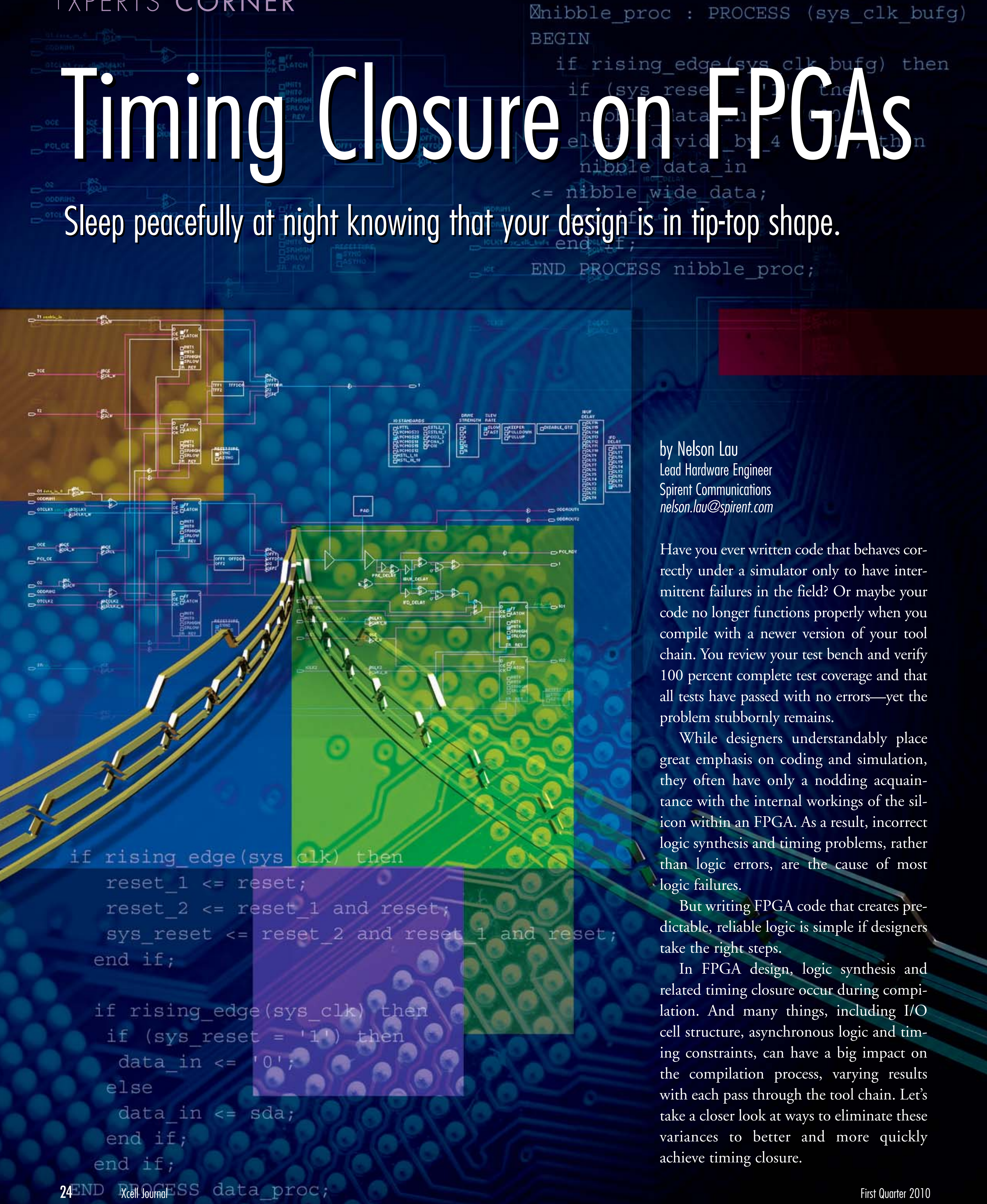
References:

[1] C. Claus, A. Laika, L. Jia and W. Stechele, "High-performance FPGA-based optical flow calculation using the census transformation," in Proceedings of IV'09, *Xi'an, China, 2009.*

[2] C. Claus, W. Stechele and A. Herkersdorf, "AutoVision: a run-time reconfigurable MPSoC architecture for future driver assistance systems," Information Technology Journal, *vol. 49, no. 3, pp. 181–187, 2007.*

Timing Closure on FPGAs

Sleep peacefully at night knowing that your design is in tip-top shape.


nibble_proc : PROCESS (sys_clk_bufg)
BEGIN
if rising_edge(sys_clk_bufg) then
nibble_data_in
<= nibble_wide_data;
end if;
END PROCESS nibble_proc;
if rising_edge(sys_clk) then
reset_1 <= reset;
reset_2 <= reset_1 and reset;
sys_reset <= reset_2 and reset_1 and reset;
end if;
if rising_edge(sys_clk) then
if (sys_reset = '1') then
data_in <= '0';
else
data_in <= sda;
end if;
end if;
END PROCESS data_proc;

by Nelson Lau
Lead Hardware Engineer
Spirent Communications
nelson.lau@spirent.com

Have you ever written code that behaves correctly under a simulator only to have intermittent failures in the field? Or maybe your code no longer functions properly when you compile with a newer version of your tool chain. You review your test bench and verify 100 percent complete test coverage and that all tests have passed with no errors—yet the problem stubbornly remains.

While designers understandably place great emphasis on coding and simulation, they often have only a nodding acquaintance with the internal workings of the silicon within an FPGA. As a result, incorrect logic synthesis and timing problems, rather than logic errors, are the cause of most logic failures.

But writing FPGA code that creates predictable, reliable logic is simple if designers take the right steps.

In FPGA design, logic synthesis and related timing closure occur during compilation. And many things, including I/O cell structure, asynchronous logic and timing constraints, can have a big impact on the compilation process, varying results with each pass through the tool chain. Let's take a closer look at ways to eliminate these variances to better and more quickly achieve timing closure.

The I/O Cell Structure

All FPGAs have I/O pins that can be highly customized. The customization affects timing, drive strength, termination and many other factors. When your I/O cell structure is not clearly defined, your tool chain will often use a default that may or may not be what you want. In the VHDL code below, the intent is to create a bidirectional I/O buffer named sda using the declaration "sda: inout std_logic;".

```
tri_state_proc : PROCESS (sys_clk)
BEGIN
    if rising_edge(sys_clk) then
      if (enable_in = '1') then
          sda <= data_in;
      else
          data_out <= sda;
          sda <= 'Z';
      end if;
    end if;
END PROCESS tri_state_proc;
```

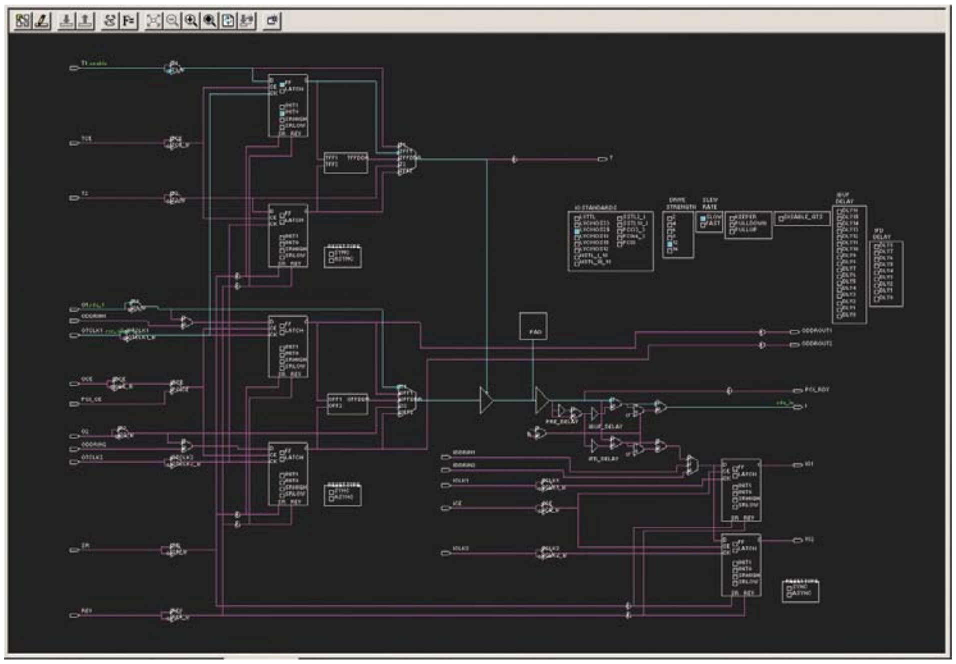


Figure 1 – The FPGA Ediitor view shows that portions of the bidirectional I/O are scattered outside the I/O buffer.

When the synthesis tool sees this block of code, there is no clear directive on how to implement the bidirectional buffer. As a result, the tool will take a best guess.

One way to accomplish the task would be to use a bidirectional buffer on the I/O ring of the FPGA (indeed, this is the desired implementation). Another option would be a tristate output buffer and input buffer, both implemented in lookup table (LUT) logic. A final possibility would be to use a tristate output buffer on the I/O ring along with an input buffer in an LUT—and this is the option that most synthesizers will choose. All three methods yield valid logic, but the last two implementations result in additional routing delays when the signal moves between the I/O pin and the LUT. They also require additional timing constraints to ensure timing closure. FPGA Editor clearly shows in Figure 1 that our bidirectional I/O has portions scattered outside the I/O buffer.

The lesson? Don't let your synthesis tool guess how to implement critical sections of your code. Even if the synthesized logic happens to be what you want, it may change when the synthesis tool goes through a new revision. Clearly define your I/O logic and any critical logic. The following VHDL code shows how to implicitly define the I/O buffer using the Xilinx® primitive IOBUF. Also note that all electrical properties of the buffer are likewise clearly defined.

```
sda_buff: IOBUF

generic map (IOSTANDARD => "LVCMOS25",
IFD_DELAY_VALUE => "0", DRIVE => 12,
SLEW => "SLOW")

port map(o=> data_out, io=> sda,
i=> data_in, t=> enable_in);
```

In Figure 2, FPGA Editor clearly shows that our bidirectional I/O has been implemented entirely within the I/O buffer.

Trials of Asynchronous Logic

Asynchronous code results in logic that is difficult to constrain, simulate and debug. Errors from asynchronous logic are often intermittent and nearly impossible to replicate. It's also not possible to generate a test bench to find errors due to asynchronous logic.

While asynchronous logic may seem easy to spot, in fact it often goes undetected, so designers must be aware of the many ways that asynchronous logic lurks in our designs. All clocked logic requires a minimum setup-and-hold time, and this also applies to the reset input of flip-flops. The code below uses an asynchronous reset. Here, there is no possible way to apply timing constraints to meet the setup-and-hold time requirements of the flip-flop.

```
data_proc : PROCESS (sys_clk,reset)
BEGIN
    if (reset = '1') then
        data_in <= '0';
    elsif rising_edge(sys_clk) then
        data_in <= serial_in;
    end if;
END PROCESS data_proc;
```

The next listing uses a synchronous reset. However, the reset signal for most systems may be a pushbutton switch or some other source that is not related to the system clock. Although reset is mostly static, and asserted or deasserted for long periods, there is still a change in level. It is the

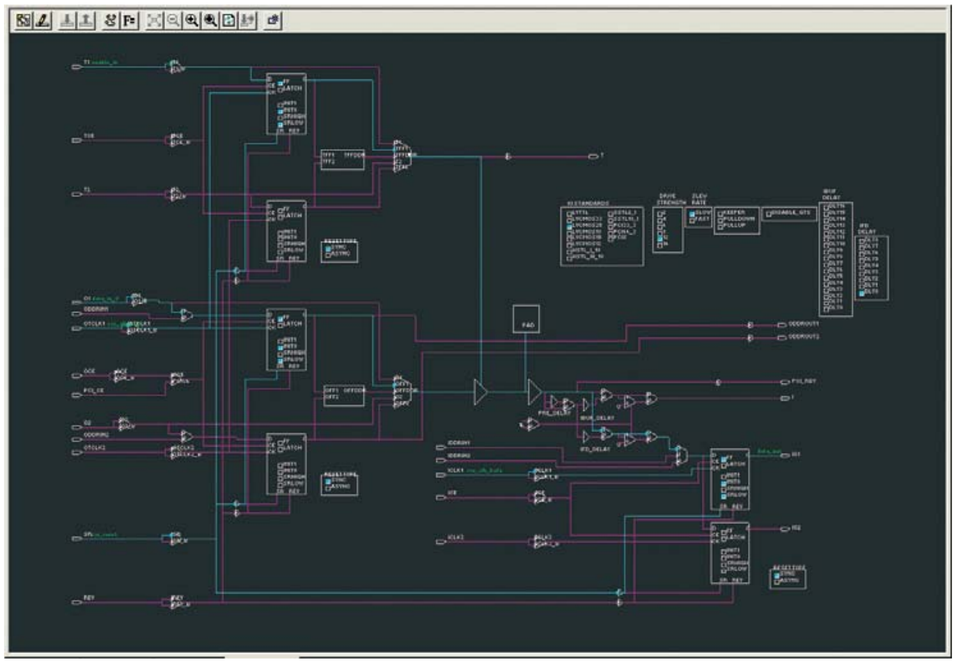


Figure 2 – With a VHDL code change to clearly define I/O logic and critical logic, our bidirectional I/O is implemented entirely within the I/O buffer.

deassertion of reset, relative to the rising edge of the system clock, that can violate the setup-time requirements of a flip-flop, and there is no way to constrain this.

```
data_proc : PROCESS (sys_clk)
BEGIN
    if rising_edge(sys_clk) then
      if (reset = '1') then
        data_in <= '0';
      else
        data_in <= serial_in;
      end if;
    end if;
  END PROCESS data_proc;
```

Once we realize that we can't directly feed an asynchronous signal into our synchronous logic, the problem becomes easy to fix. The code below creates a new reset called sys_reset that has been synchronized to our system clock sys_clk. When sampling asynchronous logic, metastability issues can arise. We can reduce the chance of its occurrence by using a laddered sample that is ANDed with the previous stages of the ladder.

```
data_proc : PROCESS (sys_clk)
BEGIN
    if rising_edge(sys_clk) then
      reset_1 <= reset;
      reset_2 <= reset_1 and reset;
      sys_reset <= reset_2 and reset_1
      and reset;
    end if;

    if rising_edge(sys_clk) then
      if (sys_reset = '1') then
        data_in <= '0';
      else
        data_in <= serial_in;
      end if;
    end if;
END PROCESS data_proc;
```

So, let's assume you've taken care to make all your logic synchronous. Nevertheless, if you're not careful, your logic can easily become decoupled from the system clock. Don't let your tool chain use local routing resources for your system clock. Doing so will make your logic impossible to constrain. Remember to clearly define all your important logic.

The VHDL code below uses the Xilinx primitive BUFG to force sys_clk onto a dedicated high-fan-out buffer that drives low-skew nets.

```
gclk1:  BUFG port map (I => sys_clk,O
=> sys_clk_bufg);

data_proc : PROCESS (sys_clk_bufg)
BEGIN
    if rising_edge(sys_clk_bufg) then
      reset_1 <= reset;
      reset_2 <= reset_1 and reset;
      sys_reset <= reset_2 and reset_1
      and reset;
    end if;

    if rising_edge(sys_clk_bufg) then
      if (sys_reset = '1') then
        data_in <= '0';
      else
        data_in <= serial_in;
      end if;
    end if;
END PROCESS data_proc;
```

Some designs use a divided version of their single master clock to process deserialized data. The VHDL code below, process nibble_proc, shows an example of data being captured at one-quarter of the system clock rate.

```
data_proc : PROCESS (sys_clk_bufg)
BEGIN
    if rising_edge(sys_clk_bufg) then
      reset_1 <= reset;
      reset_2 <= reset_1 and reset;
      sys_reset <= reset_2 and reset_1
      and reset;
    end if;

    if rising_edge(sys_clk_bufg) then
      if (sys_reset = '1') then
        two_bit_counter <= "00";
        divide_by_4 <= '0';
        nibble_wide_data <= "0000";
      else
        two_bit_counter
        <= two_bit_counter + 1;
divide_by_4 <= two_bit_counter(0) and
two_bit_counter(1);
        nibble_wide_data(0)
        <= serial_in;
        nibble_wide_data(1)
      <= nibble_wide_data(0);
      nibble_wide_data(2)
      <= nibble_wide_data(1);
      nibble_wide_data(3)
      <= nibble_wide_data(2);
    end if;
  end if;
END PROCESS data_proc;

nibble_proc : PROCESS (divide_by_4)
BEGIN
  if rising_edge(divide_by_4) then
    if (sys_reset = '1') then
      nibble_data_in <= "0000";
    else
      nibble_data_in
      <= nibble_wide_data;
    end if;
  end if;
END PROCESS nibble_proc;
```

It looks like everything is synchronous, but the nibble_proc uses a product term divide_by_4 to sample nibble_wide_data from clock domain sys_clk_bufg. Due to routing delays, there is no well-defined phase relationship between divde_by_4 and sys_clk_bufg. Moving divide_by_4 onto a BUFG will not help either, as the process incurs a routing delay. The solution is to keep nibble_proc on the sys_clk_bufg domain and use divide_by_4 as a qualifier, as shown below.

```
nibble_proc : PROCESS (sys_clk_bufg)
BEGIN
  if rising_edge(sys_clk_bufg) then
    if (sys_reset = '1') then
      nibble_data_in <= "0000";
    elsif (divide_by_4 = '1') then
      nibble_data_in
      <= nibble_wide_data;
    end if;
  end if;
END PROCESS nibble_proc;
```

Importance of Timing Constraints

Applying the proper timing constraints is a necessity if you want your logic to perform properly. If you've taken care to ensure that 100 percent of your code is synchronous and all I/Os are registered, those steps will greatly simplify timing closure. Using the above code and assuming that the system clock is 100 MHz, the timing constraint file is easily done in four lines, as shown below.

```
NET sys_clk_bufg TNM_NET =
sys_clk_bufg;
TIMESPEC TS_sys_clk_bufg = PERIOD
sys_clk_bufg 10 ns HIGH 50%;

OFFSET = IN 6 ns BEFORE sys_clk;
OFFSET = OUT 6 ns AFTER sys_clk;
```

Note that setup-and-hold times for I/O registered logic on Xilinx FPGAs are pretty much fixed and don't change much within a package. But we still apply them, mainly as a verification step to ensure that the design meets its system parameters.

Three Easy Steps

Designers will find that it's not hard to implement reliable code if they follow three simple steps.

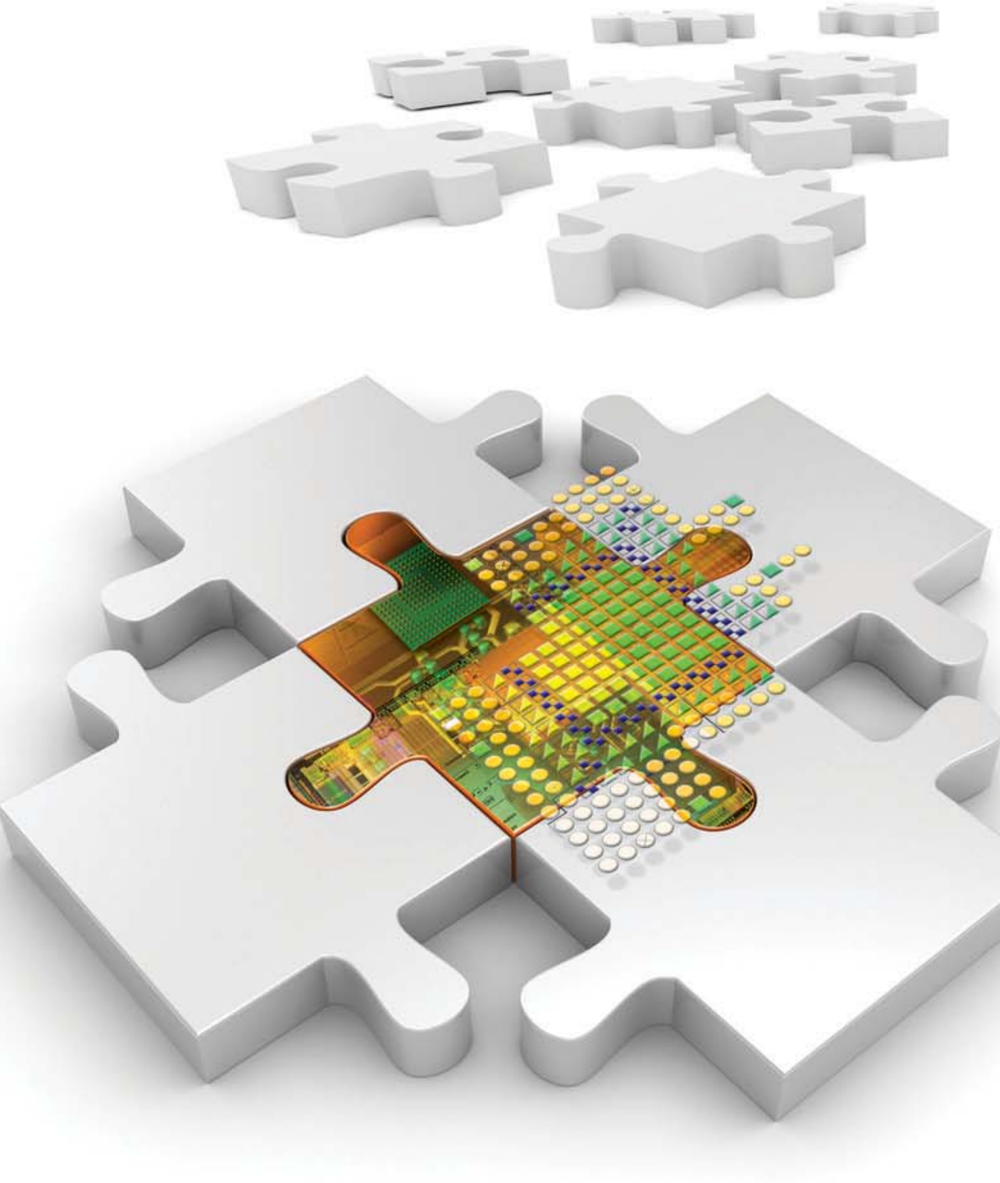
- Don't let your synthesis tool guess at what you want. Use Xilinx primitives to clearly define all I/O pins and critical logic. Be sure to define the electrical properties of your I/O pins.
- Make your logic 100 percent synchronous and reference all logic to your master clock domain.
- Apply timing constraints to ensure timing closure.

If you follow these three steps, you will have removed variances due to synthesis and timing. Abolishing those two significant obstacles will give you code that works with 100 percent reliability.

In a follow-up to this article in an upcoming issue, Nelson Lau will explore methods to mitigate flow control between unrelated clock domains, the effects of which become pronounced when the unrelated clocks have a large difference in operating frequency.

The Simple MicroBlaze Microcontroller Concept

It's easy to add this preconfigured controller to any FPGA design. Better yet, you won't need special tools or complicated scripts.



by Christophe Charpentier
Processor Specialist FAE
Xilinx, Inc.
christophe.charpentier@xilinx.com

Embedded microcontrollers are commonplace for a wide range of applications with various degrees of complexity. Xilinx has been offering hard (PowerPC® 405 and PowerPC 440) and fabric-based (MicroBlaze™) embedded microprocessors since 2000. MicroBlaze has the great advantage of being able to service complex applications, in some cases running an operating system, as well as simple general-purpose applications.

Designers can implement the MicroBlaze soft processor in all the current Xilinx architectures, providing easy transitions from family to family as well as unparalleled flexibility. However, with more than 70 parameters to choose from and a powerful set of embedded tools, designing a MicroBlaze system might prove unwieldy when your application requirements call for only a simple microcontroller.

But with the right techniques, you can build a simple, preconfigured MicroBlaze microcontroller, and easily and quickly add it to any FPGA design. The controller is instantiated directly into the HDL. You can use it immediately in a standard FPGA design flow without special scripts or complicated steps. Only three files are necessary to get started—two hardware implementation files and one software definition file. This methodology thus enables engineers to get started with FPGA embedded designs with little to no learning curve.

	Spartan-3 Family			Virtex-5 Family			Spartan-6 Family			Virtex-6 Family		
Configuration	LUTs	FFs	Slices	LUTs	FFs	Slices	LUTs	FFs	Slices	LUTs	FFs	Slices
SMM + UART + Debug	1770	1240	1120	1020	1230	460	1350	1200	470	1350	1200	520
SMM + UART	1510	900	1090	830	880	360	1060	860	320	1050	870	350
SMM + Debug	1390	790	820	820	780	330	990	750	410	990	750	430
SMM	1130	450	630	630	440	220	690	440	220	690	440	250

Table 1 – The size of the simple MicroBlaze microcontroller (SMM) will vary depending on FPGA family.

Starting in the ISE® 11.1, MicroBlaze software development includes a standalone Software Development Kit (SDK) that will let you create and debug C and C++ applications without the need for a full Embedded Development Kit (EDK).

The microcontroller comes preconfigured with two options, UART and debug. Table 1 shows the size estimate for various FPGA families based on the microcontroller configuration. Additionally, Virtex® devices use two Block RAMs and Spartan® devices use four Block RAMs. Once you have debugged the application code, you can remove the debug option to reduce the size of the controller. For example a Spartan-6 microcontroller would require only 220 slices.

Microcontroller Overview

The simple MicroBlaze microcontroller consists of a 32-bit MicroBlaze processor, 8 kbytes of RAM/ROM, a 32-bit user interface with 64 kbytes of addressing space, interrupt support, an optional UART and an optional JTAG debug interface. Figure 1 shows the system block diagram.

The clock input can be as low as desired and as high as the implementation tools will allow. The active high reset input is internally synchronized with the input clock. An interrupt input signal—acknowledged using the interrupt-acknowledge output when serviced by the microcontroller—provides interrupt support. A simple address-mapped user interface, also synchronous to the clock, enables user customization. Figure 2 shows the timing for the user interface. Byte enables are used for byte and half-word transactions.

You can decode the 16-bit-wide software-mapped address bus to connect various custom interfaces or peripherals to the microcontroller. The read data is sampled two clock cycles after you assert Chip Select.

Some of the preconfigured versions provide an option for a serial 16450 UART. The baud rate is programmed in software to keep the UART independent from the clock input. The debug option uses internal FPGA resources and connects directly to the FPGA JTAG interface, enabling application debug through the regular FPGA download cable.

FPGA Design Flow

The FPGA design flow follows the standard ISE FPGA implementation flow as shown in Figure 3. You can instantiate the microcontroller in Verilog or VHDL at any level of hierarchy within the FPGA design. Use the two hardware-related files—the microcontroller netlist (smm.ngc) and the Block RAM memory map file (smm.bmm)—to complete the implementation of the FPGA. You won't need to learn new tools or use complex, scripted flows. Doing an FPGA embedded design has never been this easy. Switching from one microcontroller configuration to another is as simple as replacing a netlist file with the desired one and reimplementing the FPGA.

After running the implementation tools, an additional file will be created that will indicate the physical location of the Block RAMs the microcontroller uses (smm_bd.bmm).

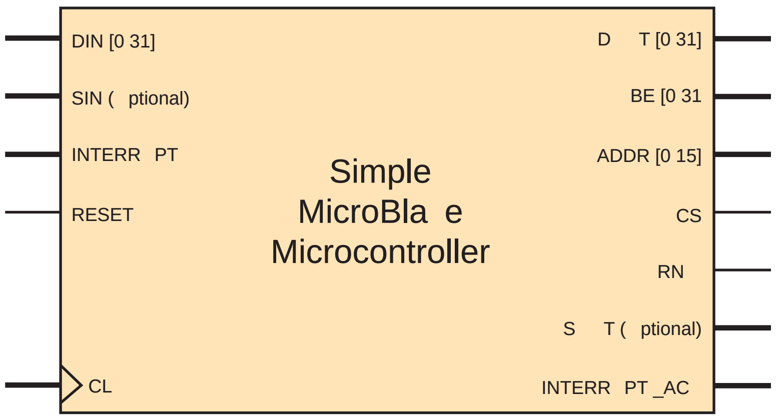


Figure 1 – The SMM consists of MicroBlaze processor, memory and interfaces.

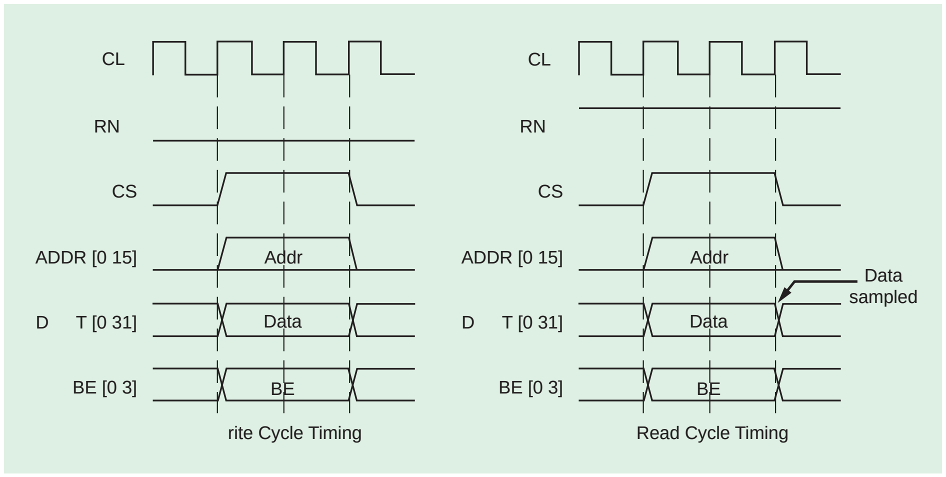


Figure 2 – A simple address-mapped user interface is synchronous to the clock.

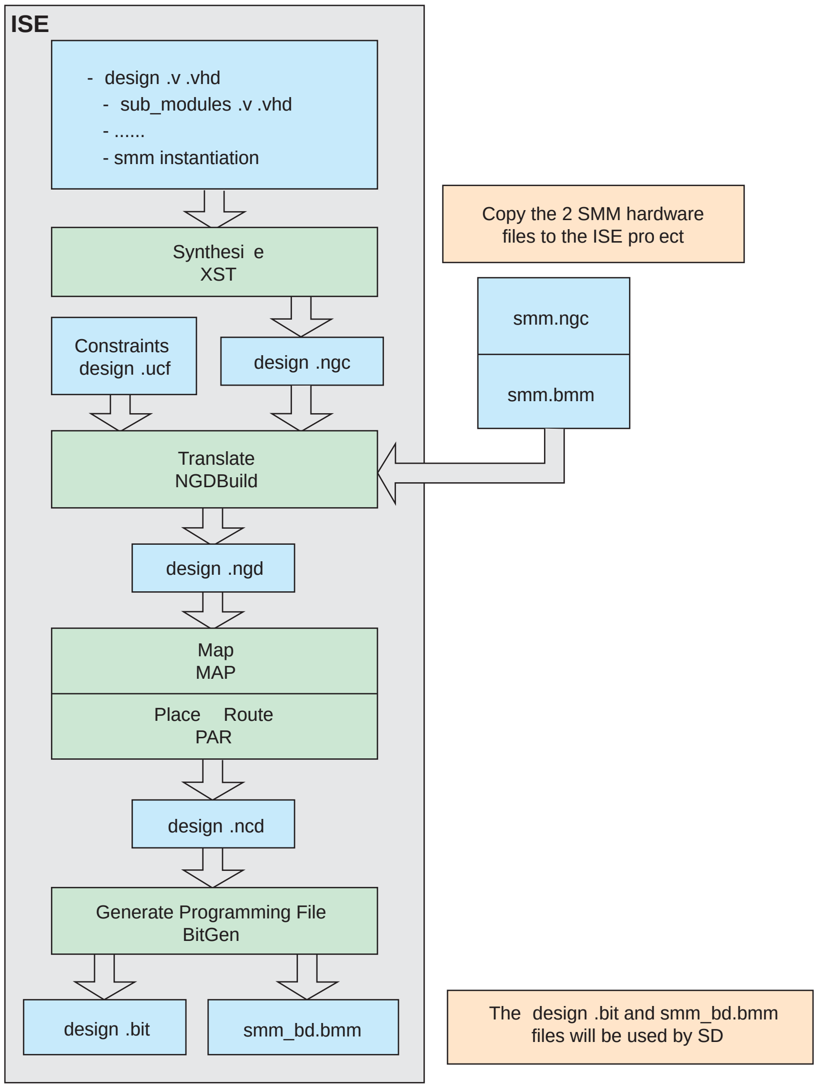


Figure 3 – The FPGA design flow follows the standard ISE FPGA implementation flow, with no need for new tools or scripts.

Software Application Design Flow

A single software description file (smm.xml) contains all the information necessary to start developing a microcontroller application. The development can start independently from the FPGA design flow and could even start before any FPGA design implementation.

Starting in ISE 11.1, the SDK is available as a standalone option. It contains all the tools, drivers, libraries and utilities you will need to complete a software application.

Figure 4 shows the standard SDK development flow starting with the software definition file. The address space for the microcontroller includes the 8 kbytes of RAM, the user interface and the UART register space when the UART option is selected.

You can write the software application in C or C++ and store it inside the microcontroller Block RAMs. The RAM space also acts as a ROM for the microcontroller, since it can be preloaded with the application within the FPGA bitstream. After you have configured the FPGA and deasserted the microcontroller reset, the application will simply start executing from the internal RAM/ROM space. The microcontroller is completely self-contained.

The debug option enables full source-level debug of the application, providing visibility into the memory, registers and variables. You can set breakpoints within the application to facilitate debugging. Designers will debug with the same cable that's used to configure the FPGA. Once finished, you can remove the debug option to reduce the size of the controller.

Example Design

Let's take a look at an LCD controller reference design for a couple of Xilinx development boards that pulls in various features of the simple MicroBlaze microcontroller. LCD controllers are well suited for a small microcontroller implementation due to their slow and simple hardware interface, long initialization sequences and large number of character codes.

Through a combination of HDL and C code, the design outputs messages to the character LCD screen on the board.

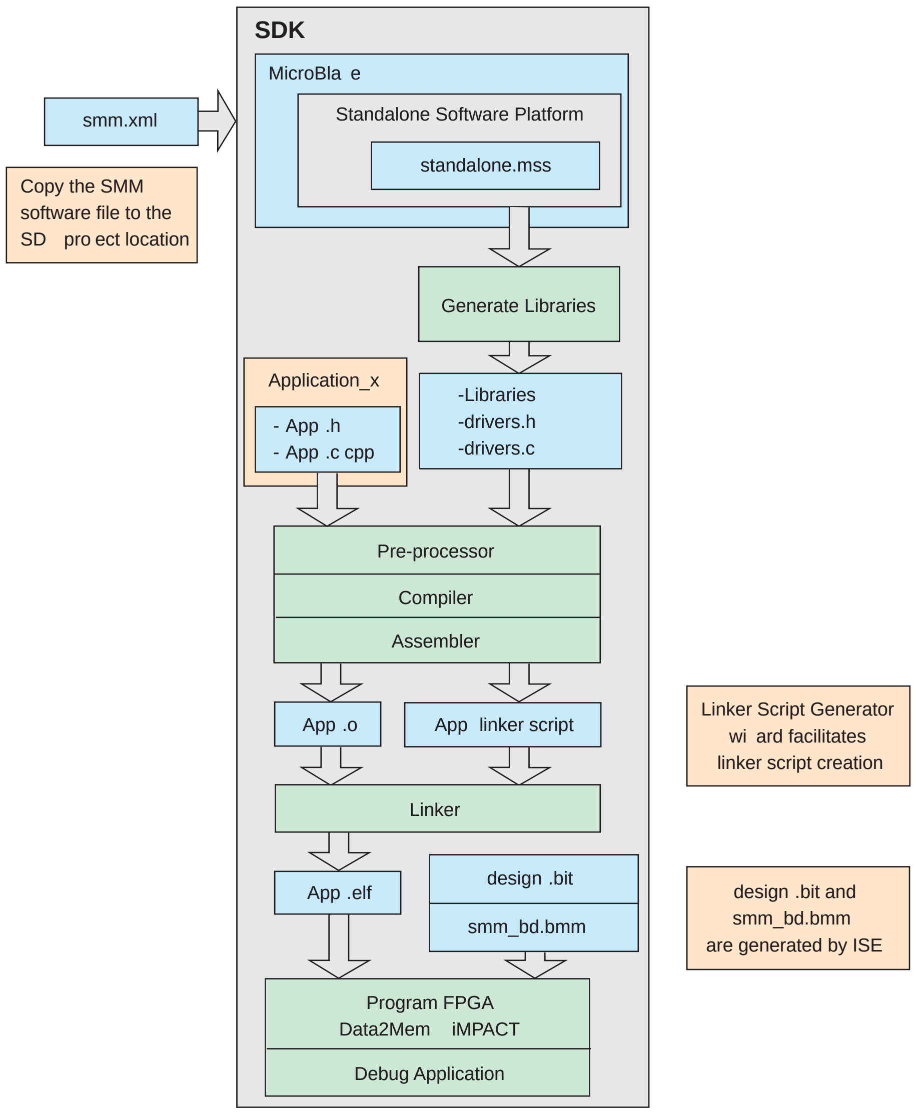


Figure 4 – The SDK development flow starts with the software definition file.

The HDL handles the hardware interface while the software initializes and controls the LCD screen.

The timing for the LCD module is slow, but at the same time requires large delays between instructions or data. For example, the instruction to clear the display requires a delay of 1.52 milliseconds before the next instruction or data can be issued. Some instructions require a delay of 40 µs, others 1 µs.

While we could use a while loop in the C code to handle the delay, it would be too inaccurate and at the mercy of compiler optimizations. A better option is to create a software-loadable 32-bit counter in the FPGA that will trigger an interrupt to the controller when it reaches the programmed delay. A MicroBlaze write to address 0x10 starts the timer based on the data that's present on the user interface data bus. The MicroBlaze will then wait for an interrupt to continue execution.

A MicroBlaze write to the user interface address 0x0 triggers the LCD controller hardware interface, the timing of which is handled in HDL. The user interface data bus captures the instruction or data value. A pushbutton input connects to the user interface at address 0x20.

The FPGA design consists of a top-level module, an LCD hardware timing module and a software-addressable programmable timer. The top-level file also contains the simple MicroBlaze microcontroller instantiation running at 66 MHz.

The C application is contained in a single file. The code enables the MicroBlaze interrupt, initializes the LCD screen, manages the different delays, prints two lines to the LCD, waits for a pushbutton input, clears the screen and outputs a new message.

On the Virtex-6 ML605 board, the design consumes less than 1 percent of FPGA resources and provides a more efficient way of partitioning a design. Compiling and debugging C code is an order of magnitude faster than implementing and debugging an HDL-only implementation.

Customizing the Microcontroller

Since the microcontroller is built using MicroBlaze, the designer will have access to many standard peripherals and options to customize the embedded system. A user might want to employ a different FPGA architecture or add more main memory, a floating-point unit or a standard SPI or I^2C peripheral.

To customize the provided systems would require the EDK. It contains many varied configurations as embedded projects, which you can modify based on user requirements. For example, if you needed 16 kbytes of memory instead of the standard 8 kbytes, you would open the EDK project, modify the MicroBlaze RAM space and generate the new netlist, Block RAM memory and software description files. You would then add the new files into the ISE and SDK projects.

While a simple MicroBlaze microcontroller won't be the answer for all possible embedded designs, it does address the needs of users requiring a simple microcontroller to efficiently provide control functions. It also offers a concept for teams wanting to share and distribute EDK designs. Regardless of the size of the embedded design, it takes only three files to implement the whole thing.

For more details on the simple MicroBlaze controller concept, see Xilinx application note XAPP1141, at *http://www.xilinx.com/support/documentation/application_notes/xapp1141.pdf.*

FPGAs Hold Key to Greener Search

FPGA-accelerated information retrieval may be the fastest route to energy efficiency in the data center.

by Wim Vanderbauwhede
Project Co-Investigator
University of Glasgow
wim@dcs.gla.ac.uk

Leif Azzopardi
Project Co-Investigator
University of Glasgow
leif@dcs.gla.ac.uk

Mahmoud Moadeli
Research Assistant
University of Glasgow
mahmoudm@dcs.gla.ac.uk

Servicing millions of user search requests and processing very large volumes of information takes massive computational resources that consume huge amounts of energy. Indeed, energy costs for computing and cooling have become the dominant cost in the operation of data centers.[1] With data centers growing in numbers and size, projections show that at the current levels of energy consumption, the CO_2 emission could overtake that of the airline industry by 2020.[2] This fact is spurring the development of energy-efficient solutions for processing large amounts of data. The greening of the data center is a win-win situation—service providers can significantly reduce their operating costs while also minimizing the impact upon the environment.

FPGAs have huge potential for accelerating common data center tasks such as Web search and similar information retrieval because of their inherent parallelism and low power consumption. Recognizing this potential, the Austrian company Matrixware had acquired an FPGA platform but lacked the internal know-how to implement a complex information retrieval application. The company commissioned our team from the Department of Computing Science at the University of Glasgow to develop a proof of concept for an FPGA-accelerated patent-search solution. The team—which consisted of the three authors and part-time research assistant Stelios Papanastasious—blended expertise in information retrieval, FPGAs and system development to form the necessary skill set to develop a prototype application. After discussion, we agreed on a real-time patent-filtering application with an FPGA-accelerated back end.

The resources for the project were limited in terms of manpower and time. For that reason, implementing the filtering algorithm in an HDL was not feasible, so we decided on a high-level programming solution developed by the Swedish company Mitrionics.

The prototype application drew much interest among patent searchers at the Information Retrieval Facility Symposium, held in Vienna, Austria, last November. Processing millions of patents usually takes a few minutes, but with the FPGA-accelerated back end, the results flooded back in seconds.

We published the results and described subsequent performance improvements at the ACM SIGIR International Conference on Information Retrieval Research and Development in July 2009 [3] and presented the detailed design of the architecture at FPL 2009, the international Conference on Field Programmable Logic, last September.[4]

Ins and Outs of Document Filtering

In general, the task of information filtering consists of matching incoming documents against a given set of information needs, or profiles.[5] This task can be performed for a number of reasons in a variety of situations, for example detecting spam in incoming e-mails, comparing patent applications against existing patents, monitoring communications for terrorist activity, detecting and tracking news story topics and so on. When faced with large volumes of incoming documents, processing needs to happen in real time, so time-based efficiency is paramount. Therefore, our aim was to filter documents efficiently in terms of both time and energy by implementing the most computationally intensive part of the filtering application on FPGAs.

In this work, we employed the relevance model proposed by Lavrenko and Croft.[6] Applied to the task of information filtering, the idea is to determine the odds of an incoming document being relevant to the topic profile using a generative probabilistic language model. If a document scores above a user-defined threshold, then it is considered relevant to the topic profile.

The algorithm implemented on the FPGA can be expressed as follows: a document is modeled as a "bag of words"—that is, a set D of pairs (t,f) where f=n(t,d) is the number of occurrences of the term t in the document d. The profile M is a set of pairs p=(t,w) where the weight is given by

$$w = \log\left(\frac{(1-\lambda)P(t \mid M)}{P(t)} + \lambda\right)$$

The score of a given document against a given profile is given by

$$score(D,M) = \sum_{\forall k \in T} f_k w_k$$

where T is the set of terms occurring in both D and M. This function is representative of the kernel of most filtering algorithms, the main difference being the weighting of terms in profiles.

The Application Architecture

The document-filtering application has a client-server architecture, consisting of a GUI-based client connected over TCP/IP to a communication server that acts as a proxy between the various back-end servers and the client (see Figure 1). A typical use case starts with the user issuing a query to the query server, a conventional search system that returns a sorted list of hits. The user then creates a profile by selecting relevant documents from that list. Next, the profile server uses the complete text of all the combined documents to construct the profile (the list of terms and weights). The profile server matches this profile against the complete document collection and returns a stream of scores to the client.

The modular client-server architecture facilitates benchmarking of the system, since it is easy to add a C++ reference implementation of the profile server running on the host CPU. As shown in Figure 1, the FPGA-accelerated part of the application is limited to the most computationally intensive task, the matching of the documents against the profile. The host system handles all other tasks (Figure 2).

The profile server filters a stream of documents against a profile received from a client and returns a stream of scores. To evaluate the performance, we created both a C++ reference implementation and an FPGA-accelerated implementation. Both versions have the same basic functionality—they receive the list of documents constituting the profile over a TCP/IP interface, construct the profile using a relevance model and score a memory-buffered document stream against this profile, returning a stream of document scores to the client over TCP/IP. The document stream is buffered in memory; otherwise, the slow disk access would limit the performance of the application.

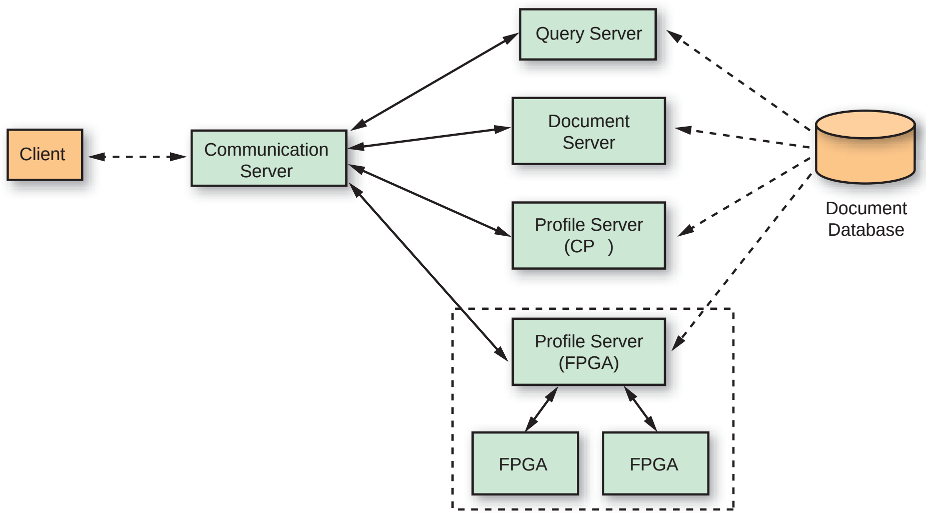


Figure 1 – System architecture centers on a communication server that acts as a proxy between client and back-end servers.

We implemented the application on an SGI Altix 4700 machine that hosts two RC100 blades. Each blade contains two Xilinx® Virtex®-4 LX200 FPGAs running at 100 MHz; each FPGA is connected to the host platform via the SGI NUMAlink high-speed I/O interface and has access to a local 64-Mbyte SRAM bank over a 128-bit data bus at a maximum speed of 16 Gbytes/second. The host system is an 80-core, 64-bit NUMA machine running 64-bit Linux (OpenSuSE). The processors are dual-core Itanium-2 devices running at 1.6 GHz; each processor has direct access to 4 Gbytes of memory, but can access the complete 320-Gbyte memory space over the NUMAlink. It is notable that the Itanium

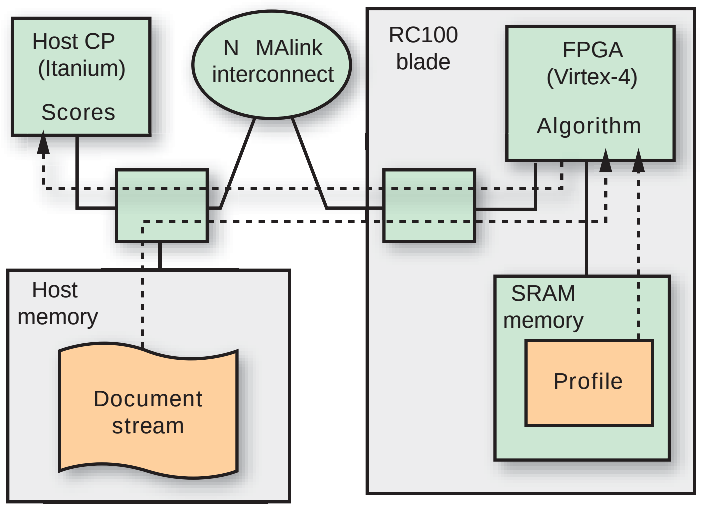


Figure 2 – In the FPGA subsystem architecture, Virtex-4 devices connect to the host platform via SGI's NUMAlink interface.

processor consumes approximately 130 watts,[7] whereas each Virtex-4 FPGA consumes only about 1.25 W.[8]

We implemented the application in C++ using the Lemur information retrieval (IR) framework and the SGI Reconfigurable Application-Specific Computing (RASC) libraries for interacting with the FPGA. The Lemur Toolkit (see *www.lemurproject.org*) is an open-source set of tools designed for research in IR, with support for indexing and various relevancy and retrieval models. The RASC library is SGI's proprietary solution to integrating FPGAs with host systems over the high-performance NUMAlink interconnect fabric. It defines a hardware abstraction API that provides control over each hardware element in the system.

We used the Mitrionics software development kit (SDK) to convert the domain-specific Mitrion-C language into VHDL. The generated VHDL can now be simply targeted to an FPGA device architecture. We employed the Xilinx ISE® tool chain with the XST synthesis tool to create the bitstream for the Virtex-4.

High-Level FPGA Programming

The Mitrionics SDK provides Mitrion-C as a high-level language specifically intended for the rapid development of applications on FPGAs. However, the suffix "C" is a bit misleading. Although the language has a C-style syntax, it is in fact a single-assignment dataflow language following a functional programming style. Mitrion-C has native support for wide (vector) and deep (pipeline) parallelism. As such it is well suited for algorithms that process streams of data, such as filtering and many other types of text- and data-mining algorithms.

Mitrion-C also provides a stream data type that results in pipelined operation when used in conjunction with the foreach looping construct; a vector data type for data-parallel operation; and a list data type for sequential lists. In particular, you can filter the output of a foreach loop over a stream to produce a smaller stream, as seen in the example Mitrion-C code below. Furthermore, programmers can create powerful data types using the tuple construct. A final feature worth noting is the language's support for variable-width integer and floating-point numbers.

```
type Word = typedef bits:128;
type Stream = typedef Word(..);
type DocId = typedef uint:64;
type Score = typedef int:48;
type DocScore = typedef tuple
{DocId,Score};
type ScoreStream = typedef
DocScore(..);

ScoreStream score_stream =
foreach ( Word w in Stream
word_stream ) {
    score = ... ; // calculate score
     keep = score > limit;
} keep ? score
```

To implement the scoring operation efficiently on the FPGA, the key issues we had to address were efficient lookup of the profile and efficient streaming I/O of the document stream.

For every term in the document, the application needs to look up the corresponding profile term to obtain the term weight. Since most of the lookups will fail (that is, most terms in most documents will not occur in the profile), it is important to discard the negatives first. For that reason we implemented a Bloom filter [9] in the FPGA Block RAM. The higher internal bandwidth of the BRAMs leads to speedy rejection of negatives. Because of the need for lookup, the profile must be implemented as some type of hash function. However, as the size of the profile is not known in advance, it is impossible to construct a perfect hash; imperfect hashes suffer collisions, which degrade the performance.

To solve this problem, we opted for a binning approach, partitioning the external SRAM into bins, each of which can contain a fixed number of profile terms. The bin size determines the number of collisions that can be handled. To assign a profile term to a bin we simply take the lower part of the term ID as the memory address; thus, there is no actual hashing.

Let the SRAM memory capacity be NM profile terms. The term ID is an unsigned integer with a range depending on the vocabulary size, which in our case is about 4 million terms, requiring 24 bits. The term weight is represented as an 8.32 fixed-point number, so the profile term takes 64 bits. The SRAM on the RC100 consists of four banks of 16 Mbytes, hence $N_M=2^{23}$. The number of bins $n_b=N_M/b$ and the bin address are computed from the term ID t as $(t\&(n_b-1)).b$.

The probability x of occupancy for a bin is given by the combinations, with replacement given the number of bins n_b

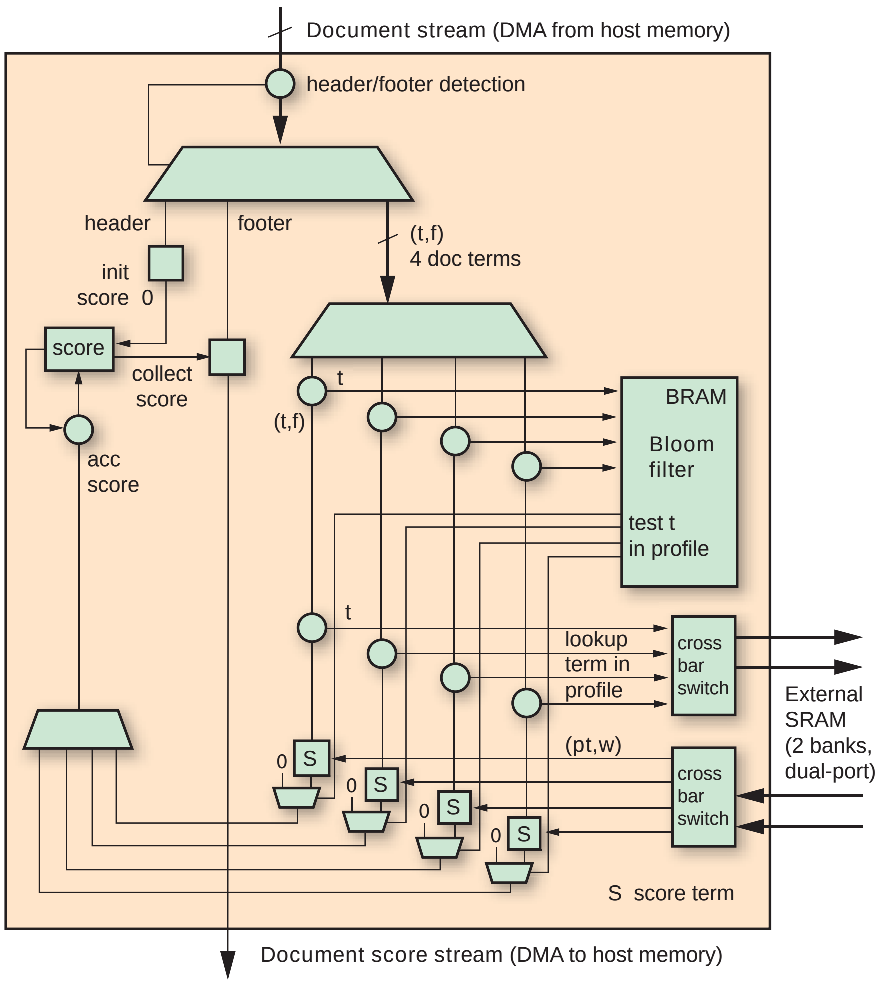


Figure 3 – Diagram of FPGA implementation of a filtering application

and the number of terms in the profile n_p. Thus we can compute the probability of bin overflow as a function of the bin size (and hence the number of bins) as $NM=b.n_b$. The larger the bin size, the slower the lookup; however, as the SRAM bank consists of four independent 64-bit addressable dual-port SRAMs, we can actually look up four profile terms in parallel. Consequently, the relative performance decreases as 1/ceil(b/4). Our analysis showed that even for the largest profiles (16K; in our study the largest profile was 12K, but in general profiles are much smaller), the bin overflow probability for b=4 (best performance) is 10^{-9}. In other words, the chance that a profile term will be discarded is less than one in a billion. It should be noted that this estimate is pessimistic, because we assume that the vocabulary size is infinite.

Using a bag-of-words representation for the document, the document stream is a list of pairs (document ID, document term pair set). Physically, the FPGA accepts a stream of 128-bit words from the NUMAlink at 1.6 Gbytes/s. Consequently, the document stream must be encoded onto this word stream. The document term pair di $=(t_i,f_i)$ can be encoded in 32 bits: 24 bits for the term ID (supporting a vocabulary of 16 million terms) and 8 bits for the term frequency. Thus, we can combine four pairs into a 128-bit word. To mark the start and end of a document, we insert header and footer words that contain the document ID (64 bits) and a marker (64 bits).

Using the lookup-table architecture and document stream format as described above, the actual lookup and scoring system (Figure 3) is quite straightforward. All that's needed is to scan the input stream for header and footer words. The header word sets the document score to 0; the footer word collects and outputs the document score. For every four terms in the document, first the Bloom filter is used to discard negatives and then four profile terms are read from the SRAM. The score is computed for each of these terms in parallel and added (Figure 4). In practice, three out of four profile term IDs will not match the document term ID; only for the fourth is the actual product computed. The score is accumulated for all terms in the document, and finally the score stream is filtered against a limit before being output to the host memory.

The host-FPGA interface transfers the document stream from the memory buffer to the FPGA and returns a score stream to the client. On receipt of a list of profile document IDs from the client, the parent process forks off a child process that builds the actual profile, loads it onto the SRAM and runs the algorithm on the FPGA. Each child process spawns a separate output thread that buffers the scores received from the FGPA and transmits them to the client over TCP/IP, thus using the network for multiplexing the score streams. Without this thread, fluctuations in the network throughput could degrade the system performance. The main advantage of this host interface architecture is that it can easily scale to large numbers of FPGAs.

Order-of-Magnitude Speedup

To evaluate the performance of the FPGA-accelerated filtering application, we performed a series of experiments to compare the FPGA-based implementation against an optimized reference implementation written in C++ and run on the Altix. For the comparison, we used three IR test collections (see Table 1): TREC Aquaint, a

Collection	# Docs	Avg. Doc. Length	Avg. Unique Terms
Aquaint	1,033,461	437	169
USPTO	1,406,200	1,718	353
EPO	989,507	3,863	705

Table 1– Collection statistics

benchmarking reference collection provided by the Text Retrieval Conference (TREC), along with two collections of patents, from the U.S. Patent and Trademark Office (USPTO) and the European Patent Office (EPO), respectively. We chose these collections to assess the impact of different document lengths and sizes of documents on filtering time.

To simulate a number of different filters, we constructed profiles for each collection by selecting a random document, using the title as the query and then selecting a fixed number of top documents returned by the query server as pseudo-relevant. We then used the returned documents to construct a relevance model that defined the profiles against which each document in the collection was matched (as if it were being streamed from the network). The number of documents in the profile varied from 1 to 50, to determine the impact on performance as the size of the profiles increased (both in number of terms and number of documents). We repeated this process 30 times and computed the average processing times.

We have summarized the results in Table 2 and Figure 5. From the table, it is clear that the FPGA implementation is typically an order of magnitude faster than the standard implementation. From the figure, it can be seen that as the profile size (the number of terms that require matching) increases, the standard implementation becomes slower and slower, while the FPGA implementation remains relatively constant. This is because the FPGA implementation pipelines the profile scoring, resulting (in first order) in a constant latency independent of the profile size.

The results clearly demonstrate the potential of FPGAs for accelerating IR tasks. The speedups are already quite dramatic, especially for large profiles; nevertheless, there is still room for improvement. Using simulations, we have verified that the FPGA algorithm scores one document term per two clock cycles. The limiting factor is the SRAM access speed of 128 bits/cycle, requiring two cycles to read four profile terms. At a clock speed of 100 MHz, this means the FPGA is capable of scoring the EPO collection in 15 seconds. The current application takes about 8.5 seconds on four FPGAs, so in principle it should be possible to increase the performance with a factor of two at least.

The reason for the discrepancy lies in the streaming I/O: the document stream is transferred from user memory space to the NUMAlink via a host operating system device driver, which instigates a direct memory access (DMA) transfer. The driver transfers a buffered block of the stream. Currently, this transfer is not implemented optimally in terms of the transferred block size, leading to suboptimal throughput. Furthermore, using a separate thread for queuing the transfers would eliminate the transfer time from the latency.

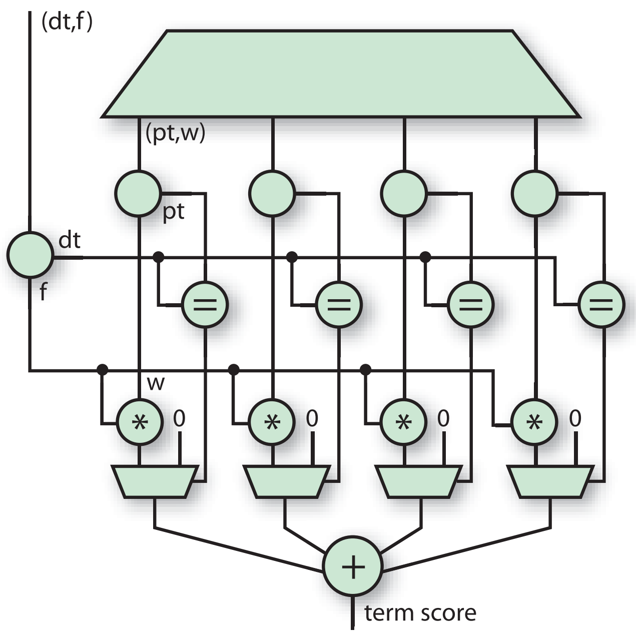


Figure 4 — Document term scoring

Issues and Lessons Learned

The project was interesting not only because it demonstrated the potential of FPGAs as accelerators for information retrieval tasks. It also provided us with valuable insights into the hardware and software requirements for FPGA-accelerated systems.

The I/O to the host system is key to performance: it was crucial that a DMA mechanism between the NUMA memory and the FPGA was supported by both the Mitrionics SDK and the SGI RASClib. In a previous project, it was necessary to transfer the data to the onboard SRAM before it could be processed. But this resulted in a big hit on performance, since the loading of data and unloading of the results formed a significant overhead. It also became clear that IR tasks, in particular, require considerable amounts of on-chip and onboard memory for maximum efficiency.

Furthermore, to make optimal use of the FPGAs, two features are essential for future platforms: it must be possible to transfer data directly between the FPGAs and it must be possible to switch off the host processor (or to control numerous FPGAs with one host processor). The ability to turn off the host processor is particularly important: on the Altix platform, the Itanium processor cannot be switched off even if it's completely idle. Unfortunately, an idle Itanium processor still consumes 90 percent of the power of a fully active one. As a result, although the energy savings resulting from the FPGA acceleration are considerable, the power saving in our current system is minimal, despite the fact that the host processor is idle during the run of the accelerator.

Another aspect of developing FPGA-accelerated systems is the software. Our

Collection	Profile # Docs	Processor # Unique Terms	CPU (Seconds)	FPGA (Seconds)	Gain
Aquaint	1	254	21.3	2.6	8.3x
	10	1,444	27.4	2.6	10.5x
	50	4,713	34.5	2.6	13.2x
USPTO	1	28	64.0	7.2	8.9x
	10	148	68.3	7.1	9.6x
	50	615	76.9	7.5	10.3x
EPO	1	1,327	107.3	8.4	12.7x
	10	4935	153.3	8.1	19.0x
	50	12,314	177.1	8.5	20.8x

Table 2 – Performance statistics

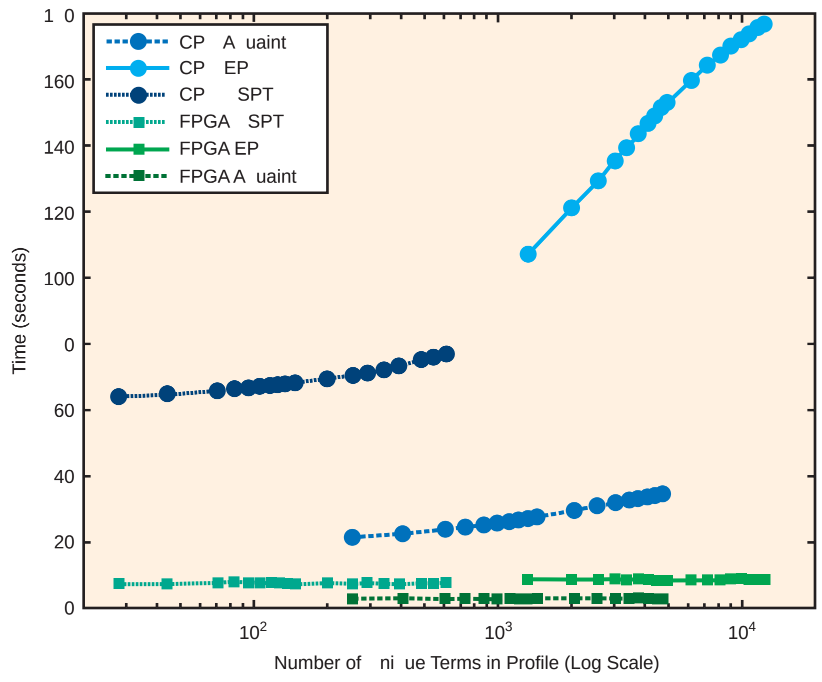


Figure 5 – Time in seconds vs. number of documents in profile

experience clearly shows that the main complexity is the interfacing between the FPGA and the host system: the actual FPGA application development in Mitrion-C was very productive; the framework for querying and serving documents using the Lemur tool kit was relatively easy to develop. However, the code interfacing the host application with the FPGA, using the RASClib, was very complicated and, because of concurrency issues, difficult to debug. As a result, this interface code dominated the development time by an order of magnitude.

A final issue with high-level programming of FPGAs is the compilation speed. Developers accustomed to languages like C++ or Java expect short build times even for very complex applications. The current FPGA tools require almost a full day to perform synthesis and place-and-route on all but the most trivial designs. These long build times seriously hamper productivity and would need to be reduced to the same order of software build times to make FPGA acceleration more attractive.

Tailored Hardware Platforms

With this project we explored the possibilities of FPGA acceleration and demonstrated the potential of FPGAs as a greener technology for the data center. We want to expand this research to investigate the full chain of tasks required for document processing: parsing, stemming, indexing, search and filtering. As it has become clear that off-the-shelf systems are limited in terms of their energy-saving potential, we want to investigate tailored hardware platforms specifically designed to perform information retrieval tasks with the highest possible efficiency. By doing so, it will be possible to speed up the execution of algorithms by an order of magnitude, while also reducing the energy consumption by an order of magnitude to create greener and faster data centers.

References

[1] C.L. Belady, "In the Data Center, Power and Cooling Costs More than the IT Equipment it Supports," *Electronics Cooling*, Vol. 13, No. 1, 2007.

[2] McKinsey & Co., "Revolutionizing Data Center Efficiency," Uptime Institute, 2008.

[3] L. Azzopardi, W. Vanderbauwhede and M. Moadeli, "Developing Energy-Efficient Filtering Systems," *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, 2009.

[4] W. Vanderbauwhede, L. Azzopardi and M. Moadeli, "FPGA-Accelerated Information Retrieval: High-Efficiency Document Filtering," *Proceedings of the 19th IEEE International Conference on Field Programmable Logic and Applications (FPL09)*, IEEE, 2009.

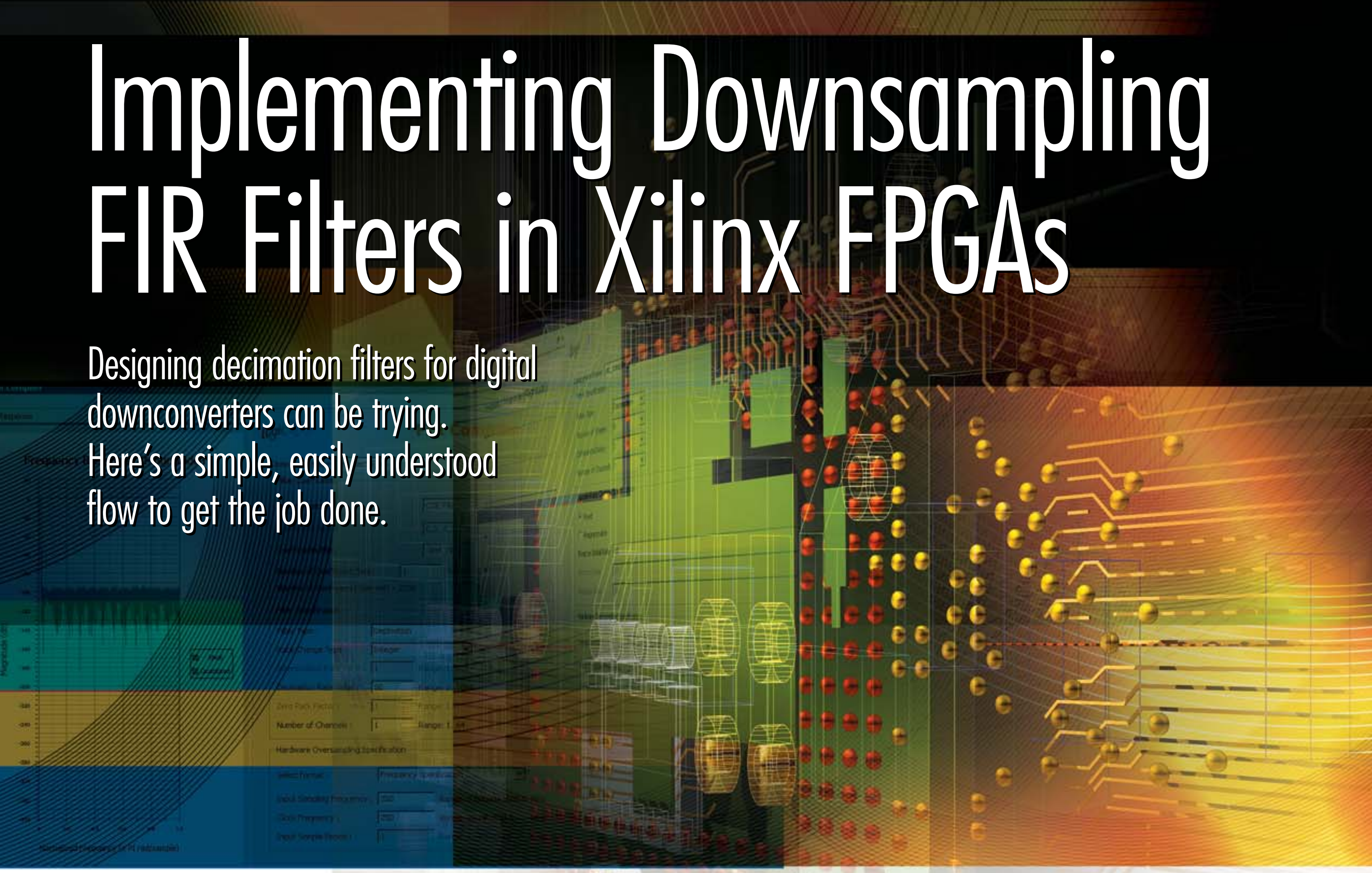
[5] N.J. Belkin and W.B. Croft, "Information Filtering and Information Retrieval: Two Sides of the Same Coin?" *Communications of the ACM*, Vol. 35, No. 12, 1992.

[6] V. Lavrenko and W.B. Croft, "Relevance-Based Language Models," *Proceedings of the 24th ACM SIGIR Conference*, 2001.

[7] C. McNairy and R. Bhatia, "Montecito: A Dual-Core, Dual-Thread Itanium Processor," *IEEE Micro*, Vol. 25, No. 2, March 2005.

[8] S. Sharp, "Virtex-4 Dynamic Power Comparison—A Case Study," December 2005. *http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/resources/Virtex4_Power_Case_Study.pdf*

[9] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, Vol. 13, No. 7, July 1970.



Implementing Downsampling FIR Filters in Xilinx FPGAs

Designing decimation filters for digital downconverters can be trying. Here's a simple, easily understood flow to get the job done.

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

Over the last two quarters, I've had several customers ask for help in designing and implementing downsampling (aka "decimation") filters for digital downconverters, which are common in both software-defined radio and data-acquisition type applications.

This isn't a trivial task for even an experienced designer. Indeed, just figuring out what resources you will need to implement a filter in an FPGA can be a big problem. Although MATLAB® (by The MathWorks) has a fantastic tool box for filter design and analysis (FDA), it presents so many ways to design a filter that a new user can easily get lost. Further, you have to be able to interpret the results that MATLAB commands generate in relation to DSP theory, which alone requires a bit of study.

With these issues in mind, but without getting too mired in theory, let's examine the design and implementation of a finite impulse response (FIR) filter for downsampling. This tutorial will, in fact, walk you through an easy and understandable flow, from the generation of the filter coefficients to the implementation of the decimation filter in the FPGA target device. The only tools needed are a fairly recent version of MATLAB (I am still using R2008a) and its FDA tool box, plus the Xilinx CORE Generator™ tool available in ISE® 11.4. These tools are mandatory for designing multirate FIR filters.

Specifically, we'll walk through two cases of fixed downsampling rate changes: integer and rational values. You should be able to apply the MATLAB instructions and the CoreGen graphical user interface (GUI) settings we'll go through in this tutorial to your designs. To illustrate the resource utilization in terms of common logic block (CLB) slices, 18-kbit memory RAM blocks (BRAM) and DSP48 multiply-and-accumulate (MAC) units, we'll use the XC6VLX75T-2ff484 as our target FPGA device.

Integer Factor Downsampler

Let us assume that after the demodulation in baseband, a signal with bandwidth of only 2.5 MHz is carried out at a rate of 250 MHz. We have to filter all the frequencies from 2.5 MHz up to 125 MHz, since they do not convey any useful information; this is the purpose of the low-pass FIR filter we plan to design and implement. According to the Nyquist theorem, the output data rate is twice the bandwidth of the signal; therefore, we need to downsample it by an integer factor of M=50. I will show two possible alternative implementations by applying a multistage filtering approach: the first method will use a chain of three FIR decimation filters and the second, both cascade-integrator-comb (CIC) and FIR filters.

Here is the MATLAB code to design the golden filter. We assume an attenuation of 0.1 dB and 100 dB respectively in the passband and stopband frequencies.

```
%% Golden reference FIR filter
Fs_in   = 250e6  % input data  rate in Hz
Fs_out  =   5e6  % output data rate in Hz
M = Fs_in/Fs_out  % down-sampling factor
% Low pass FIR filter design specs
Fp  = 0.4*Fs_out  % pass band corner freq
Fst = 0.5*Fs_out  % stop band corner freq
Ap  = 0.1;        % pass band attenuation (dB)
Ast = 100.0;      % stop band attenuation (dB)
% Filter design with FDA tool
h1=fdesign.decimator(M,'Lowpass',...
'Fp,Fst,Ap,Ast',Fp, Fst, Ap, Ast, Fs_in);
Href = design(h1);
info(Href)    % show filter info
fvtool(Href); % plot freq response
title ('reference single stage filter');
legend('reference single stage filter');
axis([0 25 -120  5]) % zoom-in 0 to 25MHz
% generating the COE file
ref_filter = Href.Numerator;
gen_coe_rad10(Href.Numerator,...
 'ref_filter_rad10.coe');
```

Assuming an FPGA clock frequency of Fclk=Fs_in, how many DSP48 MAC units do we need in the Virtex®-6 device? This is a filter to downsample by M. The theory—well explained in the FIR-Compiler 5.0 data sheet (fir_compiler_ds534.pdf)—says that we can decompose it into M phases; hence the term "polyphase." Since each phase is processing at the lower output frequency Fs_out, the DSP48 MAC can be shared in time-division multiplexing. The following theoretical computation shows that the FIR-Compiler utilizes a minimum of 22 MAC elements (total_num_MAC_ref) for this filter when it's implemented via polyphase decomposition. The filter length is 2100 (total_num_coeff), once padded with zeros to become an integer multiple of M. Note that this scheme takes coefficients symmetry into account.

```
Fclk = Fs_in; num_phases = M;
num_coeff_x_phase = ...
  ceil(numel(ref_filter) / num_phases )
% effective total number of coefficients
total_num_coeff = num_phases *...
  num_coeff_x_phase
% number of DSP48 utilized per each phase
num_MAC_x_phase = (num_coeff_x_phase * ...
  Fs_out / Fclk);
% effective number of DSP48 MAC
total_num_MAC_ref = ceil(num_phases * ...
  num_MAC_x_phase /2)+1 % /2 due to symmetry
% pad with zeros the original coeff:
pad_filt = [ref_filter zeros(1, ...
  total_num_coeff - numel(ref_filter))];
```

In MATLAB, it's easy to model the decimation process as a low-pass filtering and then downsample by M, producing respectively y and y_filt output signals. But in the FPGA device, such implementation is not efficient: it would be foolish to compute values that must then be thrown away. Instead, the polyphase decimator downsamples the input signals into M channels wk, each one filtered by its own subfilter ph(k,:). The partial results y_out(k,:) are then summed together to compose the final output result y_tot. Comparing y_tot with the reference one y, achieved by native MATLAB instruction, shows they are the same within a numerical accuracy of 3e-15 (due to the different order of the operations).

```
%% Simulating polyphase downsampler
F1 = 1e6;   % first freq  (Hz)
F2 = 3e6;   % second freq (Hz)
N  = 2^14;  % number of samples
n = 0 : 1 : N-1; % time basis
% Input signal
x = cos(n*2*pi*F1/Fs_in) + ...
    cos(n*2*pi*F2/Fs_in);
% Compute single-side amplitude spectrum
% AC component will be doubled and
% DC component will be kept as the same
X = 2*abs(fft(x,N))/N; X(1)=X(1)/2;
% Map the freq bins into frequencies in Hz
f_bin = [0 : 1 : N/2-1]*Fs_in/N;
figure; plot(f_bin, X(1:1:N/2));
grid; xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Input Spectrum');
axis([0 1e7 0 0.8]); % zoom-in 0 to 10 MHz
% Low-pass filtered output signal
y_filt  = filter(pad_filt, 1, x);
y       = y_filt(1:M:end); % ref down-sampled
NM      = length(y);
% Compute single-side amplitude spectrum:
Y     = 2*abs(fft(y,NM))/NM; Y(1)=Y(1)/2;
fs_M  = [0 : 1 : NM/2-1]*Fs_in/(M*NM);
figure; plot(fs_M, Y(1:1:NM/2));grid;
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Down-sampled Spectrum');
axis([0 1e7 0 0.8]); % zoom-in 0 to 10 MHz
%% Polyphase decomposition of the filter
ph0 = pad_filt(1 : M : end); % phase 0
w0  = x(1 : M : N);    % channel 0 of x(n)
```

```
ph_len = length(ph0); % phase length
% Output from phase 0 filter
y_out(1,:) = filter(ph0,1,w0);
y_tot = y_out(1,:); % initialize final res
for k = 2 : M % loop along the k phases
  ph(k,1:ph_len)= pad_filt(k:M:end);
  % delay x(n) by k samples:
  x_del = filter([zeros(1, k-1), 1],1,x);
  % channel k is down-sampled by M:
  wk  = x_del(1 : M : N);
  % output from k phase filter:
  y_out(k,1:NM) = filter(ph(k,:),1,wk);
  % update partial result:
  y_tot = y_tot + y_out(k, :);
end
% Reference vs. polyphase filter outputs
diff = y_tot -y;
sum_abs_diff = sum(abs(diff));
figure; plot(diff);
```

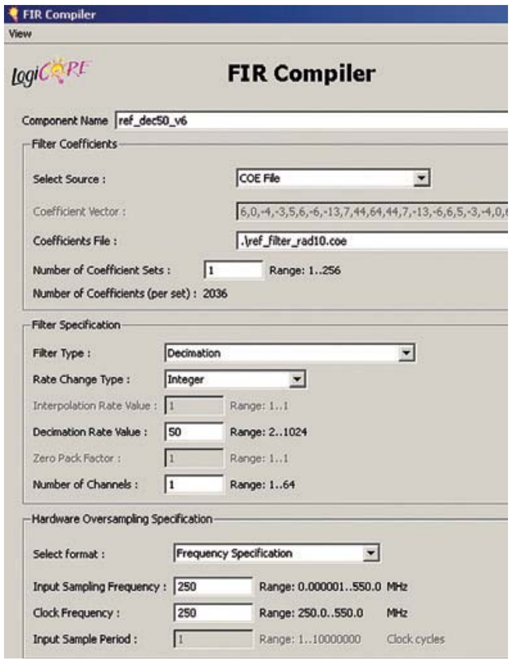
To design the reference filter, CoreGen FIR-Compiler requires the text file of coefficients, known as the COE file. The following MATLAB routine shows how to easily generate this COE file in decimal radix; FIR-Compiler will then quantize the coefficients according to the adopted settings.

```
function gen_coe_rad10(filt_num, fileName);
% max number of coefficients
num_coeffs = numel(filt_num);
fileId = fopen(fileName, 'w')
% header of COE file
fprintf(fileId, 'radix = 10;\n');
% first coefficient
fprintf(fileId,'coefdata = ',...
    '%18.17f,\n', filt_num(1)');
for i = 2 : num_coeffs-1
  fprintf(fileId,'%18.17f,\n',filt_num(i));
end
% last coefficient
fprintf(fileId,'%18.17f;\n',...
    filt_num(num_coeffs));
fclose(fileId);
end
```

Figures 1 and 2 show the design parameters applied in the first two pages of the FIR-Compiler GUI; in the remaining two pages I simply accepted the default values except for "Optimization Goal," which I set to "Speed" instead of "Area." When not explicitly mentioned, these are the settings I will adopt throughout this paper and also for the next examples.

After ISE 11.4 placement and routing, the reference single-stage downsampling filter consumes the following FPGA resources:

Number of slice flip-flops:	1,265
Number of slice LUTs:	1,744
Number of occupied slices:	502
Number of DSP48 units:	22



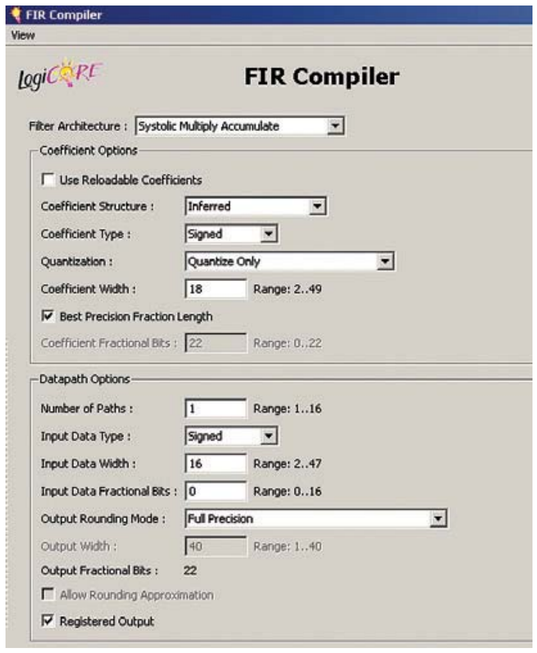
Component Name: ref_dec50_v6

Filter Coefficients
Select Source : COE File
Coefficient Vector : 6,0,-4,-3,5,6,-6,-13,7,44,64,44,7,-13,-6,6,5,-3,-4,0,
Coefficients File : .\ref_filter_rad10.coe
Number of Coefficient Sets : 1 Range: 1..256
Number of Coefficients (per set) : 2036

Filter Specification
Filter Type : Decimation
Rate Change Type : Integer
Interpolation Rate Value : 1 Range: 1..1
Decimation Rate Value : 50 Range: 2..1024
Zero Pack Factor : 1 Range: 1..1
Number of Channels : 1 Range: 1..64

Hardware Oversampling Specification
Select format : Frequency Specification
Input Sampling Frequency : 250 Range: 0.000001..550.0 MHz
Clock Frequency : 250 Range: 250.0..550.0 MHz
Input Sample Period : 1 Range: 1..10000000 Clock cycles

Figure 1 – Integer downsampling by 50. Page 1/4 of FIR-Compiler 5.0 GUI settings for the reference single-stage filter.



Filter Architecture : Systolic Multiply Accumulate

Coefficient Options
☐ Use Reloadable Coefficients
Coefficient Structure : Inferred
Coefficient Type : Signed
Quantization : Quantize Only
Coefficient Width : 18 Range: 2..49
☑ Best Precision Fraction Length
Coefficient Fractional Bits : 22 Range: 0..22

Datapath Options
Number of Paths : 1 Range: 1..16
Input Data Type : Signed
Input Data Width : 16 Range: 2..47
Input Data Fractional Bits : 0 Range: 0..16
Output Rounding Mode : Full Precision
Output Width : 40 Range: 1..40
Output Fractional Bits : 22
☐ Allow Rounding Approximation
☑ Registered Output

Figure 2 – Integer downsampling by 50. Page 2/4 of FIR-Compiler 5.0 GUI settings for the reference single-stage filter.

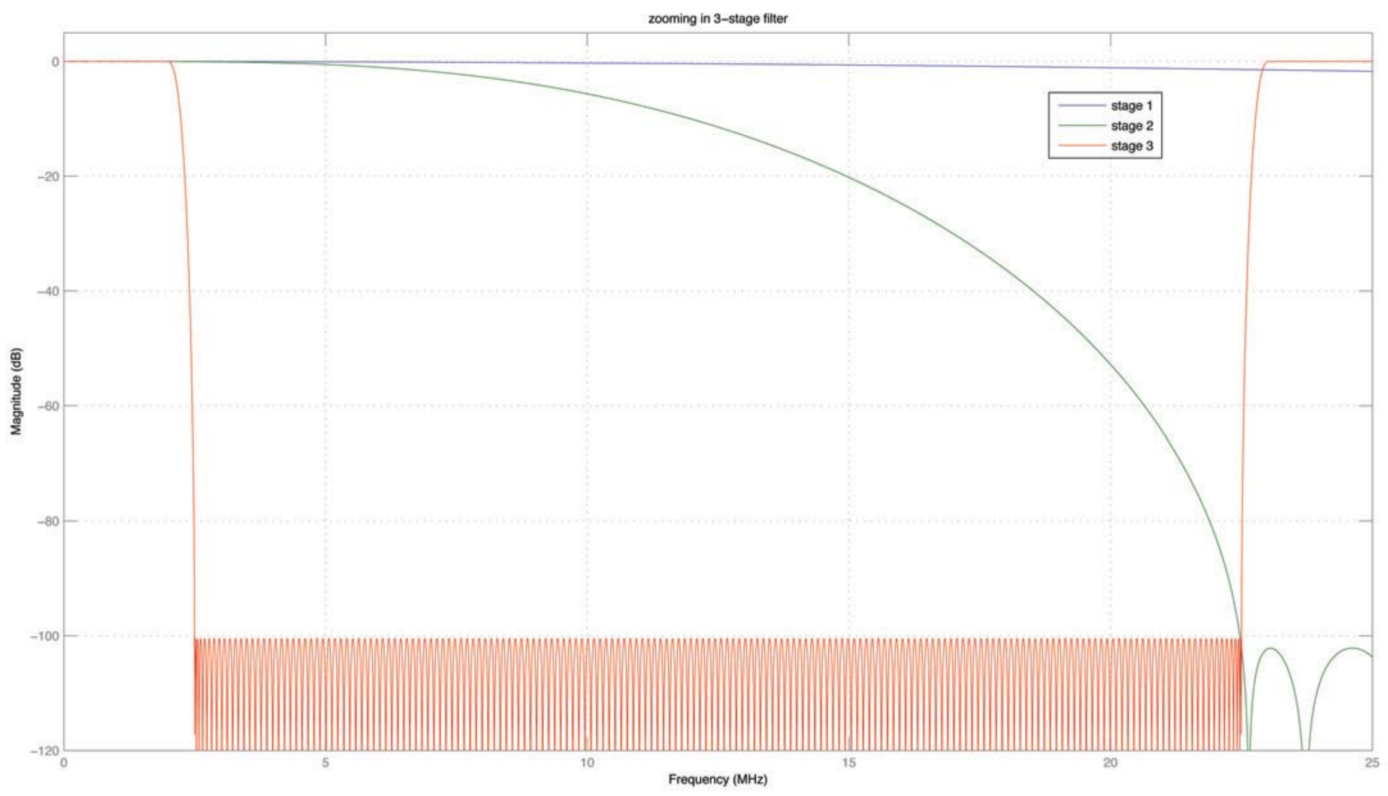


Figure 3 – Decimation by 50 via the cascade of three FIR filtering stages, here shown separately with a zoom in the frequencies from 0 to 25 MHz.

Cascade of Three FIR Filtering Stages

Let us now implement our golden decimation filter as a cascade of filtering stages. This technique will allow us to save MAC units by means of time-division multiplexing, since every new filtering stage will work at a lower data rate, given by the previous stage. I let the FDA tool decide the optimal filter type: by applying the MATLAB instruction info, you can see that it will propose a solution with three stages, respectively, of decimation factors M1=2, M2=5 and M3=5.

```
%% Multistage approach with three FIR filters
Hmulti=design(h1, 'multistage');
hm1 = Hmulti.Stage(1);
M1 = Hmulti.Stage(1).DecimationFactor;
hm2 = Hmulti.Stage(2);
M2 = Hmulti.Stage(2).DecimationFactor;
hm3 = Hmulti.Stage(3);
M3 = Hmulti.Stage(3).DecimationFactor;
info(hm1)
info(hm2)
info(hm3)
fvtool(hm1,hm2,hm3, 'Fs', ...
 [Fs_in, Fs_in/M1, Fs_in/(M1*M2)])
legend('stage 1','stage 2','stage 3');
title '3-stage filter'
fvtool(hm1,hm2,hm3, 'Fs', ...
 [Fs_in, Fs_in/M1, Fs_in/(M1*M2)])
legend('stage 1','stage 2','stage 3');
title 'zooming in 3-stage filter'
axis([0 25 -120  5])
fvtool(Href, cascade(hm1, hm2, hm3), ...
  'Fs', [Fs_in])
legend('reference filter',...
 'multi-stage FIR filters');
title 'zooming in ref. vs. s-stage filters'
axis([1.5 3 -120  5])
hm1n = hm1.Numerator;
hm2n = hm2.Numerator;
hm3n = hm3.Numerator;
gen_coe_rad10(hm1n,'filt1_rad10.coe');
gen_coe_rad10(hm2n,'filt2_rad10.coe');
gen_coe_rad10(hm3n,'filt3_rad10.coe');
```

Figure 3 shows the frequency response of the three filters composing this multistage system. The blue curve represents the first downsampling filter (M1=2); the green curve is the second (M2=5), periodic at multiples of Fs_in/M1; and the red curve represents the third downsampler (M3=5), periodic at multiples of Fs_in/(M1*M2).

The FIR-Compiler settings for the three-stage filters are pretty much the same as the ones illustrated in Figures 1 and 2. For the first stage, the only parameters that differ are the name of the COE file and the “decimation rate value,” respectively set to filt1_rad10.coe and M1=2. For the second filter, the COE file is named filt2_rad10.coe, the decimation rate value is M2=5 and the input sampling frequency is now 125 MHz, since the second stage takes input data decimated by M1=2 from the first stage. Finally, the only differences in the parameters of the third filter are the name of the COE file, filt3_rad10.coe, the decimation rate value of M3=5 and the input sampling frequency, which is now 25 MHz, since the third stage takes input data decimated by M2=5 from the second stage.

After place-and-route, the three filtering stages occupy the following FPGA resources:

Stage 1 (M1=2):

Number of slice flip-flops:	280
Number of slice LUTs:	208
Number of occupied slices:	62
Number of DSP48 MAC units:	3

Stage 2 (M2=5):

Number of slice flip-flops:	236
Number of slice LUTs:	168
Number of occupied slices:	60
Number of DSP48 MAC units:	3

Stage 3 (M3=5):

Number of slice flip-flops:	357
Number of slice LUTs:	414
Number of occupied slices:	158
Number of DSP48 MAC units:	4

Thanks to this multistage approach, we now utilize 12 fewer DSP48 MAC units than the 22 originally used by the reference filter; in relative terms, we save around 30 percent in flip-flops, 55 percent LUTs, 44 percent slices and 54 percent DSP48 elements compared with the resources taken by the single-stage golden filter.

Cascade with CIC Filters

Another possible approach to the decimation by 50 involves the cascade of a cascaded integrated comb (CIC) and CIC-compensation downsampling stages, with change rate respectively of M1=10 and M2=5. CIC filters are a special class of FIR filters that consist of N comb and integrator sections (hence the term “Nth order”). The CIC architecture is interesting since it does not require any MAC element, although the comb section could also be implemented as a “traditional” MAC-based FIR filter, thus trading DSP48 units vs. CLB slices, as also explained in the CoreGen CIC-Compiler 1.3 data sheet (cic_compiler_ds613.pdf).

The first-stage CIC filter decimating by M1=10 has a poor frequency response and therefore needs a compensation FIR filter decimating by M2=5 to offset the droop in the passband of the first-stage CIC filter itself. The following MATLAB code explains how to design such filters with the FDA tool.

```
%% Multistage with CIC and FIR filters
% First stage: CIC filter
DD =  1; % CIC differential delay
M1 = 10; % CIC decimation rate change
h2   = fdesign.decimator(M1,'cic',...
  DD,'Fp,Ast',Fp,Ast,Fs_in);
Hcic = design(h2);
% let us force a 6-th order CIC
Hcic.NumberOfSections=6;
Nsecs = Hcic.NumberOfSections
info(Hcic)
% gain of CIC filter to be normalized in
% order to get an overall unit gain
gain_cic = 2^(Nsecs*log2(M1));
% Second stage CIC compensator FIR filter
% its stop band decays like (1/f)^2.
M2 = M/M1; % FIR decimation rate change
Ast = 80;  % to relax stopband attenuation
h3 = fdesign.decimator(M2,'ciccomp',...
  DD,Nsecs,Fp,Fst,Ap,Ast,Fs_in/M1);
Hmc = design(h3, 'equiripple', ...
'StopbandShape', '1/f', ...
'StopbandDecay', 2);
info(Hmc)
% Analyze filters
fvtool(cascade(Hcic,1/gain_cic),...
Hmc,cascade(Hcic,Hmc,1/gain_cic),...
  'Fs',[Fs_in,Fs_in/M1,Fs_in],...
  'ShowReference','off')
legend('CIC decimator','CIC compensator',...
'Overall response');
title 'multistage with CIC^6 filter'
fvtool(cascade(Hcic,1/gain_cic),...
Hmc,cascade(Hcic,Hmc,1/gain_cic),...
  'Fs',[Fs_in,Fs_in/M1,Fs_in],...
  'ShowReference','off')
legend('CIC decimator',...
   'CIC compensator','Overall response');
title 'zooming in multistage with CIC^6 filter'
axis([0 10 -120  5])
cic_comp_filt = Hmc.Numerator;
gen_coe_rad10(cic_comp_filt,...
  ['ciccomp_dec', num2str(M2), '.coe']);
fvtool(Href, cascade(hm1, hm2, hm3), ...
  cascade(Hcic,Hmc, 1/gain_cic), 'Fs', ...
[Fs_in, Fs_in, Fs_in]);
legend('reference filter',...
  '3-stage FIR filter', ...
  'CIC and CIC Comp filters');
title 'ref, 3-stage vs. CIC + CIC Comp'
axis([1.5 3 -110  2])
```

Figure 4 illustrates the first page of Xilinx CoreGen CIC-Compiler 1.3 GUI settings; the remaining parameters have default values, except for the “Use Xtreme DSP Slice” optional parameter (page 2 of 3 in the GUI), which makes it possible to implement the comb section with or without DSP48 elements. The CIC compensation FIR filter design parameters in FIR-Compiler GUI are the same ones already seen in Figures 1 and 2; the only settings that differ are the COE file name (here, cic-comp_dec5.coe), the decimation rate value of M2=5 and the input sampling frequency of 25 MHz.

After place-and-route, the two filtering stages take the following FPGA resources:

First stage (**CIC decimating by 10, no “Use Xtreme DSP Slice”**)

Number of slice flip-flops:	755
Number of slice LUTs:	592
Number of occupied slices:	172
Number of DSP48 MAX units:	0

First stage (**CIC decimating by 10, with “Use Xtreme DSP Slice”**)

Number of slice flip-flops:	248
Number of slice LUTs:	154
Number of occupied slices:	42
Number of DSP48 MAC units:	7

Second stage (**CIC compensation FIR filter decimating by 5**)

Number of slice flip-flops:	271
Number of slice LUTs:	312
Number of occupied slices:	114
Number of DSP48 MAC units:	3

Both results are interesting, and the choice of whether to use Xtreme DSP slices will depend on what resources a designer most needs to save. Personally I would select the “Use Xtreme DSP Slice” option. In relative terms we would save about 59 percent of flip-flops, 73 percent LUTs, 69 percent slices and 54 percent DSP48 MAC units, compared with the resources taken by the single-stage filter. The price is a worse attenuation in the stopband, which is now 80 dB instead of the 100 dB required, as shown in Figure 5. Whether a design can accept such an attenuation value or not is really application dependent.

Figure 5 offers a comparison among the three methods for downsampling by 50: the single stage, the three stages (ratios 2-5-5) and the cascade of CIC and CIC-compensation FIR filters (ratios 10-5).

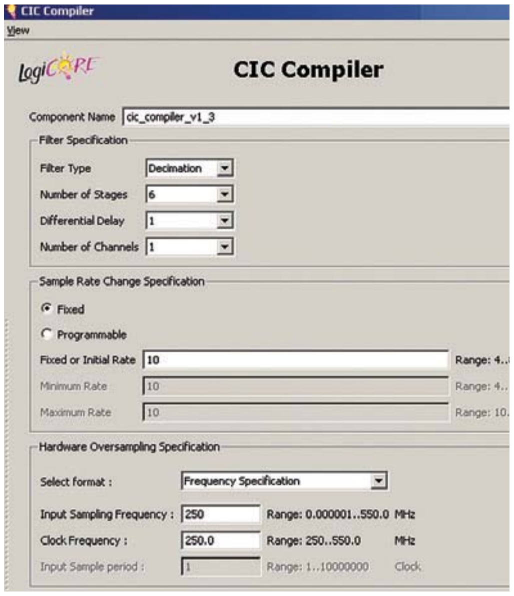

CIC Compiler
View
CIC Compiler
Component Name: cic_compiler_v1_3
Filter Specification
Filter Type: Decimation
Number of Stages: 6
Differential Delay: 1
Number of Channels: 1
Sample Rate Change Specification
(●) Fixed
( ) Programmable
Fixed or Initial Rate: 10 Range: 4..
Minimum Rate: 10 Range: 4..
Maximum Rate: 10 Range: 10.
Hardware Oversampling Specification
Select format : Frequency Specification
Input Sampling Frequency : 250 Range: 0.000001..550.0 MHz
Clock Frequency : 250.0 Range: 250..550.0 MHz
Input Sample period : 1 Range: 1..10000000 Clock

Figure 4 – Settings for the CIC filter decimating by 10. Page 1/3 of CIC-Compiler 1.3 GUI.

Downsampling by a Rational Factor

In this second application example, we assume a signal with an input data rate of 50 MHz that has to be downsampled to 12 MHz, thus requiring a rational, fixed rate change of L/M=6/25 (or in equivalent words, a decimation factor of M/L=25/6). The FPGA clock frequency is supposed to be 150 MHz.

As also explained in the FIR-Compiler 5.0 data sheet, a filter with rational rate change ideally requires two processing steps: interpolation by L, followed by decimation by M. In our specific case, once the input signal is interpolated by L=6, the output virtual sampling rate Fv will be 300 MHz. Therefore, the frequency range between Fs_in/2=25MHz and Fv/2=150MHz has to be filtered out to remove the spectra placed at integer multiple of Fs_in. Called “images” in DSP terminology, they are the reason for using the interpolation “anti-imaging” low-pass filter.

After this processing step, we need to apply a low-pass filter to remove the frequencies from Fv/(2*M)=6MHz to Fv/2=150MHz, named “alias” in DSP terminology, before the final downsampling by M. Since these two low-pass filters are in cascade and work at the same virtual data rate Fv, we can use the one with smaller bandwidth for anti-imaging and anti-aliasing at the same time, thus saving resources. In our case, the filter with minimum bandwidth is the decimation filter.

The following MATLAB fragment illustrates how to design and simulate such a downsampler with a single-stage filter. We assume an attenuation of 0.05 dB and 70 dB respectively in the passband and stopband frequencies.

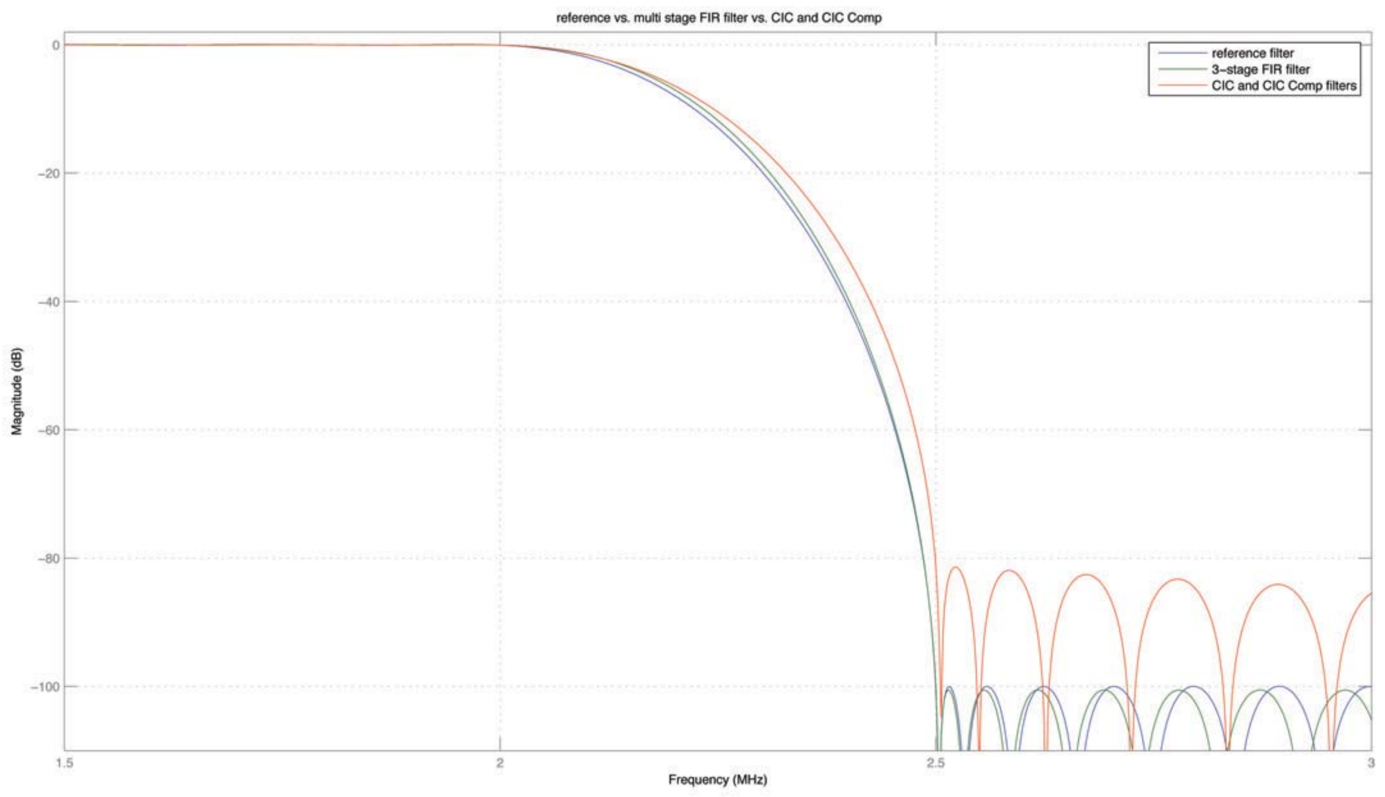


Figure 5 – Frequency responses of the three downsamplers with overall rate change of 50 and with zoom in the frequency range from 1.5 to 3 MHz. The single stage is shown in blue, the three-stage (ratios M1=2, M2=5, M3=5) is in green and the two-stage CIC-based (ratio M1=10, M2=5) is in red.

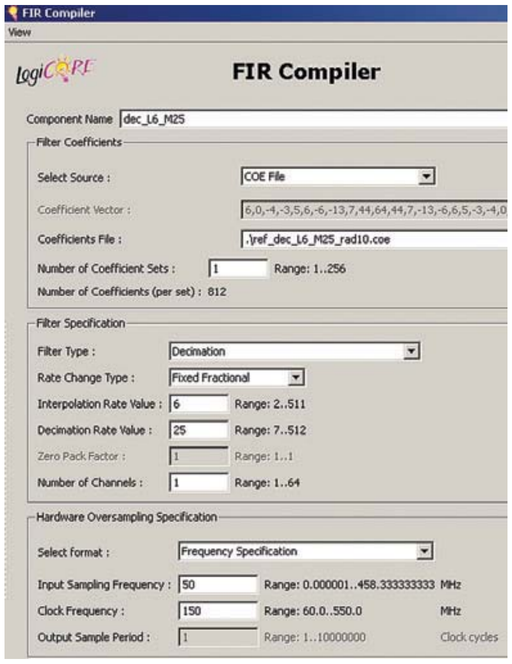
```
   %% Golden reference model
M     = 25;       % downsample factor
L     =  6;       % upsample factor
Fs_in = 50e6;     % input sampling rate
Fv     = L*Fs_in; % virtual output rate
Fout  = Fv/M;     % effective output rate
rp    = 0.05      % passband ripple (dB)
rs    = 70        % stopband attenuation (dB)
dens   = 20;      % density factor
% Deviations in linear scale
dev    = [(10^(rp/20)-1)/(10^(rp/20)+1) ...
  10^(-rs/20)]
Fpass  = 0.40*Fout % passband frequency
Fstop  = 0.50*Fout % stopband frequency
% Calculate the filter order using FIRPMORD
[N0, Fo, Ao, W] = firpmord([Fpass, ...
  Fstop]/(Fv/2), [1 0], dev);
% Calculate the coefficients using FIRPM
 b0  = firpm(N0, Fo, Ao, W, {dens});
Hd0 = dfilt.dffir(b0);
info(Hd0)
hvft= fvtool(Hd0, 'Fs', Fv);
title 'Reference Decimator by 25/6'
legend(hvft,'Reference Decimator by 25/6',...
  'Location','NorthEast' )
%% Simulation of rational decimation
F1 = 14e6; % first freq
F2 =  4e6; % second freq
N  = 2^13; % number of samples of x(n)
n  = 0 : 1 : N-1;
% input signal with two sinusoids:
x = 10*cos(n*2*pi*F1/Fs_in) + ...
  5*cos(n*2*pi*F2/Fs_in);
% Single-side amplitude spectrum of x(n):
X = 2*abs(fft(x,N))/N; X(1)=X(1)/2;
% mapping of frequency bins:
f_bin = [0 : 1 : N/2-1]*Fs_in/N;
% plot spectrum:
figure; plot(f_bin,X(1:1:N/2)); grid;
xlabel('Frequency (Hz)');
title '1-sided spectrum of input signal x(n)'
% Polyphase interpolation by L
NL = N*L;
filt_len = length(Hd0.Numerator);
real_len = L*ceil(filt_len/L);
num_coef_phase = real_len/L;
```

```
reg      = zeros(1, num_coef_phase);
filt_coe = [Hd0.Numerator ...
  zeros(1, real_len-filt_len)];
poly_phase=reshape(filt_coe,L,...
  num_coef_phase);
tap_delay = zeros(1, num_coef_phase );
w2        = zeros(1, NL);
k = 0;
for i = 1 : N
  tap_delay = [x(i) tap_delay(1:end-1)];
  w2(k+1) = tap_delay * poly_phase(1, :)';
  w2(k+2) = tap_delay * poly_phase(2, :)';
  w2(k+3) = tap_delay * poly_phase(3, :)';
  w2(k+4) = tap_delay * poly_phase(4, :)';
  w2(k+5) = tap_delay * poly_phase(5, :)';
  w2(k+6) = tap_delay * poly_phase(6, :)';
  k = k+6;
end
y  = w2(1:M:NL);% real down-sampling by M
NM = length(y); % length of down-sampled data
% Single-side amplitude spectrum:
Y  = 2*abs(fft(y,NM))/NM; Y(1)=Y(1)/2;
% mapping of frequency bins:
fsM = [0 : 1 : NM/2-1]*(Fs_in*L/M)/NM;
% plot spectrum: note all repetitions
% removed from Fout/(2M) to Fout/2
figure; plot(fsM,Y(1:1:NM/2));grid;
xlabel('Frequency (Hz)');
title 'spectrum after downsampling by 25/6'
% save COE filter coefficients
gen_coe_rad10(Hd0.Numerator,...
  'ref_dec_L6_M25_rad10.coe');
```

Note that this MATLAB code is just a behavioral model of the rational downsampling filter. In the real hardware polyphase architecture, you will only need to implement a single-phase filter and then change the set of coefficients for every new output sample (while the processing runs at Fclk rate). This is different from what happened for the polyphase downsampling filter with integer ratio.

Figure 6 reports the settings of the first FIR-Compiler GUI page. For the remaining three pages I used the same parameters already adopted in the first application example of integer downsampling. Total FPGA resource occupation after place-and-route is:

Number of slice flip-flops:	547
Number of slice LUTs:	451
Number of occupied slices:	153
Number of DSP48 units:	13
Number of BRAM units:	6



Component Name: dec_L6_M25

Filter Coefficients
- Select Source : COE File
- Coefficient Vector : 6,0,-4,-3,5,6,-6,-13,7,44,64,44,7,-13,-6,6,5,-3,-4,0
- Coefficients File : .\ref_dec_L6_M25_rad10.coe
- Number of Coefficient Sets : 1 Range: 1..256
- Number of Coefficients (per set) : 812

Filter Specification
- Filter Type : Decimation
- Rate Change Type : Fixed Fractional
- Interpolation Rate Value : 6 Range: 2..511
- Decimation Rate Value : 25 Range: 7..512
- Zero Pack Factor : 1 Range: 1..1
- Number of Channels : 1 Range: 1..64

Hardware Oversampling Specification
- Select format : Frequency Specification
- Input Sampling Frequency : 50 Range: 0.000001..458.333333333 MHz
- Clock Frequency : 150 Range: 60.0..550.0 MHz
- Output Sample Period : 1 Range: 1..10000000 Clock cycles

Figure 6 – Rational downsampling by 25/6. Page 1/4 of FIR-Compiler 5.0 GUI settings for the reference single-stage filter.

Multistage Approach

FIR-Compiler has generated a pretty small core for this polyphase L/M=6/25 filter. Nevertheless, let us again try the multistage approach, since it could allow us some further DSP48 and BRAM savings. When manually designing the multistage system, as in this specific case, all the filtering stages must have the same passband frequency (Fpass) as the reference filter.

The passband ripple is equal for all stages and is given by the reference filter passband ripple divided by the number of stages. What changes from stage to stage is the stopband frequency. The first stage does not need to cut at Fstop, since the transition bandwidth would be too sharp (too many coefficients); in reality, all we need is the first stage to cut at Fstop1=Fs_in/M1-Fs_in/(2M/L). In fact, Fs_in/M1 and all its multiples will now be the new sampling frequency at which all the replica are placed, while Fs_in/(2*M1) is half the bandwidth of the first replica in Fs_in/M1. Here is the MATLAB code.

```
%% First stage: rate change by L1/M1=1/4
L1 = 1; M1 = 4; % remember that M=25 L=6
Fs1    = Fs_in/M1 % output data rate
Fstop1 = Fs_in/M1- Fs_in/(2*M/L) % stop band
% Calculate filter order via FIRPMORD
[N1, Fo, Ao, W] = firpmord([Fpass, ...
   Fstop1]/(Fs_in/2), [1 0], dev/2);
% Calculate coefficients via FIRPM
b1  = firpm(N1, Fo, Ao, W, {dens});
Hd1 = dfilt.dffir(b1);
info(Hd1)
```

```
hvft = fvtool(Hd1, 'Fs', Fs_in);
title 'Stage1: Decimation by 4'
legend(hvft,'Stage1: Decimation by 4',...
   'Location','NorthEast')
gen_coe_rad10(Hd1.Numerator, ...
  'dec_L1_M4_rad10.coe');
%% Second stage: rate change by L2/M2=24/25
L2 = 24; M2 = 25;
Fs2    = L2*Fs1;     % virtual data rate
Fstop2 = 0.50*Fs2/M2  % Stop band Freq
% Calculate filter order via FIRPMORD
[N2, Fo, Ao, W] = firpmord([Fpass, ...
  Fstop2]/(Fs2/2), [1 0], dev/2);
% Calculate coefficients via FIRPM
b2  = firpm(N2, Fo, Ao, W, {dens});
Hd2 = dfilt.dffir(b2);
info(Hd2)
% plot spectra
info(Hd2)
hvft2 = fvtool(Hd2, 'Fs', Fs2);
title 'Stage2: L2/M2 downsampler'
legend(hvft2,'Stage2: Polyphase L2/M2', ...
'Location','NorthEast')
gen_coe_rad10(Hd2.Numerator,...
  'dec_L24_M25_rad10.coe');
%
% Let us compare the frequency response
% of reference L/M=6/25 filter vs.
% 2-stage (L1/M1)*(L2/M2)=(1/4)*(24/25)
%
hvft = fvtool(Hd0, cascade(Hd1, Hd2), ...
  'Fs', Fs2);
title 'single vs. multi-stage'
legend(hvft,'6/25 single stage', ...
  '(1/4)(24/25) multistage', ...
  'Location','NorthEast')
axis([3 8 -100 1]);
```

Since the first stage is an integer downsampler by M1=4, the FIR-Compiler GUI settings are very similar to those shown in Figure 1. The only parameters that differ are the name of the COE file, dec_L1_M4_rad10.coe; the decimation rate value (M1=4); the input sampling frequency (50 MHz); and the clock frequency (150 MHz). On the other hand, the second stage is a rational rate change of L2/M2=24/25, so the FIR-Compiler settings will be similar to the ones shown in Figure 6, again with just a few differences. The COE file here is named dec_L24_M25_rad10.coe, while the interpolation rate value is set at L2=24 and the input sampling frequency is 12.5 MHz.

After place-and-route, the two filtering stages occupy the following FPGA resources:

Stage 1 (L1/M1= 1/4):

Number of slice flip-flops	321
Number of slice LUTs:	223
Number of occupied slices:	62
Number of DSP48 MAC units:	4
Number of BRAM units:	0

Stage 2 (L2/M2 = 24/25):

Number of slice flip-flops:	206
Number of slice LUTs:	209
Number of occupied slices:	68
Number of DSP48 MAC units:	3
Number of BRAM units:	1

Thanks to the multistage approach, we now save around 3 percent in flip-flop, 4 percent LUT, 15 percent slices, 46 percent DSP48 and 83 percent BRAM elements compared with the resources taken by the single-stage golden filter. In particular, we utilize far fewer MAC and BRAM units, respectively six and five. This is due to the fact that the second filter works at a lower input sampling rate, while the first filter, featuring integer rate change, could exploit the coefficient symmetry.

Additional Resources

In this tutorial we have seen a couple of examples of downsampling filters, either of integer (50) or rational (25/6) ratios, with emphasis on a methodology for designing the filters in MATLAB and implementing them on Xilinx FPGAs via FIR-Compiler and CIC-Compiler. The data sheets offer much detail on the theory behind the parameter setting involved in implementing the filters in CORE Generator.

For those interested in delving further into the subject of DSP, two books in particular present a great mix of theory and related MATLAB instructions: *Digital Signal Processing Fundamentals and Applications* by Li Tan (Elsevier, 2007) and *Multirate Signal Processing for Communication Systems* by Fredric J. Harris (Prentice Hall, 2004). In addition, Xilinx's Web site features a number of great application notes (see especially Xapp113, 569, 1018 and 936) on multirate digital up- and downconversion.

Finally, to understand how to implement DSP algorithms efficiently, I would strongly recommend attending the Xilinx training course titled "DSP Implementation Techniques for Xilinx FPGAs."

Daniele Bagni is a DSP Specialist FAE for Xilinx EMEA in Milan, Italy. After earning a degree in quantum electronics from the Politecnico di Milano, he worked for seven years at Philips Research labs in digital video processing. For the next nine years he was project leader at STMicroelectronics' R&D labs, focusing in video coding on VLIW-architecture embedded processors, while simultaneously teaching a course in multimedia information coding as an external professor at the State Milan University. In 2006 he joined the Xilinx Milan sales office.

Cost Effectively Addressing the Insatiable Need for Bandwidth

By Manuel Uhm

THERE IS AN INSATIABLE need for bandwidth that is being driven by fixed mobile convergence and the usage of more data driven applications. While this provides a great opportunity for operators and Telecommunications Equipment Manufacturers (TEMs) to capture growth, it also presents significant technical and commercial challenges to be able to deliver on the increased expectations. Long Term Evolution (LTE) is one promising means to address the challenges; however, it requires architectural changes in the channel card in order to meet the need for increased bandwidth and capacity at lower latencies, creating a tough technical problem.

Traditional 3G architectures use a co-processor approach with DSP devices and FPGAs that partitions the baseband processing functions across the available resources. For example, while most of the uplink may be done in multiple DSP devices, the turbo decoding and FFTs may be done in the FPGA due to the computational load. The issue with this architecture is that it does not scale adequately for LTE, as the interface between the DSP devices and FPGAs becomes a bottleneck. This is particularly true at the high end where one might have a 20 MHz LTE system with 2x2 or even 4x4 MIMO (multiple input multiple output).

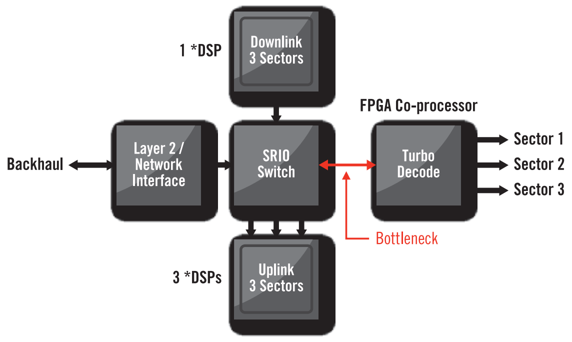


Figure 1: *Traditional 3G co-processor architectures do not scale to the increased demands of 4G*

The way to address this bottleneck is to implement a simplified sector based architecture where all the processing required for a single sector is done in a single device. This removes the bottleneck by not having to go off-chip to do all the baseband processing. Now the entire uplink, including the turbo decoding and FFTs, are done in a single chip. This architecture can also scale up to incorporate sophisticated MIMO functions. It is also more cost and power efficient than the co-processor architecture.

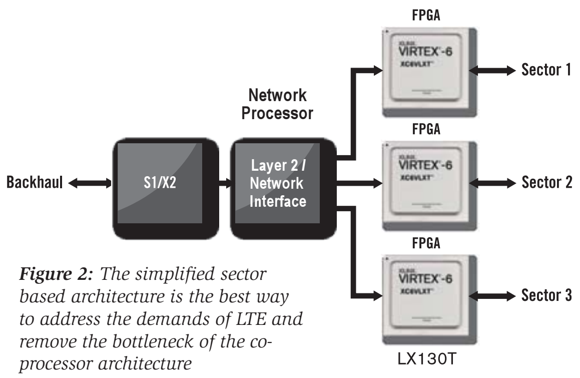


Figure 2: *The simplified sector based architecture is the best way to address the demands of LTE and remove the bottleneck of the co-processor architecture*

In order to reduce the development cost of moving to the simplified sector based architecture, Xilinx has developed an LTE Baseband Targeted Design Platform. Based on Xilinx's high-performance Virtex®-6 family of FPGAs, the LTE Baseband Targeted Design Platform provides comprehensive LTE uplink and downlink IP and reference designs that can be easily modified to meet a customer's specific requirements and areas of differentiation. They are also scalable to be cost optimized from a femtocell to a three sector macrocell. In addition, the IP and reference designs are rigorously tested to the LTE specification using test equipment and software from Xilinx's test partner, Agilent.

For more information and to watch a video demonstration on accelerating next generation LTE baseband designs with the Xilinx LTE Baseband Targeted Design Platform, visit www.xilinx.com/wireless.

About the Author: Manuel Uhm is the Director of Wireless Communications at Xilinx Inc. (San Jose, Calif.). Contact him at more_info@xilinx.com

Xilinx Tool & IP Updates

Xilinx continues to improve products and intellectual property in the ISE® Design Suite. Here are the most current updates for Xilinx design products and IP as of December 2009.

Product updates offer significant enhancements and new features to the ISE Design Suite. Keeping your installation of ISE up to date with these service packs is an easy way to ensure the best results for your design.

Updates to the ISE Design Suite are available from the Xilinx Download Center at www.xilinx.com/download.

For more information or to download a free 30-day evaluation of the ISE Design Suite, visit www.xilinx.com/ise.

ISE Design Suite: Logic Edition

Ultimate productivity for FPGA logic design
Latest version number: 11.4
Date of latest release: December 2009
Previous release: 11.3
URL to download the latest patch:
www.xilinx.com/download

New device support:
This latest release of the ISE Design Suite provides expanded support for both the Spartan®-6 and Virtex®-6 FPGA device families.

Spartan-6 FPGA support includes the following:

- Support for -4 speed grade
- Spartan-6 LX4 devices
- Spartan-6 lower power (-1L) devices
- Spartan-6 XA automotive devices

Virtex-6 FPGA support includes the following:

- Support for -3 speed grade for Virtex-6 HXT devices

Additional Improvements in the ISE Design Suite: Logic Edition 11.3

ChipScope™ Pro enhancements: A new parameter-sweep feature in IBERT 2.0 (Integrated Bit Error Ratio Test) for the Spartan-6 LXT allows channel testing that sweeps through various transceiver settings, enabling measurement of transceiver performance characteristics across that range of settings.

Core generation of IBERT 2.0 cores has been improved, providing interaction in the ChipScope Analyzer with GTX transceivers in the Virtex-6 HXT devices.

FPGA Editor enhancements: Improvements include a smaller memory footprint and faster design loading time.

PlanAhead™ design-analysis tool enhancements: PlanAhead offers a new simultaneous switching noise (SSN) prediction tool for I/O planning in the Virtex-6 family of FPGAs to help deliver improved signal integrity.

XST enhancements: The new MUX_MIN_SIZE constraint improves the device utilization for designs targeting the Virtex-6 and Spartan-6 FPGA families.

Xilinx Power Analyzer (XPA) enhancements: The Xilinx Power Analyzer now allows junction temperature calculation up to 125°C.

In addition, the Xilinx Power Estimator (XPE) spreadsheets have been updated. These product-specific spreadsheet-based power estimation tools deliver significant improvements in accuracy and ease of use when compared with other power estimation tools. Visit *www.xilinx.com/power* to learn more and to download these XPE spreadsheets.

Simulation libraries: This latest release of the ISE Design Suite includes Compxlib support for the new ModelSim DE product.

ISE Design Suite: Embedded Edition

An integrated software solution for designing embedded processing systems
Latest version number: 11.4
Date of latest release: December 2009
Previous release: 11.3
URL to download the latest patch:
www.xilinx.com/download

Revision highlights:
All ISE Design Suite Editions include the enhancements listed above for the ISE Design Suite: Logic Edition. The following are the enhancements specific to the ISE Design Suite: Embedded Edition.

Xilinx Platform Studio (XPS) and the Embedded Development Kit (EDK): Xilinx has enhanced the XPS and EDK to include the following:

- Base System Builder support for the Spartan-6 SP601 Base Board
- Flashwriter and GenACE support for the SP601 Base Board
- Base System Builder support for the Virtex-6 ML605 Base Board
- Flashwriter and GenACE support for the ML605 Base Board
- MicroBlaze™ updated to v7.20.d
- System Monitor support for the Virtex-6 FPGA

ISE Design Suite: DSP Edition

Flows and IP tailored to the needs of algorithm, system and hardware developers
Latest version number: 11.4
Date of latest release: December 2009
Previous release: 11.3
URL to download the latest patch:
www.xilinx.com/download

Revision highlights:
All ISE Design Suite Editions include the enhancements listed above for the ISE Design Suite: Logic Edition. The following enhancements are specific to the ISE Design Suite: DSP Edition.

New device support: System Generator for DSP now offers support for the Spartan-6 XA and Spartan-6-1L FPGA families. In addition, System Generator for DSP also provides support for the following new devices:

- Virtex-6 FPGA lower power (Virtex-6 -1L)
- Virtex-6 HXT FPGA
- Virtex-5Q FPGA

New JTAG hardware co-simulation support: System Generator for DSP now supports JTAG hardware co-simulation for the Spartan-6 FPGA SP605 Development Platform.

Blockset enhancements in System Generator for DSP: System Generator for DSP includes the Complex Multiplier 3.1 and DSP48 Macro 2.0 building blocks. In addition, the following forward error correction (FEC) blocks are included in the latest release of System Generator for DSP:

- Reed Solomon encoder 7.0
- Reed Solomon decoder 7.0
- Convolution encoder 7.0
- Viterbi decoder 7.0
- Interleaver/de-interleaver 5.1

Xilinx IP Updates

Name of IP: ISE IP Update 11.4
Type of IP: All

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

Latest version number: 11.4
Date of latest release: December 2009
URL to access the latest version:
www.xilinx.com/download

Informational URL: *www.xilinx.com/ipcenter/coregen/updates_11_4.htm*
Release notes: *www.xilinx.com/support/documentation/user_guides/xtp025.pdf*
Installation instructions: *www.xilinx.com/ipcenter/coregen/ip_update_install_instructions.htm*
Listing of all IP in this release: *www.xilinx.com/ipcenter/coregen/11_4_datasheets.htm*

Revision highlights:
Starting with 11.1, all ISE CORE Generator™ IP updates are bundled with quarterly ISE software updates. The latest versions of IP products have been tested and are delivered with the current IP release. There are a number of new cores in this release, as described below.

Communications and networking:

- 10 Gigabit Ethernet PCS/PMA (10GBASE-R; v1.1) – Provides XGMII interface to a 10-Gbit Ethernet MAC and implements a 10.3125-Gbps serial single-channel PHY, providing a direct connection to an XFP by means of the XFI electrical specification or SFP+ optical module using the SFI electrical specification.
- 3GPP LTE MIMO Decoder (v1.0) – Offers a highly resource-optimized and scalable MIMO decode function for 3GPP-LTE basestations. A graphical user interface allows the designer to select product parameters according to the needs of the application.
- Peak Cancellation Crest Factor Reduction (PC-CFR; v2.0) – Reduces implementation time by providing a high-performance CFR solution to customers as a parameterizable core, rather than one that needs design by hand, to potentially meet a multitude of wireless standards and performance criteria.

Audio, video and image processing:

- Image Edge Enhancement (v1.0) – Enhances the edges of objects using a set of programmable Sobel and Laplacian filters to increase the image contrast in the areas immediately around an edge. This improves the apparent sharpness of a frame of video.
- Image Noise Reduction (v1.0) – Provides a real-time filtering function that is edge adaptive to reduce the filter strength near edges so as to preserve sharpness. The filtering reduces the noise present in a frame of video.
- Image Statistics Engine (v1.0) – Implements the computationally intensive metering functionality common in digital cameras, camcorders and imaging devices. This core generates a set of statistics for color histograms, mean and variance values, edge and frequency content for 16 user-defined zones on a per-frame basis.
- Motion Adaptive Noise Reduction (v1.0) – Makes it possible to use the motion-detection function independently of the noise-reduction function for applications where noise reduction is not needed. The noise-reduction algorithm uses a recursive temporal filter with a programmable transfer function, allowing the user to control both the shape of the motion transfer and the strength of the noise reduction applied. The motion-transfer function is also programmable at run-time.

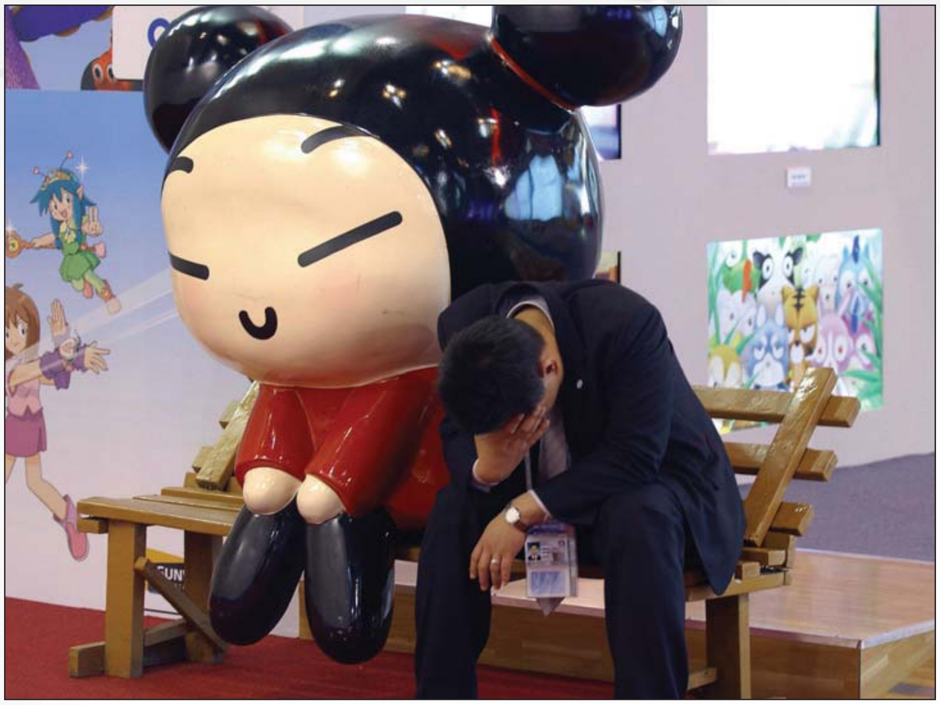
Digital signal processing (DSP):

- DSP48 Macro (v2.0) – Provides an easy-to-use interface that abstracts the XtremeDSP slice and simplifies its dynamic operation by enabling the specification of multiple operations via a set of user-defined arithmetic expressions. The user selects them via a single port on the generated core.

Standard bus interface:

- DisplayPort (v1.2) – Enables transmission of serial digital video up to 2.75 Gbps for consumer and professional displays. DisplayPort is a high-speed serial interface standard that replaces DVI and HDMI outside and LVDS inside the box for higher resolution (>FHD), higher frame rate and color bit depth display.

Xpress Yourself in Our Caption Contest

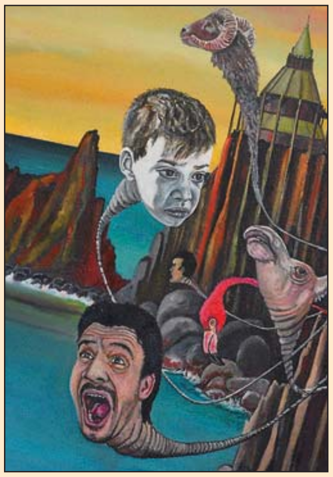


If you have a yen to Xercise your funny bone, here's your opportunity. We invite readers to step up to our verbal challenge and submit an engineering- or technology-related caption for this evocative photograph of Pucca, the kick-butt noodle shop delivery girl character who is all the rage in Asia, alongside a less-than-happy business-suited companion. The image might inspire a caption like "John's 3D virtual reality experiments proved successful—but exhausting."

Send your entries to *xcell@xilinx.com*. Include your name, job title, company affiliation and location. After due deliberation, we will print the submissions we like the best in the next issue of *Xcell Journal* and award the winner the new Xilinx® SP601 Evaluation Kit, our entry-level development environment for evaluating the Spartan®-6 family of FPGAs (approximate retail value, $295; see *http://www.xilinx.com/sp601*). Runners-up will gain notoriety, fame and a cool, Xilinx-branded gift from our SWAG closet.

The deadline for submitting entries is 5:00 pm Pacific Time (PT) on April 1, 2010. So, get writing!

MIKE BLANKENSHIP, the engineering manager at Golftek, won an SP601 Evaluation Kit with this caption for the strange, Dali-like image in Issue 69 of *Xcell Journal*:



"Jerry's last thought before drifting off to sleep was 'You'll be sticking your neck out if you tell management that the project schedule is unrealistic.' "

Congratulations as well to our two runners-up:

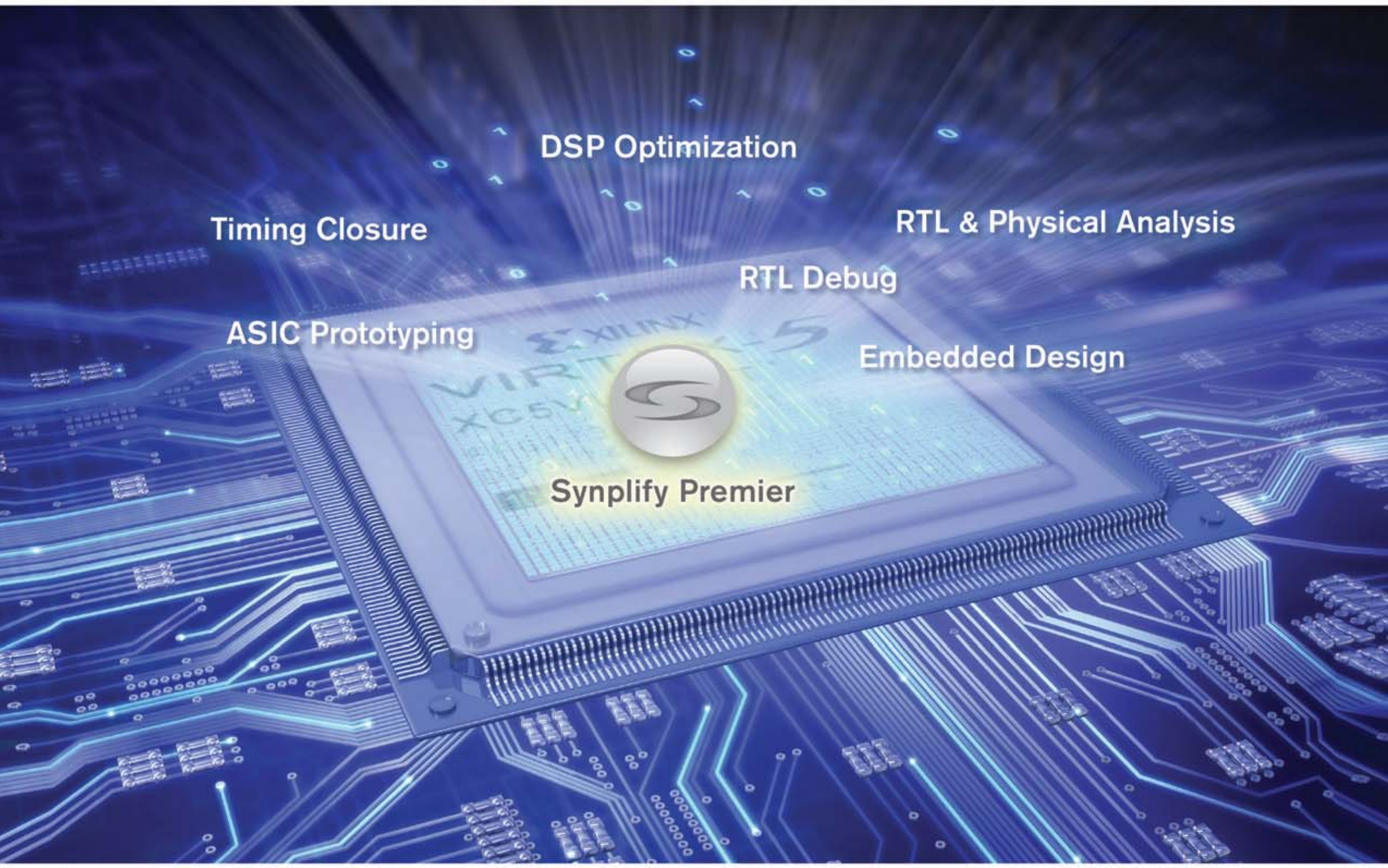
"Scott here, Captain. I'm having a wee bit of trouble with the new transporter controller. The Xilinx one worked better."

– Kirk Hobart, engineer, R2Sonic LLC

"No need to look further! Get real and stretch out of your nightmare, try the Xilinx SP601 Evaluation Kit."

– Sam Sortais, engineer, Ericsson

Accelerate FPGA Design



Synplify Premier, the Ultimate in FPGA Implementation

The Synplify® Premier software from Synopsys® is the preferred synthesis and debug environment for complex FPGA designs. It provides a comprehensive suite of tools and technologies for advanced FPGA designers as well as ASIC prototypers targeting a single FPGA. The Synplify Premier solution addresses the biggest FPGA design challenges including timing-closure, logic verification, IP support, ASIC compatibility, DSP implementation, and RTL debug while providing tight integration with Xilinx® back-end tools. Synplify Premier supports DesignWare® components and performs detailed logic placement which is passed on to the Xilinx router for final implementation. With final placement knowledge during synthesis, design iterations are significantly reduced resulting in shorter development schedules.

To learn more about how the Synplify Premier software
can help you achieve your design goals, visit
http://www.synplicity.com/synplifypremier/xcell_premier.html

Synplicity®
Simply Better Results

SYNOPSYS®
Predictable Success

XILINX.

ACCELERATE CHANGE. ADVANCE INNOVATION.

The power to innovate is in your hands. Focus your energy on innovation and add differentiating value to your electronic systems. Xilinx Targeted Design Platforms integrate advanced FPGA silicon and design tools, IP cores, development boards, and reference designs—giving you the freedom to innovate without risk. Find out more at **www.xilinx.com**

PN 2447