Issue 2
May 2006

DSP magazine

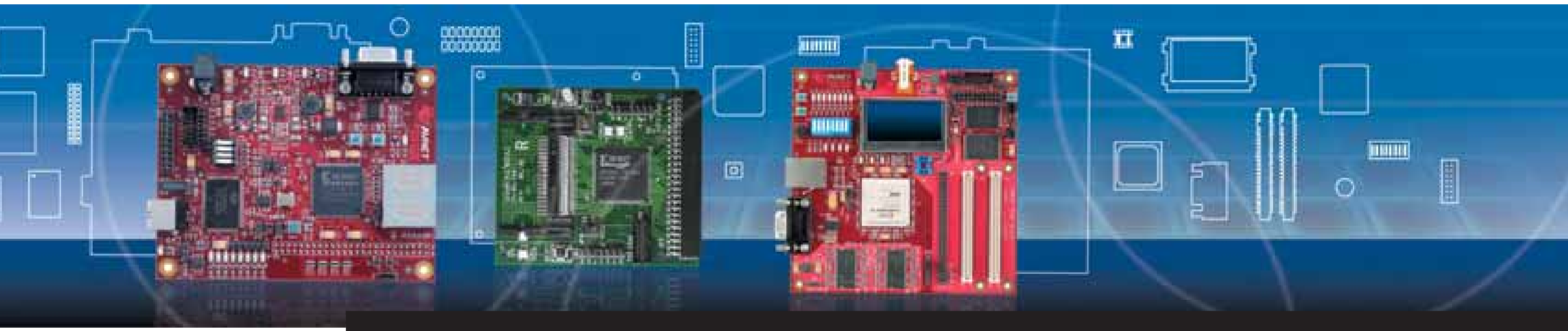
SOLUTIONS FOR HIGH-PERFORMANCE SIGNAL PROCESSING DESIGNS

Optimizing DSP System Designs

INSIDE

XILINX®

Support Across The Board.™



Design Kits Fuel Feature-Rich Applications

Build your own system by mixing and matching:

- Processors
- FPGAs
- Memory
- Networking
- Audio
- Video
- Mass storage
- Bus interface
- High-speed serial interface

Available add-ons:

- Software
- Firmware
- Drivers
- Third-party development tools

Avnet Electronics Marketing designs, manufactures, sells and supports a wide variety of hardware evaluation, development and reference design kits for developers looking to get a quick start on a new project.

With a focus on embedded processing, communications and networking applications, this growing set of modular hardware kits allows users to evaluate, experiment, benchmark, prototype, test and even deploy complete designs for field trial.

Gain hands-on experience with these design kits and other development tools by participating in a SpeedWay Design Workshop™ this spring.

For a complete listing of available boards, visit
www.avnetavenue.com

For more information about upcoming SpeedWay workshops, visit
www.em.avnet.com/speedway

XILINX®

AVNET® electronics marketing

Avnet Green Initiative

Supplier Authorized Distributor

Enabling success from the center of technology™

1 800 332 8638
em.avnet.com

DSP magazine

PUBLISHER	Forrest Couch forrest.couch@xilinx.com 408-879-5270
EDITOR	Charmaine Cooper Hussain
ART DIRECTOR	Scott Blair
ADVERTISING SALES	Dan Teie 1-800-493-5551
TECHNICAL COORDINATOR	Narinder Lall
INTERNATIONAL	Dickson Seow, Asia Pacific dickson.seow@xilinx.com Andrea Barnard, Europe/Middle East/Africa andrea.barnard@xilinx.com Yumi Homura, Japan yumi.homura@xilinx.com

www.xilinx.com/xcell/

Xcell PUBLICATIONS

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

High-Performance DSP – Executing to Plan

Welcome to the second edition of Xilinx® *DSP Magazine*. In the last issue we outlined five pillars that underline our vision in DSP: market focus, design methodology, tailored solutions, ecosystem, and awareness. Since publishing that issue, we have delivered many exciting tailored solutions, such as starter and co-processing kits for video and imaging and JTRS development platforms for software-defined radio.

With the acquisition of AccelChip and its MATLAB-to-RTL synthesis tools, we have also increased our investment in providing you with the most capable and easiest to use design methodology solutions. We are also continuing to work with our core DSP partners like Texas Instruments and The MathWorks to deliver complementary solutions, as unveiled in our recent Serial RapidIO interoperability announcement at TI's Developer Conference in February.

For this second edition of *DSP Magazine* we welcome DSP industry icon Will Strauss of Forward Concepts with snippets from his latest DSP industry research report. His insights regarding shifts in the DSP industry are provided in his article, "FPGAs for DSP: A Fast-Growing Market." In addition, our partners Avnet, Lyrtech, The MathWorks, and Nuvation highlight their latest innovations for our XtremeDSP™ platforms. Our own experts provide tutorials on implementing floating-point DSP, optimizing filter design, and achieving high-bandwidth simulations, among others.

I'm sure you'll find our second edition of *DSP Magazine* informative and inspiring as we endeavor to help you unlock the full capabilities of Xilinx reconfigurable signal processing. Enjoy the read!

Omid Tahernia

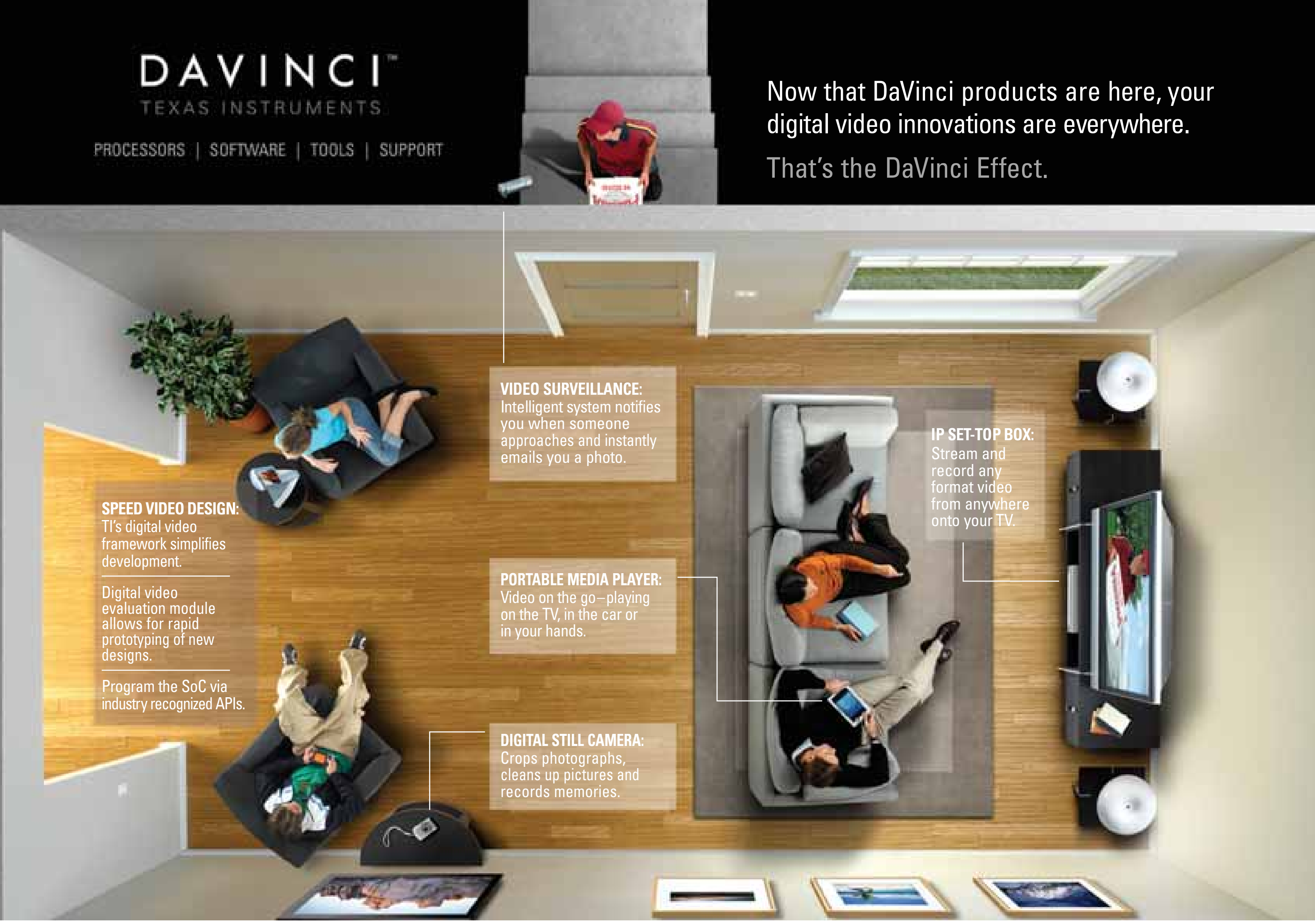
Vice President
and General Manager
Xilinx DSP Division

DAVINCI™
TEXAS INSTRUMENTS

PROCESSORS | SOFTWARE | TOOLS | SUPPORT

Now that DaVinci products are here, your digital video innovations are everywhere.

That's the DaVinci Effect.



VIDEO SURVEILLANCE: Intelligent system notifies you when someone approaches and instantly emails you a photo.

IP SET-TOP BOX: Stream and record any format video from anywhere onto your TV.

SPEED VIDEO DESIGN: TI's digital video framework simplifies development.

Digital video evaluation module allows for rapid prototyping of new designs.

Program the SoC via industry recognized APIs.

PORTABLE MEDIA PLAYER: Video on the go–playing on the TV, in the car or in your hands.

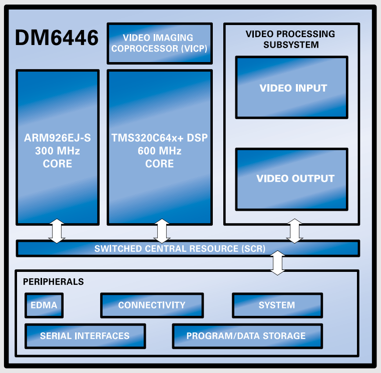
DIGITAL STILL CAMERA: Crops photographs, cleans up pictures and records memories.

DaVinci™ Technology makes astounding creativity possible in digital video devices for the hand, home and car. The DaVinci platform includes digital signal processor (DSP) based SoCs, multimedia codecs, application programming interfaces, application frameworks and development tools, all of which are optimized to enable innovation for digital video systems. DaVinci products will save OEMs months of development time and will lower overall system costs to inspire digital video innovation. So what are you waiting for? You bring the possibilities. DaVinci will help make them real.

What is DaVinci?

Processors: Digital Video SoCs:

- TMS320DM6446 – Video encode/decode
- TMS320DM6443 – Video decode



Performance Benchmarks:

STANDALONE CODECS	DM6446	DM6443
MPEG-2 MP ML Decode	1080i+ (60 fields /30 frames)	720p+
MPEG-2 MP ML Encode	D1+	n/a
MPEG-4 SP Decode	720p+	720p+
MPEG-4 SP Encode	720p+	n/a
VC1/WMV 9 Decode	720p+	720p+
VC1/WMV 9 Encode	D1+	n/a
H.264 (Baseline) Decode	D1+	D1+
H.264 (Baseline) Encode	D1+	n/a
H.264 (Main Profile) Decode	D1+	D1+

+ denotes available processor headroom for analytics and/or other features

Tools: Validated Software and Hardware Development

- DVEVM (Digital Video Evaluation Module)
- MontaVista Development Tools
- Code Composer Studio IDE

Software: Open, Optimized and Production Tested

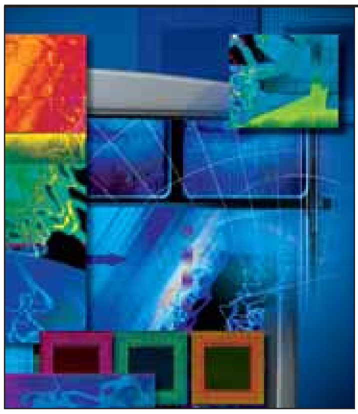
- Platform Support Package
- MontaVista Linux Support Package
- Industry-recognized APIs
- Multimedia frameworks
- Platform-optimized, multimedia codecs:

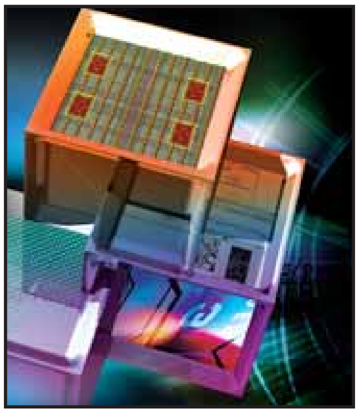
- H.264	- AAC	- G.729ab
- MPEG4	- WMA9	- WMV9/ VC1
- H.263	- MP3	
- MPEG2	- G.711	
- JPEG	- G.728	
- AAC+	- G.723.1	

>>> For complete technical documentation or to get started with our Digital Video Evaluation Module, please visit **www.thedavincieffect.com**

Portable Media Player | IP Set-Top Box | Automotive Infotainment | Digital Still Camera | Digital Video Innovations | Video Surveillance | Video Phone & Conferencing

Technology for Innovators™ TEXAS INSTRUMENTS





CONTENTS

FPGAs for DSP: A Fast-Growing Market

A survey by Forward Concepts validates the increasing role of FPGAs in DSP applications.

by Will Strauss
President
Forward Concepts
wis@fwdconcepts.com

FPGAs employed for DSP have passed the half-billion dollar mark; in fact, that market segment is growing faster than the larger and more mature DSP chip market. The reasons are varied, but performance is the prime driver, as FPGAs easily outdistance conventional DSP chips in maximum bandwidth and the number of communication channels or video streams that can be processed simultaneously.

As FPGAs have become more powerful and cheaper through advanced CMOS processing, stand-alone FPGA DSP solutions are becoming practical. In a recent survey of more than 300 DSP professionals from 30 countries, Forward Concepts asked, "Which chip types are employed for DSP algorithm execution (rather than data processing) in your applications"? The results comparing DSPs and FPGAs in Figure 1 clearly show that FPGAs have an increasing and varied role in DSP.

As expected, the general-purpose (GP) fixed-point DSP garnered the most mentions, followed by GP floating-point DSPs. But significantly, stand-alone FPGAs for DSP showed strength in the number of responses garnered, equaling the number of responses for FPGAs paired with DSPs as an accelerator. Surprisingly, FPGAs paired with RISCs also showed significant strength. This provides an appropriate segue to our next chart.

We also asked our survey participants, "If a RISC core is employed (for any purpose) in your DSP application, which brand(s) is (are) used or being strongly considered for your next design"? As expected, of those respondents employing a RISC core, ARM, Ltd. topped the responses. Unexpected, though, was the strong response for soft RISC cores, with Xilinx® MicroBlaze™ and PicoBlaze™ processors ranking a clear second to ARM in popularity, as indicated in Figure 2.

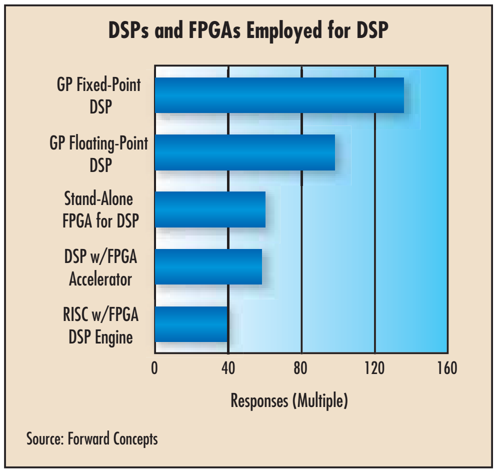


Figure 1 – FPGAs exhibit growing roles in DSP.

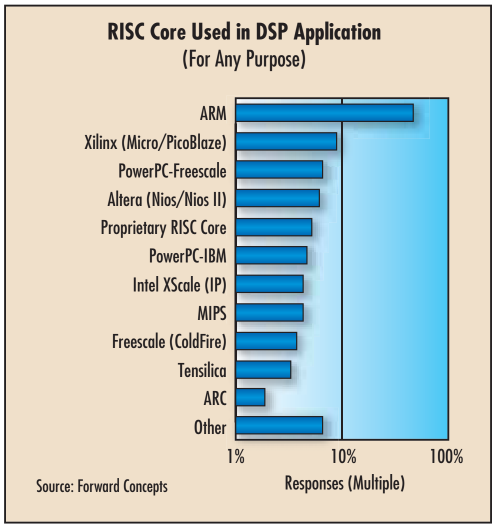


Figure 2 – FPGA soft RISCs prove popular in DSP applications.

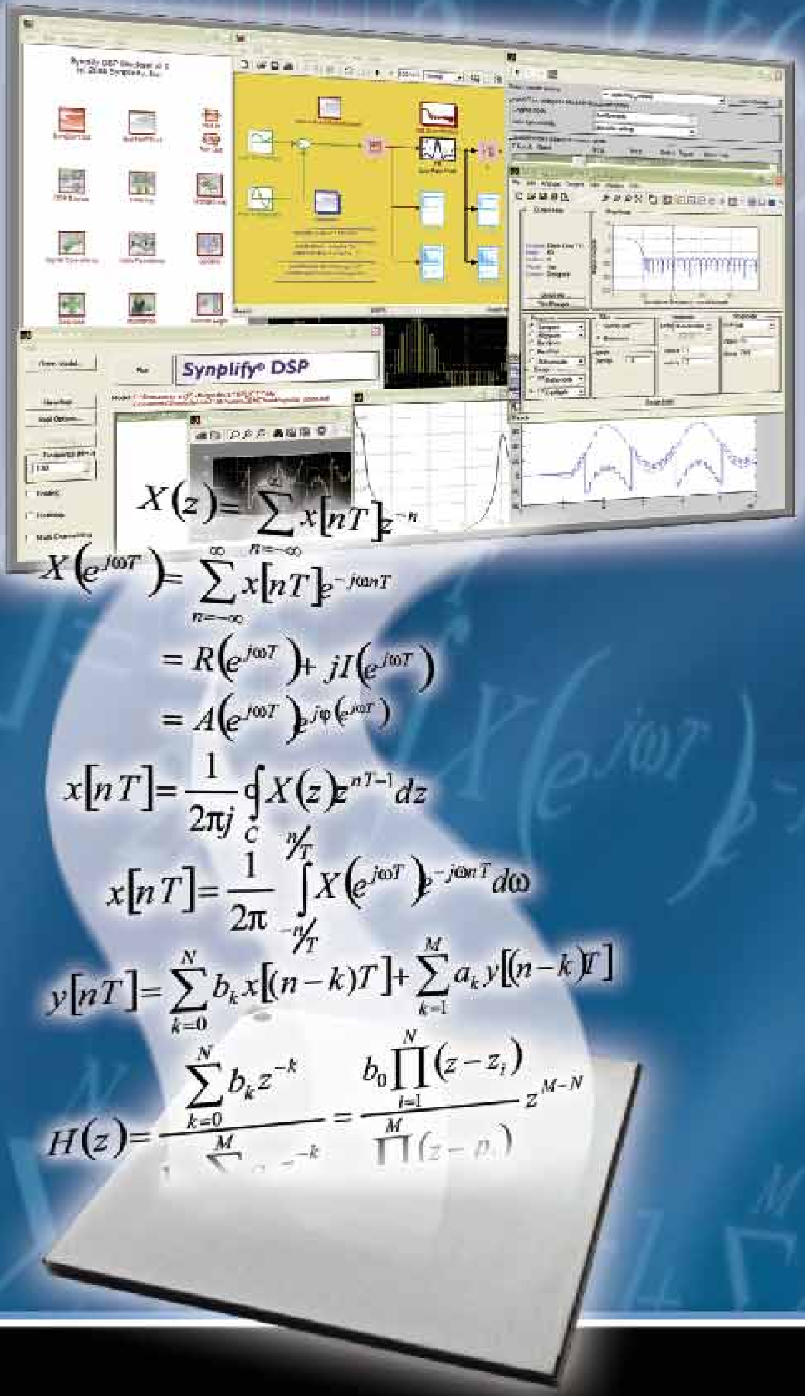
Although we did not explicitly ask if a soft RISC was employed in the "stand-alone FPGAs for DSP" in Figure 1, we can conclude that is probably the case for such implementations. Considering that the soft RISC is more concerned with control code and that the FPGA array can be significantly devoted to DSP functions, the pairing makes a lot of sense. Moreover, our informal interviews after the survey was performed (Q4/05) revealed that there are a number of FPGA implementations employing multiple soft RISC cores, with MicroBlaze processors mentioned most often.

It is clear that FPGAs have a strong play in the world of DSP and Xilinx is aggressively providing new products and application support to meet increasing market demand for ever-higher performance DSP products in communications and professional multimedia.

The charts in this article are excerpts from Forward Concepts's new 322-page market study, "DSP Strategies: Embedded Chip Trend Continues" (www.fwdconcepts.com).

Synplify® DSP

A Breakthrough in DSP Synthesis



Synplicity's Synplify DSP synthesis solution offers DSP designers an efficient automated path from Simulink® to optimized RTL code. And, unlike other solutions, Synplify DSP software provides technology independent device targeting and advanced synthesis optimizations for performance and area (cost).

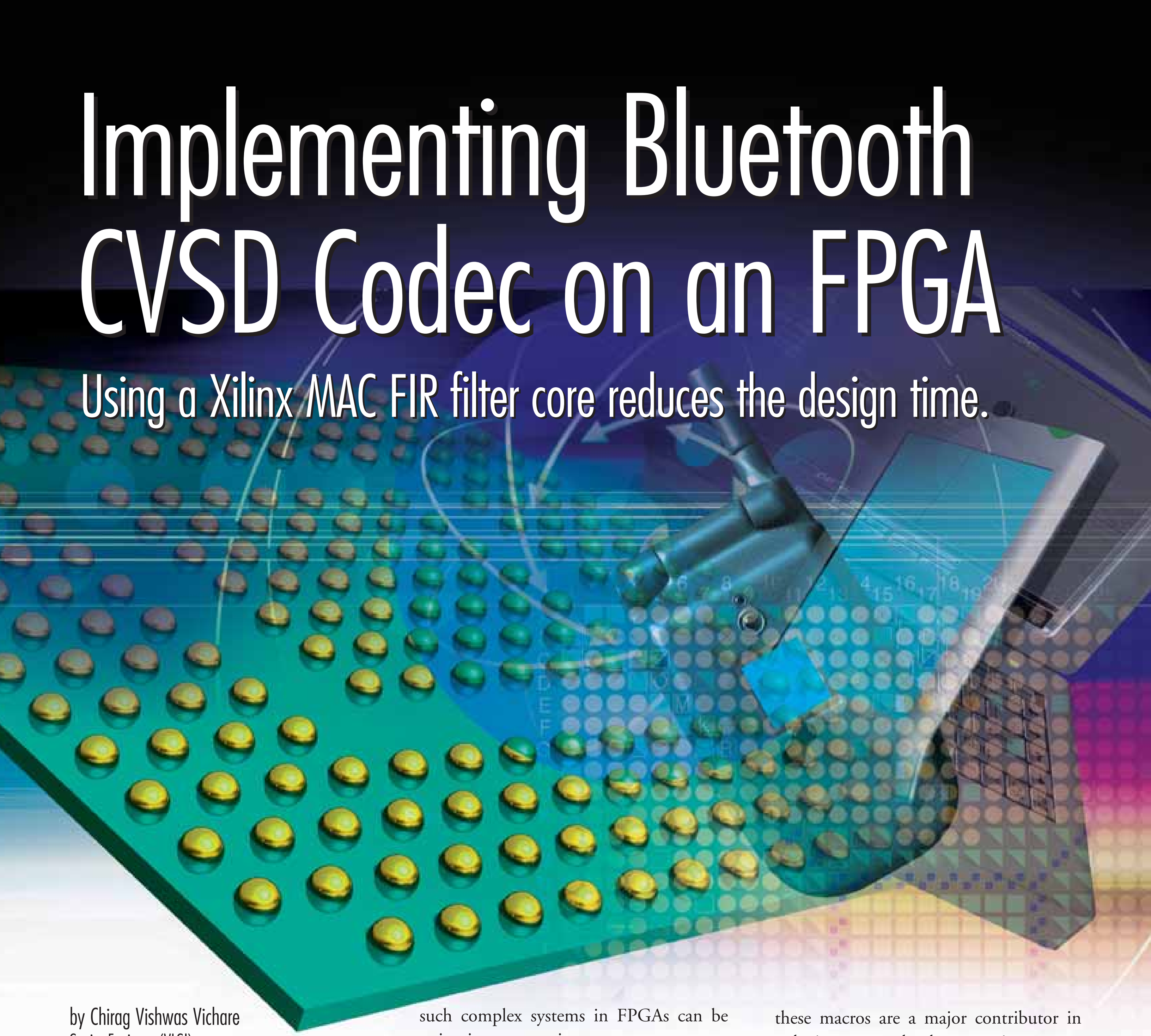
Synplify DSP Software Uniquely Offers:

- A User-extensible DSP library: quickly create and re-use custom functions
- Sample-rate-based synthesis engine: capture and explore algorithm behavior without worrying about target hardware
- Retiming: automatic pipelining and register insertion for higher speeds
- Folding: automatically apply resource sharing logic for lower area (device cost)
- Multichannelization: automatically apply resource sharing across multiple channels for lower area (device cost)

For more information on Synplicity's Synplify DSP solution and all of Synplicity's offerings, please visit our website at www.synplicity.com or contact info@synplicity.com

Synplicity®

Simply Better Results



Implementing Bluetooth CVSD Codec on an FPGA

Using a Xilinx MAC FIR filter core reduces the design time.

by Chirag Vishwas Vichare
Senior Engineer (VLSI)
MindTree Consulting Pvt. Ltd.
chirag_vichare@mindtree.com

Over last few years, FPGAs have evolved to include significant DSP-centric enhancements in their architectures. These enhancements have enabled FPGAs to support many complex DSP applications in domains such as telecommunications (base station signal processing, radar signal processing), multimedia processing (video processing, audio signal processing), and other application areas. However, implementing such complex systems in FPGAs can be quite time-consuming.

In most of these DSP systems, the algorithms used were developed for processor-based systems. Translating these algorithms into hardware to achieve equal or better performance can take months, compared to few weeks on DSP processors. This is largely attributed to the mature development tools available to DSP engineers while working with DSP processors such as TI or analog devices. Usually these tools provide optimized macros (in high-level language or assembly language) for many DSP algorithms, and these macros are a major contributor in reducing system development time.

Including More System-Level IP Cores

To achieve similar results in terms of development cost, time, and performance when implementing such complex DSP systems in hardware, it is just not enough to have only primitive macros such as adders and multipliers already available in a hardware designer's library. The hardware DSP design flow should incorporate more and more system-level macros for sub-blocks such as encoders, decoders, FFTs, DCTs, trigonometric

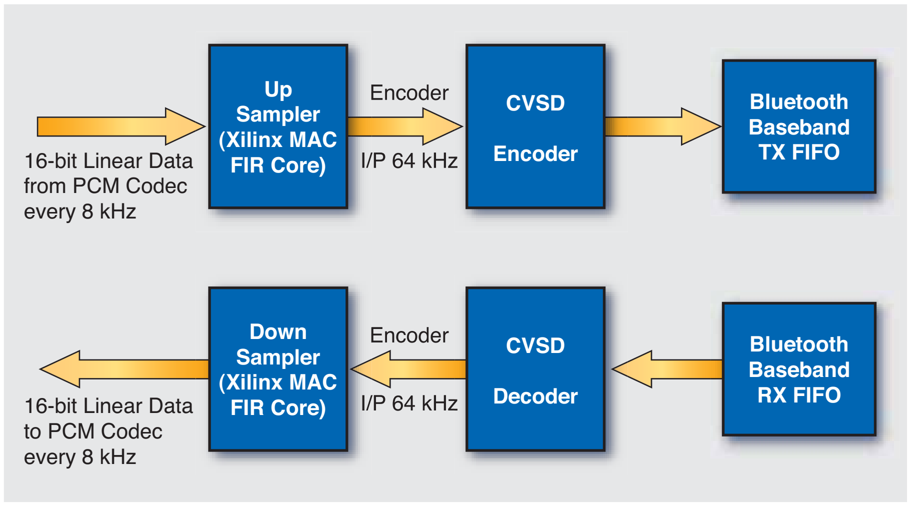

16-bit Linear Data from PCM Codec every 8 kHz
Up Sampler (Xilinx MAC FIR Core)
Encoder I/P 64 kHz
CVSD Encoder
Bluetooth Baseband TX FIFO
16-bit Linear Data to PCM Codec every 8 kHz
Down Sampler (Xilinx MAC FIR Core)
Encoder I/P 64 kHz
CVSD Decoder
Bluetooth Baseband RX FIFO

Figure 1 – Bluetooth CVSD codec block diagram

functions, sample rate converters, and FIR filters, which are optimal implementations in terms of area, memory, or speed and already proven in hardware. This can reduce development time drastically, as the only task is to integrate these IP cores into the system architecture and verify the functionality of the overall system. Even if your goal is to finally develop your own macros and you are using the FPGA only as an ASIC prototype or demonstration platform, using these macros to validate your design can give you the opportunity and time to explore the design space in terms of area and performance (speed and power).

To illustrate the methodology, let's use as an example a MindTree bluetooth CVSD codec implemented on a Xilinx® Virtex™-II FPGA, which uses a CORE Generator™ software MAC FIR filter as a sub-block for interpolation and decimation filtering.

The aim here was to validate the continuous variable slope delta modulation (CVSD) codec on a Virtex-II-based ASIC prototype platform for MindTree's Bluetooth baseband.

CVSD Codec Validation

Bluetooth technology has reached a stage of maturity and is being extensively integrated in many devices such as mobile phones, PDAs, laptops, and GPS receivers. One of the major applications for Bluetooth in these devices is as a carrier of voice data over synchronous logic transport (SCO). The Bluetooth standard specifies a 64 kbps log PCM (pulse code modulation) format (A-law or μ-law) or a 64 kbps CVSD format for on-the-air interface. CVSD encoding is considered the more robust format for voice-over-the-air interfaces. However, CVSD is a complex technique, which involves adaptive delta pulse-code modulation, where only two levels are used to represent the differential in amplitude (delta). The CVSD encoder/decoder processes 16-bit samples (as an encoder) and single-bit symbols at 64 kHz (as a decoder).

Figure 1 shows the block diagram of the Bluetooth CVSD codec.

The CVSD codec requires an interpolation filter in the encoder path to upsample the 8 KHz samples from voice codec to 64 kHz, which is given as an input to the CVSD encoder. Similarly, the 64 kHz output of the decoder must be down-sampled to 8 kHz using a low-pass filter with negligible spectral power density (above 4 kHz) before being played back with the 8 kHz voice codec.

After verifying the CVSD encoder/decoder block independently with the golden reference model, I took the approach of validating the CVSD codec on the Virtex-II FPGA platform, with the Xilinx FIR filter core used as an interpolation and decimation filter.

Integrating these filters into the MindTree Bluetooth baseband core was a fairly easy task, as the MAC FIR I/O interface is clearly defined in Xilinx CORE Generator software.

The only task was then to generate filter coefficients for interpolation and a decimation filter from SCILAB (a free scientific software package for numerical computations), and pass them to Xilinx CORE Generator software to produce an .edn file for filters, which was included during place and route.

Using a Xilinx MAC FIR filter core helped in exploring the optimal set of parameters (filter length, coefficients, word-length precision) for these filters without compromising CVSD voice quality. It also helped in terms of reduced design time by avoiding the design iterations to achieve the desired voice quality. This is largely because the design had already been proven in hardware much earlier in design cycle. It also helped in hardware optimization during actual implementation of interpolation and decimation filters.

An Alternative Approach

CORE Generator software also provides a behavioral model for all of its IP cores. You can integrate these models into the system to be developed during the verification stage, which can aid in analyzing the performance and fine-tuning the design parameters.

Conclusion

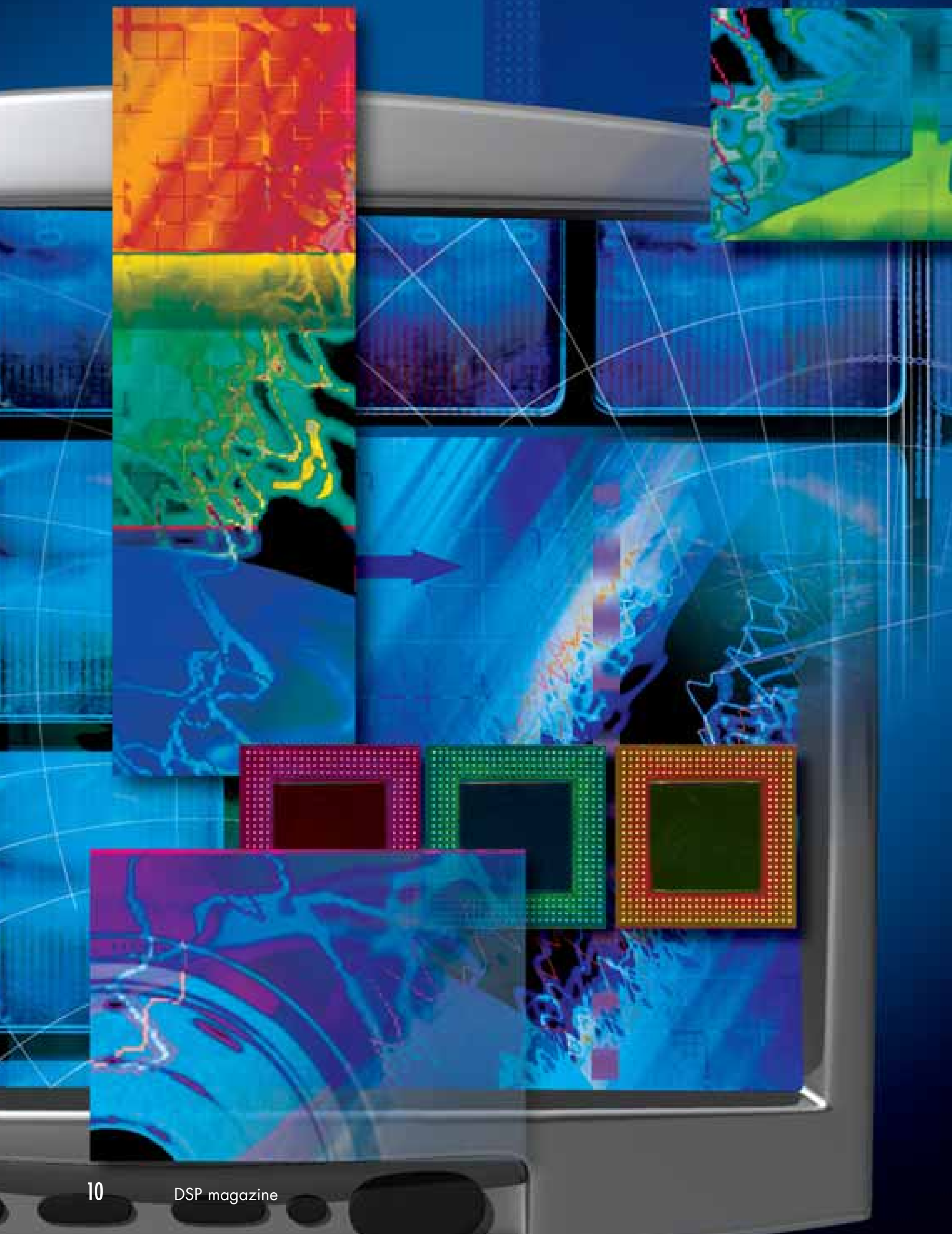
Today's FPGAs are capable of implementing most high-performance complex DSP algorithms efficiently. However, achieving your desired performance goals and meeting tight time-to-market constraints requires a change from conventional DSP design flows.

Incorporating pre-verified and more often optimized system-level IP cores at various stages in the DSP design flow can significantly help in reducing development time and achieving your desired performance objectives through design space exploration.

For more information on CORE Generator software, visit *www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=dr_dt_coregenerator* or *www.xilinx.com/ise/products/coregen_overview.pdf*.

Developing Video IP in a Fully Integrated Design Environment

The Video Starter Kit is an ideal prototyping platform for multimedia, video, and imaging.



by Sabine Lam
DSP Technical Marketing Engineer
Xilinx, Inc.
sabine.lam@xilinx.com

Often the implementation of video processing systems requires support for various video and audio standards and involves converting signals from one standard to another. Multimedia applications require processing signals at video rates, which means that simulation should run in real time during the development process.

Typical video processing systems use a microprocessor to control a video pipeline comprising a video source and sink, a large memory for storage of video data, and a video processing system (Figure 1).

As you implement and debug the various video algorithms, you will need to verify the functionality through software and hardware simulation. Simulation of video-processing applications creates special challenges given the real-time nature of video streams and the enormous amount of video data required per frame.

Design Environment

The Video Starter Kit (VSK) enables rapid development and debugging of high-performance video processing systems for a wide range of video applications. The VSK is powered by the Xilinx® Virtex™-4 XC4VSX35 device, which is optimized for DSP processing thanks to the high ratio of multiply accumulate blocks (also known as DSP48) in the fabric and supported by a rich feature set of video interfaces such as DVI, VGA, component (HD), composite, S-Video, and SDI.

Typically, developing video algorithms requires hardware to prove the video operation on real-time data streams, as well as a simulation environment to develop and test the video processing components. The VSK provides both software simulation and real-time operation for each of the components in a video system, allowing you to develop video IP (including filters, video blocksets, accelerators, and video interface conversion) or end applications such as codecs, image enhancement, dynamic gamma correction, and motion estimation. The integration with the tool kit and the I/O diversity makes it fast and easy to get video onto the board and optimize algorithms to operate on it.

Also provided with the VSK are reference designs, some in HDL and others modeled in the Xilinx System Generator for DSP design environment. To abstract away the complexity of bringing data in through the various video interfaces and sending them down to the Virtex-4 device, a library of video interface blocksets is included, all controlled through a MicroBlaze™ controller.

To highlight some of the VSK capabilities, I'll describe the MPEG-4 Part 2 decoder demonstration design.

MPEG-4 Part 2

The MPEG-4 decoder demonstration system comprises an FPGA hardware evaluation platform, Xilinx IP cores, and embedded software operating together to perform video decompression on industry-standard encoded video bitstreams.

For this design, the FPGA is programmed to perform the decompression and drive the video display. A Compact Flash card holds several compressed video streams and the FPGA configuration bitstreams. An embedded processor within the FPGA reads the bitstream out of the Compact Flash card, writes it to an external DDR memory, and sends it to the MPEG-4 Part 2 decoder. The output from the decoder is then reformatted to the video standard of your choice for display on an external monitor through the video I/O daughtercard.

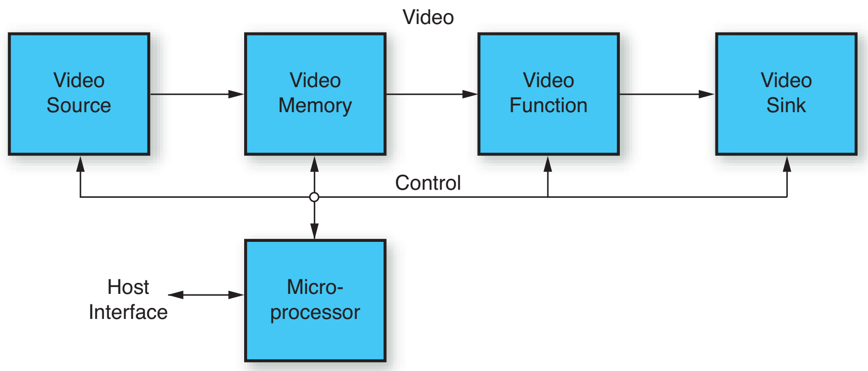


Figure 1 – Video system diagram

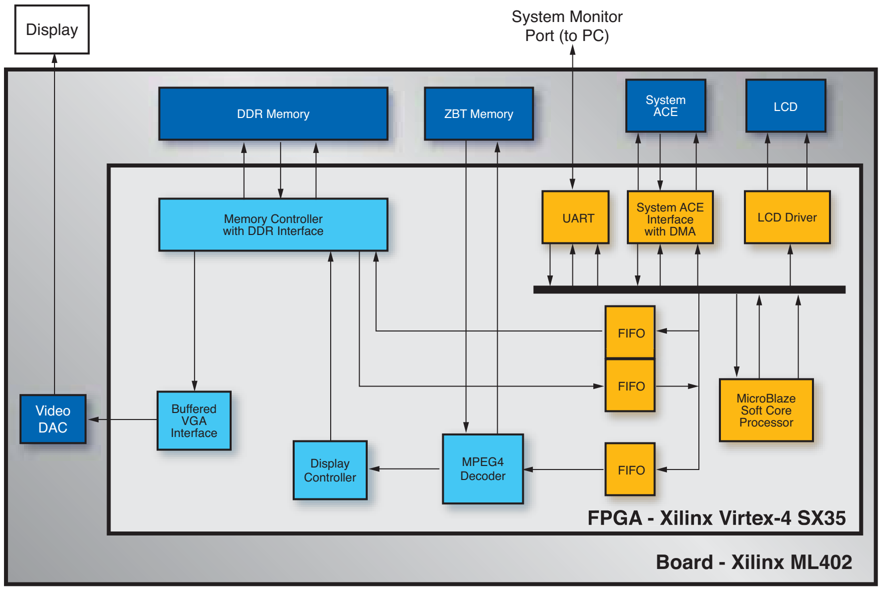

Display
System Monitor Port (to PC)
DDR Memory
ZBT Memory
System ACE
LCD
Memory Controller with DDR Interface
UART
System ACE Interface with DMA
LCD Driver
FIFO
FIFO
MicroBlaze Soft Core Processor
Video DAC
Buffered VGA Interface
Display Controller
MPEG4 Decoder
FIFO
FPGA - Xilinx Virtex-4 SX35
Board - Xilinx ML402

Figure 2 – MPEG-4 design overview

An overview of the system is shown in Figure 2. The MPEG-4 decoder core, DDR memory controller, color space converter, VGA interface, macroblock format converter, and MicroBlaze soft-core processor and associated peripherals are implemented in the XC4VSX35 FPGA. The ZBT memory, DDR memories, System ACE™ technology, Compact Flash connector, two-line LCD display, and a digital-to-analog converter are located on the hardware platform.

Embedded Processor

Video systems often require a control processor. The processor is typically used to communicate with a host system, set up video processing operations, compute coefficients, and generally operate as a low-rate data processor.

In the MPEG-4 demonstration design, the embedded MicroBlaze processor operates as the overall system-level controller,

The video standard input and output sources featured on the VSK, coupled with System Generator hardware co-simulation capability, allow you to quickly test and debug your system with real-time video streams.

handling such functions as the user interface, reading compressed bitstreams from Compact Flash, transmitting the bitstream into the MPEG-4 decoder core, and monitoring all system status flags.

With Xilinx System Generator for DSP, the design flow for incorporating a MicroBlaze processor into the framework is greatly simplified. You can use Xilinx System Generator and the Embedded Development Kit (EDK) software tools together to implement and simulate a system with a processor and FPGA video processor functions operating on live video streams. System Generator automatically generates software drivers to read and write data to the System Generator design.

Two methodologies are currently supported to integrate a MicroBlaze controller:

- A System Generator design exported into an EDK system. When used in pcore (processor core) export mode, the memory map block and all other blocks are packaged into a pcore peripheral. Software drivers and documentation for the memory-map interface are also generated and delivered with the peripheral.
- An EDK project imported into a System Generator design for hardware co-simulation. When used in EDK import mode, an EDK project file is imported into System Generator by running the EDK import wizard. When the import wizard is completed, the EDK system is pulled into the System Generator design as a black box. During the import process, the EDK system is augmented with Fast Simplex Link (FSL) interfaces that communicate with the memory map.

Hardware Co-Simulation

Viewing the resulting output video is an important quality measurement metric for all video systems. The video standard input and output sources featured on the VSK, coupled with System Generator hardware co-simulation capability, allow you to quickly test and debug your system with real-time video streams.

System Generator provides hardware co-simulation interfaces that make it possible to compile a System Generator diagram into an FPGA bitstream and associate this bitstream with a new run-time hardware co-simulation block. When the design is simulated in Simulink, results for the compiled portion are calculated in hardware instead of software.

System Generator provides high-speed hardware co-simulation interfaces that allow the full contents of a Simulink vector or matrix signal to be read from or written to FPGA hardware in a single transaction. By using these interfaces, you can significantly reduce the number of PC/hardware transactions during simulation and further accelerate simulation speeds beyond what is traditionally possible with hardware co-simulation.

By taking advantage of the ubiquity and advancement of Ethernet technologies, the interface facilitates a convenient and high-bandwidth co-simulation to an external FPGA device.

The VSK supports two Ethernet co-simulation modes:

- The network-based Ethernet hardware co-simulation interface provides co-simulation access to an FPGA platform over an IPv4 network infrastructure. Because IPv4 networks are widespread, the interface provides a straightforward way to communicate with remote hardware connected to either a wired or wireless network. This interface is ideal in situations where the FPGA platform is remote (such as across the office or across the country) or when multiple designers must share a single development board. The network-based Ethernet interface supports operations in 10/100 Mbps half/full duplex modes.
- The point-to-point Ethernet hardware co-simulation provides a co-simulation interface using a raw Ethernet connection. The raw Ethernet connection refers to a Layer 2 (data link layer) Ethernet connection, between a supported FPGA development board and a host PC, with no routing network equipment along the path. The point-to-point Ethernet interface supports operations in 10/100/1000 Mbps half/full duplex modes. Jumbo frames are also supported on a Gigabit Ethernet connection, as long as it is enabled by the underlying connection.

Conclusion

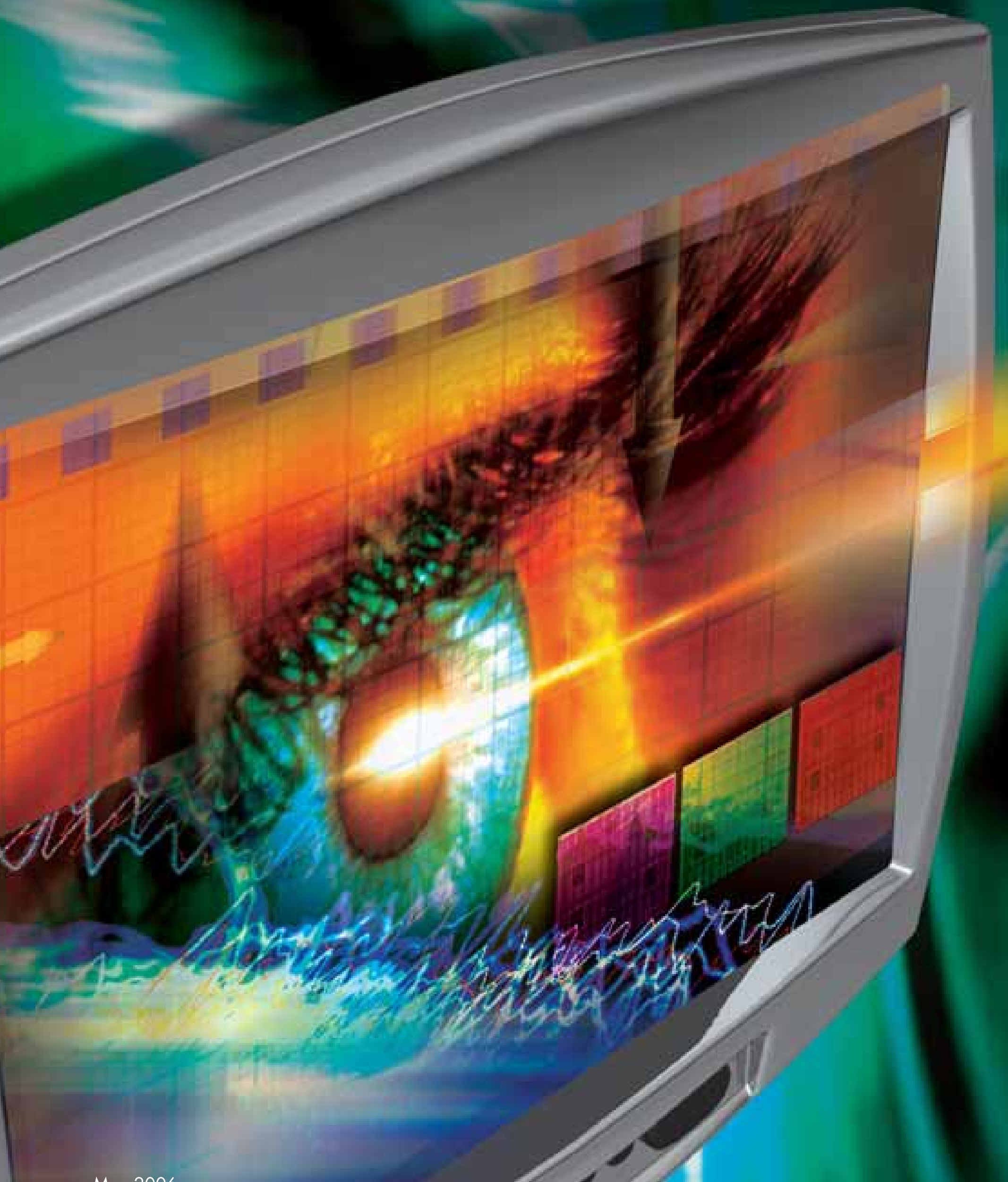
With this complete and easy-to-use solution, the Video Starter Kit is the ideal hardware platform to evaluate Xilinx FPGAs in a wide range of video and imaging applications. Fully integrated and supported by the Xilinx System Generator for DSP software, the VSK takes advantage of the new high-speed Ethernet hardware co-simulation capability and enables system integration, development and verification of codecs, IP, and video algorithms in real time.

The VSK comprises software, hardware, camera, cables, and a detailed users guide and reference designs. It includes a limited edition of System Generator for DSP, ISE™ software, and Embedded Design Kit (EDK) FPGA design tools, as well as a Xilinx ML402-SX35 development board, video I/O daughtercard (VIODC), CMOS image sensor camera, power supply, and cables.

For more information, see the VSK User Guide at *www.xilinx.com/bvdocs/userguides/ug217.pdf*, or, for the MPEG-4 demonstration design, *www.xilinx.com/bvdocs/userguides/ug234.pdf*.

Accelerate Video DSP Co-Processing Designs

Design, develop, and test your algorithms with the Video Virtual Socket Adapter.



by Chris Hallahan
VP Sales & Marketing
Nuvation
chris.hallahan@nuvation.com

Did you know that you can evaluate, test, develop, and benchmark custom video processing applications utilizing an appropriate mixture of FPGA and DSP processing? The new Xilinx® Video Virtual Socket Adapter (VSA) is designed to accelerate FPGA/DSP video co-processing development.

The VSA is a plug-and-play system comprising a Spectrum Digital DM642 EVM DSP evaluation board; a Spectrum Digital XEVM642 daughtercard; a VHDL "virtual socket" framework for the Virtex™-4 SX FPGA; DSP firmware; a System Generator for DSP demo module featuring a two-dimensional 5 x 5 Video FIR filter; user guide; application notes; and a PC-based network streaming MJPEG video player. With the bundled demo system, you instantly get the infrastructure to rapidly prototype your video applications to accelerate FPGA/DSP co-processing product development. The system was developed by Nuvation, a Xilinx Alliance Program design services firm, and is being distributed by Xilinx.

The VSA System

Figure 1 shows a block diagram of the VSA system. Spectrum Digital's DM642 EVM showcases TI's DM642 digital media processor (TMS320DM642). On-board components include 32 MB SDRAM, 4 MB Linear Flash, two video decoders, one video encoder, two S-Video/composite video inputs, one S-Video/composite/VGA output, 10/100 Ethernet PHY, mic and headphone jacks, and an off-board connector driven through the DM642's EMIF interface. You can develop DSP firmware on the DM642 EVM with TI's Code Composer Studio and a JTAG emulator.

XEVMDM642 Virtex-4 Daughtercard

Spectrum Digital's XEVM642 is a Virtex-4 SX35-based daughtercard that plugs into the DM642 EVM. In addition to the Virtex-4 device, the XEVM has memory, clocks, a JTAG port, and a Compact Flash card socket. From video algorithm acceleration, data compression filters, and custom logic, the Virtex-4 FPGA is easily programmable with Xilinx System Generator for DSP and ISE™ software.

VHDL and Firmware

The Video VSA includes a set of VHDL modules and the firmware to directly control them (illustrated in Figure 2). These modules include a generic user logic module (the function that fits into the "virtual socket"), a video input module, a test pattern generator, a 2:1 video switch, a video output module, and a host interface module. All of these modules – and the firmware that controls them – are reusable.

The Video VSA modules are connected together in a way that provides a "virtual socket" where the user function can reside. Any appropriate functionality and implementation approach of the user function is possible; the surrounding Video VSA modules are connected together to form the infrastructure that allows you to focus your design effort on the user function.

The demo firmware comprises four main components, shown in Figure 3 as shaded blocks within the overall demo firmware framework.

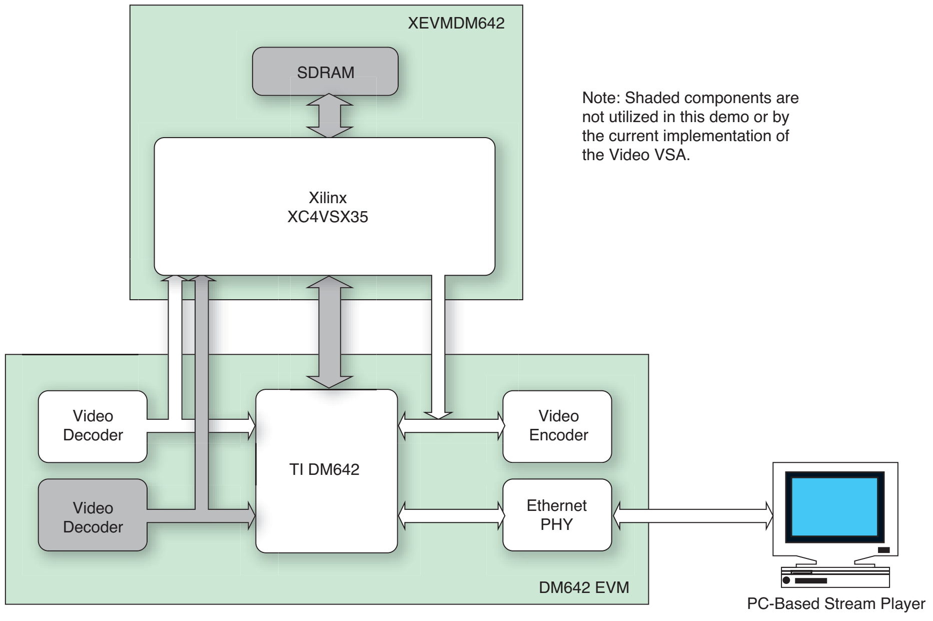


Figure 1 – Block diagram of EVM642 and XEVM system

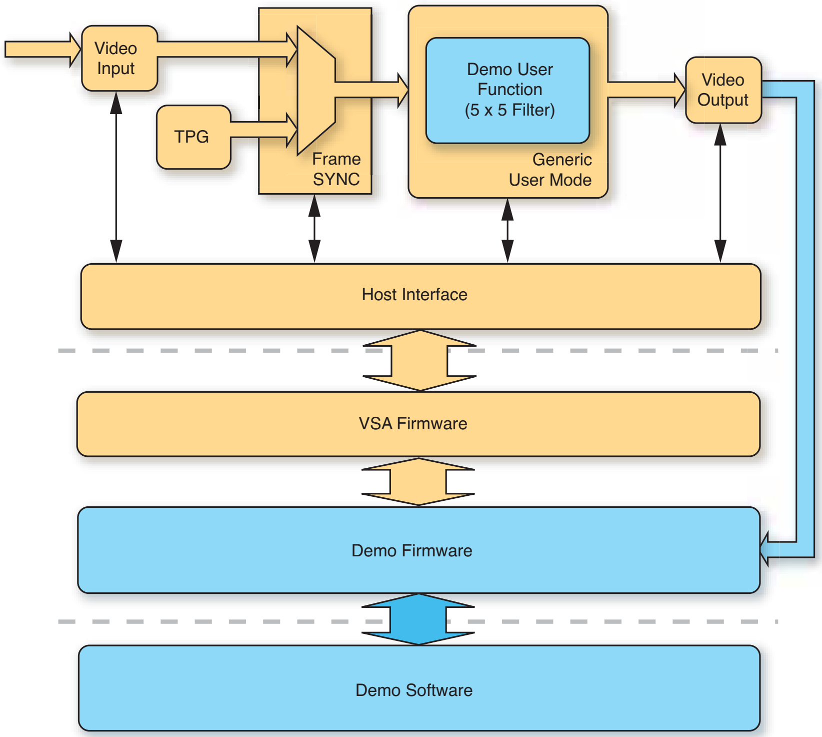


Figure 2 – Video VSA system block diagram

Video Conversion Pipeline

This component comprises two independently running pipelined tasks. The first receives a video stream from the FPGA video output port through a DM642 video input port and converts it to YUV420 format. The second compresses it into an in-memory JPEG image for use by the MJPEG player server and HTTP server components. It passes new frames to the MJPEG player server when they are available.

MJPEG Player Server

This task awaits incoming connections from the PC-based MJPEG player client and streams out newly captured JPEG images while the connection is active. This task receives notification from the video conversion pipeline when a new JPEG image is available.

Demo Control

The demo control component is a task that polls the video's locked status and video standard for changes. This task will re-initialize the video conversion pipeline when video lock has been restored. It is also used to handle 525/625 video standard changes.

During the same polling loop, the demo's processing window position is updated (if movement is enabled). This implements the window's bouncing behavior.

The remainder of the demo control component is a series of demo-specific functions responsible for controlling the following demo settings:

- Processing window position, size, enabled status, and auto move
- Filter kernel (including chroma bypass)
- Video source selection (test pattern or live video)

Web Server

The Web server is configured to use a standard HTTP port and offers the following content/services:

- The current video frame is made available as a JPEG file
- A JavaScript-based player/control console for the demo is available as the default web page on the server; this page also loads two static logo images from the Web server
- Three dynamic CGI scripts process incoming HTTP POST configuration change requests used by both the MJPEG player and the default Web-based player to control the demo
- A dynamic website for running automated tests on the host interface

VSA Demo Application

The system includes a filter application that demonstrates one of many possible functions that you can implement in the VSA. The function is a two-dimensional 5 x 5 FIR filter with a configurable rectangular window that filters video samples within the processing window while passing all other samples unmodified. The position and size of the processing window is implemented to allow uninterrupted video streaming during modification. Figure 4 shows a snapshot of a live streaming video display featuring an edge detect filter kernel.

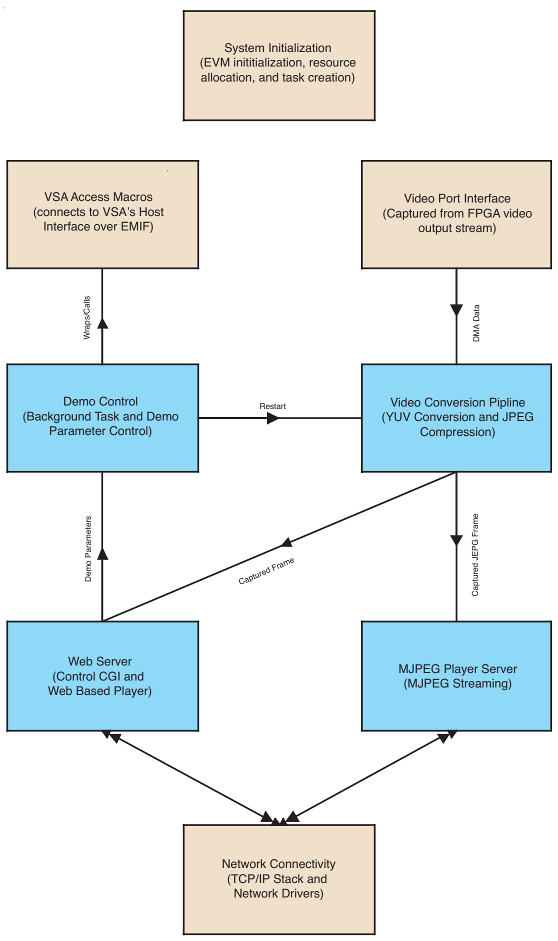

System Initialization (EVM inititialization, resource allocation, and task creation)
VSA Access Macros (connects to VSA's Host Interface over EMIF)
Video Port Interface (Captured from FPGA video output stream)
Wraps/Calls
DMA Data
Demo Control (Background Task and Demo Parameter Control)
Restart
Video Conversion Pipline (YUV Conversion and JPEG Compression)
Demo Parameters
Captured Frame
Captured JEPG Frame
Web Server (Control CGI and Web Based Player)
MJPEG Player Server (MJPEG Streaming)
Network Connectivity (TCP/IP Stack and Network Drivers)

Figure 3 – VSA firmware component overview

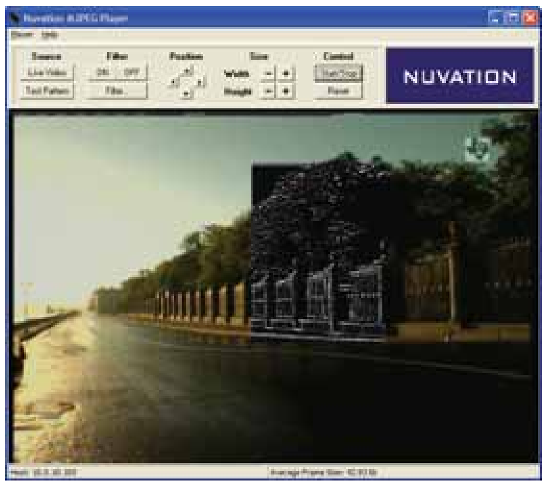


Figure 4 – MJPEG player (showing live video with edge-detect function)

The filter coefficients are represented in 16-bit signed 2's complement fixed-point format, allowing implementations of high-precision video filters with gain. The coefficients are loadable at runtime and are designed to engage without disturbing the video stream. The resulting video is normalized and clamped in accordance with ITU-R BT.656/601 as a post-processing step in the filter.

The filter is fully implemented in a Xilinx System Generator for DSP workflow that operates under The MathWorks's Simulink environment. System Generator for DSP provides abstractions that enable you to develop highly parallel systems in Xilinx FPGAs, providing system modeling and automatic code generation from Simulink and MATLAB, also from The MathWorks.

The purpose of the Video VSA demo is to showcase the process to customize Video VSA modules for your application. A detailed application note is included with the package, along with a demo user guide to facilitate a quick start to your development.

Conclusion

The new Video Virtual Socket Adapter from Xilinx enables rapid algorithm porting and verification for video system development, utilizing Xilinx Virtex-4 SX platform devices and TI DM642 digital media processor DSPs in the System Generator for DSP tool flow. The VSA hardware and associated TI DSP tools are available from Spectrum Digital (*www.spectrumdigital.com*). For all other VSA inquiries, please contact your local Xilinx representative or visit *www.nuvation.com.*

The Role of FPGAs in Digital Radio Subsystems

Digital techniques for reducing analog costs.

by Steve Cooper
CTO
Axis Network Technology
steve.cooper@axisnt.com

A significant goal for mobile wireless infrastructure suppliers is to develop base stations (BTS) light enough to be deployed next to an antenna and reliable enough not to require tower climbs for servicing. These products ultimately will have the lowest cost structure, both in capital expenditure (CAPEX) and operating expenditure (OPEX). CAPEX and OPEX are two of the biggest issues affecting operators, and therefore base station OEMs, today.

Hardware and site preparation are major contributors to CAPEX costs, whereas major OPEX costs are site leasing, backhaul, and electricity. Products such as remote radio heads (RRHs) or compact integrated base stations (CiBTS) will go a long way in improving these contributing factors.

Key to making compact integrated products successful (in terms of size and reliability) is reducing power consumption. The power amplifier in the base station is the component that consumes most power. A number of DSP algorithms are available and under development that work to improve power amplifier efficiency. FPGAs play a significant role in the implementation of these algorithms.

Historically Inefficient

In some UMTS base stations, only a tiny fraction of the total DC power consumed is actually transmitted as useful RF power. Around 50% of the radio frequency (RF) power output from the cabinet is dissipated in the feeder cable running up the tower. The remaining power is dissipated as heat, requiring large heatsinks, air conditioning, and large cabinets.

A base station with the enhancements outlined in this article can benefit from a ten-fold increase in conversion efficiency. This significantly reduces heat dissipation in the system, allowing convection-cooled products to be deployed and enabling a dramatic size reduction. Smaller convection-cooled products can be mounted at the antenna, saving the cost of the feeder cable.

Leasing and installation costs are directly linked to the size, weight, and complexity of a base station. Small, convection-cooled CiBTSs or RRHs provide many more deployment options – and hence a reduction in leasing costs. Naturally, a ten-fold increase in conversion efficiency causes a similarly significant reduction in electricity costs.

For these enhancements to work, it is critical that DSP algorithms maintain excellent signal performance and keep the power consumption of the transistors to a minimum.

Crest Factor Reduction

Telecommunications standards are proliferating (UMTS, HSDPA, HSUPA, WiMAX, DVB-T, DAB, UWB). To maximize spectral efficiency, each of these air interfaces uses complex modulation schemes that have a high peak-to-average power ratio (also known as PAPR, or "crest factor"). Figure 1 shows the different crest factors evident in orthogonal frequency division multiplexing (OFDM) signals. Signals with high crest factors require a large range of dynamic linearity from the amplifier. This means that the power amplifier has to be set to operate well away (backed off) from its most efficient point.

DSP can reduce the peaks of the signal, while some techniques in the baseband processing of the base station can reduce the incidence of the peaks. For example, code selection and tone reservation are two approaches proposed for wideband code division multiple access (WCDMA) and OFDM, respectively. These approaches typically have good performance, although they require intervention in the baseband processing layer before the individual codes or tones are combined into a composite stream.

Fortunately, a number of peak-limiting algorithms can be implemented on the composite I and Q signal. In one approach called peak windowing, the signal is attenuated in the region of each peak. An alternative method is to clip the signal using polar or Cartesian clipping. With Cartesian clipping, the in-phase and quadrature components are clipped independently. With polar clipping, the magnitude of the signal is clipped while preserving the phase. Although either method can be used to limit the crest factor of the signal, polar clipping provides better results in terms of overall signal distortion (lower error vector magnitude [EVM]).

By reducing the crest factor, it is possible to obtain significantly more RF power from the same power transistor. Alternatively, you can use smaller transistors and achieve the same output requirements. The cost of a power transistor is proportional to its peak power handling.

An unclipped UMTS waveform, such as 3GPP-defined Test Model 1, has a 10 dB peak-to-average ratio. Thus, a 20W UMTS base station without a crest factor reduction (CFR) algorithm requires 200W of peak power handling. Using one of the crest factor algorithms discussed here can reduce the peak requirement by half, saving significant cost and power per transmit path.

Moving silicon (and cost) away from the power amplifier and into the FPGA will become more prevalent as technology advances continue to add functionality to FPGAs, and the cost-per-gate continues to track the downward trend of Moore's law.

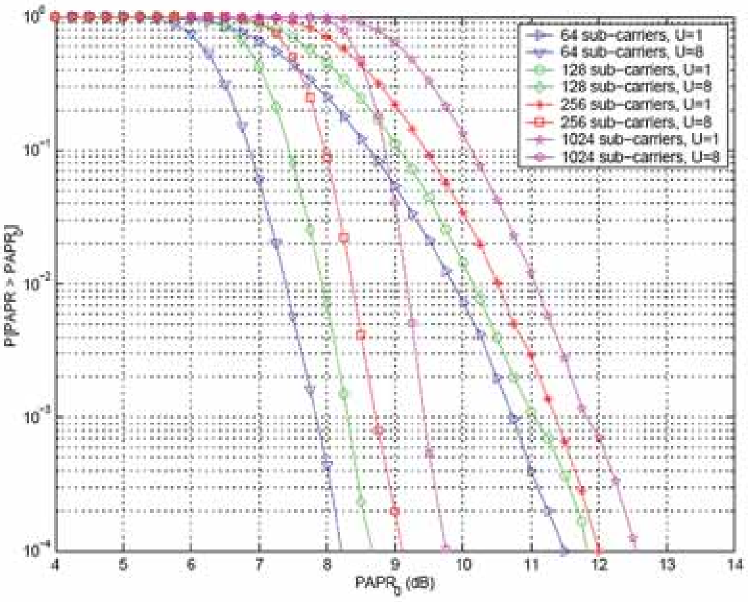


Figure 1 – Cumulative distribution function for OFDM signals (taken from "Peak to Average Power Ratio Reduction of OFDM Symbols" by T. Aaron Gulliver, Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC Canada)

In addition to reducing the total system cost, CFR will also significantly improve the power efficiency of the base station, because not only is the price of the power transistor proportional to its peak power, but so is its power consumption. In today's UMTS base stations, the power transistor is biased to handle its peak power. Therefore the peak power sets the efficiency of the power transistor and the overall system power consumption.

For these reasons, CFR algorithms are becoming commonplace in UMTS systems. It is now possible to implement a

CFR algorithm and obtain a significant clipping reduction by using a Xilinx® Virtex™-4 FPGA.

An unclipped UMTS signal has a cumulative distribution function, as shown in Figure 2. If clipping is turned on, the crest is clipped by 4 dB to 6.5 dB, as shown in Figure 3. Figure 4 is a plot that compares the results of both clipped and unclipped measurements. In the plots, the black trace has a similar spectral emission to the blue trace. However, the black trace is output from the amplifier at 2 dB higher average output power. You can achieve 60% more average power with the same power transistor using clipping. Plus, the increase in power consumption is only marginal, leading to significantly enhanced efficiency numbers.

Achieving the same performance in adjacent channel and spectral emissions – but driving the amplifier harder – has a significant impact on amplifier efficiency. An amplifier operates in its most efficient region when it is most compressed. Results on typical UMTS 20W amplifiers show that driving the amplifier 2 dB higher results in a power consumption increase of only 25%.

Limitations of CFR

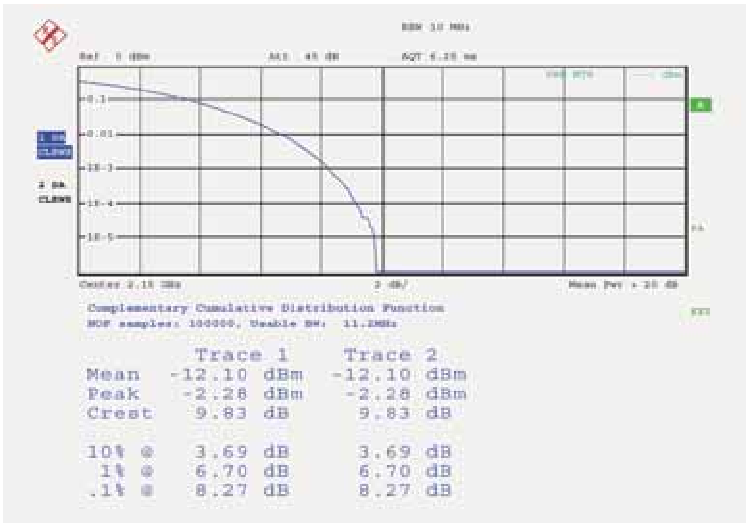
Unfortunately, as discussed, clipping the peaks of a signal degrades its purity and increases the occurrence of bit errors, especially in areas of weak reception. UMTS Release 99 uses the QPSK modulation scheme. This scheme is relatively tolerant of signal impurities; the 3GPP standard for UMTS allows as much as 17.5% EVM degradation. As UMTS networks are typically interference limited, the impact of the increased EVM is of limited importance, as other factors dominate the system bit error rate. Current state-of-the-art UMTS CFR algorithms are demonstrating clipping to a 6 dB peak-to-average ratio while meeting 3GPP Release 99 EVM requirements.

As modulation schemes (shown in Figures 5 and 6, respectively) change from QPSK (UMTS Release 99) to higher level schemes such as 16 and 64 QAM (used by HSDPA and WiMAX), the tolerance of the system to any impurity is reduced. As Figures 5, 6, and 7 show, the relative distance between each point on the constellation diagram is reduced. Impurities in the signal will cause the detection points to merge together, creating bit errors. This error can be seen in the constellation measurements shown in Figures 7 and 8. Currently algorithms are only providing 8 dB of peak-to-average ratio levels while meeting the tight EVM requirements for 64 QAM signals.

Clearly, clipping is of value in those systems that can tolerate higher levels of EVM degradation. But to improve the efficiency of systems using higher level modulation schemes, additional techniques are required.

Digital Pre-Distortion

Another important parameter affecting the power transistor choice is the adjacent channel power ratio (ACPR). The plots shown in Figure 4 were deliberately chosen to show an ampli-

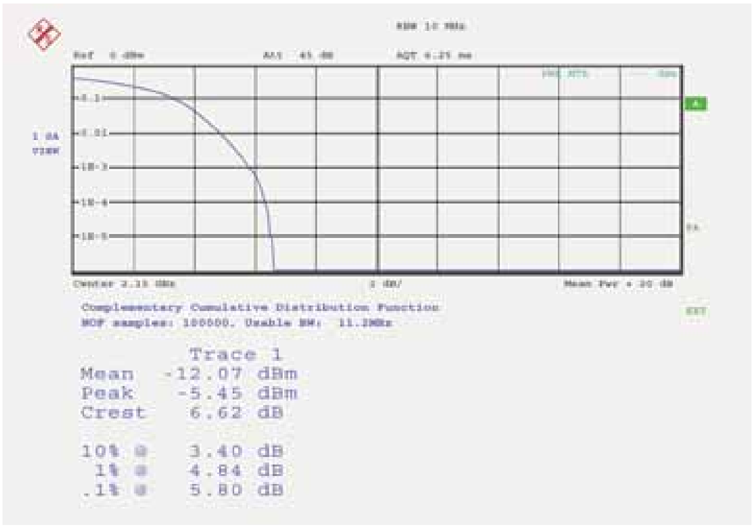


```
Complementary Cumulative Distribution Function
NOF samples: 100000, Usable BW:  11.2MHz

          Trace 1       Trace 2
Mean   -12.10 dBm    -12.10 dBm
Peak    -2.28 dBm     -2.28 dBm
Crest    9.83 dB       9.83 dB

10% @    3.69 dB       3.69 dB
 1% @    6.70 dB       6.70 dB
.1% @    8.27 dB       8.27 dB
```

Figure 2 – Cumulative distribution of unclipped signal



```
Complementary Cumulative Distribution Function
NOF samples: 100000, Usable BW:  11.2MHz

          Trace 1
Mean   -12.07 dBm
Peak    -5.45 dBm
Crest    6.62 dB

10% @    3.40 dB
 1% @    4.84 dB
.1% @    5.80 dB
```

Figure 3 – Clipped cumulative distribution

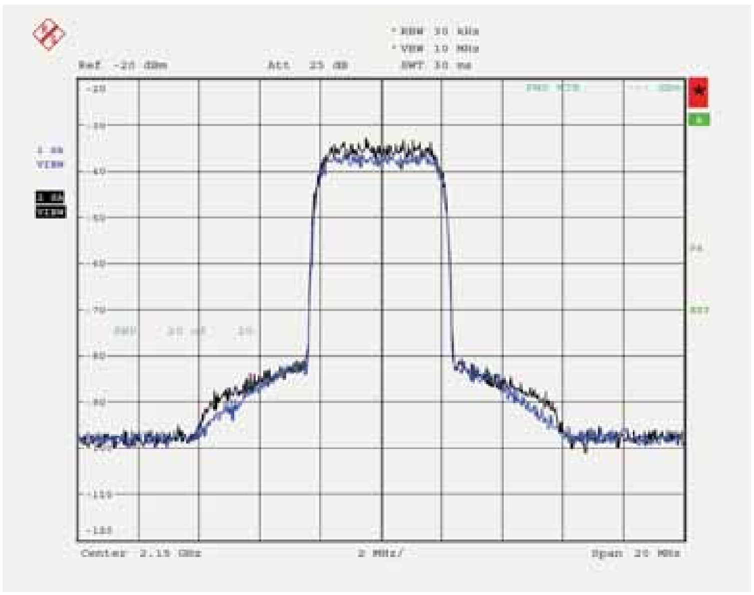


Figure 4 – Plot showing the comparative results of both clipped and unclipped signals

As FPGA performance continues its rapid advancement, the manipulation of the array required for digital pre-distortion can be carried out on the FPGA. This leads to a one-chip solution that can interface to the DAC for the RF transmit path and the ADC for the pre-distortion capture receiver.

fier that was passing the 3GPP adjacent channel and spectral emission requirements without linearization. Linearization allows the operation of the amplifier even further into its highest efficiency area. A number of available techniques will have this effect. These techniques originated in the analog domain with feed-forward and cross-cancellation and have now moved into digital pre-distortion carried out in the I and Q domain.

Pre-distorters have been demonstrated that have almost perfect performance – removing all non-linearities and minimizing the adjacent channel power down to the noise floor of the system. However, the algorithms that achieve this performance are very processor-intensive and typically deployed in very large ASICs. The algorithms in the ASICs must be able to cope with many different amplifiers and topologies to address the broadest market. This in turn makes the ASIC more complex and power-hungry.

Instead of an ASIC, using an FPGA to implement pre-distortion allows you to use the flexibility of the device and implement a specific algorithm tailored to the specific amplifier being pre-distorted.

This is ideal for compact integrated products, as the transceiver, algorithm, and amplifier are permanently integrated together in one field-upgradable unit. It is not necessary for the algorithm to be overly complex; hence it takes less silicon space and can compete in cost with an all-encompassing ASIC.

By working closely with power transistor suppliers, it is possible to develop very code-efficient, custom digital pre-distortion algorithms. During factory testing the best algorithm for each particular amplifier can be chosen and programmed into the FPGA.

Pre-distortion not only improves the spectral emissions of the amplifier but can also have a significantly positive effect on the signal EVM. This fact will likely lead to pre-distortion being widely used within WiMAX systems.

The characteristics of an amplifier change according to the signal passing through them, temperature and frequency effects, and device technology. To produce consistent optimized performance, pre-distortion algorithms require the wideband capture of the amplifier output. This analog signal must be converted into the digital domain using high-performance ADCs. Once in the digital domain, very rapid, real-time manipulation of large mathematical arrays is required. Essentially, the inverse of the non-linear response of the amplifier must be applied to the input signal. This inverse response must closely track the changes within the power transistor. The mathematical processing is often carried out by a dedicated DSP device.

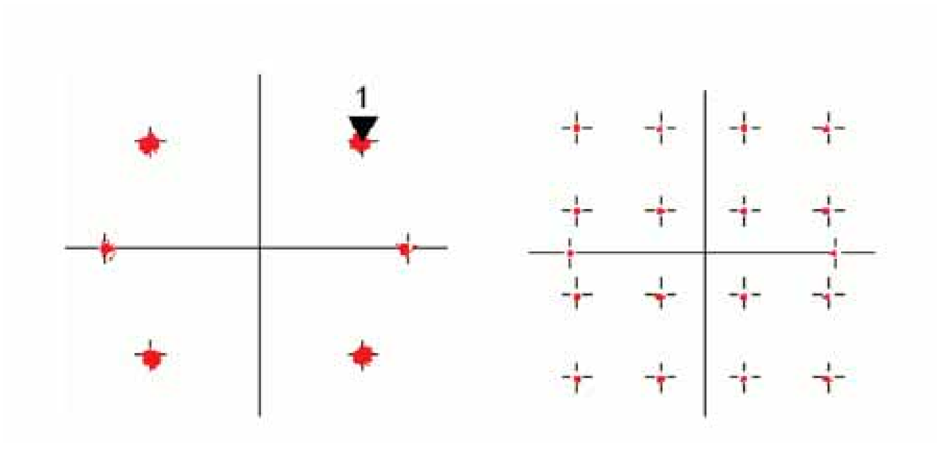


Figure 5 and 6 – Plots showing relative constellations of QPSK (left) and QAM16 (right)

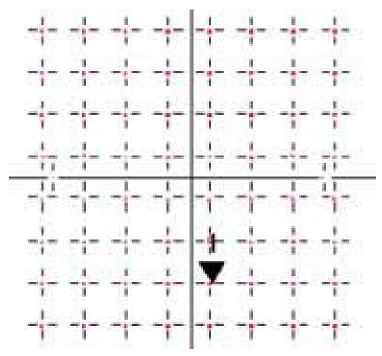


Figure 7 – Plot showing 64 QAM constellations with good EVM

As FPGA performance continues its rapid advancement, the manipulation of the array required for digital pre-distortion can be carried out on the FPGA. This leads to a one-chip solution that can interface to the DAC for the RF transmit path and the ADC for the pre-distortion capture receiver. This same single chip can also carry out the signal processing required for digital upconversion, CFR, and all of the processing and algorithm manipulation for digital pre-distortion.

Combined together, digital pre-distortion and CFR are the current state-of-the-

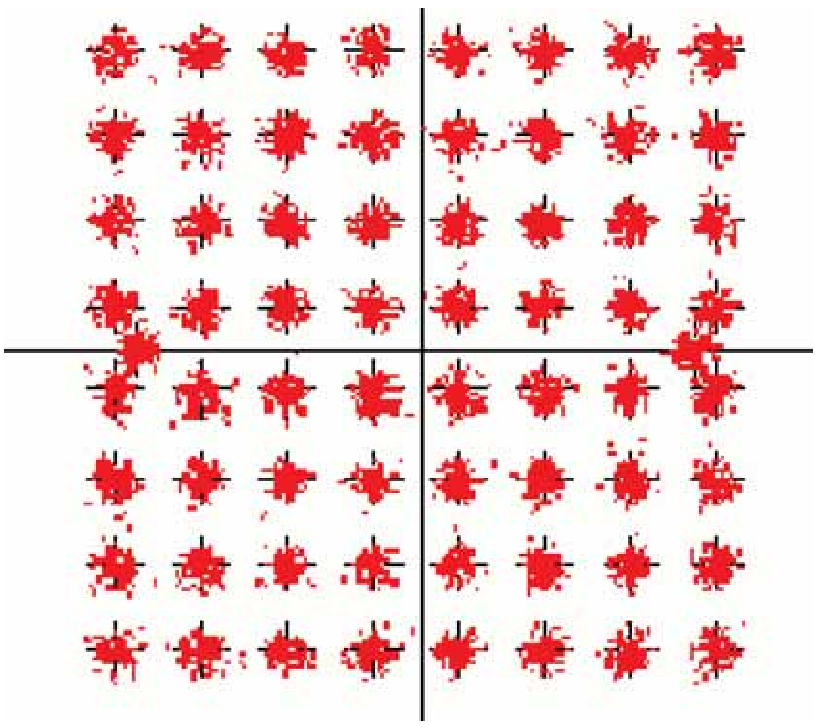


Figure 8 – Plot showing 64 QAM constellations with bad EVM caused by clipping

art techniques for improving efficiency of UMTS systems. The majority of existing systems use ASICs or ASSPs. However, as both amplifier technology and higher level modulation schemes are implemented, the algorithms required will need enhancement and upgrading in the field.

Other Analog Techniques

Of course there are techniques in the analog domain to improve efficiency. A technique known as Doherty has started being deployed in UMTS systems. Doherty uses two output stage transistors biased at different points. One of the transistors is on all the time; the second only turns on as the signal approaches its peak. This reduces current consumption as the transistors are not turned on all the time, and when they are on they are operating in their more efficient regions. It is important to note that Doherty works at its best for signals with 6 dB of peak-to-average. In simple terms, the first amplifier covers the lower 3 dB and the second amplifier covers the upper 3 dB. Efficiencies of 32% have been demonstrated for signals with a 6 dB peak-to-average ratio.

For signals with more than 6dB peak-to-average, the two transistors are not able to operate completely within their most efficient regions. Therefore for signals such as 64 QAM (currently with 8 dB PAR), the improvements obtained with Doherty are not as significant.

Envelope Tracking

One efficiency enhancement technique generating significant interest is envelope tracking, in which the drain voltage of the transistor is varied at the same time as the signal passing through the transistor. When traffic is light, it continues to run efficiently by reducing the current consumption of the transistor. These techniques hold the promise of 35% efficient power amplifiers, even if the PAR is greater than 8 dB.

Supporting envelope tracking in the digital domain and combining it with digital pre-distortion allows for the development of reliable compact integrated products. These two algorithms are similar: they both require very fast tracking loops, processing of the I and Q data stream, and they vary between different amplifier designs.

Envelope tracking requires access to the I and Q data. It is necessary for the I and Q stream to be both delayed and processed so that the power amplifier biasing signal is modulated at exactly the same moment as the composite analog waveform. This requires different digital modules implemented in the FPGA, according to the particular efficiency technique deployed. A diagram showing this technique at the block level is shown in Figure 9.

Conclusion

The Virtex-4 family of FPGAs is ideally suited to implement the existing algorithms required for a digital radio modem. In addition, the use of an FPGA within the digital modem provides the future-proofing necassary to allow software field upgrades for higher efficiency and higher level modulation systems.

For more information about how you can implement these techniques within digital radio systems, please contact the author at *steve.cooper@axisnt.com.*

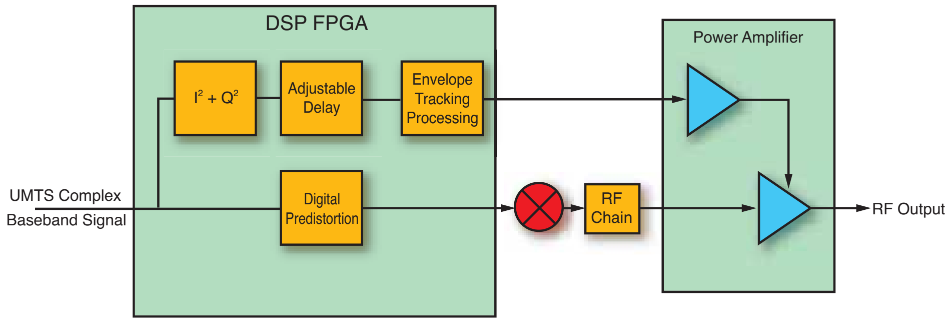
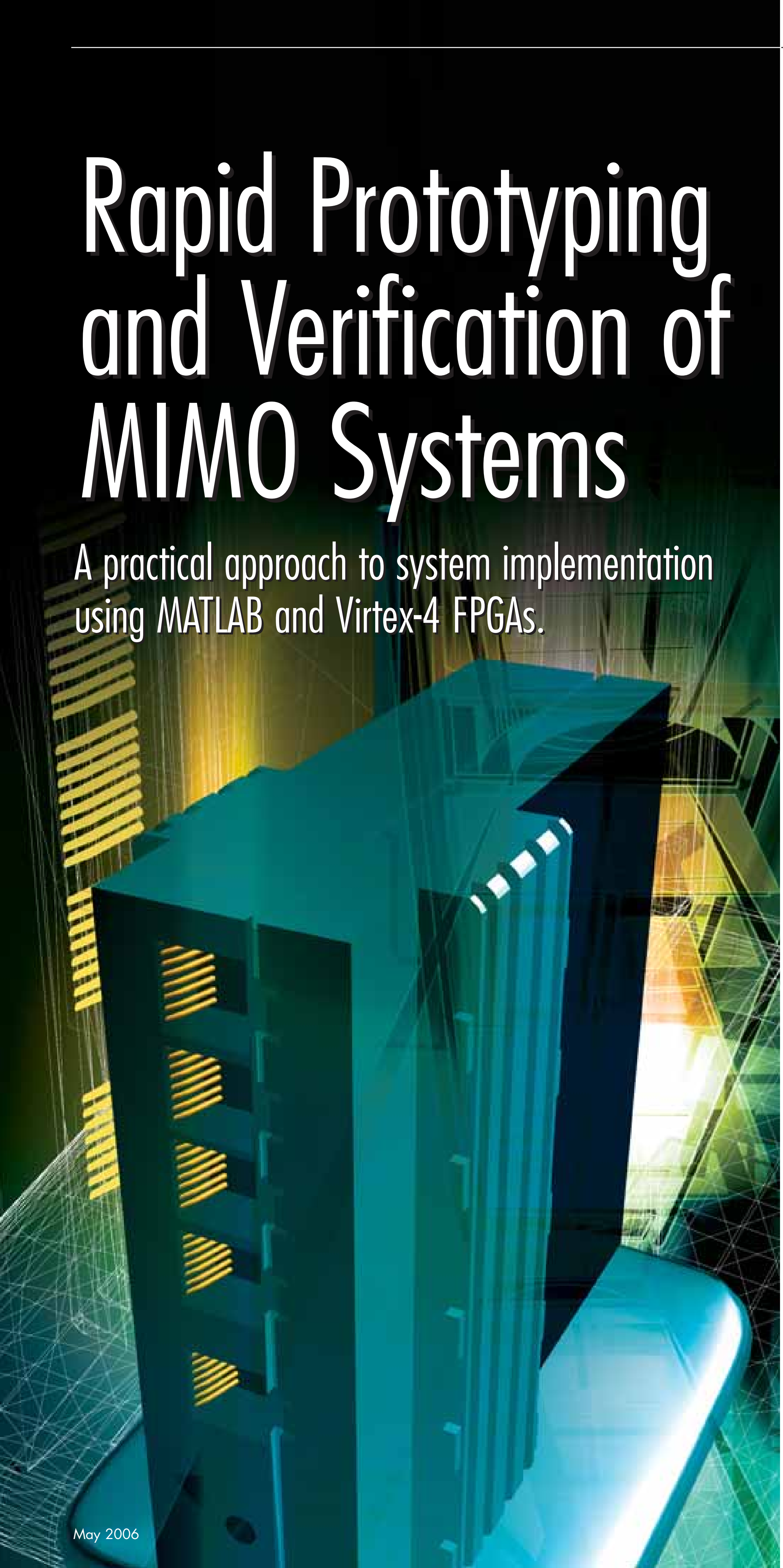


Figure 9 – Block diagram of envelope tracking implementation using FPGAs to control and bias the power amplifier transistor efficiently.

The author would like to thank Rohde and Schwarz for the use of the test equipment (FSQ and SMU) required to carry out the measurements.



Rapid Prototyping and Verification of MIMO Systems

A practical approach to system implementation using MATLAB and Virtex-4 FPGAs.

by Tom Feist
Director, DSP Tools Marketing
Xilinx, Inc.
thomas.feist@xilinx.com

Spatially multiplexed multiple-input multiple-output (MIMO) transmitters and receivers promise significant performance gains for wireless communications systems over their existing single-input single-output (SISO) counterparts. Next-generation wireless standards, such as 802.11n, will support data transmission rates as high as 600 Mbps and wireless local area network transmission rates in excess of 1 GHz.

The design of these systems, however, forces a compromise in cost and power that can have significant consequences for handheld devices running on batteries. The challenge facing design teams is to determine the optimal balance between these design requirements for their particular application.

At the heart of this technology is the concept of multipath, which refers to the reflection of radio frequency (RF) signals in a physical environment. Whereas multipath degrades the performance of existing 802.11 devices, spatially multiplexed orthogonal frequency division multiplexing (OFDM) MIMO – a key element of the 802.11n standard – takes advantage of these reflections to "tune" transmissions, minimize errors, and improve overall performance. But at these bandwidths, scattering, diffraction, and absorption by objects in the transmission path are an important consideration. Designing a MIMO system requires that these effects are profiled as accurately as possible in the form of a channel model.

There are three primary sources of channel models: software-based mathematical models, often available from the standards committees; hardware-based MIMO channel emulators, either designed in-house or provided by companies such as Azimuth; and, best of all, the real-world environment that the MIMO system is intended to operate. Verifying a MIMO system in the real world requires the ability to rapidly prototype the transmitter and receiver on a MIMO-oriented FPGA hardware platform, such as the VHS-ADC-V4 card from Lyrtech.

The MIMO Performance Advantage

The benefit of spatially multiplexed MIMO technology is the ability to increase transmission speed with the number of antennas. The data rate of a today's existing SISO systems is determined by the formula:

R = Es * Bw

where R is the data rate (bits/second), Es is the spectral efficiency (bits/second/Hertz), and Bw is the communications bandwidth (Hz). For instance, for the 802.11a standard the peak data rate is determined by the formula:

Bw = 20 MHz

Es = 2.7 bps/Hz

R = 54 Mbps

An additional variable "Ns" is introduced into this equation when using MIMO, which is the number of independent data streams that are transmitted simultaneously in the same bandwidth but in different spatial paths. The spectral efficiency is now measured as the transmission per stream Ess, and the data rate of the MIMO system becomes:

R = Ess * Bw * Ns

Let's compare the previous 802.11a example with what is obtainable with the current 802.11n proposal, operating at a 20 MHz bandwidth and using four antennas:

Bw = 20 MHz

Ess = 3.6 bps/Hz

Ns = 4

R = 288 Mbps

The use of MIMO technology has delivered a 5.3x data rate improvement for the 802.11n proposed standard.

MIMO System Hardware Complexity

The performance gains of a spatially multiplexed MIMO system come at the expense of hardware complexity. A transmit/receive system that uses multiple antennas not only transmits data between the corresponding antennas but also between adjacent antennas. As you can see in Figure 1, data is received in the form of a "MIMO channel matrix."

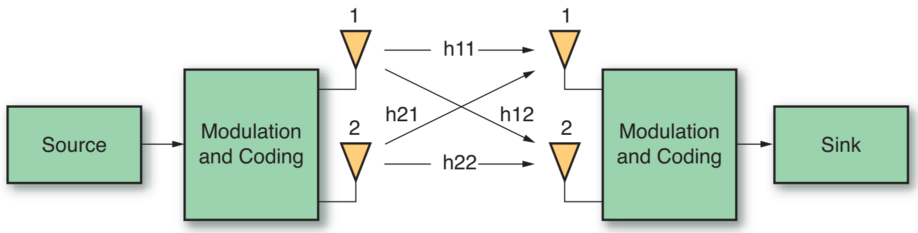


Figure 1 – MIMO channel

Linear algebra techniques such as singular value decomposition (SVD) or matrix inversion are required to decouple the channel matrix in the spatial domain and recover the transmitted data. Backwards compatibility requirements to the 802.11g standard limit the number of antennas for the 802.11n standard to either two or four, which subsequently limits the channel matrix size to either a 2 x 2 or 4 x 4.

Developing a MIMO system prototype in hardware that performs at the actual system data rates requires the use of an FPGA-based hardware platform. The Xilinx® Virtex™-4 family of FPGAs provides far greater performance than a DSP processor for this class of applications by providing as many as 512 hardware multipliers capable of parallel operation. In designing this prototyping system, however, you are faced with two considerable challenges: the first is to design something as complex as an SVD or matrix inverse in hardware and the second is tuning the implementation for optimal performance.

Implementing Matrix Operations on FPGAs

The specific SVD or matrix inversion algorithm selected for implementation will be a tradeoff between numerical stability and hardware efficiency. You will need to develop a high-level MATLAB model to determine the most efficient algorithm for a particular application. In the case of the SVD, this may involve choices between adaptive estimation techniques, vector rotations, or other simplifications that result from channel matrices with special properties such as symmetry.

```
function [v, w] = givens_rotation(x, y)
r_sqr = x(1)*x(1) + y(1)*y(1);
r_inv = 1/sqrt(r_sqr);
sin_phi = y(1)*r_inv;
cos_phi = x(1)*r_inv;
vt = x*cos_phi + y*sin_phi;
wt = y*cos_phi – x*sin_phi;
if (x(1) == 0) & (y(1) == 0)
  v = x;
  w = y;
else
  w = wt;
  v = vt;
end
```

Figure 2 – Givens rotation algorithm

Once an algorithm has been finalized, you will need to tune the hardware performance to overall system requirements. Maximizing the performance of a MIMO system in hardware will require that partial parallelism of the multiplication operations be implemented in key areas of the design that will have the greatest impact on overall performance. The Givens rotation algorithm shown in Figure 2 provides a nice example of the performance gains possible through parallel multiplication operations. Givens rota-

Development and verification cycle times are reduced by using the MATLAB algorithm as the golden source for FPGA development and eliminating re-writes into other languages or design environments.

tions are commonly used to solve the symmetric eigenvalue problem and are a key building block of the QRD matrix inverse.

You can implement this algorithm using either multipliers or a CORDIC approximation method. The Xilinx AccelDSP™ Synthesis tool's design exploration features were used to increase performance by inserting parallelism into the architecture without code rewrites. As shown in Table 1, this allowed performance gains as much as 10x over the parallel CORDIC implementation. Algorithms based on Givens rotations have received greater attention recently because they lend themselves nicely to a parallel implementation.

Architecture	DSP48s	Slices	Throughput
Resource Shared Multiplier	26	943	2.8 MSPS
Parallel Multipliers	46	1774	54 MSPS
Resource Shared CORDIC	0	870	1.3 MSPS
Parallel CORDIC	0	2237	5.7 MSPS

Table 1 – The range of results obtained by synthesizing a 4 x 4 matrix using the AccelDSP Synthesis tool and targeting a Virtex-4 device.

For large systems, the added hardware that results from increased parallelism must not exceed the resources of the target FPGA. The number of architectural possibilities you must evaluate can be considerable. The process of determining optimal hardware architecture is well suited for a high-level algorithmic synthesis tool such as AccelDSP.

A MATLAB-Based FPGA Design Flow

MATLAB from The MathWorks provides a truly unique environment for the design and implementation of spatially multiplexed MIMO systems. The inherent language support for loops, complex numbers, vector and matrix operations, and mathematical functions provides a highly efficient modeling environment for the linear algebra algorithms required for MIMO.

Figure 3 illustrates the benefits of the AccelDSP Synthesis tool, including the flexibility to define and implement custom architectures for spatially multiplexed MIMO systems on FPGAs using floating-point MATLAB. Automated floating- to fixed-point conversion is provided to assist in solving the complex quantization issues resulting from the iterative nature of linear algebra functions such as an SVD. Once you have determined an acceptable fixed-point model, you can rapidly explore performance-versus-hardware tradeoffs using algorithmic synthesis, quickly increasing the number of dedicated hardware multipliers to improve performance and take full advantage of the flexibility of the Virtex-4 architecture. The generated RTL from AccelDSP Synthesis is automatically verified against the golden-source MATLAB to ensure bit-true functional correctness.

The AccelDSP Synthesis tool and Lyrtech prototyping environment both have interfaces to the Xilinx System Generator for DSP design environment to provide an automated MATLAB to prototyping design flow.

Conclusion

Prototyping a spatially multiplexed MIMO system for use in real-world verification is dramatically simplified through the adoption of a MATLAB-based design flow for the channel-matrix DSP hardware development. Development and verification cycle times are reduced by using the MATLAB algorithm as the golden source for FPGA development and eliminating re-writes into other languages or design environments. Additionally, the high-level nature of MATLAB allows the AccelDSP Synthesis tool to quickly explore hardware alternatives for an algorithm, including the use of DSP blocks, RAMs, and pipelining.

For more information about the AccelChip solution, visit *www.accelchip.com.*

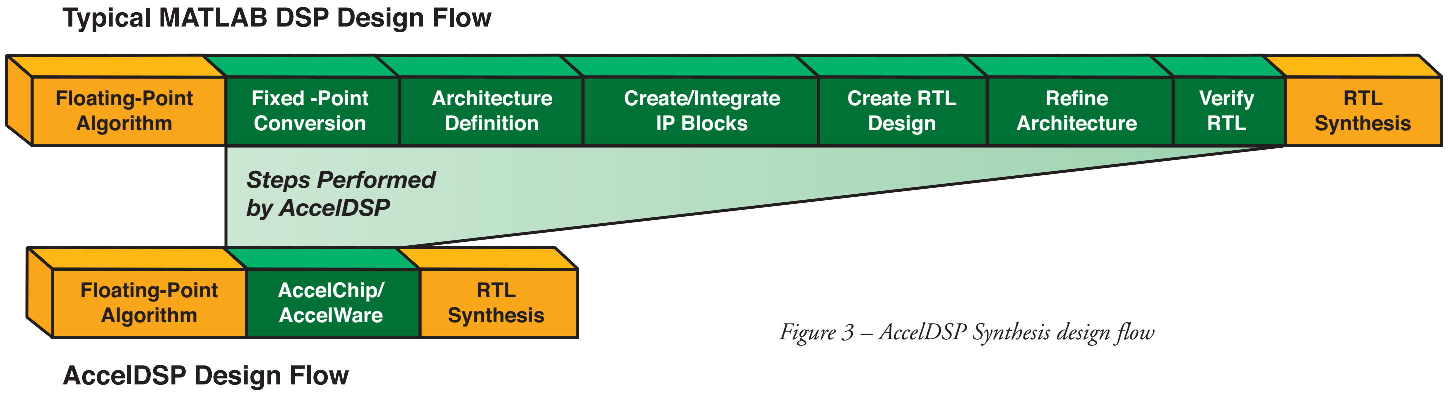


Figure 3 – AccelDSP Synthesis design flow

High VELOCITY LEARNING

XpressTrack

Nu Horizons Electronics Corp. is proud to present our newest education and training program - **XpressTrack** - which offers engineers the opportunity to participate in technical seminars conducted around the country by experts focused on the latest technologies from Xilinx. This program provides higher velocity learning to help minimize start-up time to quickly begin your design process utilizing the latest development tools, software and products from both Nu Horizons and Xilinx.

For a complete list of course offerings, or to register for a seminar near you, please visit:

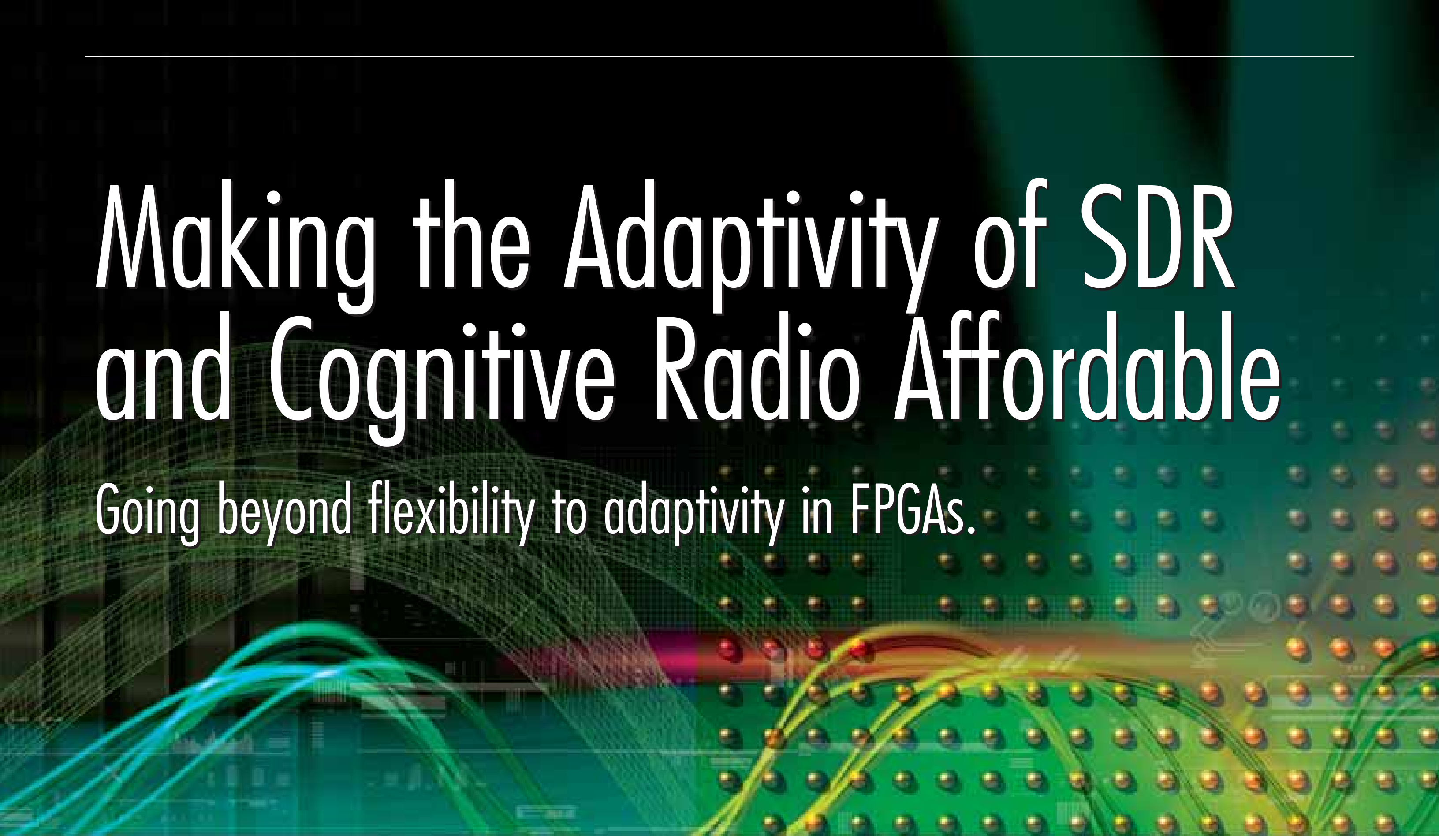
www.nuhorizons.com/xpresstrack

Courses Available

- **Optimizing MicroBlaze Soft Processor Systems**
  - 3 hour class
  - Covers building a complete customized MicroBlaze soft processor system
- **Video/Imaging Algorithms in FPGAs**
  - 3 hour class
  - Verify designs onto actual hardware using Ethernet-based hardware-in-the-loop co-simulation.
- **Introduction to Embedded PowerPC and Co-Processor Code Accelerators**
  - 3 hour class
  - Covers how to build a high performance embedded PowerPC system
- **Introduction to Programming FPGAs with ANSI C**
  - 4 hour class
  - Covers an introduction to C for FPGAs
- **Fundamentals of FPGA**
  - 1 day class
  - Covers ISE 8.1 features
- **ISE Design Entry**
  - 1 day class
  - Covers XST, ECS, StateCAD and ISE simulator
- **Fundamentals of CPLD Design**
  - 1 day class
  - Covers CPLD basics and interpreting reports for optimum performance
- **Design Techniques for Low Cost**
  - 1 day class
  - Covers developing low cost products particularly in high volume markets
- **VHDL for Design Engineers**
  - 1 day class
  - CoversVHDL language and implementation for FPGAs & CPLDs

XILINX®

NU HORIZONS ELECTRONICS CORP.



Making the Adaptivity of SDR and Cognitive Radio Affordable

Going beyond flexibility to adaptivity in FPGAs.

by Manuel Uhm
Senior Marketing Manager, DSP Division
Xilinx, Inc.
manuel.uhm@xilinx.com

The flexibility of software-defined radios (SDRs) and cognitive radios (CRs) provides great value relative to interoperability, upgradability, and future-proofing. This flexibility also enables a highly desired attribute in SDRs and CRs: "adaptivity." Adaptivity can range from a cognitive radio's ability to adapt to its spectral environment to a software-defined radio's ability to adapt a waveform to compensate for channel fading. Like flexibility, adaptivity is enabled by the reprogrammable and reconfigurable processors used in SDRs/CRs, FPGAs, DSPs, and general-purpose processors (GPPs).

Unfortunately, this adaptivity typically comes at a price – both in terms of power and system cost. Recent technological advances, however, are making adaptivity more affordable. For example, partially reconfigurable (PR) FPGAs that embed GPP processors and DSP engines on platform FPGAs can provide adaptivity to a wide range of SDR and CR applications, while lowering the power and cost penalties.

Adaptivity in SDRs and CRs

As a key capability of SDRs/CRs (particularly for military or homeland security purposes), adaptivity can take many forms on the battlefield. With it, you can:

- Change waveforms to interoperate with other friendly communication devices
- Choose the most appropriate communications channel or network for transmission
- Create a mesh network through ad-hoc networking
- Adapt to the radio frequency (RF) environment by using spectral awareness to transmit in an open area of spectrum
- Adapt the waveform to compensate for channel fading
- Collaborate with multiple radios to receive a weak signal that could not otherwise be detected by individual radios
- Jam or null an interfering signal
- Accommodate damage to some of a radio's processing resources by reconfiguring the remaining resources to support the most critical services

For the purposes of an SDR/CR, adaptivity falls into four broad classifications, as illustrated in Figure 1. The lowest functional levels include filters or transforms, such as Kahlman filters, finite impulse response (FIRs), and fast Fourier transforms (FFTs). These low-level functions are basic building blocks for most SDRs/CRs. Thus, you would probably have to adapt the parameters of a function such as an FIR to support a waveform with changing bit rates.

At the component level, adaptivity in an SDR/CR is useful in digital down converters (DDCs) and digital up converters (DUCs). These components must often adapt to waveforms that support different bit rates or sampling rates.

In an SDR/CR, adaptivity at the function or component level is "under the hood," insofar as it is transparent to the end user. At those levels, it does not matter what modifications are necessary to support the service required. On the other hand, the next two levels – the application and services levels – are visible and, as such, you may desire some form of control over adaptivity.

Adaptivity at the application level supports modifications that occur within a

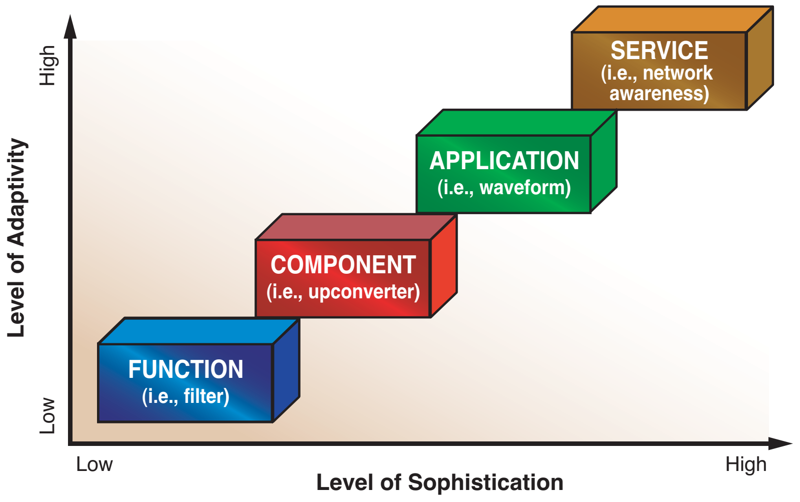


Figure 1 – Levels of adaptivity in an SDR/CR

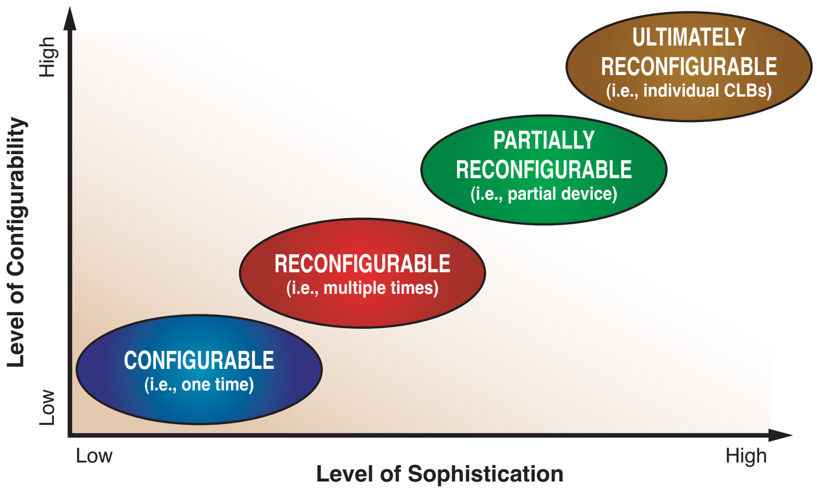


Figure 2 – Levels of configurability in FPGAs

given application. The most common applications in an SDR/CR are waveforms such as the Wideband Networking Waveform (WNW). These waveforms comprise various waveform components, and depending on the mission profile, you may require access to different waveforms on an as-needed basis.

The highest functional level of adaptivity is at the services level. Services like radio services, network awareness services, ad-hoc networking, and even anti-jam services must be able to adapt to changing conditions by calling on available applications as needed.

These levels of adaptivity are in many ways interdependent because adaptivity at each level depends on the previous levels for implementation. For example, you might call on the radio service to transmit data. The service would include adapting to available spectrum by scanning RF, and then selecting the best waveform for sending the data. If the waveform is an adaptive waveform, then certain channel characteristics (such as channel fading) might require modification of waveform components and functions to compensate.

Supporting Adaptivity in FPGAs

FPGAs provide great value as processing platforms in today's SDRs/CRs because of their processing throughput. They also offer a variety of adaptivity levels to this environment, as depicted in Figure 2.

At the lowest level, FPGAs can be one-time configurable (hence not reconfigurable). Obviously, such devices are not ideal for SDRs/CRs, as they cannot adapt to support new functionality after the first time they are programmed.

The FPGA's inherent re-configurability, and more specifically, the capability some FPGAs have to be dynamically reconfigured on the fly, makes them ideally suited for SDRs/CRs. However, in many cases these FPGAs have to be completely reconfigured, which limits the capability of the FPGA to support adaptivity at the component or function level because these levels involve more granularity. As a result, support is generally limited to adaptivity at the application level. Most reconfigurable FPGAs today, including the Xilinx® Spartan™ family of FPGAs, fit into this category.

PR FPGAs allow for the next level of adaptivity. Like reconfigurable FPGAs, you can dynamically reconfigure PR FPGAs multiple times. However, only a portion of the device can be configured at any given time. This provides a level of granularity suitable for adaptivity at the component and even the function level. The Xilinx Virtex™-II and Virtex-4 device families are examples of PR FPGAs.

Finally, the "ultimate" level of reconfigurability is the ability to reconfigure on an individual basis the smallest atomic programmable unit of an FPGA, the configurable logic block (CLB). This allows for even finer levels of granularity for adaptivity at the function level. However, it is not clear that the benefit of such a fine-grained approach outweighs the cost of implementation associated with such a high level of sophistication.

Use Cases

SDRs/CRs can benefit from the adaptivity of a PR FPGA in many ways, from the function level to the service level. Two such cases are offered here. The first is an SDR supporting an adaptive waveform, which demonstrates adaptivity at the application level. The second is a multi-INT platform providing multiple intelligence-related applications, demonstrating adaptivity at the service level.

In the first use case, you would transmit voice or data on an adaptive waveform from your SDR to another radio. At some point, perhaps because of environmental conditions, the channel starts to fade. To the SDR, this is characterized by an increased

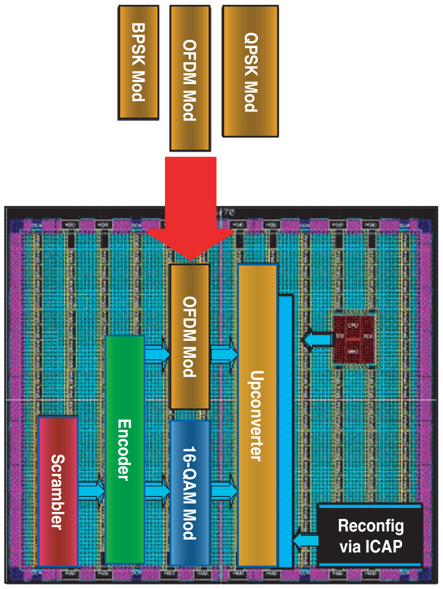


Figure 3 – Adaptivity at the application level in a PR FPGA

bit error rate beyond a certain threshold. To maintain the channel, the radio determines that the waveform must be adapted to the new environmental conditions.

Adaptivity in this case could take many potential forms, including changing the modulation technique, changing the method of forward error correction, or changing the bit rate. For the example illustrated in Figure 3, let's assume that the radio has determined that a change in modulation technique is optimal. The modulator component is represented in Figure 3 as a 16-QAM modulator. Hence, this needs to be swapped out with another available modulator component, in this case a BPSK, QPSK, or OFDM modulator. The OFDM modulator is chosen for its resistance to multipath.

To support this type of component level adaptivity, a regular reconfigurable FPGA would have to be sufficiently large to load all possible components, even if many of them are not being used at any single point in time. Moreover, reconfiguring the entire device would result in losing the communications channel – an unacceptable outcome.

By contrast, a PR FPGA would only have to load the OFDM modulator component in an available portion of the device and then make the switch from the 16-QAM modulator to the OFDM modulator. The 16-QAM modulator could then be unloaded to free up resources for another application or component. The outcome is the same but the PR FPGA can be much smaller – resulting in significant power and cost savings.

The second use case, as illustrated in Figure 4, involves a multi-INT platform that is capable of using services to invoke many possible applications, including radio, spectral analysis, surveillance, jamming, and anti-jamming. Although multi-INT platforms are not commonly used today, the advent of CR will bring about the next revolution in communications.

In this scenario, you may be receiving data. The radio service is being utilized to call on two applications – the spectral analysis application to characterize the spectrum and identify potential threats or signals of interest, and the radio application to receive the data. At some point, an interfering signal may be attempting to jam the receiver, severely impairing your ability to receive crucial intelligence.

In such a case, you may call on the anti-jamming service to null the interfering signal. This service would characterize the interfering signal using the spectral analysis application, and would then load the anti-jammer application to null the interferer. Once the jamming signal goes away, the anti-jammer application can be unloaded by the anti-jamming service. Other services could then load available applications on an as-needed basis, such as an ad-hoc networking application to create a mesh network.

Conclusion

It is clear that adaptivity at all levels, from functions to services, is a highly desired attribute in SDRs/CRs. Although you will primarily be exposed to adaptivity at the service and application level, adaptivity at the component and function level is necessary for implementation.

As CRs and multi-INT platforms become more prevalent, the need to dynamically adapt to changing conditions will increase and will ultimately become a competitive advantage for those vendors who are better able to accommodate different levels of adaptivity. PR FPGAs are ideally suited for driving adaptivity at all levels. PR FPGAs have sufficient granularity to allow you to reconfigure portions of the device down to the size of typical functions in an SDR/CR. They are also able to support whole applications.

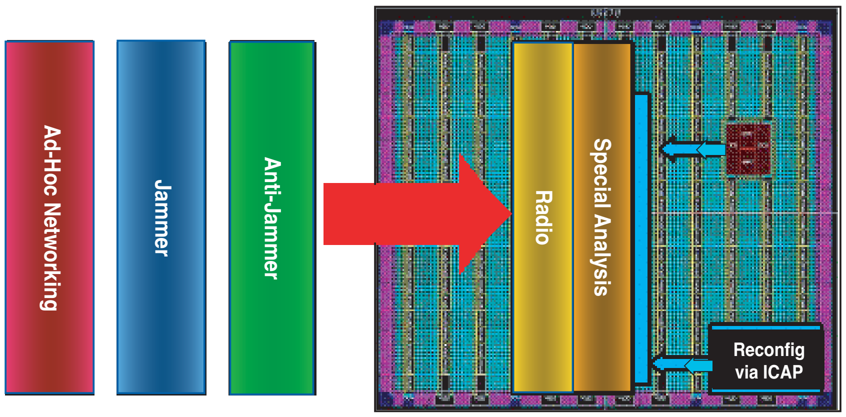
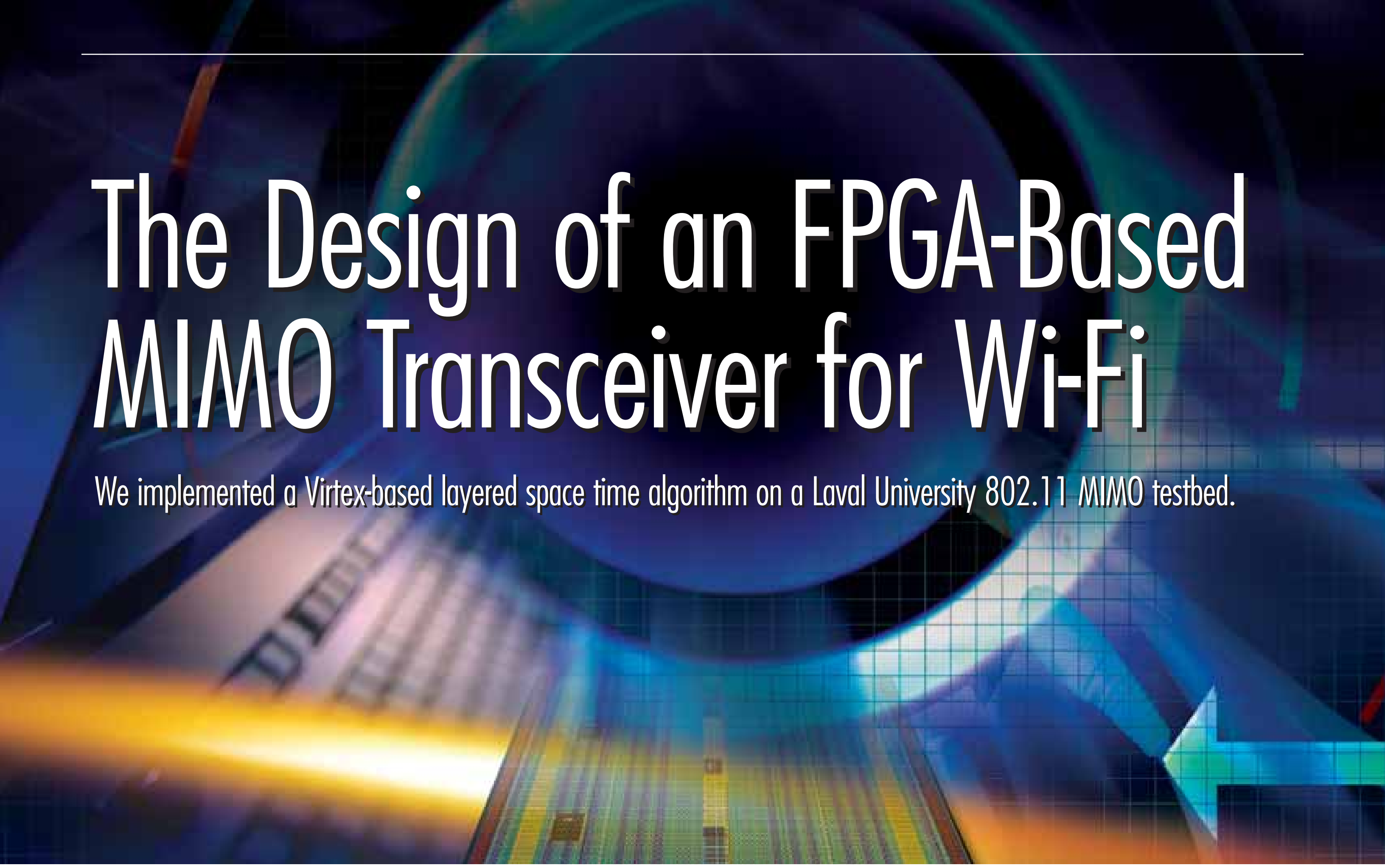


Figure 4 – Adaptivity at the service level in a multi-INT platform



The Design of an FPGA-Based MIMO Transceiver for Wi-Fi

We implemented a Virtex-based layered space time algorithm on a Laval University 802.11 MIMO testbed.

by Sébastien Roy
Professor, Department of Electrical and Computer Engineering
Université Laval
sebastien.roy@gel.ulaval.ca

Louis Bélanger
Executive Vice President, Product Management and Co-Founder
Lyrtech
louis.belanger@lyrtech.com

Wireless communications, through cellular telephony, came of age in the 1980s and 1990s, achieving as an industry a growth rate beyond microcomputers. It seems clear that cellular is on the verge of a major evolutionary leap, as the advent of 4G will spell the end of the connection-oriented voice-centric paradigm in favor of a network-oriented (read packet) resource-sharing multiservice framework. This evolution also implies the definitive advent of broadband wireless, an elusive proposition so far in the cellular world.

The Fast-Moving Wi-Fi World

Upstart wireless LAN (WLAN) technologies under the 802.11 (Wi-Fi) umbrella have leapfrogged cellular and other efforts edging towards broadband wireless (such as 802.16/WiMAX) and have led to the first widespread, commercially successful broadband wireless access technology. In fact, Wi-Fi is a runaway success around the globe.

However, all is not perfect in the WLAN world. Offering nominal bit rates of 11 Mbps (802.11b) and 54 Mbps (802.11a and 802.11g), the effective throughputs are actually much lower – owing to packet collisions, protocol overhead, and interference in the increasingly congested unlicensed bands at 2.4 GHz and 5 GHz. Furthermore, operation in these bands entails a strict regulatory transmit power constraint, thus limiting range and even bit rates beyond a certain distance. Compare this with Gigabit Ethernet, and you will see that a huge rate gap exists between the wired and wireless portions of the network.

MIMO Techniques and 802.11n

Enter MIMO (multiple input multiple output), a wireless technology allowing huge increase in bit rates without consuming additional bandwidth. It is currently a very hot trend in the wireless industry, and with good reason. It basically works by having multiple antennas at both transmitter and receiver and performing appropriate signal processing at both ends. This can be used to effectively create a plurality of channels in space sharing the same bandwidth, a feat referred to as spatial multiplexing. MIMO happens to be the means through which the 802.11n workgroup plans to boost Wi-Fi nominal bit rates to hundreds of megabits per second, perhaps as much as 600 Mbps over a maximum bandwidth of 40 MHz.

Compliant terminals and access points would be equipped with anywhere from one to four antennas. In principle, having four antennas at both ends enables a fourfold rate increase within a given bandwidth. Wi-Fi channels have a nominal bandwidth of 20 MHz. 802.11n proposals advocate a combination of spatial multiplexing, advanced modulation, beamforming, and space-time coding, as well as channel bonding (merging two adjacent channels to form a 40 MHz aggregate channel) to achieve the highest bit rates.

One recent specification proposal by the Enhanced Wireless Consortium details the

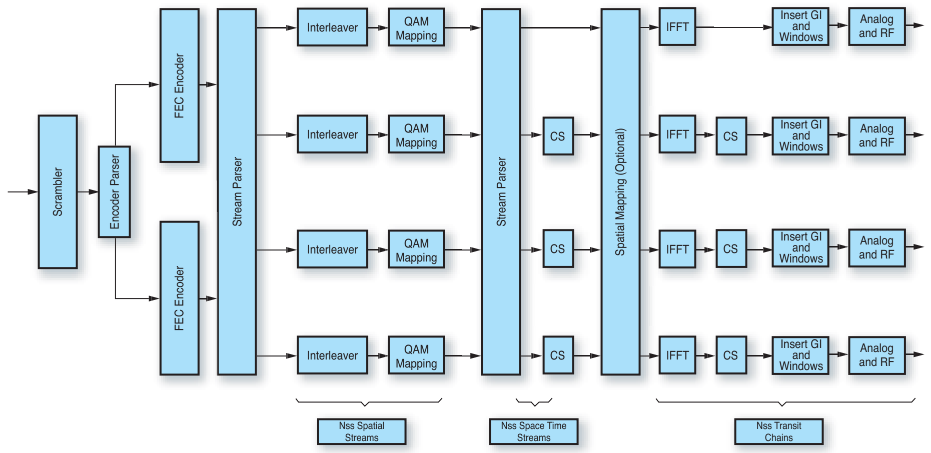


Figure 1 – Transmitter block diagram

802.11n transmitter structure shown in Figure 1. The data from the MAC layer is first scrambled and then demultiplexed into a number N_{ES} of streams, which are routed to N_{ES} forward-error correction (FEC) encoders. These encoders can be either binary convolutional coders with puncturing or low-density parity check (LDPC) coders. The encoder outputs are then divided into a number N_{SS} of spatial streams, each being interleaved and mapped onto a QAM constellation. Collectively, the spatial streams are then optionally subjected to space-time block coding (STBC), a form of two-dimensional linear coding generally intended to improve reliability and not necessarily bit rate (see sidebar, "Space-Time Techniques").

Another optional feature is spatial mapping (beamforming), which relies on foreknowledge of the MIMO channel obtained through a priori sounding. This mechanism aims to maximize energy delivery to the receiving array. In general, the SNR at the receiver is influenced by spatial mapping and the STBC, while the bit rate is a function of the number of bitstreams and the size of the QAM constellation.

Whether you use the optional features or not, the receiver must somehow separate the superimposed spatial streams. This can be accomplished through spatial filtering, provided there are at least as many antennas as there are streams. One effective receiver architecture exploits both spatial filtering and successive interference cancellation (SIC) in a layered structure. Such a layered space-time (LST) receiver was recently the object of an FPGA implementation at Laval University, in collaboration with Lyrtech as part of an 802.11n-oriented research testbed under construction.

Figure 2 – Laval University MIMO testbed using Lyrtech VHS hardware

Part of this testbed is shown in Figure 2. It comprises a custom-built RF front-end supporting as many as four antennas at 2.4 GHz and 5 GHz, Skycross printed antennas, and Lyrtech SignalMaster Quad platforms with integrated multichannel ADC and DAC modules.

The structure of the LST receiver is shown in Figure 3. Our implementation constructs the covariance matrix from vector channel estimates according to

$$\mathbf{R}_{xx} = \sum_{n=1}^{N} \hat{\mathbf{c}}_n \hat{\mathbf{c}}_n^H + \mathbf{I}\sigma_n^2$$

were $\hat{\mathrm{c}}_n$ is the vector channel estimate associated with the nth spatial stream, $<\cdot>^H$ is the conjugate transpose (Hermitian) operator, I is the identity matrix, and σ_n^2 is the thermal noise power on each antenna.

Co-author Sébastien Roy and colleague I. Laroche developed a novel matrix inversion circuit specifically for this application. The covariance matrix structure is exploited to jointly perform the inversions at all layers simultaneously, a complexity/speed gain of order N_{SS}.

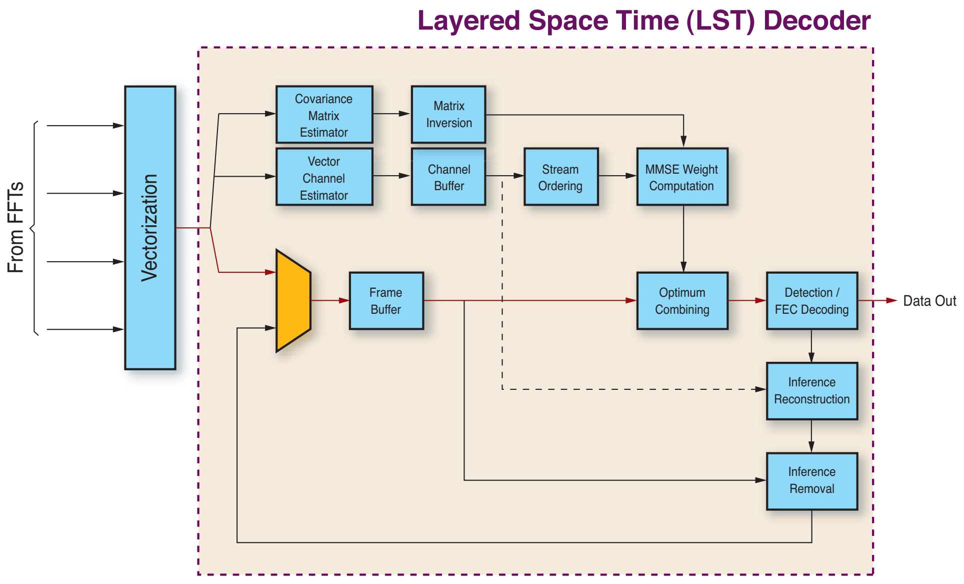


Figure 3 – LST decoder

Implementation Results

The entire receiver was implemented in VHDL on a Xilinx® Virtex™-II XC2V8000 FPGA. It achieves a top frequency of 88 MHz while utilizing 27% of the slices and 95% of the dedicated 18x18 multipliers. Implementation results for the matrix inversion unit by itself are listed in Table 1, also showing results on a Xilinx Virtex-4 XC4VFX140 device while consuming 8% of the slices and 52% of the dedicated DSP48 multiply and accumulate (MAC) units with a top speed of 140 MHz.

	XC2V6000	XC2V8000	XC4VFX140
Slices	17%	12%	8%
Four-Input LUTs	12%	8%	5%
Block RAM	2%	2%	N/A
FIFO16/RAMB16	N/A	N/A	0%
MULT18x18	70%	60%	N/A
DSP48	N/A	N/A	52%
GCLK	6%	6%	3%
Max Clock Frequency (MHz)	108.3	92.1	140.0

Table 1 – Implementation results for the matrix inversion unit on Virtex-II and Virtex-4 FPGAs

Conclusion

MIMO techniques for 802.11 wireless LAN promise huge increases in bit rates. Several different approaches are still being debated by industry standardization bodies.

In collaboration between Laval University and Lyrtech, we implemented the LST technique, which exploits both spatial filtering and successive interference cancellation (SIC), in Virtex-II and Virtex-4 FPGAs on an 802.11n-oriented testbed. The next steps will be to optimize and test the LST implementation extensively over 2.4 GHz and 5 GHz channels, while the testbed will be extended to encompass other aspects of the upcoming 802.11n standard, including hardware-efficient LDPC codecs and beamforming techniques.

Space-Time Techniques

Given a single transmit antenna and two or more receive antennas, it is well known that spatial diversity can be exploited to improve link quality. Provided that the two receive antennas are sufficiently spaced, each will observe the received signal through a different multipath channel. Therefore, the probability that both channels experience a deep fade simultaneously is very low, and you can exploit this diversity advantage in a number of ways. One simple method is to utilize a linear weight-and-sum structure, which co-phases and weighs each branch in proportion to its SNR before adding them up. This is called maximal-ratio combining (MRC) and it results in an optimal SNR in the presence of white noise only, without other sources of interference.

Space-Time Coding

The first and simplest of the space-time block coding schemes was proposed in 1998 by Alamouti as a clever means to procure the same diversity advantage as MRC when the antenna array is situated at the transmitter instead of the receiver.

The original code is designed for twin antennas at the transmitter and a single antenna at the receiver, although various generalizations exist for arbitrarily sized antenna arrays at both ends. A pair of symbols is combined and sent twice in succession over two signaling intervals as two vectors, which are spatially orthogonal. This allows the single antenna to decode each symbol with a diversity order of two.

Given a pair of symbols s_1 and s_2, the transmitter/coding matrix is:

$$\mathbf{X} = \begin{bmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{bmatrix}$$

where each column corresponds to a signaling interval and each row to an antenna.

The received signal during the first and second signaling intervals are given by:

$$r_1 = h_1 s_1 - h_2 s_2^* + n_1$$

$$r_2 = h_1 s_2 + h_2 s_1 + n_2$$

where h_n is the channel (expressed as a complex gain according to baseband equivalent conventions) between the nth transmit antenna and the receiver, and $\{n_1, n_2\}$ are thermal white-noise components.

The receiver estimates h_1 and h_2 and decodes the first symbol by performing the following combination:

$$\begin{aligned} y_1 &= h_1^* r_1 + h_2 r_2^* \\ &= |h_1|^2 s_1 - h_1^* h_2 s_2^* + h_1^* h_2 s_2^* + |h_2|^2 s_1 + h_1^* n_1 + h_2 n_2^* \\ &= \left(|h_1|^2 + |h_2|^2\right) s_1 + v_1, \end{aligned}$$

where $v_1 = h_1^* n_1 + h_2 n_2^*$ is the noise component after combining.

Likewise, the second symbol is estimated as:

$$\begin{aligned} y_2 &= -h_2 r_1^* + h_1^* r_2 \\ &= \left(|h_1|^2 + |h_2|^2\right) s_2 + v_2 . \end{aligned}$$

Because each symbol's amplitude is proportional to the sum of the individual channel powers, this is indeed equivalent to MRC.

Layered Space-Time Architecture

The LST architecture was originally proposed by Foschini. With N antennas at both transmit and receive ends, this scheme allows the spatial multiplexing of N streams, while the receiver complexity grows only linearly with N.

Little or no processing is required at the transmitter. In the variant designated V-BLAST (Vertical Bell Labs LAyered Space-Time), each stream is simply assigned to a transmit antenna, so that for N=4 the transmission matrix is:

$$X = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & x_1^4 \cdots \\ x_2^1 & x_2^2 & x_2^3 & x_2^4 \cdots \\ x_3^1 & x_3^2 & x_3^3 & x_3^4 \cdots \\ x_4^1 & x_4^2 & x_4^3 & x_4^4 \cdots \end{bmatrix}$$

where x_n^k is the kth symbol in the nth stream.

D-BLAST (Diagonal BLAST) adds temporal staggering of the streams:

$$X = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & x_1^4 \cdots \\ 0 & x_2^1 & x_2^2 & x_2^3 \cdots \\ 0 & 0 & x_3^1 & x_3^2 \cdots \\ 0 & 0 & 0 & x_4^1 \cdots \end{bmatrix}$$

Although D-BLAST can improve performance when combined with FEC of individual streams, it is spectrally inefficient because of the zeros introduced. Threaded space-time (TST) eliminates this problem by simply rotating the assignment of streams:

$$X = \begin{bmatrix} x_1^1 & x_1^4 & x_1^3 & x_1^2 \cdots \\ x_2^1 & x_2^3 & x_2^4 & x_2^3 \cdots \\ x_3^1 & x_3^2 & x_3^1 & x_3^4 \cdots \\ x_4^1 & x_4^1 & x_4^2 & x_4^1 \cdots \end{bmatrix}$$

The received signal vector at a given instant is:

$$\mathbf{r} = \mathbf{Hx} + \mathbf{v}$$

where H is the NxN channel matrix, x is a column of X and v is the thermal noise vector. The received vectors corresponding to a whole frame are stored in a buffer for multi-pass processing. The individual received signals are then ordered according to their relative power. For the sake of simplicity and without loss of generality, the V-BLAST variety will be assumed henceforth, where each of the received signals corresponds directly to one of the streams at the transmitter.

Thus, the most powerful stream is detected first by passing the buffer contents through a linear combiner:

$$y_{(1)} = \mathbf{w}_{(1)}^H \mathbf{r}$$

where $<\cdot>^H$ denotes complex conjugate transposition of the weight vector $\mathbf{w}_{(1)}$ is typically chosen according to the zero-forcing criterion, which forces to zero the interfering contributions of the N-1 signals in $y_{(1)}$. The vector is given as:

$$\mathbf{w}_{(1)} = \mathbf{R}_{I_{(1)}}^{+} \mathbf{h}_{(1)}$$

where $<\cdot>^+$ denotes the pseudo-inverse, and

$$\mathbf{R}_{I_{(1)}} = \begin{bmatrix} \mathbf{h}_{(2)} & \mathbf{h}_{(3)} & \cdots & \mathbf{h}_{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{h}_{(2)} \\ \mathbf{h}_{(3)} \\ \vdots \\ \mathbf{h}_{(N)} \end{bmatrix}^*$$

is the interference covariance matrix from the point-of-view of the first stream denoted by the subscript (1).The symbols can then be detected with a high degree of reliability.

Given an estimate of the corresponding channel $\hat{\mathbf{h}}_{(1)}$, the first signal's contribution can be subtracted from the received signal buffer. Processing then moves on to the next layer, which targets the second most powerful signal. The buffer contents are then $\mathbf{r} - \hat{\mathbf{h}}_{(1)}\hat{x}_{(1)}$, where $\hat{\mathbf{h}}_{(1)}$ is an estimate of $\mathbf{h}_{(1)}$ and $\hat{x}_{(1)}$ is an estimate of the corresponding transmitted stream $x_{(1)}$.

It follows that $y_{(2)} = \mathbf{w}_{(2)}^H\left(\mathbf{r} - \hat{\mathbf{h}}_{(1)}\hat{x}_{(1)}\right)$. The process continues until, at the last layer, only the weakest signal x_N is left in the buffer. The contents are

$$\mathbf{r} - \sum_{n=1}^{N-1} \hat{\mathbf{h}}_{(n)}\hat{x}_{(n)} \approx \mathbf{r} - \sum_{n=1}^{N-1} \mathbf{h}_{(n)}x_{(n)} = \mathbf{h}_{(N)}x_{(N)} + \mathbf{v}$$

Because in the absence of detection errors no interference is left at this point, MRC is used:

$$\mathbf{w}_{(N)} = \mathbf{h}_{(N)}$$

Only the interference nulling capacity of the zero forcing array combining is used for the strongest signal. On the other hand, detection of the weakest signal relies on successive interference cancellation. A combination of nulling and cancelling applies for the intermediate signals. An interesting aspect of this approach is that the weakest signal enjoys the highest degree of spatial diversity.

Floating- to Fixed-Point Conversion of MATLAB Algorithms Targeting FPGAs

Accelerating fixed-point model generation and verification using the AccelDSP Synthesis tool.

by Tom Hill
Technical Marketing Engineer, DSP Tools Marketing
Xilinx, Inc.
tom.hill@xilinx.com

In a recent survey conducted by AccelChip Inc. (recently acquired by Xilinx), 53% of the respondents identified floating- to fixed-point conversion as the most difficult aspect of implementing an algorithm on an FPGA (Figure 1).

Although MATLAB is a powerful algorithm development tool, many of its benefits are reduced during the fixed-point conversion process. For example, new mathematical errors are introduced into the algorithm because of the reduced precision of the fixed-point arithmetic. You must rewrite code to replace high-level functions and operators with low-level models that reflect the actual hardware macro-architecture. And simulation run times can be as much as 50 times longer. For these reasons, MATLAB, the overwhelming choice for algorithm development, is often abandoned in favor of C/C++ for fixed-point modeling.

Generating Fixed-Point Models

The fixed-point representation of a floating-point MATLAB algorithm will not truly reflect the response of the final hardware if the high-level functions and operators are not replaced with hardware-accurate macro-architectures (Figure 2).

This is highlighted in Figure 3, which compares the fixed-point response of a MATLAB divide operator against a hardware-implementable CORDIC divide algorithm using a random set of input vectors quantized to 8-bit signed twos complement. Depending on the data values, a significant divergence exists between the calculated outputs.

During the fixed-point generation process, the AccelDSP™ Synthesis tool's IP Explorer™ technology will automatically replace high-level MATLAB functions and operators with hardware-accurate representations (Figure 4). This step is transparent and does not require MATLAB code modifications. You can redefine the initial macro- and micro-architecture selections by using a synthesis directive.

Once these operations have been replaced with hardware-accurate macro-architectures, the process of quantization may begin.

Graphically Assisted Auto-Quantization

The FPGA fabric, unlike a fixed-point DSP processor, allows for variable, fixed-point word lengths. By not limiting a variable to a fixed 16- or 24-bit boundary, you can perform arithmetic calculations requiring bit growth without incurring additional numeric error. This is a tremendous advantage for applications such as radar, navigation, and guidance systems that require a high degree of numeric precision.

In most cases bit growth rules are straightforward and well understood. The result of an addition, for example, grows by one bit and the result of a multiplication grows to a length equal to the sum of the input word lengths (Figure 5). Making these determinations for variables in an actual design, however, is a highly iterative process. Allowing unchecked bit growth to occur is expensive in hardware and generally unnecessary. If you're savvy, you can employ a variety of techniques to minimize word lengths while preserving numerical accuracy.

The process of determining an initial quantization value for a variable and the subsequent refinement of that value is well suited for automation. The AccelDSP Synthesis tool includes automated floating- to fixed-point conversion in which the floating-point MATLAB model is analyzed during simulation to determine the dynamic range requirements of the input data and constants. These values provide the starting points to an auto-quantization process that then draws upon a wealth of built-in experience, gained from more than 6,000 designs, to determine optimal word lengths for the downstream variables.

The initial fixed-point model obtained through auto-quantization provides a good starting point, but refinements

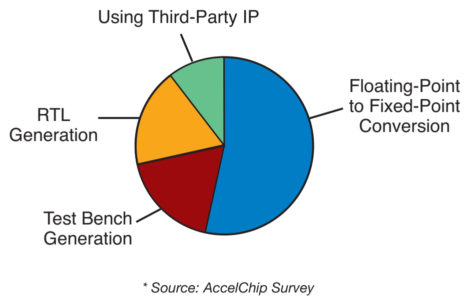


Figure 1 – AccelChip DSP design challenges survey

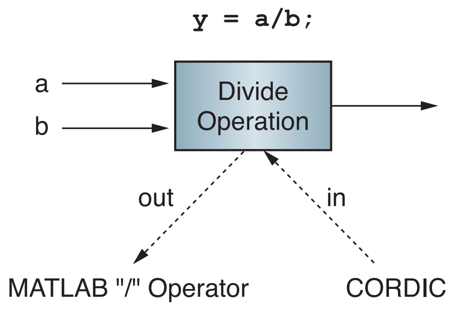


Figure 2 – Replacing built-in operators and functions

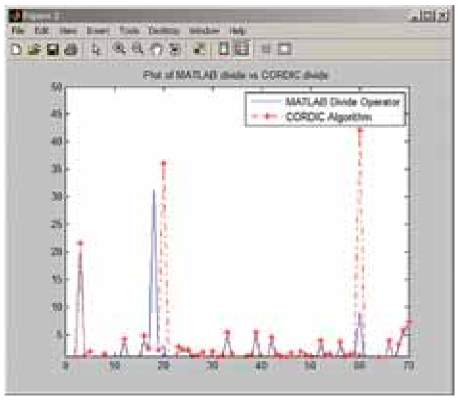


Figure 3 – Fixed-point response of the MATLAB "/" versus CORDIC

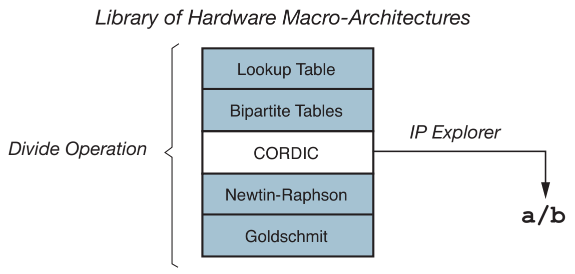


Figure 4 – Automatic hardware-accurate IP insertion

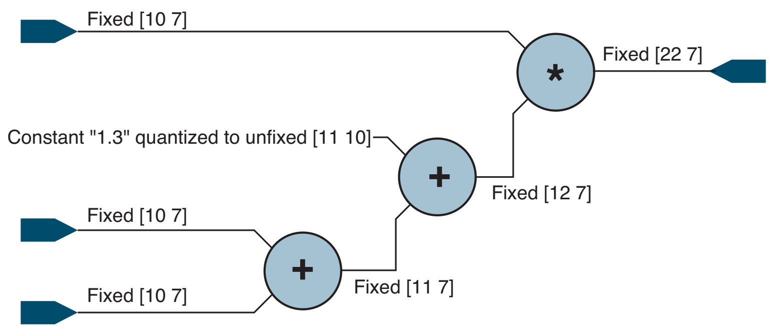


Figure 5 – Fixed-point bit growth

MATLAB provides a highly efficient environment for developing the mathematics of an algorithm that you can generally accomplish with a small set of simulation vectors.

to the model are generally necessary. This process is highly iterative and tightly coupled to analysis of the data effects. To minimize this iteration cycle time, the AccelDSP Synthesis tool provides an accelerated fixed-point simulation flow.

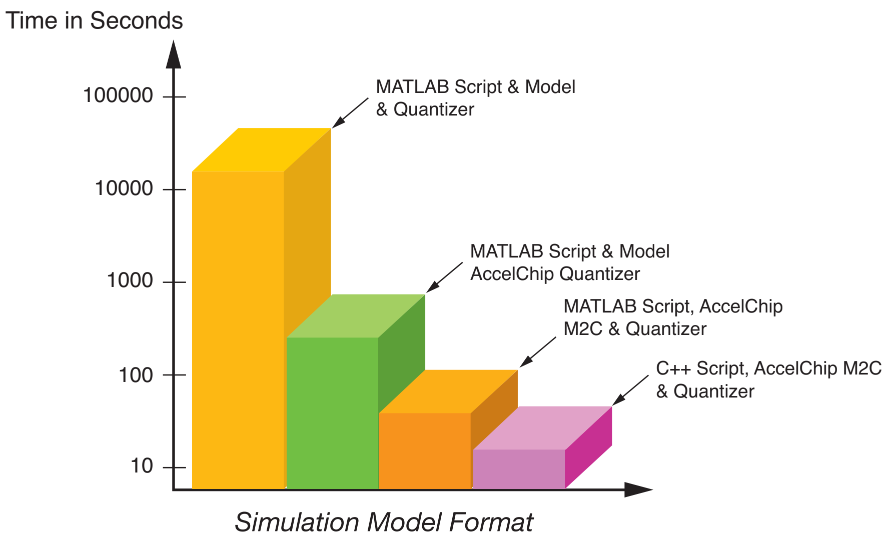


Figure 6 – FFT example simulation run times

Analyzing Fixed-Point Data Effects

MATLAB provides a highly efficient environment for developing the mathematics of an algorithm that you can generally accomplish with a small set of simulation vectors. When targeting that algorithm to fixed-point hardware, however, you will need increased data sets to accurately determine the real-world environment response. MATLAB, which is an interpreted simulator, may not provide the necessary performance for these larger, more CPU-intensive fixed-point simulations. For this, developers often turn to C/C++.

Accelerated Fixed-Point Simulation

The AccelDSP Synthesis tool's M2C-Accelerator automatically generates a hardware-accurate fixed-point C++ model and test bench to accelerate fixed-point simulations. Eliminating the manual recoding step saves development time and minimizes the introduction of errors. Because C++ is compiled, it can provide as much as a 1000x simulation performance advantage (Figure 6).

This level of performance is often necessary for the large vector sets required to understand fixed-point data effects.

If you wish to continue using the MATLAB visualization environment, including the plotting features, M2C-Accelerator also generates a fixed-point C/C++ dll that can be simulated with the original MATLAB test bench script file.

When you have obtained the initial fixed-point results, the process of analysis and refinement can begin. The AccelDSP Synthesis tool provides a set of graphical aids, including tabulated reports, variable probes, and plots to assist in this process.

Observing Fixed-Point Bit Growth

A design must be considered in its entirety to effectively convert a floating-point algorithm into a fixed-point model. If left unchecked early in the datapath, bit growth can quickly escalate to produce unreasonable hardware, while overly constrained bit growth may result in an unacceptable loss of numeric accuracy. A common technique to gain better observability into bit-growth progression is to enter the variables into a spreadsheet. The AccelDSP Synthesis tool provides this same level of observability by generating a tabular, formatted Fixed Point Report (Figure 7).

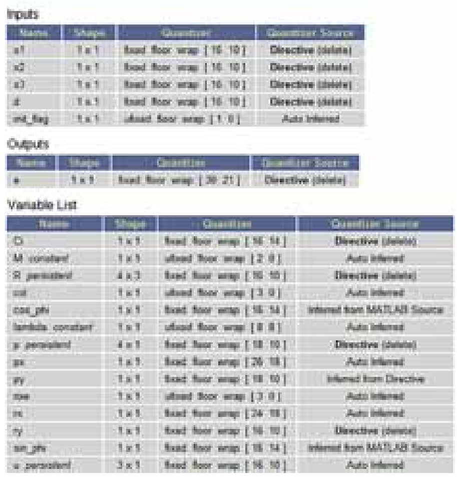
Before optimizing the hardware, you must obtain an acceptable fixed-point response. If the signal-to-noise ratio (SNR) of an output is not above a desired specification, then adjustments to the inferred quantization values are required. This process typically starts by looking for gross errors caused by variable overflows and underflows.

Overflows and Underflows

Poor assumptions about the dynamic range of the input data can lead to large fixed-point errors caused by overflowing the most significant bit (MSB) and (to a lesser degree) underflowing the least significant bit (LSB) of a variable. You will need to address these errors first before observing and correcting more subtle fixed-point errors.

Overflow and underflow reporting, inherent to MATLAB fixed-point data types, are not native to C/C++ and are often sacrificed during the model rewrite. The C++ models generated by M2C-Accelerator, however, include quantization routines that report all overflows and underflows encountered during a simulation. When these conditions occur, they are summarized in the "Verify FixedPoint Report" (Figure 8).

Once you have addressed any overflow and underflow issues, the refinement of the fixed-point model becomes more dependent on visualization. If additional fixed-point numeric errors persist, then



Inputs

Name	Shape	Quantizer	Quantizer Source
a1	1 x 1	fixed floor wrap [16 10]	Directive (delete)
a2	1 x 1	fixed floor wrap [16 10]	Directive (delete)
a3	1 x 1	fixed floor wrap [16 10]	Directive (delete)
d	1 x 1	fixed floor wrap [16 10]	Directive (delete)
init_flag	1 x 1	ufixed floor wrap [1 0]	Auto Inferred

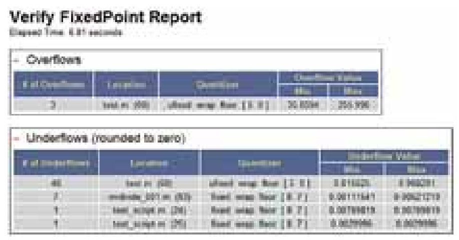
Outputs

Name	Shape	Quantizer	Quantizer Source
e	1 x 1	fixed floor wrap [30 21]	Directive (delete)

Variable List

Name	Shape	Quantizer	Quantizer Source
Ci	1 x 1	fixed floor wrap [16 14]	Directive (delete)
M constant	1 x 1	ufixed floor wrap [2 0]	Auto Inferred
R persistent	4 x 3	fixed floor wrap [16 10]	Directive (delete)
col	1 x 1	ufixed floor wrap [3 0]	Auto Inferred
cos_phi	1 x 1	fixed floor wrap [16 14]	Inferred from MATLAB Source
lambda constant	1 x 1	ufixed floor wrap [8 8]	Auto Inferred
p persistent	4 x 1	fixed floor wrap [18 10]	Directive (delete)
px	1 x 1	fixed floor wrap [26 18]	Auto Inferred
py	1 x 1	fixed floor wrap [18 10]	Inferred from Directive
row	1 x 1	ufixed floor wrap [3 0]	Auto Inferred
rx	1 x 1	fixed floor wrap [24 18]	Auto Inferred
ry	1 x 1	fixed floor wrap [16 10]	Directive (delete)
sin_phi	1 x 1	fixed floor wrap [16 14]	Inferred from MATLAB Source
u persistent	3 x 1	fixed floor wrap [16 10]	Auto Inferred

Figure 7 – AccelDSP Synthesis Fixed Point Report for an adaptive filter



Verify FixedPoint Report

Elapsed Time: 6.81 seconds

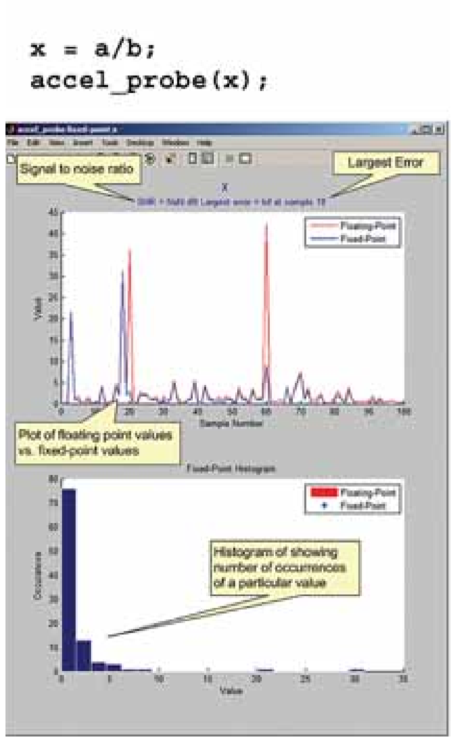
– Overflows

# of Overflows	Location	Quantizer	Overflow Value	
			Min	Max
3	test.m (69)	ufixed wrap floor [8 0]	30.8094	255.996

– Underflows (rounded to zero)

# of Underflows	Location	Quantizer	Underflow Value	
			Min	Max
40	test.m (68)	ufixed wrap floor [2 0]	0.01025	0.960231
7	mdivide_001.m (63)	fixed wrap floor [8 7]	0.00111541	0.00621219
1	test_script.m (24)	fixed wrap floor [8 7]	0.00789819	0.00789819
1	test_script.m (25)	fixed wrap floor [8 7]	0.0029996	0.0029996

Figure 8 – AccelDSP Verify FixedPoint Report



```
x = a/b;
accel_probe(x);
```

Signal to noise ratio
Largest Error
Floating-Point
Fixed-Point
Value
Sample Number
Plot of floating point values vs. fixed-point values
Fixed-Point Histogram
Histogram of showing number of occurrences of a particular value
Occurrences
Value

Figure 9 – Accel probe plot for a variable

you must analyze the effects of constants. Otherwise, you can continue the process of refining the hardware by reducing variable bit widths. In both cases, knowing the fixed-point error introduced by the quantization of a particular variable is a valuable aid in the refinement process.

Fixed-Point Visualization

Determining the appropriate fixed-point response of an algorithm to a given set of data is generally not an exact science. You will often have to make compromises in numerical accuracy to improve hardware efficiency. This process is highly iterative and tightly coupled to a visual analysis of the fixed-point effects displayed in plots. Observing an unacceptable SNR on an output signal, however, does not always indicate where a quantization value has been incorrectly specified. For that, additional analysis is necessary.

To assist in this process, AccelDSP Synthesis's AccelProbe graphically compares the floating- and fixed-point values for any variable during a given simulation (Figure 9). If you are using AccelProbe, you will quickly gain a sense of the magnitude that a particular variable's contribution makes to the cumulative error of the final result. You can "probe" a variable by adding the statement, "accel_probe(variable_name)" to the MATLAB source.

The "Fixed-Point Histogram" plot gives you a sense of how often a value may be encountered during simulation. The additional hardware required to store a value in the upper or lower dynamic range may be of little value if that value rarely occurs.

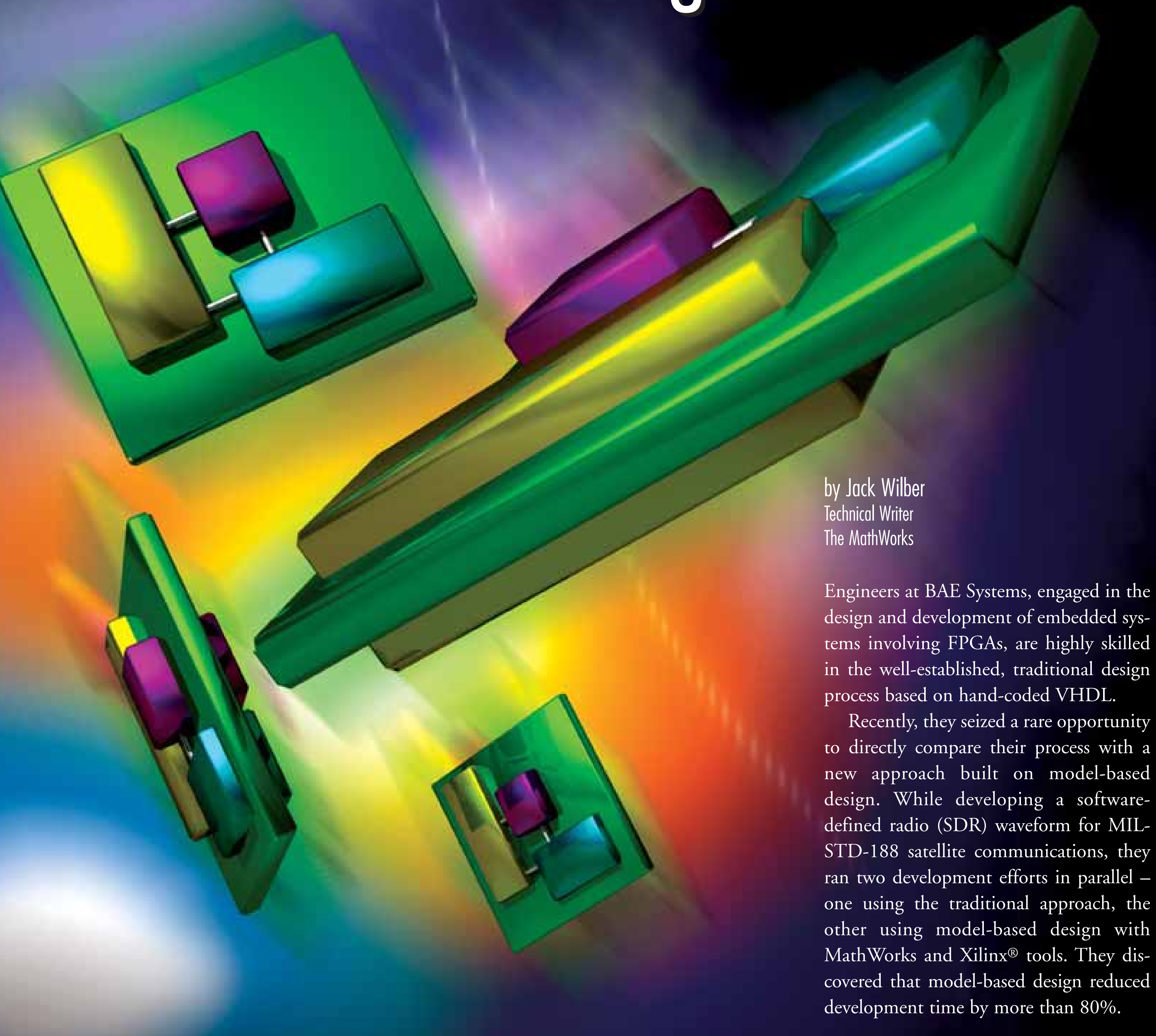
Conclusion

When inventing the mathematics of a DSP algorithm, MATLAB is the natural choice and should be used unencumbered by hardware considerations. Converting an algorithm into a fixed-point model for implementation on an FPGA is an involved process that benefits greatly from the automation, acceleration, and visualization offered by the AccelDSP Synthesis tool.

For more information about the AccelDSP Synthesis tool, visit *www.xilinx.com/dsp.*

BAE Systems Proves the Advantages of Model-Based Design

BAE Systems achieved an 80% reduction in software-defined radio development time with MathWorks and Xilinx tools.



by Jack Wilber
Technical Writer
The MathWorks

Engineers at BAE Systems, engaged in the design and development of embedded systems involving FPGAs, are highly skilled in the well-established, traditional design process based on hand-coded VHDL.

Recently, they seized a rare opportunity to directly compare their process with a new approach built on model-based design. While developing a software-defined radio (SDR) waveform for MIL-STD-188 satellite communications, they ran two development efforts in parallel – one using the traditional approach, the other using model-based design with MathWorks and Xilinx® tools. They discovered that model-based design reduced development time by more than 80%.

Traditional Design Flow vs. Model-Based Design

The traditional design flow at BAE Systems involves three distinct phases (Figure 1):

- The system engineering phase involves translating a set of system requirements to a system architecture and a waveform design to be implemented, for which a model is constructed and performance is verified against system requirements.
- The hardware engineering phase produces a second model of the algorithm, this time by hand-coding in VHDL, and requires a second round of simulation, debugging, and analysis to verify that this implementation matches the system engineering model.
- The physical design phase involves converting the VHDL behavioral model to an FPGA-compatible netlist, integrating onto the FPGA hardware and verifying that operation on the part matches expected behavioral performance.

Ideally, all of the information and detail defining the algorithm is carried forward from the system engineers to the hardware engineers who will implement the design, but there is inevitably some loss in this transfer. Engineers attempt to fully capture the detail of the design as a hardware specification and as input/output test vectors. Not only is this process quite lengthy, but it is also prone to error.

To overcome these limitations, engineers used model-based design with Simulink from The MathWorks and Xilinx System Generator to build a model that becomes an executable specification of the system for development teams to follow. This approach eliminates the need to pass written documentation to a software team for VHDL coding and reduces the three phases of the traditional design approach to two phases (Figure 2):

- The system and algorithm design phase now encompasses both the system and hardware engineering phases of the original process.
- The physical design phase, as before, focuses on integration and verification of performance on the hardware.

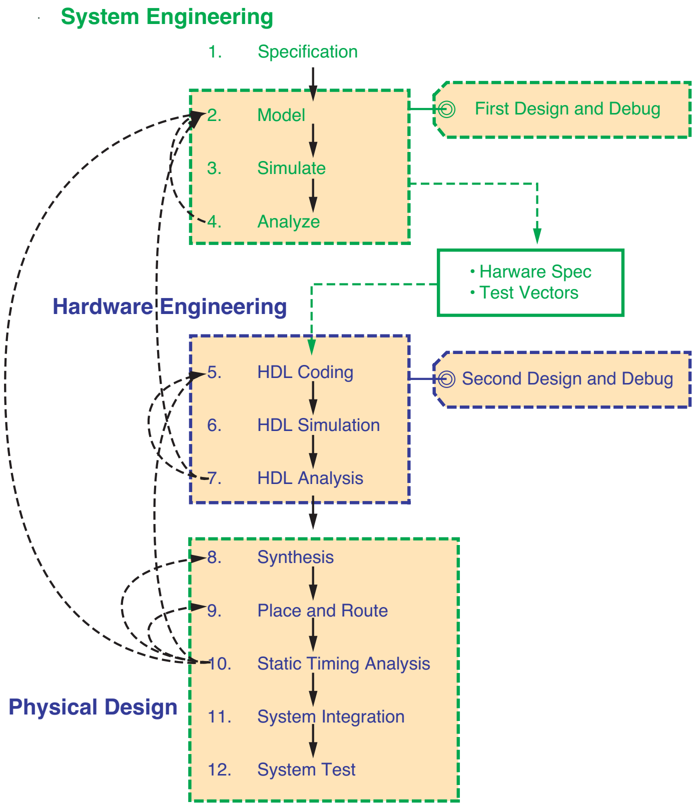


Figure 1 – The three phases of the traditional design flow at BAE Systems

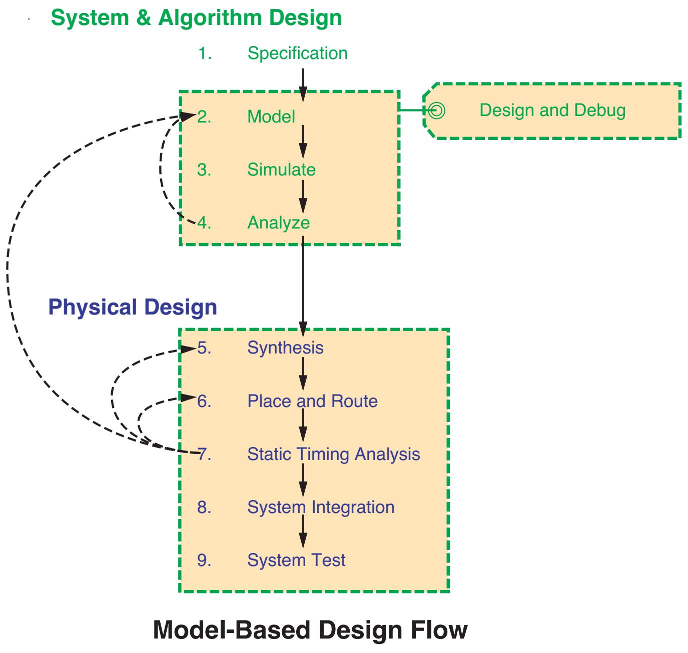


Figure 2 – Model-based design eliminates hand-coding in VHDL.

An Experiment in Concurrent Development

Two groups worked in parallel to develop the SDR waveform's signal-processing chain, hardware interfaces, and clocking. One group, led by Robert Regis, a senior engineer at BAE Systems with more than 15 years of experience in VHDL development, used the traditional workflow. The other, led by Sean Gallagher, a senior engineer at Xilinx, used model-based design.

Each group tracked the hours spent on the following tasks:

- Algorithm interface specification and documentation
- Module design definition
- Modeling, simulation, and design verification
- VHDL coding
- VHDL code behavioral verification
- Hardware integration and lab testing

Regis and Gallagher each implemented the same subset of MIL-STD-188-165a (Figure 3). In this design, the transmitted data passes through a scrambler, differential encoder, Reed-Solomon encoder, matrix interleaver, convolutional encoder, and quadrature amplitude modulation (QAM) modulator to produce baseband complex samples, which in the transmitter are passed to a pair of digital-to-analog converters (DAC) through low-voltage differential signal (LVDS) serial links.

The receive chain reverses these steps: the complex baseband samples received by a pair of analog-to-digital converters (ADC) pass through a QAM demodulator, Viterbi decoder, matrix deinterleaver, Reed-Solomon decoder, differential decoder, and descrambler to produce a serial bitstream. A sync detect function identifies Reed-Solomon block boundaries.

Both groups had access to an equivalent set of pre-validated components (IP cores): Regis used existing VHDL code for the Reed-Solomon encoder, while Gallagher used a Reed-Solomon encoder block available within System Generator. Similarly, both had access to a Reed-Solomon decoder, Viterbi decoder, and interleavers. In Regis's case these were available as IP cores, and in Gallagher's case they were incorporated through the inclusion of an associated System Generator block that, in turn, referenced and instantiated a Xilinx IP core.

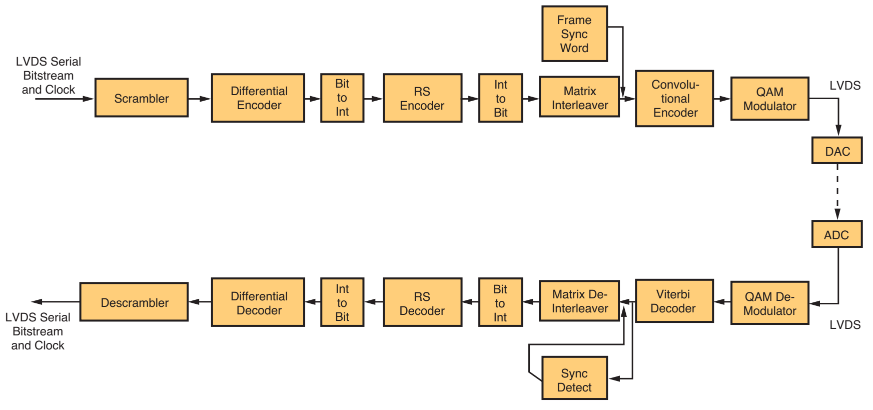


Figure 3 – A generic satellite communications transceiver

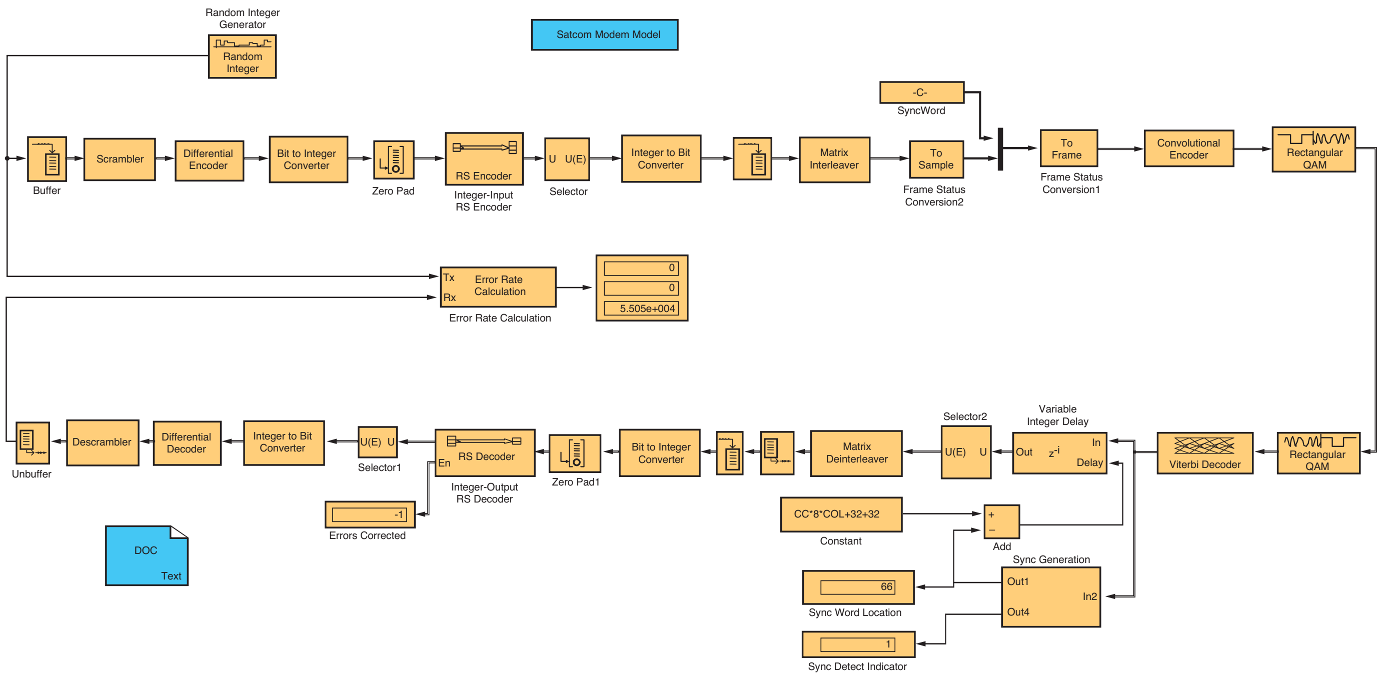


Figure 4 – Simulink model of the satellite communications transceiver

Model-Based Design with Simulink and System Generator

BAE Systems engineers modeled and simulated the transceiver waveform using Simulink and the Communications Blockset (Figure 4). They used frame-based processing in the model to increase simulation speed. (Frame-based connections cause the model to pass an entire frame, or packet of data, between blocks, thereby reducing block execution management overhead and increasing simulation speed.)

The Simulink model gave engineers the flexibility to implement the waveform on a range of targets – for example, they could have used Real-Time Workshop from The MathWorks to automatically generate code for a DSP implementation that met the performance requirements of the original specification. To ensure a valid comparison with the traditional design flow, however, they implemented the design on an FPGA.

The Simulink model was handed off to Gallagher, who used it as a reference design in building the equivalent model in Xilinx System Generator. Gallagher used existing Xilinx blocks for Reed-Solomon encoding, matrix interleaving, and Viterbi decoding. He built other high-level blocks for which there was no direct substitute using lower-level Xilinx blocks (Figure 5).

Gallagher used scopes and bit-error-rate meters to debug the model and verify operational performance before using Xilinx System Generator to automatically generate VHDL and synthesize the FPGA.

The waveform involves the processing of single bits or scalar, complex values produced by the QAM modulator. In the receiver, real soft symbols produced by the QAM demodulator are represented using finite precision values. Therefore, unlike the floating-point double-precision values that exist in the Simulink model, in the System Generator model it was necessary to select the scaling to be used to represent these values. Gallagher selected the scaling parameters based on trial and error, until bit widths were as small as possible without a significant impact to performance.

Regis's team required 645 hours to complete the design and development of the signal-processing chain (Table 1). Gallagher needed only 46 hours, a reduction in development time greater than 10 to 1. Factoring in equal amounts of hardware integration and lab testing – estimated at 137 hours – the end-to-end improvement is still greater than 4 to 1.

David Haessig, a senior BAE Systems technical staff member who was involved in both efforts, points out that in addition to the obvious time savings provided by automatic code generation, model-based design accelerated the project by shifting debugging and analysis forward in the schedule. "When following the traditional design flow process, a large portion of the simulation and debugging work tends to occur later in the design, during VHDL coding," Haessig says. "With model-based design the model defines the code, and you are therefore obliged to include in the model every detail needed to define the waveform.

"Typically the model is built and tested incrementally, and you deal with the bugs and the algorithmic issues as they occur. Debugging is handled almost entirely during the modeling phase of the design, with a bit-true, cycle-true model. With access to Similink and MATLAB data visualization tools, bugs are much easier to identify and fix prior to VHDL. The alternative, debugging the VHDL code, is more difficult and tedious."

Further Advantages of Model-Based Design

BAE Systems identified three additional factors that contributed to the substantial reduction in development time achieved using model-based design: clocking, defect discovery, and component interfaces.

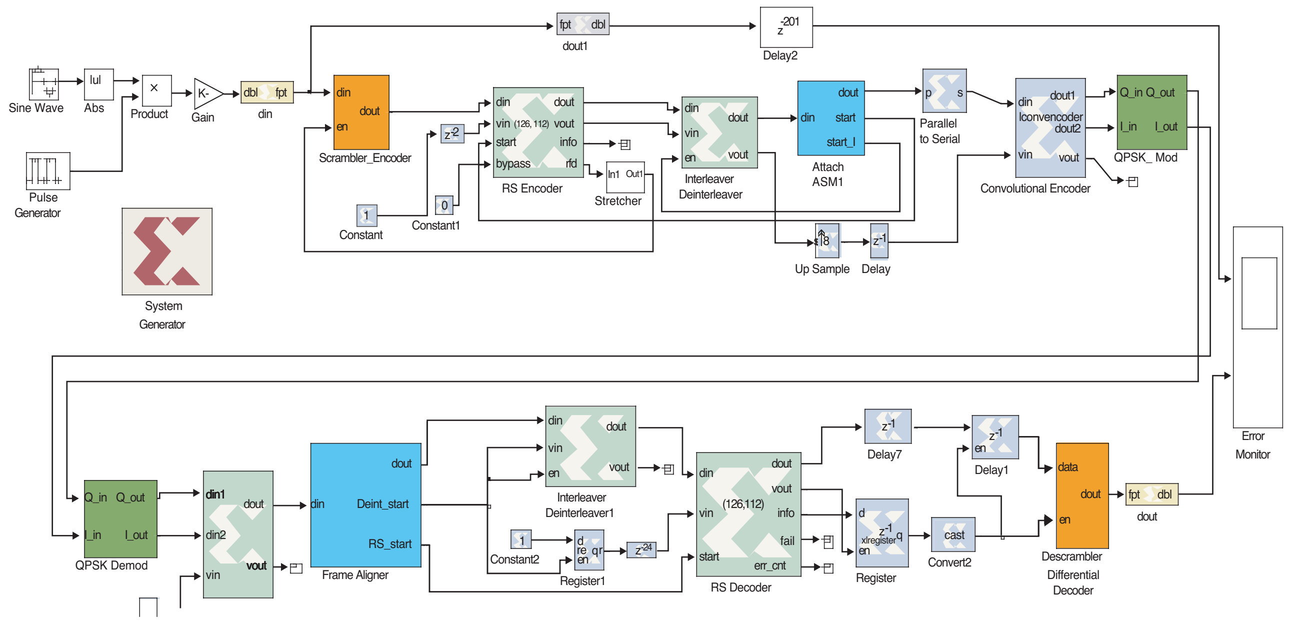


Figure 5 – Simulink model of the satellite communications transceiver using Xilinx blocks.

In the traditional design flow, Regis spent substantial time hand-calculating a combination of clock enables and multiple clock domains necessary for generating the odd sample rates associated with Reed-Solomon encoding and decoding. In contrast, Gallagher relied on Xilinx System Generator to automatically derive a common clock at the highest rate and build enabling logic to throttle multiple rates.

As reflected in the time logged for verification in each approach, finding and repairing defects was greatly simplified in the approach using model-based design. "With Simulink and System Generator, the model is directly connected to the resulting code, which enables you to discover bugs at the modeling stage using source and sink blocks, not at the VHDL behavioral test stage using test benches," Haessig explains.

Regis notes that while he had to carefully read the interface specifications of each block he used, Gallagher could wire blocks together with relative ease. "With System Generator, it is amazing how easily the IP blocks are connected together. There is no need to study data sheets or clocking and control options. This may be the most underrated aspect of model-based design."

Looking Ahead: SCA-Compliant SDR

Based on the results of the experiment, Haessig expects model-based design with MathWorks tools to become an integral part of the BAE Systems software development process.

In addition, BAE Systems is exploring ways to use model-based design tools to enable waveform portability by automatically generating software communications architecture (SCA)-compliant software and firmware. Joint Tactical Radio System (JTRS) radios must follow SCA as a standard for achieving waveform portability. Responding to a request from the Joint Program Executive Office (JPEO), BAE Systems is involved in the deployment of waveforms that JPEO can use in current and future radio systems without redeveloping the waveform components. The same SDR code will be able to run on new hardware platforms without modification.

BAE Systems is also working with The MathWorks, Virginia Tech, Xilinx, and Zeligsoft to create an interface that will enable code generated by Real-Time Workshop and System Generator to be directly incorporated into SCA-compliant radios. According to Haessig, "This initiative holds great potential for reducing time to market and the cost of JTRS radio development, allowing seamless transition from simulation to an SCA-compliant implementation; increasing reusability and portability of components; and expanding the lifespan of source code."

For more information, please visit the aerospace and defense (*www.mathworks.com/industries/aerospace/*) and communications (*www.mathworks.com/industries/comms/*) sections of The MathWorks website, or contact Dan Raun at *dan.raun@mathworks.com*, (508) 647-7098 or Mike McHenry at *mike.mchenry@mathworks.com*, (508) 647-7858.

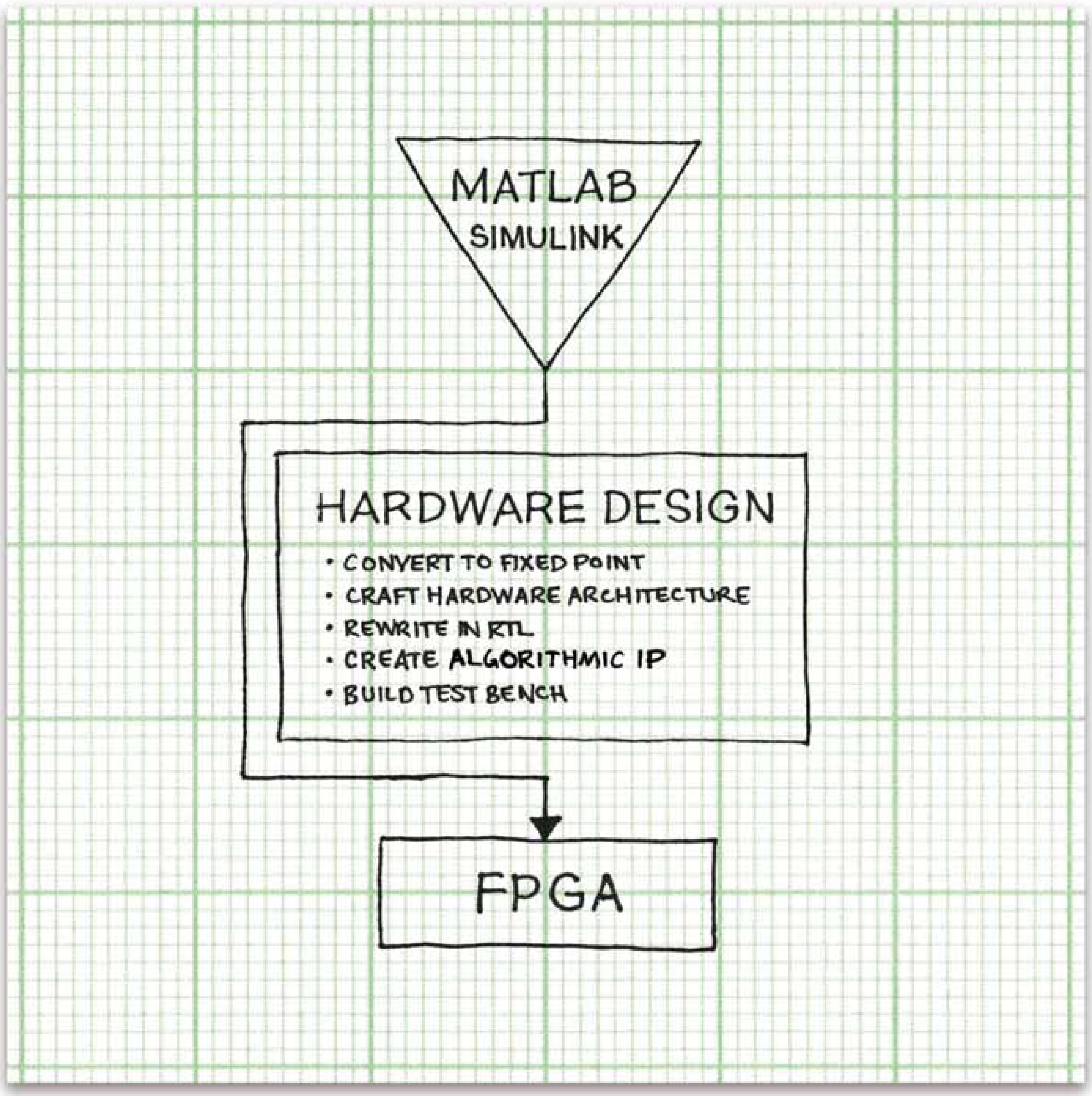
Traditional Approach (hours worked)

	Algorithm Interface Specification/ Documentation	Module Design Definition	Modeling, Simulation, and Design Verification	VHDL Coding	VHDL Code Behavioral Verification	Hardware Integration & Lab Testing	Notes
Reed-Solomon RS Encode	40	40	0	40	60	20	Integrate purchased IP
Reed-Solomon Decode	20	80	0	60	100	20	Integrate purchased IP
Scrambler / Descrambler	1	1	0	1	6	3	
Convolutional Encode	1	1	0	1	1	1	
Viterbi Decode	8	8	0	8	16	24	Integrate inhouse IP, development not shown
Differential Encoder / Decoder	1	1	0	1	4	2	
Interleaver / Deinterleaver	40	16	0	16	36	60	
PSK Modulator (2,4,8)	5	5	0	4	3	3	
RS Frame Sync	4	6	0	4	6	4	
TOTALS:	**120**	**158**	**0**	**135**	**232**	**137**	**782**

Rapid Development Approach (hours worked)

	Algorithm Interface Specification/ Documentation	Module Design Definition	Modeling, Simulation, and Design Verification	VHDL Coding	VHDL Code Behavioral Verification	Hardware Integration & Lab Testing	
Reed-Solomon RS Encode	1	0.25	2	0	0	*	
Reed-Solomon Decode	1	0.5	3	0	0	*	
Scrambler / Descrambler	0	0.25	3	0	0	*	
Convolutional Encode	0	0.25	1.5	0	0	*	
Viterbi Decode	0	0.5	2	0	0	*	
Differential Encoder / Decoder	0	0.25	1	0	0	*	
Interleaver / Deinterleaver	0	0.5	2	0	0	*	
PSK Modulator (2,4,8)	1	0.5	4	0	0	*	
RS Frame Sync	1	4	16	0	0	*	
TOTALS:	**4**	**7**	**34.5**	**0**	**0**		**45.5**

Table 1 – Table of results



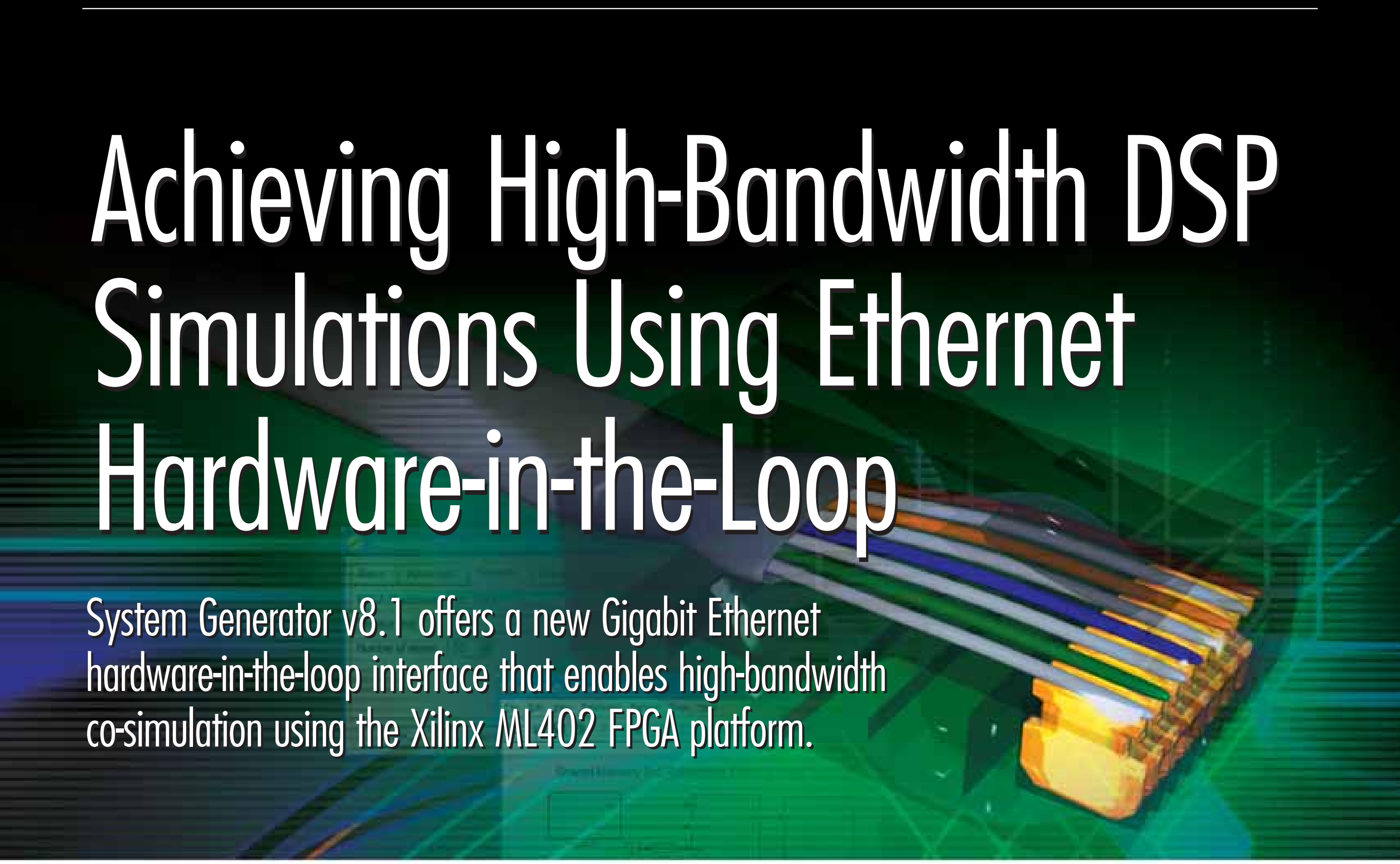
NOW, THERE'S A FLOW YOU COULD GET USED TO.

XtremeDSP™ Sidestep tedious hardware development tasks with Xilinx System Generator for DSP and AccelDSP design tools. When combined with our library of algorithmic IP and hardware platforms, Xilinx XtremeDSP Solutions get you from design to verified silicon faster than ever.

Visit ***www.xilinx.com/dsp*** today and speed your design with AccelDSP and System Generator for DSP.

XILINX®

www.xilinx.com/dsp



Achieving High-Bandwidth DSP Simulations Using Ethernet Hardware-in-the-Loop

System Generator v8.1 offers a new Gigabit Ethernet hardware-in-the-loop interface that enables high-bandwidth co-simulation using the Xilinx ML402 FPGA platform.

by Ben Chan
Software Engineer II
Xilinx, Inc.
ben.chan@xilinx.com

Nabeel Shirazi
Senior Staff Software Engineer
Xilinx, Inc.
nabeel.shirazi@xilinx.com

Jonathan Ballagh
Staff Software Engineer
Xilinx, Inc.
jonathan.ballagh@xilinx.com

Designers often struggle with lengthy simulation times when designing large FPGA signal-processing systems. FPGA design tools such as Xilinx® System Generator for DSP aim to address this challenge by providing robust hardware-in-the-loop interfaces that allow you to bring FPGA hardware directly into your design simulations.

By emulating a portion of your design in hardware, these interfaces enable considerable simulation acceleration – typically by an order of magnitude or more. Using hardware-in-the-loop also brings you real-time FPGA hardware debugging and verification capabilities.

System Generator for DSP offers hardware-in-the-loop interfaces for many types of FPGA development platforms. These platforms typically expose different types of physical interfaces through which the PC communicates with the FPGA hardware. For example, a JTAG co-simulation interface allows any FPGA board with a JTAG header and Xilinx FPGA to be co-simulated inside System Generator for DSP. Other boards, such as the XtremeDSP™ Development Kit, communicate over a PCI bus connection. Until recently, co-simulation of systems with high memory bandwidth and throughput requirements (such as video and image processing) were limited exclusively to development boards that connected directly to a PC using PCI or PCMCIA interfaces.

Co-Simulation over Ethernet

System Generator for DSP 8.1 includes a new Ethernet co-simulation interface that for the first time brings high-bandwidth co-simulation capabilities to the Xilinx ML402 Evaluation Platform. The ML402 board connects to your PC either directly using a standard Ethernet cable or remotely across a network.

At the heart of the interface is the Xilinx tri-mode Ethernet MAC core, which supports operation in 10/100/1000 Mbps half- and full-duplex modes. When you generate a design using the Ethernet hardware co-simulation interface, System Generator for DSP automatically wraps your design with the logic necessary to communicate with the FPGA over an Ethernet connection during simulation (Figure 1).

You may generate a design for Ethernet hardware co-simulation by double-clicking on the System Generator block in any design to open its configuration parameters box. From the compilation menu, select the ML402/Ethernet compilation (see Figure 2) under the hardware co-simulation menu. You can choose between two different modes of Ethernet co-simulation.

Network-Based Co-Simulation

A network-based interface allows you to co-simulate FPGA hardware that is

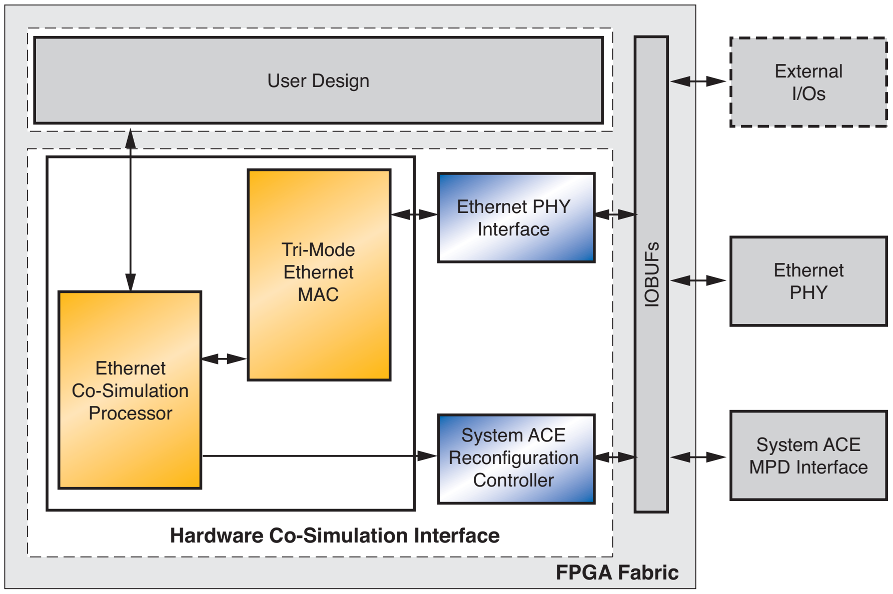
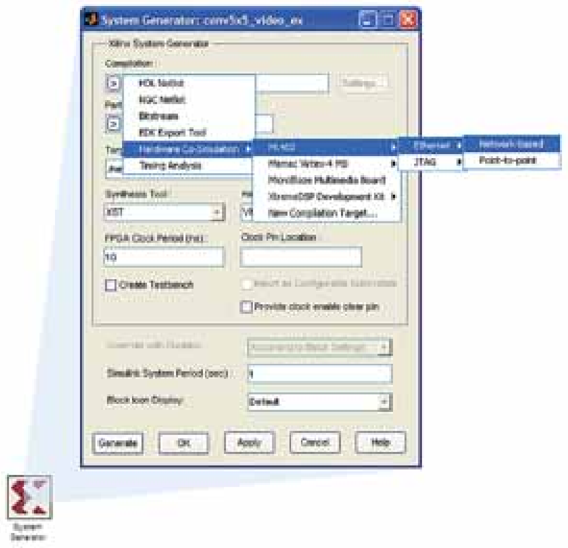


Figure 1 – Block diagram of FPGA fabric using Ethernet hardware co-simulation interface



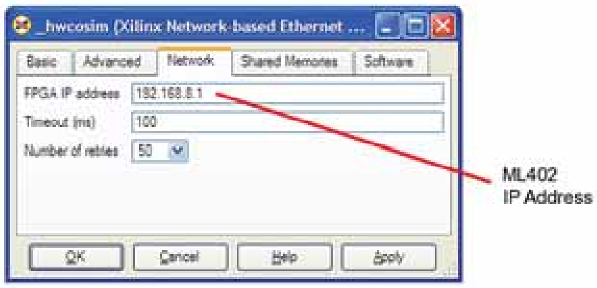
Synthesis Tool: XST

FPGA Clock Period (ns): 10

Simulink System Period (sec): 1

Block Icon Display: Default

Figure 2 – Selecting Ethernet hardware co-simulation as the System Generator compilation type



FPGA IP address: 192.168.8.1

Timeout (ms): 100

Number of retries: 50

Figure 3 – Specifying the IP address of an ML402 board for Ethernet hardware co-simulation

attached to a standard IPv4 network. Because these networks are virtually ubiquitous, the network-based interface provides a convenient way to reach a remote FPGA development board connected to a wired or wireless network. The interface manages the details of communication and error handling (retransmissions after packet loss) behind the scenes. System Generator for DSP uses the IP address of an ML402 board to determine which platform to communicate with during co-simulation (Figure 3).

Point-to-Point Co-Simulation

The second Ethernet co-simulation mode is a point-to-point interface that uses raw Ethernet frames to enable high-bandwidth communication with the ML402 board over the data link layer. In contrast to the network-based counterpart, the point-to-point interface focuses on low-level communications on a local Ethernet segment. Co-simulation data is transmitted across a standard UTP Ethernet cable that connects the ML402 board directly to the PC. This means that you must have an available Ethernet jack exposed on your PC to make the connection.

The point-to-point interface supports the Gigabit Ethernet standard, which when configured to use jumbo frames substantially bolsters the performance of large data transfers. Using this interface allows you to co-simulate even the most bandwidth-intensive applications.

Device Configuration

Both Ethernet co-simulation interfaces support a novel approach to device configuration, using the Xilinx System ACE™ solution to support configuration over an Ethernet cable. The configuration process is performed over the same Ethernet connection used for co-simulation, thus eliminating the need for a second programming cable (such as a Xilinx Parallel Cable IV or Platform Cable USB). A CompactFlash card is installed on the ML402 board and contains a special boot-loader image that is automatically loaded into the FPGA at power up. This image allows the FPGA to be reconfigured with new FPGA co-simulation bitstreams that are transferred over the Ethernet cable at the start of a simulation. The entire configuration process is handled transparently by System Generator for DSP.

Design Example

A 5x5 filter operator design model named conv5x5_video_ex is included with the System Generator for DSP 8.1 software tool. This design shows how a 2D image filter can be realized efficiently using n-tap MAC FIR filters. The System Generator for DSP top-level design is shown in Figure 4.

Also included with the design is a hardware co-simulation test bench for streaming a looped video sequence through the 5x5 kernel at real-time frame rates. During each simulation cycle, individual video frames are transmitted to the FPGA for processing. Once in the FPGA, each frame is filtered using a 5x5 kernel, and then transmitted back to the PC for analysis in Simulink.

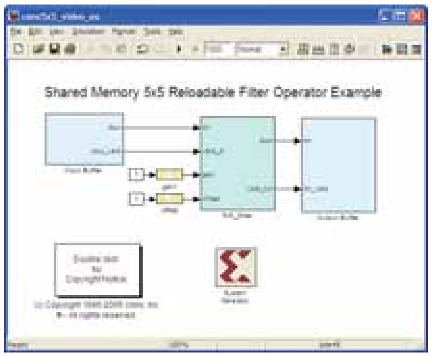


Figure 4 – System Generator for DSP 5x5 filter operator example

Two Simulink Matrix Viewer blocks display the unfiltered and filtered images during simulation. Data flow through the test bench is shown in Figure 5.

Benchmarking

The 5x5 filter design example was compiled for point-to-point Ethernet hardware co-simulation and co-simulated using the Xilinx ML402 development board. The simulation speed in hardware was compared against the simulation speed in software. Specifically, the benchmark considers the number of processed frames being read back per second, and the results are compared to the software simulation time of the filter operation on a single frame.

Figure 6 summarizes the simulation speedup achieved through Ethernet co-simulation with respect to a pure software simulation. The results show a significant speedup in simulation by a factor of approximately 50 to 1,000 times. In reality, the achievable speedup may vary based on different factors: design complexity, number of I/O ports, and volume of I/O data. The figure also reflects two other important factors regarding the Ethernet settings – the link speed and maximum frame size – that can affect co-simulation performance.

With the increase of link speed, we see a dramatic reduction in simulation time because more bandwidth is available for co-simulation data. With jumbo frames enabled on a gigabit connection, the co-simulation performance is further bolstered by increasing the maximum frame size to ensure the greatest efficiency of burst data transfers.

Conclusion

The System Generator for DSP Ethernet hardware co-simulation interfaces provide convenient, high-bandwidth solutions for simulating video and image processing applications on the Xilinx ML402 platform. These interfaces make it possible to simulate remote FPGA platforms, or for higher performance, a board attached directly to the host PC using an Ethernet cable. By using the SystemACE solution, device configuration is accomplished over an Ethernet connection, eliminating the need for a second programming cable. As shown from the benchmark results, the interface can enable simulation speedups by several orders of magnitude.

Both the Ethernet co-simulation interfaces and video processing reference design are distributed with the Xilinx System Generator v8.1 software tool.

To learn more about System Generator and Ethernet co-simulation, see the User Guide at *www.xilinx.com/system_generator.htm.*

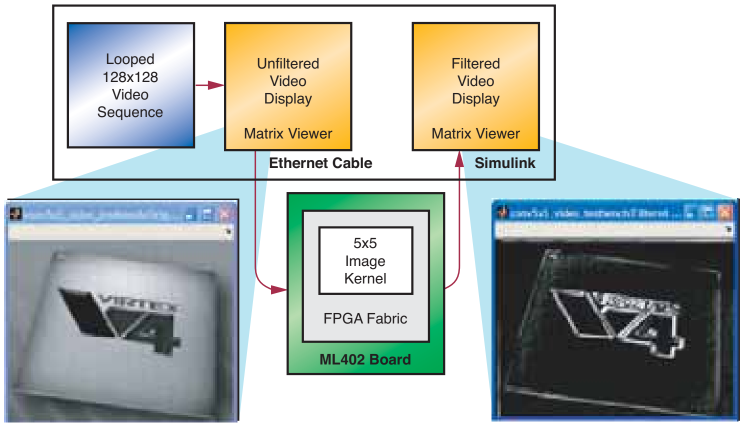


Figure 5 – System Generator for DSP 5x5 filter streaming video test bench

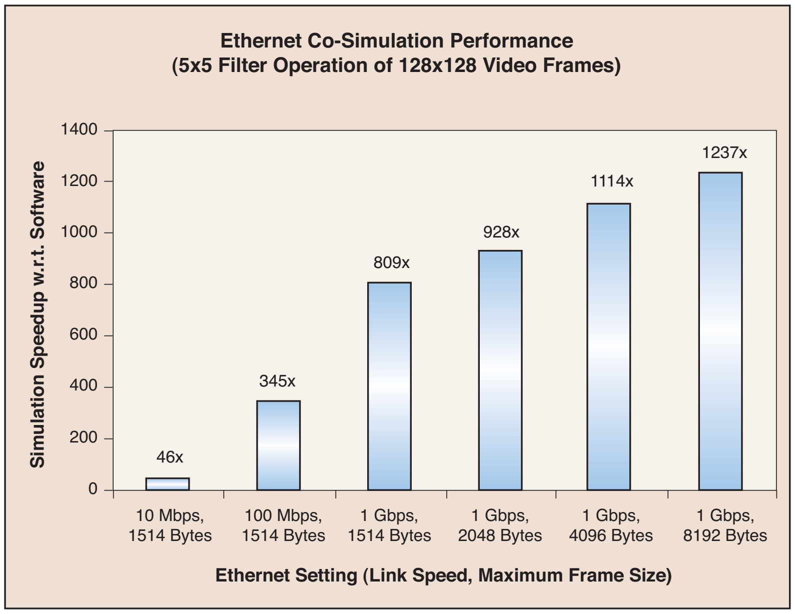
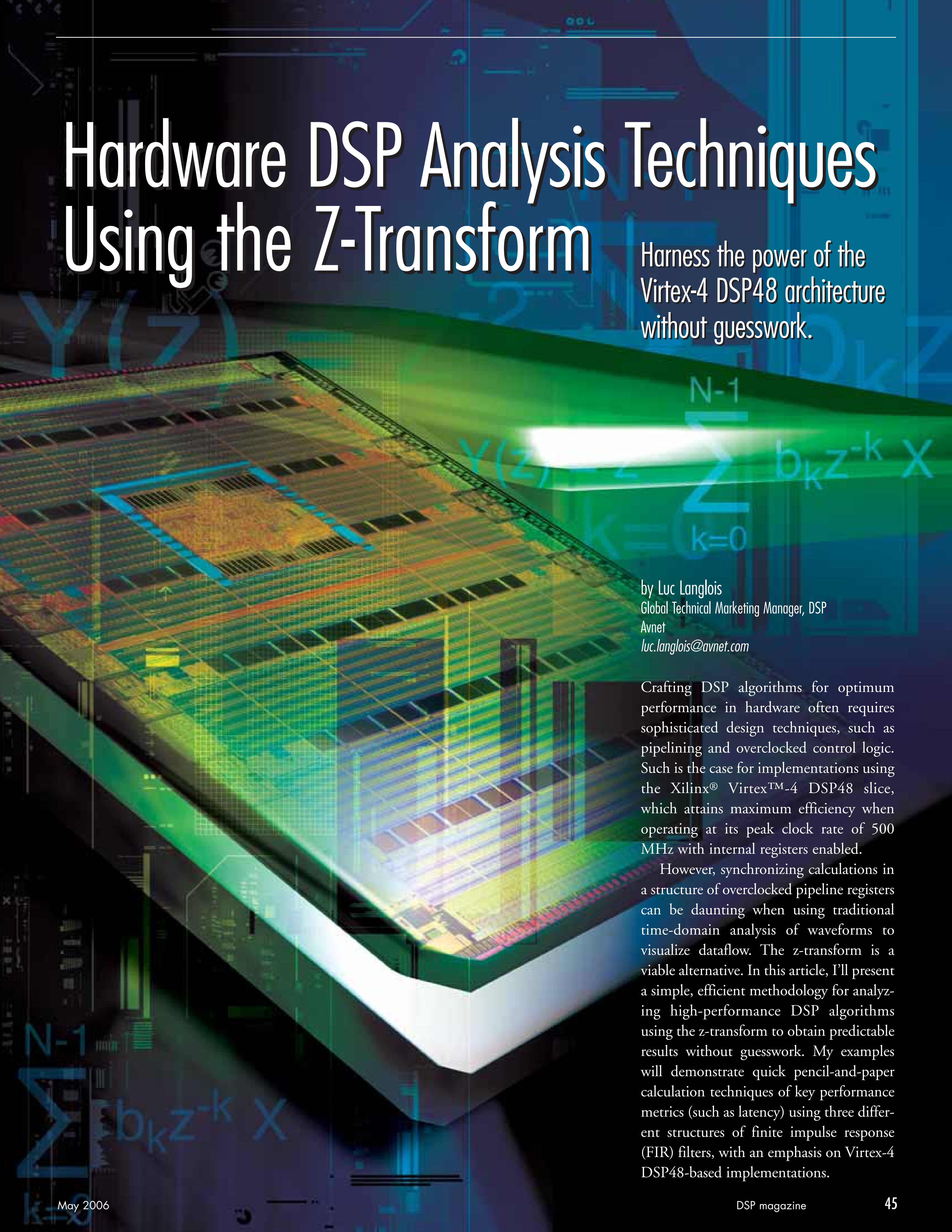


Figure 6 – System Generator for DSP 5x5 filter benchmark results



Hardware DSP Analysis Techniques Using the Z-Transform

Harness the power of the Virtex-4 DSP48 architecture without guesswork.

by Luc Langlois
Global Technical Marketing Manager, DSP
Avnet
luc.langlois@avnet.com

Crafting DSP algorithms for optimum performance in hardware often requires sophisticated design techniques, such as pipelining and overclocked control logic. Such is the case for implementations using the Xilinx® Virtex™-4 DSP48 slice, which attains maximum efficiency when operating at its peak clock rate of 500 MHz with internal registers enabled.

However, synchronizing calculations in a structure of overclocked pipeline registers can be daunting when using traditional time-domain analysis of waveforms to visualize dataflow. The z-transform is a viable alternative. In this article, I'll present a simple, efficient methodology for analyzing high-performance DSP algorithms using the z-transform to obtain predictable results without guesswork. My examples will demonstrate quick pencil-and-paper calculation techniques of key performance metrics (such as latency) using three different structures of finite impulse response (FIR) filters, with an emphasis on Virtex-4 DSP48-based implementations.

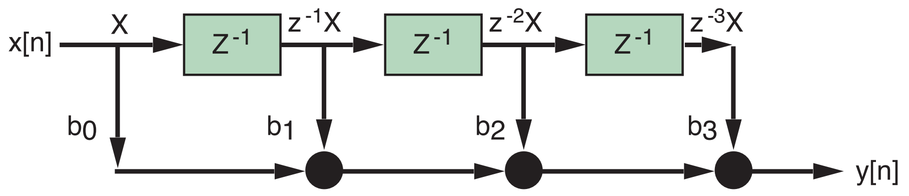


Figure 1 – Signal flow graph of direct-form FIR filter

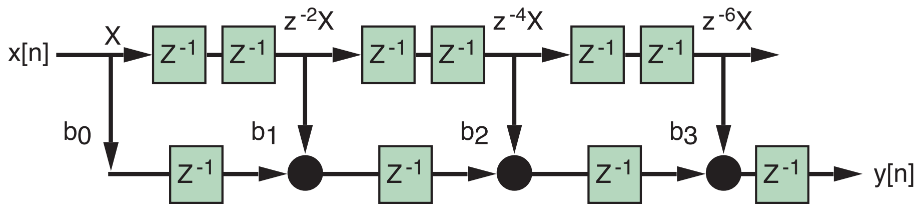


Figure 2 – Signal flow graph of parallel systolic FIR filter

The Z-Transform

DSP uses the z-transform to operate on sampled signals in discrete time, as opposed to the Laplace and Fourier transforms used for analog signals in continuous time. Hardware designers will recognize the standard notation, z^{-1} for a unit-sample delay, commonly implemented with a register. This refers to an important property of the z-transform: a delay in the time domain corresponds to the z-transform of the signal without delay, multiplied by a power of z in the frequency domain. The expression of this relationship between a signal delayed by k unit samples and its z-transform is:

$$x[n-k] \leftrightarrow z^{-k} X(z)$$

The Signal Flow Graph

The signal flow graph is a time-tested tool for visualizing DSP algorithms. Figure 1 is the signal flow graph of a direct-form FIR filter.

Three elements comprise a signal flow graph:

- Branch node: sends a copy of the input signal to several output paths
- Summing node: outputs the sum of all signals flowing into it
- Delay element: $\rightarrow\boxed{z^{-1}}$ stores a delayed sample of the input signal

Note the filter coefficients $b_{0...3}$, which multiply the signal flowing through each branch. For simplicity, multipliers are not explicitly shown.

Analysis

Referring to Figure 1, I recommend the following method to analyze the signal flow graph using z-transforms:

1. Annotate each node of the time-skew buffer with the z-transform of the input signal, multiplied by increasing negative powers of z as the signal moves through delay elements.

2. Derive the output by tracing the signal through the graph, multiplying the input signal by the coefficient in each branch, and summing the resulting products in the summing nodes:

$$Y(z) = b_0X + b_1z^{-1}X + b_2z^{-2}X + b_3z^{-3}X$$

3. Simplify:

$$Y(z) = \sum_{k=0}^{N-1} b_k z^{-k} X$$

The result is the familiar FIR filter sum-of-products, equivalent to discrete-time convolution in the time domain:

$$y[n] = x[n]*b[n].$$

Pipelining for Performance

Applying the analysis method previously discussed to a high-performance FIR filter structure known as the parallel systolic form results in Figure 2. It is derived by pipelining the time-skew buffer and the adder chain of the direct form, producing a structure that maps naturally to the Virtex-4 DSP48 slice (for more details, see Chapter 4 of the XtremeDSP User Guide).

A glance at this structure would suggest extra latency compared to the direct form of Figure 1, but how can you quickly determine the exact amount of latency without the time-consuming exercise of actually building a model for simulation? With pencil and paper, a three-step analysis process using z-transforms will answer this in no time.

1. The z-transform annotated time-skew buffer is shown in Figure 2. Note the even powers of z in the time-skew buffer because of the double registers.

2. Derive the output by tracing the signal through the graph. The trick here is to recognize that each signal crossing of a register from left to right in the adder tree causes a z^{-1} to multiply the entire bracketed expression thus far:

$$Y(z) = \{[(b_0z^{-1}X + b_1z^{-2}X)z^{-1} + b_2z^{-4}X]z^{-1} + b_3z^{-6}X\}z^{-1}$$

3. Simplify:

$$Y(z) = z^{-4}\sum_{k=0}^{N-1} b_k z^{-k} X$$

Again the familiar FIR filter sum-of-products appears, revealing a latency of four sample periods when factored out.

Splitting the Unit Delay

The semi-parallel FIR filter is a structure of time-shared DSP48s, each operating on a subset of coefficient taps at an overclocked computation rate f_{clk} relative to the data sampling rate f_s (also referred to as throughput). Extensive pipelining (displayed as red squares) allows clocking of the DSP48 at its maximum computation rate (500 MHz in a Virtex-4 -12 speed grade device), for an optimal trade-off of

With a simple, efficient methodology using the z-transform for the analysis of DSP algorithms, you can easily apply high-performance techniques such as pipelining and overclocked control logic to your hardware DSP designs.

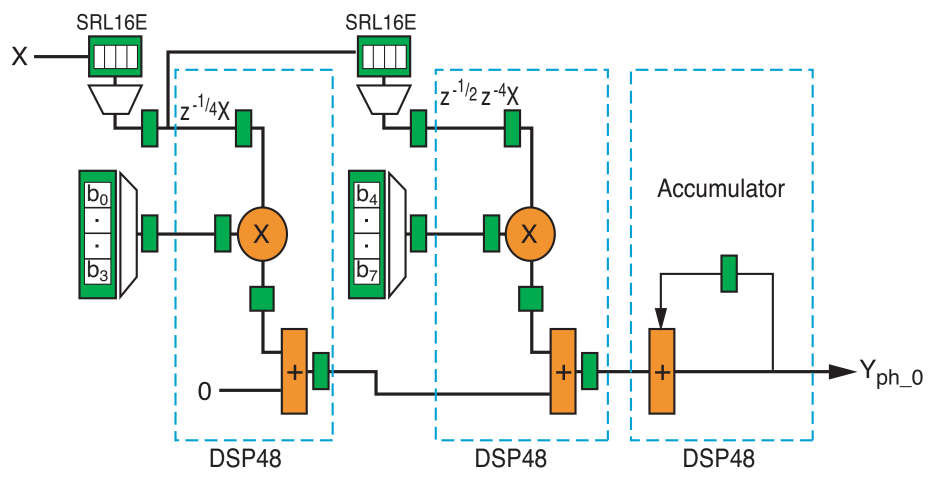


Figure 3 – Eight-tap, semi-parallel FIR filter with 4x overclocking (fclk/fs = 4)

throughput versus filter order. The defining quantity is taps/DSP48 = f_{clk} / f_s = number of computation phases required to compute each output value Y (Chapter 5 of the XtremeDSP User Guide).

Figure 3 shows an 8-tap semi-parallel FIR structure using DSP48s, with 4x overclocking. The time-skew buffer is a cascade of addressable shift registers with shift enable, known as SRL16E, in FPGA fabric. The enable signal (not shown) is asserted every $1/f_s$ to shift data through the time-skew buffer at the sampling rate, resulting in unit-delays (whole powers of z).

At each of four computation phases, shift register addressing operating at $f_{clk} = 4f_s$ (not shown) selects one of four data samples in each SRL16E and presents it to the DSP48 input register, while the corresponding coefficient, denoted b_k, is fetched from distributed memory. The four computation phases are summed sequentially to produce the output Y in the accumulator, which is then cleared to start calculation of the next output. Overclocking all registers at $f_{clk} = 4f_s$ accounts for the fractional-delays $z^{-1/4}$.

Aligning the computation phases to ensure proper synchronization of these operations can be a daunting task using time-domain waveforms to visualize dataflow in simulation. Z-transforms reduce the job to simple algebra by grouping unit and fractional delays as lumped sums in the exponent, as follows:

1. The z-transform annotated time-skew buffer is shown in Figure 3 for the first of four computation phases. Note the combination of fractional and whole powers of z in the time-skew buffer.

2. Each 4x overclocked register causes the signal to accumulate a $z^{-1/4}$ delay. The post-adder register applies its $z^{-1/4}$ delay by multiplying the entire bracketed signal expression thus far. The first computation phase Y_{ph_0} is :

$$Y_{ph\_0} = (b_0 z^{-1}X + b_4 z^{-5})\, z^{-1/4}$$

effect of post-adder register

The full output expression of the 8-tap FIR filter is the accumulated sum of the four consecutive computation phases, each of which is shown delayed an extra $1/(4f_s)$ from the previous phase in the sequence:

$$\begin{aligned} Y = {} & (b_0 z^{-1}X + b_4 z^{-5})\, z^{-1/4} && \text{(1st computation phase)} \\ & + z^{-1/4}(b_1 z^{-2}X + b_5 z^{-6}X)\, z^{-1/4} && \text{(2nd computation phase)} \\ & + z^{-1/2}(b_2 z^{-3}X + b_6 z^{-7}X)\, z^{-1/4} && \text{(3rd computation phase)} \\ & + z^{-3/4}(b_3 z^{-4}X + b_7 z^{-8}X)\, z^{-1/4} && \text{(4th computation phase)} \end{aligned}$$

The accumulator has the effect of "realigning" each computation phase to produce the sequentially accumulated sum Y at the next $1/f_s$ boundary. This is accounted for with extra fractional delay:

$$\begin{aligned} Y = {} & z^{-3/4}\ (b_0 z^{-1}X + b_4 z^{-5}X)\, z^{-1/4} \\ & + z^{-1/2}\ z^{-1/4}(b_1 z^{-2}X + b_5 z^{-6}X)\, z^{-1/4} \\ & + z^{-1/4}\ z^{-1/2}(b_2 z^{-3}X + b_6 z^{-7}X)\, z^{-1/4} \\ & + z^{0}\ \ z^{-3/4}(b_3 z^{-4}X + b_7 z^{-8}X)\, z^{-1/4} \end{aligned}$$

computation phases re-aligned in the accumulator

3. Simplify:

$$Y(z) = z^{-2} \sum_{k=0}^{N-1} b_k z^{-k} X$$

Again the familiar FIR filter sum-of-products appears, revealing a latency of two sample periods when factored out.

Conclusion

With a simple, efficient methodology using the z-transform for the analysis of DSP algorithms, you can easily apply high-performance techniques such as pipelining and overclocked control logic to your hardware DSP designs.

The techniques described in this article were presented at Speedway 2005 DSP sessions. The Spring 2006 Speedway Design Workshop Series features two new DSP-related workshops: "Xilinx DSP Development Workshop" and "Xilinx DSP for Video Workshop." For more information, visit *http://em.avnet.com/xlxspringspeedway.*

Implementing Optimal Filters Quickly

You can obtain high performance with minimal resources in Virtex-4 FPGAs using the new FIR Compiler.

by Niall Battson
DSP Technical Marketing Manager
Xilinx, Inc.
niall.battson@xilinx.com

Although the well-known finite impulse response (FIR) filter algorithm is extremely simple, the number of variants in the implementation specifics is immense. These implementation specifics have kept research institutions busy and DSP hardware engineers struggling to reach optimal performance and usage of the silicon available to them.

Introduced in September 2004, Xilinx® Virtex™-4 devices demonstrated that 400 MHz DSP designs (in the slowest speed grade) were feasible, especially for designs that had an abundance of FIR filters. This is certainly the case in wireless and defense systems today, especially in the radio portion.

The FIR Compiler v1.0 is a new line of powerful and comprehensive IP from Xilinx. The FIR Compiler allows hardware engineers and DSP algorithm engineers to rapidly generate the high-performance filters that Virtex-4 devices promise.

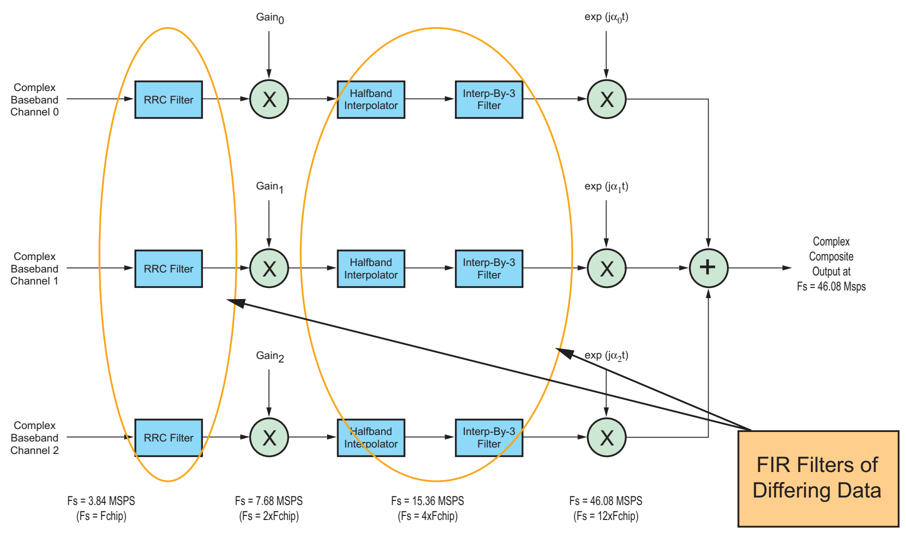
Figure 1 shows a typical three-carrier UMTS digital up converter, of which about 60% of the design is consumed by the FIR filters. This significant 2x performance improvement over Virtex-II Pro FPGAs enables DSP designs to shrink in resource utilization significantly – often more than 50% – allowing support for more channels, functionality, and a lower power or cost solution.

At the forefront of this performance leap is the XtremeDSP™ slice (also referred to as the DSP48). The XtremeDSP slice is a unique high-performance multiplier and arithmetic unit with great flexibility, laid out in a column structure in the FPGA with dedicated cascade routing between each slice.

However, to take advantage of the significant improvements in Virtex-4 DSP, hardware engineers must adopt a new implementation style for their FIR filters. This implementation is based on an adder chain architecture and takes specific advantage of the XtremeDSP slice. But as the FIR filter is one of the most ubiquitous and fundamental building blocks in DSP systems, the amount of time spent understanding and reworking old designs can easily become very significant. To alleviate this impact to hardware engineers and accelerate time to market for those adopting DSP in FPGAs, Xilinx has created the FIR Compiler.

The FIR Compiler v1.0 is a new line of powerful and comprehensive IP from Xilinx. The FIR Compiler allows hardware engineers and DSP algorithm engineers to rapidly generate the high-performance filters that Virtex-4 devices promise. Furthermore, the FIR Compiler allows you to make trade-offs between differing high-performance hardware implementations of your FIR filter specification.

- Adder-chain based multiply accumulate FIR (MACFIR)
- Adder tree-based MACFIR
- Distributed arithmetic FIR (DAFIR)



Complex Baseband Channel 0
Complex Baseband Channel 1
Complex Baseband Channel 2
RRC Filter
Gain$_0$
Gain$_1$
Gain$_2$
X
Halfband Interpolator
Interp-By-3 Filter
exp (jα_0t)
exp (jα_1t)
exp (jα_2t)
+
Complex Composite Output at Fs = 46.08 Msps
FIR Filters of Differing Data
Fs = 3.84 MSPS (Fs = Fchip)
Fs = 7.68 MSPS (Fs = 2xFchip)
Fs = 15.36 MSPS (Fs = 4xFchip)
Fs = 46.08 MSPS (Fs = 12xFchip)

Figure 1 – FIR filters consume the majority of digital radio designs.

The more traditional adder tree-based MACFIR architecture is still an excellent fit for low-cost Spartan™ devices, as these devices do not have the XtremeDSP slice. The DAFIR is extremely valuable for low-bit-width applications and logic-slice-heavy FPGAs.

These easy trade-offs give you the ability to select the most resource- and power-efficient solutions.

Trade-Offs and Optimizations

One of the fundamental trade-offs that the FIR Compiler enables is data rate versus area. For example, a 16-bit single-rate 64-tap filter will yield three very different results, depending on the data rate required. Comfortable with the knowledge that high clock frequencies of 400 MHz are easily obtainable with the FIR Compiler, Figures 2, 3, and 4 illustrate the most optimum structures for 6.35 MHz, 25 MHz, and 100 MHz data rates, respectively.

Note how the available clock cycles in the lower data rate designs are exploited to result in the smaller resource solutions. Also note how the XtremeDSP slice is used and the cascade routing exploited by the adder chain structures for more than single multiplier implementations.

(For more detailed information on these architectures, please refer to the Virtex-4 XtremeDSP Slice User Guide.)

The trade-off between data rate and area is very simply made in the FIR Compiler tool through the sample frequency parameter in the GUI interface (see Figure 5). The structures and size of the filter that can be implemented are very different based on the data rate requirement. The FIR Compiler makes automatic decisions about whether to use a block memory or distributed memory structure, as well as the amount of multipliers required to meet the sample frequency entered. These automatic capabilities keep resource usage to a minimum and greatly reduce design time for filter implementations.

Higher Clock Performance, Smaller Designs

Clearly, one of the most valuable aspects of the FIR Compiler is its ability to shrink the size of a design by exploiting the performance on the silicon in the FPGA. Figures 2, 3, and 4 demonstrate that resources are kept to a minimum by achieving high clock frequencies. It also means a higher data-rate capability for the parallel filter shown in Figure 4.

Figure 6 really emphasizes this point, adjusting the clock frequency for a 33-tap filter while maintaining a constant sample rate of 10 MSPS. It also compares the differences between symmetrical and non-symmetrical coefficient sets, emphasizing the benefits offered by both. Overall, you should aim to maximize clock frequency, as the significant reduction in area cannot be overlooked.

The Complexity of the FIR Compiler

FIR filter specifications are, however, more complex than what I have discussed so far. As shown in Figure 1, both interpolation and multi-channel capabilities are critical in designing the system. Multiple channels of data are very common in video (red, green, and blue); wireless communications (antenna diversity, adaptive antenna arrays); and general DSP processing (complex data).

You can exploit these multiple streams of data and use a single FIR filter structure in a time-division multiplexed fashion to filter the channels. This provides a significant resource utilization reduction over multiple instances of the same filter. However, the clock frequency must run faster for a multi-channel filter versus a single-channel filter; specifically, the number of channels multiplied by the clock frequency. The promised high-performance clock capabilities of the FIR Compiler make implementing these multi-channel filters feasible, easy to generate in the tool, and greatly reduce resource utilization.

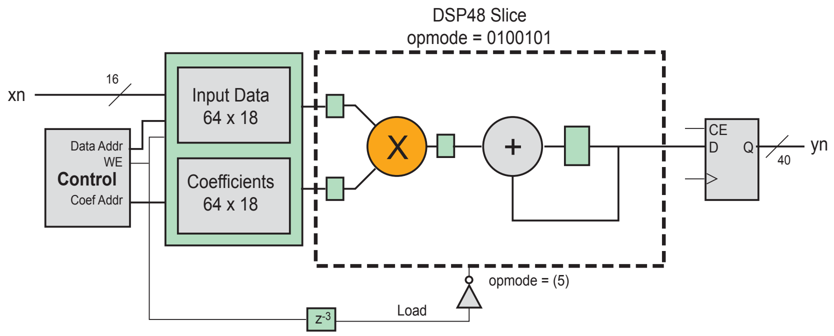


Figure 2 – 6.35 MSPS single-rate 64-tap FIR filter

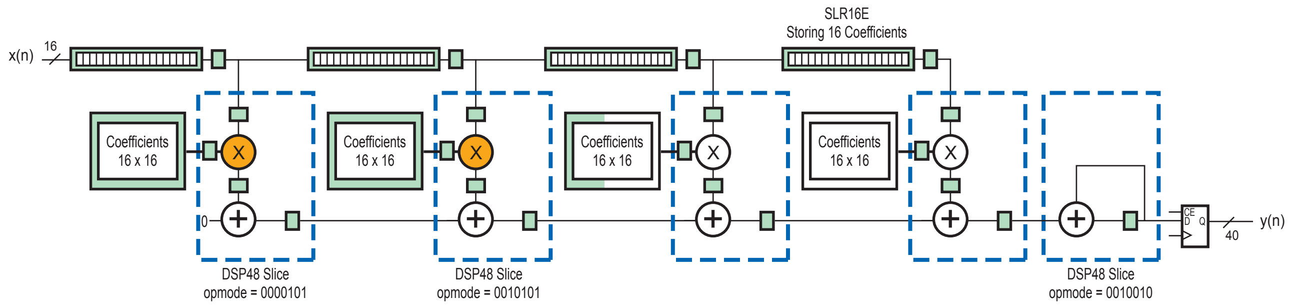


Figure 3 – 25 MSPS single-rate 64-tap FIR filter

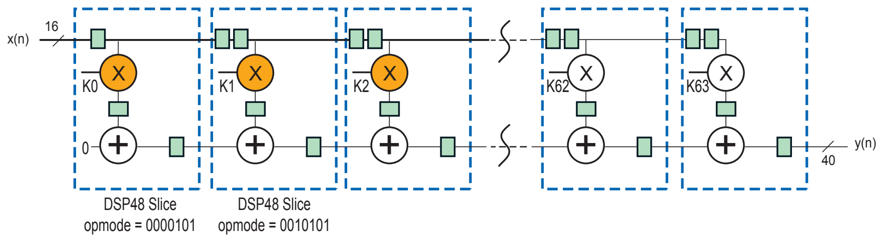
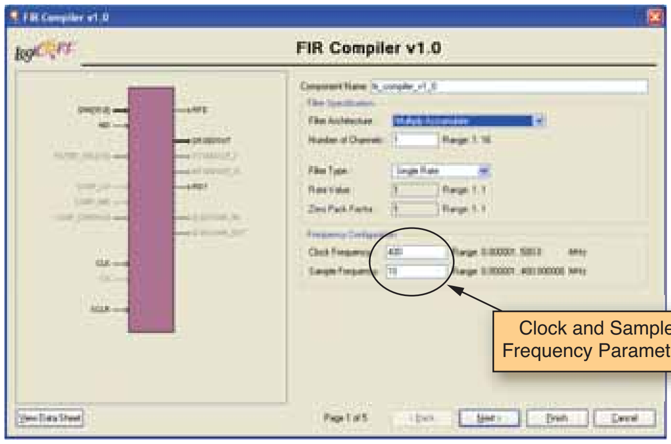


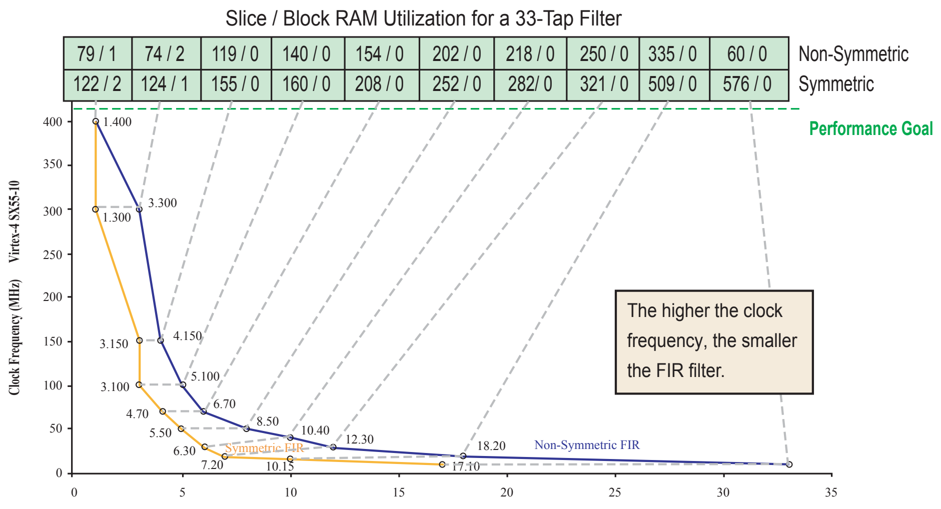
Figure 4 – 400 MSPS single-rate 64-tap FIR filter

Multi-rate filters are also extremely common in DSP designs, especially in digital radios (demonstrated in Figure 1). With these filters, the input and output data rates are not the same. For an interpolation filter, the output is larger by a factor of the interpolation ratio; in a decimation filter, the output is smaller by a factor of the decimation ratio. You can exploit these differences in input and output frequencies by using a well-known technique that creates what is known as a



Component Name: fir_compiler_v1_0
Filter Specification
Filter Architecture: Multiply Accumulate
Number of Channels: 1 Range: 1..16
Filter Type: Single Rate
Rate Value: 1 Range: 1..1
Zero Pack Factor: 1 Range: 1..1
Frequency Configuration
Clock Frequency: 400 Range: 0.00001..5813 MHz
Sample Frequency: 10 Range: 0.00001..400.000000 MHz

Figure 5 – First page of FIR Compiler GUI



79 / 1	74 / 2	119 / 0	140 / 0	154 / 0	202 / 0	218 / 0	250 / 0	335 / 0	60 / 0	Non-Symmetric
122 / 2	124 / 1	155 / 0	160 / 0	208 / 0	252 / 0	282/ 0	321 / 0	509 / 0	576 / 0	Symmetric

Figure 6 – 33-tap 10 MSPS FIR filter resource utilization for differing clock performance

Filter Type	Clock Frequency (MHz)	Resource Utilization: Slices	Resource Utilization: DSP48	Resource Utilization: Block RAM
395 MSPS, 128-Tap, Decimate by 4, Single-Channel, 16-Bit-Data FIR Filter	395	500	33	0
3.5 MSPS, 196-Tap, Interpolate by 2, 8-Channel 12-Bit-Data FIR Filter	399	300	15	14
22 MSPS, 20-Tap, Single-Rate, 3-Channel, 18-Bit-Data FIR Filter	400	123	5	0
3.84 MSPS, 47-Tap, Interpolate by 2 RRC, 6-Channel, 16-Bit-Data FIR Filter	305	234	4	0
7.68 MSPS, 23-Tap, Half-Band Interpolator, 6-Channel, 16-Bit-Data FIR Filter	333	119	1	2

Table 1 – Designs generated using the FIR Compiler.

polyphase filter to reduce the computational requirement. This polyphase filter technique, added to the high clock frequency performance and automatic generation, means that the FIR Compiler can rapidly create extremely resource- and power-efficient multi-rate filters. The tool even has the ingenuity to optimize the structure even further if the filter is a half-band multi-rate filter, as required in digital radios. Any hardware engineer implementing digital radios will take advantage of these capabilities.

In addition to the filter types I've described, the FIR Compiler also offers the ability to change coefficients in the filter on the fly. This is very important in video filtering and agile digital radio receivers. You can select a fully reloadable coefficient filter or a filter that contains as many as 16 different coefficient sets; a control port selects the set being employed. Once again, the FIR Compiler can make an optimal choice between block memory or distributed memory.

Furthermore, you can combine all of these capabilities to provide a very wide range of possible filters, with the most complicated being multiple channel, multirate FIR filters with reloadable coefficients. Table 1 shows performance and resource utilization for numerous complicated FIR filters, including the filters in Figure 1, and demonstrates the capabilities of the FIR Compiler.

Conclusion

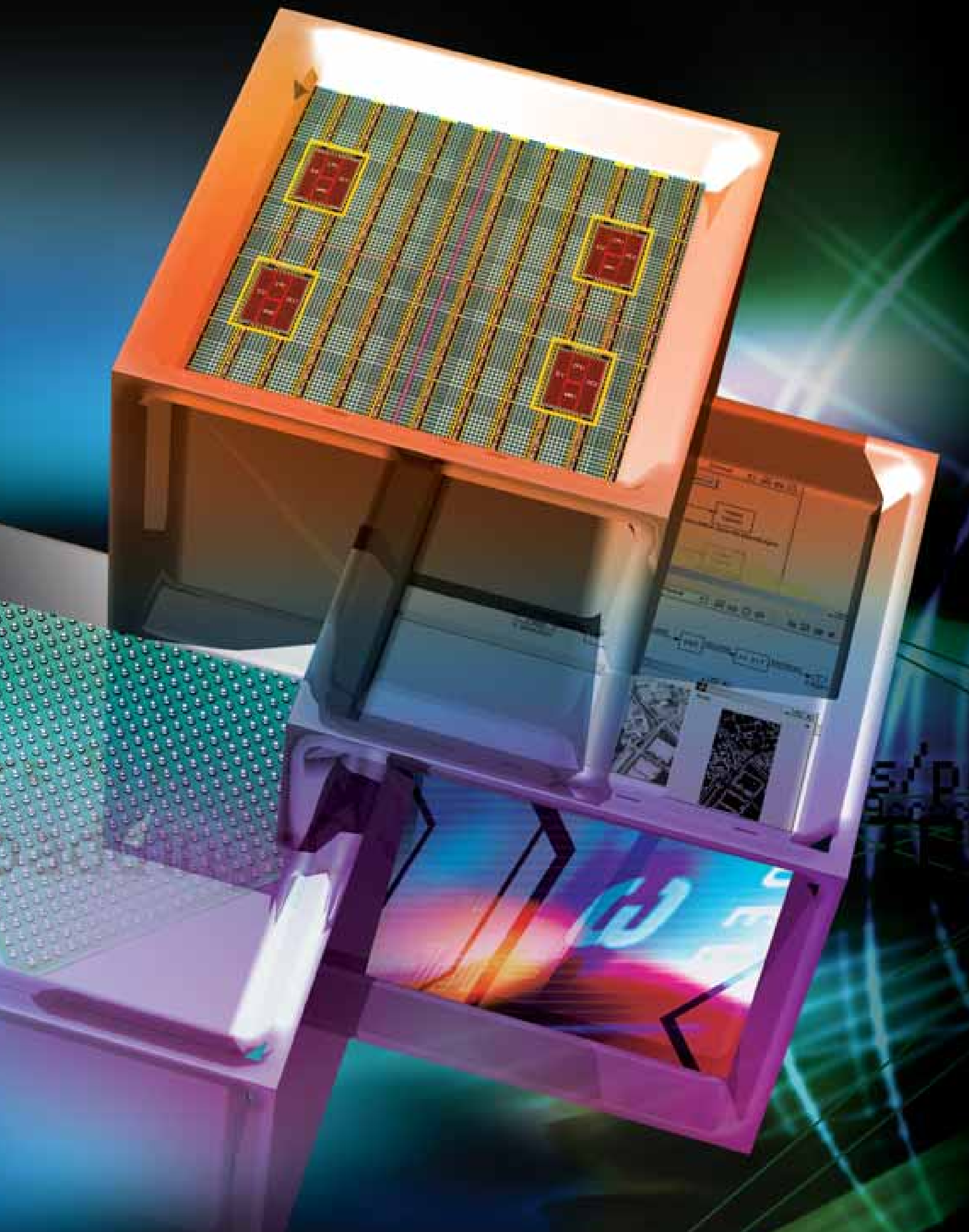
The FIR Compiler is available in both System Generator and Core Generator™ software and is an extremely valuable tool for both DSP algorithm and hardware engineers. It provides rapid generation of difficult-to-implement, high-performance FIR filters and positively impacts design time and risk.

Most importantly, the filters generated take full advantage of the FPGA; consequently, their performance reaches the maximum 400 MHz offered by a Virtex-4 device (-10 slowest speed grade) with extremely efficient resource utilization.

For more information about the FIR Compiler, visit *www.xilinx.com/ipcenter.*

Model-Based Design

A methodology that addresses today's growing challenges of designing embedded systems.



by Ali Behboodian
Applications Engineer
The MathWorks
ali.behboodian@mathworks.com

Embedded systems have transformed technology products – from everyday consumer electronic devices to complex industrial systems. As hardware and memory become less expensive and more powerful, embedded systems will become even more pervasive. At the same time, the designs will become more complex. To meet this demand, engineers must find ways to efficiently develop software and hardware at an even faster rate. A methodology that addresses this is model-based design.

The MathWorks Simulink product family enables you to apply model-based design in a graphical, interactive environment, where you can visualize your system models and subsystem designs using intuitive block diagrams. The models are hierarchical and you can partition the system into functional units. The graphical environment allows you to understand the design and the interactions of the subsystems more easily than text-based models.

In this article, I'll present a model-based design methodology in the context of the design and implementation of the Sobel edge-detection algorithm on an FPGA. Note that you can readily apply these concepts for embedded designs in a wide range of applications in different industries, such as aerospace and defense, automotive, communications, consumer electronics, and medical electronics.

Figure 1 shows the elements of model-based design. The center focus of this design methodology is a model, whose four main elements are:

- Executable specifications
- Design with simulation
- Implementation with code generation
- Continuous test and verification

I'll explain the above four elements and apply them to the design and implementation of the Sobel edge-detection algorithm. For a more comprehensive application of model-based design, see the article, "The Design and Implementation of a GPS Receiver Channel" from Issue 1 of *DSP Magazine* (*www.xilinx.com/publications/magazines/dsp_01/dsp_gps01.htm*).

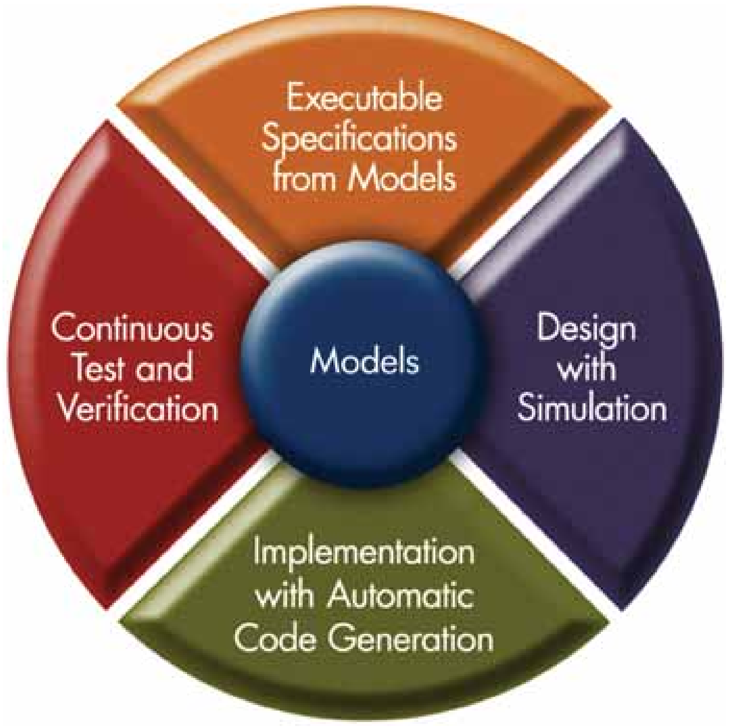

Executable Specifications from Models
Design with Simulation
Implementation with Automatic Code Generation
Continuous Test and Verification
Models

Figure 1 – Elements of model-based design

Executable Specification

As designs become larger and more complicated, it becomes necessary to first describe them at a high level of abstraction. Simulink, together with application-specific blocksets such as the Signal Processing Blockset, the Communications Blockset, and the Video and Image Processing Blockset, provides an excellent graphical environment for a high-level description of embedded algorithms. System engineers usually develop this high-level description.

A high-level Simulink model serves several purposes:

- It enables designers to perform simulations by directly executing the Simulink model
- It is used throughout the development process for testing, verification, and implementation
- It allows developers to identify bugs early on and avoid costly bug discovery towards the end of development
- It eliminates the need for paper-based specification, which is easily prone to misinterpretations, and replaces it with the executable specification
- Each member of a design team can understand and execute the model and can focus further in developing parts of the main model

We call this high-level model the executable specification, or golden reference.

The executable specification for the Sobel edge-detection algorithm is illustrated in Figure 2. The algorithm comprises two 2D filters, each with a 3 x 3 kernel (one filter estimating the edges at the x direction and one in the y direction), two square operations, and a threshold operation. The model is the start for a path that will lead all the way to an FPGA implementation.

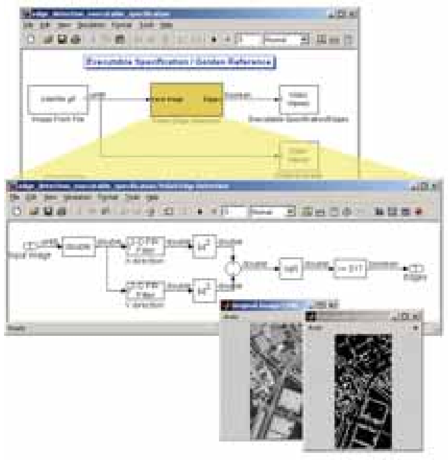


Figure 2 – Executable specification/golden reference for the Sobel edge-detection algorithm

Figure 2 also shows the input image to the algorithm as well as the output of the algorithm. In the Simulink environment, you can also examine and visualize every signal throughout the model.

Note that the input and output images in the executable specification are test vectors for the algorithm. You can use these test vectors throughout the design process to validate your design against the executable specification. Because the entire design is performed in the Simulink environment, there is no need for extra overhead in porting the test vectors into different applications, or creating test harnesses in HDL that are prone to human errors. The test harness used in the executable specification is used throughout the design.

Design with Simulation

When designing the executable specification, the system engineer generally does not keep the implementation details in mind, but rather designs the algorithm to match the behavioral requirements for the system. Once the system engineer submits the executable specification to the development team, the team may need to make modifications to it to fit the design into a real-time embedded system that may have limited resources, such as memory or processing power. These modifications may cause the output of the new design to deviate from the original design. Design engineers should decide if the deviation is acceptable.

In this section, I'll make two modifications to the algorithm to make it suitable for hardware implementation and demonstrate how to continuously verify the design against the executable specification.

Redesigning the Algorithm

Let's say that the developers decide to eliminate the square operations in Figure 2 and replace them with the absolute value operations for more efficient hardware implementation. Generally, such changes in the model are required for hardware implementation and are mostly done by experienced engineers in a design team. Simulink provides an environment where you can redesign an algorithm and validate your designs in a relatively short time. After switching the square operations with the absolute value operations, the final result does not exactly match the output of the executable specification, but the difference is quite small and in this case acceptable.

Fixed-Point Implementation

Because the ultimate goal is to implement the algorithm in an FPGA, for my example I must convert my double-precision design to a fixed-point design. This can be done easily using Simulink. I used the double-precision model I developed to directly develop a fixed-point model without introducing any new blocks.

Simulink allows you to determine the number of bits and scaling for data as well as mathematical operations, and provides a great environment for analyzing the fixed-point operation of a system.

In the fixed-point design, the inputs to the filters are signed 9-bit integers and the outputs of the filters are signed 11-bit integers. The developers can tune the bit width and scaling related to the internal computations of the blocks. This gives huge leverage to the designer to compromise between matching the output of the executable specification while using the least number of bits necessary to save area on the device.

Figure 3 shows the new fixed-point design after replacing the square operations with absolute value operations. In this figure, the new design is compared to the executable specification and the difference is shown both visually as well as numerically. Continuous test and verification is a key part of model-based design and is crucial to the success of a project. Simulink provides an excellent environment for this purpose.

Elaboration of the Design

In my example, the input to the edge-detection algorithm has been a two-dimensional image of 200 x 100 pixels. In a real-time system, the input is most likely not a matrix but a serial stream of data; for example, this serial stream of data can be generated by a charge-coupled device (CCD). Therefore, I need to modify the structure of the design such that the edge-detection algorithm accepts and performs 2D filtering on a serial stream of data.

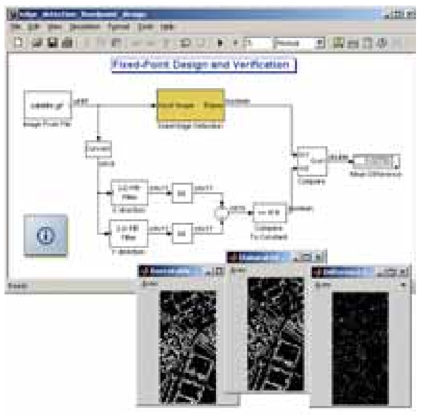


Figure 3 -The image on the left shows the output of the executable specification. The one in the middle shows the output of the new design, including fixed-point design, and replaces the square operations of Figure 2 with absolute value operations. The one on the right depicts the difference between the two designs. The mean difference is 2.793%.

To this extent, I first serialized the input image. Then I performed the 2D filtering on this serial data. I later de-serialized the stream of data to be able to compare the output to the executable specification. This operation is done only for the bottom filter. I also added two delay elements to compensate for the buffering in the serializer block. As expected, the new design is still producing the same exact results as before.

This design also showcases the multirate capability of Simulink. The output rate of the serializer block is 20,000 times higher than the input rate. (Remember that the image size is 200 x 100. Because the image rate is 1 image per second, the sample rate after serialization is 20,000 samples per second.) Figure 4 illustrates the elaborated design.

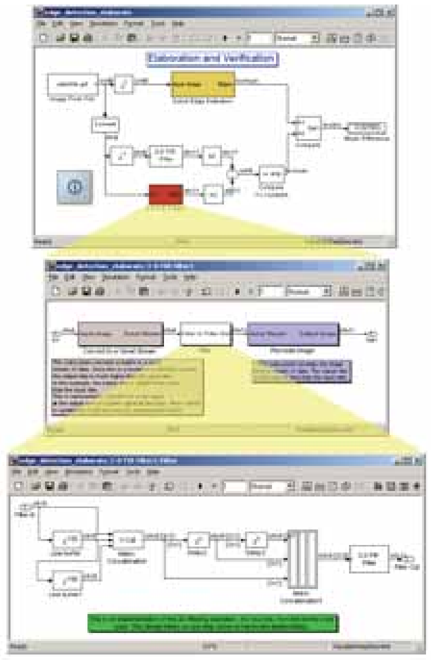


Figure 4 – Elaborated design. A serializer and deserializer are designed and the 2D filter is operating on a stream of 1D data.

Implementation

Now that I've elaborated the design of one of the 2D filters in the Sobel edge-detection algorithm, I can now hand the elaborated design to the hardware designers for HDL implementation. There are two different approaches to consider in this section. The first approach assumes that the hardware designers will hand-code the filter algorithm in VHDL or Verilog. The second approach assumes that the developers will translate the Simulink model from the last section to a Simulink model based on Xilinx System Generator blocks and automatically generate HDL code. In both cases, the developers will verify their design against the executable specification and check the validity of their design in the Simulink environment.

Manual HDL, Co-Simulation, and Verification

The HDL designer on the development team can use the 2D filter design depicted in the bottom window of Figure 4 to write the corresponding VHDL or Verilog code. Once the code is written, the HDL designer can use Link for ModelSim, also from The MathWorks, to simulate the HDL design using ModelSim in the Simulink environment and compare the output of the HDL design to the output of the executable specification. Note that in this process, there is no need to generate an HDL test bench. The Simulink model feeds the input test vector to ModelSim through Link for ModelSim and extracts the data from ModelSim back to the Simulink environment. The HDL designer can

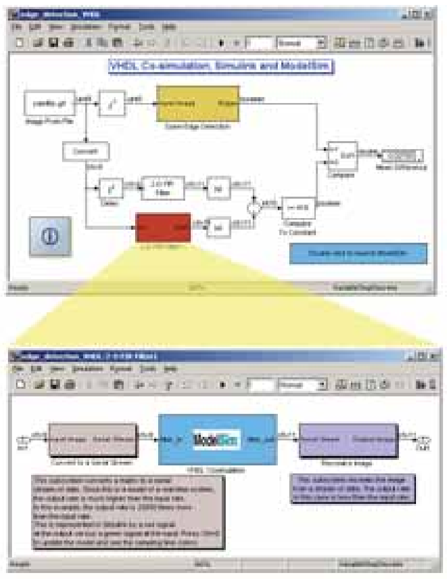


Figure 5 – Co-simulation with ModelSim

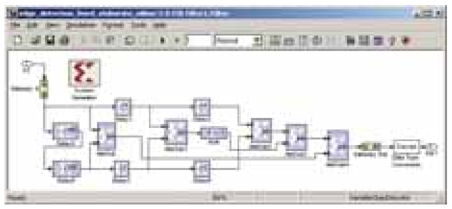


Figure 6 – Modeling of the 2D filter depicted in the bottom window of Figure 4 using Xilinx System Generator

readily verify whether the HDL code runs in accordance with the specifications. The model in Figure 5 co-simulates ModelSim and Simulink and allows you to verify the validity of the VHDL code. As you can see, the mean difference is the same as the previous model.

Automatic HDL Generation, Xilinx System Generator

Using Xilinx System Generator, you can build and debug DSP systems in Simulink using the Xilinx blockset. You can also automatically generate VHDL or Verilog code and run hardware-in-the-loop simulations. Figure 6 illustrates a filter design (using Xilinx System Generator blocks) that is equivalent to the filter depicted in the bottom window of Figure 4. The simulation in the Simulink environment is bit-true and cycle-true. Once you have verified the results of the System Generator design against the executable specification, you can automatically generate synthesizable VHDL or Verilog code for the filter.

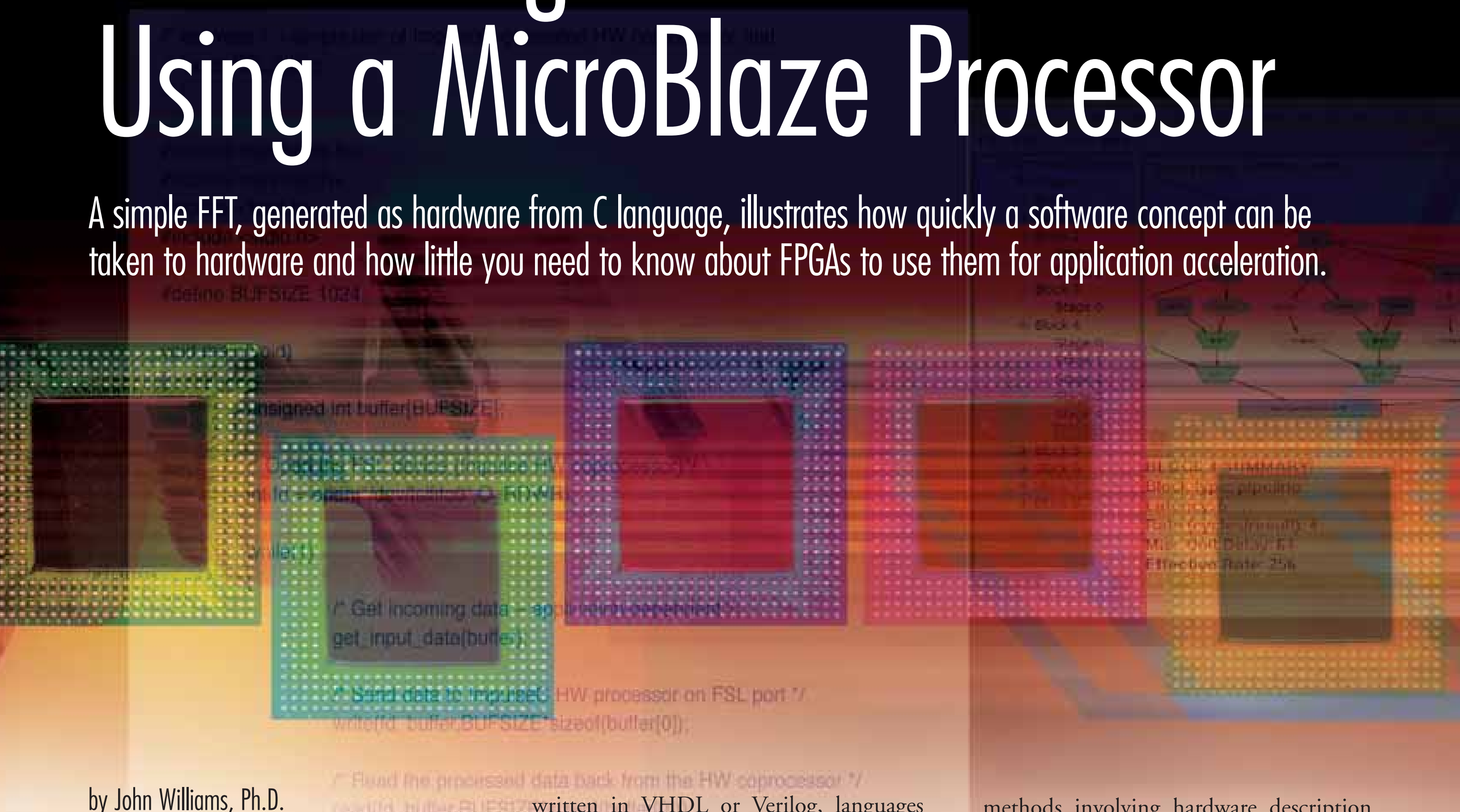
Conclusion

Model-based design helps you create better embedded software and hardware by increasing the accuracy and speed of system development. You can confidently begin integration, test, and deployment of your embedded application knowing that you have identified design errors and met your requirements.

Model-based design provides a proven solution that reduces development time and cost and fosters quality and innovation in the development of embedded systems. For more information, visit *www.mathworks.com/applications/dsp_comm/.*

Accelerating FFTs in Hardware Using a MicroBlaze Processor

A simple FFT, generated as hardware from C language, illustrates how quickly a software concept can be taken to hardware and how little you need to know about FPGAs to use them for application acceleration.



by John Williams, Ph.D.
CEO
PetaLogix
john.williams@petalogix.com

Scott Thibault, Ph.D.
President
Green Mountain Computing Systems, Inc.
thibault@gmvhdl.com

David Pellerin
CTO
Impulse Accelerated Technologies, Inc.
david.pellerin@impulsec.com

FPGAs are compelling platforms for hardware acceleration of embedded systems. These devices, by virtue of their massively parallel structures, provide embedded systems designers with new alternatives for creating high-performance applications.

There are challenges to using FPGAs as software platforms, however. Historically, low-level hardware descriptions must be written in VHDL or Verilog, languages that are not generally part of a software programmer's expertise. Other challenges have included deciding how and when to partition complex applications between hardware and software and how to structure an application to take maximum advantage of hardware parallelism.

Tools providing C compilation and optimization for FPGAs can help solve these problems by providing a new level of programming abstraction. When FPGAs first appeared two decades ago, the primary method of design for these devices was the venerable schematic. FPGA application developers used schematics to assemble low-level components (registers, logic gates, and larger blocks such as counters and adders/subtractors) to create FPGA-based systems. As FPGA devices became more complex and applications targeting them grew larger, schematics were gradually replaced by higher level methods involving hardware description languages like VHDL and Verilog. Now, with ever-higher FPGA gate densities and the proliferation of FPGA embedded processors, there is strong demand for even higher levels of abstraction. C represents that next generation of abstraction, allowing you to access the resources of FPGAs for application acceleration.

For applications that involve embedded processors, a C-to-hardware tool such as Impulse C (Figure 1) can abstract away many of the details of hardware-to-software communication, allowing you to focus on application partitioning without having to worry about the low-level details of the hardware. This also allows you to experiment with alternative software/hardware implementations.

Although such tools can dramatically improve your ability to create FPGA-based applications, for the highest performance you still need to understand

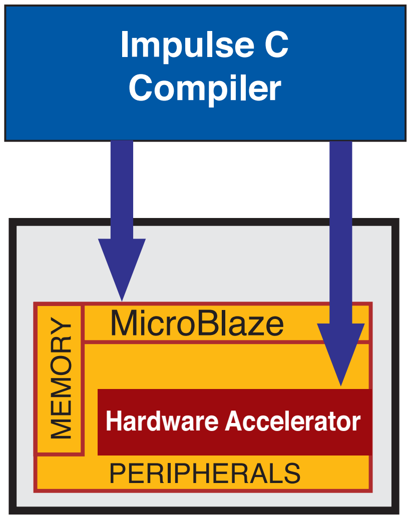
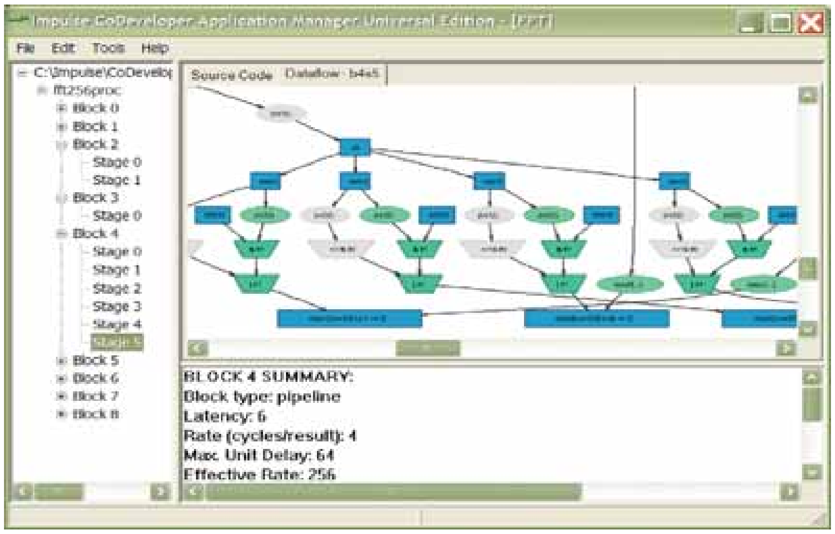


Figure 1 – Impulse C custom hardware accelerators run in the FPGA fabric to accelerate µClinux processor-based applications.

certain aspects of the underlying hardware. In particular, you must understand how partitioning decisions and C coding styles will impact performance, size, and power usage. For example, the acceleration of critical computations and inner-code loops must be balanced against the expense of moving data between hardware and software. Fortunately, modern tools for FPGA compilation provide various types of analysis tools that can help you more clearly understand and respond to these issues.

Practically speaking, the initial results of software-to-hardware compilation from C-language descriptions will not equal the performance of hand-coded VHDL, but the turnaround time to get those first results working may be an order of magnitude better. Performance improvements occur iteratively, through an analysis of how the application is being compiled to the hardware and through the experimentation that C-language programming allows.

Graphical tools (see Figure 2) can help to provide initial estimates of algorithm throughput such as loop latencies and pipeline effective rates. Using such tools, you can interactively change optimization options or iteratively modify and recompile C code to obtain higher performance. Such design iterations may take only a matter of minutes when using C, whereas the same iterations may require hours of even days when using VHDL or Verilog.



```
BLOCK 4 SUMMARY:
Block type: pipeline
Latency: 6
Rate (cycles/result): 4
Max. Unit Delay: 64
Effective Rate: 256
```

Figure 2 – A dataflow graph allows C programmers to analyze the generated hardware and perform explorative optimizations to balance tradeoffs between size and speed. Illustrated in this graph is the final stage of a six-stage pipelined loop. This graph also helps C programmers understand how sequential C statements are parallelized and optimized.

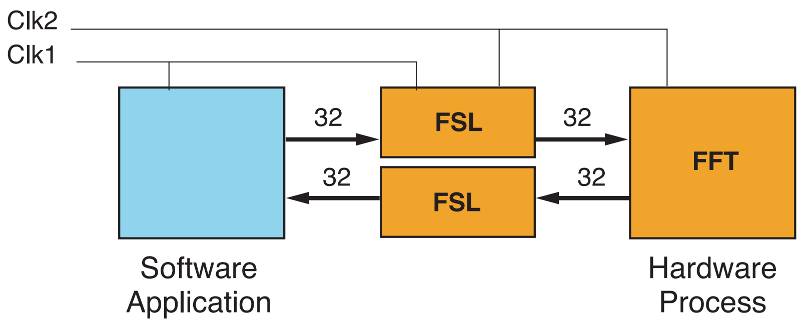


Figure 3 – The FFT includes a 32-bit stream input, a 32-bit stream output, and two clocks, allowing the FFT to be clocked at a different rate than the embedded processor.

Case Study: Accelerating an FFT

The Fast Fourier Transform (FFT) is an example of a DSP function that must accept sample data on its inputs and generate the resulting filtered values on its outputs. Using C-to-hardware tools, you can combine traditional C programming methods with hardware/software partitioning to create an accelerated DSP application. The FFT developer for this example is compatible with any Xilinx® FPGA target, and demonstrates that you can achieve results similar to hand-coded HDL without resorting to low-level programming methods.

Our FFT, illustrated in Figure 3, utilizes a 32-bit stream input, a 32-bit stream output, and two clocks, allowing the FFT to be clocked at a different rate than the embedded processor with which it communicates. The algorithm itself is described using relatively straightforward, hardware-independent C code, with some minor C-level optimizations for increased parallelism and performance.

The FFT is a divide and conquer algorithm that is most easily expressed recursively. Of course, recursion is not possible on the FPGA, so the algorithm must be implemented using iteration instead. In fact, almost all software implementations are written iteratively (using a loop) for efficiency. Once the algorithm has been implemented as a loop, we are able to enable the automatic pipelining capabilities of the Impulse compiler.

Pipelining introduces a potentially high degree of parallelism in the generated

The Impulse compiler generates appropriate FIFO buffers and Fast Simplex Link (FSL) interconnections for the target platform, thereby saving you from the low-level hardware design that would otherwise be needed.

logic, allowing us to achieve the best possible throughput. Our radix-4 FFT algorithm on 256 samples requires approximately 3,000 multiplications and 6,000 additions. Nonetheless, using the pipelining feature of Impulse C, we were able to generate hardware to compute the FFT in just 263 clock cycles.

We then integrated the resulting FFT hardware processing core into an embedded Linux (μClinux) application running on the Xilinx MicroBlaze™ soft-processor core. MicroBlaze μClinux is a free Linux-variant operating system ported at the University of Queensland and commercially supported by PetaLogix.

The software side of the application running under the control of the operating system interacts with the FFT through data streams to send and receive data, and to initialize the hardware process. The streams themselves are defined using abstract communication methods provided in the Impulse C libraries. These stream communication functions include functions for opening and closing data streams and reading and writing those streams. Other functions allow the size (width and depth) of the streams to be defined.

By using these functions on both the software and hardware sides of the application, it is easy to create applications in which hardware/software communication is abstracted through a software API. The Impulse compiler generates appropriate FIFO buffers and Fast Simplex Link (FSL) interconnections for the target platform, thereby saving you from the low-level hardware design that would otherwise be needed.

Embedded Linux Integration

The default Impulse C tool flow targets a standalone MicroBlaze software system. In some applications, however, a fully featured operating system like μClinux is required. Advantages of embedded Linux include a familiar development environment (applications may be prototyped on desktop Linux machines), a feature-rich set of networking and file storage capabilities, a tremendous array of existing software, and no per-unit distribution royalties.

The μClinux (pronounced "you-see-Linux") operating system is a port of the open-source Linux version 2.4. The μClinux kernel is a compact operating system appropriate for a wide variety of 32-bit, non-memory management unit (MMU) processor cores. μClinux supports a huge range of microprocessor architectures, including the Xilinx MicroBlaze processor, and is deployed in millions of consumer and industrial embedded systems worldwide.

Integrating an Impulse C hardware core into μClinux is straightforward; the Impulse tools include support for μClinux and can generate the required hardware/software interfaces automatically, as well as generate a makefile and associated software libraries to implement the streaming and other functions mentioned previously. Using the Xilinx FSL hardware interface, combined with a freely available generic FSL device

```
/* example 1 – simple use of ImpulseC-generated HW coprocessor and
 * Linux FSL driver
 * /

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

#define BUFSIZE 1024

void main(void)
{
        unsigned int buffer[BUFSIZE];

        /* Open the FSL device (Impulse HW coprocessor)*/
        int fd = open("/dev/fslfifo0",O_RDWR);

        while(1)
        {
                /* Get incoming data – application dependent*/
                get_input_data(buffer);

                /* Send data to ImpulseC HW processor on FSL port */
                write(fd, buffer,BUFSIZE*sizeof(buffer[0]);

                /* Read the processed data back from the HW coprocessor */
                read(fd, buffer,BUFSIZE*sizeof(buffer[0]));

                /* Do something with the data – application dependent */
                send_output_data(buffer);
        }
}
```

Figure 4 – Simple communication between μClinux applications and ImpulseC hardware using the generic FSL FIFO device driver

driver in the MicroBlaze μClinux kernel, makes the process of connecting the software application to the Impulse C hardware accelerator relatively easy.

The generic FSL device driver maps the FSL ports onto regular Linux device nodes, named /dev/fslfifo0 through to fslfifo7, with the numbers corresponding to the physical FSL channel ID.

The FIFO semantics of the FSL channels map naturally onto the standard Linux software FIFO model, and to the streaming programming model of Impulse C. An FSL port may be opened, read, or written to, just like a normal file. Here is a simple example that shows how easily a software application can interface to a hardware co-processing core through the FSL interconnect (Figure 4).

```
/* example 2 – Overlapping communication and computation to exploit
 * parallelism
 * /

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

#define BUFSIZE 1024

void main(void)
{
        unsigned int buffer1[BUFSIZE],buffer2[BUFSIZE];
        unsigned int *buf1=buffer1;
        unsigned int *buf2=buffer2;
        unsigned int *tmp;

        /* Open the FSL device (Impulse HW coprocessor)*/
        int fd = open("/dev/fslfifo0",O_RDWR);

        /* Get incoming data – application dependent*/
        get_input_data(buf1);

        while(1)
        {

                /* Send data to ImpulseC HW processor on FSL port */
                write(fd, buf1,BUFSIZE*sizeof(buffer[0]);

                /* Read more data while HW coprocessor is working */
                get_input_data(buf2);

                /* Read the processed data back from the HW processor */
                read(fd, buf1,BUFSIZE*sizeof(buffer[0]));

                /* Do something with the data – application dependent */
                send_output_data(buf1);

                /* Swap buffers */
                tmp=buf1;
                buf1=buf2;
                buf2=tmp;
        }
}
```

Figure 5 – Overlapping communication and computation for greater system throughput

You can easily modify this basic structure to further exploit the parallelism available. One easy performance improvement is to overlap I/O and computation, using a double-buffering approach (Figure 5).

From these basic building blocks, you are ready to tune and optimize your application. For example, it becomes a simple matter to instantiate a second FFT core in the system, connect it to the MicroBlaze processor, and integrate it into an embedded Linux application.

An interesting benefit of the embedded Linux integration approach is that it allows developers to take advantage of all that Linux has to offer. For example, with the FFT core mapped onto FSL channel 0, we can use MicroBlaze Linux shell commands to drive and test the core:

```
$ cat input.dat > /dev/fslfifo0 &; cat /dev/fslfifo0
> output.dat;
```

Linux symbolic links permit us to alias the device names onto something more user-friendly:

```
$ ln -s /dev/fslfifo0 fft_core

$ cat input.dat > fft_core &; cat fft_core >
output.dat;
```

Conclusion

Although our example demonstrates how you can accelerate a single embedded application using one FSL-attached accelerator, Xilinx Platform Studio tools also permit multiple MicroBlaze CPUs to be instantiated in the same system, on the same FPGA. By connecting these CPUs with FSL channels and employing the generic FSL device driver architecture, it becomes possible to create a small-scale, single-chip multiprocessor system with fast inter-processor communication. In such a system, each CPU may have one or more hardware acceleration modules (generated using Impulse C), providing a balanced and scalable multi-processor hybrid architecture. The result is, in essence, a single-chip, hardware-accelerated cluster computer.

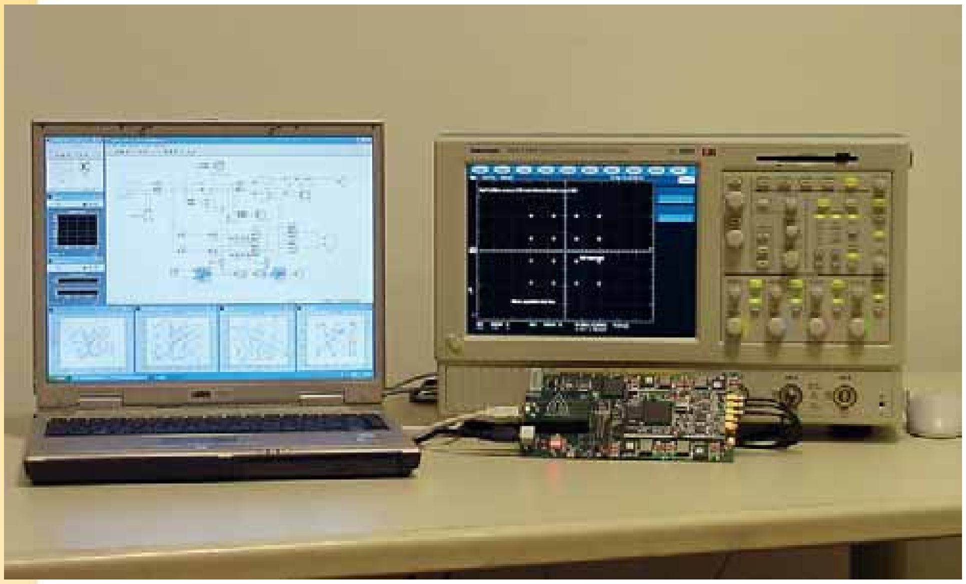
To discover what reconfigurable cluster-on-chip technology combined with C-to-hardware compilation can do for your application, visit *www.petalogix.com* and *www.impulsec.com*.

Virtex™-4 SX 35 XtremeDSP™ Development Kit for Digital Communication Applications

Creating extremely high-performance digital communications signal-processing solutions can present significant challenges in both design complexity and time to market. The XtremeDSP™ Development Platform from Xilinx provides a complete development solution, so your designs will be faster, easier, and earlier to market.

Virtex-4 SX FPGAs feature up to 512, XtremeDSP slices, each capable of running at 500 MHz. This performance makes them the ideal co-processors for your DSP processors and the best way to increase your system performance by several orders of magnitude.

The XtremeDSP Development Platform — together with the Xilinx System Generator for DSP software and Xilinx DSP IP algorithms — provide the ideal development environment for developing Virtex-II Pro based signal-processing designs.

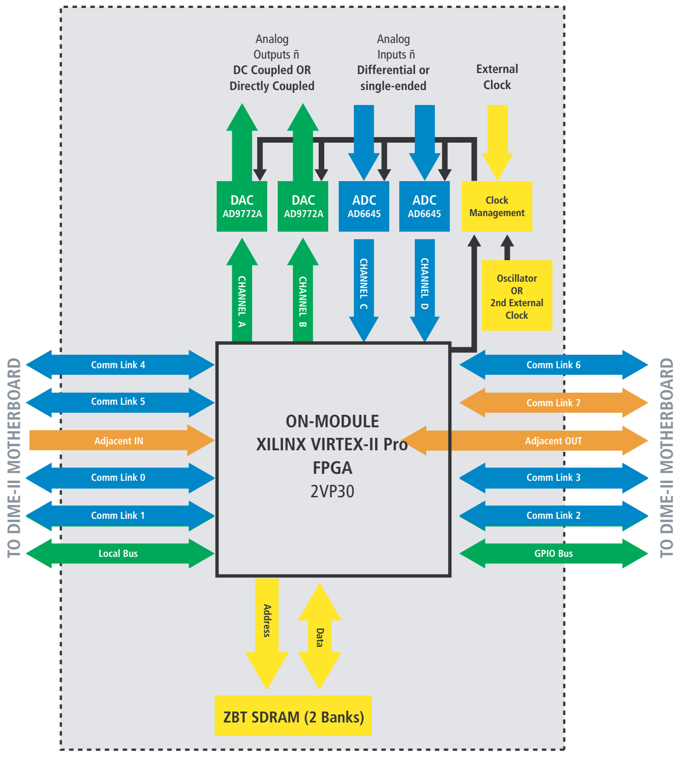

Hardware co-simulation with the XtremeDSP Platform and Xilinx System Generator for DSP

Your Complete Devleopment Platform

Developed with Nallatech, the Virtex-4 SX XtremeDSP Development Platform offers everything you need to create high-performance signal-processing designs more quickly and efficiently.

- **Exceptional Performance –** The dual-channel, high-performance ADCs and DACs, coupled with a user-programmable Virtex-4 SX-10 FPGA, make this platform ideal for implementing high-performance digital communication systems such as Software Defined Radios. The SX 35 FPGA features over 55,000 logic cells, 192 XtremeDSP slices.
- **Ease of Use –** Combining the Xilinx System Generator for DSP software tool and the XtremeDSP Development Kit provides an easy transition to using FPGAs for high-performance signal processing—from algorithm concept to hardware verification. The System Generator tool interfaces with MATLAB®/Simulink® and enables you to perform hardware co-simulation on the XtremeDSP Development Platform via PCI or JTAG. This provides simulation acceleration by an order of magnitude and allows you to debug and verify the design on the FPGA.
- **Comprehensive Support –** Reduce your time to knowledge with the Xilinx DSP Design Flow and DSP Implementation Techniques courses. You can also take advantage of senior DSP support engineer expertise on the Xilinx Hotline.

Finish Faster with Xilinx DSP Design Solutions



Dime II module functional diagram

Hardware Platform Specifications

- XtremeDSP development board consisting of a motherboard (“BenONE-Kit Motherboard”) populated with a daughter card (“BenADDA DIME-II Module”).

BenONE-Kit Motherboard

- Supports the supplied BenADDA DIME-II module only
- Spartan-II™ FPGA for 3.3V/5V PCI or USB interface
- Host interfacing via 3.3V/5V PCI 32-bit/33-MHz or USB v1.1 interfaces
- Status LEDs
- JTAG configuration headers
- User 0.1-inch pitch pin headers connected directly to user programmable FPGA I/O

BenADDA DIME-II module

- Virtex-4 SX 35 user FPGA: XC4VSX35
- Two independent ADC channels: AD6645 ADC (14 bits up to 105 MSPS)
- Two independent DAC channels: AD9772 DAC (14 bits up to 160MSPS)
- Support for external clock, on-board oscillator, and programmable clocks
- Two banks of ZBT-SRAM (133 MHz, 512 Kx32 bits per bank)
- Multiple clocking options: internal and external
- Status LEDs

Also included with the XtremeDSP Platform

- External power supply (US Mains cable with separate UK, European or Australian Mains adapters)
- Wide ranging input (90 - 264Vac), multiple output, power supply, generating +5 Volts @ 5A, and +12 Volts @ 2A, -12 Volts @ 800mA
- USB v1.1-compatible cable, two meters long
- Five MCX-to-BNC cables for connecting to the ADC/DAC and external clock connectors
- PCI back-plate and two screws
- 2x BNC jack-to-jack adapters for use in loop-back configurations
- Large carrying case

XtremeDSP Installation Pack

- Nallatech FUSE Software CD — Enables control and configuration of FPGAs and provides tools to transfer data between the Kit and a host PC via a GUI or a C-based API

Applications

This multi-purpose board can be used for many digital communications applications including:

- Narrow-band systems (QAM demodulation, carrier timing recovery, channel coding)
- Spread-spectrum systems (e.g. chip rate processing, RACH, path profiling, TCC)
- Multi-carrier systems (e.g. OFDM, MIMO, TCC)
- And many more.

Take the Next Step

Purchase your XtremeDSP Platform at **www.xilinx.com/store**. For more information, visit **www.xilinx.com/dsp**. To learn more about the complete Nallatech platform offering, visit **www.nallatech.com**.
Price: $2,495

NALLATECH The High Performance FPGA Solutions Company

Corporate Headquarters
Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Tel: (408) 559-7778
Fax: (408) 559-7114
Web: www.xilinx.com

European Headquarters
Xilinx
Citywest Business Campus
Saggart,
Co. Dublin
Ireland
Tel: +353-1-464-0311
Fax: +353-1-464-0324
Web: www.xilinx.com

Japan
Xilinx, K.K.
Shinjuku Square Tower 18F
6-22-1 Nishi-Shinjuku
Shinjuku-ku, Tokyo
163-1118, Japan
Tel: 81-3-5321-7711
Fax: 81-3-5321-7765
Web: www.xilinx.co.jp

Asia Pacific
Xilinx Asia Pacific Pte. Ltd.
No. 3 Changi Business Park Vista, #04-01
Singapore 486051
Tel: (65) 6544-8999
Fax: (65) 6789-8886
RCB no: 20-0312557-M
Web: www.xilinx.com

XILINX® The Programmable Logic Company℠

Distributed By:

FORTUNE 2005 100 BEST COMPANIES TO WORK FOR

DSP Design Flow

DSP10000-8-ILT (v1.0)

Course Specification

Course Description

The DSP Design Flow course provides the advanced tools and expertise you need to develop advanced, low-cost DSP designs. This intermediate course in implementing DSP functions focuses on learning how to use System Generator for DSP, design implementation tools, HDL co-simulation, and hardware-in-the-loop verification. Through hands-on exercises, you will implement a design from algorithm concept to hardware verification by using Xilinx FPGA capabilities.

Level – Intermediate
Course Duration – 3 days
Price – $1500 USD or 15 Training Credits
Course Part Number – DSP10000-8-ILT
Who Should Attend? – System engineers/designers, logic designers, and experienced hardware engineers who are implementing DSP algorithms using MathWorks MATLAB and Simulink and using Xilinx System Generator for DSP
Prerequisites

- Fundamentals of MATLAB/Simulink and Xilinx FPGAs
- Basics of digital signal processing theory for functions, such as FIR (Finite Impulse Response) filters, oscillators and mixers, and FFT (Fast Fourier Transform) algorithms

Software Tools

- ISE™ 8.1i
- System Generator for DSP 8.1
- EDK 8.1
- ISIM Simulator 8.1
- ChipScope™ 8.1
- Mentor Graphics ModelSim PE 6.0c
- MATLAB with Simulink R14 SP1

After completing this comprehensive training, you will have the necessary skills to:

- Describe the different design flows for implementing DSP functions, with a large focus on System Generator
- Identify Xilinx FPGA capabilities and know how to implement a design from algorithm concept to hardware simulation
- Implement a design from start to finish by using System Generator
- Perform hardware-in-the-loop and HDL co-simulations and improve productivity
- Integrate the ChipScope Pro block in a design and analyze the design
- Develop a hardware co-simulation model using System Generator Board Description Builder
- Integrate a System Generator design as a peripheral in a MicroBlaze™ processor-based system
- Utilize timing analyzer block to improve design performance

Course Outline

Note: Target architectures include Virtex™-4, Virtex-II Pro, and Spartan™-3E FPGAs.

Day 1

- Introduction
- DSP Design Flows in FPGAs
- **Lab 1:** Creating a 12 x 8 MAC Using the Xilinx System Generator
- Digital Filtering
- **Lab 2:** Designing a FIR Filter
- HDL Co-Simulation
- **Lab 3:** MAC FIR Filter Verification Using Simultaneous Co-Simulations

Day 2

- Looking Under the Hood
- **Lab 4:** Looking Under the Hood
- Controlling the System
- **Lab 5:** Controlling the System
- Multirate Systems
- **Lab 6:** Designing a MAC-Based FIR Using the DSP48 Slice

Day 3

- Advanced Features
- **Lab 7:** Integrating the ChipScope Pro Analyzer
- **Lab 8:** A System Generator Design as an XPS Peripheral
- **Lab 9:** Multiple Clock Domains Design Using Shared Memories
- **Lab 10:** Improving Design Performance Using Timing Analyzer
- **Lab 11.** Designing Using the PicoBlaze™ MicroController
- **Lab 12.** Creating Parametric Designs

Lab Descriptions

This lab-intensive class gives you hands-on experience by using System Generator for DSP to visualize, simulate, verify, and implement DSP algorithms in Xilinx FPGAs. The labs start at a descriptive level and build on each other. You should expect each successive lesson's challenges to increase. In addition, the labs included in the Advanced Features module provide you experience with other tools such as the ChipScope Pro analyzer and the Embedded Development Kit. System Generator for DSP 8.1 features are identified, including hardware and software co-simulation verification.

Register Today

Xilinx delivers public and private courses in locations throughout the world. Please contact Xilinx Education Services for more information, to view schedules, or to register online.

Visit **www.xilinx.com/education**, and click on the region where you want to attend a course.

North America, send your inquiries to registrar@xilinx.com, or contact the registrar at 877-XLX-CLAS (877-959-2527). To register online, search by **Keyword** "DSP" in the Training Catalog at https://xilinx.onsaba.net/xilinx.

Europe, send your inquiries to eurotraining@xilinx.com, call +44-870-7350-548, or send a fax to +44-870-7350-620.

Asia Pacific, contact our training providers at: www.xilinx.com/support/training/asia-learning-catalog.htm, send your inquiries to education_ap@xilinx.com, or call: +852-2424-5200.

Japan, see the Japanese training schedule at: www.xilinx.co.jp/support/training/japan-learning-catalog.htm, send your inquiries to education_kk@xilinx.com, or call: +81-3-5321-7772.

You must have your tuition payment information available when you enroll. We accept credit cards (Visa, MasterCard, or American Express) as well as purchase orders and training credits.

DSP Implementation Techniques for Xilinx FPGAs

DSP20000-7-ILT (v1.0)

Course Specification

Course Description

This course shows you how to take advantage of the features available in the Xilinx FPGA architecture, including the Virtex™-4 FPGA, and describes how DSP algorithms can be implemented efficiently. The techniques also demonstrate which decisions at the system level have the greatest impact on the implementation process and product costs.

Level – Advanced
Course Duration – 3 days
Price – $1800 USD or 18 Training Credits
Course Part Number – DSP20000-7-ILT
Who Should Attend? – Engineers and designers who have an interest in developing products that use digital signal processing
Prerequisites
A fundamental understanding of digital signal processing theory, including an understanding of the following principles:

- Sample rates
- Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters
- Oscillators and mixers
- Fast Fourier Transform (FFT) algorithm

After completing this comprehensive training, you will have the necessary skills to:

- Describe how DSP algorithms can be implemented efficiently by using Xilinx FPGA technology
- Identify the capabilities and features of the various Xilinx FPGA families to implement efficient DSP algorithms
- Establish methods for the accurate estimation of silicon area consumption and cost
- Evaluate which algorithms are best suited for FPGA implementation and identify which algorithms are less desirable
- Assess how system-level decisions impact hardware implementation and how hardware implementation can enhance results at the system level

Course Outline

Day 1

- On the Same Wavelength
  - Basic terminology and acronyms used in DSP design
  - Sample rates and bit widths used in DSP applications
  - DSP building blocks and processing requirements
- Some Bits About Numbers
  - Numbering formats, range, and precision
  - Mathematical operations using a variety of formats
- Tuning the Receiver
  - Structure and Resources of Xilinx Devices
  - Estimating DSP building block sizes

Day 2

- Tuning the Receiver (continued)
  - Implementing the multiplication function
  - Bit-width impact on system-level decisions
- Memories are Made of This
  - Block versus distributed memory
  - SRL16E and the delay function
  - Memory aspect ratios and their manipulation
- Selective Filters
  - FIR filter specifications and implementation
  - Selecting a technique for a given specification
  - Effects of halfband and interpolated filters

Day 3

- One Filter Does Not Make a System
  - Options to be considered with multiple channels
  - Interpolation and decimation
  - Rate changing and its effect on FIR filter choice
  - Filtering algorithms that exploit device architecture
  - Importance of connectivity versus isolated functions
- Do Not Block the Datapath
  - Numeric controlled oscillators and mixers
  - Strategies for FFT implementation
  - Achieving bandwidth requirements of the FFT
  - Using the FPGA as an efficient co-processor

Course Exercises

- MAC Rates and Memory Requirements
- Constructing a 128-Tap FIR Filter
- Fractional Number Formats
- Twos Complement Arithmetic
- Summation by Addition Tree
- Summation by Addition Chain
- Full Adder: How Many Slices?
- Summation Structure Sizes
- Serial Summation Structure
- 8-Bit by 12-Bit Multiplier
- KCM Multipliers
- Distributed RAM for FIFO
- Size Estimates for Delay Structures
- Using the SRL16E as a FIFO
- Creating Larger RAM Structures
- Selecting a MAC FIR Technique
- Parallel FIR Filter Size
- Symmetry, Interpolation, and Phases
- Decimation Filter
- “fs/4” Mixing and Decimation
- Designing a Numeric Controlled Oscillator (NCO)
- FFT: Benchmarks and Transform Time
- Collection Time = Processing Time
- 128-Point FFT in 1.28 μs

Register Today

Xilinx delivers public and private courses in locations throughout the world. Please contact Xilinx Education Services for more information, to view schedules, or to register online.

Visit **www.xilinx.com/education**, and click on the region where you want to attend a course.

North America, send your inquiries to registrar@xilinx.com, or contact the registrar at 877-XLX-CLAS (877-959-2527). To register online, search by **Keyword** “DSP” in the Training Catalog at https://xilinx.onsaba.net/xilinx.

Europe, send your inquiries to eurotraining@xilinx.com, call +44-870-7350-548, or send a fax to +44-870-7350-620.

Asia Pacific, contact our training providers at: www.xilinx.com/support/training/asia-learning-catalog.htm, send your inquiries to education_ap@xilinx.com, or call: +852-2424-5200.

Japan, see the Japanese training schedule at: www.xilinx.co.jp/support/training/japan-learning-catalog.htm, send your inquiries to education_kk@xilinx.com, or call: +81-3-5321-7772.

You must have your tuition payment information available when you enroll. We accept credit cards (Visa, MasterCard, or American Express) as well as purchase orders and training credits.

XILINX®

Designing with Multi-Gigabit Serial I/O

RIO22000-8-ILT (v2.0)

Course Specification

Course Description

Learn how to employ RocketIO™ MGT serial transceivers in your Virtex™-II Pro design. Understand and utilize the features of the RocketIO transceiver blocks, such as CRC, 8b/10b encoding, channel bonding, clock correction, and comma detection. Additional highlighted topics include debugging techniques, use of the Architecture Wizard, synthesis and implementation considerations, and standards compliance. This course balances lecture modules and practical hands-on labs.

Level – Intermediate
Course Duration – 2 days
Price – $1000 USD or 10 Training Credits
Course Part Number – RIO22000-8-ILT
Who Should Attend? – FPGA designers and logic designers
Prerequisites

- Verilog or VHDL experience (or the *Introduction to Verilog* or the *Introduction to VHDL* course)
- Synthesis and simulation experience
- FPGA design experience or the *Fundamentals of FPGA Design* course
- Knowledge of high-speed serial I/O protocols and standards (SONET, Gigabit Ethernet, InfiniBand) is a plus

Software Tools

- ISE 8.1i
- ModelSim PE 6.0

After completing this comprehensive training, you will have the necessary skills to:

- Effectively use all of the advanced RocketIO features, such as CRC, channel bonding, clock correction, comma detection, 8b/10b encoding/decoding, programmable termination, and pre-emphasis
- Utilize the ports and attributes of RocketIO transceivers that control the RocketIO features
- Use the Architecture Wizard to instantiate RocketIO primitives in your design
- Achieve compatibility with high-speed I/O standards by using RocketIO transceivers

Course Outline

Day 1

- Introduction
- Clocking and Resets
- 8b/10b Encoder and Decoder Details
- **Lab 1:** 8b/10b Disparity and Bypass Lab
- Commas and Deserializer Alignment Details
- **Lab 2:** Commas and K-Characters Lab
- Cyclical Redundancy Check Details
- **Lab 3:** Cyclical Redundancy Check Lab
- Clock Correction Details
- **Lab 4:** Clock Correction Lab

Day 2

- Channel Bonding Details
- **Lab 5:** Channel Bonding Lab
- Architecture Wizard Overview
- Implementing a RocketIO Design
- **Lab 6:** Synthesis and Implementation Lab
- IP Overview: Aurora Reference Design
- **Lab 7:** Aurora Protocol Engine Lab
- Common Serial I/O Standards Compliance
- Physical Media Attachment Overview

Lab Descriptions

- **Lab 1:** 8b/10b Disparity and Bypass Lab – Utilize the 8b/10b encoder/decoder and manipulate running disparity. Learn how to bypass the 8b/10b encoder/decoder
- **Lab 2:** Commas and K-Characters Lab – Use programmable comma detection to align a serial data stream
- **Lab 3:** CRC Lab – Modify a design to use the CRC feature for both the user mode and the Fiber Channel mode of CRC
- **Lab 4:** Clock Correction Lab – Utilize the clock correction logic to compensate for frequency differences on the TX and RX side of a link
- **Lab 5:** Channel Bonding Lab – Modify a design to use two transceivers bonded together to form one virtual channel
- **Lab 6:** Synthesis and Implementation Lab – Use the Architecture Wizard to instantiate RocketIO primitives, synthesize a design, and implement the design.
- **Lab 7:** Aurora Protocol Engine Lab – Use the Aurora reference design to send and receive data

Register Today

Xilinx delivers public and private courses in locations throughout the world. Please contact Xilinx Education Services for more information, to view schedules, or to register online.

Visit **www.xilinx.com/education**, and click on the region where you want to attend a course.

North America, send your inquiries to registrar@xilinx.com, or contact the registrar at 877-XLX-CLAS (877-959-2527). To register online, search by **Keyword** "High-Speed" in the Training Catalog at https://xilinx.onsaba.net/xilinx.

Europe, send your inquiries to eurotraining@xilinx.com, call +44-870-7350-548 or send a fax to +44-870-7350-620.

Asia Pacific, contact our training providers at: www.xilinx.com/support/training/asia-learning-catalog.htm, send your inquiries to education_ap@xilinx.com, or call: +852-2424-5200.

Japan, see the Japanese training schedule at: www.xilinx.co.jp/support/training/japan-learning-catalog.htm, send your inquiries to education_kk@xilinx.com, or call: +81-3-5321-7772.

You must have your tuition payment information available when you enroll. We accept credit cards (Visa, MasterCard, or American Express) as well as purchase orders and training credits.

FREE on-line training with Demos On Demand

A series of compelling, highly technical DSP product demonstrations, presented by Xilinx DSP experts, is now available on-line. These comprehensive DSP videos provide excellent, step-by-step tutorials and quick refreshers on a wide array of key topics. The videos are segmented into short chapters to respect your time and make for easy viewing.

XtremeDSP

Ready for viewing, anytime you are

- *DSP Video Starter Kit*
- *DSP Video Co-processing Kit*
- *System Generator for DSP*
- *Algorithm to Hardware in 60 Minutes*
- *XtremeDSP Slice*
- *FPGAs for Signal Processing*
- *Designing QAM Demodulators*
- *Spectrum Channelization*

A complete on-line archive is easily accessible at your fingertips. Also, a *free* DVD containing all the all of the demonstrations is available at **www.xilinx.com/dod**. Order yours today!

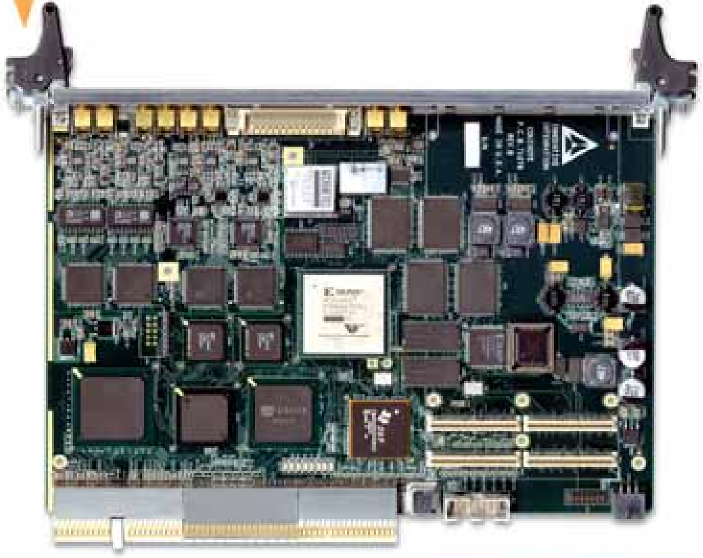
XILINX®

www.xilinx.com/dod

Earth Friendly

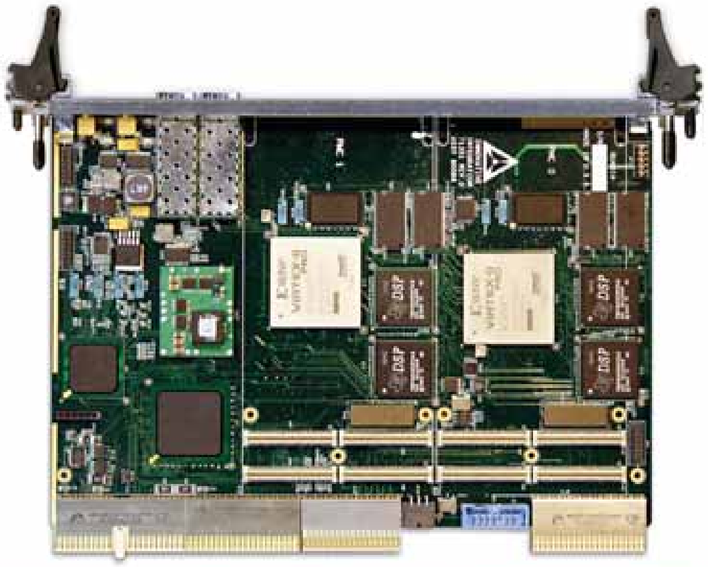
Pb-free devices available now

DSP-Based RF Solutions!



Quixote

Quixote
- 1 GHz TMS320C6416 DSP
- 2 to 6 MGATE Virtex-II FPGA
- 32 MB SDRAM, 8 MB ZBT SBSRAM
- A/D: 105 MSPS 14-bit
- 64/32 bit CompactPCI, 66 MHz, 5V/3.3V
- Complex trigger modes w/HW event log
- PMC Site with Jn4 to FPGA DIO
- PICMG2.17 StarFabric Compliant
- MatLab & Framework Logic Development



Quadia

Quadia
- 1 GHz TMS320C6416 DSP (x4)
- 64MB SDRAM per Processor
- Flexible communication mesh
- 64-bit/66MHz CompactPCI
- 2 100MHz PMC/XMC Sites - 800MB/s data rate
- External Data Port, up to 2Gbit/s
- Dual Xilinx VP40 FPGAs
- Two 2MB private SBSRAM per FPGA
- Up to 2 128MB private DDR SDRAM per FPGA
- Up to 512MB Global DDR SDRAM
- MatLab & Framework Logic Development

Complete DSP-Based PMC/XMC RF Solutions

SIO
High-Speed Tx/Rx Serial I/O
Dual 1Gbit/sec Full Duplex

DR
16 Channel Digital Receiver
Four A/D 125MHz

UWB
Ultra-Wide Digital Receiver
Dual A/D 210MHz

TX
Digital Transmitter
Four 1 GSPS DAC

Free Instant
On-Line Pricing!

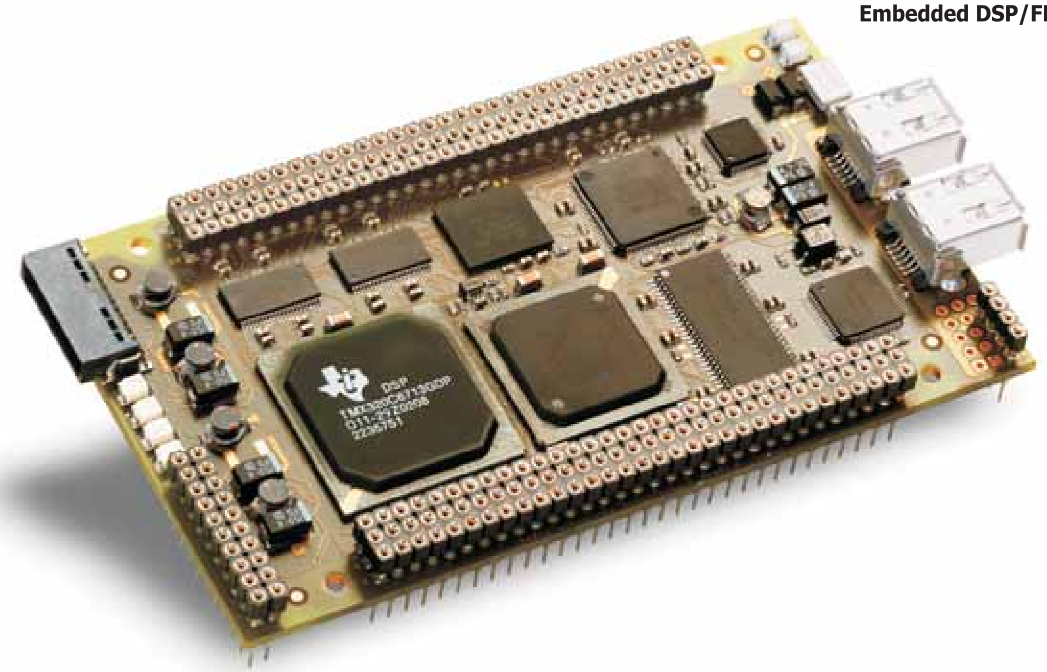
805.578.4260 phone
www.innovative-dsp.com

Innovative Integration
... real time solutions!

TRAQUAIR
www.traquair.com

The Compact DSP & FPGA Solution

micro-line C6713Compact
Embedded DSP/FPGA board



Virtex-II FPGA
250k, 500k or 1MGates

TMS320C6713 DSP
Up to 2400MIPS/1800 MFLOPS

IEEE 1394 FireWire
400MBit/sec Communications
IIDC DCAM Video Framecaputre

Optional Data Storge
HD or Compact FLASH
FAT32 Filesystem Support

Optional Ethernet
10/100BaseT,
TCP/IP, UDP, ICMP, IGMP, Telnet, HTTP, SMTP, POP3, FTP ,
Embedded Web Server

Optional Analog I/O
12/14/16-bit multi-channel A/D/A
via FPGA I/O Pins or DSP EMIF

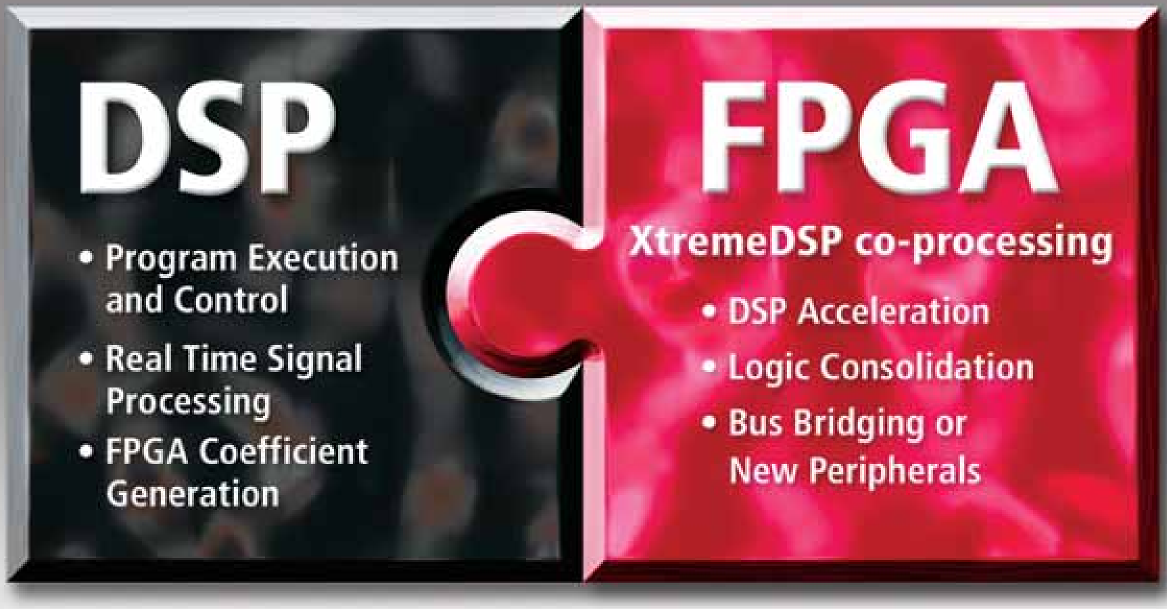
The micro-line C6713Compact is a high performance single board DSP/FPGA solution, offering exceptional capabilities and flexibility.

Measuring only 67 x 120mm, it combines a Xilinx Virtex-II FPGA with Texas Instruments' most powerful floating point DSP processor, the TMS320C6713, as well as with up to 64MBytes of SDRAM, 8MBytes of FLASH ROM, FireWire, and optional Ethernet communications and analog I/O.

It is suitable for stand-alone operation or as a mezzanine daughter card, and has extensive digital I/O capabilities for easy integration with the on-board FPGA, DSP and FireWire resources.

Traquair Data Systems, Inc. Tel: 607-266-6000 Email: sales@traquair.com Web: www.traquair.com

Turbocharge your DSP performance


DSP
• Program Execution and Control
• Real Time Signal Processing
• FPGA Coefficient Generation
FPGA
XtremeDSP co-processing
• DSP Acceleration
• Logic Consolidation
• Bus Bridging or New Peripherals

Achieve high-definition, higher frame rates or multiple video streams

When complimenting a TI DSP, Xilinx XtremeDSP co-processing offers the performance, versatility, and economy for today's high-end video and imaging applications. Whether it's high-definition, motion estimation, video scaling, or any number of compute intensive functions, a Xilinx Virtex-4 or Spartan-3/3E FPGA can boost your DSP performance. XtremeDSP co-processing delivers higher resolution, higher frame rate video processing than a standalone DSP processor, plus the ability to handle multiple video streams.

Reduce power and cost per channel in wireless systems

For implementing custom wireless functions, such as multi-carrier crest factor reduction (CFR), digital pre-distortion (DPD), MIMO and other advanced antenna processing, our FPGAs lower your costs and power per channel. With up to 256 GMAC/s performance, you have the advantage of offloading compute intensive tasks from a TI DSP to a Xilinx FPGA while increasing channel density in your wireless system.

Visit us at *www.xilinx.com/dsp/coprocessing* to learn more about the highest-performance DSP in the industry, and download your FREE evaluation copy of XtremeDSP software.

XILINX®

www.xilinx.com/dsp/coprocessing

FORTUNE® 2005
100 BEST COMPANIES TO WORK FOR

PN 0010925