



magazine

# Embedded

EMBEDDED SOLUTIONS FOR PROGRAMMABLE LOGIC DESIGNS

## Endless Possibilities

### INSIDE

**New EDK 8.1 Simplifies  
Embedded Design**

**Change Is Good**

**ESL Tools for FPGAs**

**Algorithmic Acceleration  
Through Automated  
Generation of  
FPGA Coprocessors**

**Bringing Floating-Point  
Math to the Masses**

 **XILINX®**

# Support Across The Board.™



## Accelerate Your Learning Curve on New Application Solutions

### Xilinx Spring 2006 SpeedWay Series

- Xilinx MicroBlaze™ Development Workshop
- Xilinx PowerPC® Development Workshop
- Xilinx Embedded Software Development Workshop
- Introduction to FPGA Design Workshop
- Creating a Low-Cost PCI Express Design Workshop
- Embedded Networking with Xilinx FPGAs Workshop
- Xilinx DSP Development Workshop
- Xilinx DSP for Video Workshop
- Improving Design Performance Workshop

Avnet Electronics Marketing offers a series of technical, hands-on SpeedWay Design Workshops™ that will dramatically accelerate your learning curve on new application solutions, products and technologies like the Philips-Xilinx PCI Express two-chip solution. Our FAE presenters use detailed laboratory exercises and Avnet developed design kits to reinforce the key topics presented during the workshops, ensuring that when you leave the class you will be able to apply newly learned concepts to your current design.

- Every workshop features an Avnet developed design kit
- Workshops are systems and solutions focused
- Design alternatives and trade-offs for targeted applications are discussed

For more information about upcoming Xilinx SpeedWay workshops, visit:

[www.em.avnet.com/xilixspeedway](http://www.em.avnet.com/xilixspeedway)



*Enabling success from the center of technology™*

1 800 332 8638  
[www.em.avnet.com](http://www.em.avnet.com)



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

[www.xilinx.com/xcell/embedded](http://www.xilinx.com/xcell/embedded)



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

© 2006 Xilinx, Inc. All rights reserved. XILINX, the Xilinx Logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

The articles, information, and other materials included in this issue are provided solely for the convenience of our readers. Xilinx makes no warranties, express, implied, statutory, or otherwise, and accepts no liability with respect to any such articles, information, or other materials or their use, and any use thereof is solely at the risk of the user. Any person or entity using such information in any way releases and waives any claim it might have against Xilinx for any loss, damage, or expense caused thereby.

# Endless Possibilities

Welcome to our third edition of Xilinx *Embedded Magazine*. As we prepared this issue, the theme for this year's Embedded Systems Conference – “Five Days, One Location, Endless Possibilities” – resonated with the array of potential articles. Simply stated, we seemed to have endless possibilities for our embedded solutions to choose from and to share with you.

To capitalize on this theme, our ease-of-use initiative continues with Xilinx® Platform Studio and the Embedded Development Kit (EDK), as we recently released our latest version, 8.1i. This comes on the heels of EDN's recognition of our 32-bit MicroBlaze™ soft-processor core as one of the “Hot 100 Products of 2005.” Taken together, the MicroBlaze core, the industry-standard PowerPC™ core embedded in our Virtex™ family of FPGAs, and a growing list of IP and supported industry standards offer more options than ever to create, debug, and launch an embedded system for production.

The latest version of our kit serves one of the greatest appeals that the embedded solution holds for our FPGA customers – to create a “just-what-I-needed” processor subsystem that “just works.” In so doing, our customers can concentrate on the added value that differentiates their products in their marketplace. Here again, “endless possibilities” resonates with unlimited design flexibility.

In this issue of *Embedded Magazine* we offer a collection of diverse articles unlocking the endless possibilities with Xilinx platforms. We welcome industry icon Jim Turley and his clever insight regarding shifts in the embedded industry with his article “Change Is Good.” In addition, our partners Echolab, Impulse, PetaLogix, Poseidon, Teja, Avnet, and Nu Horizons highlight their latest innovations for our embedded platforms. Our own experts provide tutorials on the latest release of EDK 8.1, along with a background look at the newly launched Xilinx ESL Initiative.

Join us as we plumb the depths of these exciting new embedded solutions. I'm sure you'll find our third edition of *Embedded Magazine* informative and inspiring as we endeavor to help you unlock the power of Xilinx programmability. The advantages are enormous, the possibilities ... endless!



Mark Aaldering

Vice President  
Embedded Processing  
& IP Divisions





IBM, the IBM logo and the On Demand Business logo are registered trademarks or trademarks of International Business Machines Corporation in the United States and/or other countries. Other company, product and service names may be trademarks or service marks of others. ©2006 IBM Corporation. All rights reserved.

# YOU CAN DELIVER INNOVATION

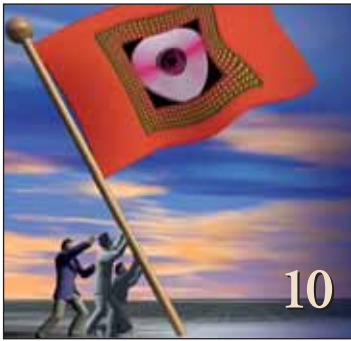
Time to market, developer and programmer productivity, choice in fabrication facilities and EDA retooling costs for smaller and smaller geometries are all putting tremendous strain on system development groups around the globe.

Enter IBM. Whether your design priorities are low power or high performance, or both, IBM's Power Architecture™ microprocessors and cores can help you accelerate innovation in your designs. Find out what the world's fastest supercomputer, Internet routers and switches, the Mars Rover, and the next generation game consoles all have in common. For more information visit [ibm.com/power](http://ibm.com/power)

**ON DEMAND BUSINESS™**



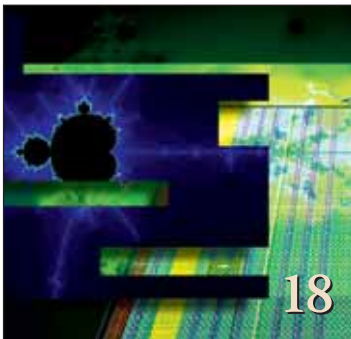
6



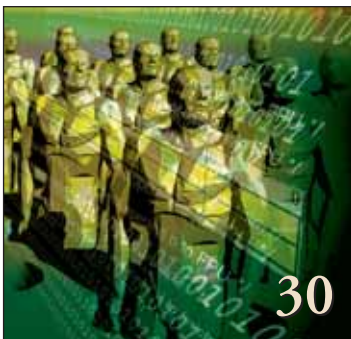
10



15



18



30

CONTENTS

Welcome .....3  
 Contents .....5

ARTICLES

New EDK 8.1 Simplifies Embedded Design .....6  
 Change Is Good .....10  
 Implementing Floating-Point DSP .....12  
 ESL Tools for FPGAs.....15  
 Algorithmic Acceleration Through Automated Generation of FPGA Coprocessors .....18  
 Generating Efficient Board Support Packages.....24  
 Bringing Floating-Point Math to the Masses .....30  
 Packet Subsystem on a Chip.....34  
 Accelerating FFTs in Hardware Using a MicroBlaze Processor.....38  
 Eliminating Data Corruption in Embedded Systems.....42  
 Boost Your Processor-Based Performance with ESL Tools.....46

CUSTOMER SUCCESS

Unveiling Nova .....51

WEBCASTS

Implementing a Lightweight Web Server Using PowerPC and Tri-Mode Ethernet MAC in Virtex-4 FX FPGAs .....57

BOARDS

Development Kits Accelerate Embedded Design .....61

PRODUCTS

MicroBlaze – The Low-Cost and Flexible Processing Solution .....64



# New EDK 8.1 Simplifies Embedded Design

Platform Studio enhancements streamline processor system development.

Platform Studio™

by Jay Gould  
Product Marketing Manager,  
Xilinx Embedded Solutions Marketing  
Xilinx, Inc.  
[jay.gould@xilinx.com](mailto:jay.gould@xilinx.com)

After achieving an industry milestone, what's next? In 2005, the Xilinx® Platform Studio tool suite (XPS) included in the Embedded Development Kit (EDK) won the IEC's DesignVision Award for innovation in embedded design. The revolutionary approach of design wizards brought abstraction and automation to an otherwise manual and error-prone development process for embedded system creation.

The year 2006 brings a new version 8.1 update to the Platform Studio tool suite, with an emphasis on simplifying the development process and providing a more visible environment. The result is a shortened learning curve for new users and an even more complete and easier-to-use environment for existing designers.

## Xilinx has updated the main user interface of Platform Studio to provide an intuitive feel for both hardware and software engineers ...

Just getting a complex design started can take a significant amount of time out of a critical schedule, so Xilinx started with a premise that the first steps to a working core design should be automated. The Xilinx Base System Builder design wizard within the Platform Studio tool suite provides a step-by-step interface to walk you through the critical first stages of a design. Design wizards are a great innovation because they can provide a quick path to a working core design even if you have minimal expertise. The “smarter” the install wizard is, the fewer issues occur, and the less experience you need to have.

Pre-configured hardware/software development kits are also extremely valuable for getting a design “off the napkin” and into a quick but stable state. Xilinx hardware/software development kits provide working hardware boards, hardware-aware tools, and pre-verified reference designs. The benefit here is that you can power up hardware, download a working design to a board, and start investigating a “working” core system in a very short period of time, skipping past the delays and complexities of debugging new hardware, new firmware, and new software all at the same time.

A majority of the embedded design cycle, before full system verification, is spent iterating on the core design, incrementally introducing new features, adding individual capabilities, and repeatedly debugging after each step. Because this is excessively tedious and time consuming, this stage should be as easy and streamlined as possible. Version 8.1 has a focus on making common (and repetitive) tasks simple and intuitive, benefiting both new and existing users.

### All Users Benefit from V8.1

Xilinx has updated the main user interface of Platform Studio to provide an intuitive feel for both hardware and software engineers, making multiple views and customization easy for all. The integrated development environment (IDE) in Figure 1 displays a wide array of information, but also allows you to filter views and customize the toolbars. The left-hand pane provides an industry-standard “tab” method of displaying or hiding information panels on the design “Project,” “Applications,” or “IP Catalog.” Just toggle on the tab of choice to display the contents of that pane.

The “Project” tab contains a variety of helpful information about the design, including specific Xilinx device selection and settings (for example, a specific Virtex™-4 or Virtex-II Pro device with one or two PowerPC™ processor cores) and project file locations (hardware and software project descriptions as well as log and report files for steps like synthesis), as well as simulation setup details.

You can view software applications under the “Applications” tab, which provides access to all of the C source and header files that make up the embedded system design. This view also provides views of the compiler options and even the block RAM initialization process.

The “IP Catalog” tab contains in-depth information about the IP cores created, bought, or imported for the design. Xilinx provides several scores of processing IP cores in the Embedded Development Kit software bundle as well as some high-value cores for time-limited evaluations. You can research Xilinx processor IP at [www.xilinx.com/ise/embedded/edk\\_ip.htm](http://www.xilinx.com/ise/embedded/edk_ip.htm).

The middle panel is the “Connectivity” view, and the adjacent panel to the right of that is the associated “System Assembly” view. The connectivity view gives a clear visual of the design busing structure and also provides a dynamic tool for creating new or editing existing connections. The color-coded view quickly makes it clear – even to novice users – the specifics of the bus type and how it might relate to IP. For example, in this view, peripherals connected to the PLB (processor local bus) are presented in orange; OPB (on-chip peripheral bus) connections are green; and point-to-point connections with a processor core, in this case the PowerPC 405, are in purple. The panel “filter” buttons allow you to customize or simplify the connection views so that you can focus on specific bus elements without the distraction of other elements.

Platform Studio reduces the errors that a designer might make by maintaining correct connections by construction – that is,

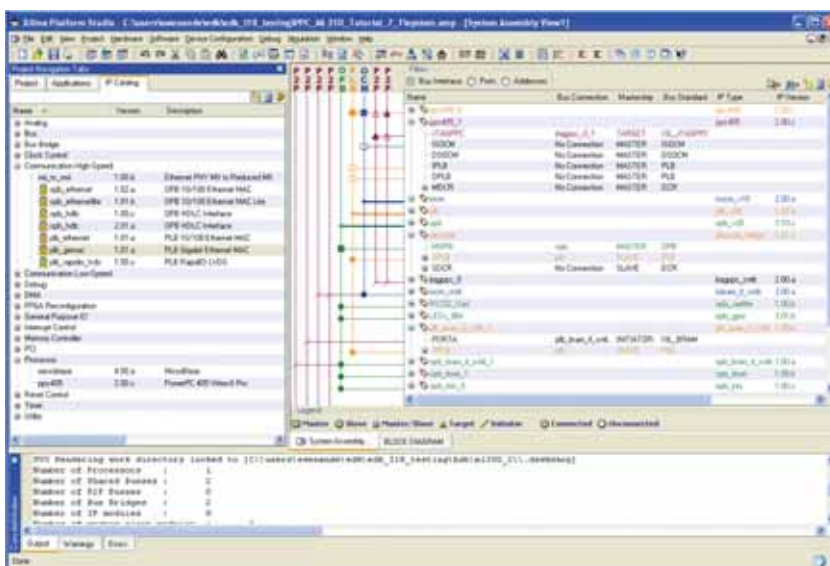


Figure 1 – New 8.1 Platform Studio GUI



XPS will only display connection options for compatible bus types. This saves debug headaches with tools that allow incompatible connections.

The system assembly view (see Figure 2) more clearly displays an example of dynamic system construction using a “drag-and-drop connectivity instantiation.” In the figure, the gray highlighted “opb\_uartlite” IP core is selected on the left panel from the IP Catalog and has been dragged and dropped into the right assembly window, creating a new OPB bus connection option automatically; just mouse-click to connect. The views on the right also provide helpful information such as IP types for perusing and IP version numbers for project version control. Now, at a glance, you can distinguish the system structure without reading reams of documentation.

However, if design documentation is what your project and team require, Platform Studio 8.1 has the powerful capability to generate full design-reference material, including a full block diagram view of the system elements and their interconnections. This automatic generation of the docs saves valuable time (instead of creating the materials manually) and reduces errors by creating the materials directly from the design. This method keeps the docs and the design accurately in sync as well as displaying a clear high-level view of the entire project.

### New Enhancements Help Existing Users

Current Platform Studio users will be pleased to see advances in the support of sophisticated software development, IP support, and the migration or upgrades of older designs. Figure 3 is an example of what the IP Catalog tab might look like for a design, including all IP cores categorically grouped on the left-hand side by logical names. The specific IP cores will display a version number for design control as well

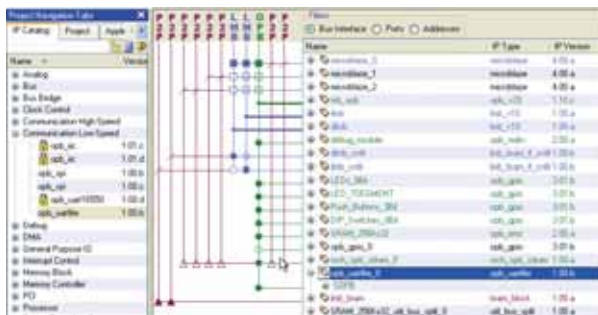


Figure 2 – System assembly view

Name	Description	Date	Processor Support	Type	Early Access
opb_uartlite	OPB UARTlite	1.0.0	PPC	PERIPHERAL	
opb_uartlite_2	OPB UARTlite 2	1.0.1	PPC	PERIPHERAL	
opb_uartlite_3	OPB UARTlite 3	1.0.2	PPC	PERIPHERAL	
opb_uartlite_4	OPB UARTlite 4	1.0.3	PPC	PERIPHERAL	
opb_uartlite_5	OPB UARTlite 5	1.0.4	PPC	PERIPHERAL	
opb_uartlite_6	OPB UARTlite 6	1.0.5	PPC	PERIPHERAL	
opb_uartlite_7	OPB UARTlite 7	1.0.6	PPC	PERIPHERAL	
opb_uartlite_8	OPB UARTlite 8	1.0.7	PPC	PERIPHERAL	
opb_uartlite_9	OPB UARTlite 9	1.0.8	PPC	PERIPHERAL	
opb_uartlite_10	OPB UARTlite 10	1.0.9	PPC	PERIPHERAL	
opb_uartlite_11	OPB UARTlite 11	1.0.10	PPC	PERIPHERAL	
opb_uartlite_12	OPB UARTlite 12	1.0.11	PPC	PERIPHERAL	
opb_uartlite_13	OPB UARTlite 13	1.0.12	PPC	PERIPHERAL	
opb_uartlite_14	OPB UARTlite 14	1.0.13	PPC	PERIPHERAL	
opb_uartlite_15	OPB UARTlite 15	1.0.14	PPC	PERIPHERAL	
opb_uartlite_16	OPB UARTlite 16	1.0.15	PPC	PERIPHERAL	
opb_uartlite_17	OPB UARTlite 17	1.0.16	PPC	PERIPHERAL	
opb_uartlite_18	OPB UARTlite 18	1.0.17	PPC	PERIPHERAL	
opb_uartlite_19	OPB UARTlite 19	1.0.18	PPC	PERIPHERAL	
opb_uartlite_20	OPB UARTlite 20	1.0.19	PPC	PERIPHERAL	
opb_uartlite_21	OPB UARTlite 21	1.0.20	PPC	PERIPHERAL	
opb_uartlite_22	OPB UARTlite 22	1.0.21	PPC	PERIPHERAL	
opb_uartlite_23	OPB UARTlite 23	1.0.22	PPC	PERIPHERAL	
opb_uartlite_24	OPB UARTlite 24	1.0.23	PPC	PERIPHERAL	
opb_uartlite_25	OPB UARTlite 25	1.0.24	PPC	PERIPHERAL	
opb_uartlite_26	OPB UARTlite 26	1.0.25	PPC	PERIPHERAL	
opb_uartlite_27	OPB UARTlite 27	1.0.26	PPC	PERIPHERAL	
opb_uartlite_28	OPB UARTlite 28	1.0.27	PPC	PERIPHERAL	
opb_uartlite_29	OPB UARTlite 29	1.0.28	PPC	PERIPHERAL	
opb_uartlite_30	OPB UARTlite 30	1.0.29	PPC	PERIPHERAL	
opb_uartlite_31	OPB UARTlite 31	1.0.30	PPC	PERIPHERAL	
opb_uartlite_32	OPB UARTlite 32	1.0.31	PPC	PERIPHERAL	
opb_uartlite_33	OPB UARTlite 33	1.0.32	PPC	PERIPHERAL	
opb_uartlite_34	OPB UARTlite 34	1.0.33	PPC	PERIPHERAL	
opb_uartlite_35	OPB UARTlite 35	1.0.34	PPC	PERIPHERAL	
opb_uartlite_36	OPB UARTlite 36	1.0.35	PPC	PERIPHERAL	
opb_uartlite_37	OPB UARTlite 37	1.0.36	PPC	PERIPHERAL	
opb_uartlite_38	OPB UARTlite 38	1.0.37	PPC	PERIPHERAL	
opb_uartlite_39	OPB UARTlite 39	1.0.38	PPC	PERIPHERAL	
opb_uartlite_40	OPB UARTlite 40	1.0.39	PPC	PERIPHERAL	
opb_uartlite_41	OPB UARTlite 41	1.0.40	PPC	PERIPHERAL	
opb_uartlite_42	OPB UARTlite 42	1.0.41	PPC	PERIPHERAL	
opb_uartlite_43	OPB UARTlite 43	1.0.42	PPC	PERIPHERAL	
opb_uartlite_44	OPB UARTlite 44	1.0.43	PPC	PERIPHERAL	
opb_uartlite_45	OPB UARTlite 45	1.0.44	PPC	PERIPHERAL	
opb_uartlite_46	OPB UARTlite 46	1.0.45	PPC	PERIPHERAL	
opb_uartlite_47	OPB UARTlite 47	1.0.46	PPC	PERIPHERAL	
opb_uartlite_48	OPB UARTlite 48	1.0.47	PPC	PERIPHERAL	
opb_uartlite_49	OPB UARTlite 49	1.0.48	PPC	PERIPHERAL	
opb_uartlite_50	OPB UARTlite 50	1.0.49	PPC	PERIPHERAL	
opb_uartlite_51	OPB UARTlite 51	1.0.50	PPC	PERIPHERAL	
opb_uartlite_52	OPB UARTlite 52	1.0.51	PPC	PERIPHERAL	
opb_uartlite_53	OPB UARTlite 53	1.0.52	PPC	PERIPHERAL	
opb_uartlite_54	OPB UARTlite 54	1.0.53	PPC	PERIPHERAL	
opb_uartlite_55	OPB UARTlite 55	1.0.54	PPC	PERIPHERAL	
opb_uartlite_56	OPB UARTlite 56	1.0.55	PPC	PERIPHERAL	
opb_uartlite_57	OPB UARTlite 57	1.0.56	PPC	PERIPHERAL	
opb_uartlite_58	OPB UARTlite 58	1.0.57	PPC	PERIPHERAL	
opb_uartlite_59	OPB UARTlite 59	1.0.58	PPC	PERIPHERAL	
opb_uartlite_60	OPB UARTlite 60	1.0.59	PPC	PERIPHERAL	
opb_uartlite_61	OPB UARTlite 61	1.0.60	PPC	PERIPHERAL	
opb_uartlite_62	OPB UARTlite 62	1.0.61	PPC	PERIPHERAL	
opb_uartlite_63	OPB UARTlite 63	1.0.62	PPC	PERIPHERAL	
opb_uartlite_64	OPB UARTlite 64	1.0.63	PPC	PERIPHERAL	
opb_uartlite_65	OPB UARTlite 65	1.0.64	PPC	PERIPHERAL	
opb_uartlite_66	OPB UARTlite 66	1.0.65	PPC	PERIPHERAL	
opb_uartlite_67	OPB UARTlite 67	1.0.66	PPC	PERIPHERAL	
opb_uartlite_68	OPB UARTlite 68	1.0.67	PPC	PERIPHERAL	
opb_uartlite_69	OPB UARTlite 69	1.0.68	PPC	PERIPHERAL	
opb_uartlite_70	OPB UARTlite 70	1.0.69	PPC	PERIPHERAL	
opb_uartlite_71	OPB UARTlite 71	1.0.70	PPC	PERIPHERAL	
opb_uartlite_72	OPB UARTlite 72	1.0.71	PPC	PERIPHERAL	
opb_uartlite_73	OPB UARTlite 73	1.0.72	PPC	PERIPHERAL	
opb_uartlite_74	OPB UARTlite 74	1.0.73	PPC	PERIPHERAL	
opb_uartlite_75	OPB UARTlite 75	1.0.74	PPC	PERIPHERAL	
opb_uartlite_76	OPB UARTlite 76	1.0.75	PPC	PERIPHERAL	
opb_uartlite_77	OPB UARTlite 77	1.0.76	PPC	PERIPHERAL	
opb_uartlite_78	OPB UARTlite 78	1.0.77	PPC	PERIPHERAL	
opb_uartlite_79	OPB UARTlite 79	1.0.78	PPC	PERIPHERAL	
opb_uartlite_80	OPB UARTlite 80	1.0.79	PPC	PERIPHERAL	
opb_uartlite_81	OPB UARTlite 81	1.0.80	PPC	PERIPHERAL	
opb_uartlite_82	OPB UARTlite 82	1.0.81	PPC	PERIPHERAL	
opb_uartlite_83	OPB UARTlite 83	1.0.82	PPC	PERIPHERAL	
opb_uartlite_84	OPB UARTlite 84	1.0.83	PPC	PERIPHERAL	
opb_uartlite_85	OPB UARTlite 85	1.0.84	PPC	PERIPHERAL	
opb_uartlite_86	OPB UARTlite 86	1.0.85	PPC	PERIPHERAL	
opb_uartlite_87	OPB UARTlite 87	1.0.86	PPC	PERIPHERAL	
opb_uartlite_88	OPB UARTlite 88	1.0.87	PPC	PERIPHERAL	
opb_uartlite_89	OPB UARTlite 89	1.0.88	PPC	PERIPHERAL	
opb_uartlite_90	OPB UARTlite 90	1.0.89	PPC	PERIPHERAL	
opb_uartlite_91	OPB UARTlite 91	1.0.90	PPC	PERIPHERAL	
opb_uartlite_92	OPB UARTlite 92	1.0.91	PPC	PERIPHERAL	
opb_uartlite_93	OPB UARTlite 93	1.0.92	PPC	PERIPHERAL	
opb_uartlite_94	OPB UARTlite 94	1.0.93	PPC	PERIPHERAL	
opb_uartlite_95	OPB UARTlite 95	1.0.94	PPC	PERIPHERAL	
opb_uartlite_96	OPB UARTlite 96	1.0.95	PPC	PERIPHERAL	
opb_uartlite_97	OPB UARTlite 97	1.0.96	PPC	PERIPHERAL	
opb_uartlite_98	OPB UARTlite 98	1.0.97	PPC	PERIPHERAL	
opb_uartlite_99	OPB UARTlite 99	1.0.98	PPC	PERIPHERAL	
opb_uartlite_100	OPB UARTlite 100	1.0.99	PPC	PERIPHERAL	

Figure 3 – XPS IP catalog

as a brief language description if the names are too brief for context. This view allows you to manage your old and current IP as well as future IP upgrades (more powerful versions of cores with more features but often faster and smaller in size).

Additional information is available as well, such as which processor cores the IP supports. Because Xilinx offers flexible support for both high-performance PowerPC hard and flexible MicroBlaze™ soft-processor cores, it is useful to know which IP cores are dedicated to one processor, the other, or both. In fact, a right mouse-click on the IP from the catalog yields quick access to the IP change history as well as complete PDF datasheets on the specifics. Software drivers for the peripherals have a similar platform settings view for clarity, including version control and embedded OS support.

When a new version of tools and IP becomes available, the upward design

migration ought to be as painless as possible. Nobody wants to re-invest design, debugging, and test time to move an older design to a newer set of tools or IP. However, there are often great advantages in new IP/tools that make it advantageous to upgrade. Platform Studio 8.1 has a migration capability (Figure 4) that steps you through a wizard to automate and accelerate the process.

XPS 8.1 can browse existing design projects, flag out-of-date projects and IP cores, and then walk you through the process of confirming automated updates to the new IP and project files. The migration wizard updates the project description files and summarizes the migration changes in document form. Minimizing labor-intensive steps means that you can take advantage of new advancements without as much manual re-entering or porting of designs.

Savvy software developers working on more sophisticated code applications will be happy with the enhancements to the

XPS Software Development Kit IDE, based on Eclipse. The XPS-SDK has an enhanced toolbar that more logically groups similar functions and buttons while still allowing user customization.



Figure 4 – XPS design migration



Version 8.1 introduces a more powerful C/C++ editor supporting code folding of functions, methods, classes, structures, and macros, as well as new compiler advancements. This new support provides the ability to specify linker scripts and customized compiler options for PowerPC and MicroBlaze processor cores, plus a C++ class creation wizard. Combine this powerful software environment with the innovative performance profiling views and unique XPS capability of integrated hardware/software debuggers, and 8.1 users will be creating better, more powerful embedded systems in less time than ever before.

### Conclusion

The award-winning Platform Studio has already streamlined embedded system design. Automated design wizards and pre-configured hardware/software development kits help kick-start designs while reducing errors and tail-chasing.

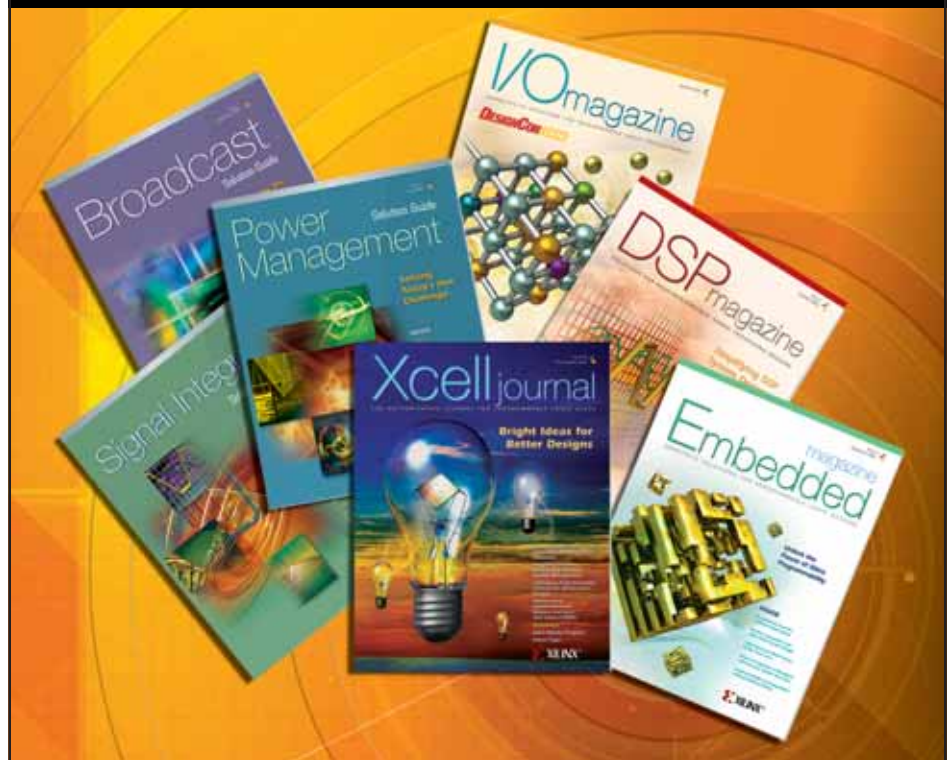
Now that we have an industry-proven success in ramping-up the “getting started” process, it is time to improve the time-consuming and cyclical nature at the heart of the development process. Create – Debug – Edit – Repeat. Have you ever used a computer-aided tool where most of the steps were intuitive? Where you could guess what a button did before you read the manual or saw a screen in which the contents were all self-evident?

EDK/XPS version 8.1 focuses on ease-of-use improvements across the board, including enhancements to the main user interface, the software development environment (including editing and compiling), the upgrading of IP, the migrating of old projects, documenting designs, viewing and editing bus-based systems, and much more.

By making common tasks simple and intuitive, we can make designing a little bit easier for experienced embedded engineers as well as those brand-new to designing with processors in programmable FPGA platforms. Use the extra time saved during the development process to innovate your own embedded products.

For more information about EDK version 8.1 and all of our embedded processing solutions, visit [www.xilinx.com/edk](http://www.xilinx.com/edk).

# WHAT'S NEW



To complement our flagship publication *Xcell Journal*, we've recently launched three new technology magazines:

- *Embedded Magazine*, focusing on the use of embedded processors in Xilinx® programmable logic devices.
- *DSP Magazine*, focusing on the high-performance capabilities of our FPGA-based reconfigurable DSPs.
- *I/O Magazine*, focusing on the wide range of serial and parallel connectivity options available in Xilinx devices.

In addition to these new magazines, we've created a family of Solution Guides, designed to provide useful information on a wide range of hot topics such as

*Broadcast Engineering, Power Management, and Signal Integrity.*

Others are planned throughout the year.



See all of the new publications on our website:

[www.xilinx.com/xcell](http://www.xilinx.com/xcell)

# Change Is Good

Hardware designers and software designers can't often agree, but there is a middle ground that both might enjoy.

by Jim Turley  
Editor in Chief, Embedded Systems Design  
CMP Media LLC  
[jim@jimturley.com](mailto:jim@jimturley.com)

If you shout “microprocessor” in a crowded theatre, most people will think “Pentium.” Intel’s famous little chip has captured the public imagination to the point where many people think 90% of all the chips made come streaming out of Intel’s factories.

Nothing could be further from the truth. The fact is, Pentium accounts for less than 2% of all of the microprocessors made and sold throughout the world. The lions’ share – that other 98% – are processors embedded into everyday appliances, automobiles, cell phones, washers, dryers, DVD players, video games, and a million other “invisible computers” all around us. PCs are a statistically insignificant part of the larger world – and Pentium sales are a rounding error.

Take heart, embedded developers, for though you may toil in obscurity, your deeds are great, your creations mighty, and your number legion. With few exceptions, most engineers are embedded systems developers. We’re the rule, not the exception.



## Processors As Simulators

What is exceptional is the number of different ways we approach a problem. All PCs look pretty much the same, but there's no such thing as a typical embedded system. They're all different. We don't standardize on one operating system, one processor (or even processor family), or one power supply, package, or peripheral mix. Among 32-bit processors alone there are more than 100 different chips available from more than a dozen different suppliers, each one with happy customers designing systems around them. Hardly a homogenous group, are we?

There's even a school of thought that microprocessors themselves are a mistake – a technical dead-end. The theory goes that microprocessors merely simulate physical functions (addition, subtraction, FFT analysis), rather than performing the function directly. Decoding and executing instructions, handling interrupts, and calculating branches is all just overhead. A close look at any modern processor chip would seem to bear this theory out: only about 15% of the chip's transistors do any actual work. The rest are dedicated to cracking opcodes, handling flag bits, routing buses, managing caches, and other effluvia necessary to make the hardware do what the software tells it to do.

The only reason processors were ever invented in the first place (so the thinking goes) is because they were more malleable than “real” hardware. You could change your code over time – but you couldn't change your hardware.

But that isn't true any more.

Following this line of reasoning, the right approach is to do away with processors and software altogether and implement your functions directly in hardware. Forget that 85% of processor overhead logic and get right down to the nuts and bolts. Make every one of those little transistors work for a living. And hey, if you change your mind, you can change your hardware – if it's programmable.

## The Malleable Engineer

So now we're faced with the proverbial (and overused) paradigm shift. We can toss out everything we know about programming, operating systems, software, real-time code, compilers, boot loaders, and bit-twiddling and go straight to hard-wired hardware implementations.

Or not.

Maybe we like programming. There's something about software design that appeals to the inner artist in us. It's a whole different way of thinking compared to hardware design, at least for a lot of engineers. Software is like poetry; hardware is



like architecture. There's plenty of bad poetry because anyone can do it, but you don't see people tossing up buildings just to see if they stand. Programming requires much less discipline and training than hardware engineering. That's why there are so many programmers in the world.

This is a good thing. Really. The easier it is to enter the engineering profession, the more (and better) engineers we're likely to have. And since hardware- and software-design mindsets are different, we get to draw from a bigger cross section of the populace. Variety is good.

More to the point, it's no longer an either/or decision. The two disciplines are not mutually exclusive; engineering is not a zero-sum game. We don't have to come down firmly on the side of hardware or software; we can straddle the middle ground as it suits us. When your hardware is programmable, you can choose to “program” it or “design” it using traditional circuitry methods. Take your pick. Let whimsy or convention be your guide.

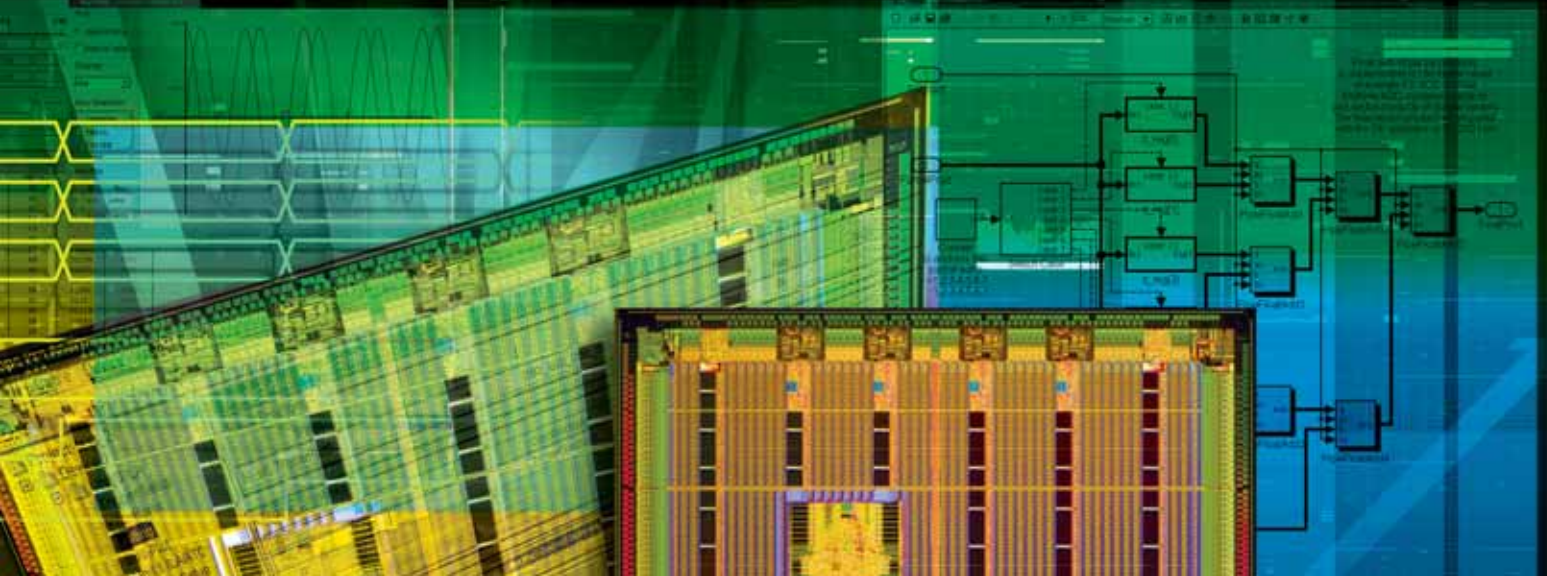
Engineers, like most craftsmen, place great stock in their tools. A recent survey revealed that most developers choose their tools (compiler, logic analyzer, IDE) first and the “platform” they work on second. For example, they let their choice of compiler determine their choice of processor, not vice versa. The hardware – a microprocessor, generally – is treated as a canvas or work piece on which they ply their trade. This comes as a bit of a blow to some of the more traditional microprocessor makers, who'd always assumed that the world revolved around their instruction set.

The takeaway from this part of the survey was that keeping developers in their comfort zone is paramount. Engineers don't like to modify their skills or habits to accommodate someone else's hardware. Instead, the hardware should adapt to them. In the best case, the hardware should even adapt to a code jockey one day and a circuit snob the next. Different tools for different approaches, but with one goal in mind: to create a great design within time and budget (and power, and heat, and pinout, and cost, and performance) constraints.

There hasn't been anything to accommodate this flexibility until pretty recently. Hardware was hardware; code was code. But with “soft processors” in FPGAs living alongside seas of gates and coprocessors, we've got the ultimate canvas for creative developers. Whether it's VHDL or C++, these new chips can be customized in whatever way suits you. They're as flexible as any software program, and as fast and efficient as “real” hardware implementations. We may finally have achieved the best of both worlds. ●●●

# Implementing Floating-Point DSP

Using PicoBlaze processors for high-performance, power-efficient, floating-point DSP.



by Jiří Kadlec  
DSP Researcher  
UTIA Prague, Czech Republic  
[kadlec@utia.cas.cz](mailto:kadlec@utia.cas.cz)

Stephen P.G. Chappell  
Director, Applications Engineering  
Celoxica Ltd., Abingdon, UK  
[steve.chappell@celoxica.com](mailto:steve.chappell@celoxica.com)

For developers using FPGAs for the implementation of floating-point DSP functions, one key challenge is how to decompose the computation algorithm into sequences of parallel hardware processes while efficiently managing data flow through the parallel pipelines of these processes.

In this article, we'll discuss our experiences exploring architectures with Xilinx® PicoBlaze™ controllers, and present a design strategy employing the ESL techniques of model-based and C-based design to demonstrate how you can rapidly integrate highly parameterizable DSP hardware primitives into power-efficient

high-performance implementations in Spartan™ devices.

## Hardware Acceleration and Reuse

High-performance implementations of floating-point DSP algorithms in FPGAs require single-cycle parallel memory accesses and effective use of pipelined arithmetic operators. Many common DSP vector and matrix operations can be split into batch calculations fulfilling these requirements. Our architectures comprise Xilinx PicoBlaze worker processors, each with a dedicated DSP hardware accelerator (Figure 1). Each worker can do preparatory tasks for the next batch in parallel with its hardware accelerator. Once the DSP hardware accelerator finishes the computation, it issues an interrupt to the worker. The worker's job is to combine the accelerated parts of the computation into a complete DSP algorithm.

It is ideal if you limit implementations to the batch operations of each worker starting in a block RAM, performing a rel-

atively simple sequence of pipelined operations at the maximum clock speed and returning the result(s) back to another block RAM. You can effectively map these primitives to hardware, including the complete autonomous data-flow control in hardware. You can also code the related dedicated generators of address counters and control signals in Handel-C, using several synchronized do-while loops. Simulink is effective for fast derivation of bit-exact models of the batch calculations in DSP hardware accelerators.

## Floating-Point Processor on a Single FPGA

Let's consider an architecture for the evaluation of a 1024 x 1024 vector product in 18m12 floating point. (In the format AmB, A is for the word length and B is for the number of bits in the mantissa, including the leading hidden bit representing 1.0.)

We implemented this architecture using five PicoBlaze processors on a single FPGA: one master and four simplified workers (Figure 1). The master is connect-



ed to the workers by I/O-mapped dual-ported block RAMs organized in 2,048 8-bit words. The master maintains the real-time base with 1  $\mu$ s resolution and provides RS232 user-interface functions. Each worker serves as a controller to a dedicated floating-point DSP hardware accelerator connected through three dual-ported block

RAMs organized in 1,024 18-bit words. Two block RAMs hold source vectors and one holds results data. In this case, the workers perform one quarter of the computation each, namely a 256 x 256 vector product. The DSP hardware accelerators are implemented in hardware, from block RAM data source to block RAM data sink,

using one 18m12 multiplier (FP MUL) and one 18m12 adder (FP ADD).

### Scalable, Short-Latency Floating-Point Modules

We used a newly released version of a scalable, short-latency pipelined floating-point library from Celoxica to build our DSP hardware accelerators. Table 1 considers some of the parameterizations of this FPGA vendor-independent library to the formats 18m12, 32m24, and 64m53. The library includes IEEE754 rounding, including the round to even. It provides bit-exact results to the Xilinx LogiCORE™ floating-point operators (v2.0), with latency set to approximately one half. The resulting maximum system clock is compatible with PicoBlaze and MicroBlaze™ embedded processors.

### Simulink and the DK Design Suite

Our design flow is based on the bit-exact modeling of Handel-C floating-point units in a Simulink framework, where the Handel-C is developed in the DK Design Suite combined simulation and synthesis environment. This enabled us to decompose a floating-point algorithm into a sequence of simple operations with rapid development and testing of different combinations.

### Step 1: Model in Simulink

First, we built a model of the DSP hardware accelerator in Simulink (Figure 2). The data sources and sinks in this model will be the block RAMs shared with the PicoBlaze worker in the final implementation.

Because the FPGA floating-point operations are written in cycle-accurate and bit-exact Handel-C, we benefited from a single source for both implementation and simulation. For modeling, we exported the Handel-C functions to S-functions using Celoxica's DK Design Suite. We then incorporated these into a bit-exact Simulink model. In this fast functional simulation, we use delay blocks in Simulink to model pipeline stages (see the 5-stage pipeline of the FP ADD operator and related registers in Figure 2). We used separate Simulink subsystems to model the bit-exact operation of the final "pipeline flushing," or "wind-up opera-

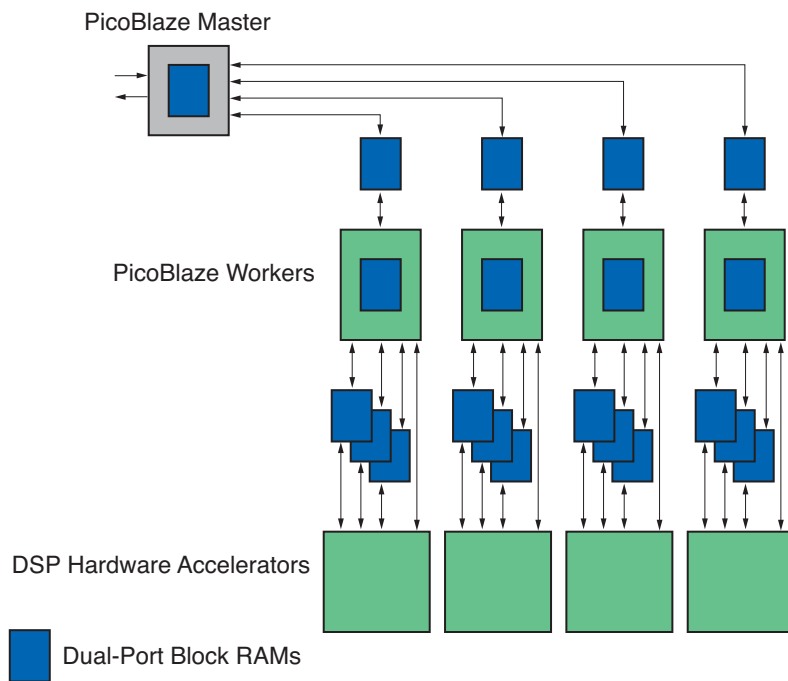


Figure 1 – PicoBlaze-based architecture for floating-point DSP. The DSP hardware accelerators are modeled and implemented using Celoxica DK.

Part:	18m12:			32m24:			64m53:		
xc2v1000-4	125 MHz			110 MHz			100 MHz		
xc3s1500L-4	84 MHz			84 MHz			72 MHz		
Pipelined:	FF	LUT	Pipe	FF	LUT	Pipe	FF	LUT	Pipe
ADD	834	793	5	1158	1290	5	1686	2007	5
MULT	639	488	3	967	626	4	2029	1256	5
F2FIXPT	581	637	4	649	744	4	808	1053	4
FIXPT2F	695	709	6	792	787	6	1008	946	6
Sequential:			Cycles			Cycles			Cycles
DIV	739	605	17	987	772	29	2119	1143	58
SQRT	766	604	16	1053	802	28	1729	1303	57

Table 1 – Used flip-flops, LUTs, pipelinel latency, and maximum clock for Celoxica floating-point modules in system implementations in the Celoxica RC200E (Virtex-II FPGA) and RC10 (Spartan-3L FPGA) boards. Modules are pipelined, with the exception of DIV and SQRT.

tion.” In this case, six partial sums have to be added by a single reused FP ADD module (Figure 3). The corresponding hardware computes the final sum of the partial sums by reconnecting the pipelined floating-point adder to different contexts for several final clock cycles.

### Step 2: Cycle-Accurate Verification

Our next stage was to create test vectors using Simulink and feed these into a bit-exact and cycle-accurate simulation of the DSP hardware accelerator in the DK Design Suite’s debugger. Once we confirmed identical results for both the DK and Simulink models, we compiled the Handel-C code to an EDIF netlist.

### Step 3: Hardware Test

We took advantage of a layered design approach by using a single communication API for data I/O functions that applies to both simulation and implementation. This allowed us to verify the DSP hardware accelerator design on real FPGA hardware by “linking” with an appropriate board support library for implementation. We can optionally insert this hardware test back into the Simulink model for hardware-in-the-loop simulations. The test on FPGA hardware provides reliable area and clock figures.

### Step 4: Create Reusable Module and Connect to Worker

Finally, we treated the verified block RAM-to-block RAM DSP hardware accelerator as a new module and integrated it into our main design by compiling the Handel-C to EDIF or RTL using the DK Design Suite. This reusable module is connected to the PicoBlaze network by wiring the ports of the block RAMs and the

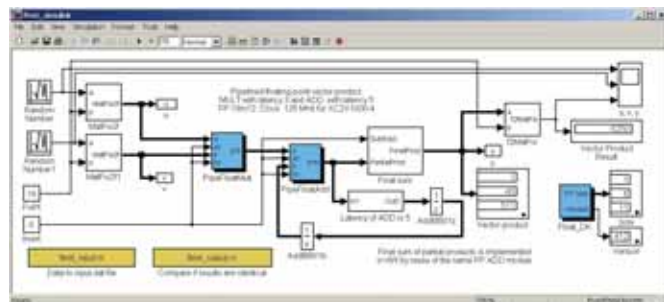


Figure 2 – Simulink test bench for floating-point 18m12 vector product based on Handel-C bit-exact models

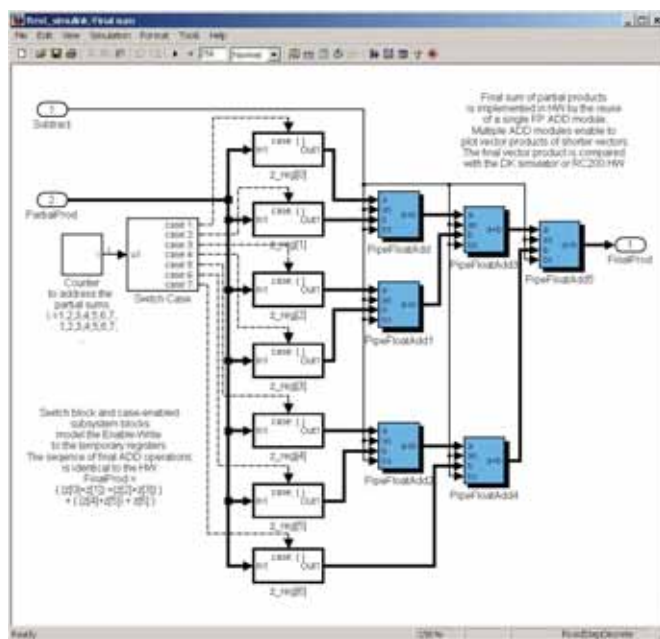


Figure 3 – Simulink subsystem based on Handel-C bit-exact models, including delay model of calculation wind-up at the end of the vector product batch

Part:	MHz	MFLOPs	mW
xc2v1000-4	100	700	1360
xc3s1500L-4	84	588	263

Table 2 – Results for 1024 x 1024 vector product in 18m12 floating point on the Celoxica RC200E (Virtex-II FPGA) and RC10 (Spartan-3L FPGA) boards

appropriate enable and controller interrupt signals. At this stage we tested the function of the DSP hardware accelerator under worker control using memory dump user support from the master.

### Step 5: Develop Complete DSP Design

We next assembled the complete design of workers and master, moving to assembly pro-

gramming of individual PicoBlaze workers and their interactions.

### Performance Results

Test results using the Celoxica RC200E (Virtex™-II FPGA) and RC10 (Spartan™-3L FPGA) boards are shown in Table 2. It is interesting to compare the power consumption of the PicoBlaze network architecture on Virtex-II devices (RC200E) with the identical design on the low-power 90 nm Spartan-3L device (RC10). The latter part gives a highly favorable floating-point performance-to-power ratio.

### Conclusion

With minimal overhead, PicoBlaze workers add flexibility to floating-point DSP hardware accelerators by their ability to call and reuse software functions (even if in assembly language only). Our proposed architecture enables more flexible and generic floating-point algorithms without the additional increase of hardware complexity associated with hardware-only implementations due to irregularities and complex multiplexing of pipelined structures. PicoBlaze cores are compact, simple, and

therefore manageable, without designers needing to combine too many new skills.

The use of floating-point designs developed using the DK Design Suite in combination with a Simulink framework provides an effective design path that is relatively easy to debug and scalable to more complex designs.

Spartan-3L technology considerably reduces power consumption compared to Virtex-II devices. Considering the benefits in terms of performance/power/price, Spartan-3L FPGA implementations of floating-point DSP pipelines using networks of PicoBlaze processors are an interesting option.

You can find complete information on the design and technology discussed in this article at [www.celoxica.com/xilinx](http://www.celoxica.com/xilinx).



# ESL Tools for FPGAs

Empowering software developers to design with programmable hardware.

by Milan Saini

Technical Marketing Manager

Xilinx, Inc.

[milan.saini@xilinx.com](mailto:milan.saini@xilinx.com)

A fundamental change is taking place in the world of logic design. A new generation of design tools is empowering software developers to take their algorithmic expressions straight into hardware without having to learn traditional hardware design techniques.

These tools and associated design methodologies are classified collectively as electronic system level (ESL) design, broadly referring to system design and verification methodologies that begin at a higher level of abstraction than the current mainstream register transfer level (RTL). ESL design languages are closer in syntax and semantics to the popular ANSI C than to hardware languages like Verilog and VHDL.

## How is ESL Relevant to FPGAs?

ESL tools have been around for a while, and many perceive that these tools are predominantly focused on ASIC design flows. The reality, however, is that an increasing number of ESL tool providers are focusing on programmable logic; currently, several tools in the market support a system design flow specifically optimized for Xilinx® FPGAs. ESL flows are a natural evolution for FPGA design tools, allowing the flexibility of programmable hardware to be more easily accessed by a wider and more software-centric user base.

Consider a couple of scenarios in which ESL and FPGAs make a great combination:

1. Together, ESL tools and programmable hardware enable a desktop-based hardware development environment that fits into a software developer's workflow model. Tools can provide optimized support for specific FPGA-based reference boards, which software developers can use to start a project evaluation or a prototype. The availability of these boards and the corresponding reference applications written in higher level languages makes creating customized, hardware-accelerated systems much faster and easier. In fact, software programmers are now able to use FPGA-based reference boards and tools in much the same way as microprocessor reference boards and tools.
2. With high-performance embedded processors now very common in FPGAs, software and hardware design components can fit into a single device. Starting from a software description of a system, you can implement individual design blocks in hardware or software depending on the applications' performance requirements. ESL tools add value by enabling intelligent partitioning and automated export of software functions into equivalent hardware functions.

ESL promotes the concept of "exploratory design and optimization." Using ESL methodologies in combination with programmable hardware, it becomes possible to try a much larger number of possible application implementations, as well as rapidly experiment with dramatically different software/hardware partitioning strategies. This ability to experiment – to try new approaches and quickly analyze performance and size trade-offs – makes it possible for ESL/FPGA users to achieve higher overall performance in less time than it would take using traditional RTL methods.

Additionally, by working at a more abstract level, you can express your intent using fewer keystrokes and writing fewer lines of code. This typically means a much faster time to design completion, and less chance of making errors that require tedious, low-level debugging.

## ESL's Target Audience

The main benefits of ESL flows for prospective FPGA users are their productivity and ease-of-use. By abstracting the implementation details involved in generating a hardware circuit, the tools are marketing their appeal to a software-centric user base (Figure 1). Working at a higher level of abstraction allows designers with skills in traditional software programming languages like C to more quickly explore their ideas in hardware. In most instances, you can implement an entire design in hardware without the assistance of an

experienced hardware designer. Software-centric application and algorithm developers who have successfully applied the benefits of this methodology to FPGAs include systems engineers, scientists, mathematicians, and embedded and firmware developers.

The profile of applications suitable for ESL methodologies includes computationally intensive algorithms with extensive inner-loop constructs. These applications can realize tremendous acceleration through the concurrent parallel execution possible in hardware. ESL tools have helped with successful project deployments in application domains such as audio/video/image processing, encryption, signal and packet processing, gene sequencing, bioinformatics, geophysics, and astrophysics.

### ESL Design Flows

ESL tools that are relevant to FPGAs cover two main design flows:

1. High-level language (HLL) synthesis. HLL synthesis covers algorithmic or behavioral synthesis, which can produce hardware circuits from C or C-like software languages. Various partner solutions take different paths to converting a high-level design description into an FPGA implementation. How this is done goes to the root of the differences between the various ESL offerings.

You can use HLL synthesis for a variety of use cases, including:

- Module generation. In this mode of use, the HLL compiler can convert a functional block expressed in C (for example, as a C subroutine) into a corresponding hardware block. The generated hardware block is then assimilated in the overall hardware/software design. In this way, the HLL compiler generates a submodule of the overall design.

Module generation allows software engineers to participate in the overall system design by quickly generating, then integrating, algorithmic hardware components. Hardware engineers seek-

ing a fast way to prototype new, computation-oriented hardware blocks can also use module generation.

- Processor acceleration. In this mode of use, the HLL compiler allows time-critical or bottleneck functions running on a processor to be accelerated by enabling the creation of a custom accelerator block in the programmable fabric of the FPGA. In addition to creating the accelerator, the tools can also automatically infer memories and generate the required hardware-software interface circuitry, as well as the software device drivers that enable communication between the processor and the hardware accelerator block (Figure 2). When compared to code running on a CPU, FPGA-accelerated code can run orders of magnitude faster while consuming significantly less power.

2. System modeling. System simulations using traditional RTL models can be very slow for large designs, or when processors are part of the complete design. A popular emerging ESL approach uses high-speed transaction-level models, typically written in C++, to significantly speed up system simulations. ESL tools provide you with a virtual platform-based verification environment where you can analyze and tune the functional and performance attributes of your design. This means much earlier access to a virtual representation of the system, enabling greater design exploration and what-if analysis. You can evaluate and refine performance issues such as latency, throughput, and bandwidth, as well as alternative software/hardware partitioning strategies. Once the design meets its performance objectives, it can be committed to implementation in silicon.

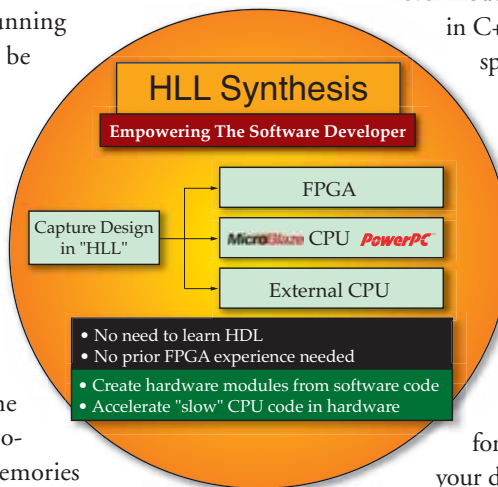


Figure 1 – Most of the ESL tools for FPGAs are targeted at a software-centric user base.

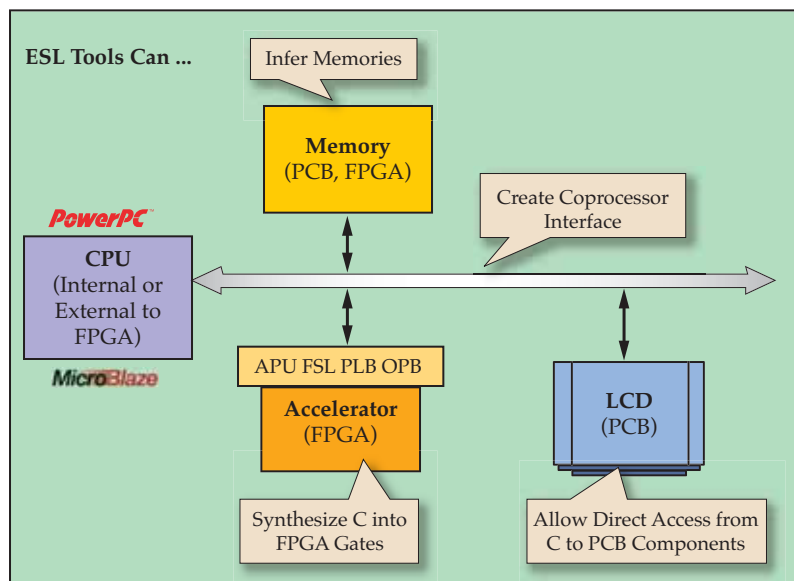


Figure 2 – ESL tools abstract the details associated with accelerating processor applications in the FPGA.



# ...today's ESL technologies are ready to deliver substantial practical value to a potentially large target audience.

## The Challenges Faced by ESL Tool Providers

In relative terms, ESL tools for FPGAs are new to the market; customer adoption remains a key challenge. One of the biggest challenges faced by ESL tool providers is overcoming a general lack of awareness as to what is possible with ESL and FPGAs, what solutions and capabilities already exist, and the practical uses and benefits of the technology. Other challenges include user apprehension and concerns over the quality of results and learning curve associated with ESL adoption.

Although paradigm shifts such as those introduced by ESL will take time to become fully accepted within the existing FPGA user community, there is a need to tackle some of the key issues that currently prohibit adoption. This is particularly important because today's ESL technologies are ready to deliver substantial practical value to a potentially large target audience.

## Xilinx ESL Initiative

Xilinx believes that ESL tools have the promise and potential to radically change the way hardware and software designers create, optimize, and verify complex electronic systems. To bring the full range of benefits of this emerging technology to its customers and to establish a common platform for ESL technologies that target FPGAs in particular, Xilinx has proactively formed a collaborative joint ESL Initiative with its ecosystem partners (Table 1).

The overall theme of the initiative is to accelerate the pace of ESL innovation for FPGAs and to bring the technology closer to the needs of the software-centric user base. As part of the initiative, there are two main areas of emphasis:

1. Engineering collaboration. Xilinx will work closely with its partners to continue to further increase the value of ESL product offerings. This will

include working to improve the compiler quality of results and enhance tool interoperability and overall ease-of-use.

2. ESL awareness and evangelization.

Xilinx will evangelize the value and benefits of ESL flows for FPGAs to current and prospective new customers. The program will seek to inform and educate users on the types of ESL solutions that currently exist and how the various offerings can provide better approaches to solving existing problems. The aim is to empower users to make informed decisions on the suitability and fit of various partner ESL offerings to meet their specific application needs. Greater awareness will lead to increased customer adoption, which in turn will contribute to a sustainable partner ESL for FPGAs ecosystem.

## Getting Started With ESL

As a first step to building greater awareness on the various ESL for FPGA efforts, Xilinx has put together a comprehensive ESL website. The content covers the specific and unique aspects of each of the currently

available partner ESL solutions and is designed to help you decide which, if any, of the available solutions are a good fit for your applications. To get started with your ESL orientation, visit [www.xilinx.com/esl](http://www.xilinx.com/esl).

Additionally, Xilinx has also started a new ESL for FPGAs discussion forum at <http://toolbox.xilinx.com/cgi-bin/forum>. Here, you can participate in a variety of discussions on topics related to ESL design for FPGAs.

## Conclusion

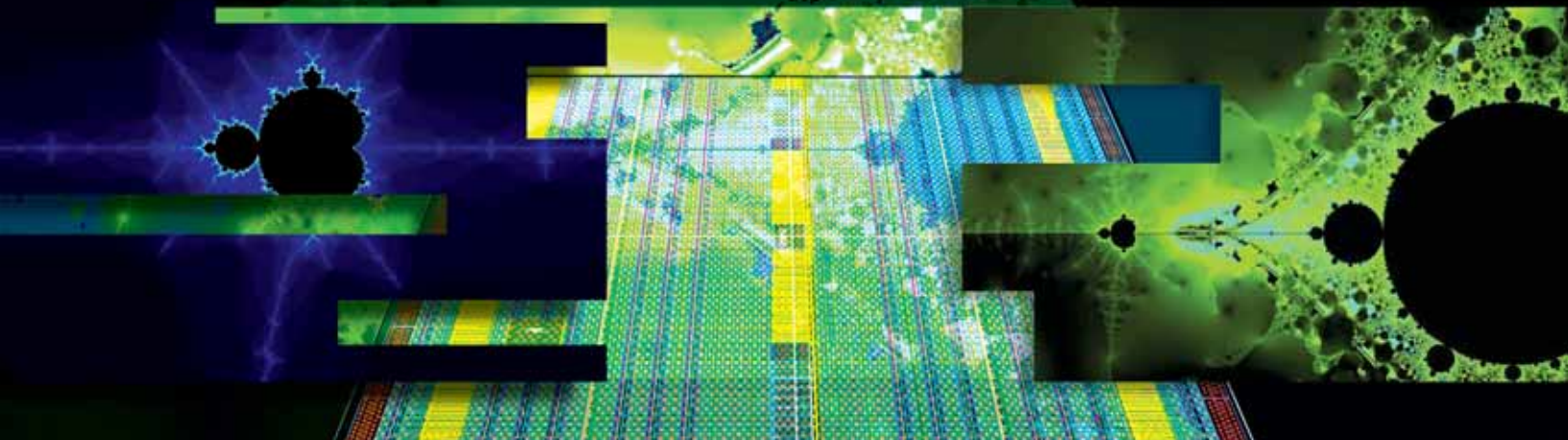
ESL tools for FPGAs give you the power to explore your ideas with programmable hardware without needing to learn low-level details associated with hardware design. Today, you have the opportunity to select from a wide spectrum of innovative and productivity-enhancing solutions that have been specifically optimized for Xilinx FPGAs. With the formal launching of the ESL Initiative, Xilinx is thoroughly committed to working with its third-party ecosystem in bringing the best-in-class ESL tools to its current and potential future customers. Stay tuned for continuing updates and new developments. 🌈

Partner	FPGA Synthesis	Xilinx CPU Support	FPGA Computing Solution
Celoxica	●	●	Handel-C, SystemC to gates
Impulse	●	●	Impulse C to gates
Poseidon	●	●	HW/SW partitioning, acceleration
Critical Blue	●		Co-processor synthesis
Teja		●	C to multi-core processing
Mittrion	●		Adaptable parallel processor in FPGA
System Crafter	●		SystemC to gates
Bluespec	●		SystemVerilog-based synthesis to RTL
Nallatech	●	●	High-performance computing

Table 1 – Xilinx ESL partners take different approaches from high-level languages to FPGA implementation.

# Algorithmic Acceleration Through Automated Generation of FPGA Coprocessors

C-to-FPGA design methods allow rapid creation of hardware-accelerated embedded systems.



by Glenn Steiner

Sr. Engineering Manager, Advanced Products Division  
Xilinx, Inc.  
[glenn.steiner@xilinx.com](mailto:glenn.steiner@xilinx.com)

Kunal Shenoy

Design Engineer, Advanced Products Division  
Xilinx, Inc.  
[kunal.shenoy@xilinx.com](mailto:kunal.shenoy@xilinx.com)

Dan Isaacs

Director of Embedded Processing, Advanced Products Division  
Xilinx, Inc.  
[dan.isaacs@xilinx.com](mailto:dan.isaacs@xilinx.com)

David Pellerin

Chief Technology Officer  
Impulse Accelerated Technologies  
[david.pellerin@impulsec.com](mailto:david.pellerin@impulsec.com)

Today's designers are constrained by space, power, and cost, and they simply cannot afford to implement embedded designs with gigahertz-class computers. Fortunately, in embedded systems, the greatest computational requirements are frequently determined by a relatively small number of algorithms. These algorithms, identified through profiling techniques, can be rapidly

converted into hardware coprocessors using design automation tools. The coprocessors can then be efficiently interfaced to the offloaded processor, yielding "gigahertz-class" performance.

In this article, we'll explore code acceleration and techniques for code conversion to hardware coprocessors. We will also demonstrate the process for making trade-off decisions with benchmark data through an actual image-rendering case study involving an auxiliary processor unit (APU)-based technique. The design uses an immersed PowerPC™ implemented in a platform FPGA.

## The Value of a Coprocessor

A coprocessor is a processing element that is used alongside a primary processing unit to offload computations normally performed by the primary processing unit. Typically, the coprocessor function implemented in hardware replaces several software instructions. Code acceleration is thus achieved by both reducing multiple code instructions to a single instruction as well as the direct implementation of the instruction in hardware.

The most frequently used coprocessor is the floating-point unit (FPU), the only common coprocessor that is tightly coupled to the CPU. There are no general-purpose libraries of coprocessors. Even if there were, it is still difficult to readily couple a coprocessor to a CPU, such as a Pentium 4.

As shown in Figure 1, the Xilinx® Virtex™-4 FX FPGA has one or two PowerPCs, each with an APU interface. By embedding a processor within an FPGA, you now have the opportunity to implement complete processing systems of your own design within a single chip.

The integrated PowerPC with APU interface enables a tightly coupled coprocessor that can be implemented within the FPGA. Frequency requirements and pin number limits make an external coprocessor less capable. Thus, you can now create application-specific coprocessors attached directly to the PowerPC, providing significant software acceleration. Because FPGAs are reprogrammable, you can rapidly develop and test CPU-attached coprocessor solutions.

## Coprocessor Connection Models

Coprocessors are available in three basic forms: CPU bus connected, I/O connected, and instruction-pipeline connected. Mixed variants also exist.

### CPU Bus Connection

Processor bus-connected accelerators require the CPU to move data and send commands through a bus. Typically, a single data transaction can require many processor cycles. Data transactions can be hindered by bus arbitration and the necessity for the bus to be clocked at a fraction of the processor clock speed. A bus-connected accelerator can include a direct memory access (DMA) engine. At the cost of additional logic, the DMA engine allows a coprocessor to operate on blocks of data located on bus-connected memory, independent of the CPU.

### I/O Connection

I/O-connected accelerators are attached directly to a dedicated I/O port. Data and control are typically provided through GET or PUT functions.

Lacking arbitration, reduced control complexity, and fewer attached devices, these interfaces are typically clocked faster than a processor bus. A good example of such an interface is the Xilinx Fast Simplex Link (FSL). The FSL is a simple FIFO interface that can be attached to either the Xilinx MicroBlaze™ soft-core processor or a Virtex-4 FX PowerPC. Data movement through the FSL has lower latency and a higher data rate than data movement through a processor bus interface.

### Instruction Pipeline Connection

Instruction-pipeline connected accelerators attach directly to the computing core of a CPU. Being coupled to the instruction pipeline, instructions not recognized by the CPU can be executed by the coprocessor. Operands, results, and status are passed

directly to and from the data execution pipeline. A single operation can result in two operands being processed, with both a result and status being returned.

As a directly connected interface, the instruction-pipeline connected accelerators can be clocked faster than a processor bus. The Xilinx implementation for this type of coprocessor connection model through the APU interface demonstrates a 10x clock cycle reduction in the control and movement

PowerPC. Although the APU connection is instruction-pipeline-based, the C-to-HDL tool set implements an I/O pipeline interface with a resulting behavior more typical of an I/O-connected accelerator.

### FPGA/PowerPC/APU Interface

FPGAs allow hardware designers to implement a complete computing system with processor, decode logic, peripherals, and coprocessors all on one chip. An FPGA can contain a few thousand to hundreds of thousands of logic cells. A processor can be implemented from the logic cells, as in the Xilinx PicoBlaze™ or MicroBlaze processors, or it can be one or more hard logic elements, as in the Virtex-4 FX PowerPC. The high number of logic cells enables you to implement data-processing elements that work with the processor system and are controlled or monitored by the processor.

FPGAs, being reprogrammable elements, allow you to program parts and test them at any stage during the design process. If you find a design flaw, you can immediately reprogram a part. FPGAs also allow you to implement hardware computing

functions that were previously cost-prohibitive. The tight coupling of a CPU pipeline to FPGA logic, as in the Virtex-4 FX PowerPC, enables you to create high-performance software accelerators.

Figure 2 is a block diagram showing the PowerPC, integrated APU controller, and an attached coprocessor. Instructions from cache or memory are simultaneously presented to the CPU decoder and the APU controller. If the CPU recognizes the instruction, it is executed. If not, the APU controller has the opportunity to acknowledge the instruction and execute it. Optionally, one or two operands can be passed to the coprocessor and a result or status can be returned. The APU interface also supports the ability to transfer a data element with a single instruction. The data element ranges

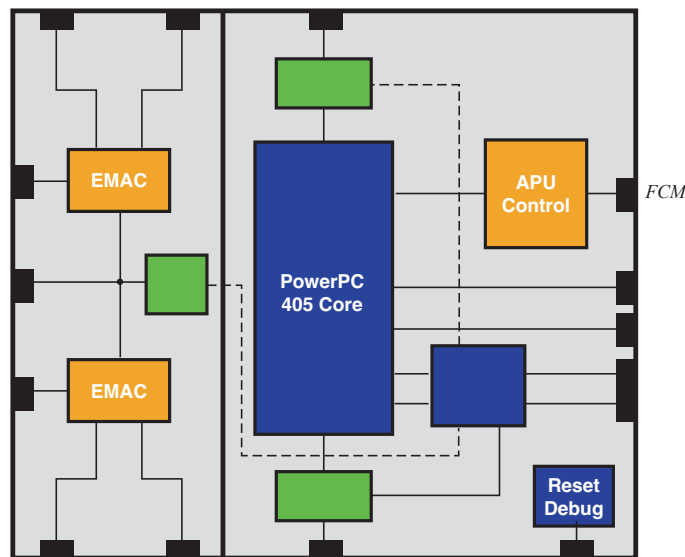


Figure 1 – Virtex-4 FX processor with APU interface and EMAC blocks

of data for a typical double-operand instruction. The APU controller is also connected to the data-cache controller and can perform data load/store operations through it. Thus, the APU interface is capable of moving hundreds of millions of bytes per second, approaching DMA speeds.

Either I/O-connected accelerators or instruction-pipeline-connected accelerators can be combined with bus-connected accelerators. At the cost of additional logic, you can create an accelerator that receives commands and returns status through a fast, low-latency interface while operating on blocks of data located in bus-connected memory.

The C-to-HDL tool set described in this article is capable of implementing bus-connected and I/O-connected accelerators. It is also capable of implementing an accelerator connected to the APU interface of the



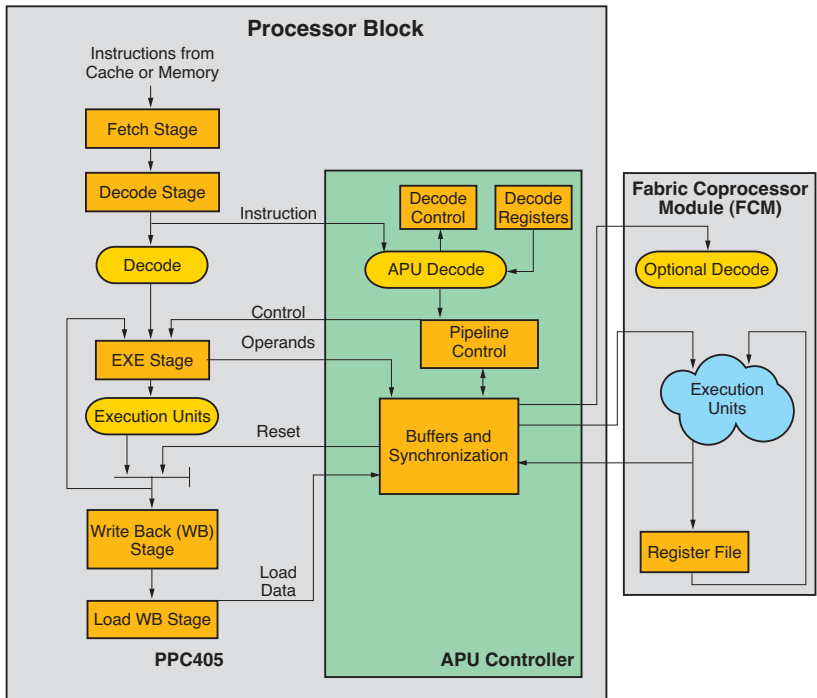


Figure 2 – PowerPC, integrated APU controller, and coprocessor

in size from one byte to four 32-bit words. One or more coprocessors can be attached to the APU interface through a fabric coprocessor bus (FCB). Coprocessors attached to the bus range from off-the-shelf cores, such as an FPU, to user-created coprocessors. A coprocessor can connect to the FCB for control and status operations and to a processor

Implementation	Performance
Software Implementation	2 MFLOPS
FPU Connected to Processor Bus	16 MFLOPS
FPU Connected to APU Interface via FCB	60 MFLOPS

Table 1 – Non-accelerated vs. accelerated floating-point performance

bus, enabling direct access to memory data blocks and DMA data passing. A simplified connection scheme, such as the FSL, can also be used between the FCB and coprocessor, enabling FIFO data and control communication at the cost of some performance.

To demonstrate the performance advantage of an instruction-pipeline-con-

nected accelerator, we first implemented a design with a processor bus-connected FPU and then with an APU/FCB-connected FPU. Table 1 summarizes the performance for a finite impulse response (FIR) filter for each case.

As noted in the table, an FPU connected to an instruction pipeline accelerates software floating-point operations by 30x, and the APU interface provides a nearly 4x improvement over a bus-connected FPU.

**Converting C Code to HDL**

Converting C code to an HDL accelerator with a C-to-HDL tool is an efficient method for creating hardware coprocessors. Figure 3 and the steps below summarize the C-to-HDL conversion process:

1. Implement the application or algorithm using standard C tools. Develop a software test bench for baseline performance and correctness (host or desktop simulations). Use a profiler (such as gprof) to begin identifying critical functions.

2. Determine if floating-to-fixed point conversion is appropriate. Use libraries or macros to aid in this conversion. Use a baseline test bench to analyze performance and accuracy. Use the profiler to reevaluate critical functions.

3. Using a C-to-HDL tool, such as Impulse C, iterate on each of the critical functions to:

- Partition the algorithm into parallel processes
- Create hardware/software process interfaces (streams, shared memories, signals)
- Automatically optimize and parallelize the critical code sections (such as inner code loops)
- Test and verify the resulting parallel algorithm using desktop simulation, cycle-accurate C simulation, and actual in-system testing.

4. Using the C-to-HDL tool, convert the critical code segment to an HDL coprocessor.

5. Attach the coprocessor to the APU interface for final testing.

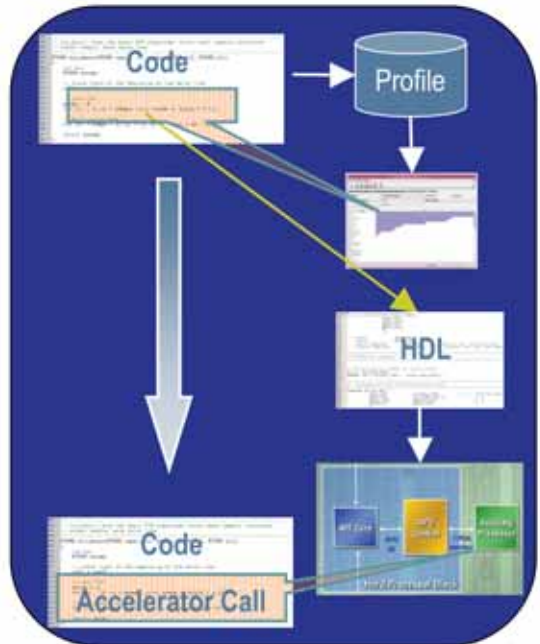


Figure 3 – C-to-HDL design flow

# Impulse C is designed for dataflow-oriented applications, but it is also flexible enough to support alternate programming models...

## Impulse: C-to-HDL Tool

Impulse C, shown in Figure 4, enables embedded system designers to create highly parallel, FPGA-accelerated applications by using C-compatible library functions in combination with the Impulse

some applications, it makes more sense to move data between the embedded processor and the FPGA through block memory reads and writes; in other cases, a streaming communication channel might provide higher performance. The ability to

- Standard C (with associated library calls) that can be compiled onto supported microprocessors through the use of widely available C cross-compilers

The complete CoDeveloper development environment includes desktop simulation libraries compatible with standard C compilers and debuggers, including Microsoft Visual Studio and GCC/GDB. Using these libraries, Impulse C programmers are able to compile and execute their applications for algorithm verification and debugging purposes. C programmers are also able to examine parallel processes, analyze data movement, and resolve process-to-process communication problems using the CoDeveloper Application Monitor.

The output of an Impulse C application, when compiled, is a set of hardware and software source files that are ready for importing into FPGA synthesis tools. These files include:

- Automatically generated HDL files representing the compiled hardware process.
- Automatically generated HDL files representing the stream, signal, and memory components needed to connect hardware processes to a system bus.
- Automatically generated software components (including a run-time library) establishing the software side of any hardware/software stream connections.
- Additional files, including script files, for importing the generated application into the target FPGA place and route environment.

The result of this compilation process is a complete application, including the required hardware/software interfaces, ready for implementation on an FPGA-based programmable platform.

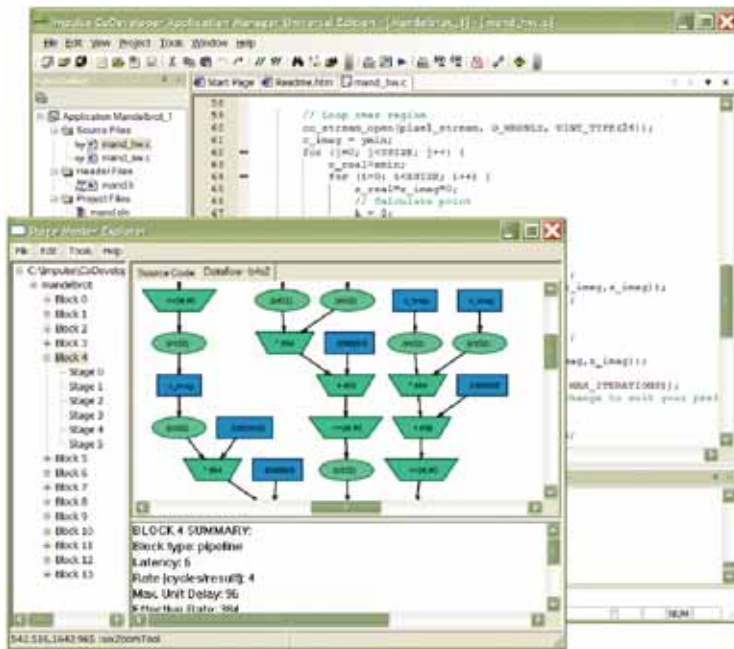


Figure 4 – Impulse C

CoDeveloper C-to-hardware compiler. Impulse C simplifies the design of mixed hardware/software applications through the use of well-defined data communication, message passing, and synchronization mechanisms. Impulse C provides automated optimization of C code (such as loop pipelining, unrolling, and operator scheduling) and interactive tools, allowing you to analyze cycle-by-cycle hardware behavior.

Impulse C is designed for dataflow-oriented applications, but it is also flexible enough to support alternate programming models, including the use of shared memory. This is important because different FPGA-based applications have different performance and data requirements. In

quickly model, compile, and evaluate alternate algorithm approaches is an important part of achieving the best possible results for a given application.

To this end, the Impulse C library comprises minimal extensions to the C language in the form of new data types and predefined function calls. Using Impulse C function calls, you can define multiple, parallel program segments (called processes) and describe their interconnections using streams, signals, and other mechanisms. The Impulse C compiler translates and optimizes these C-language processes into either:

- Lower-level HDL that can be synthesized to FPGAs, or

## Design Example

The Mandelbrot image shown in Figure 5, a classic example of fractal geometry, is widely used in the scientific and engineering communities to simulate chaotic events such as weather. Fractals are also

(hardware to software) into streams; and the addition of compiler directives to optimize the generated hardware. We subsequently used the CoDeveloper tool set to create the Pcore coprocessor that was imported into Xilinx Platform Studio (XPS). Using XPS,

## Performance Improvement Examples

We measured performance improvements for the Mandelbrot image texturing problem, an image filtering application, and triple DES encryption. Table 2 documents the performance improvements, demonstrating acceleration ranging from 11x to 34x that of software.

## Conclusion

Constrained by power, space, and cost, you might need to make a non-ideal processor choice. Frequently, it is a choice where the processor is of lower performance than desired. When the software code does not run fast enough, a coprocessor code accelerator becomes an attractive solution. You can handcraft an accelerator in HDL or use a C-to-HDL tool to automatically convert the C code to HDL.

Using a C-to-HDL tool such as Impulse C enables quick and easy accelerator generation. Virtex-4 FX FPGAs, with one or two embedded PowerPCs, enable tight coupling of the processor instruction pipeline to software accelerators. As demonstrated in this article, critical software routines can be accelerated from 10x to more than 30x, enabling a 300 MHz PowerPC to provide performance equaling or exceeding that of a high-performance multi-gigahertz processor. The above examples were generated in just a few days each, demonstrating the rapid design, implementation, and testing possible with a C-to-HDL flow. 🌈

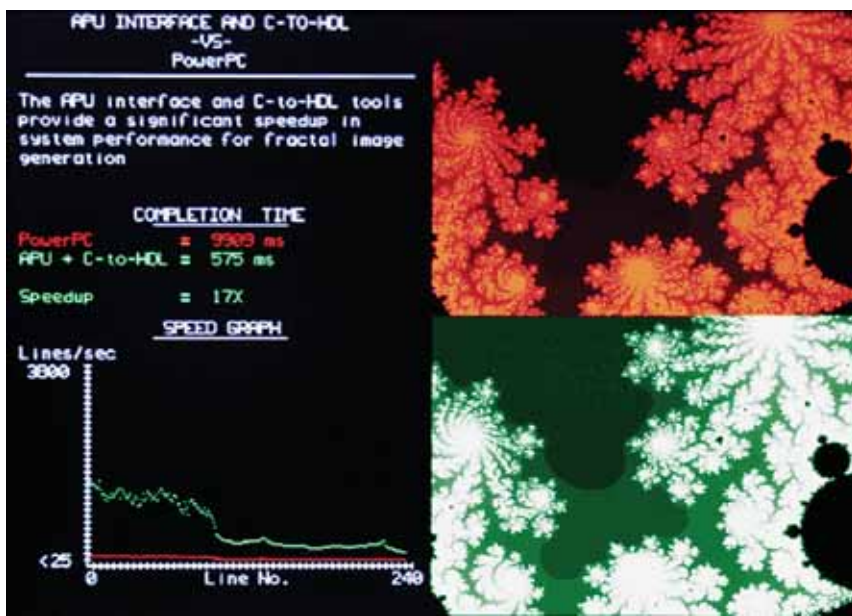


Figure 5 – Mandelbrot image and code acceleration

used to generate textures and imaging in video-rendering applications. Mandelbrot images are described as self-similar; on magnifying a portion of the image, another image similar to the whole is obtained.

The Mandelbrot image is an ideal candidate for hardware/software co-design because it has a single computation-intensive function. Making this critical function faster by moving it to the hardware domain significantly increases the speed of the whole system. The Mandelbrot application also lends itself nicely to clear divisions between hardware and software processes, making it easy to implement using C-to-HDL tools.

We used the CoDeveloper tool set as the C-to-HDL tool set for this design example. We modified a software-only Mandelbrot C program to make it compatible with the C-to-HDL tools. Our changes included division of the software project into distinct processes (independent units of sequential execution); conversion of function interfaces

we attached the PC to the PowerPC APU controller interface and tested the system.

Xilinx application note XAPP901 ([www.xilinx.com/bvdocs/appnotes/xapp901.pdf](http://www.xilinx.com/bvdocs/appnotes/xapp901.pdf)) provides a full description of the design along with design files for downloading. User Guide UG096 ([www.xilinx.com/bvdocs/userguides/ug096.pdf](http://www.xilinx.com/bvdocs/userguides/ug096.pdf)) provides a step-by-step tutorial in implementing the design example.

Application	PowerPC Only (300 MHz)	PowerPC + Coprocessor (300/50 MHz)	Acceleration
Image Texturing (Mandelbrot/Fractal)	21 sec	1.2 sec	17x
Image Filter (Edge Detection)	0.14 sec	0.012 sec	11x
Encryption (Triple DES)	2.3 sec	0.067 sec	34x

Table 2 – Algorithm acceleration through coprocessor accelerators



# High **VELOCITY** LEARNING



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](http://www.nuhorizons.com/xpresstrack)



## Courses Available

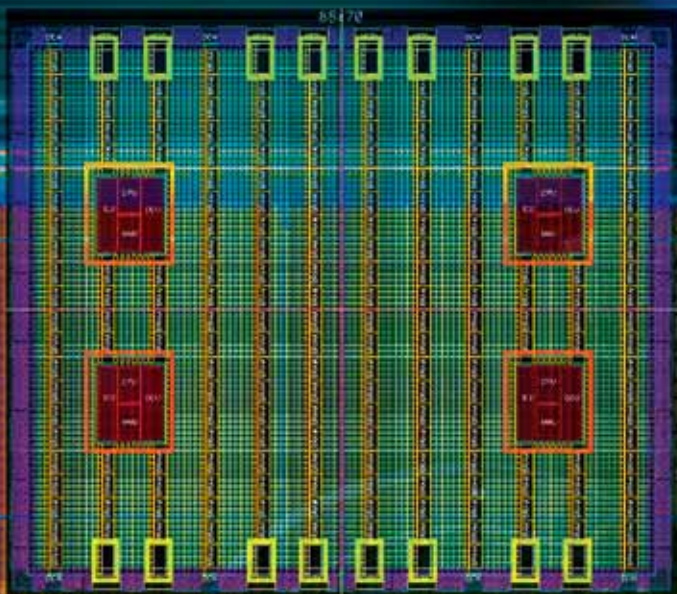
- **Optimizing MicroBlaze Soft Processor Systems**
  - 3 hour class
  - Covers building a complete customized MicroBlaze soft processor system
- **Video/Imaging Algorithms in FPGAs**
  - 3 hour class
  - Verify designs onto actual hardware using Ethernet-based hardware-in-the-loop co-simulation.
- **Introduction 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
  - Covers VHDL language and implementation for FPGAs & CPLDs

# Generating Efficient Board Support Packages

by Rick Molerres  
Manager of Software IP  
Xilinx, Inc.  
[rick.molerres@xilinx.com](mailto:rick.molerres@xilinx.com)

The Xilinx Platform Studio toolset enables quick and easy BSP generation for Virtex FPGAs with immersed PowerPC processors.

Milan Saini  
Technical Marketing Manager  
Xilinx, Inc.  
[milan.saini@xilinx.com](mailto:milan.saini@xilinx.com)



Platform FPGAs with embedded processors offer you unprecedented levels of flexibility, integration, and performance. It is now possible to develop extremely sophisticated and highly customized embedded systems inside a single programmable logic device.

With silicon capabilities advancing, the challenge centers on keeping design methods efficient and productive. In embedded systems development, one of the key activities is the development of the board support package (BSP). The BSP allows an embedded software application to successfully initialize and communicate with the hardware resources connected to the processor. Typical BSP components include boot code, device driver code, and initialization code.

Creating a BSP can be a lengthy and tedious process that must be incurred every time the microprocessor complex (processor plus associated peripherals) changes. With FPGAs, fast design iterations combined with the inherent flexibility of the platform can make the task of managing the BSP even more challenging (Figure 1). This situation clearly underscores the need for and importance of providing an efficient process for managing BSPs.

In this article, we'll describe an innovative solution from Xilinx that simplifies the creation and management of RTOS BSPs. We chose the WindRiver VxWorks flow to illustrate the concept; however, the underlying technology is generic and equally applicable to all other OS solutions that support Xilinx® processors.



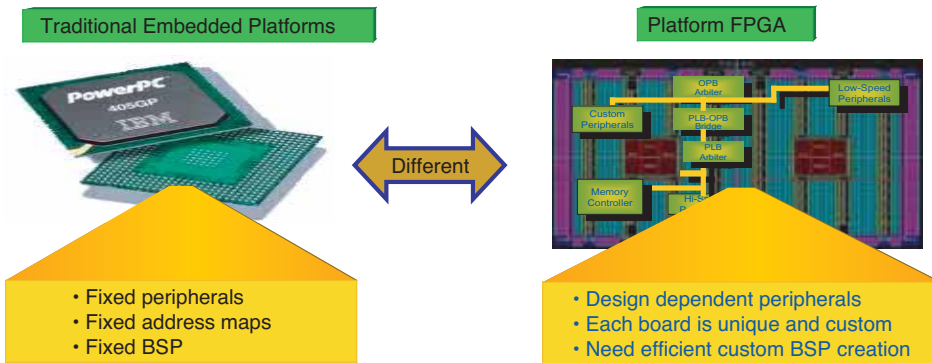


Figure 1 – Platform FPGA flexibility requires the software BSP generation process to be efficient.

### Xilinx Design Flow and Software BSP Generation

Designing for a Xilinx processor involves a hardware platform assembly flow and an embedded software development flow. Both of these flows are managed within the Xilinx Platform Studio (XPS) tool, which is part of the Xilinx Embedded Development Kit (EDK).

You would typically begin a design by assembling and configuring the processor and its connected components in XPS. Once the hardware platform has been defined, you can then configure the software parameters of the system. A key feature of Platform Studio is its ability to produce a BSP that is customized based on your selection and configuration of processor, peripherals, and embedded OS. As the system evolves through iterative changes to the hardware design, the BSP evolves with the platform.

An automatically generated BSP enables embedded system designers to:

- Automatically create a BSP that completely matches the hardware design
- Eliminate BSP design bugs by using pre-certified components
- Increase designer productivity by jump-starting application software development

### Creating BSPs for WindRiver VxWorks

Platform Studio can generate a customized Tornado 2.0.x (VxWorks 5.4) or Tornado 2.2.x (VxWorks 5.5) BSP for the PowerPC™ 405 processor and its periph-

erals in Xilinx Virtex™-II Pro and Virtex-4 FPGAs. The generated BSP contains all of the necessary support software for a system, including boot code, device drivers, and VxWorks initialization.

Once a hardware system with the PowerPC 405 processor is defined in Platform Studio, you need only follow these three steps to generate a BSP for VxWorks:

1. Use the Software Settings dialog box (see Figure 2) to select the OS you plan to use for the system. Platform Studio users can select vxworks5\_4 or vxworks5\_5 as their target operating system.
2. Once you have selected the OS, you can go to the Library/OS Parameters tab, as shown in Figure 3, to tailor the Tornado BSP to the custom hardware. You have the option of selecting any UART device in the system as the standard I/O device (stdin and stdout). This results in the device being used as the VxWorks console device.

You can also choose which peripherals are connected peripherals, or which devices will be tightly integrated into the VxWorks OS. For example, the Xilinx 10/100 Ethernet MAC can be integrated into the VxWorks Enhanced Network Driver (END) interface. Alternately, the Ethernet device need not be connected to the END interface and can instead be accessed directly from the VxWorks application.

3. Generate the Tornado BSP by selecting the Tools > Generate Libraries and BSP menu option. The resulting BSP resembles a traditional Tornado BSP and is located in the Platform Studio project directory under `ppc405_0/bsp_ppc405_0` (see Figure 4).

Note that `ppc405_0` refers to the instance name of the PowerPC 405 processor in the hardware design. Platform Studio users can specify a different instance name, in which case the subdirectory names for the BSP will match the processor instance name.

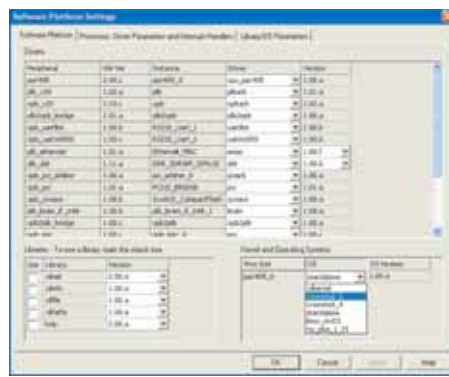


Figure 2 – Setting the embedded OS selection

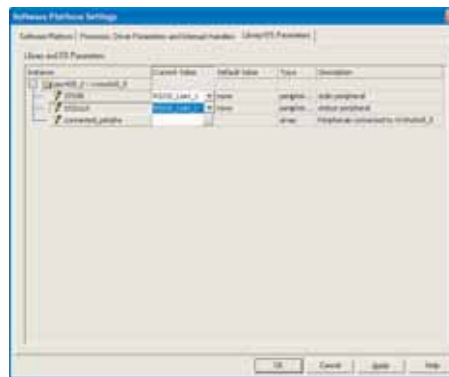


Figure 3 – Configuring the OS-specific parameters

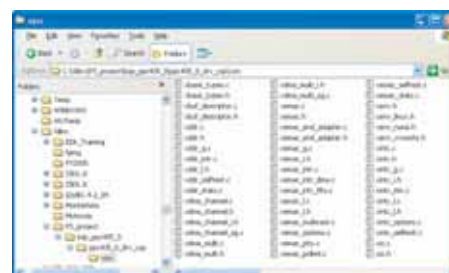


Figure 4 – Generated BSP directory structure



The Tornado BSP is completely self-contained and transportable to other directory locations, such as the standard Tornado installation directory for BSPs at `target/config`.

### Customized BSP Details

The XPS-generated BSP for VxWorks resembles most other Tornado BSPs except for the placement of Xilinx device driver code. Off-the-shelf device driver code distributed with Tornado typically resides in the `target/src/drv` directory in the Tornado distribution directory. Device driver code for a BSP that is automatically generated by Platform Studio resides in the BSP directory itself.

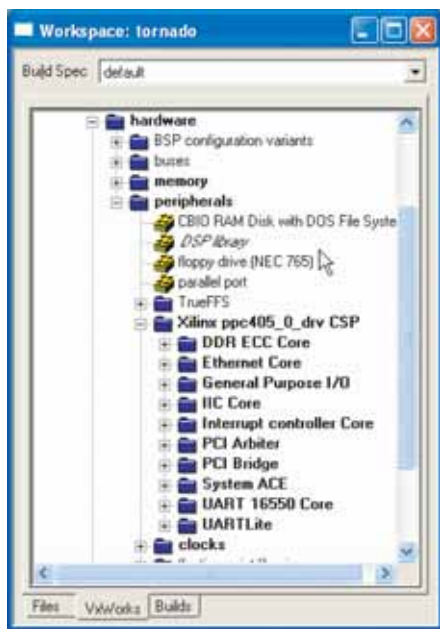


Figure 5 – Tornado 2.x Project: VxWorks tab

This minor deviation is due to the dynamic nature of FPGA-based embedded systems. Because an FPGA-based embedded system can be reprogrammed with new or changed IP, the device driver configuration can change, calling for a more dynamic placement of device driver source files. The directory tree for the automatically generated BSP is shown in Figure 4. The Xilinx device drivers are placed in the `ppc405_0_drv_csp/xsrc` subdirectory of the BSP.

## The Tornado BSP created by Platform Studio has a makefile that you can modify at the command line if you would rather use the `diab` compiler instead of the `gnu` compiler.

Xilinx device drivers are implemented in C and are distributed among several source files, unlike traditional VxWorks drivers, which typically consist of single C header and implementation files. In addition, there is an OS-independent implementation and an optional OS-dependent implementation for device drivers.

The OS-independent part of the driver is designed for use with any OS or any processor. It provides an application program interface (API) that abstracts the func-

named `ppc405_0_drv_<driver_version>.c` in the BSP directory. This file includes the driver source files (\*.c) for the given device and is automatically compiled by the BSP makefile.

This process is analogous to how VxWorks' `sysLib.c` includes source for Wind River-supplied drivers. The reason why Xilinx driver files are not simply included in `sysLib.c` like the rest of the drivers is because of namespace conflicts and maintainability issues. If all Xilinx driver files are part of a single compilation unit, static functions and data are no longer private. This places restrictions on the device drivers and would negate their OS independence.

### Integration with the Tornado IDE

The automatically generated BSP is integrated into the Tornado IDE (Project Facility). The BSP is compilable from the command line using the Tornado make tools or from the Tornado Project. Once the BSP is generated, you can simply type `make vxWorks` from the command line to compile a bootable RAM image. This assumes that the Tornado environment has been previously set up, which you can do through the command line using the `host/x86-win32/bin/torVars.bat` script (on a Windows platform). If you are using the Tornado Project facility, you can create a project based on the newly generated BSP, then use the build environment provided through the IDE to compile the BSP.

In Tornado 2.2.x, the `diab` compiler is supported in addition to the `gnu` compiler. The Tornado BSP created by Platform Studio has a makefile that you can modify at the command line if you would rather use the `diab` compiler instead of the `gnu` compiler. Look for the make variable named `TOOLS` and set the value to “`diab`” instead of “`gnu`.” If using the Tornado Project facility, you can select the desired compiler when the project is first created.

The file `50ppc405_0.cdf` resides in the BSP directory and is tailored during creation of

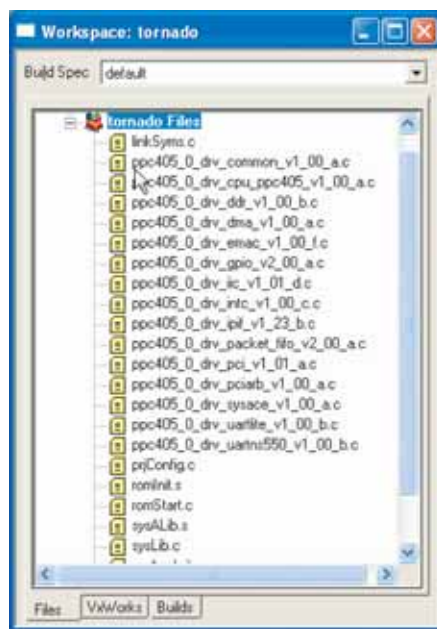


Figure 6 – Tornado 2.x Project: Files tab

tionality of the underlying hardware. The OS-dependent part of the driver adapts the driver for use with an OS such as VxWorks. Examples are Serial IO drivers for serial ports and END drivers for Ethernet controllers. Only drivers that can be tightly integrated into a standard OS interface require an OS-dependent driver.

Xilinx driver source files are included in the build of a VxWorks image in the same way that other BSP files are included in the build. For every driver, a file exists

the BSP. This file integrates the device drivers into the Tornado IDE menu system. The drivers are hooked into the BSP at the Hardware > Peripherals subfolder. Below this are individual device driver folders. Figure 5 shows a menu with Xilinx device drivers.

The Files tab of the Tornado Project Facility will also show the number of files used to integrate the Xilinx device drivers into the Tornado build process. These files are automatically created by Platform Studio and you need only be aware that the files exist. Figure 6 shows an example of the driver build files.


Some of the commonly used devices are tightly integrated with the OS, while other devices are accessible from the application by directly using the device drivers. The device drivers that have been tightly integrated into VxWorks include:

- 10/100 Ethernet MAC
- 10/100 Ethernet Lite MAC
- 1 Gigabit Ethernet MAC
- 16550/16450 UART
- UART Lite
- Interrupt Controller
- System ACE™ technology
- PCI

All other devices and associated device drivers are not tightly integrated into a VxWorks interface; instead, they are loosely integrated. Access to these devices is available by directly accessing the associated device drivers from the user application.

## Conclusion

With the popularity and usage of embedded processor-based FPGAs continuing to grow, tool solutions that effectively synchronize and tie the hardware and software flows together are key to helping designer productivity keep pace with advances in silicon.

Xilinx users have been very positive about Platform Studio and its integration with VxWorks 5.4 and 5.5. Xilinx fully intends to continue its development support for the Wind River flow that will soon include support for VxWorks 6.0 and Workbench IDE. 

## Microprocessor Library Definition (MLD)

The technology that enables dynamic and custom BSP generation is based on a Xilinx proprietary format known as Microprocessor Library Definition (MLD). This format provides third-party vendors with a plug-in interface to Xilinx Platform Studio to enable custom library and OS-specific BSP generation (see Figure 7). The MLD interface is typically written by third-party companies for their specific flows. It enables the following add-on functionality:

- Enables custom design rule checks
- Provides the ability to customize device drivers for the target OS environment
- Provides the ability to custom-produce the BSP in a format and folder structure tailored to the OS tool chain
- Provides the ability to customize an OS/kernel based on the hardware system under consideration

The MLD interface is an ASCII-based open and published standard. Each RTOS flow will have its own set of unique MLD files. An MLD file set comprises the following two files:

- A data definition (.mld) file. This file defines the library or operating system through a set of parameters set by the Platform Studio. The values of these parameters are stored in an internal Platform Studio database and intended for use by the script file during the output generation.
- A .tcl script file. This is the file that is called by XPS to create the custom BSP. The file contains a set of procedures that have access to the complete design database and hence can write a custom output format based on the requirements of the flow.

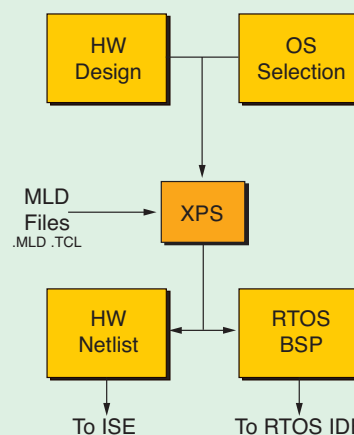


Figure 7 – Structure of an MLD flow

The MLD syntax is described in detail in the EDK documentation (see “Platform Specification Format Reference Manual” at [www.xilinx.com/ise/embedded/psf\\_rm.pdf](http://www.xilinx.com/ise/embedded/psf_rm.pdf)). You can also find MLD examples in the EDK installation directory under [sw/lib/bsp](http://sw/lib/bsp).

Once MLD files for a specific RTOS flow have been created, they need to be installed in a specific path for Xilinx Platform Studio to pick up on its next invocation. The specific RTOS menu selection now becomes active in the XPS dialog box (Project > SW Platform Settings > Software Platform > OS).

Currently, the following partners’ MLD files are available for use within XPS:

- Wind River (VxWorks 5.4, 5.5) (included in Xilinx Platform Studio)
- MontaVista (Linux) (included in Xilinx Platform Studio)
- Mentor Accelerated Technologies (Nucleus) (download from [www.xilinx.com/ise/embedded/mld/](http://www.xilinx.com/ise/embedded/mld/))
- GreenHills Software (Integrity) (download from [www.xilinx.com/ise/embedded/mld/](http://www.xilinx.com/ise/embedded/mld/))
- Micrium (µC/OS-II) (download from [www.xilinx.com/ise/embedded/mld/](http://www.xilinx.com/ise/embedded/mld/))
- µLinux (download from [www.xilinx.com/ise/embedded/mld/](http://www.xilinx.com/ise/embedded/mld/)).

Visit us at  
**Embedded  
Systems CONFERENCE**  
**SILICON VALLEY**

**BOOTH 1208**

**REGISTER TODAY FOR AVNET®  
SPEEDWAY DESIGN WORKSHOP**

“Software Development with  
Xilinx Embedded Processors”

Wed, April 5 and Thurs, April 6 (Room K)

[www.em.avnet.com](http://www.em.avnet.com)

Some things  
**have**  
to go out  
on time



As an embedded software developer, you're always facing the next deadline. We know it's important to get your products to market before your competitors, and we can help. With our Eclipse-based development tools, tightly integrated embedded software and support that is second to none, we offer you a partner to get your product to market quickly and easily.

The EDGE Eclipse-based development environment provides a set of top-notch development tools in the industry today. You'll see how quickly you can code, collaborate on and deliver your final product. Additionally, the Nucleus range of royalty-free RTOS and middleware products gives you a proven kernel with everything else you need in a modern OS. Open, available, affordable solutions.

Finally, our Customer Support has one goal: *provide the most experienced, timely and one-on-one customer support in the industry.* As the only five-time recipient of the Software Technical Assistance Recognition (STAR) Award for technical support excellence and global support center practices certified by the Support Center Practices (SCP), we are dedicated to your success.

For a free evaluation of EDGE Development Tools, visit our website

**[Mentor.com/embedded](http://Mentor.com/embedded)**

or e-mail **[embedded\\_info@mentor.com](mailto:embedded_info@mentor.com)**

Embedded made easy.

**Mentor  
Graphics®**



# Getting You Started With *On-Site Embedded Training*



[www.xilinx.com/epq](http://www.xilinx.com/epq)

With Embedded Processing QuickStart!, Xilinx offers on-site training and support right from the start. Our solution includes a dedicated expert application engineer for one week, to train your hardware team on creating embedded systems, instruct software engineers how to best use supporting FPGA features, and help your team finish on time and on budget.

## **The Quicker Solution to Embedded Design**

QuickStart! features include configuration of the Xilinx ISE™ design environment, an embedded processing and EDK (Embedded Development Kit) training class, and design architecture and implementation consultations with Xilinx experts. Get your team started today!

Contact your Xilinx representative or go to [www.xilinx.com/epq](http://www.xilinx.com/epq) for more information.



# Bringing Floating-Point Math to the Masses

Xilinx makes high-performance floating-point processing available to a wider range of applications.

by Geir Kjosavik  
Senior Staff Product Marketing Engineer,  
Embedded Processing Division  
Xilinx, Inc.  
[geir.kjosavik@xilinx.com](mailto:geir.kjosavik@xilinx.com)

Inside microprocessors, numbers are represented as integers – one or several bytes strung together. A four-byte value comprising 32 bits can hold a relatively large range of numbers:  $2^{32}$ , to be specific. The 32 bits can represent the numbers 0 to 4,294,967,295 or, alternatively, -2,147,483,648 to +2,147,483,647. A 32-bit processor is architected such that basic arithmetic operations on 32-bit integer numbers can be completed in just a few clock cycles, and with some performance overhead a 32-bit CPU can also support operations on 64-bit numbers. The largest value that can be represented by 64 bits is really astronomical: 18,446,744,073,709,551,615. In fact, if a Pentium processor could count 64-bit values at a frequency of 2.4 GHz, it would take it 243 years to count from zero to the maximum 64-bit integer.



**Dynamic Range and Rounding Error Problems**

Considering this, you would think that integers work fine, but that is not always the case. The problem with integers is the lack of dynamic range and rounding errors.

The quantization introduced through a finite resolution in the number format distorts the representation of the signal. However, as long as a signal is utilizing the range of numbers that can be represented by integer numbers, also known as the dynamic range, this distortion may be negligible.

Figure 1 shows what a quantized signal looks like for large and small dynamic swings, respectively. Clearly, with the smaller amplitude, each quantization step is bigger relative to the signal swing and introduces higher distortion or inaccuracy.

The following example illustrates how integer math can mess things up.

**A Calculation Gone Bad**

An electronic motor control measures the velocity of a spinning motor, which typically ranges from 0 to 10,000 RPM. The value is measured using a 32-bit counter. To allow some overflow margin, let's assume that the measurement is scaled so that 15,000 RPM represents the maximum 32-bit value, 4,294,967,296. If the motor is spinning at 105 RPM, this value corresponds to the number 30,064,771 within 0.000033%, which you would think is accurate enough for most practical purposes.

Assume that the motor control is instructed to increase motor velocity by 0.015% of the current value. Because we are operating with integers, multiplying with 1.0015 is out of the question – as is multiplying by 10,015 and dividing by 10,000 – because the intermediate result will cause overflow.

The only option is to divide by integer 10,000 and multiply by integer 10,015. If you do that, you end up with 30,094,064; but the correct answer is 30,109,868.

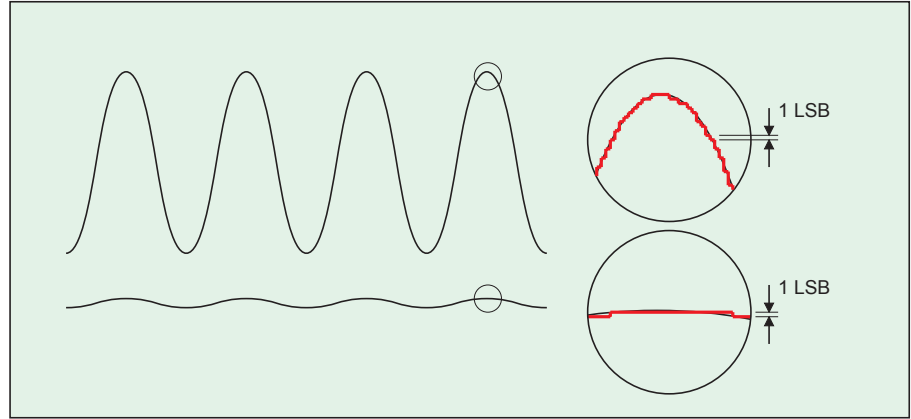


Figure 1 – Signal quantization and dynamic range

Because of the truncation that happens when you divide by 10,000, the resulting velocity increase is 10.6% smaller than what you asked for. Now, an error of 10.6% of 0.015% may not sound like anything to worry about, but as you continue to perform similar adjustments to the motor speed, these errors will almost certainly accumulate to a point where they become a problem.

What you need to overcome this problem is a numeric computer representation that represents small and large numbers with equal precision. That is exactly what floating-point arithmetic does.

**Floating Point to the Rescue**

As you have probably guessed, floating-point arithmetic is important in industrial applications like motor control, but also in a variety of other applications. An increasing number of applications that traditionally have used integer math are turning to floating-point representation. I'll discuss this once we have looked at how floating-point math is performed inside a computer.

**IEEE 754 at a Glance**

A floating-point number representation on a computer uses something similar to a scientific notation with a base and an exponent. A scientific representation of 30,064,771 is  $3.0064771 \times 10^7$ , whereas

1.001 can be written as  $1.001 \times 10^0$ .

In the first example, 3.0064771 is called the mantissa, 10 the exponent base, and 7 the exponent.

IEEE standard 754 specifies a common format for representing floating-point numbers in a computer. Two grades of precision are defined: single precision and double precision. The representations use 32 and 64 bits, respectively. This is shown in Figure 2.

In IEEE 754 floating-point representation, each number comprises three basic components: the sign, the exponent, and the mantissa. To maximize the range of possible numbers, the mantissa is divided into a fraction and leading digit. As I'll explain, the latter is implicit and left out of the representation.

The sign bit simply defines the polarity of the number. A value of zero means that the number is positive, whereas a 1 denotes a negative number.

The exponent represents a range of numbers, positive and negative; thus a bias value must be subtracted from the stored exponent to yield the actual exponent. The single precision bias is 127, and the double precision bias is 1,023. This means that a stored value of 100 indicates a single-precision exponent of -27. The exponent base is always 2, and this implicit value is not stored.

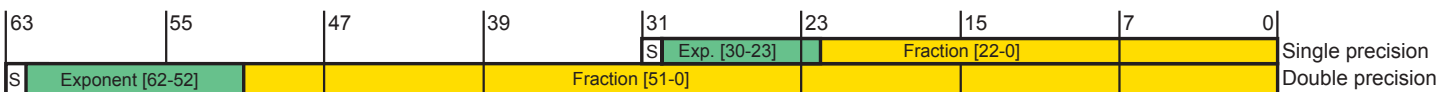


Figure 2 – IEEE floating-point formats



For both representations, exponent representations of all 0s and all 1s are reserved and indicate special numbers:

- Zero: all digits set to 0, sign bit can be either 0 or 1
- $\pm\infty$ : exponent all 1s, fraction all 0s
- Not a Number (NaN): exponent all 1s, non-zero fraction. Two versions of NaN are used to signal the result of invalid operations such as dividing by zero, and indeterminate results such as operations with non-initialized operand(s).

The mantissa represents the number to be multiplied by 2 raised to the power of the exponent. Numbers are always normalized; that is, represented with one non-zero leading digit in front of the radix point. In binary math, there is only one non-zero number, 1. Thus the leading digit is always 1, allowing us to leave it out and use all the mantissa bits to represent the fraction (the decimals).

Following the previous number examples, here is what the single precision representation of the decimal value 30,064,771 will look like:

The binary integer representation of 30,064,771 is 1 1100 1010 1100 0000 1000 0011. This can be written as  $1.110010101100000010000011 \times 2^{24}$ . The leading digit is omitted, and the fraction – the string of digits following the radix point – is 1100 1010 1100 0000 1000 0011. The sign is positive and the exponent is 24 decimal. Adding the bias of 127 and converting to binary yields an IEEE 754 exponent of 1001 0111.

Putting all of the pieces together, the single representation for 30,064,771 is shown in Figure 3.

### Gain Some, Lose Some

Notice that you lose the least significant bit (LSB) of value 1 from the 32-bit integer representation – this is because of the limited precision for this format.

The range of numbers that can be rep-

resented with single precision IEEE 754 representation is  $\pm(2-2^{-23}) \times 2^{127}$ , or approximately  $\pm 10^{38.53}$ . This range is astronomical compared to the maximum range of 32-bit integer numbers, which by comparison is limited to around  $\pm 2.15 \times 10^9$ . Also, whereas the integer representation cannot represent values between 0 and 1, single-precision floating-point can represent values down to  $\pm 2^{-149}$ , or  $\pm 10^{-44.85}$ . And we are still using only 32 bits – so this has to be a much more convenient way to represent numbers, right?

The answer depends on the requirements.

- Yes, because in our example of multiplying 30,064,771 by 1.001, we can simply multiply the two numbers and the result will be extremely accurate.
- No, because as in the preceding example the number 30,064,771 is not represented with full precision. In fact, 30,064,771 and 30,064,770 are represented by the exact same 32-bit bit pattern, meaning that a software algorithm will treat the numbers as identical. Worse yet, if you increment either number by 1 a billion times, none of them will change. By using 64 bits and representing the numbers in double precision format, that particular example could be made to work, but even double-precision representation will face the same limitations once the numbers get big – or small enough.
- No, because most embedded processor cores ALUs (arithmetic logic units) only support integer operations, which leaves floating-point operations to be emulated in software. This severely affects processor performance. A 32-bit CPU can add two 32-bit integers with one machine code instruction; however, a library routine including bit manipulations and multiple arithmetic operations is needed to add two IEEE single-precision floating-point values.

With multiplication and division, the performance gap just increases; thus for many applications, software floating-point emulation is not practical.

### Floating Point Co-Processor Units

For those who remember PCs based on the Intel 8086 or 8088 processor, they came with the option of adding a floating-point coprocessor unit (FPU), the 8087. Though a compiler switch, you could tell the compiler that an 8087 was present in the system. Whenever the 8086 encountered a floating-point operation, the 8087 would take over, do the operation in hardware, and present the result on the bus.

Hardware FPUs are complex logic circuits, and in the 1980s the cost of the additional circuitry was significant; thus Intel decided that only those who needed floating-point performance would have to pay for it. The FPU was kept as an optional discrete solution until the introduction of the 80486, which came in two versions, one with and one without an FPU. With the Pentium family, the FPU was offered as a standard feature.

### Floating Point is Gaining Ground

These days, applications using 32-bit embedded processors with far less processing power than a Pentium also require floating-point math. Our initial example of motor control is one of many – other applications that benefit from FPUs are industrial process control, automotive control, navigation, image processing, CAD tools, and 3D computer graphics, including games.

As floating-point capability becomes more affordable and proliferated, applications that traditionally have used integer math turn to floating-point representation. Examples of the latter include high-end audio and image processing. The latest version of Adobe Photoshop, for example, supports image formats where each color channel is represented by a floating-point number rather than the usual integer representation. The increased dynamic range fixes some problems inherent in integer-based digital imaging.

1100 1011 1110 0101 0110 0000 0100 0001

Figure 3 – 30,064,771 represented in IEEE 754 single-precision format

Operation	CPU Cycles without FPU	CPU Cycles with CPU	Acceleration
Addition	400	6	67x
Subtraction	400	6	67x
Division	750	30	25x
Multiplication	400	6	67x
Comparison	450	3	150x

Table 1 – MicroBlaze floating-point acceleration

If you have ever taken a picture of a person against a bright blue sky, you know that without a powerful flash you are left with two choices; a silhouette of the person against a blue sky or a detailed face against a washed-out white sky. A floating-point image format partly solves this problem, as it makes it possible to represent subtle nuances in a picture with a wide range in brightness.

Compared to software emulation, FPUs can speed up floating-point math operations by a factor of 20 to 100 (depending on type of operation) but the availability of embedded processors with on-chip FPUs is limited. Although this feature is becoming increasingly more common at the higher end of the performance spectrum, these derivatives often come with an extensive selection of advanced peripherals and very high-performance processor cores – features and performance that you have to pay for even if you only need the floating-point math capability.

### FPUs on Embedded Processors

With the MicroBlaze™ 4.00 processor, Xilinx makes an optional single precision FPU available. You now have the choice whether to spend some extra logic to achieve real floating-point performance or to do traditional software emulation and free up some logic (20-30% of a typical processor system) for other functions.

### Why Integrated FPU is the Way to Go

A soft processor without hardware support for floating-point math can be connected to an external FPU implemented on an

FPGA. Similarly, any microcontroller can be connected to an external FPU. However, unless you take special considerations on the compiler side, you cannot expect seamless cooperation between the two.

C-compilers for CPU architecture families that have no floating-point capability will always emulate floating-point operations in software by linking in the necessary library routines. If you were to connect an FPU to the processor bus, FPU access would occur through specifically designed driver routines such as this one:

```
void user_fmud(float *op1, float *op2, float *res)
{
    FPU_operand1=*op1;    /* write operand a to FPU */
    FPU_operand2=*op2;    /* write operand b to FPU */
    FPU_operation=MUL;    /* tell FPU to multiply */
    while (!(FPU_stat & FPUready)); /* wait for FPU to finish */
    *res = FPU_result      /* return result */
}
```

To do the operation,  $z = x*y$  in the main program, you would have to call the above driver function as:

```
float x, y, z;
user_fmud (&x, &y, &z);
```

For small and simple operations, this may work reasonably well, but for complex operations involving multiple additions, subtractions, divisions, and multiplications, such as a proportional integral derivative (PID) algorithm, this approach has three major drawbacks:

- The code will be hard to write, maintain, and debug

- The overhead in function calls will severely decrease performance
- Each operation involves at least five bus transactions; as the bus is likely to be shared with other resources, this not only affects performance, but the time needed to perform an operation will be dependent on the bus load in the moment

### The MicroBlaze Way

The optional MicroBlaze soft processor with FPU is a fully integrated solution that offers high performance, deterministic timing, and ease of use. The FPU operation is completely transparent to the user.

When you build a system with an FPU, the development tools automatically equip the CPU core with a set of floating-point assembly instructions known to the compiler.

To perform  $y = x*y$ , you would simply write:

```
float x, y, z;
y = x * z;
```

and the compiler will use those special instructions to invoke the FPU and perform the operation.

Not only is this simpler, but a hardware-connected FPU guarantees a constant number of CPU cycles for each floating-point operation. Finally, the FPU provides an extreme performance boost. Every basic floating-point operation is accelerated by a factor 25 to 150, as shown in Table 1.

### Conclusion

Floating-point arithmetic is necessary to meet precision and performance requirements for an increasing number of applications.

Today, most 32-bit embedded processors that offer this functionality are derivatives at the higher end of the price range.

The MicroBlaze soft processor with FPU can be a cost-effective alternative to ASSP products, and results show that with the correct implementation you can benefit not only from ease-of-use but vast improvements in performance as well.

For more information on the MicroBlaze FPU, visit [www.xilinx.com/ipcenter/processor\\_central/microblaze/microblaze\\_fpu.htm](http://www.xilinx.com/ipcenter/processor_central/microblaze/microblaze_fpu.htm).



# Packet Subsystem on a Chip

Teja's packet-engine technology integrates all of the key aspects of a flexible packet processor.

by Bryon Moyer  
VP, Product Marketing  
Teja Technologies, Inc.  
[bmoyer@teja.com](mailto:bmoyer@teja.com)

As the world gets connected, more and more systems rely on access to the network as a standard part of product configuration. Traditional circuit-based communications systems like the telephone infrastructure are gradually moving towards packet-based technology. Even technologies like Asynchronous Transfer Mode (ATM) are starting to yield to the Internet Protocol (IP) in places. All of this has dramatically increased the need for packet-processing technology.

The content being passed over this infrastructure has increased the demands on available bandwidth. Core routers target 10 Gbps; edge and access equipment work in the 1-5 Gbps range. Even some end-user equipment is starting to break the 100 Mbps range. The question is how to design systems to accommodate these speeds.

These systems implement a wide variety of network protocols. Because the protocols start out as software, it's easiest for network designers if as much of the functionality as possible can remain in software. So the further software programmability can be pushed up the speed range, the better. Although FPGAs can handle network speeds as high as 10 Gbps, RTL has typically been required for 1 Gbps and higher.



# Most traffic that goes through the system looks alike, and processors can be optimized for that kind of traffic.

Teja Technologies specializes in packet-processing technologies implemented in high-level software on multi-core environments. Teja has adapted its technology to Xilinx® Virtex™-4 FPGAs, allowing high-level software programmability of a packet-processing engine built out of multiple MicroBlaze™ soft-processor cores. This combination of high-level packet technology and Xilinx silicon and core technology – using Virtex-4 devices with on-board MACs, PHYs, PowerPC™ hard-core processors, and ample memory – provides a complete packet-processing subsystem that can process more than 1 Gbps in network traffic.

## The Typical Packet Subsystem

The network “stack” shown in Figure 1 is typically divided between the “control plane” and the “data plane.” All of the packets are handled in the data plane; the control plane makes decisions on how the packets should be processed. The lowest layer sees every packet; higher layers will see fewer packets.

The control plane comprises a huge amount of sophisticated software. The data-plane software is simpler, but must operate at very high speed at the lowest layers because it has such a high volume of packets. Packet-processing acceleration usually focuses on layers one to three of the network stack, and sometimes layer four.

Most traffic that goes through the system looks alike, and processors can be optimized for that kind of traffic. For this reason, data-plane systems are often divided into the “fast path,” which handles average traffic, and the “slow path,” which handles exceptions. Although the slow path can be managed by a standard RISC processor like a PowerPC, the fast path usually uses a dedicated structure like a network processor or an ASIC. The focus of the fast path is typically IP, ATM, VLAN, and similar protocols in layers two and three. Layer four protocols like TCP and UDP are also often accelerated.

Of course, to process packets, there must be a way to deliver the packets to and from the fast-path processor. Coming off an Ethernet port, the packets must first traverse the physical layer logic (layer one of the stack, often a dedicated chip) and then the MAC (part of layer two, also often its own dedicated chip).

One of the most critical elements in getting performance is the memory.

Memory is required for packet storage, table storage, and for program and data storage for both the fast and slow paths. Memory latency has a dramatic impact on speed, so careful construction of the memory architecture is paramount.

Finally, there must be a way for the control plane to access the subsystem. This is important for initialization, making table changes, diagnostics, and other control functions. Such access is typically accomplished through a combination of serial connections and dedicated Ethernet connections, each requiring logic to implement.

A diagram of this subsystem is shown in Figure 2; all of the pieces of this subsystem are critical to achieving the highest performance.

## The Teja Packet Pipeline

One effective way to accelerate processing is to use a multi-core pipeline. This allows you to divide the functionality into stages and add parallel elements as needed to hit performance. If you were to try to assemble such a structure manually, you would immediately

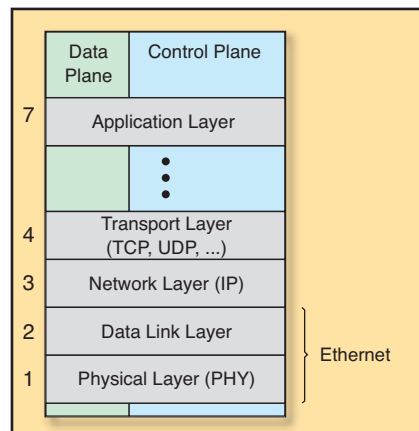


Figure 1 – The network protocol stack

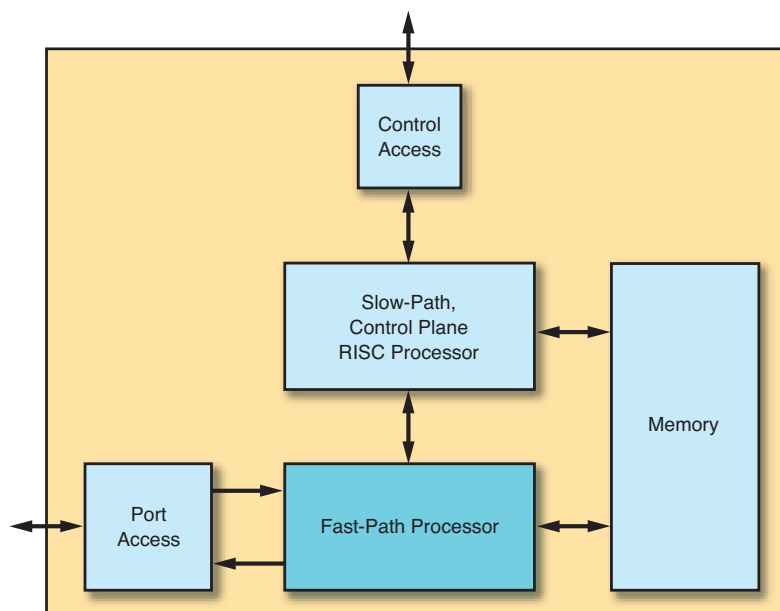


Figure 2 – Typical packet-processing subsystem

ly encounter the kinds of challenges faced by experienced multi-core designers: how to structure communication between stages, scheduling, and shared resource access.

Teja has developed a pipeline structure by creating its own blocks that implement the

Access to the pipeline is provided by a block that takes each packet and delivers the critical parts to the pipeline. Because this block is in the critical path for every packet, it must be very fast, and has been designed by Teja for very high performance.

you can add more parallel processing, or create another pipeline stage. The reverse is also true: if a given pipeline provides more performance than the target system requires, you can remove engines, making the subsystem more economical.

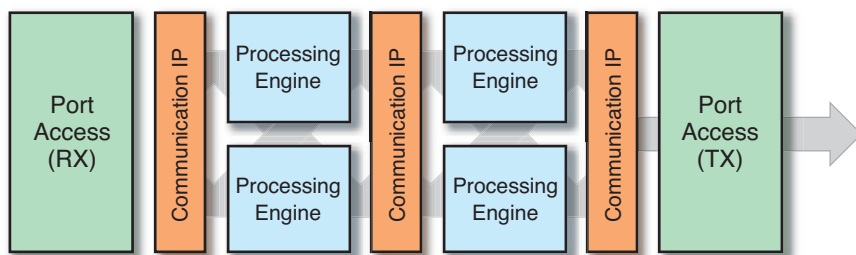


Figure 3 – Teja packet-processing pipeline

necessary functions for efficient processing and inter-communication. By taking advantage of this existing infrastructure, you can assemble pipelines easily in a scalable fashion.

The pipeline comprises processing engines connected by communication blocks and accessed through packet access blocks. Figure 3 illustrates this arrangement.

The engine consists primarily of a MicroBlaze processor and some private block RAM on the FPGA. In addition, if a stage has a particularly compute-intensive function like a checksum, or a longer-lead function like an external memory read or write, an offload can be included to accelerate that function. Because the offload can be created as asynchronous if desired, the MicroBlaze processor is free to work on something else while the offload is operating.

The communication blocks manage the transition from stage to stage. As packet information moves forward, the communication block can perform load balancing or route a packet to a particular engine. Although the direction of progress is usually “forward” (left to right, as shown in Figure 3), there are times when a packet must move backwards. An example of this is with IPv4/v6 forwarding, when an IPv6 packet is tunneled on IPv4. Once the IPv4 packet is decapsulated, an internal IPv6 packet is found, and it must go back for IPv6 decapsulation.

The result of this structure is that each MicroBlaze processor and offload can be working on a different packet at any given time. High performance is achieved because many in-flight packets are being handled at once.

The key to this structure is its scalability. Anytime additional performance is needed,

### The Rest of the Subsystem

What is so powerful about the combination of Teja’s data-plane engine and the Virtex-4 FX devices is that most of the rest of the subsystem can be moved on-chip. Much of the external memory can now be moved into internal block RAM. Some external memory will still be required, but high-speed DRAM can be directly accessed by the Virtex-4 family, so no intervening glue is required. The chips have built-in Ethernet MACs which, combined with the available PHY IP and RocketIO™ technology, allow direct access from Ethernet ports onto the chip.

The integrated PowerPC cores (as many as two) allow you to implement the slow path and even the entire control plane on the same chip over an embedded operating system such as Linux. You can also provide

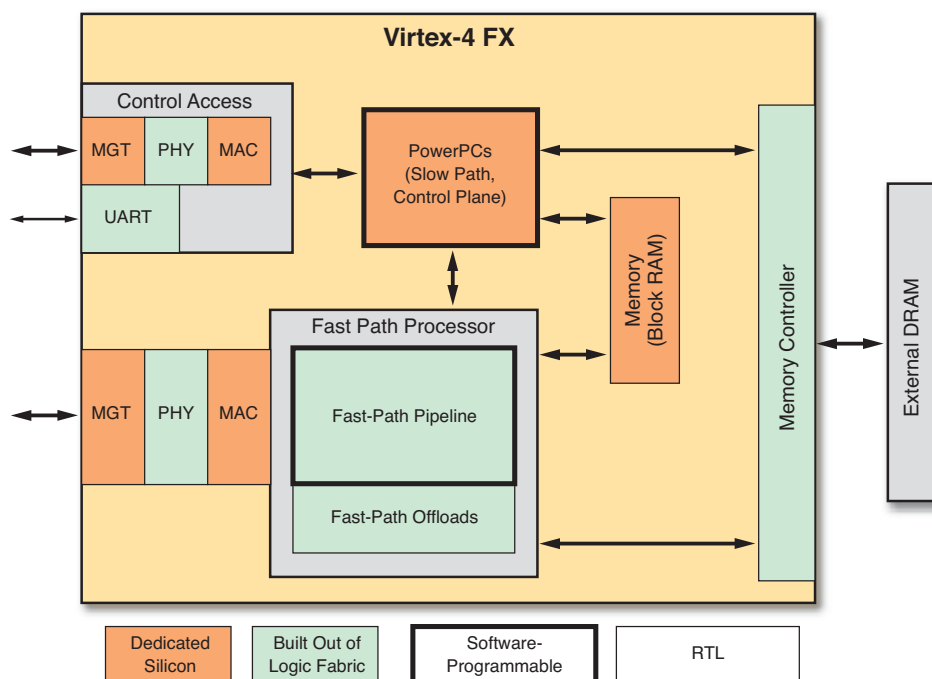


Figure 4 – Single-chip packet-processing subsystem

control access through serial and Ethernet ports using existing IP.

As a result, the entire subsystem shown in Figure 2 (with the exception of some external memory) can be implemented on a single chip, as illustrated in Figure 4.

### Flexibility: Customizing, Resizing, Upgrading

Teja's packet-processing infrastructure provides access to our company's real strength: providing data-plane applications that you can customize. We deliver applications such as packet forwarding, TCP, secure gateways, and others with source code. The reason for delivering source code is that if

One of the most important aspects of software programmability is field upgrades. With a software upgrade, you can change your code – as long as you stay within the amount of code store available. As the Teja FPGA packet engine is software-programmable, you can perform software upgrades. But because it uses an FPGA, you can also upgrade the underlying hardware in the field. For example, if a software upgrade requires more code store than is available, you can make a hardware change to make more code store available, and then the software upgrade will be successful. Only an FPGA provides this flexibility.

now IPv4 still dominates. At its most basic, IPv4 comprises the following functions:

- Filtering
- Decapsulation
- Classification
- Validation
- Broadcast check
- Lookup/Next Hop calculation
- Encapsulation

Teja has implemented these in a two-stage pipeline, as shown in Figure 5. Offloads are used for the following functions:

- Checksum calculation
- Hash lookup
- Longest-prefix match
- Memory access

This arrangement provides full gigabit line-rate processing of a continuous stream of 64-byte packets, which is the most stringent Ethernet load.

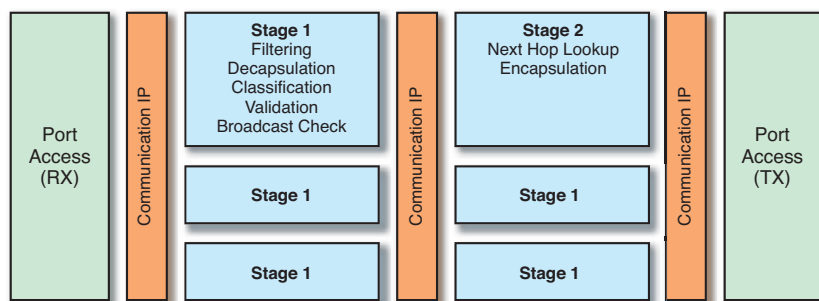


Figure 5 – IPv4 forwarding engine

you need to customize the operation of the application, you can alter the delivered application using straight ANSI C. Even though you are using an FPGA, it is still software-programmable, and you can design using standard software methods.

An application as delivered by Teja is guaranteed to operate at a given line rate. When you modify that application, however, the performance may change. Teja's scalable infrastructure allows you to tailor the processor architecture to accommodate the performance requirements in light of changed functionality.

In a non-FPGA implementation, if you cannot meet performance, then you typically have to go to a much larger device, which will most likely be under-utilized (but cost full price). The beauty of FPGA implementation is that the pipeline can be tweaked to be just the right configuration, and only the amount of hardware required is used. The rest is available for other functions.

Because a structure like this is typically designed by high-level system designers and architects, it is important that ANSI C is the primary language. At the lowest level, the hardware infrastructure, the mappings between software and hardware, and the software programs themselves are expressed in C. Teja has created an extensive set of APIs that allow both compile-time and real-time access from the software to the various hardware resources. Additional tools will simplify the task of implementing programs on the pipeline.

### IPv4 Forwarding Provides Proof

Teja provides IPv4 and IPv6 forwarding as a complete data-plane application. IPv4 is a relatively simple application that can illustrate the power of this packet engine. It is the workhorse application under most of the Internet today. IPv6 is gradually gaining some ground, with its promise of plenty of IP addresses for the future, but for

### Conclusion

Teja Technologies has adapted its packet-processing technology to the Virtex-4 FX family, creating an infrastructure of IP blocks and APIs that take advantage of Virtex-4 FX features. The high-level customizable applications that Teja offers can be implemented using software methodologies on a MicroBlaze multi-core fabric while achieving speeds higher than a gigabit per second. Software programmability adds to the flexibility and ease of design already inherent in the Virtex family.

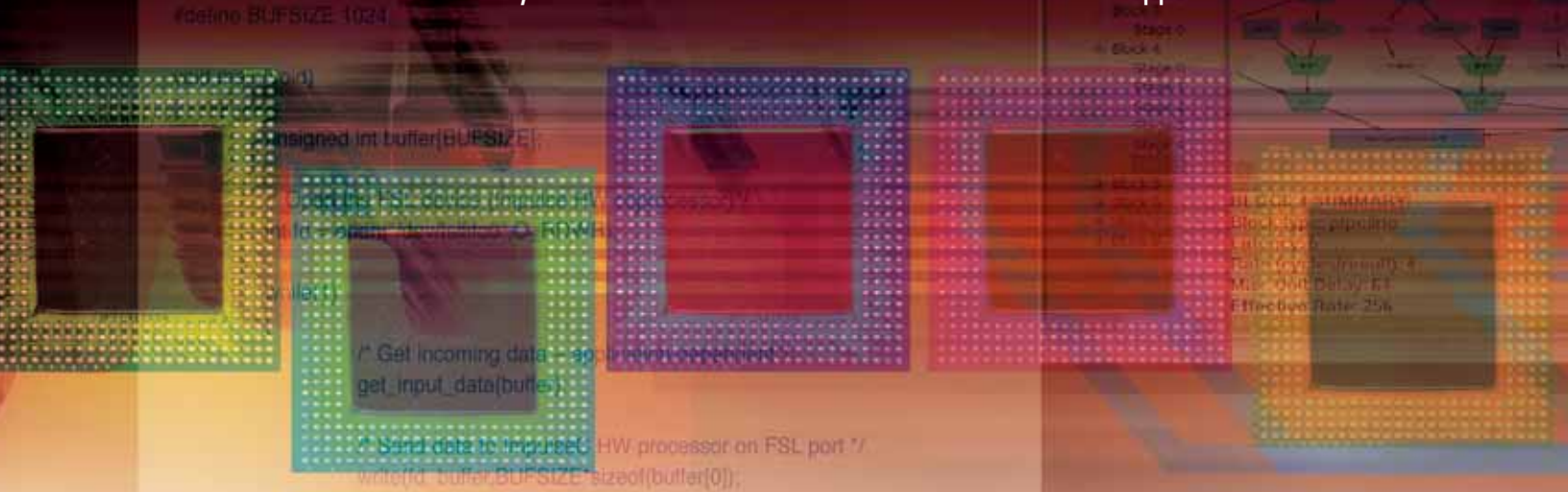
The flexibility of the high-level source code algorithms is bolstered by the fact that the underlying hardware utilization can be specifically tuned to the performance requirements of the system. And once deployed, both software and hardware upgrades are possible, dramatically extending the potential life of the system in the field.

Teja Technologies, the Virtex-4 FX family, and the MicroBlaze core provide a single-chip customizable, resizable, and upgradable packet-processing solution. ●●●



# 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](mailto:john.williams@petalogix.com)

Scott Thibault, Ph.D.  
President  
Green Mountain Computing Systems, Inc.  
[thibault@gmvhdl.com](mailto:thibault@gmvhdl.com)

David Pellerin  
CTO  
Impulse Accelerated Technologies, Inc.  
[david.pellerin@impulsec.com](mailto: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

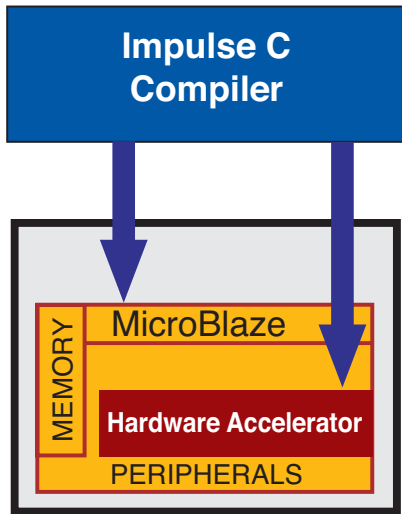


Figure 1 – Impulse C custom hardware accelerators run in the FPGA fabric to accelerate  $\mu$ Clinix processor-based applications.

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.

### 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

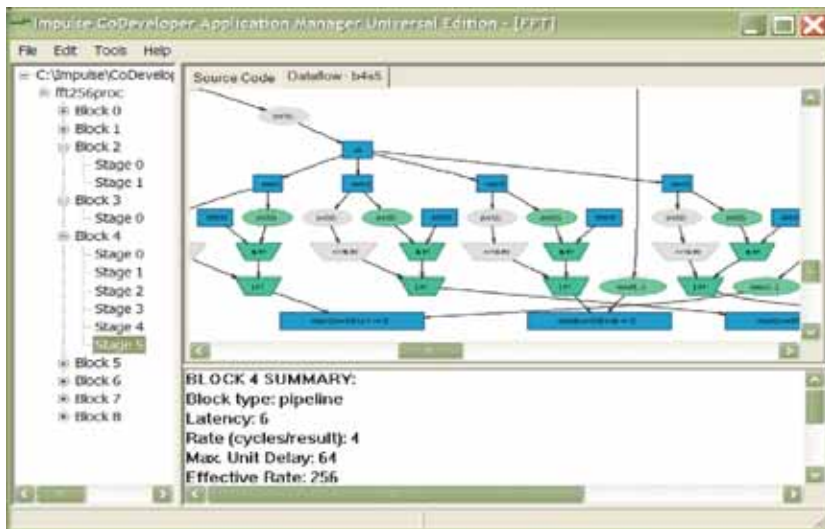


Figure 2 – A dataflow graph allows C programmers to analyze the generated hardware and perform explorative optimizations to balance tradeoffs between size and speed. Illustrated in this graph is the final stage of a six-stage pipelined loop. This graph also helps C programmers understand how sequential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

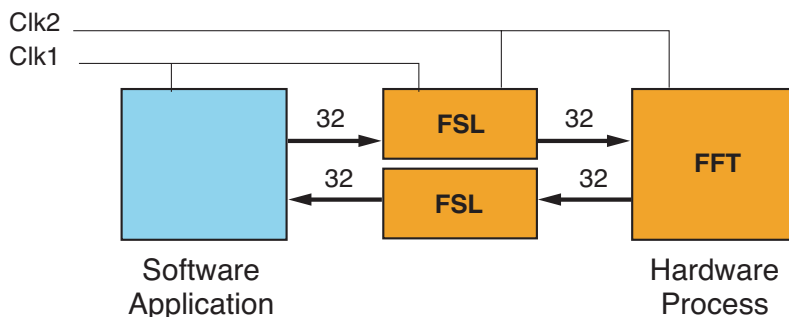


Figure 3 – The FFT includes a 32-bit stream input, a 32-bit stream output, and two clocks, allowing the FFT to be clocked at a different rate than the embedded processor.

## 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 ( $\mu$ Clinux) application running on the Xilinx MicroBlaze™ soft-processor core. MicroBlaze  $\mu$ Clinux 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  $\mu$ Clinux is required. Advantages of embedded Linux include a familiar development environment (appli-

cations 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  $\mu$ Clinux (pronounced “you-see-Linux”) operating system is a port of the open-source Linux version 2.4. The  $\mu$ Clinux kernel is a compact operating system appropriate for a wide variety of 32-bit, non-memory management unit (MMU) processor cores.  $\mu$ Clinux 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  $\mu$ Clinux is straightforward; the Impulse tools include support for  $\mu$ Clinux 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/fslfifo",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  $\mu$ Clinux applications and ImpulseC hardware using the generic FSL FIFO device driver



driver in the MicroBlaze  $\mu$ Clinux 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).

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](http://www.petalogix.com) and [www.impulsec.com](http://www.impulsec.com).

```
/* 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

# Eliminating Data Corruption in Embedded Systems

SiliconSystems' patented PowerArmor technology eliminates unscheduled system downtime caused by power disturbances.

by Gary Drossel  
Director of Product Marketing  
SiliconSystems, Inc.  
[gdrossel@siliconsystems.com](mailto:gdrossel@siliconsystems.com)

Embedded systems often operate in less than ideal power conditions. Power disturbances ranging from spikes to brown-outs can cause a significant amount of data and storage system corruption, causing field failures and potential loss of revenue from equipment returns. You must consider how your storage solution will operate in environments with varying power input stability. If the host system loses power in the middle of a write operation, critical data may be overwritten or sector errors may result, causing the system to fail.

## Data Corruption

The host system reads and writes data in minimum 512-byte increments called sectors. Data corruption can occur when the system loses power during the sector write operation, either because the system did not have time to finish or because the data was not written to the proper location.

In the first scenario, the data in the sector does not match the sector's error-checking information. A read sector error will occur the next time the host system attempts to read that sector. Many applications that encounter such an error will automatically produce a system-level error that will result in system downtime until the error is corrected.

In addition to losing the data, the solid-state drive may thereafter interpret the read sector error as defective, and may unnecessarily replace it with a spare. In general, about 1% of the storage capacity is reserved for spares. Once the factory-defined spares are depleted, the drive will fail and must be replaced.

In another potential problem, a brown-out or low-power condition could cause the address lines of the storage device to become unstable. If this happens – but there is still enough power to program the non-volatile storage component – data could be written to the wrong address. This again could result in a critical failure, requiring drive replacement.

These types of errors can be frustrating for you to find. At the point the drive fails, it is sent back to the factory. The factory then validates the error during the failure analysis process. However, for the vendor to determine whether or not it is a component failure, they must re-initialize the drive back to factory settings. After re-initialization, the drive operates normally and is sent back. This is the typical return scenario that flags a power issue.

The critical point to understand is that the drive is not physically damaged, nor is it “worn out.” In both cases, the data will be lost and downtime will occur, but the drive can be used again.

### Solid-State Drives versus Flash Cards

Not all solid-state storage products are created equal. Even though they physically look the same, solid-state drives in a CompactFlash (CF) form factor differ greatly from CompactFlash cards designed for the consumer electronics market. The inherent technology differences are outlined in Table 1.

The term “standard CompactFlash card” can be a bit misleading. Flash cards must pass the test suite defined by the CompactFlash Association (CFA). These tests allow for a relatively wide range in some timing parameters, which can vary with both hardware and firmware changes. Most consumer-oriented products have enough other system overhead to render these changes insignificant.

Embedded systems, on the other hand, usually have very strict timing requirements. Even the slightest changes to the host interface can cause significant issues.

The most industry-recognized differentiator between solid-state storage and commercial flash cards is in write/erase cycle endurance. Solid-state drives have superior error-correction capabilities and wear leveling algorithms that can significantly extend the life of standard storage components. The power issues I’ve described are significantly less well-known but are much more prevalent in embedded applications. Approximately 75% of field failures are the result of power-related corruption.

Solid-state drives specifically designed for

embedded systems can integrate various techniques for mitigating these power-related issues. These techniques are not economically viable for consumer-based applications, but are essential to eliminate unscheduled downtime in embedded systems.

SiliconSystems patented its PowerArmor technology to eliminate drive corruption caused by power disturbances. Figure 1 shows how PowerArmor integrates voltage-detection circuitry to provide an early warning of a possible power anomaly. Once a voltage threshold has been reached, the SiliconDrive sends a busy signal to the host so that no more commands are received until the power level stabilizes.

Next, address lines are latched (as shown

Parameter	SiliconSystems SiliconDrive CF	CompactFlash Card
Write/Erase Endurance	>2 M Cycles per Block	<100 K Cycles per Card
Error Correction (ECC)	6 bit	1-2 bit
Wear Leveling Algorithm	Over the Entire SiliconDrive	Over Free Space Only
Power-Down Protection	Yes	No
R/W Speed (MBps) Single Sector Large File	1-2 MBps 6-8 MBps	30-40 KBps 3-5 MBps
Re-Qualification Cycle	3-5 Years	1 Year
Maximum Capacity	8 GB	4 GB

Table 1 – Key performance metrics distinguishing solid-state drives designed for embedded systems from flash cards used in consumer electronic devices

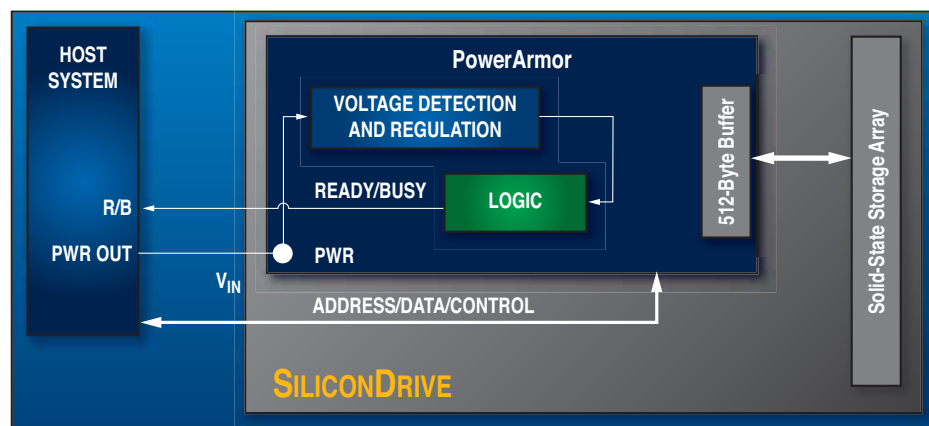


Figure 1 – SiliconDrive with integrated PowerArmor technology



## Solid-state storage will continue to replace rotating hard disk drives in embedded systems as the cost per usable gigabyte (that is, the cost for the capacity required by the application) continues its rapid decline.

in Figure 2) to ensure that data is written to the proper location. In contrast, most microcontroller-based flash cards' address lines can float to undetermined states if input power drops below the specified minimum operating level.

SiliconDrive technology integrates a RISC-based DSP that allows for a 512-byte

The ability to handle power problems has an impact on overall reliability, customer goodwill, and total cost of ownership.

You must consider many factors when developing a test system that characterizes the effects of power disturbances on the storage device. First, the power-down ramp rate on power and all I/O pins must obvi-

time – yet the storage device may continue to operate. The worst-case scenario is that the first read sector error occurs in a system file or critical data area. You may choose to run the test until it encounters the first read sector error or until the drive becomes inoperable.

SiliconSystems has designed a general-purpose power-down tester based on the Xilinx® ML401 development platform. This system, along with available source code to facilitate multi-threaded operation, allows you to quickly qualify and validate various storage media based on individual application needs.

### Conclusion

Solid-state storage will continue to replace rotating hard disk drives in embedded systems as the cost per usable gigabyte (that is, the cost for the capacity required by the application) continues its rapid decline. The immense success of Apple's iPod nano, which uses solid-state storage instead of hard drives, is currently the most visible example of things to come.

This trend will, however, come with a price. The engineering tradeoffs between storage component reliability – lower write/erase cycle endurance and higher bit-error rates – and cost will accelerate the need for PowerArmor, enhanced error correction, and wear leveling capabilities. In addition, new technologies like SiliconSystems' SiSMART, which monitors and predicts the SiliconDrive's remaining usable life span, will be required for embedded systems with multi-year deployments. This will drive an even larger differentiation between solid-state drives designed specifically for embedded systems and flash cards designed for consumer electronic applications.

For more information, visit [www.siliconsystems.com](http://www.siliconsystems.com).

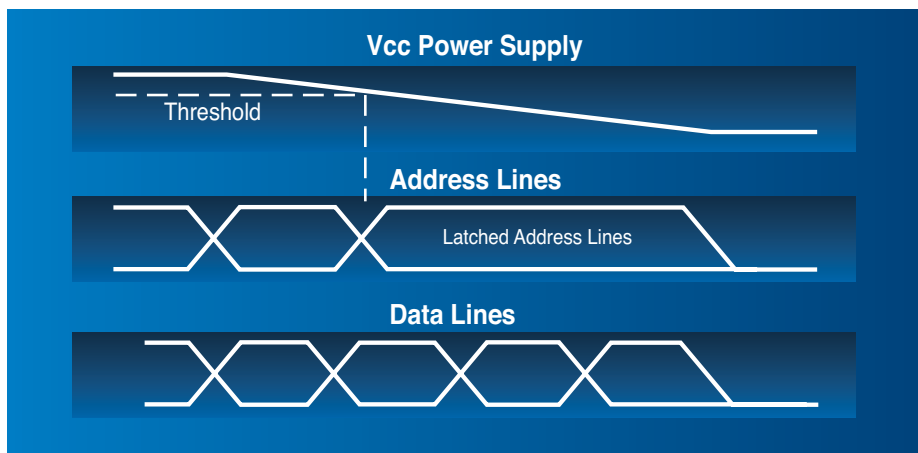


Figure 2 – Latching the address lines when Vcc reaches its lower threshold will ensure data is written to the proper location.

buffer size. This is 1/4 to 1/8 the size of a microcontroller-based flash card. The smaller buffer size limits the time that data is in volatile memory, thereby reducing the power and time required to clear the buffer, update the control bytes, and complete the data transaction.

Larger buffer sizes require additional capacitance to hold the power up longer. This extra capacitance is physically very large and will generally not scale to form factors smaller than 2.5- or 3.5-inch drives. SiliconDrive technology does not require this added capacitance and can therefore scale to virtually any industry-standard form factor.

### Testing for Power Issues

Many companies have included power-down testing in the qualification process for any new storage media they are considering.

ously be fast enough to emulate the loss of system power. In the case of implementing a CF or PC card form factor, the ramp rate must also be able to emulate the steeper ramp associated with pulling out the product during a write cycle.

A robust, repeatable power-down test requires power disturbances at various intervals of the write cycle. The ability to strictly control and repeat this variable will expose any weaknesses in the write cycle and accelerate any failures. It is also extremely important to cut power to I/O and Vcc at the same time. Many test platforms may only cut power to Vcc and leave the I/O powered. This will result in invalid test data and potential damage to the device under test.

The amount of verification performed will directly impact the test time. Testing after every cycle could be prohibitive, and read sector errors may occur at any point in

# A "little" Mistake That Could Cost You \$100,000 Per Year



A "little" mistake is in thinking that all solid-state storage is engineered equal.

Not even close.

SiliconSystems SiliconDrive is advanced storage technology engineered to overcome the problems associated with hard drives and flash cards.

What does this mean to you?

For starters, thousands of dollars in costs savings from eliminating unscheduled downtime resulting from field failures, product wear-out or forced product re-qualifications.

SiliconDrive is certified for use with the Xilinx® System ACE™ Development Environment.

SiliconDrive Highlights	
PowerArmor (eliminates drive corruption)	✓
SiSMART (forecasts useful life)	✓
6-Bit ECC	✓
Wear-Leveling Over Entire Drive	✓
Eliminates Bit-Flip Errors	✓
MTBF > 4,000,000 Hours	✓
Low Power Consumption	✓
Scalable Technology	✓
Enhanced Data Security	✓



To learn more about SiliconDrive and its many advantages, visit us at [www.siliconsystems.com/xilinx](http://www.siliconsystems.com/xilinx)



# Boost Your Processor-Based Performance with ESL Tools

Shorten design cycles and lower costs while increasing performance for Xilinx FPGAs using Poseidon's breakthrough ESL technology.

by Farzad Zarrinfar  
VP of Worldwide Sales & Marketing  
Poseidon Design Systems  
[farzad.zarrinfar@poseidon-systems.com](mailto:farzad.zarrinfar@poseidon-systems.com)

Bill Salefski  
VP of Technology  
Poseidon Design Systems  
[bills@poseidon-systems.com](mailto:bills@poseidon-systems.com)

Stephen Simon  
Sr. Director of Marketing and Business Development  
Poseidon Design Systems  
[ssimon@poseidon-systems.com](mailto:ssimon@poseidon-systems.com)

Rapid increases in FPGA platform capability are fueling a corresponding increase in complex SOC designs. This added complexity is pushing designers to rethink their traditional design flow. Designers are looking to system-level tools to aid them in making critical architectural decisions; they are also using pre-defined cores to simplify their design tasks and reduce the overall design cycle. The additions of embedded processors like the PowerPC™ and Xilinx® MicroBlaze™ soft-core processor have greatly expanded the capability of these platforms and have likewise added to the overall complexity.



The embedded processor provides real advantages over discrete implementations when developing complex systems: reduced silicon area, higher flexibility, and a shorter design cycle. These advantages are confirmed with the current rapid growth of embedded FPGA designs, but they do have some limitations. The major challenges of these designs are:

- Insufficient processor performance to support ever-increasing functional demands
- Complex memory hierarchy and system architecture bottlenecks
- Hardware/software partitioning
- Implementing closely coupled interfaces such as APU or FSL
- Productivity limitations and dealing with the complexity of RTL design

Processor solutions run out of processing power rapidly when implementing math-intensive operations. It is very costly to simply increase the clock rate or move the newest technology to enable the design to be scalable. These designs must be able to scale the performance of the processor architecture to match the ever-increasing demand placed on the system. It is costly to move to new processor architectures or utilize higher performance FPGAs.

The second critical issue is the need to support complex memory hierarchies and system peripherals. To reach the performance potential of modern-day processors, designers have turned to memories and architectures, which rely on special burst modes and caching to keep up with processor data requirements. It is critical that the memories and architectures operate in these modes to have a high-efficiency system design.

Effective hardware and software partitioning can be a confusing task. It is difficult to determine the proper mix of hardware and software to meet system requirements without the costly and time-consuming task of moving large portions of an algorithm to hardware.

Special interfaces bring expanded capability to the system architecture. These features add an efficient high-speed interface to the processor core and are ideal for partitioning certain processing-intensive functions to hardware. But this capability does come at a price. These interfaces are complex and take a substantial amount of time and expertise to effectively utilize when interfacing to custom logic.

Designing an efficient processor-based system architecture with overall system performance and optimized for a specific application is not trivial; accomplishing this feat requires breakthrough tools and

methodologies to augment your architectural design methodology. It is also difficult to perform an effective partition of the hardware/software early in the design, which is critical to an effective development of the architecture.

To maximize the benefits of processor-based designs, you must verify architectures early in the design cycle. You cannot wait until RTL development to discover that your architecture does not support your system requirements. This “redesign loop” quickly erodes time-to-market advantages, which makes FPGA-based systems preferred.

With Poseidon tools, you can overcome these design challenges.

### Triton Tool Suite

Poseidon’s Triton Tool suite is a system design and acceleration environment that enables you to quickly develop, analyze, and optimize system architectures. With these ESL tools, the abstraction level of the design is above RTL – you can quickly address system issues without having to solve the detailed issues surrounding RTL implementation. Thus, you can quickly perform architectural “what-if” analysis and accelerate time to market. The tool suite was created specifically for processor-based systems requiring efficient, robust architectures with the need to optimize performance, power, and cost.

Triton comprises two main tools:

- Triton Tuner – a system and software analysis tool
- Triton Builder – a hardware/software partitioning tool

Triton Tuner is a simulation and analysis environment based on SystemC. Simulation is performed at the transaction level from models of both the processor and surrounding buses and peripherals. Using Tuner, you co-simulate the hardware with the application software.

During simulation, the tool collects both hardware and software performance data. The environment then provides tools to visualize and analyze the data, reducing the effort required to identify inefficiencies in the design. Figure 1 shows a bus activity

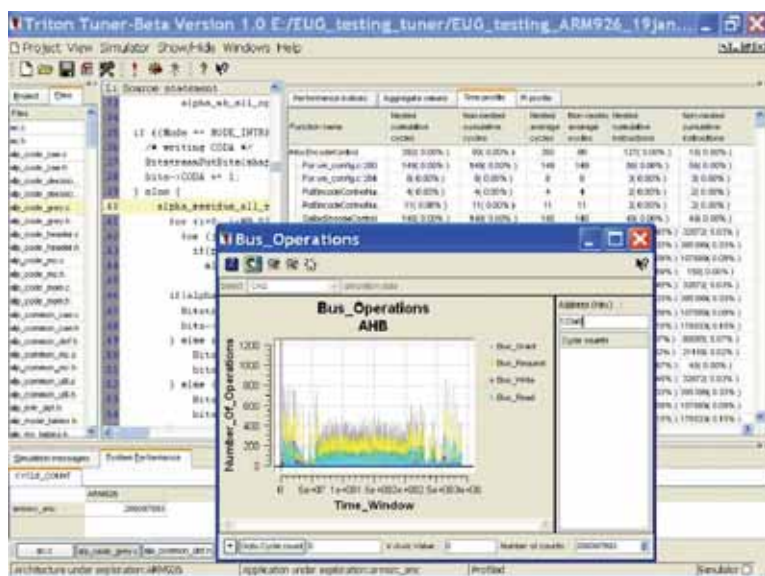


Figure 1 – Bus activity graph

# With Triton Builder, you can easily offload compute-intensive loops and functions from software to hardware.

graph – a visualization tool to help identify bus congestion and bottlenecks in the system design.

Triton Tuner also links hardware and software events, giving you a key tool in determining causal relationships between hardware and software systems. This feature greatly simplifies the task of evaluat-

environment that unobtrusively plugs into a Xilinx EDK design flow, enabling you to quickly make dramatic improvements in the performance and power consumption of your application.

Triton Builder generates the RTL for the complete accelerator, as well as the driver required to invoke the accelerator,

ware components are exactly matched for trouble-free design. The tool also generates an RTL test bench and SystemC model for verifying the new hardware.

## FPGA-Based Accelerator Design Architecture

The Builder tools create a design architecture ideally matched to FPGA design. Using Triton Builder, you can quickly create a peripheral using the C code that runs on the processor. The tool moves the computationally intensive portions of the code into a hardware accelerator. This acceleration hardware is connected directly onto the processor bus or the tightly coupled interface (APU or FSL) and is implemented on the FPGA fabric. A block diagram of a typical accelerated architecture is shown in Figure 2.

If while executing the application code the processor hits a section of code that has been moved to hardware, control is passed to the accelerator through the inserted driver, which then performs the accelerated function. The accelerator runs independently from the processor, freeing it to perform other tasks. When the accelerated task is completed, the results are passed back to the application program. You can implement multiple independent accelerators within the same system.

To create a complete accelerator peripheral, you need more than just a C synthesis tool. Poseidon Builder includes not just a C-to-RTL synthesizer that creates the compute core; it also generates the communication and control hardware. The Poseidon tool creates a complete accelerator peripheral, the block diagram of which is shown in Figure 3. In addition to the compute core, the accelerator includes a multi-channel DMA controller, bus interface, local memories, and other logic to create a plug-and-play peripheral.

## Interfacing to Xilinx Tools

Triton Tools are extremely flexible, allowing you to use either Triton Tuner or

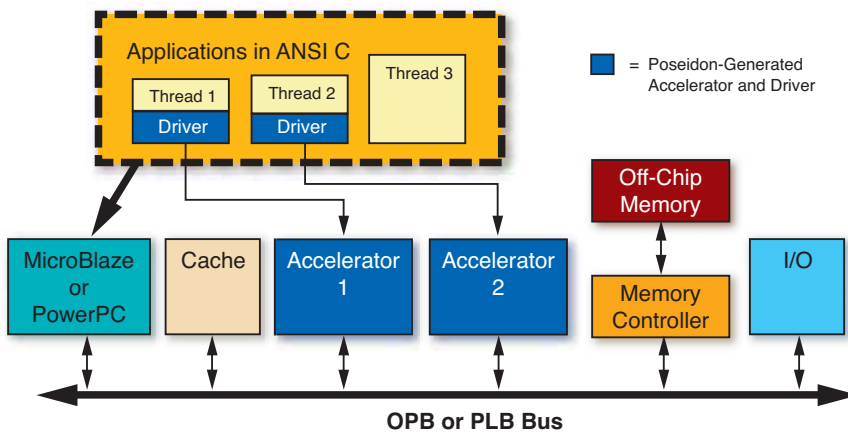


Figure 2 – Accelerated system architecture

ing and optimizing system performance. Triton Tuner also provides tools to optimize the memory structure. This can greatly reduce the need for costly caches and other high-speed memories.

Triton Builder is an automated partitioning tool, which simplifies the task of accelerating the performance of processor-based algorithms. With Triton Builder, you can easily offload compute-intensive loops and functions from software to hardware. The tool automatically creates a direct memory access (DMA)-based hardware accelerator to perform the offloaded task. All hardware and source-code modifications are also created to greatly simplify the re-partitioning process.

With the Builder tool, you can control a number of key parameters in which to optimize the solution to your system requirements. Together, these tools provide a system design and optimization

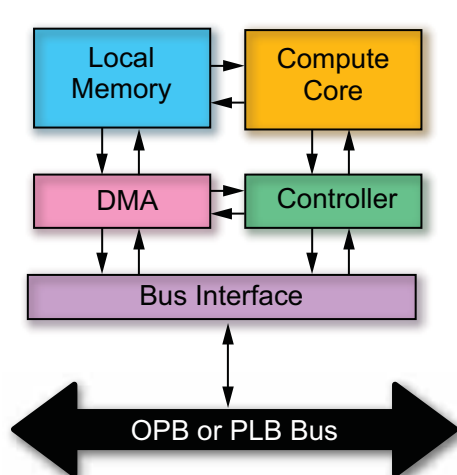


Figure 3 – Accelerator block diagram

and inserts the driver into the proper place in the original application. The RTL can be generated in either Verilog or VHDL. This integral generation process ensures that necessary hardware and soft-

Triton Builder independently or together as an integrated suite. These tools were developed to enhance the productivity in developing processor-based designs and vastly increase their capability.

Triton Tools link seamlessly to EDK tools. A typical system design flow is usually an iterative process where you would

analyze system performance, determine inefficiencies, modify the system, and check the resulting performance. When the architecture performs to the desired level, the system is then transferred back into the Xilinx tool chain. Triton Tools accelerate the process of identifying problem areas and establish an integrated flow

that allows you to move between the tools to develop the optimal architecture.

A typical flow (shown in Figure 4) comprises these steps:

- The designer develops the target architecture using selected Xilinx tools
- The architecture description is read from the microprocessor hardware specification file (.mhs) from EDK, and the ANSI C application source code is read into the Triton tools
- Triton Tuner profiles the ANSI C code, reveals bottlenecks in the code or architecture, and eliminates inefficiencies
- The designer selects which software will be partitioned to hardware
- Triton Builder partitions compute-intensive algorithms in ANSI C into hardware and generates a hardware accelerator
- Triton Tuner verifies that the new system performs to the desired level
- RTL, test bench, modified C code, driver, and architecture are exported back into the Xilinx environment

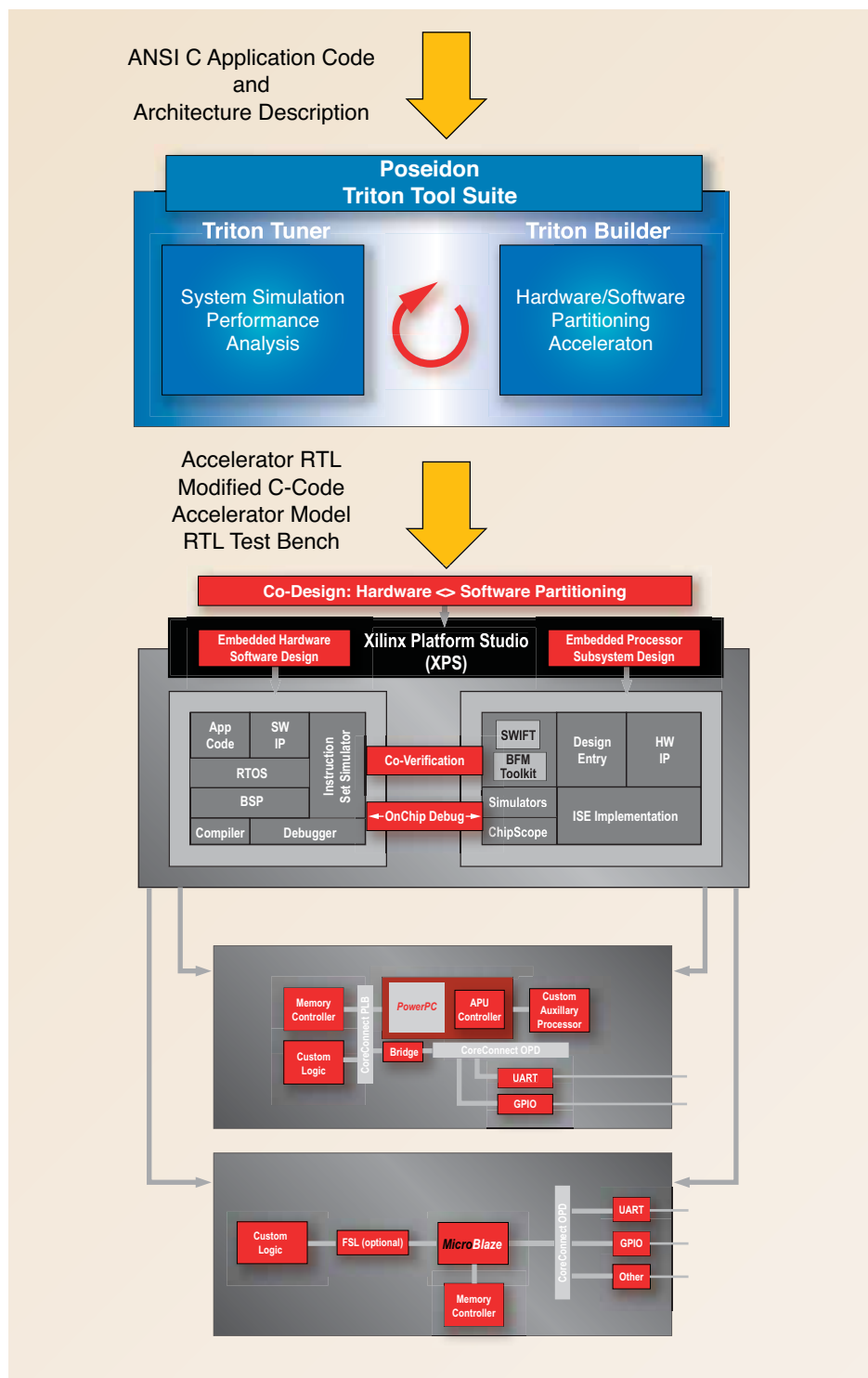


Figure 4 – Integrated Poseidon and Xilinx tool flow

## Conclusion

Poseidon's Triton Tool suite enables you to rapidly and predictably analyze, optimize, and accelerate processor-based architectures. With Triton Tuner's SystemC simulation environment, you can develop robust efficient processor architectures. With Triton Builder, you can add sophisticated hardware acceleration to your processor-based systems and generate RTL from C.

The Triton tool suite enables design architects to achieve higher system throughput, reduced power consumption, and cost, as well as shorten design cycles. This is for FPGAs implementing DSP-intensive applications using embedded Xilinx PowerPC and MicroBlaze processors.

For more information, visit our website at [www.poseidon-systems.com](http://www.poseidon-systems.com) or contact a sales representative at (925) 292-1670. To be qualified for a free tool evaluation of the Poseidon Builder and Tuner and a free white paper, e-mail [farzad.zarrinfar@poseidon-systems.com](mailto:farzad.zarrinfar@poseidon-systems.com).



# FREE on-line training with Demos On Demand



**A** series of compelling, highly technical product demonstrations, presented by Xilinx experts, is now available on-line. These comprehensive 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.

## **Ready for viewing, anytime you are**

Offering live demonstrations of powerful tools, the videos enable you to achieve complex design requirements *and* save time. A complete on-line archive is easily accessible at your fingertips. Also, a *free* DVD containing all the video demos is available at [www.xilinx.com/dod](http://www.xilinx.com/dod). Order yours today!



[www.xilinx.com/dod](http://www.xilinx.com/dod)



Pb-free devices  
available now

# Unveiling Nova

Using a single Virtex-II Pro FPGA to create a reprogrammable video switching system.



By Roger Smith  
Chief Engineer  
Echolab LLC, Inc.  
[rsmith@echolab.com](mailto:rsmith@echolab.com)

Television mixers have historically been built with dedicated hardware to achieve a specific fixed functionality. As mixers have evolved from devices built with discrete transistors to more modern mixers with advanced large-scale integration (LSI) integrated circuits, a common limitation has been that these devices were built with fixed signal and data paths that pre-define the topology of the mixer. Thus, a mixer targeted for two mix/effects (M/Es) and two keyers per M/E is built specifically for that function, with limited or no future adaptability.

The Echolab Nova series completely breaks with tradition in this regard, moving to a completely reconfigurable platform based on a system-on-chip architecture.

We used a single Xilinx® Virtex™-II Pro FPGA to create a completely reprogrammable video switching system. By absorbing the interconnects of the mixer into a single FPGA, the limitations of a fixed signal path architecture have been removed, and the topology of the mixer can be redefined again and again throughout the life of the product.



## Nova Family

In 2004, Echolab launched the first member of the Nova series, the Nova 1716. This product is a full program/preset mixer with 16 inputs and 16 outputs, two downstream keyers, and a full M/E upstream complete with two effects keyers. In 2005, we introduced several new members of the family. The Nova 1932 is a 32-input program/preset mixer, with two upstream M/Es and a

Xilinx parts is support for internal termination of high-speed differential inputs. Because of the high-speed nature of this video I/O, the wires must be treated as transmission lines, with great care paid to electrical termination. Historically, these termination resistors would be placed directly outside the chip, as close as possible to the end of the line. The need for a substantial amount of terminations so close

clock cannot be initially used to sample all of the incoming signals.

Typically, most SDI mixers use individual clock and data separator circuits on each input so that the hardware can recover the individual bits from each stream. After the data streams are separated and decoded, sync detector circuits are used to write these streams into FIFO memories. A common genlock clock and reference is then used to read out the video streams from the memories for effects processing downstream.

This topology is not viable for Echolab's system-on-chip architecture. The large number of clocks created by these front-end clock and data separators would overwhelm the FPGA's support for the total number of clocks, as well as the inherent need to crowd these parts near the FPGA. The video interface to the FPGA must be simpler, and require few or no parts near the FPGA. Thus, traditional clock and data separation techniques do not work here.

Echolab applied an asynchronous data-recovery technique from Xilinx application note XAPP224 ("Data Recovery") originally developed for the networking market. The technique uses precise low-skew clocks to sample the inputs in excess of a gigahertz. The samples are examined to determine the location of the data bit cell transitions. The encoded data is extracted from the stream and can cross into the genlock clock domain without ever extracting the clock from the input stream. For more information about this technique, see [www.xilinx.com/bvdocs/appnotes/xapp224.pdf](http://www.xilinx.com/bvdocs/appnotes/xapp224.pdf).

The FPGA's ability to route nets and guarantee skew performances on the order of picoseconds has enabled development and implementation of this unique SDI input.

Another design challenge in digital mixers has been the implementation of video line delays and FIFOs, which have historically been used in large numbers to time internal video paths and outputs. It is often necessary to add delay lines to AUX bus outputs to keep them in time with the primary mixer outputs. These delay lines and FIFOs have typically been implemented with discrete memory devices.

The Xilinx FPGA solution contains large numbers of video line-length memo-



Figure 1 – The Nova switcher family

full complement of six keyers. The identity4 is a 1 M/E 16-input look-ahead preview mixer. The identity4 brings tremendous advances in video layering, with four upstream and two downstream keyers and five internal pattern generators, all in a flight-pack-size panel and frame.

As shown in Figure 1, what is unique about these mixers is that they share a common frame and electronics, which are simply re-programmed to fit the target mixer design.

## I/O

Major advances in FPGA design were necessary to undertake such a dramatic shift in mixer architectures. One of the first challenges that we had to overcome was how to get the video bandwidth into and out of the FPGA.

Even the smallest Nova family member (shown in Figure 2) has 16 SDI inputs and 16 SDI outputs. At 270 Mbps, this is an aggregate video data bandwidth of more than 8.5 Gbps. The Nova 1732 and 1932 family members have 32 inputs and 16 outputs, approaching approximately 13 Gbps.

One of the key aspects of the Virtex-II Pro device is its ability to support I/O bandwidths in excess of 400 MHz on all FPGA I/O pins. Another key feature of

to the chip would be a layout complication. Because Xilinx can support internal termination of these high-speed video signals, this has greatly simplified the architecture, making the support of large blocks of high-speed video I/O possible.

The next challenge was deciphering the SMPTE 259M stream. Because the SDI streams are coming from all over the studio, even when genlocked (synchronizing a video source with other television signals to allow the signals to be mixed), there can be large phase shifts of +/- half a line between these signals, and therefore, a common



Figure 2 – Nova SDI I/O



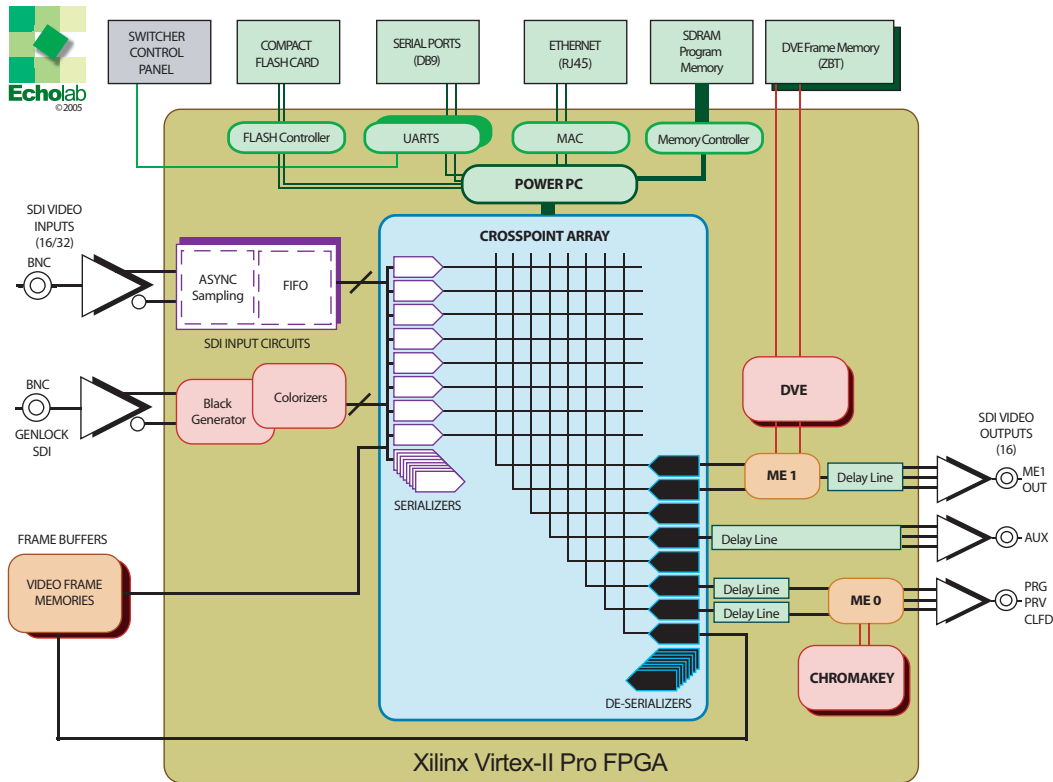


Figure 3 – Virtex-II Pro switcher implementation

ries on chip, which lend themselves naturally to delay lines. Xilinx has enough of these memories on-chip to allow all AUX bus outputs on the Nova series to be timed.

### Crosspoint Array

The next challenge was to implement a crosspoint array of sufficient size to support these large mixers without overwhelming the resources of the chip. Although the crosspoint array appears to be a reasonable size from the front panel, the array is often much larger because it must support all of the internal sources and functions inside the mixer.

The Nova 1716 has 16 external inputs, but it also has 3 internal colorizers, black, three internal frame buffers, and several intermediate sources generated by the upstream M/E. For outputs from the array, each M/E requires seven video buses (A/B, key 1 cut and fill, key 2 cut and fill, as well as video for borders). There are also 12 AUX bus feeds and dedicated buses to support capture on the internal frame buffers.

By the time you add it all up, the required internal crosspoint array is easily 30 x 30 for the 16-input mixer. To develop this crosspoint array as a 10-bit wide parallel implementation would consume a large amount of resources within the FPGA.

A more effective use requires the design of high-speed serializers and de-

serializers within the part. The Virtex-II Pro FPGA's ability to generate multiples of the gen-lock clock within the part with low skew allows you to create whole sections of the chip that can run at the SMPTE 259M bit-serial rate of 270 MHz. This implementation of an SDI serial rate crosspoint array (Figure 3) effectively limits the use of valuable FPGA fabric to less than 10 percent of the capacity of the target chip.

### Effects Generation

Once the streams have gone through the crosspoint array and have been de-serialized, the video buses, now "timed," can be routed to the appropriate processing blocks within the FPGA to perform various video effects (see Figure 4).

The creation of video effects within the FPGA such as wipes, mixes, and keys is easily performed with the basic building blocks of the FPGA. Most basic video effects can be performed with nothing more than simple combinations of addition, subtraction, and multiplication. Hundreds of embedded high-speed multipliers within the FPGA fabric allow a variety of video effects to be performed effortlessly with very high precision.

Embedded memory can also be used for LUTs and filter coefficient storage. An example of an often-used filter would be an interpolating filter for 4:4:4 up-sampling before a DVE or Chromakey.

A high-precision circle wipe requires a square-root function. This complicated mathematical function was implemented with a CORDIC core provided by Xilinx. These blocks of pre-built and tested IP from Xilinx and other third-party developers allow you to rapidly deploy designs without having to develop and test every building block from scratch. Development with these IP cores can remain at a very high level, providing quick time to market. Also, the optimized



Figure 4 – Effects

size of these cores allows the design of complex mixers within a single chip.

### Embedded PowerPC

The computer horsepower necessary to run today's large vision mixers has grown immensely over the last dozen years. The need to accurately support field-rate effects on multiple M/E banks while at the same time communicating with complex control panels and many external devices has tested the limits of earlier 8- and 16-bit microprocessors.

Most manufacturers have either used several smaller distributed processors or a larger, faster 32-bit microprocessor. Echolab's system-on-chip architecture (see Figure 5 – PowerPC™ implementation) takes advantage of two 32-bit PowerPCs

letproof as previous generations of mixers.

Echolab has chosen Micrium's  $\mu$ C/OS-II real-time operating system (RTOS) for the Nova series (see [www.micrium.com](http://www.micrium.com)). This OS is a priority-based, pre-emptive multitasking kernel that has been certified for use in safety-critical applications in medical and aviation instruments. Time-critical video processing is assigned to the highest priority task. Management of the file system, console I/O, and network stack are allocated to lower priority tasks, allowing the processor to utilize spare processor cycles in the background without ever interfering with the video hardware.

The tasks communicate with each other – and synchronize their activity – with thread-safe semaphores and message queues provided by the OS.

6), designed to hold all of the firmware and software to configure and boot the Nova. With a user-accessible mode switch, you can load as many as eight different on-line configurations of Nova firmware and software from a single flash card.

The remaining storage on the card is available to store user data such as sequences and panel saves, as well as key memories and other user settings. Also, the Compact Flash is designed to hold all of the graphics and stills online. Support for cards up to 2 GB or more provides an immense storage capacity, well beyond the archaic floppy disks found in competitive products.

Given Nova's unique architecture, it is easy for the product to be extended through downloadable firmware updates. These updates can be as simple as a routine

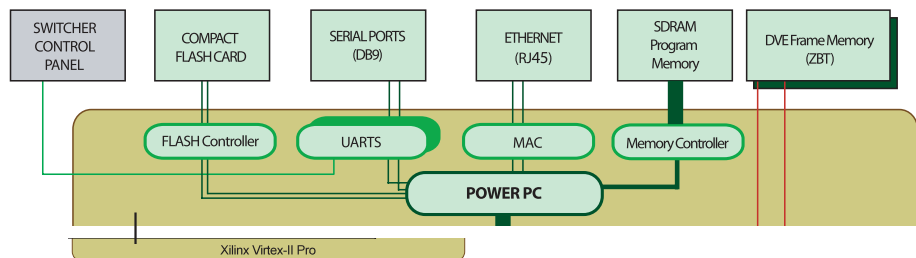


Figure 5 – PowerPC implementation

(running at 270 MHz) embedded directly in the fabric of the Xilinx FPGA. This tight coupling between the processor and the mixer hardware leads to substantial performance improvements.

A vast library of pre-built processor peripherals from Xilinx and third parties makes it easy for functions like serial ports, memory controllers, and even Ethernet peripherals to be dropped right into the design. Custom peripherals are also easy to design.

Several large switcher manufacturers have gone to commercial operating systems like Windows or Linux to improve their software productivity. Although the early benefits can be appealing, the downside to this transition is a loss of control over the reliability of the switcher's code base, making a device that is not as robust and bul-

### System Connectivity

All Nova series switchers have multiple ports for broadcast studio interconnectivity. An industry-standard RS-422 port allows for the implementation of industry-standard editing protocols. A standard RS-232 port is available for PC connectivity. An Ethernet port allows the switcher to be directly connected to a network.

Under control of Nova's  $\mu$ C/OS-II, several servers are running concurrently that provide an integrated Web server for remote status and display, an embedded XML-RPC server for remote control, as well as a full TFTP server for remote upload and download of graphics and stills.

### Compact Flash and Re-Configurability

At the heart of the Nova system is an industry-standard Compact Flash card (Figure



Figure 6 – Compact Flash

software patch, as complex as adding a keyer to an AUX bus output, or restructuring the internal video flow within an ME for a specialized application. Architectures as different as the 2 ME Nova 1716 (with its program/preset architecture) and the Nova identity4 (with a six-keyer look-ahead preview structure) can be loaded into the same hardware.

### Roadmap to the Future

Next-generation Virtex-4 FPGA technology from Xilinx will allow Echolab to move its system-on-chip architecture to support high-definition products. Virtex-4 FPGAs bring a

higher level of performance to the embedded logic, as well as the embedded peripherals. Some of these enhancements include:

#### Fabric enhancements

- Larger arrays
- Faster
- Lower power

#### I/O enhancements

- General-purpose I/O speeds to 1 GHz
- Dedicated Rocket IO™ transceiver speeds beyond 10 GHz

#### Dedicated hardware resources

- Multiple tri-mode Ethernet MACs
- 500 MHz multipliers with integrated 48-bit accumulators for DSP functions
- Block memories now have dedicated address generators for FIFO support


#### Conclusion

Television mixers have grown larger and more complex in the last dozen years. More and more, they are the focal point for the interconnection of a wide range of studio equipment.

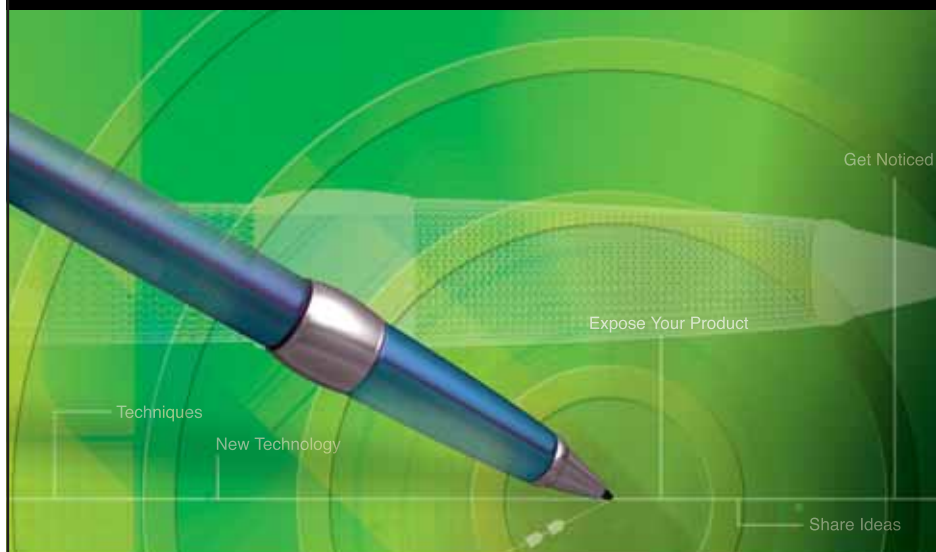
One of the primary benefits of the system-on-chip architecture is reduced parts count. This reduction in parts count contributes directly to lower power, reduced PCB complexity, higher reliability, and reduced cost.

Another major benefit of the system-on-chip architecture is that it is almost entirely reconfigurable. This has allowed multiple products with different video architectures to be built on a common platform. This also lends itself to easy customization for specialized applications or specific vertical markets.

As the computer network continues to play more of a role in today's modern television studio, the Nova series will be ready with support for streaming video over Gigabit Ethernet. H.264 and WMV9 codecs will drop right into the Nova's system-on-chip architecture, providing future features on today's hardware.

For more information, please feel free to see Echolab's complete line of television mixers at [www.echolab.com](http://www.echolab.com). 

# GET PUBLISHED



## WOULD YOU LIKE TO WRITE FOR XCELL PUBLICATIONS?

It's easier than you think!

Submit an article draft for our Web-based or printed publications and we will assign an editor and a graphic artist to work with you to make your work look as good as possible.

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

Forrest Couch  
Publisher, Xcell Publications  
[xcell@xilinx.com](mailto:xcell@xilinx.com)



See all the new publications on our website.

[www.xilinx.com/xcell](http://www.xilinx.com/xcell)



# Board Level Solutions DSP plus FPGA!

## Four Screamers

...four screaming fast  
6416 DSPs, that is



## Radio frequency signal processing solution

- ▶ Four, 720 MHz TMS320C6416 DSPs each with:
  - 64MB SDRAM
  - Dedicated 200 MB/s links between DSPs and FPGAs
- ▶ Two, 4M Gate User-Programmable FPGAs each with:
  - 18MB private DDR SDRAM
  - 128 MB private DDR SDRAM
- ▶ 64-bit/66MHz CompactPCI
  - 512MB Global DDR SDRAM
  - Two PMC Sites with JN4 to FPGA
  - External Data Port, up to 12Gb/s
  - StarFabric P1CM62.17 Compliant Port

Download  
Complete  
User Manuals  
NOW!



**Quadra**

**Innovative  
Integration**  
... real time solutions!

www.innovative-dsp.com • 805.520.3300 phone

*Hungry like the wolf...*  
for an ultra fast digital waveform  
capture and playback solution?

Get your Lobo Data Sheets &  
On-line Pricing Now!  
www.innovative-dsp.com/lobo

### Lobo's Features

- ▶ 225MHz/1350MFLOPS TMS320C6713 DSP
- ▶ 3 or 6 MGATE Virtex-II FPGA
- ▶ 32MB SDRAM, Up to 2GB DDR SDRAM for FPGA
- ▶ External I/O Interface for Data Card to FPGA DIO
- ▶ Separate Receive/Transmit Channels



**Innovative  
Integration**  
... real time solutions!

www.innovative-dsp.com • 805.520.3300 phone

**lobo**

> powerful  
connections



Network with the SBC6713e, our highest-performance stand-alone  
DSP board with on-board 10/100 Ethernet

### Features

- ▶ 300MHz, 32-bit, floating-point TMS320C6713 DSP
  - Powerful C/C++ libraries, Windows debugger
- ▶ 10/100 Ethernet via DM642 coprocessor for real-time network I/O
- ▶ Two OMNIBUS I/O expansion sites
  - Wide selection of analog input/output
  - Up to 24-bit resolution, up to 64 MHz sample rate
- ▶ Capable of 100% stand-alone operation
  - 3U Size 300mm x 160mm
  - 4 Mb Flash ROM
- ▶ Fantastic, on-board peripherals
  - RS232, 32-bit digital I/O, watchdog
  - 600K gate Spartan-III for user-logic (optional)

### Applications

- ▶ Embedded Servo Control
- ▶ Remote Data Acquisition
- ▶ Industrial Test & Measurement
- ▶ OEM Instrumentation

Download  
Data Sheets  
NOW!

**Innovative  
Integration**  
... real time solutions!

805.520.3300 phone  
www.innovative-dsp.com

*Onward to Glory*

Complete Software  
Radio Solution on a  
single 6U card

### Features

- ▶ 6 Million gate Virtex II FPGA
- ▶ Scalable Software Defined Radio
- ▶ Embedded Processing via MatLab
- ▶ TMS320C6416 DSP
- ▶ 105 MHz, 14 bit 2 Ch. Analog I/O
- ▶ 32 MB RAM
- ▶ 32/64-bit cPCI bus
- ▶ PMC expansion site
- ▶ STAR Fabric interface

Get your data sheets now!  
www.innovative-dsp.com/quixote

sales@innovative-dsp.com  
805.520.3300 phone • 805.579.1730 fax



**Quixote**

**Innovative  
Integration**  
... real time solutions!

Free Instant  
On-Line Pricing!

805.520.3300 phone  
www.innovative-dsp.com

**Innovative  
Integration**  
... real time solutions!



# Implementing a Lightweight Web Server Using PowerPC and Tri-Mode Ethernet MAC in Virtex-4 FX FPGAs

Using Ethernet, you can connect your device to the Internet, monitoring and controlling it remotely.

by Jue Sun  
Systems Engineering Intern  
Xilinx, Inc.  
[juebejue@gmail.com](mailto:juebejue@gmail.com)

Peter Ryser  
Sr. Manager, Systems Engineering  
Xilinx, Inc.  
[peter.ryser@xilinx.com](mailto:peter.ryser@xilinx.com)

The Tri-Mode Ethernet MAC (TEMAC) UltraController-II Module (UCM) is a minimal footprint, embedded network processing engine based on the PowerPC™ 405 (PPC405) processor core and the TEMAC core embedded within a Xilinx® Virtex™-4 FX Platform FPGA. It allows you to interact with your Virtex-4-based system through an Ethernet connection and to control or monitor your system remotely using TCP/IP from miles away. Our design uses minimal resources and ensures that you will have enough logic area for your application.

In this article, we will explain the advantage of our design and how it is implemented on an ML403 board. We will also introduce a few applications that you can build on top of this implementation.

## Implementation

The TEMAC UCM leverages key innovations in Virtex-4 FPGAs to implement TCP/IP applications with minimal resource usage, as shown in Figure 1. The whole implementation takes one embedded PPC405, one integrated TEMAC, two Virtex-4 FIFO16s, 20 slice flip-flops, and 18 look-up tables (LUTs).

On the ML403 board, the TEMAC UCM connects to an external PHY through a gigabit media independent interface (GMII) and a management data input/output (MDIO) interface, and auto-negotiates tri-mode (10/100/1000 Mbps) Ethernet speeds. Other physical interfaces like RGMII and SGMII are possible and take a minimum amount of change in the reference design that is provided as source code.

The software is ported from the open-source  $\mu$ IP TCP/IP stack and runs completely out of the PPC405's 16 KB instruction and 16 KB data cache. It accesses Ethernet frames in two FIFO16s through the PPC405 on-chip memory (OCM) interface. One FIFO buffers inbound while the other buffers outbound Ethernet frames. The FIFOs also act as entities to synchronize clock domains between the PPC405 OCM and the TEMAC. The PPC405 is capable of running the software at maximum frequency, for example, 350 MHz in a -10 speed grade FPGA.

The application implemented in Xilinx Application Note XAPP807, "Minimal Footprint Tri-Mode Ethernet MAC Processing Engine," ([www.xilinx.com/bvdocs/appnotes/xapp807.pdf](http://www.xilinx.com/bvdocs/appnotes/xapp807.pdf)) runs a Web server on top of the  $\mu$ IP TCP/IP stack. The Web server serves a Web form to connecting clients. You simply enter a string in a text field and submit it to the server, which interprets the data and displays the string on a two-line character LCD. See Figure 2 for a picture of the process.

The  $\mu$ IP TCP/IP stack has a well-defined API and comes with other demonstration applications like telnet. Based on the well-written  $\mu$ IP documentation, you can implement your own application without much effort. The stack is optimized for minimal footprint at the cost of performance. The restriction on TCP/IP perform-

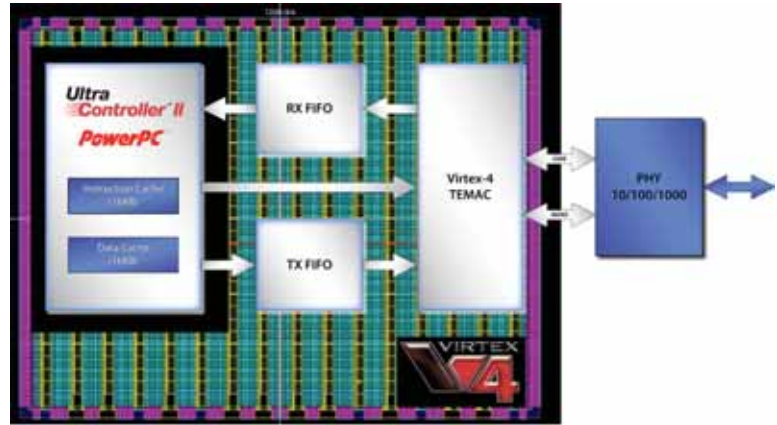


Figure 1 – General datapath of TEMAC UCM



Figure 2 – Example Web server allowing remote control of ML403

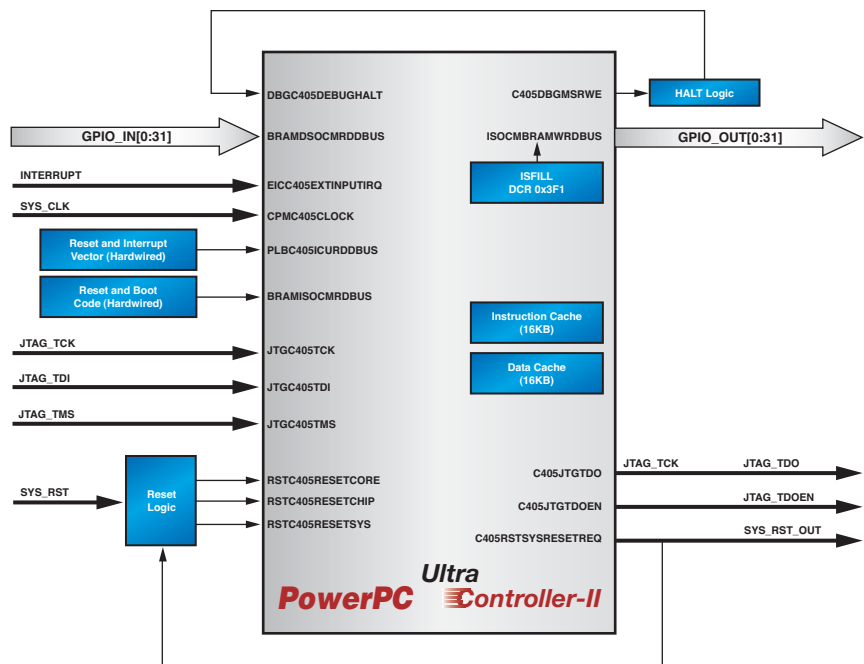


Figure 3 – UltraController-II ports



## Other configuration solutions besides System ACE CF can load code and data into the PPC405 caches. Such solutions include Xilinx Platform Flash as well as external processors and methods...

ance is not a limiting issue for most monitoring and control applications.

### Implementation Flow

Implementing the TEMAC UCM is straightforward. The implementation flow comprises three main parts: one flow in Project Navigator for hardware bit file generation; one flow in EDK for software ELF file generation; and another flow to combine the software and hardware file into a single bitstream or PROM file to program the Virtex-4 FX12 FPGA or the Platform Flash on the ML403 board, respectively. Project files and scripts for all of these flows are included in XAPP807. The hardware source code for the TEMAC UCM is available in Verilog and VHDL. The software source code is written in C.

The TEMAC UCM is based on UltraController-II, as shown in Figure 3 and documented in XAPP575, "UltraController-II: Minimal Footprint Embedded Processing Engine" ([www.xilinx.com/bvdocs/appnotes/xapp575.pdf](http://www.xilinx.com/bvdocs/appnotes/xapp575.pdf)). UltraController-II is a black-box processing engine that includes 32 bits of user-defined general-purpose input and output, as well as interrupt handling capability.

### Loading the Software into Caches

Loading software into the instruction and data caches of the PowerPC has been possible for some time through System ACE™ CF technology or other JTAG-based methods. However, for the first time, the TEMAC UCM makes this also possible through all configuration modes, including JTAG mode, slave and master serial modes, and slave and master SelectMap modes. Other configuration solutions besides System ACE CF can load code and data into the PPC405 caches. Such solutions include Xilinx Platform Flash as well as external processors and methods described in application notes such as XAPP058, "Xilinx In-System Programming Using an

Embedded Microcontroller" ([www.xilinx.com/bvdocs/appnotes/xapp058.pdf](http://www.xilinx.com/bvdocs/appnotes/xapp058.pdf)).

The cache loading solution emphasizes another new feature in the Virtex-4 FPGA family. The USER\_ACCESS register is a 32-bit register that provides a port from the configuration block into the FPGA fabric. For loading the caches, the USER\_ACCESS register is connected through a small state machine to the JTAG port of the PPC405. A script converts a software ELF file into a bitstream that you can load into the processor caches with Impact or the ChipScope™ Analyzer. If you are interested in finding out more about this solution, Xilinx Application Note XAPP719, "PowerPC Cache Configuration Using the USER\_ACCESS\_VIRTEX4 Register," ([www.xilinx.com/bvdocs/appnotes/xapp719.pdf](http://www.xilinx.com/bvdocs/appnotes/xapp719.pdf)) provides full details.

### Use Cases for the TEMAC UCM

Figure 4 shows some use cases for the TEMAC UCM. Besides monitoring and control application, other applications include statistics gathering, system diag-

nostics, display control, or math and data manipulation operations.

A system application that deals with MPEG transport streams gathers statistical information about the video and audio streams. The TEMAC UCM makes this information available through the Web interface. On the control side, the content provider dynamically loads decryption keys for encrypted streams into the system through the same Web interface.

In another example, an FPGA controlled machine gathers information about erroneous conditions that the machine manufacturer accesses locally or remotely for diagnosis. In return, the manufacturer loads new machine parameters into the system to adjust undesired behavior.

In a third example, some networking equipment reports its operational status on a regular basis through e-mail to a control center where the information is processed. The control center equipment or personnel takes action if the equipment reports a malfunction or fails to report at all.

In all of these cases, the TEMAC UCM fits into a design for which it was

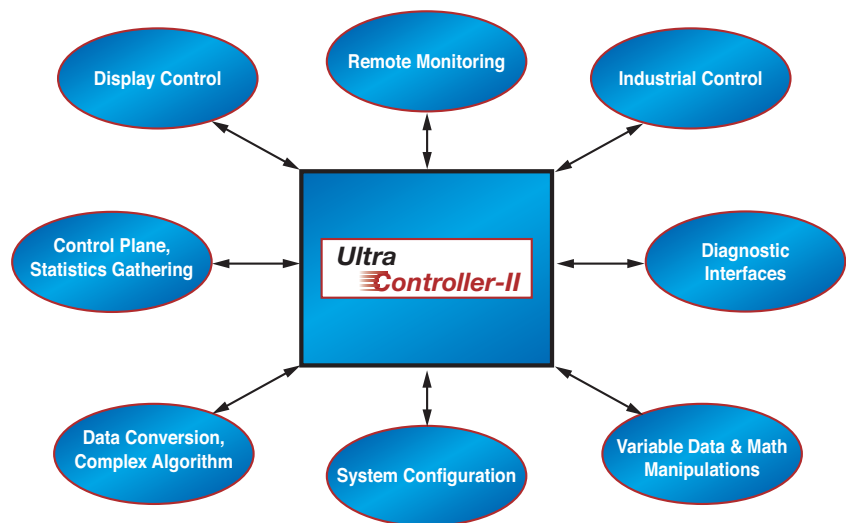


Figure 4 – UltraController-II's possible applications

not originally designed. Its small hardware and software footprint makes it suitable for designs that already use all of the available FPGA resources.

### Network-Attached Co-Processor

In a recent experiment, we combined the TEMAC UCM with the auxiliary processor unit (APU) on the PPC405. The APU provides a direct way to access hardware accelerators by the way of user-defined

instructions (UDI). Typically, a software application running on the PPC405 uses the APU to accelerate the execution speed. In this case, however, the application software runs on a regular PC that distributes the workload to multiple ML403 boards with integrated TEMAC UCM and APU, as shown in Figure 5. The ML403 acts as a network-attached co-processor, speeding up the application running on the PC.

For example, we compute pictures gener-

ated by the Mandelbrot function. The software running on the PC divides the whole picture into smaller areas and distributes the parameters for these smaller areas to the ML403 boards through TCP connections. The PPC405 on each ML403 board reads the parameters from the TCP socket, loops through the area, and calculates every pixel using a MandelPoint co-processor connected to the APU interface. The PPC405 sends the results of this calculation back to the PC through the same TCP connection. Finally, the PC collects all the results from the network-attached co-processors and displays the final picture on the screen.

Figure 6 shows the result of the acceleration. A single ML403 connected to the PC computes a 1024 x 768 picture in about three seconds, including communication overhead. Adding more ML403 boards brings the overall computation time down below 1.5 seconds for three boards. Four boards do not show a significant improvement, as communication overhead becomes the main part of the computation.

Solving the same problem natively on a high-end Linux workstation takes about three seconds, about the same amount of time as one network-attached ML403 with TEMAC UCM and APU acceleration.

### Conclusion

The TEMAC UCM provides a convenient and inexpensive way of adding Ethernet functionality to any design. It uses a minimal amount of FPGA resources and the software runs from the embedded PPC405 caches. As a result of these simple interfaces, development time is reduced. The Web server application is used to demonstrate the design, but you can use it for other solutions, as shown with the example of the network-attached co-processors.

For more information about the TEMAC UltraController Module, download XAPP807, including the reference design for the Xilinx ML403 board, and see the Xilinx Webcast, "Learn about the UltraController-II Reference Design with PowerPC and Tri-Mode EMAC in Virtex-4 FPGAs," at [www.xilinx.com/events/webcasts/110105\\_ultracontroller2.htm](http://www.xilinx.com/events/webcasts/110105_ultracontroller2.htm).

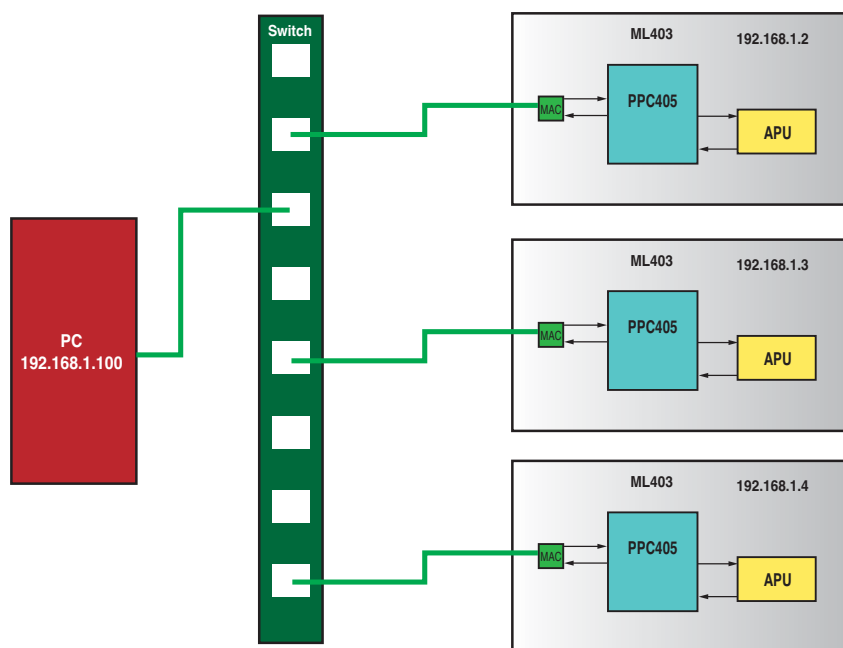


Figure 5 – Several ML403s with APU activated to accelerate computation

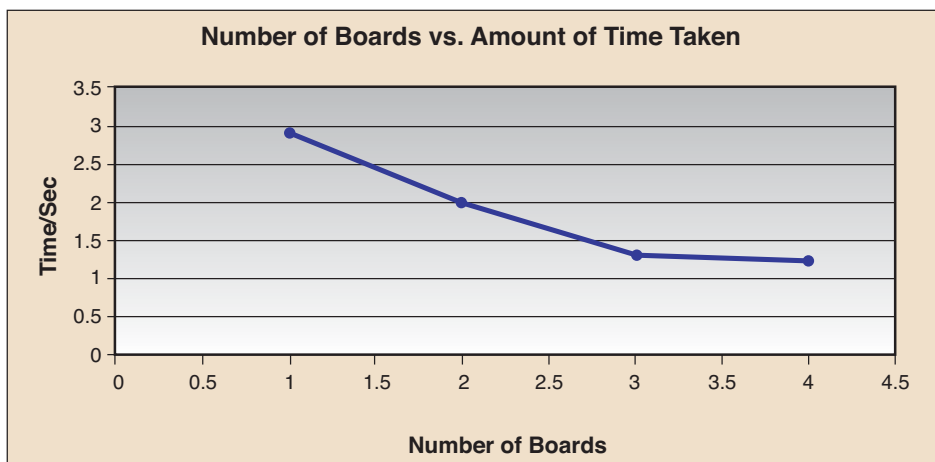


Figure 6 – Use multiple boards to shorten computation time

# Development Kits Accelerate Embedded Design

A single Virtex-4 FX12 Development Kit supports both PowerPC and MicroBlaze processor design.



by Jay Gould  
Product Marketing Manager,  
Xilinx Embedded Solutions Marketing  
Xilinx, Inc.  
[jay.gould@xilinx.com](mailto:jay.gould@xilinx.com)

Do you always have the best balance of processor, performance, IP, and peripherals on your first embedded application design partition? Would experimenting with customizable IP and a choice of hard- and/or soft-processor cores give you a better feel for the best fit to your embedded application before committing to devices and hardware? Programmable platforms allow powerful customization of processor-based systems, but which one do you start with? Why not utilize a development kit that supports it all?

Ideally, modern development kits provide a comprehensive design environment with everything embedded developers require to create processor-based systems. They should include both the hardware platform and software tools so that you can get started creating modules to prove out a design concept before the time-con-

suming task of building custom hardware. By starting with a “working” hardware platform, embedded teams can evaluate the merits of one processor core versus another, begin writing application code, and even experiment with IP peripherals before prematurely committing to a specific hardware design.

A development kit should provide a stable hardware evaluation environment, complete with all required cables and probes, so that you can focus on your own application and not worry about debugging problems with a new board or broken connection. Imagine the time-to-market advantages of being able to immediately write, test, and debug code on a working reference board well in advance of any custom hardware being ready.

## Processor “P” or Processor “M”?

Choosing the most appropriate processor for a specific embedded application is always a design concern. A processor used on a previous project may not be a good fit for the next project. Requirements for performance, features, size, and cost may

change wildly from one product to the next. Having multiple processor core choices and the flexibility to change the design after the development cycle has started is highly desirable.

Xilinx offers system developers the choice of high-performance, “hard” PowerPC™ or flexible “soft” Xilinx® MicroBlaze™ processor cores for embedded designs. The PowerPC cores provide high performance because they are hard-immersed into the Virtex™-4 FPGA device fabric. They also have additional performance-enhancing features, such as integrated 10/100/1000 Mbps Ethernet and Auxiliary Processor Unit (APU) controllers.

The 32-bit RISC MicroBlaze soft IP processor can be instantiated in any of the Xilinx Platform FPGAs because it is built like a macro out of FPGA elements. Although the hard PowerPC processor will always be faster and smaller by nature, the MicroBlaze soft-processor core has the flexibility to instantiate at any time during the design cycle. MicroBlaze cores can also be used as complements to PowerPC cores in Virtex-4 platforms, often provid-



ing workload distribution to optimize total system performance.

With a unified strategy of supporting both processors with the same IP library and design tool suite, you have the power to choose the best processor/IP configuration for any given design requirement. With this scenario, you do not have to throw away good work or start a design over again with different tools or IP libraries just because a processor change has been made.

### Best of Both Worlds

The Virtex-4 FX family of FPGAs combines the best of both hard- and soft-processing options with a unique format for the embedded industry. For example, the powerful Virtex-4 FX12 device implemented on the Xilinx ML403 development board (Figure 1) offers an immersed PowerPC core at a value price point. This same device and board can easily support the addition of MicroBlaze soft-processor designs, creating one unique platform supporting two core options.

Engineers researching the appropriate processor choice for their next project can now experiment with a broader set of processor and IP offerings while making their decision. With a single tool suite and IP library supporting both processor options, you can really do an apples-to-apples evaluation of the benefits versus your own design requirements for the best fit.

The ML403 development board provides a rich set of features, including peripherals, memory, audio, video, and user interfaces, enabling you to easily and cost-effectively prototype your embedded system design. The actual board itself includes:

- Memory interfaces for DDR SDRAM, ZBT SRAM, and flash
- Audio and video interfaces
- Numerous user interfaces: dual PS/2, IIC Bus, RS-232, USB, and tri-mode Ethernet
- Multiple FPGA programming modes: Platform Flash, the System ACE™ solution, Linear Flash, and PC4
- Support for multiple clock sources and differential clock inputs

The Virtex-4 FX device also supports an APU controller that provides a high-bandwidth interface between the PowerPC 405 core and co-processors to execute custom instructions in the FPGA fabric. Additionally, the ML403 showcases the FX12 implementation of two fully integrated 10/100/1000 Ethernet MACs for system communication and management functions. These FX capabilities enable developers to optimize their designs for high performance.

### Completely Integrated Development Kit

The Xilinx PowerPC and MicroBlaze Development Kit Virtex-4 FX12 Edition integrates a complete environment for embedded development. The kit supports both the PowerPC 405 immersed hard processor and the MicroBlaze soft processor

for Virtex-4 FX12 designs. The kit comprises the following contents (Figure 2):

- Virtex-4 FX12 development board – ML403
- Platform Studio embedded tool suite and Embedded Development Kit (EDK)
- ISE™ (Integrated Software Environment) FPGA tools
- JTAG probe, Ethernet, and serial cables
- ChipScope™ Pro Analyzer (Evaluation Version)
- Reference designs

In addition to the ML403 board, the kit showcases the IEC DesignVision award-winning Xilinx Platform Studio (XPS) embedded tool suite. XPS is the integrated development environment that includes the design GUI, automated configuration wizards, compiler, and debugger. XPS is built on the Eclipse framework and supports the GNU tool chain. Design wizards automate the process of configuring the processor-based system, connect and customize the IP, and organize the project. Additionally, Platform Studio can automatically generate example test code, software drivers, and even BSPs for some of the most popular RTOSs.



Figure 1 – ML403 development board



Figure 2 – Virtex-4 FX12 PPC/MicroBlaze Development Kit

XPS is fully aware of the ML403 development board and, unlike “universal” tools, automatically populates the GUI with the feature options supported by that board. Intelligent tools automate the flow, reduce learning curves, and accelerate the design process. Users are amazed that they can create a basic embedded system using Platform Studio’s automated flow in less than 15 minutes.

EDK bundles the XPS tool suite with the impressive embedded IP library and MicroBlaze processor license. This is a one-time license; there is no royalty to be paid for MicroBlaze designs shipped. The IP library supports the CoreConnect bus, bridge, and arbiters, and offers a large peripheral catalog including more than 60 memory controllers, Ethernet, UARTs, timers, GPIO, DMA, and interrupt controllers, as well as many other cores.

ISE FPGA tools are the design utilities employed for the FPGA implementation, including entry, synthesis, verification, and place and route. This design flow can be invoked directly from the Platform Studio IDE.

Target boards need a connection for the various kinds of communication coming from the host computer where tools are executed and design files created. The most common method of connection to an embedded target board is through an industry-standard JTAG probe. Xilinx offers a choice of either parallel or USB for JTAG probes; this single connection can be used for both FPGA download/debugging and embedded software download/debugging. This capability reduces your need for multiple probes and the inconvenience of constantly swapping probes. The FX12 kit also includes Ethernet and serial cables.

One unique, powerful feature that Xilinx offers is “platform debug” – the integration of the embedded software debugger and hardware debugger. By integrating the cross-triggering of both hardware and software debuggers, embedded engineers can now find and fix system bugs faster. Can’t detect the problem when the software goes off into the weeds? Have the software debugger cross-trigger the hardware debug-

ger so that you can examine the state of the hardware. Alternatively, if you locate a problem after a hardware event, cross-trigger the software debugger to step through the code that is executing at that time. The FX12 kit includes an evaluation version of the Xilinx ChipScope Pro hardware debugging tool to encourage you to explore the merits of platform debug.

### Pre-Verified Reference Designs

The last and critical ingredient of the integrated development kit that can really jump-start your entire design process is a collection of reference designs or reference systems. By including pre-verified or pre-existing working example designs with the kit, you can unpack the box and have the basic system up and running in minutes. Reference designs can prove that the hardware and connections are working before you start creating new code or IP, and prevent you from mistakenly debugging your design when what you really have is a bad board or cable.

These reference systems also act as examples for the broad set of features on the ML403 platform, such as Ethernet, DDR memory, video, and audio functions. You can use these examples as templates to get your own design features modeled, or run as-is if your custom board targets the same feature.

The FX12 Development Kit includes boot-able eOS (embedded operating systems) or RTOS examples for Wind River Systems’ VxWorks and MontaVista embedded Linux from a Compact Flash device. The ML403 board is supported by other third-party embedded RTOS or hardware/software design tool partners as well.

Sample reference systems include:

- PowerPC Reference System with VxWorks and Linux
- MicroBlaze Reference System
- DCM Phase Shift Using MicroBlaze Reference System
- UltraController-II Reference Design
- Gigabit System Reference Design (GSRD)
- APU Co-Processing Acceleration


These reference systems can save you days and even months of development time compared to manually generating every design module yourself. Leveraging existing examples jump-starts your design cycle.

### Conclusion

A complete development environment of hardware board, design tools, IP, and pre-verified reference designs can dramatically accelerate embedded development. Additionally, for a quick, out-of-the-box experience, the ideal development environment also provides all of the extra JTAG probe, serial, and Ethernet cables. Intelligent, award-winning design tools that are “platform aware” make the kit easy to use, providing wizards that help you configure the system by automating the flow for the known features on the development board.

The programmable platform and intelligent XPS tools provided with the PowerPC and MicroBlaze Development Kit enable you to craft embedded systems with the optimal combination of features, performance, area, and cost. You can choose the most effective processor core for the target application, customize IP, optimize the performance, and validate the software on a development board before your own hardware is even back from the shop.

Engineers need options and flexibility when trying to satisfy all of their project requirements. The Xilinx Virtex-4 FX12 Development Kit uniquely offers two processor cores for easy evaluation in a single, integrated environment. A single IP library and common tool suite unifies the design environment for both cores, providing total embedded architecture flexibility. The inclusion of numerous pre-verified reference designs empowers you to jump-start the development cycle so that you can spend more time adding value to your end application.

To learn more about the low-cost Virtex-4 FX12 Development Kit supporting both PowerPC and MicroBlaze processor cores, please visit [www.xilinx.com/embdevkits](http://www.xilinx.com/embdevkits). A good starting point to learn about all of our embedded processing solutions is [www.xilinx.com/processor](http://www.xilinx.com/processor). 

## MicroBlaze – The Low-Cost and Flexible Processing Solution

Finding a processor to meet performance, feature, and cost targets can be very challenging in today's competitive environment. With the continued advances in FPGA technology features, increased performance and higher density devices, scalable processor systems can now be offered as an economical, superior alternative for real-time processing needs. A flexible processor system that's easy-to-use, area-efficient, optimized for cost-sensitive designs, and able to support you well into the future is delivered by the award-winning Xilinx MicroBlaze™ solution.

The Xilinx MicroBlaze processor, named to EDN's Hot 100 Products of 2005, is a 32-bit RISC "soft" core operating at up to 200 MHz on the Virtex™-4 FPGA and costs less than 50 cents to implement in the Spartan™-3E FPGA. Because the processor is a soft core, you can choose from any combination of highly customizable features that will bring your products to market faster, extend your product's life cycle, and avoid processor obsolescence.



[www.edn.com](http://www.edn.com)



### The MicroBlaze Advantage

Xilinx unleashes the potential of embedded FPGA designs with the award-winning MicroBlaze soft processor solution. The MicroBlaze core is a 3-stage pipeline 32-bit RISC Harvard architecture soft processor core with 32 general purpose registers, ALU, and a rich instruction set optimized for embedded applications. It supports both on-chip block RAM and/or external memory. With the MicroBlaze soft processor solution, you have complete flexibility to select any combination of peripherals, memory and interface features that you need to give you the best system performance at the lowest cost on a single FPGA.

### Hardware Acceleration using Fast Simplex Link

The MicroBlaze Fast Simplex Link (FSL) lets you connect hardware co-processors to accelerate time-critical algorithms. The FSL channels are dedicated uni-directional point-to-point data streaming interfaces. Each FSL channel provides a low latency interface to the processor pipeline making them ideal for extending the processor's execution unit with custom hardware accelerators.

### Floating-Point Unit Support

MicroBlaze introduces an integrated single precision, IEEE-754 compatible Floating Point Unit (FPU) option optimized for embedded applications such as industrial control, automotive, and office automation. The MicroBlaze FPU provides designers with a processor tailored to execute both integer and floating point operations.

### Hardware Configurability

The MicroBlaze processor solution provides a high level of configurability to tailor the processor sub-system to the exact needs of the target embedded application. Configurable features such as: the barrel shifter, divider, multiplier, instruction and data caches, FPU, FSL interfaces, hardware debug logic, and the hardware exceptions, provide great flexibility but does not add to the cost if they are not used.

### Award-Winning Platform Studio Tool Suite

The Embedded Development Kit (EDK) is an all encompassing solution for designing embedded programmable systems. This pre-configured kit includes the award-winning Platform Studio™ Tool Suite, the MicroBlaze soft processor core as well as all the documentation and soft peripheral IP that you require for designing FPGA-based embedded processor systems.



## Embedded Development Kit and Platform Studio Tool Suite

For development, Xilinx offers the Embedded Development Kit (EDK), which is the common design environment for both MicroBlaze and PowerPC-based embedded systems. The EDK is a set of microprocessor design tools and common software platforms, such as device drivers and protocol stacks. The EDK includes the Platform Studio tool suite, the MicroBlaze core, and a library of peripheral IP cores.

Using these tools, design engineers can define the processor subsystem hardware and configure the software platform, including generating a Board Support Package (BSP) for a variety of development boards. Platform Studio Software Development Kit (SDK) is based on the Eclipse open-source C development tool kit and includes a full-featured development environment and a feature-rich GUI debugger. The MicroBlaze processor is supported by the GNU compiler and debugger tools. The debugger connects the MicroBlaze via JTAG. For debugging visibility and control over the embedded system, design engineers can add the ChipScope Pro™ verification tools from Xilinx, which are integrated into the hardware/software debug capabilities of the EDK.

Device Family	Max Clock Frequency	Max Dhrystone 2.1 Performance
Virtex-4	200 MHz	166 DMIPS
Virtex-II Pro	170 MHz	138 DMIPS
Spartan-3	100 MHz	92 DMIPS

Note: The maximum clock frequency and maximum Dhrystone 2.1 performance benchmarks are not based on the same system. Depending on the configured options, the MicroBlaze soft processor core size is about 900 – 2600 LUTs.

## MicroBlaze Hardware Options and Configurable Blocks

### Hardware Functions

- Hardware Barrel Shifter
- Hardware Divider
- Machine Status Set and Clear Instructions
- Hardware Exception Support
- Pattern Compare Instructions
- Floating-Point Unit (FPU)
- Hardware Multiplier Enable
- Hardware Debug Logic

### Cache and Cache Interface

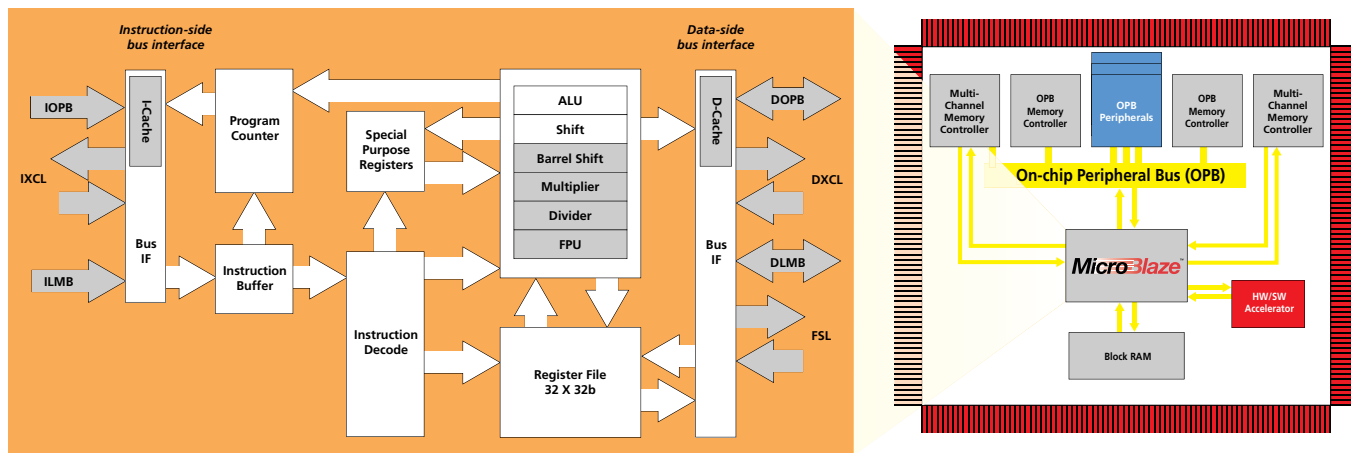
- Data Cache (D-Cache)
- Instruction Cache (I-Cache)
- Instruction-side Xilinx Cache Link (IXCL)
- Data-side Xilinx Cache Link (DXCL)

### Bus Infrastructure

- Data-side On-chip Peripheral Bus (DOPB)
- Instruction-side On-chip Peripheral Bus (IOPB)
- Data-side Local Memory Bus (DLMB)
- Instruction-side Local Memory Bus (ILMB)
- Fast Simplex Link (FSL)

## Take the Next Step

Visit our website [www.xilinx.com/microblaze](http://www.xilinx.com/microblaze) for more information .  
To order your Embedded Development Kit visit [www.xilinx.com/edk](http://www.xilinx.com/edk).



□ Basic Processor Functions    ■ Configurable Functions    ■ Designer Defined Blocks    ■ Peripherals – Xilinx or 3rd Party or Designer Defined

### Corporate Headquarters

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

### Distributed By:

FORTUNE 2005  
100 BEST COMPANIES TO WORK FOR

**XILINX**  
The Programmable Logic Company™

© 2006 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

# Develop Your Processing Skills

with Xilinx Education Services.



[www.xilinx.com/education](http://www.xilinx.com/education)

Take advantage of two specially designed embedded processing courses by Xilinx, featuring in-depth training, hands-on labs and much more. Use your Xilinx Training Credits or simply pay as you go.

#### **Course I — Embedded Systems Development (2 Days)**

For experienced FPGA designers, an introduction to developing embedded systems using hard (embedded IBM PowerPC™) or soft (MicroBlaze™) processor cores within the Embedded Development Kit (EDK) design environment.

#### **Course II — Advanced Features and Techniques of Embedded Systems Development (2 Days)**

Embedded systems developers will develop the necessary skills to architect complex embedded systems and improve every design by using the tools available in the EDK. They will also learn the advanced features of Xilinx Platform Studio™ to optimize system performance.

Visit [www.xilinx.com/education](http://www.xilinx.com/education) for more information and registration. Or call us at 1-877-XLX-CLASS.



# NOT ALL CODE IS HARD TO UNDERSTAND

```
/*
*****
/*      FPGA embedded challenge      */
*****
volatile project;
const    short schedule;
complex hardware, software;

if (FPGA && CPU) {
    success = seamless_FPGA(symbolic_debug,
                             early_HW_access, easy2use);
    performance = 1000 * faster;
    visibility++;
}
else {
    visibility = NULL;
    performance = SLOW;
    vacation = 0;
}

// evaluate it at www.seamlessFPGA.com
```

INTEGRATED SYSTEM DESIGN + DESIGN FOR MANUFACTURING + ELECTRONIC SYSTEM LEVEL DESIGN + FUNCTIONAL VERIFICATION

**Hardware/Software Integration** | You work late nights and weekends, but that can only get you so far when you have a huge device with logic, memory, processors, complex IP - and a tight schedule. How do you ensure that hardware/software integration goes smoothly? Seamless® FPGA from Mentor Graphics® is the only hardware/software integration tool that matches support for the Virtex FPGA family with models for the PowerPC and MicroBlaze processor architectures, giving you full software and hardware visibility and the performance you need to spot hardware/software integration issues early in the design cycle. To learn more about Seamless FPGA, go to [www.seamlessFPGA.com](http://www.seamlessFPGA.com), email [seamless\\_fpga@mentor.com](mailto:seamless_fpga@mentor.com) or call 800.547.3000.

**Mentor  
Graphics®**  
THE EDA TECHNOLOGY LEADER



# Xilinx is Embedded Processing



**D**iscover endless possibilities as Xilinx brings you hands-on, FREE workshops at ESC Silicon Valley. You'll gain valuable embedded design experience in real time. And only Xilinx Platform FPGAs offer programmable flexibility with the optimal choice of integrated hard and/or soft processors — that means fast and flexible solutions for your processing needs.

#### **FREE Hands-On Workshops (90 minutes each) Room C1**

- High Performance Design with the Embedded PowerPC and Co-Processor Code Accelerators *(April 4 – 10am, 2pm & 4pm)*
- Learn to Build and Optimize a MicroBlaze Soft Processor System in Minutes *(April 5 – 10am, Noon, 2pm & 4pm)*
- Design and Verify Video/Imaging Algorithms with the Xilinx Video Starter Kit *(April 6 – 9am, 11am & 1pm)*



#### **Live Demonstrations**

Explore embedded processing with the flexible PowerPC™ and MicroBlaze 32-bit processor cores, plus XtremeDSP performance and our PCI Express solutions. And see for yourself why the embedded industry has recognized award-winning products like Platform Studio and MicroBlaze.

Xilinx will also offer numerous Theatre Presentations and Technical Program Papers. Check in at booth #315 for a program schedule, and find out why Xilinx is embedded processing.

**Visit Xilinx at Embedded Systems Conference Silicon Valley, April 4–6, 2006 — Booth #315**

IN-DEPTH PRODUCT DEMONSTRATIONS & PRESENTATIONS | FREE HANDS-ON WORKSHOPS

