XCELL

Issue 29
Third Quarter
1998

THE QUARTERLY JOURNAL FOR XILINX PROGRAMMABLE LOGIC USERS

XILINX®
The Programmable Logic Company℠

Inside This Issue:

HDL VERIFICATION SPECIAL SECTION

A 19-page section that looks at a wide range of verification issues, including productivity, PCI designs, HDL simulation and more ...

See pages 22-40

PRODUCT INFORMATION

Chip-Scale Packaging

A new 48-lead, rugged ball grid array package ideally suited to today's portable and small form-factor applications...

See page 3

DEVELOPMENT SYSTEMS

Synplify Extends Timing Constraint

The timing of macrofunctions not synthesized in Synplify may now be characterized using SCOPE...

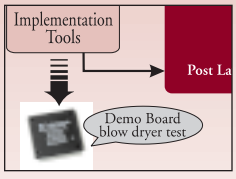
See pages 18-19

Marconi, S.p.A.: FPGA Design Cycle Time Reduced

Genova, Italy based Marconi, S.p.A. has created Synopsys scripts that can make the most advanced synthesis techniques simpler for FPGA users...

See pages 20-21

DESIGN TIPS & HINTS



The Best HDL Design Flow?

Benchmark tests on common designs prove that Xilinx has by far the most efficient HDL design flow...

See page 41

Moving Towards a Perfect World…

by Carlis Collins,
Managing Editor
of Corporate
Communications,
editor@xilinx.com

Software is the catalyst that subtly combines your mental creativity with a piece of physical silicon to produce a "living" representation of your thoughts, something that is both physical and non-physical, both matter and energy, heart and spirit. Isn't it a wonder? Perhaps with perfect software, and the perfect device, your thoughts could instantly manifest themselves into real working designs; there would be no limits to your creative potential.

In an imperfect world, the closest you can come to the ideal of instant gratification is through the use of Xilinx FPGAs and development tools. Designs are not instantaneous yet, but with each new generation of Xilinx silicon and software, your design capability increases and your design time decreases.

The version 1.5 release of the Xilinx Alliance Series and Foundation Series software moves you one more step closer to realizing the ideal creative environment. These tools now offer 50% faster compile times and 30% faster clock speeds for average designs, plus they support our new Virtex family, available in Q498, which will offer an unprecedented one million gate density. There has never been anything like this before.

Better software performance is always a good thing, and we are constantly improving our tools with new algorithms and processes. However, as you begin to approach million gate designs you'll need all the software "horsepower" you can get. So, we've added many new performance and productivity enhancing features to our version 1.5 software, smoothing the transition of our FPGAs into ASIC applications.

New AKA*speed* Technology

AKA*speed* is an array of new algorithms and new algorithmic strategies, combined with advanced new feature sets and applications that are optimized for higher performance, higher density designs. This new software technology, in both the Alliance Series 1.5 and Foundation Series 1.5 software, creates a performance-driven design environment, so you get the industry's highest performance for high-density HDL designs.

AKA*speed* includes many new features, plus enhancements to the existing technology such as timing-driven implementation, K-paths, advanced timing analysis algorithms, a robust constraints language, and incremental design capability. This is a powerful and easy to use combination that gives excellent results. AKA*speed* includes:

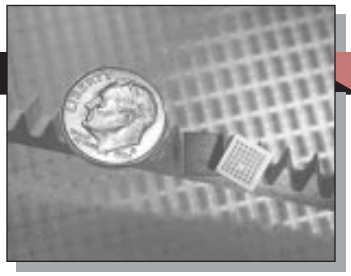
- A floorplanner that takes advantage of your knowledge of the structure of your design.
- The Xilinx CORE Generator to help you use our rapidly expanding family of cores from Xilinx and our AllianceCORE partners.
- A new Graphical Constraints Editor to help you achieve optimal results on the first pass.
- Two new design guides for the industry's leading HDL solutions.
- The ability to test your design under both best- and worst-case operating conditions.

Your Best HDL Solution

High Level Design Languages, such as VHDL and Verilog, are becoming more attractive as device densities increase. Without these high-level tools, it would be very difficult to design and debug the very large designs that now fit on a single FPGA. Our Alliance Series and Foundation Series software provide an easy-to-use HDL design environment that achieves the highest performance designs.

Though we keep making breakthroughs, there probably is no final "perfect solution," because the problem rapidly evolves. However, Xilinx aggressively pursues in-house development programs, and partnerships with all of the key EDA vendors, to keep you on the leading edge. There is no better way. ◆

XCell
Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124-3450
Phone: 408-559-7778
FAX: 408-879-4780



NEW

Chip-Scale Packaging:

Ideally Suited to Today's Portable and Small Form-Factor Applications

by Frank Toth, Marketing Manager for FastFLASH Products, toth@xilinx.com

Xilinx recently unveiled a new 48-lead chip-scale package (CSP) that offers all the benefits of an extremely small form factor in a rugged ball grid array package. This package is ideally suited for a growing number of applications where minimal board space and package thickness are important, such as portable and wireless designs, PCMCIA cards, PC boards, and PC add-in cards. Xilinx is the first non-memory manufacturer to have chip-scale packaging (CSP) technology available now.

Chip-scale packages are about 20% larger than the size of the die, with a ball pitch less than one millimeter. With these packages, the requirements for handling and lead coplanarity are greatly reduced because there are no fragile leads to bend. The package is also very thin (1.3 to 1.8mm) and light weight (0.17 gram) which makes it ideally suited for weight conscious portable applications like cell phones, hand held inventory and bar code reader systems, and personal digital assistants.

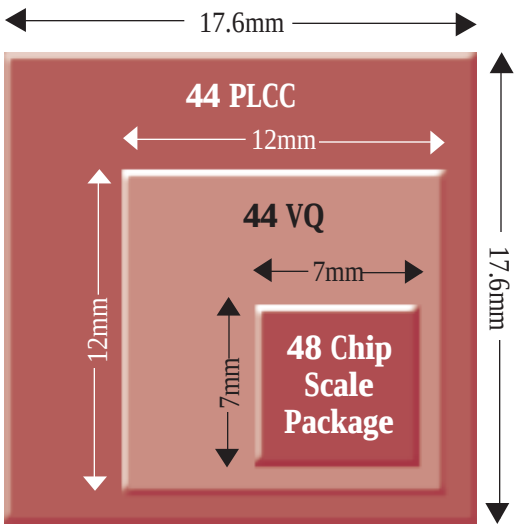


Figure 2

Figure 1 shows a side view of how the package is constructed. The die is mounted on top, bonded to the substrate surface, and wire bonded using industry-standard techniques. Thermal resistance of the package (Θ_{JA}) is 45.5°C per watt, which is comparable to the VQ44 package at about 42°C per watt. System manufacturers using CSPs can benefit from CPLD speed and cost improvements (die shrinks) without ever having to change package footprint, because die shrinks can be accommodated without having to change package dimensions.

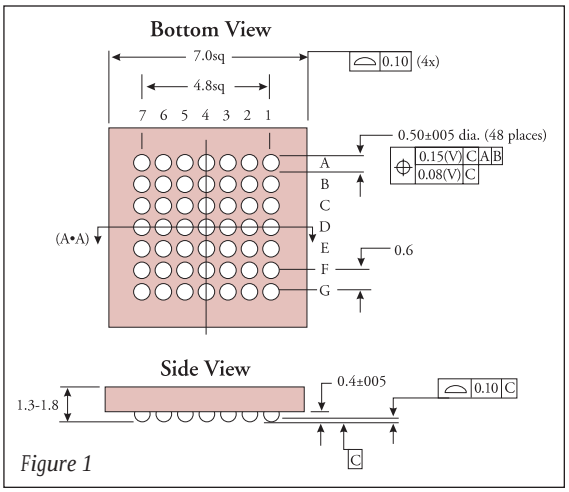


Figure 1

The XC9536 is the first XC9500 FastFLASH ISP family device offered in this package. It features 34 I/Os, full IEEE 1149.1 JTAG support, 10,000 program/erase cycles, 20 years of data retention, and unmatched logic flexibility with the industry's best CPLD pin-locking. The XC9536 CSP has 48 pins arranged in a seven-millimeter by seven-millimeter configuration using a 0.8 millimeter solder ball pitch. (Package technologies with a ball pitch greater than 0.8 millimeters are considered ball grid arrays and not CSP.) The footprint is three times smaller than the 44-pin very thin quad (TQ44) package and 40 percent smaller than the 48-pin thin quad (TQ48) package, as shown in **Figure 2**.

For more information on chip-scale packaging, visit WebLINX at: www.xilinx.com/products/csp.htm ◆

XILINX®

New Spartan -4 Devices for High-Speed Applications

by Marc Baker, Xilinx Applications Engineer, marc.baker@xilinx.com

The low-cost Spartan FPGA family is excellent for high-volume consumer applications such as PCs, digital cameras, set-top boxes, and DVD equipment. Now, with the introduction of the new Spartan -4 speed grade, the family can be used in very high-performance applications as well, delivering on the promise of being the industry's no-compromise ASIC-replacement FPGAs.

For example, the two largest Spartan devices can now meet the stringent requirements of 33 MHz PCI applications with no wait states, making this the first low-cost FPGA family to provide such high performance, with many applications running at beyond 80 MHz.

Last year, 76% of ASIC design starts required 80 MHz speed or less. With an I/O speed beyond 100 MHz for the smaller members of the family, and internal frequencies that can go much higher, the Spartan devices easily meet this requirement.

Spartan –4 Performance

The new Spartan -4 speed grade is approximately 25% faster than the original Spartan -3. This makes it comparable to the XC4000E-1, and faster than any competitor's 5V FPGAs. I/O frequency is commonly used as a performance benchmark, because it indicates how fast two devices can "talk" with each other. I/O frequency is measured by taking the inverse of the pin-to-pin clock-to-out and setup delays. The Spartan -4 excels in this respect, with an I/O frequency of 92 MHz for the XCS30; this is 12% faster than the comparable XC4013E-1, at 82 MHz. For the two smaller Spartan devices, the XCS05 and XCS10, the I/O frequency surpasses 100 MHz *(see **Figure 1**)*.

Spartan is a full-featured family with on-chip SelectRAM memory and 238 to 1836 logic cells (5K to 40K system gates). It takes advantage of a unique process that utilizes 0.5 micron technology for transistors and 0.35 micron technology for interconnect. This provides the smallest and fastest logic, using a 5V supply.

"...the two largest Spartan devices can now meet the stringent requirements of 33 MHz PCI applications with no wait states..."

Spartan –4 Speed Files

The -4 speed files were generated with a new characterization methodology that more accurately models the worst-case delays in the silicon while reducing the test cost. The result is a very fast, yet low-cost solution. By applying this new methodology to the volume production units now being processed for the Spartan family, the Spartan -3 speed grade was simplified and overall performance was improved as well.

To use the new –4 speed grade, download the speed files from WebLINX, the Xilinx website, at http://www.xilinx.com/techdocs/htm_index/sw_M1.4_alliance.htm. The initial Spartan support in the Xilinx software (Alliance Series and Foundation Series version 1.4) includes a placeholder for the -4 speed files, making it easy to drop in the new files; installation instructions are included in the "readme" file. The production versions of the Xilinx Alliance Series and Foundation Series version 1.5 software will include the new speed files. If you are using the beta version of the 1.5 release, make sure you have the correct speed files.

"Spartan is a full-featured family with on-chip SelectRAM memory and 238 to 1836 logic cells (5K to 40K system gates)."

Spartan Speed Designator

The Spartan series uses a new speed designator that starts at an arbitrary number and increases for higher speed. Thus, the Spartan -4 is faster than the Spartan -3, but the "-4" doesn't indicate any relative speed against other products. All future Xilinx products will use this new nomenclature, which is similar to speed designators in the ASIC world. This avoids the problems of having two-digit speed grades. It also avoids the potential for basing performance assumptions on a single specification.

For example, most Xilinx devices use the delay through the Look-Up Table (LUT) for the speed designation. The Spartan -4 has a 1.2 ns LUT delay, the fastest of any Xilinx 5-volt FPGA, equal to the XC4000XL-09 and faster than the XC4000E-1 (1.3 ns). However, the Spartan -4 overall performance is typically slower than the XC4000XL-09, and comparable to the XC4000E-1 *(see **Figure 2**)*.

The lookup table delay, being just one small part of any given path in a design, does not accurately reflect overall performance, and does not work well as a general method of comparing speeds. Overall design speed depends on the function being implemented and the routing delays in the critical path. Those routing delays can be as short as 0 ns, especially when using carry logic. As an example, a 16-bit counter runs at 96 MHz in the XCS30-4. The Xilinx Timing Analyzer and third-party simulation tools will report worst-case delays with a 0.1 ns resolution.

Spartan Core Support

The new Xilinx PCI32 Spartan master and slave interface further reduces the cost of implementing a programmable PCI solution (*see the related article on p 14)*. Xilinx also offers several DSP-related cores for the Spartan architecture, along with AllianceCORE solutions from other vendors. All of these pre-defined solutions can immediately take advantage of the higher speed offered by the Spartan -4 speed files. Download the new files today to take advantage of the fastest low-cost FPGAs available.

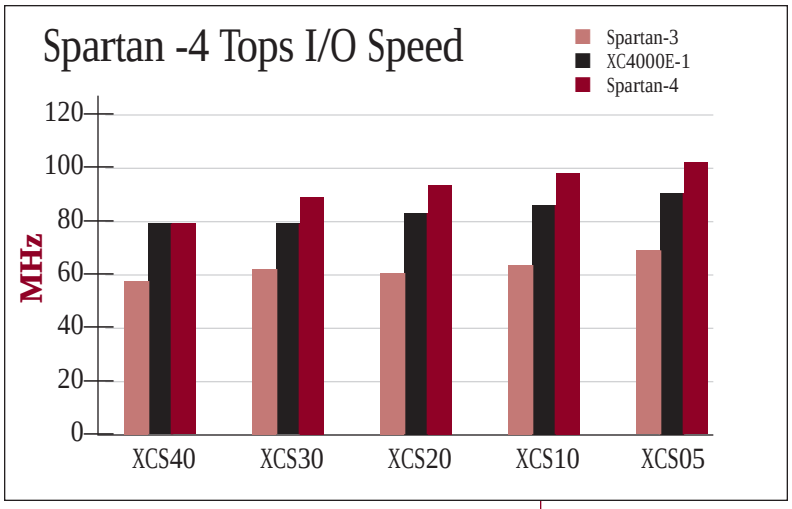


Figure 1

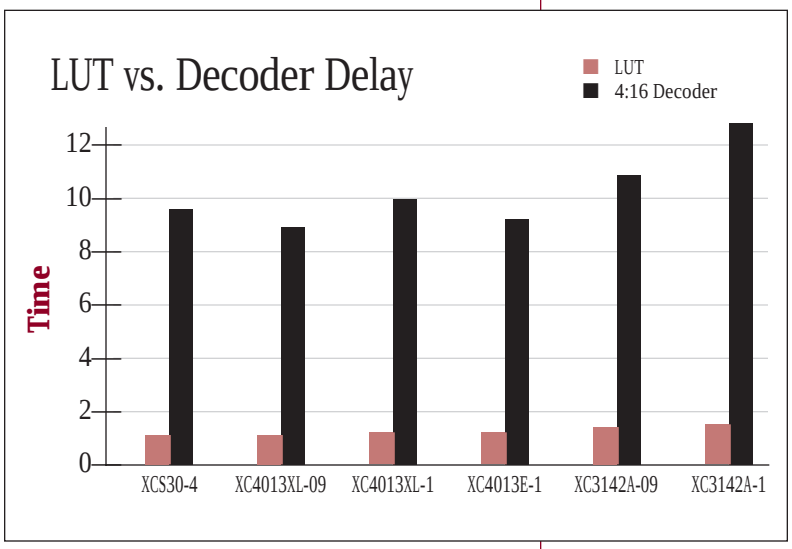


Figure 2

You can download the latest datasheet from WebLINX at http://www.xilinx.com/partinfo/spartan.pdf. Note that as with all other Xilinx FPGAs, the datasheet provides the guaranteed worst-case pin-to-pin setup and hold times, which are not reflected in the speed files or timing reports.

Spartan –4 devices are available on distributor shelves today, for all densities and packages. ***Contact your local salesperson or distributor for pricing and lead-times.*** ◆

The New XC95144

by John Ahn, CPLD
Product Manager,
john.ahn@xilinx.com

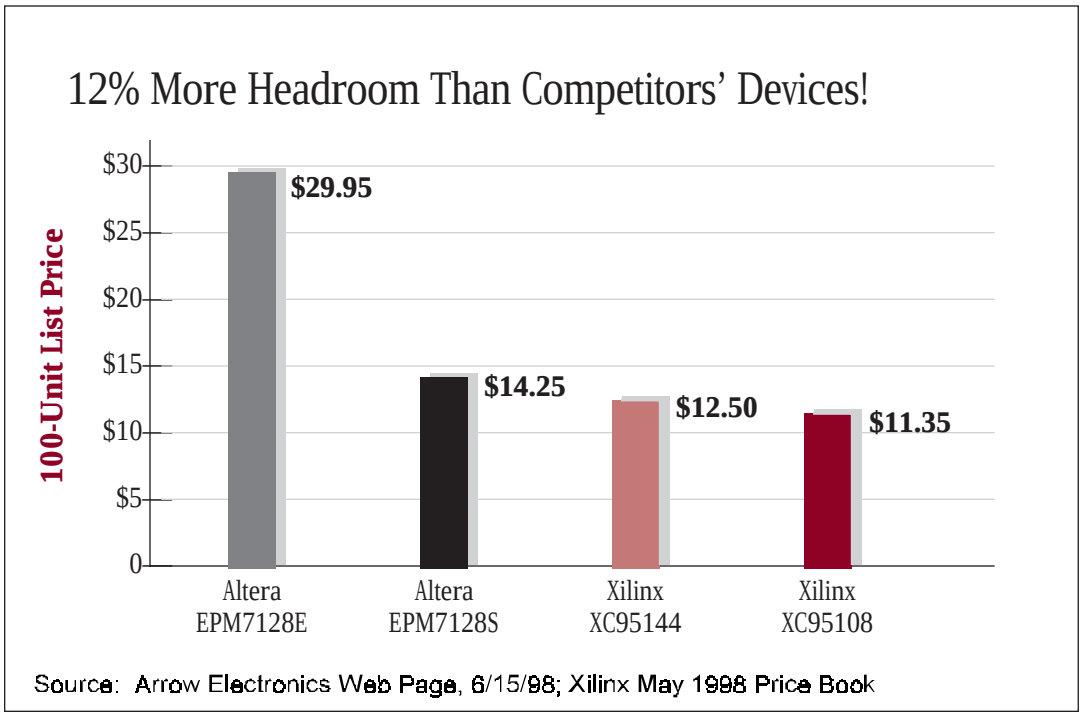
The FastFLASH family of CPLDs just got better with the recent introduction of the XC95144. This newest member of the XC9500 family completes the fastest growing line of CPLDs in the industry. The XC95144 features 144 macrocells with 7.5 ns pin-to-pin delays and is offered in 100-pin TQFP, 100-pin PQFP, and 160-pin PQFP packages.

The XC95144 is the first CPLD to use the new advanced FastFLASH process from United Semiconductor Corporation in Taiwan. This new 0.5-micron process technology offers up to a 50% die size reduction from the previous 0.6-micron process. The rest of the XC9500 family will be transitioned over to the new process by the end of 1998.

Higher Density at a Lower Price

The 50% die size reduction enables Xilinx to maintain its CPLD price leadership into the future. As you can see, the XC95144 offers 12% more capacity than the competitors' conventional 128-macrocell device at a 62% lower price *(see chart below)*.

The new XC95144 is a superior CPLD solution at a rock bottom price. ◆



Xilinx Unveils New "Q*PRO*" Products

For aerospace, defense, and high reliability markets

by Howard Bogrow, Xilinx Marketing Manager for Hi-Rel Products, howard.bogrow@xilinx.com

Xilinx recently announced the new Q*PRO* line of QML-certified programmable logic devices designed to meet the evolving requirements of the aerospace and defense markets, supporting a procurement trend toward products built using the best commercial practices. Xilinx is one of only 20 semiconductor suppliers in the world to receive Qualified Manufacturer Listing, or QML.

As a QML supplier, Xilinx can quickly introduce state-of-the-art, military-grade products because our world-class processes and materials have been qualified in advance, eliminating the need to qualify individual products or production lots. Xilinx can now offer the latest, highest-performance products to both commercial and military-aerospace customers, at the same time; the usual 12-to-18 month delay for military qualification is eliminated.

Q*PRO* FPGAs are guaranteed to operate in extended temperature and rugged environments. For example, N-grade plastic and M-grade ceramic packages are rated for operating environments ranging from –55°C to +125°C. Packaging options include thermally enhanced plastic quad flat packs and ball grid arrays as well as hermetic pin grid arrays and top-brazed ceramic quad flat packs.

The Q*PRO* solutions include the Xilinx XQ4000X FPGAs, which are available as military temperature range ceramic and plastic encapsulated devices that deliver densities up to 130,000 system gates. The Q*PRO* line is also supported by the Xilinx Foundation Series and Alliance Series software, and a variety of verified software cores that can be used to create system-level functions such as standard bus interfaces.

"Our new Q*PRO* offerings continue a commitment by Xilinx to serve the changing needs of these specialized markets, and they cap our recent efforts of winning full QML status," said Rick Padovani, director of the Xilinx aerospace and defense business. "Xilinx is leading the logic industry in delivering off-the-shelf, commercial products that are cost-effective and meet the exacting standards of performance and reliability in rugged environments that our aerospace and defense customers demand. Moreover, Xilinx provides reliability of supply and a superior alternative to traditional ASICs for customers designing new systems or upgrading older equipment."

> *"Xilinx provides reliability of supply and a superior alternative to traditional ASICs"*

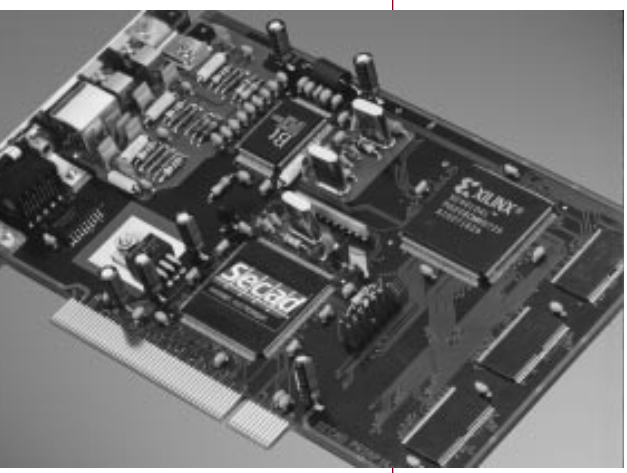
Xilinx has been serving the military and aerospace market for more than a decade and provides customers with continuity of supply, specialized products, and guaranteed mask and process controls that are unmatched by any other programmable logic manufacturer. The Q*PRO* brand name reflects the Xilinx commitment to QML, **P**erformance, **R**eliability of supply, and **O**ff-the-shelf products. ◆

PCI Reconfigurable Image Advanced Processor (PRIAP)

by Denis Rousseau,
SECAD Product Manager,
secad38@compuserve.com

SECAD, a French company specializing in Xilinx FPGA design, real-time image-processing board design and manufacturing, recently designed a PCI-based Reconfigurable Image Advanced Processor (PRIAP) using the XC4010XL. This board includes a color or B/W video decoder and three 16-Mbit memory planes, expandable to 64Mbit. These memory planes and the output of the decoder are accessed by the FPGA simultaneously, allowing real-time computation with four data flows. The board includes a PCI interface and an 80 MHz 24-bit DSP, which can be devoted to data communication between the memory planes and the PCI interface.

The first application, developed for the Viewpoint company, is for real-time target tracking. Other possible real-time computations are:



- 2D or time filtering
- Mathematical morphology
- Correlation
- Thresholding
- Movement detection
- Any high-performance, reconfigurable computing, with or without image processing

SECAD chose the Xilinx 4000XL family for a number of reasons:

- SECAD has used Xilinx for more than 10 years in all of their image processing boards.
- Different densities are available within the PQ208 footprint, from 4005XL to 4044XL, allowing different processing capabilities without board redesign.
- The Foundation Series software is easy to use with the Metamor VHDL compiler, the ALDEC simulator, and other tools from Xilinx like X-BLOX.

The only problem at the beginning of the design was the use of the beta version of the software, running under Windows95, which caused some problems. Also it was the first design we have done in VHDL, and we had to learn a lot about it. VHDL allows us to reuse most of our design, and we will develop a VHDL library for this board, allowing us to rapidly develop new designs.

We do not plan to use a HardWire version, because, with the RAM-based version, we can program new algorithms without changing the board. A HardWire version will be interesting only if we find an OEM customer for 1K or 10K/year quantity, with a fixed image-processing algorithm need.

The SECAD company was founded in 1983. It has 19 employees, and its main activities are:

- Electronic board and systems design, and manufacturing for industry OEM market.
- Image processing board design and manufacturing for OEMs.
- Design and manufacturing of proprietary imaging products.
- Turn-key image processing applications for industry, military, medical.
- Xilinx FPGA development and service.
- VHDL and Xilinx FPGA training.
- Providing FPGA design expertise.
- FPGA and CPLD development, using VHDL or schematics.
- Complete board development, and manufacturing.

For information on SECAD, call: +33 4 76 33 05 21, or fax : +33 4 76 33 05 56, or e-mail secad38@compuserve.com ◆

SPARTAN™ FPGA

ESAOTE BIOMEDICA

A Spartan Success Story

by Marc Baker, Xilinx Applications Engineer, marc.baker@xilinx.com

When the designers at Esaote Biomedica (Genoa, Italy) were looking for a logic solution that provided dual-port RAM, they found that only the Spartan Series of FPGAs from Xilinx met their cost requirements. Esaote was able to begin their design even before the production devices were available, because they were one of the first to receive the Alliance Series version 1.4 development system and Spartan device samples. As a result, Esaote placed the first volume order for Spartan devices.

The Esaote Florence R&D team is developing the next-generation of diagnostic ultrasound equipment. The target is a portable ultrasound scanner that is light, small, easy-to-use, and fast to produce results. The core of the application needs to process a high volume of data very fast, using dual-port RAM, at low power consumption and low cost. This is a perfect application for the Spartan family.

Esaote originally considered the XC4000XL FPGA family, taking advantage of the on-chip Select-RAM and high speed. True dual-port RAM was built by using the built-in dual-port read capability of the Xilinx Select-RAM and then adding a second block of RAM for dual-port write. However, the resulting implementation did not meet cost targets. In October 1997, the Xilinx representative firm working with Esaote, Silverstar-Celdis, presented advance information on the Spartan Series. The no-compromises Spartan family met the technical, performance, and cost requirements of the system. The Spartan solution was more cost-effective than even an ASIC alternative.

Esaote was able to prototype their design using the 5-V Spartan XCS20-3TQ144 samples. For implementation software, they used Alliance 1.4, which provided software support even before the Spartan announcement. Esaote engineers also used the beta version of the Xilinx CORE Generator to create some of their DSP functions.

Esaote has always paid careful attention to the issues associated with the cost of health care, as demonstrated by its cost-effective family of products. In this design, there are 32 Spartan devices per board, and two boards per system. With Esaote expecting to build 1,000 systems, the total Spartan usage is 64,000 devices. The Spartan family is what made this low-cost product feasible.

Views from Esaote Biomedia's manufacturing facilities.

Esaote Background

The Esaote Group designs, manufactures, markets, and services non-invasive diagnostic medical imaging systems and specialty medical monitoring equipment worldwide. The Esaote Group is the leading European manufacturer of diagnostic ultrasound equipment and the world leader in Dedicated Magnetic Resonance Imaging. In Italy the Esaote Group is the leading provider of electronic diagnostic medical equipment. Group headquarters are located in Genoa, Italy.

Esaote's technologically-advanced imaging products include a broad line of diagnostic ultrasound machines and an innovative Magnetic Resonance Imaging (MRI) system designed specifically to scan joints and extremities (Artoscan). The non-imaging products include electrocardiograph (EKG) and electroencephalograph (EEG) diagnostic monitors.

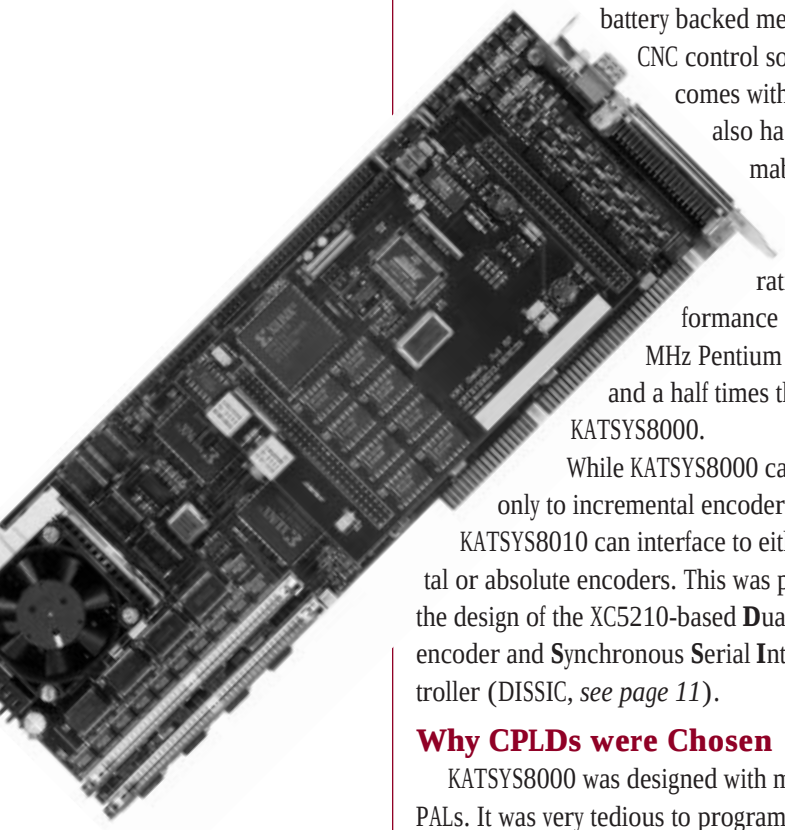
For more information on Esaote see their website at www.esaote.com ◆

The KATSYS8010 CNC

by T.S.N. Murthy, Principal Design Engineer, KAT GmbH, tsnm@giaspn01.vsnl.net.ni

KATSYS8010 is the second in the series of high-performance CNC controllers from KAT GmbH, Bremen, Germany. The first product, KATSYS8000, was designed in 1993. KATSYS8000 was a multiprocessor solution, integrated into an industrial PC. The CNC machine can be controlled via either digital drives, interfaced over the **SE**rial **R**ealtime **CO**mmunication **S**ystem (SERCOS), or via analog drives.

KATSYS8010, a compact version with a Pentium 200 MHz processor, is a PC plug-in CNC controller. KATSYS8010 is also configurable, providing either four axes of analog control or digital control via SERCOS standards. The CNC programs and data are downloaded from the PC, which is stored in battery backed memory. The CNC control software that comes with KATSYS8010 also has a programmable mask generator for user configuration. The performance of the 200 MHz Pentium is about two and a half times that of KATSYS8000.



While KATSYS8000 can interface only to incremental encoders, KATSYS8010 can interface to either incremental or absolute encoders. This was possible due to the design of the XC5210-based **D**ual **I**ncremental encoder and **S**ynchronous **S**erial **I**nterface **C**ontroller (DISSIC, *see page 11*).

Why CPLDs were Chosen

KATSYS8000 was designed with more than 40 PALs. It was very tedious to program, label, and insert these into sockets. So, the first design goal for KATSYS8010 was to remove this problem. Accordingly, the complete logic was split into four different blocks and each of these were initially targeted for XC3100A series FPGAs, mainly due to the need for 50 MHz bus speeds. Soon, it was discovered that pin-locking was becoming a major issue. CG-CoreEl, of Pune, India, then recommended the use of CPLDs. The designs were then redesigned and targeted to XC9572 CPLDs which have excellent pin-locking capability.

The CPLDs provided In-System Programmability, no loss of components due to design revisions, and greatly reduced manufacturing times. All the design work was carried out in India with the support of CG-CoreEl, Pune, India, the local representatives. Now, KAT GmbH uses Xilinx CPLDs in all current and future designs.

The KATSYS8010 Design

Though there are four CPLDs in any particular configuration, five CPLDs were actually designed. The functions of these CPLDs are as follows:

- **PC Interface CPLD**: All the necessary interface circuitry for accessing the memory- and PC-related I/O from the ISA bus.
- **RAM Control CPLD**: The battery-backed RAM is dual-ported to both PC and Pentium. Address decoding, arbitration, command, and bus-multiplexer control signals were implemented in this CPLD.
- **I/O Control CPLD**: All the I/O devices like interrupt controllers, timers, 24V I/Os, serial communication ports, watchdog timers, and system ID are controlled by this CPLD.
- **Axes Control CPLD**: All the timing and control needed for interfacing to analog devices such as ADCs, DACs, DISSIC, watch dog timer, module ID, are generated by this CPLD.
- **SERCOS Control CPLD**: All the timing and control needed for interfacing to SERCOS controller, auxiliary ADC, 64-bit to 16-bit data bus steering, are generated by this CPLD.

Controller *from KAT GmbH*

"The XC9500 CPLDs pack in so much logic, that it has eliminated a lot of PALs and GALs and greatly reduced the power consumption. The Xilinx software is very reliable - all the five designs worked right the first time. And, four of the five CPLDs were running with a 50 MHz clock. Thanks to the accurate simulation results of the Xilinx software" said TSN Murthy, principal design engineer.

"The Xilinx FastFlash CPLD technology is so elegant, that we want to use only Xilinx CPLDs for all our future logic designs" said Ulrich Schulz, project leader.

End Use

KATSYS8010 can be used in all types of CNC machines. In order to apply KATSYS8010 in diverse machines, KAT supplies user-configurable software to suit the machine and its configuration. ◆

KAT GmbH *Using Xilinx XC5210 FPGAs For a Dual Incremental Encoder and Synchronous Serial Interface Controller (DISSIC)*

KAT GmbH recently used the Xilinx XC5210-PC84 FPGAs to create a Dual Incremental Encoder and Synchronous Serial Interface Controller. DISSIC interfaces to two encoder types — **incremental** and **absolute**.

The incremental encoder interface takes in A, B, and R pulses from an encoder, filters the spikes, decodes, and counts. The up/down counter is 32 bits wide and can be preloaded with a user value for a reference position. The position value in the counter can be stored into a register either via a software command or a hardware signal. Other registers can be used to store the counter values on the activation of some external signals.

The incremental encoder interface can also be used for digitizing. In this case one axis moves to a target position, while the other axes store the position of their respective counters. The maximum input frequency of the A, B pulses is around 950 KHz.

The absolute encoder interface implements the Synchronous Serial Interface. DISSIC provides the clock for the position data transfer; the position data can be 32 bits wide. When the position data is less than 32 bits, the remaining bits can be used to receive information such as parity, power fail, or other check bits. The position data can be located anywhere within the 32 bit data and can be coded in either binary or gray codes. The data transfer can be initiated either by a software command or a hardware signal. Maximum clock frequency is 400 KHz.

DISSIC has a 16 bit, Intel-style microprocessor interface and is available in two implementation versions — V2.0 has one absolute and two incremental encoder interfaces, and V2.1 which has two absolute encoder interfaces.

DISSIC was the first Xilinx FPGA design for KAT GmbH, and they were very pleased with the performance of the XC5210-PC84 FPGAs. ◆

> *"DISSIC was the first Xilinx FPGA design for KAT GmbH, and they were very pleased with the performance of the XC5210-PC84 FPGAs."*

The Core Story: *A Breakthrough*

This year, Xilinx will ship products with unprecedented logic density. Our FPGAs already offer you 250K system gates, an order of magnitude greater than what was available just a few years ago. Devices twice as large are expected around mid-year, and our first million-gate FPGA will be sampling by the end of 1998.

How will you create the logic to fill these huge devices?

It's obvious that designing one gate at a time is not going to work. The answer lies with intellectual property (IP), or cores, which are predefined system functions, used for nearly a decade by designers of traditional mask-programmed custom ASICs.

It's only recently that cores have started making inroads into FPGA designs. Three main reasons account for this new migration. First, FPGAs are now large enough to accommodate cores, and a surprisingly large and diverse library of compatible cores is coming into existence. These cores include functions such as PCI and PCMCIA bus interfaces, digital signal processing algorithms, RISC microprocessors, standard peripheral controllers, and asynchronous transfer mode (ATM) functions. The expansive logic resources of our FPGAs, coupled with cores, allow you to create system-level designs on a single chip.

"The great attraction of cores is that they allow you to quickly and reliably create the most difficult sections of your designs. For small designs, cores are a welcome convenience. But for larger designs, they are becoming a necessity."

The great attraction of cores is that they allow you to quickly and reliably create the most difficult sections of your designs. For small designs, cores are a welcome convenience. But for larger designs, they are becoming a necessity.

For example, one of our customers, a forward-looking company, created an embedded application that combines a RISC processor core with several DSP core functions, communicating over a PCI bus whose interface logic was also created by a core. This would have been a monumental undertaking if designed from scratch.

New tools are already on the horizon that will make the task of grouping multiple cores on a single FPGA even easier. These tools will allow you to pull cores from a common library and place them at predetermined locations to make the most efficient use of resources and achieve your performance requirements.

A second reason cores are coming to the FPGA marketplace is performance. The latest generation of FPGAs operate at system speeds in the 80—100MHz range, and they will soon exceed 150MHz, fast enough to handle 66 MHz PCI 64 or communications protocols such as a 155Mbps synchronous optical network (Sonet). These functions are significant design challenges in themselves, and it's a great benefit to you if the functionality is available as a core.

Third, intellectual property developers have traditionally built their cores around standard high-level description languages such as VHDL and Verilog, the mainstream tools of ASIC designers that provide a large degree of flexibility. These languages are now becoming the basis for more and more PLD tools, and that is one of the many things attracting IP developers to the programmable logic market. In fact, developers are discovering that FPGAs are excellent prototyping vehicles for cores.

Developers can silicon-test their designs directly on programmable logic devices and polish the code much quicker and with less expense than they could by going through an ASIC vendor

in Time to Market

by Rich Sevcik, Senior Vice President of Software, Xilinx

and lining up a customer as a development partner. Our SRAM-based PLDs permit core designers to "rewire" the devices immediately by reprogramming them with new designs.

Moreover, the growing use of FPGAs presents independent IP developers, and programmable logic vendors themselves, with a "mass market" for their products. The worldwide universe of programmable logic customers numbers in the tens of thousands, compared to several hundred very large companies that buy IP for their high-volume mask-programmed ASICs.

Cores are also helping you answer the classic "make or buy" question. For example, as ubiquitous as it is today, the PCI interface remains a complex standard, rife with timing-critical specifications. Making a PCI interface from scratch can add from six to nine engineering months to a design. Buying it, on the other hand, can mean a substantial saving of time and money, especially for PCI designs where engineering time, cost, and volume may not justify going to a traditional ASIC solution. Additionally, buying cores frees you to concentrate on the intellectual value they add – beyond the basic PCI bus interface – to the product that's on the drawing board.

The Xilinx LogiCORE PCI interface illustrates one model of how cores are being delivered in the FPGA market today. Our PCI offering consists of pre-defined functions (target and initiator) that allow you to create a complete PCI interface on a single FPGA, and still have ample logic remaining to create the unique back-end interface required for your application.

Two points illustrate why the market has quickly accepted a product like the LogiCORE PCI core (more than 300 electronic equipment manufacturers have licensed it to date). First, the design is pre-verified and tested, ensuring that it will comply with the rigorous PCI specification. Second, it has a pre-defined layout, and it's optimized for our FPGA architecture. Therefore, timing for critical paths is fixed, ensuring predictable and consistent performance.

Cores are flowing from a growing community of independent IP developers who are coming to realize that their cores must be tuned for a particular device architecture using the tools designed to program that device. Power consumption, performance, and core predictability vary considerably based on differences in the FPGA vendors' place-and-route tools, device interconnect structure, and on-chip memory resources.

"Cores are flowing from a growing community of independent IP developers who are coming to realize that their cores must be tuned for a particular device architecture using the tools designed to program that device."

Design verification and device optimization are critical elements for the success of PLD cores, whether they are sold and supported by device suppliers or by IP developers. In fact, independent IP developers are beginning to align themselves closely with programmable logic vendors in order to accomplish this. Xilinx, for example, has partnered with nearly two dozen IP providers worldwide through its AllianceCORE program, and expects to expand the number of partners in the program this year. Such partnerships help to ensure that cores will reach you only after they are verified and optimized, and only when a strong support system is in place.

The momentum is clearly behind PLD cores, and during 1998 you can expect to see significant new developments in this segment of the market. Larger and faster devices, new FPGA architectures, powerful tools, and targeted IP offerings are shaping up to combine cores and FPGAs into true system-level solutions. ◆

The Low-Cost PCI Solution

by Per Holmberg,
LogiCORE Product Manager,
per@xilinx.com

Xilinx provides the most cost-effective and highest-performance PCI solution in the market by leveraging the flexibility of Xilinx FPGAs. We make PCI easy to design by providing a complete solution of proven cores, intuitive development tools, and comprehensive support.

Why Xilinx PCI?

By integrating a fully compliant PCI interface with an application-specific back-end design into one FPGA, you can achieve higher integration and higher performance than other PCI solutions. The flexibility of Xilinx FPGAs makes it possible to update the PCI board, using software alone, in development or in the field. This significantly reduces your design risk and cuts development time.

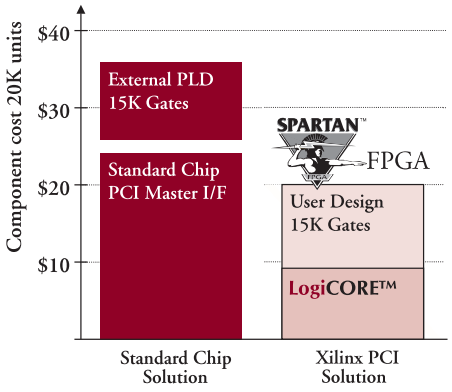


Figure 1: The Xilinx Cost Advantage

Furthermore, the Xilinx PCI solution can be customized for a specific application and, as a result, the highest possible performance can be achieved. Xilinx high-speed FPGAs support zero wait-state burst operations and by integrating scalable, dual-port FIFOs on the chip, our customers have achieved a sustained bandwidth of up to 132 Mbytes per second (the theoretical maximum for a 32-bit, 33MHz PCI interface).

PCI32 XC4000 Devices– The high-performance PCI solution

This solution integrates a PCI interface with up to 124,000 system gates. The core supports zero wait-state burst operations and a sustained bandwidth of up to 132 Mbytes per second.

PCI32 Spartan – The low-cost PCI solution

This solution integrates a PCI interface plus up to 30,000 system gates at a price below standard chip solutions.

To minimize the learning curve and simplify the design process, Xilinx provides fully proven and predictable PCI cores (LogiCORE PCI) that can be integrated into your design using the standard Xilinx implementation tools. LogiCORE PCI products use Xilinx Smart-IP technology, are easily configured and downloaded with an intuitive user interface from WebLINX (the Xilinx website), and come with VHDL and Verilog simulation models and testbenches.

Xilinx and its partners can also provide reference design examples, prototyping boards, PCI drivers, driver development tools, and design services.

Conclusion

Because Xilinx FPGAs integrate the PCI interface plus 15,000 to 124,000 user gates, no external PLD is required for glue logic. The result is a highly integrated, flexible, one-chip, PCI solution at a lower cost than most standard PCI chip sets.

For more information, visit WebLINX at: www.xilinx.com/products/logicore/pci/pci_sol.htm ◆

New XC9500 CORE Support

by Dave Grace, CPLD Software Product Manager,
dave.grace@xilinx.com

Xilinx CPLDs offer you a number of advantages such as high system clock speeds, short pin-to-pin delays, industry standard JTAG support, and non-volatile FLASH-based In-System Programmability (ISP). Xilinx XC9500 CPLDs are perfect for integrating many off-the-shelf ICs, and because you can easily customize or enhance any design, CPLDs offer you the ability to optimize your system for speed, density, cost, or all three. Now, you also have the time-to-market and ease-of-use advantages of intellectual property (cores) as well.

Full-Featured Xilinx CPLD Starter Kit for $99.00 from Insight Electronics

by Chris Skipworth, chris_skipworth@ins.memec.com

Insight Electronics is offering a new, low-cost CPLD development system. The Xilinx CPLD Starter Kit includes Xilinx Foundation Series software, an ISP/JTAG download cable, and an XC9536 demo board; everything you need to easily create and test high-performance XC9500 designs, within minutes of opening the box.

The XC9500 family is the industry's most advanced CPLD product, offering 5-ns pin-to-pin speeds, full IEEE 1149.1 JTAG support, ultra-reliable pin-locking, and advanced surface mount packaging. In addition, the XC9500 family commands the lowest cost per macro cell in the industry, due to the advanced Xilinx proprietary FastFLASH technology.

The CPLD-Starter Kit is a very inexpensive way for you to begin designing with low-density Xilinx products. This full-featured development system provides everything you need including simulation capability and a hardware download cable for in-system programming and real time logic debugging. You can enter designs via schematic capture, state diagrams, and high-level description language (HDL), plus the kit can be upgraded to the industry standard VHDL for use with higher density designs.

CPLD Starter Kit Features

- Foundation Series Base V1.4 software
- Device support for all XC9500 CPLD products
- XC9500 demo board
- Parallel download cable for in-system programming
- XC9500 product description sheet
- CPLD application guide
- XC9500 example design
- Upgrade to full VHDL system for just $390 (special offer)
- Memec Design Services information
- Support for Xilinx lower density FPGAs (XC4000E/X up to XC4010E/X, and Spartan)
- Price: $99.00

Hardware Requirements

- Windows 95- or Windows NT 4.0-compatible PC
- 32MB RAM, minimum

This starter kit is ideal if you want to create low-density designs, quickly, efficiently, and cost effectively.

Contact your nearest Insight-Electronics Sales Office to purchase the CPLD-Starter Kit at 1-800-677-7716 (USA), or 1-800-204-0010 (Canada). ◆

There are three ways you can take advantage of XC9500 devices using COREs:

- **LogiBLOX** - The XC9500 family is now fully supported by the LogiBLOX module generator in the new Alliance Series and Foundation Series 1.5 release, shipping this summer.
- **AllianceCORE** - The XC9500 family is also supported under the Xilinx AllianceCORE program. Core designs for UARTs, MicroProgram Controllers, Peripheral Interface Controllers, DRAM Controllers, and Synchronous DRAM Controllers are available today.
- **Xilinx CORE Generator** - XC9500 functionality and support will be incorporated into the Xilinx CORE Generator in the first half of 1999.

Simplifying the design process through the use of proven, high-performance cores gives you significant benefits. Now, Xilinx adds the XC9500 ISP CPLD family to the list of core-compatible device architectures that help you achieve higher performance results with significant reductions in design time. ◆

DSP Design Tools
for Xilinx FPGAs

by Nick Lethaby, Director of Business Development, Elanix, Inc., nick@elanix.com

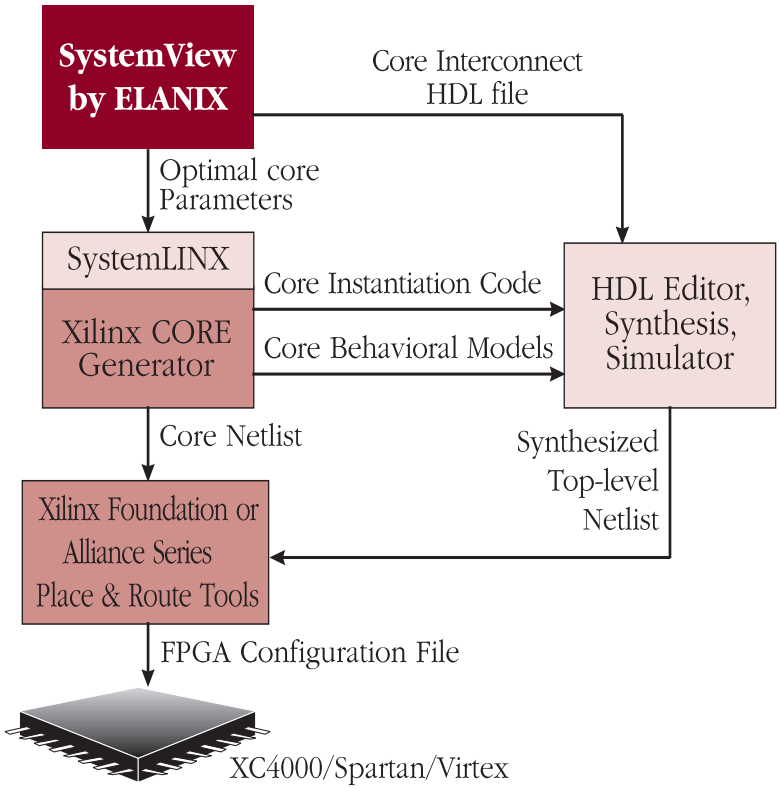
Over the past year, Xilinx has simplified the task of implementing DSP functions in FPGAs through the release of its optimized DSP LogiCOREs and its CORE Generator technology. Many users have experienced significant gains in productivity when using these DSP cores, because they no longer have to implement functions such as FIR filters from the ground up.

In a core-based design methodology, an efficient implementation is dependent on identifying the optimal parameters for a core. For example, a core that is programmed to use a 16-bit data path will use many more gates than a core employing only a 9-bit data path. Furthermore, even without bit-width optimization, DSP function design already involves significant complexity such as calculating the number of taps or determining coefficient values for a filter. Without access to a tool specifically designed for optimizing DSP functions, it may be extremely difficult to develop an efficient final silicon implementation.

System-level design tools that support the design of DSP functions have existed for some time. However, the traditional offerings in this arena have had a number of drawbacks. Some system design tools only effectively enable DSP function development using single- or double-precision floating point arithmetic. The digital designer is left with the difficult task of converting this design into fixed-point integer arithmetic using as few bits as possible. Although some tools have supported fixed-point and bit-width optimization, these have been not only very expensive but also very difficult to use.

During an evaluation of DSP system-level design tools, Ken Chapman, a Xilinx Applications Specialist focusing on DSP, identified SystemView by Elanix as a good system-level design solution for FPGA implementations. Chapman has found that digital designers tasked with doing DSP designs often lack a formal background in DSP theory. When combined with the lack of tools, it was clear that many engineers had little chance of producing a truly optimal design. "As a digital design engineer who lacked a strong mathematical background, I found SystemView enabled me to grasp key DSP concepts in minutes," stated Chapman, "In addition, SystemView was the only tool that combined ease-of-use with the sophistication needed for today's designs."

SystemView is integrated with the Xilinx DSP tools, providing a smooth flow from system-level design to silicon implementation.



Integration with the Xilinx DSP design flow

To further enhance SystemView for FPGA users, Xilinx and Elanix have integrated SystemView with the Xilinx DSP LogiCORE functions and CORE Generator.

Each Xilinx DSP LogiCORE, such as a FIR filter or multiplier, has a corresponding token in SystemView's DSP library. SystemView includes Xilinx-specific parameter checking to verify that the token parameter values are supported by the

> *"You can now use a visual design environment to directly generate high performance silicon implementations for DSP applications."*

actual LogiCORE core. Once the design is completed in SystemView, you can automatically invoke and pass core parameters to the Xilinx CORE Generator. The CORE Generator then produces netlists of the fully parameterized cores, HDL simulation models, schematic symbols, and HDL instantiation code.

In future releases, SystemView by Elanix will produce structural VHDL code, detailing the interconnectivity between the cores required to implement the whole DSP subsystem. This structural VHDL includes the instantiation code produced by the CORE Generator for each core. SystemView will automatically invoke the Xilinx Foundation VHDL editor so you can integrate the DSP subsystem with the remainder of the design.

About SystemView

SystemView consists of a core tool that combines design entry, simulation, analysis, and filter design. In addition, a range of optional token libraries are available. Users of Xilinx DSP solutions must obtain the core SystemView tool and the DSP library, along with a Xilinx FPGA option.

Some of the more important benefits of SystemView for designers of DSP and communication applications are briefly summarized below:

- **Faster design iteration:** You can build models using high-level functional blocks (tokens), without needing to worry about low-level details such as clocking. Compared to traditional block-diagram tools, SystemView's parameter inheritance capabilities greatly reduce the number of parameters that must be entered for each. Since simulation occurs at the algorithmic level, simulation speed is orders of magnitude greater than HDL simulators. As a result of these attributes, you can build, evaluate, and change models very quickly and rapidly explore different design options.
- **Bit-true DSP token library:** When using an FPGA, you can reduce the final gate count by using only the minimal number of bits required to maintain signal integrity. The SystemView DSP library provides bit-true DSP function simulation, including quantization to the exact arithmetic mode. This enables quick determination of the appropriate bit-widths without the need to develop bit-true C models or hardware prototypes.
- **Token libraries:** In addition to a bit-true DSP library, SystemView provides CDMA/IS-95, communications, RF/Analog, and TTL logic libraries. If desired, you can build end-to-end communication systems. In addition, mixed-mode behavior, such as an FPGA-based DSP function interfacing to an A/D, can be modeled up-front, reducing the likelihood of encountering unexpected problems later in the design cycle.
- **Analysis tools:** Tools such as HDL simulators do not provide adequate signal analysis. As a result, you may have to write custom programs to plot numbers in a meaningful way. SystemView provides a range of analysis tools specifically designed for signal analysis. These tools enable you to quickly produce meaningful data plots ranging from power spectrums to QAM constellations or phase locked loop phase planes.

Conclusion

This integration of SystemView by Elanix and the Xilinx CORE Generator provides a faster and simpler method for designing high-performance DSP applications. You can now use a visual design environment to directly generate high-performance silicon implementations for DSP applications.

For further information on Elanix and its products, visit the Elanix website at: www.elanix.com or call 818-597-1414. ◆

ELANIX INCORPORATED

**The term 'SystemView' is used as a shorthand form for SystemView by Elanix*

Synplify Extends Timing Constraint

by Jim Tatsukawa, Partner Programs Manager, Synplicity Inc., jimt@synplicity.com

Synplicity has expanded its Synthesis Constraint Optimization Environment (SCOPE) to allow you to characterize the timing of macrofunctions not synthesized in Synplify. These new constraints integrate SCOPE more tightly for mixed mode design entry than any other FPGA constraint solution. SCOPE provides total control of your synthesis results by using an innovative multi-level constraint approach. The addition of the timing constraints for mixed mode entry complements the existing timing constraint.

New Synplify 3.0C Timing Constraints:

- Black Box Propagation Delays
- Black Box Setup Delays
- Black Box Clock to Output Delays

Existing Timing Constraint Features:

- Clock Frequency
- Input Delays
- Output Delays
- Delays To Registers
- Delays From Registers
- Multicycle Paths
- Improve Timing Constraint
- Route Timing Constraint

> *"Synplify allows you to define timing constraints that automatically control synthesis to meet the system requirements."*

When synthesizing a design, logic can often be made faster at the expense of more logic. The most direct method of evaluating these "area vs. performance" trade-offs is by analyzing the performance of the synthesized logic. Synplify allows you to define timing constraints that automatically control synthesis to meet the system requirements.

Consider the Xilinx cores and macrofunctions that have been carefully structured to fit into your design. Most synthesis tools do not allow you to specify the timing characteristics of these functions when incorporated into your design. Therefore, the synthesis tool does not understand whether inputs and outputs are registered or combinatorial. Additionally, the delays inside these block are unknown, leading to paths that become over- or under-constrained.

If a design is under-constrained, the logic will not be synthesized to map to the optimal amount of logic and will perform slower than required. If a design is over-constrained, synthesis compromises on design performance and area to achieve the goals. The over-constrained design may either be larger than required, or may be slower for the overall design.

Defining Hierarchy with the Black_Box Attribute

Synplify supports mixed mode design entry by instantiating components and attaching the "black_box" timing attribute. The black_box attribute allows the integration of schematics, LogiBlox, COREgen, Xilinx Core Solutions, as well as any other design that is not to be synthesized in VHDL or Verilog.

To use the black_box attribute create a stub for the macrofunction (logic content will be ignored). The stub must declare the ports and the port directions. By placing the "black_box" synthesis directive just before the semicolon in the module declaration, Synplify will ignore the internal logic. The body of the code then instantiates the component. The netlist that Synplify generates will be combined with the other design files when compiled in the Alliance Series tools.

Control For Mixed Mode Entry

Timing Constraints For Black_Box Modules

After the black_box module has been specified, Synplify allows the full timing characterization of the black_box module through the use of three types of timing attributes that are attached to the black_box definition. The following summaries describe how to effectively use these timing constraints to fully characterize your design.

- **Combinatorial delays** through the black_box module are defined by the syn_tpd attribute. It specifies the delays from the inputs of the black_box module to the outputs. Delays are given in nanoseconds.

```
Syntax: syn_tpd1="{input or input
 bus}->{output or output
 bus}={propagation delay in ns}"
```

- **Registered inputs** for the black_box module use the syn_tsu attribute to specify the setup time required for the inputs relative to the clock. Synplify will then recognize the black_box module as the destination of register to register propagation delays. Many designs use both rising edge and falling edge clocks. Synplify allows you to specify the rising edge and falling edge clocks. The syn_tsu specifies falling edge clocks by the additional use of the "!" character.

```
Syntax: syn_tsu[0-9]="{input or
 input bus}->{clock input}=
 {propagation delay in ns}"

Syntax: syn_tsu[0-9]="{input or
 input bus}->!{clock input}=
 {propagation delay in ns}"
```

> *"Synplify supports mixed mode design entry by instantiating components and attaching the "black_box" timing attribute."*

- **Registered outputs** for the black_box module use the syn_tco attribute to specify the clock to output delays within the black_box module. The syn_tco additionally specifies the black_box module as the source of register to register propagation delays. The syn_tco attribute supports the rising or falling edge clock specification as in the syn_tsu.

```
Syntax: syn_tco[0-9]="{clock
 input}->{output or output
 bus}={propagation delay in ns}"

Syntax: syn_tco[0-9]="!{clock
 input}->{output or output
 bus}={propagation delay in ns}"
```

◆

Example Design

```
module ram32x4(z, d, addr, we, clk);
/* synthesis black_box
     syn_tpd1="addr[3:0]->z[3:0]=8.0"
     syn_tsu1="addr[3:0]->clk=2.0"
     syn_tsu2="we->clk=3.0"
 */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;

endmodule

/*
 * Build a bigger ram
 */
```

```
module ram64x4(z, d, addr, we, clk);
output [3:0] z;
input [3:0] d;
input [4:0] addr;
input we /* xsynthesis
 syn_input_delay=10.0 */;
input clk;

wire [3:0] za, zb;

wire wea = we & ~addr[4];
wire web = we & addr[4];
ram32x4 r1 (za, d, addr[3:0], wea, clk);
ram32x4 r2 (zb, d, addr[3:0], web, clk);
assign z = addr[4] ? zb : za;

endmodule ◆
```

by Stefano Lorenzini,
CAE-VLSI Department,
Marconi S.p.A.,
Genova (Italy)

FPGA Design Cycle Time *Reduction and Optimization*

With the availability of high-density FPGAs (XC40250XV, 500K system gates) design implementation and verification are performed in parallel to meet time-to-market requirements. You are often forced to make functional changes to your design during the synthesis and implementation phase. These changes commonly termed as "Engineering Change Orders" (ECO) need to be implemented with minimal impact to the netlist, to preserve the original layout. For minor RTL changes, the traditional top-down iterative design methodology requires time consuming re-synthesis and re-layout for the complete design.

At Marconi S.p.A., the designers have developed a top-down/bottom-up synthesis methodology to accommodate ECOs, eliminating the need for complete design re-synthesis and re-layout. This methodology preserves netlist names of all current parts not affected by RTL functional changes enabling the design to use guided place and route. This methodology works with Synopsys FPGA and Design Compilers and requires in-depth knowledge of dc_shell, the Synopsys scripting language.

TRADITIONAL TOP-DOWN DESIGN METHODOLOGY

The traditional FPGA design cycle represented in **Figure 1** requires you to complete the RTL before entering the synthesis and layout cycle.

Using this approach, every small change in RTL necessitates repeating the synthesis and place and route stages of the complete design because synthesis does not preserve the netlist names. As a result, design iterations are very time-consuming and synthesis and layout can become the significant part of the design cycle. This design strategy is quite simple but has its challenges:

- Start of the synthesis phase constrained to the end of the RTL description
- Large amount of memory required for the synthesis process for large designs (above 50K gates)
- CPU time increase for synthesis and layout iterations
- RTL changes imply new synthesis and layout iterations
- No layout reuse capability

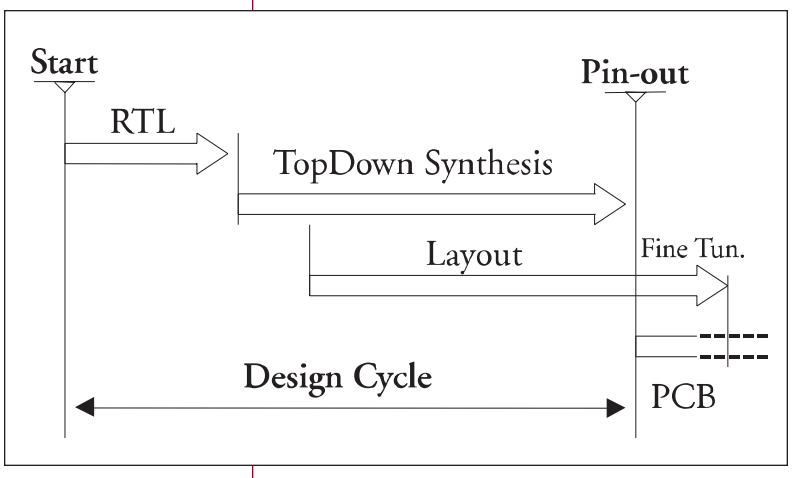


Figure 1

> *"...the designers have developed a top-down/bottom-up synthesis methodology to accommodate ECOs, eliminating the need for complete design re-synthesis and re-layout."*

MARCONI FPGA DESIGN METHODOLOGY

Due to the limitations using the traditional top-down design methodology and intense time-to-market requirements, the engineers at Marconi have developed a methodology that reduces the design cycle (as shown in **Figure 2**) using Top-Down/Bottom-Up synthesis, and post-layout synthesis capability (layout reuse).

Top-Down/Bottom-Up synthesis capability

This synthesis strategy is commonly used in an ASIC design flow. The hierarchical levels in the design are synthesized individually, then merged at the top-level. The complete design is then placed and routed using the Xilinx Alliance Series implementation tools. It leads to a tradeoff between process memory allocation and design constraints management.

The procedure is simply managed by setting, in a dedicated file, some mandatory variables such as the synthesis session name, the RTL file names, and the top-level module name. Once the design has been completely synthesized, the first layout iteration can take place.

Post-layout synthesis capability

After completion of the first design iteration, a post-layout synthesis algorithm (dc_shell script) is used to preserve netlist names during re-synthesis in all circuit parts not affected by the RTL functional change. This post layout synthesis capability is also called ECO capability. Since this methodology is very efficient for small design changes, highly partitioned designs and good partitioning RTL rules are essential for efficiency.

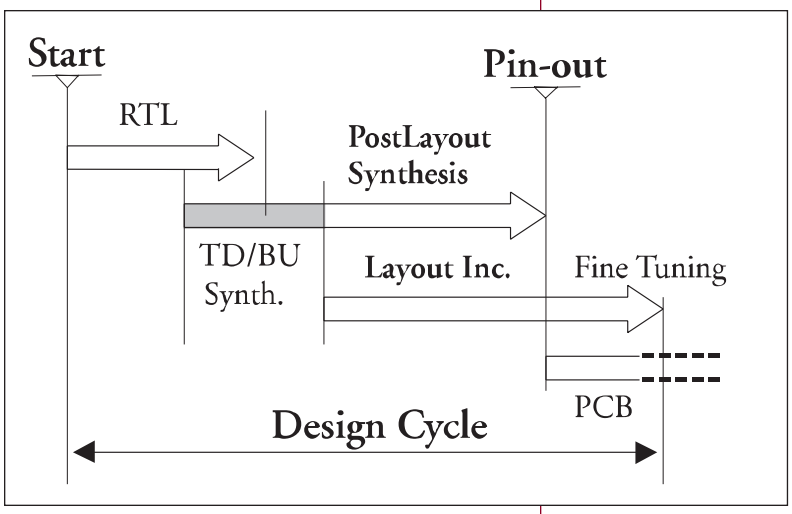


Figure 2

This ECO synthesis methodology, working in conjunction with a guide file, may reduce the design runtimes up to 10x.

Conclusion

Because FPGA densities are increasing rapidly, ASIC design methodologies are now suitable. The ASIC flow proposed here can reduce your FPGA design cycle time by means of advanced synthesis and layout techniques allowing you to control the design development at every phase, avoiding the limitations imposed by the standard synthesis techniques. In the future, with the availability of one million gate FPGAs (the Virtex family of devices) formal verification algorithms will be essential to further reduce the design cycle time for ECOs.

The dc_shells used for the Marconi design methodology may be obtained by contacting your local Xilinx FAE. ◆

Device Programmer Support

You can find all the latest information about third party programmer support for Serial Configuration PROMs and CPLD devices by visiting WebLINX, our website, at: http://www.xilinx.com/support/programr/dev_sup.htm.

Here's what you'll see:

- Guide to Device Support Tables
- XC1700 PROMs
- XC7200 CPLDs
- XC7300 CPLDs
- XC9500 ISP CPLDs

The XC95144 and the XC17Sxx Spartan Serial PROM family are the newest additions to the list. ◆

Verification *for Higher Productivity*

We take you to the leaders.

by Hitesh Patel and Carol Fields, Xilinx Alliance Marketing, hiteshp@xilinx.com, carol@xilinx.com

To maintain competitiveness, many designers have found that high-level Hardware Description Languages (HDLs) are essential for representing and verifying their designs. ASIC designers adopted the HDL design and verification methodology several years ago, because it works at a higher level of abstraction, allowing them to produce more gates per day.

With the introduction of the Xilinx Spartan family in 1997, along with the Xilinx mask programmed HardWire capability, FPGAs are rapidly becoming Gate Array replacements, and FPGA designers are now looking for a verification strategy to help improve productivity.

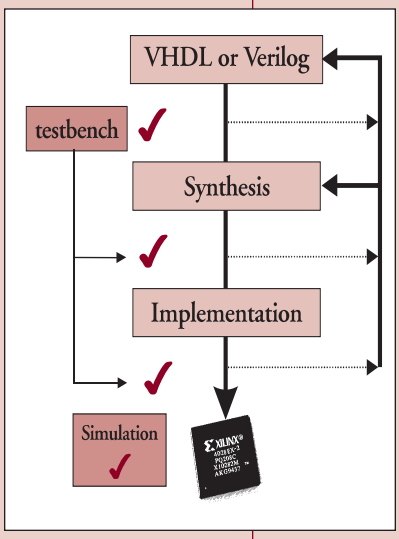


Figure 1: HDL Verification Design Flow

Verification Methodology

As design density and complexity increases, the cost of correcting errors increases exponentially with time. An error, if detected early in the design cycle (during RTL simulation), is fairly inexpensive to fix. That same error, if caught late in the design cycle, may necessitate a redesign and re-verification. Furthermore, debugging a finished device is very time consuming and often impossible because the debugging tools may not allow you to observe all internal nodes.

Perhaps the most publicized example of a costly bug was the one found in Intel's Pentium processor, which lead to Intel announcing a $475 million loss against fourth quarter earnings in 1994. To avoid such costly mistakes, designers are simulating their designs using a testbench methodology that allows them to observe all internal nodes and isolate errors early in the design process.

The design and verification methodology, shown in **Figure 1**, consists of four major steps: design entry, synthesis, implementation, and verification. You will often need to move between these four steps to either correct or change the circuit, and that requires you to verify the design at the following stages:

- RTL functional simulation
- Post-synthesis simulation
- Post-layout static timing analysis
- Post-layout simulation

Using a Testbench Methodology

A testbench is a separate set of VHDL or Verilog code that you use to specify circuit input stimuli and output responses, then you test to that specification at various points in your design cycle. This allows you to readily identify and resolve problems early in the design process, thereby saving significant time and costs. Steve Winkelman of DisplayTech states "I reduced my design cycle by 25% by adopting an HDL simulation methodology" *(see **Figure 2**)*.

A testbench also allows you to perform hardware regression, software debug, and system debug, in parallel. The same testbench (and simulator) are used before implementation to catch timing errors, and after synthesis to catch critical path problems (with estimated logic block and net delays).

Many FPGA designers are constantly pushing to higher density FPGAs such as the XC40250XV (500k system gates). The number of vectors required to verify these designs has risen many times faster than the size of the devices themselves, as shown in **Figure 3**. As a result, the amount of time spent in simulation increases dramatically. To help verify more in less time, other verification tools and strategies similar to ASIC strategies are being adopted. These tools include static timing, formal verification, cycle

XILINX®

base simulation, and emulation. They significantly speed-up the verification cycle times as opposed to simulation, which can take a long time and consume large amounts of memory.

The Changing Role of Verification

A design for verification strategy focuses simulation at the RTL level where it runs the fastest. The strategy replaces gate-level simulation with equivalence checking, which does a complete job of verifying that the low-level logic matches the RTL specification in a fraction of the time needed to simulate a meaningful vector set. With this strategy, only netlists that have already been proven equivalent to the "golden" RTL code are checked for timing requirements. As a result, timing issues are separated from functional issues and the timing simulation can focus solely on timing issues.

Emulation, used in conjunction with equivalence checking, allows you to run real applications with the certainty that these diagnostics also check the corresponding RTL code. Today, system designers and their verification counterparts can emulate large sections or even whole systems using the high capacity, high performance Xilinx FPGAs currently available.

Formal equivalence checking replaces regression simulation of gate-level implementations. Once an RTL specification is signed off, it serves as the reference against which implementations are compared. The fully verified implementation serves as the reference for later revisions. These formal comparisons provide complete coverage without vectors.

Why Should You Choose Xilinx?

Xilinx , from the very beginning, has used the industry standards such as VITAL, VHDL, Verilog, and SDF. These standards were developed for ASICs and have been applied to FPGAs, because as FPGAs replace ASICs, these standards are becoming critical for seamlessly integrating FPGAs and CPLDs into your existing flows.

The Xilinx AllianceEDA program and the Alliance Series software team provides the highest quality support for the industry-leading HDL simulators and emerging verification tools. By closely monitoring industry trends, we are prepared to meet your changing requirements. In the Alliance Series 1.5 release you will achieve the highest quality results. You will also find minimum delay characterization for hold time and race condition checking, Global Set-Reset (GSR) simulation models that depict the true behavior of the silicon, and a robust set of documentation supporting the industry's leading HDL simulators.

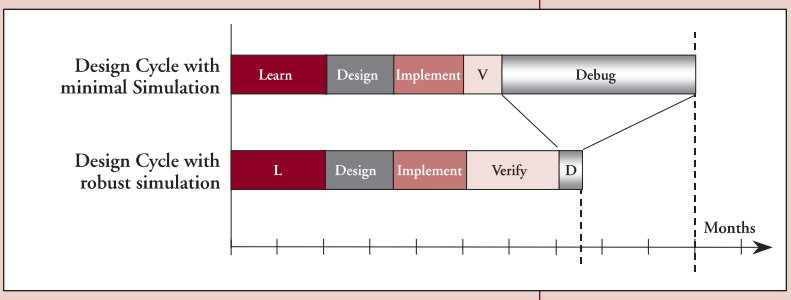


Figure 2: Productivity increases when using HDL simulation

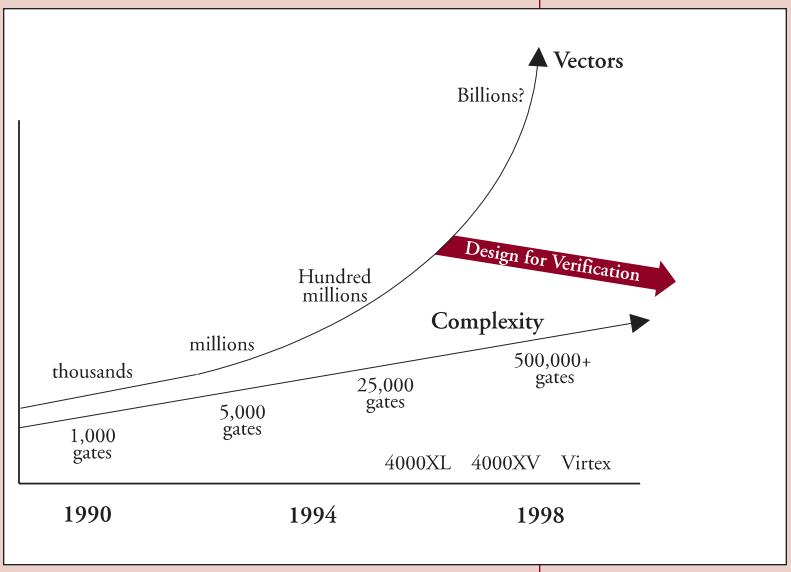


Figure 3: Vector requirements as gate densities increase.

Summary

Design verification is arguably the most critical task in successfully creating complex designs and decreasing time to market. Without verification, error isolation is a very tedious and time intensive effort.

In practice, engineers have found that a verification strategy improves their productivity and keeps design projects on schedule. As devices surpass one million gates, a design for verification strategy provides you with the confidence that you are releasing functionally correct devices.

Xilinx will continue to lead the way in developing advanced methodologies with our premier EDA partners. ◆

We take you to the leaders.

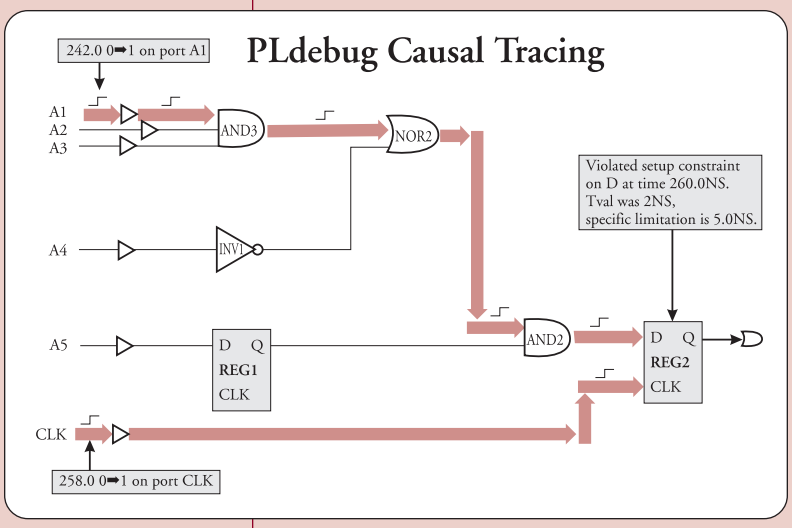
HDL VERIFICATION SPECIAL SECTION

by Mick Posner, Technical Marketing Manager, Synopsys, Logic Modeling

Using Synopsys SmartModel FPGA

FPGAs are often the most critical part of a system and must be extensively tested. However, today's designs are typically too large and too complex to rely on manual debugging methods; strong verification and debugging tools are required.

SmartModel FPGA models, known as SmartCircuit, provide you with the advanced verification and debugging features that are required to successfully verify your design in the shortest timeframe. The SmartCircuit models are basically templates of unconfigured devices. The models are programmed by a design netlist in the same format produced by the Xilinx place and route tools (standard EDIF). The SmartCircuit FPGA increases productivity by allowing you to focus on the design and system verification tasks rather than the simulation details.



PLdebug - Advanced Debugging and Event Tracing

One key to successfully verifying a design is the ability to quickly debug functional and timing errors. The SmartCircuit PLdebug feature allows you to do just that, by causally tracing back to the root cause of any logic event or timing error. Without PLdebug you would be forced to manually analyze hundreds of possible paths to identify a logic or timing error.

If you encounter functional or timing errors during a simulation run, it is imperative to trace them back to the parent event that is the root cause. In a large design this is usually a very complex task. The PLdebug feature uses an automated history mechanism to deliver this capability while making a minimal performance effect on the simulation. The causal tracing features work from user-specified trigger points to perform the following tasks:

- Trace back to locate the root cause of any logic event error.
- Trace forward to find the effects of any specific logic event.
- Identify the root cause of any timing constraint violation.

The PLdebug reports quickly identify the root of the logic or timing error by generating a list of events internal to the FPGA that are causally related to the problem event.

FIGURE 1 - CAUSAL TRACE

```
SmartModel TRACE:

Instance /TESTBENCH/DUT/
  SMART(XC4005E_84),at time 586.3 NS.

Beginning cause report from "DBUS<6>":

586.3 ns Z->X on model port DBUS<6>

586.3 ns Z->X on cell port /XSYM4/O,
  net DBUS<6>

586.3 ns 1->0 on cell port /XSYM4/T,
  net DBUS_ENABLE<0>

569.4 ns 1->0 on cell port /XSYM40/
  O,net DBUS_ENABLE<0>

564.9 ns 0->1 on cell port /XSYM44/O,
  net ENABLEBUS_SIG

562.6 ns 1->0 on cell port /XSYM43/O,
  net U2;N735

560.6 ns 0->1 on cell port /XSYM42/O,
  net YSIG2

558.1 ns 0->1 on cell port /U2;MODE<1>/
  Q,net U2;MODE<1>

555.3 ns 0->1 on cell port /U2;MODE<1>/
  C,net CLOCK1

553.9 ns 0->1 on cell port /BUFGS_TL/O,
  net CLOCK1

550.0 ns 0->1 on model port CLOCK

Report completed.
```

Note: The report is triggered on a user-specified event and then traces that event back through time to the parent event. In this case the parent event is the CLOCK port.

Adapted from a Synopsys Application Note.

Models to Verify Xilinx FPGA Designs

FIGURE 2 - EFFECT TRACE

```
SmartModel TRACE:

Instance /TESTBENCH/DUT/
  SMART(XC4005E_84),at time 1087.1 NS.

Triggering effect report from “DBUS<6>”
  at 1087.1 ns:

1087.1 ns Effect 0->X on cell port /
  XSYM3/O,net U3;N163

1090.9 ns Effect 0->X on cell port /
  XSYM50/O, net YSIG6

1090.9 ns Effect 0->X on cell port /
  U3;I<6>/D,net YSIG6

Report completed.
```

By tracing the effect of an 0 to X transition on the DBUS<6> port *(see **Figure 2**)*, you can see that the X propagates through the design to the U3/I<6> instance net. You can control the scope of the report, and target multiple events and simulation times.

FIGURE 3 - CAUSAL TRACE TRIGGERED BY TIMING CONSTRAINT VIOLATION

```
SmartModel ERROR:

Violated pulsewidth constraint PW_CLR+
  on CLR for cell U2;MODE<1> at time
  12.1 ns.


Actual pulsewidth time 3.0ns, specified
  minimum is 4.0 ns.

Instance /TESTBENCH/DUT/
  SMART(XC4005E_84),at time 12.1 NS.


SmartModel TRACE:

Constraint causal report for event on
  “CLR” at 12.1 ns:

12.1 ns 1->0 on cell port /XSYM72/O,
  net YSIG27

10.5 ns 1->0 on cell port /XSYM37/O,
  net U2;N658

5.5 ns 0->1 on cell port /XSYM33/O, net
  N4

3.0 ns 0->1 on model port RESET

Report completed.
```

Using Pldebug you can quickly identify the source of a functional error and the source of a timing constraint violation. In **Figure 3**, you can see that the source of the timing violation was a short pulse on the RESET port.

Advanced Debugging - Windows and Monitors

Visibility into the FPGA design during simulation is another critical success factor. The ability to trace the contents of an internal net or register will aid in the debugging of the overall design. Another feature of the SmartCircuit FPGA models is the ability to look inside the FPGA design using the Windows feature — no longer is the FPGA a black box within the simulation. The Windows feature allows you to trace, in the simulation waveform window, any nets, ports, or states. This gives you full visibility into the design at a level that is easily understood. Having this visibility substantially eases the FPGA verification process and the subsequent debugging. You can trace the designated nets, ports, or states and force values on them, allowing you to recreate corner cases and evaluate a design's functionality in those cases. The monitor feature enables you to create a text print-out of the values on the nets, ports, or states in the selected portions of the design within the simulator's transcript window.

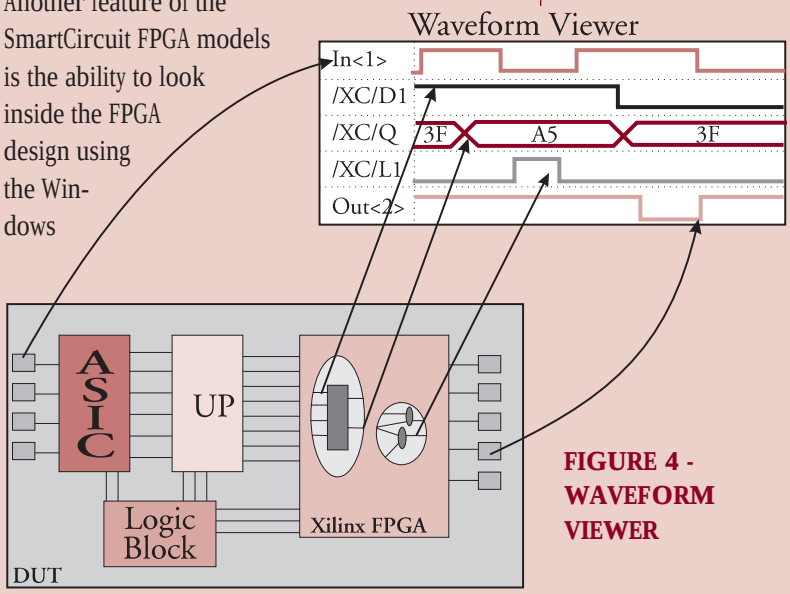


FIGURE 4 - WAVEFORM VIEWER

Advanced Debugging - Visual SmartBrowser

Visual SmartBrowser (VSB) allows you to visually display the FPGA netlist using an on-demand viewing technique. With very large and complex FPGA designs, you are typically only interested in a small section of the netlist. VSB allows you to

Continued on the following page

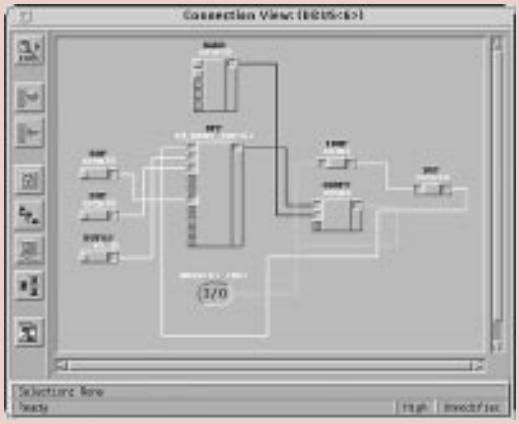


FIGURE 5 - VISUAL SMARTBROWSER

concentrate on only the section that interests you.

VSB also incorporates a set of tools that allow you to identify sections of the design you want to display. VSB then generates the model command file (mcf) required to make these sections available in the simulation. The only file that you have to create, to simulate using a SmartCircuit model, can be automatically generated using VSB.

Advanced Debugging - VSB helps identify where a timing constraint fix should be implemented

VSB is run on the SmartCircuit netlist that contains all of the design's specific delay and timing information, extracted from the original vendor netlist. This delay and timing information is put at your fingertips using the VSB examine cell view**.**

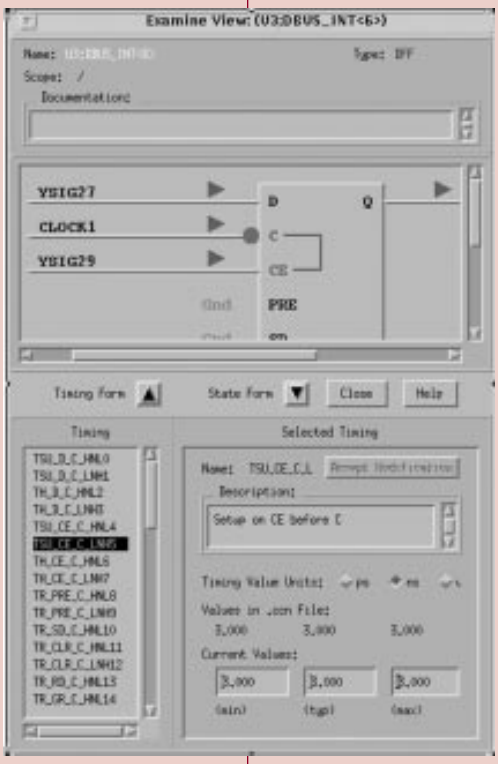


FIGURE 6 – EXAMINE CELL VIEW

After using PLdebug to identify the root cause of a timing violation you can view the actual delays being used by each cell in that specific path. You can then use VSB to change the timing of any parameter on the cell and thus experiment with "what if" scenarios. The incremental changes do not affect the source netlist, but allow you to exactly identify where the problem lies. With that information you can return to the original source of the design and fix the problem. Doing very quick incremental changes on the SmartCircuit netlist, to evaluate if a change does fix the problem, speeds up the overall design cycle.

SmartModel FPGA models fit straight into your existing design flow.

Below is a simplified view of the design flow using SmartModels in system verification.

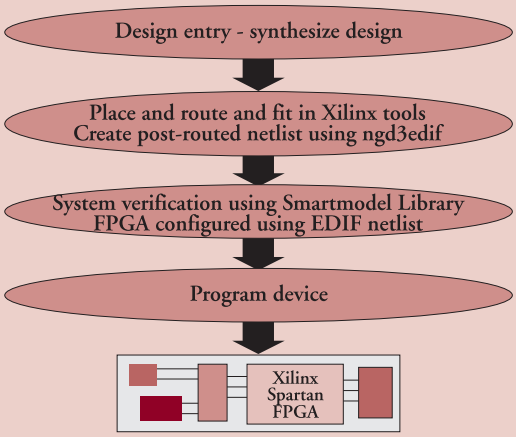


FIGURE 7 – SMARTMODEL DESIGN FLOW

Using SmartModels with Xilinx Alliance Tools

You don't need to do anything special in the Xilinx Alliance Series tools to target a SmartCircuit FPGA model. The following options must be selected in the Design Manager tool:

- From the design pull down select **implement and options**. Select **edit implementation template**. In the **Interface** section make sure that the simulation data options are set to **Generic EDIF**. (These are the Design Manager Tools default settings.)
- Under **implement options** make sure **Produce Timing Simulation Data** is selected under the **Optional Targets** section. (Configuration Data is checked by default.)
- Run the tools as normal and they will produce a time_sim.edn file. This is your post-routed netlist file that will be used to configure the SmartCircuit models.

Conclusion

Using SmartModel FPGA models from Synopsys can save valuable time and minimize the difficulty of verifying and debugging complex FPGA designs

For full SmartModel documentation see the Synopsys home page at, http://www.synopsys.com/products/lm/docs/swift_r41/intro.html. ◆

Mixed Schematic and HDL Design Entry

We take you to the leaders.
HDL VERIFICATION SPECIAL SECTION

by Peter Lin, Technical Marketing Engineer, Alliance Series, peterl@xilinx.com

Xilinx allows complete design flexibility. Even though a significant amount of designs are done totally in schematics or HDL, there is also an ability to mix these two popular entry methods. With a top-level HDL or schematic, the submodules can be either HDL or schematics. The advantages to mixed-mode design entry are:

- Flexible design environment for a design team with mixed skill members. (Some are skilled in schematics while others are skilled in HDL.)
- You can use existing schematic designs.
- Allows a visual representation of large logic blocks and interconnect via a top-level schematic.
- HDL descriptions of large blocks are easier to understand, and simulate quickly and easily.
- You have total control of logic mapping and constraints in a schematic.
- You can automatically optimize and map an HDL design using synthesis tools.

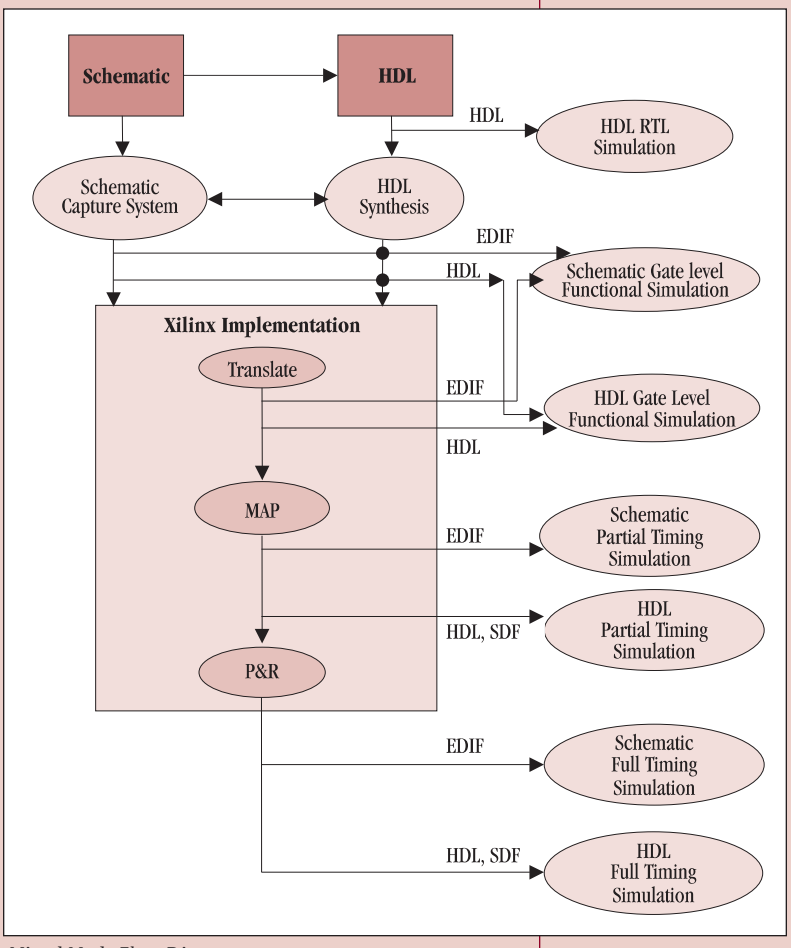
A mixed mode design can be simulated, with a top-level HDL or schematic, at the following five stages of your design cycle:

1. Before synthesis.
2. After synthesis.
3. After the Translate step in the Flow Engine.
4. After the Map step in the Flow Engine.
5. After the Place & Route step in the Flow Engine.

Simulation with a Schematic at the Top Level

1. Before synthesis, instantiate HDL modules into the schematic and simulate the schematic using test vectors. The HDL modules are simulated for functionality before synthesis using an HDL testbench. It is also possible to simulate the complete design using appropriate simulation tools. In the Mentor design environment, you can use QuickHDL Pro to co-simulate schematics and HDL. In the Cadence design environment, you can use Verilog XL to simulate the complete design by merging the pre-synthesis HDL modules with the gate level HDL netlist written by Concept.
2. After synthesis, generate an EDIF netlist for the HDL modules. In the Mentor design environment, you can merge schematic EDIF netlists and HDL EDIF netlists, and simulate the complete design using Quicksim, which is a schematic gate-level simulator. In the Cadence design environment, you can re-simulate the design using Verilog XL after synthesis by writing a post-synthesis HDL netlist merged with the gate level HDL netlist written by Concept.
3. After translate, create an EDIF file by running NGD2EDIF from the command line to re-simulate the gate level netlist after translation to Xilinx primitives.
4. After map and before place and route, simulate your design with block delays but not routing delays. Use NGDANNO from the command line to create a back annotated simulation file and use NGD2EDIF to create an EDIF file. You can

Continued on the following page



Mixed Mode Flow Diagram

We take you to the leaders.

HDL VERIFICATION SPECIAL SECTION

by Nupur Shah,
Design Engineer,
nupur@xilinx.com

Verifying PCI Designs

Your PCI design must be PCI compliant to work along with other vendors' PCs and PCI add-in boards. Therefore the final stage in the design flow of a PCI design is design verification, which consists of two steps: **functional verification** and **timing verification.**

By using a pre-designed PCI interface core, with predictable timing (such as Xilinx LogiCORE PCI products) the PCI protocol and timing is already verified. Therefore, you can focus on your unique back-end design and how it interfaces to the PCI core. However, even though the direct interface to the PCI bus is pre-verified, it is still your responsibility to verify the compliance of your final design.

The PCI protocol is very complex, and as a result, it is difficult to test every possible combination of transactions. Although simulation covers most of the functional test, the only way to verify full PCI compliance is to test the actual hardware. This can be accomplished at the quarterly PCI-SIG "plug-fests." At these events, PC manufacturers and PCI board manufacturers gather to test their

Mixed Schematic

Continued from the previous page

then perform timing simulation with block delays. This is generally a good way to test whether the design is meeting your timing requirements before spending the time to do a full place and route.

5. After place and route (if you have previously selected EDIF as the simulation data format), simulate your complete design with both block and routing delays.

Simulation with an HDL at the Top Level

1. Before synthesis, simulate the HDL modules of your design using an HDL testbench. You can simulate schematic components using a gate level simulator. You can simulate the complete design as in the case of a top-level schematic.
2. After synthesis, generate an EDIF netlist for the HDL modules. You can simulate the gate level function of the complete design as in the case of a top-level schematic.
3. After translate, generate a structural HDL netlist using NGD2VER or NGD2VHDL for gate level simulation after the design is translated to Xilinx primitives.
4. After map, simulate the design with block delays similar to a top-level schematic except using the command line NGD2VHDL or NGD2VER to create a structural simulation netlist.
5. After place and route, (if you have previously selected VHDL or Verilog as the simulation data format), simulate the complete design with both block and routing delays with VHDL/Verilog and associated SDF (standard delay format) file.

Bus Notation in Schematic and in Synthesis

In a mixed mode design, the Bus Dimension Separator style in your schematic must match your synthesis bus style. If you use a synthesis option to generate one bus style and then use the EDIF from your schematic to generate a different bus style, the implementation tool will not merge the schematic module with the rest of the design, leaving it unexpanded.

Conclusion

Xilinx allows mixed mode design methodology with both HDL and schematic submodules. With a schematic at the top level, you can reuse test vectors throughout the design cycle; while with HDL at the top level, you can reuse your testbench throughout the design. ◆

products. By implementing your PCI design in a Xilinx FPGA, the hardware can easily be changed and fixed if a problem is found.

PCI Functional Verification

To test PCI compliance, you will need to implement the PCI Special Interest Group (SIG) Test Scenarios for Compliance Testing. These scenarios test the basic transactions between two agents on a PCI bus; one agent being the device under test (DUT) and other being the behavioral model of a PCI Initiator. These tests verify if the DUT is PCI complaint or not. You have the option to purchase a testbench from a third party vendor. However, learning these testbenches can prove to be just as difficult as developing one yourself, and there is no guarantee that they provide full fault coverage.

If you choose to develop a testbench, there are two types of testing that must be performed in order to verify that your design is functional: system-level testing and PCI bus protocol testing. System-level testing sets up modules in a system to transfer data back and forth. It checks to see if the data was actually sent and whether it arrived at the destination or not. It also checks for the validity of the data. However, the correct operating procedure is not checked in system-level testing.

PCI bus protocol testing is used to determine if the modules in a system operate within the rules of the protocol. Besides verifying the data, protocol testing verifies that the agents are PCI compliant. PCI protocol tests should include the basic functional testing that is outlined in the PCI test scenarios. However, these test scenarios do not test every bus situation. There are several other conditions that have high probability of occurring and are recommended for implementation.

Some of the recommended exercises that should be implemented are:

Target termination sequences: Sequences where the target issues a termination when the master inserts wait states before, during, and after the termination.

Parity checking: Sequences where the incorrect address and data parity are generated to check the ability of the DUT to report errors.

Protocol checker: Checks to see if all the operating rules are obeyed.

> *"The PCI protocol is very complex, and as a result, it is difficult to test every possible combination of transactions."*

Developing an extensive testbench will help you determine the correctness of the design before continuing on to the implementation stage of the design cycle. After implementation, the testbench can be used for backannotated timing simulation.

PCI Timing Verification

Timing verification occurs after the design has been implemented. There are two stages to timing verification: static timing analysis and back annotated timing simulation.

Static timing analysis is used to determine if the design meets all the PCI timing specifications such as setup/hold time and clock-to-out timing. Static timing analyzers are provided by FPGA vendors and will determine if 100% of the design met the timing specifications. You have the responsibility of specifying these timing parameters to the tool and determining what parts of the design should be attached to the each parameter. If the design does not meet timing, the static timing analyzer can be used to probe the design and determine where the design is failing. Based on this investigation, you have the option to re-implement the design using a tighter set of timing specifications or redesign parts of the design to contain less levels of logic.

After the design passes the static timing analysis, it is beneficial to run the physical netlist, with timing information, through the functional testbench. This will allow you to determine if, in fact, the design meets timing using actual physical delays (not simply the unit delays that are applied during functional simulation). Once you have verified the design, it is ready to be downloaded to a device and used in a physical system.

Conclusion

Using a pre-verified Xilinx PCI LogiCORE can save you a lot of time and effort. These cores have been proven in hundreds of designs. However, you must still test your completed design to guarantee full PCI compliance. ◆

We take you to the leaders.

HDL VERIFICATION SPECIAL SECTION

by Philip Lewer,
Product Marketing,
Viewlogic,
plewer@viewlogic.com

Viewlogic's Mixed-Design Verification Methodology

Viewlogic Systems offers a comprehensive and flexible environment for language-based design using VHDL or Verilog, or a combination of both. By supporting multiple design styles and by allowing styles to be mixed, Viewlogic has created a verification flow that is flexible enough to encompass a wide range of design methodologies. You have the ability to verify functionality prior to synthesis, after synthesis, and after place and route from the Fusion simulation environment. With tight integration to board-level design, verification can even be extended to systems with multiple FPGAs.

Design Entry Options

The benefits of using hardware description languages (HDLs), such as VHDL or Verilog, are numerous, especially when you consider the ever increasing device densities such as those available in the Xilinx Virtex family. Language-based design provides you with the ability to work at higher levels of abstraction, increasing productivity and reducing design costs.

A significant advantage of language is that it enables functional verification prior to committing a design to a particular technology. Once functional correctness has been verified, the design can be input to a language synthesis tool, such as FPGA Express, for optimization to the selected Xilinx device family.

At Viewlogic, we have found that language is not well suited to all types of design description. Some designers prefer using a block diagram for their top-level structural instantiation as opposed to writing structural VHDL or Verilog; some system houses require schematics at the top.

In other cases, designers may find it easier to use bubble diagram graphical editors for state machine entry. Being able to mix and match these alternative design styles and verify them in a single process is critical to design success. The use of IP cores may also require you to instantiate language modules into a block diagram, thereby requiring a verification environment that can handle a mix of design styles.

Co-Simulation with the Fusion Verification Environment

To handle this complete variety of mixed design styles, Viewlogic offers the Fusion simulation environment. Fusion is an environment that encompasses four simulators. There are three digital simulators and one analog simulator. The digital simulators are:

- Fusion/ViewSim for gate level simulation.
- Fusion/Speedwave for VHDL.
- Fusion/VCSi for Verilog simulation.

In the Fusion environment, you can use any one of the simulators, or a mix of all three. Each simulator operates on its piece of the design, fully communicating with the other simulators as the design dictates. You see the output presented as if the Fusion environment were a single simulator; there is one set of input stimuli and one set of output.

Functional Verification

Before committing your design to place and route or synthesis, you should verify that your design is functionally correct; finding bugs early in the design cycle makes them much easier to resolve. Viewlogic gives multiple options for functionally verifying your design.

For pure schematic designs, after schematic capture you perform gate-level simulation in the Viewlogic Fusion/ViewSim simulator. Xilinx provides a complete set of ViewSim models with the Alliance Series software that enables you to simulate these schematics. You can translate your schematics into a ViewSim netlist, load the design into ViewSim, stimulate your design, and verify the results. Stimulus is provided via the ViewSim command language. Usually, these commands are executed from a file, but they can also be run interactively. ViewSim has the ability to check outputs against a given set of inputs, which gives it

a regression capability, and ViewSim can also generate a waveform file, which can be inspected in the Vwaves waveform reader program.

VHDL designs can be verified with the Fusion/SpeedWave VHDL simulator, and you will usually write a testbench to stimulate and verify your designs.

If you are using Verilog, your design can be verified with the Fusion/VCSi Verilog simulator. Like SpeedWave and ViewSim, Fusion/VCSi is accessed from the Fusion simulation environment. Now you have the option of using a ViewSim command file or a Verilog test fixture. The test fixture and Verilog design files are loaded into Fusion/VCSi where they are compiled on the fly and automatically loaded into the Verilog simulator. Again, like SpeedWave and ViewSim you can generate waveform files for inspection in Vwaves.

An additional advantage of Verilog is the PLI programming language interface. The PLI defines how to interface user-defined C language functions with a Verilog design. This means that you can co-simulate your Verilog design with the C functions.

Viewlogic's Fusion simulation environment also supports mixed design styles such as a top-level schematic with underlying VHDL and/or Verilog blocks for representing state machines or synthesizable cores. Through Viewlogic's Fusion environment, you have the ability to functionally simulate schematics containing Xilinx primitives with VHDL blocks and Verilog blocks. The power of this environment is that you keep your full language debugging capability seamlessly flowing among gate-level, VHDL, and Verilog simulation. When you use this methodology, you will typically write your stimulus in ViewSim command format and not use HDL testbenches or test fixtures

Post-Synthesis Simulation

Post-synthesis simulation is used to verify that your VHDL or Verilog code was synthesized into the logic you expected. FPGA Express has the ability to output VHDL and Verilog netlists representing the synthesized design. It is at this point that your testbench or test fixture can be used again. This time the unit under test is not your original code but is the top-level entity/module of the netlist synthesized by FPGA Express. The synthesized netlist is written as structural VHDL or Verilog, instantiating primitives from the Xilinx UNISIM library with the primitive behaviors defined. By using this netlist in conjunction with the UNISIM library and your testbench/test fixture, your design can be verified. Note that FPGA Express can also output these netlists with the primitive behavior defined.

Post Place-and-Route Simulation

After place and route, the Xilinx Alliance Series tools can output a structural netlist in Viewlogic-compatible EDIF, VHDL, or Verilog. If you are creating schematic-based designs, you will most likely choose EDIF. This timing-embedded EDIF can be converted into a ViewSim netlist, which can be very useful if you are also using ViewSim for board level verification. If you choose this path, you can use your original simulation command file to verify timing through your design. If you are doing VHDL or Verilog designs, you may also choose EDIF or you can have the Alliance Series Core tools output structural VHDL or Verilog with a corresponding SDF timing file instead. Unlike the structural netlist generated by FPGA Express, the structural VHDL and Verilog netlists output by the Xilinx Alliance Series tools instantiate SIMPRIM primitives. Xilinx provides libraries for the SIMPRIM primitives for use with Fusion/ViewSim, Fusion/SpeedWave (VHDL/VITAL), and Fusion/VCSi (Verilog).

"As you move from schematics to language-based design, it is important to have an environment that allows multiple verification options."

Conclusion

As you move from schematics to language-based design, it is important to have an environment that allows multiple verification options. Viewlogic provides a set of tools for language verification, schematic verification, and mixed schematic/language-based verification. Xilinx, through its UNISIM and SIMPRIM libraries, provides the models that enable these flows. By covering these three types of design styles, you may pick a verification flow that best suits your design methodology.

For more information on the Viewlogic solution for FPGAs and other types of programmable devices, contact Viewlogic at 1-800-873-8439 or visit our website at www.viewlogic.com ◆

We take you to the leaders.

HDL VERIFICATION SPECIAL SECTION

by Mahadevan Ramasame, Technical Marketing Engineer, Alliance Series, mahadeva@xilinx.com

The Basic Elements of HDL Simulation

This article introduces the basic facts and terminology of HDL simulation for FPGAs and CPLDs, to help you simulate your design more efficiently.

There are three stages in the FPGA design process in which you conduct simulation:

- **Register Transfer Level** - To verify the syntax and functionality without the timing information. The majority of the design development is done through repetitive RTL simulation until you get the required functionality. Errors identified early in the design cycle are inexpensive to fix compared to functional errors identified during silicon debug.
- **Gate-level Functional Simulation** – After the RTL simulation is error free, the HDL design is synthesized to gates. The post-synthesized gate-level simulation is a functional simulation with unit delay timing. The simulation can be used to identify initialization issues and to analyze don't care conditions. "The don't care space of a design may be larger than the functional space," says Michael Bohm, VP and Chief Scientist at Exemplar Logic. The post synthesis simulation generally uses the same testbench as functional simulation.
- **Gate-level Timing Simulation** – Gate-level timing simulation is a back-annotated timing simulation. Timing simulation is important in verifying the operation of your circuit after the worst case place and route delays are calculated for your design. The back annotation process produces a netlist of library components annotated in an SDF file with the appropriate block and net delays from the place and route process. The simulation will identify any race conditions and setup-and-hold violations based on the operating conditions for the specified functionality.

> *"The top-down design method refers to applying a single optimization at the top level of your design; the bottom-up design method refers to performing individual optimizations on sub-blocks of your design."*

Design Techniques for Better Simulation Results

Design techniques are used in the process of applying optimizations to an FPGA design. The top-down design method refers to applying a single optimization at the top level of your design; the bottom-up design method refers to performing individual optimizations on sub-blocks of your design.

To improve the quality of the results of the simulation, use the following guidelines for design partitioning:

- Limit gate counts in sub-blocks; 10k to 50k gates.
- Limit clocks to one per block.
- Group similar logic together, such as state machines, data path logic, decoder logic, and ROMs.
- Partition state machines into separate blocks of hierarchy.
- Separate timing-critical blocks from non timing-critical blocks.

These features eliminate any ambiguity in your design by providing better quality simulation results. You can also add other features to the design such as breaking the asynchronous feedback loops, and design stitching to build the entire design after optimizations have been performed on individual subblocks. You can also unfold the netlist to perform two different optimizations (such as area or delay) on two different instances of common sub-blocks. These features can help improve the quality of simulation results.

Simulation Libraries

The following libraries are available for the Xilinx simulation flow:

- **UNISIM Library** - Used for functional simulation and contains default unit delays. This library includes all of the Xilinx Unified Library components that are inferred by most popular

synthesis tools. The UNISIM library also includes components that are commonly instantiated such as I/O's and memory cells. You can instantiate the UNISIM library components in your design (VHDL or Verilog) and simulate them during the RTL simulation. The HDL code must refer to the compiled UNISIM library. The HDL simulator must map the logical library to the physical location of the compiled library.

- **LogiBLOX and Coregen Library** - LogiBLOX is a module generator used for schematic-based design entry of modules such as adders, counters, and large memory blocks. LogiBLOX can be used in the HDL flow to generate large blocks of memory for instantiation. LogiBLOX components are simulated with behavioral code. They are not intended to be synthesized, but they can be simulated. Coregen library models are high level VHDL behavioral or RTL models that are mapped to SIMPRIM structural models in the back-annotated netlist. The behavioral model is used for any post-synthesis simulation because synthesis processes these modules as a black box.
- **SIMPRIM Library** - Used for simulations at the following steps in the design flow:
  - RTL simulations that include instantiated LogiBLOX modules.
  - Post-implementation simulations.
  - Timing simulation.

LIBRARY COMPILATION

The UNISIM libraries are used for RTL and post-synthesis simulations. Because industry standard simulators like ModelSim use pre-compiled libraries, Xilinx recommends compiling the UNISIM components that are instantiated in the current design. The UNISIM VHDL or Verilog Library can be compiled to any physical location. The order in which the VHDL source files for the UNISIM library must be compiled is listed in the Xilinx simulation design guide.

The LogiBLOX library is not a library of modules. It is a set of packages required by the LogiBLOX models that are created on-the-fly by the LogiBLOX tool. The source libraries for the LogiBLOX packages must be compiled into a library named LogiBLOX. These packages are available separately for VHDL and Verilog designs. The component model from the LogiBLOX GUI should be compiled into your working directory with your design.

The Simprim VHDL or Verilog Library can be compiled to any physical location and can be named Simprim.

VHDL Simulation

The Xilinx simulation flow supports the VHDL language standard IEEE-STD-1076-87 and the standard logic package IEEE-STD-1164-93. In VHDL designs, you must declare as ports any signals that are simulated or monitored from outside a module. Global GSR and GTS signals are used to initialize the simulation and require access ports if controlled from the testbench. The addition of these ports makes the pre and post implementations of your design different and your original testbench is no longer applicable to both versions of your design.

However, it's usually a good idea to get a pre-route VHDL description (used in gate-level functional simulation) that can be used for functional simulations with the GSR and GTS characteristics that match post-route results (gate-level timing simulation). This enables you to predict your design description accuracy at an earlier stage and reduces your design modification after place and route; therefore, this reduces your total design time. This is achieved by the addition of new library cells to simulate the GSR/GTS behavior.

Global Signal Methodology

To match the simulation behavior at all the three stages in the FPGA design, add a behavioral representation for GSR and Xilinx implementation directives. This directive is used to specify, to the place and route tools, the use of the special purpose GSR net that is pre-routed on the chip, and not to use the local asynchronous set/reset pins. Hence it utilizes the existing routing resources and significantly improves the performance. The

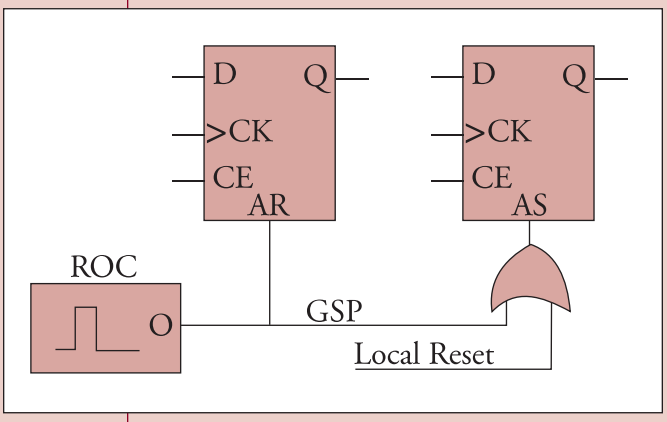
Continued on the following page

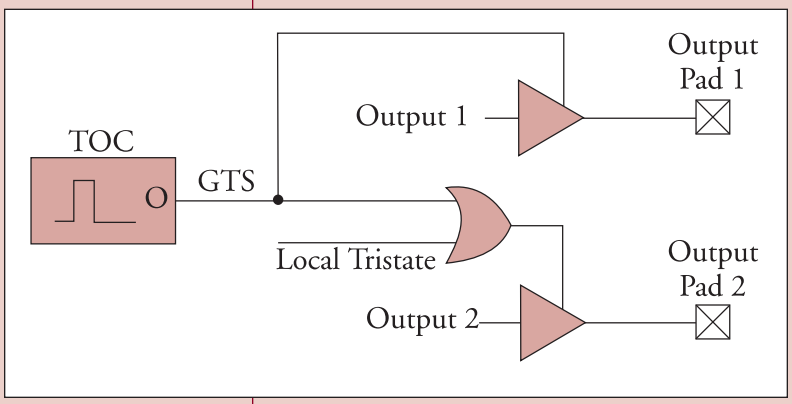
new library cells introduced have both the behavioral representation and the implementation directives. The new library cells are:

- ROC – Emulates the reset on configuration pulse.
- ROCBUF – Allows the test bench to drive the chip-generated reset on configuration without implementing an actual input pin on the chip.
- TOC – Emulates the chip-generated 3-state on configuration pulse.
- TOCBUF – Allows the test bench to drive the chip-generated 3-state on configuration without actually implementing the actual input on the chip.
- STARTBUF – A technology-independent version of the STARTUP block supported for simulation.

These five cells allow you to control the global reset and 3-state signal emulation, so you can get pre-route initialization simulations to match post-route simulations. The cells also drive implementation tools to add or delete pins and also help in the selection of nets for routing.

Model of ROC for Functional Simulation





Model of TOC for Functional Simulation

Models of the ROC and TOC cells, used for functional simulation, are given below.

Xilinx VHDL simulation supports the VITAL modeling standard IEEE-STD-1076.4 – 95 and Standard Delay Format version 2.1. This standard allows you to simulate your designs with any VITAL-compliant simulator and hence it accelerates your design compilation times, resulting in improved performance.

These features allow you to do high performance designs with shorter design cycles.

Verilog Simulation

The Xilinx simulation flow supports the Verilog language standard IEEE-STD-1364-95. The Verilog version of the UNISIM library may not need to be compiled, depending on the Verilog tool. Because there are a few cells with functional differences between Xilinx devices, a separate library is provided for each supported device. The libraries are in uppercase only and if needed, lower case libraries are provided in Xilinx/Cadence interface.

Unlike VHDL, Verilog can simulate internal signals and these signals are driven directly from the testbench without instantiating any specific component. The global set/reset net is present in your implemented design even if you do not instantiate the STARTUP block in your design. The function of STARTUP is to give you the option to control the global reset net from an external pin.

The general procedure for specifying the global set/reset during a post-synthesis Verilog UNISIM simulation involves defining the global reset signals with suitable Verilog macros. This is necessary because these global nets do not exist in the UNISIM libraries and as a result, the reset of UNISIM components is controlled by the detection of those macros. Also, the global set/reset signals need to be declared as either a **wire** or **reg** and the choice depends on whether the design contains a STARTUP component or not.

At the beginning of an FPGA design simulation, the global set/reset signal or the GR global reset signal must be toggled to emulate the power-on reset of the FPGA. This is to ensure that the flip-flops and latches in your simulation function correctly. The general procedure for specifying GTS is similar to that used for specifying the global set/reset signals, GSR and GR.

Testbenches

A testbench is a separate set of VHDL or Verilog code that connects to the inputs and outputs of a design. The testbench has two main purposes:

- It provides the stimulus and response information (clocks, set/reset, input data, and so on) that the design will encounter when it is implemented in an FPGA and installed into the final system.
- The testbench contains regression checking constructs, which allow design functionality to be tested throughout the FPGA HDL Simulation flow (RTL, Functional Gate, and Timing Gate).

A SAMPLE TESTBENCH MODEL:

```
library declarations ;
entity sample of testbench is
end ;
architecture test of testbench is
 instantiation of a component of the design ;
 signal declarations ;
begin
 port mapping of the component to the signals
 declared ;
process
     begin
                  clock period declaration ;
end process ;
process
     begin
                  Test vectors for design ;
end process ;
end test ;
configuration statement to configure
 architecture to the component
instantiation; (optional )
```


Use the steps for simulation in an industry standard simulator such as ModelSim:

- Create a working library.
- Compile the RTL/post-synthesis/place and route HDL design.
- Compile the testbench.
- Simulate the testbench and design. For place and route HDL design, simulate the testbench and design, with timing information.
- Run until the testbench stops.

You can create the testbench using a particular coding style for supplying the stimulus by referring to the synthesis or simulation vendors' documentation.

Conclusion

Xilinx offers a wide range of FPGA and CPLD solutions, including large density devices and low cost devices. You can successfully implement and correlate global initialization behavior of user-defined logic-, LogiBLOX-, and CORE Generator-based designs at all simulation phases, from RTL to back-annotated netlists. ◆

"Xilinx offers a wide range of FPGA and CPLD solutions, including large density devices and low cost devices. You can successfully implement and correlate global initialization behavior of user-defined logic-, LogiBLOX-, and CORE Generator-based designs at all simulation phases, from RTL to back-annotated netlists."

We take you to the leaders.

HDL VERIFICATION SPECIAL SECTION

by Troy Scott, Technical Product Manager, OrCAD Express

Board Design and Simulation Using OrCAD Express

OrCAD Express complements the Xilinx Alliance Series and Foundation Series software by providing robust system-level design capabilities. Because a majority of electronic design engineers who develop programmable logic are also responsible for the system-level documentation (such as schematics, bill of materials for production, and a PCB layout netlist), OrCAD set out to ensure that, with OrCAD Express, it would be easy to verify and integrate Xilinx CPLDs and FPGAs into the system. The typical workflow for one or more progammable devices adjacent to the system design workflow is illustrated in **Figure 1**.

Device Design and Simulation

OrCAD Express includes Xilinx Unified Libraries and a LogiBLOX interface for schematic design, VHDL models for functional and timing simulation, as well as RT-level synthesis which provides a solution for a majority of the programmable logic design flows.

All Xilinx CPLDs and FPGAs can be debugged and confirmed with the behavioral and gate-level simulation of OrCAD Express. Schematic debugging is eased with schematic cross-probing and signal-state annotations on the schematic, as well as the necessary VHDL debugging facilities to confirm that models operate correctly before synthesis.

Auto Symbol Generation

To help document large I/O count devices at the system level, many engineering groups create electrical symbols of the FPGA using a pin naming convention based on the names used in the design. **Figure 2**. illustrates a typical signal-to-package-pin name transition. **Figure 3** shows the electrical symbol created by the Generate Part tool of Express.

Step 1 locks a signal to a specific package pin with schematic properties (as in the LogiBLOX pad symbol) or as a VHDL signal attribute of RT-level source. Step 2 uses Xilinx Alliance Series software to implement the CPLD or FPGA and create a signal/pin map file. Step 3 illustrates the electrical symbol of the FPGA that will appear on the system schematic for production and PCB layout. Pins programmed by NGDBuild appear here on the electrical symbol.

Board Design with Express CIS

OrCAD Express includes thousands of symbols for system-level design and, when used in conjunction with the Component Information System (CIS) option, ensures that correct and complete data about the Xilinx FPGA or CPLD is available for production of the system. **Figure 4** illustrates the part data base explorer which adds important parametric data to the basic electrical FPGA or

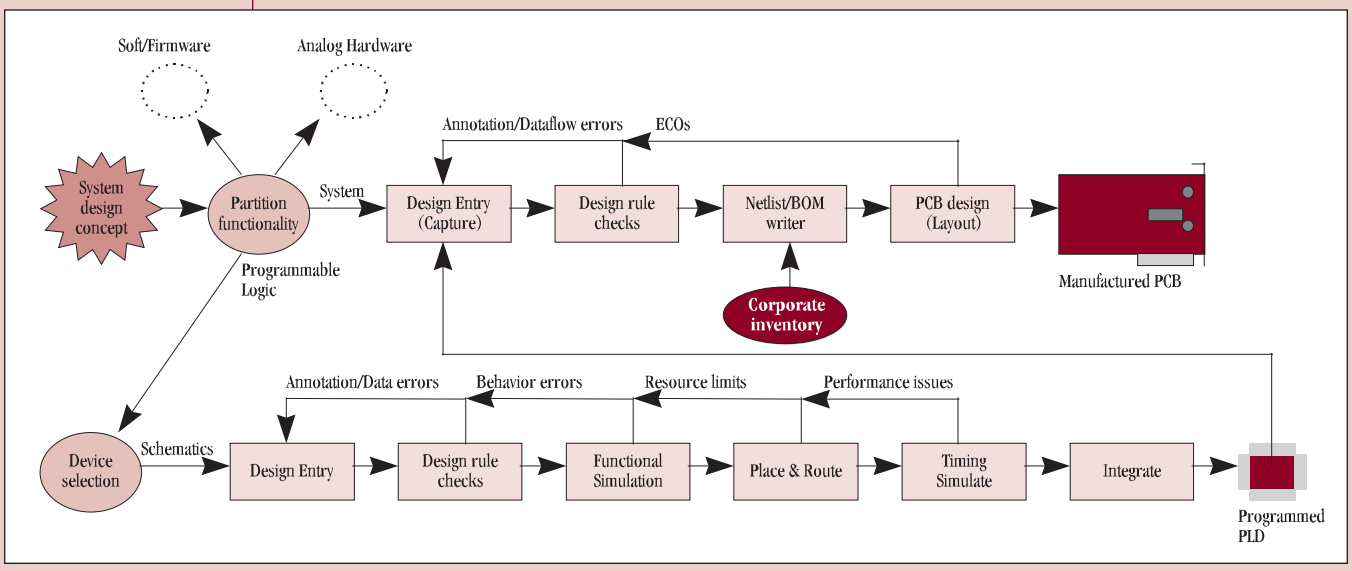


Figure 1. Programmable logic in the system design workflow.

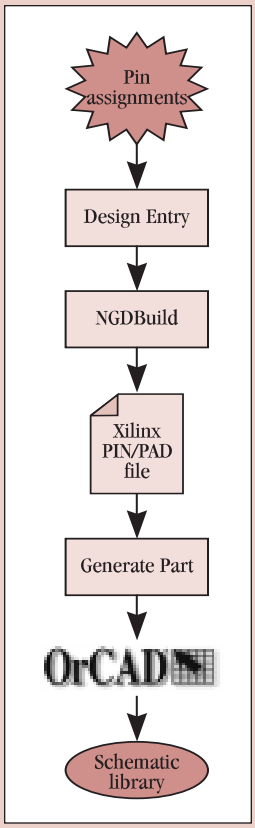


Figure 2. Pin constraints to schematic symbol flow.

CPLD symbol. Properties such as PCB footprint, and corporate inventory part number, that are crucial for the engineering hand-off to production, are transferred to the schematic or merged with the bill of materials.

Simulating Multiple Devices

To confirm the operation of multiple devices created by the Xilinx Alliance Series or Foundation Series software, Express allows you to model the gate-level netlists and VHDL model descriptions of the CPLD or FPGA. Other digital components of the system schematic such as memory or bus interfaces can be modeled with VHDL bus-functional models to verify system behavior.

Simulating a Board Using VHDL Models

Express allows the use of RT-level VHDL models for synthesis as well as a general modeling language for system-level simulation. The Express schematic system makes it easy to get started with automatic model template generation. **Figure 5** illustrates the steps to quickly create a VHDL model template from a symbol on the system schematic.

For more information on Orcad Express, visit www.orcad.com.

Author biography

Troy Scott is the Technical Product Manager for OrCAD Express at OrCAD (Beaverton, Ore.). He received his BSCE degree with Technical Communication Option from the Oregon Institute of Technology (Klamath Falls, Ore.). During his six years at OrCAD, Scott has worked in technical services, documentation, testing, marketing, and software engineering. His current interest is high-level design methodologies for programmable logic using EDA technology. ◆

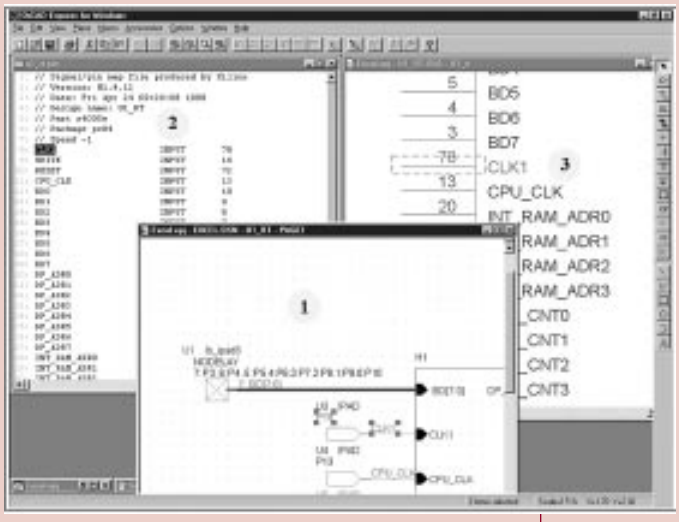


Figure 3. Xilinx pin (PLOC) constraints to symbol generation in OrCAD Express.

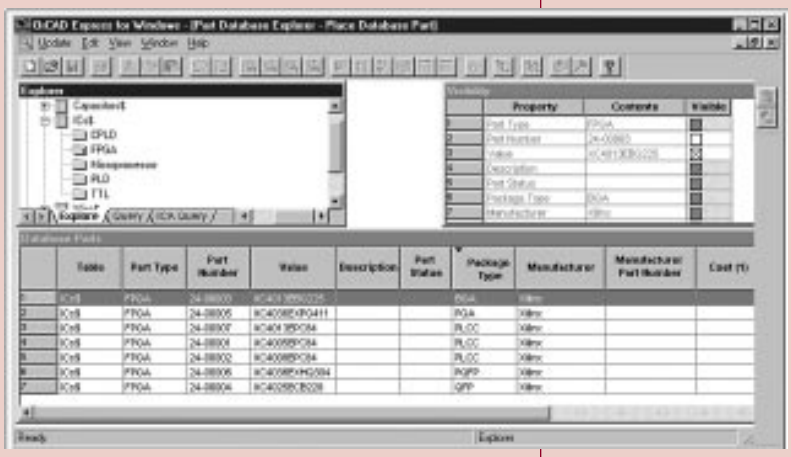


Figure 4. Express CIS part database explorer.

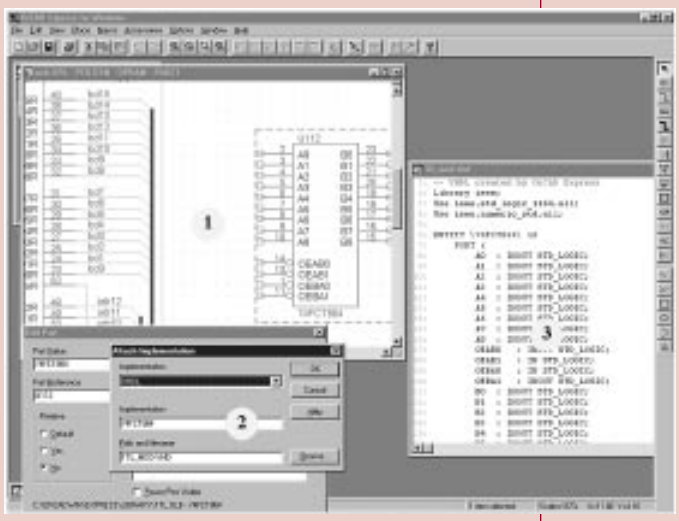


Figure 5. OrCAD Express generates VHDL model template from the schematic editor.

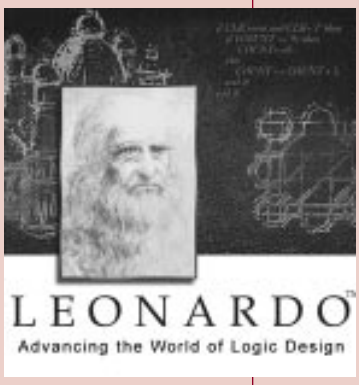
We take you to the leaders.

HDL VERIFICATION SPECIAL SECTION

by Tom Hill, FPGA Relations Manager, Exemplar, tom.hill@exemplar.com

Post-Route Timing Analysis

The Xilinx Alliance Series place and route environment has built-in timing analysis that calculates actual delays for the chip and verifies timing. Leonardo Spectrum can augment the functionality of the Alliance Series software by providing additional functionality such as critical path identification, schematic correlation, and cross highlighting. And because Leonardo Spectrum fully supports backannotation of post-route timing information to the Alliance Series environment, you can take advantage of these features to help verify timing for Xilinx devices.



Leonardo Spectrum is the only FPGA synthesis tool on the market that supports backannotated timing analysis for Xilinx Alliance Series place and route. The Alliance Series environment generates post-routed netlists using "simprims" which are special Xilinx primitives used only for simulation. This "simprims" library is built directly into Leonardo Spectrum along with a netlist interface that reads mapped EDIF netlists and SDF backannotation files, all of which are required for post route static timing analysis.

The Advantages of Backannotated Static Timing Analysis

The advantage of performing post place and route static timing analysis with Leonardo is the timing analysis features available to the user. Some of the more important features are described below:

CRITICAL PATH IDENTIFICATION

The Alliance Series place and route function performs delay analysis on the entire design, presenting all paths to you for viewing via reports. Leonardo Spectrum can perform critical path analysis and report only the timing paths that violate your timing constraints.

CORRELATION TO SCHEMATIC AND HDL SOURCE

Leonardo Spectrum not only has the ability to analyze and report a critical timing path but also to generate a concise schematic fragment showing only the nets and instances of the critical path. This provides a powerful analysis tool for locating timing problems. Using Leonardo Spectrum's backannotated timing interface, critical path schematic fragment viewing is available with the actual post-route timing delays. In addition to generation of critical path fragments, Leonardo Spectrum can also generate fan-in and fan-out fragments from any selected net or instance. For example, clock or reset circuits can be reconstructed from a flip-flop being displayed on the critical path fragment schematic.

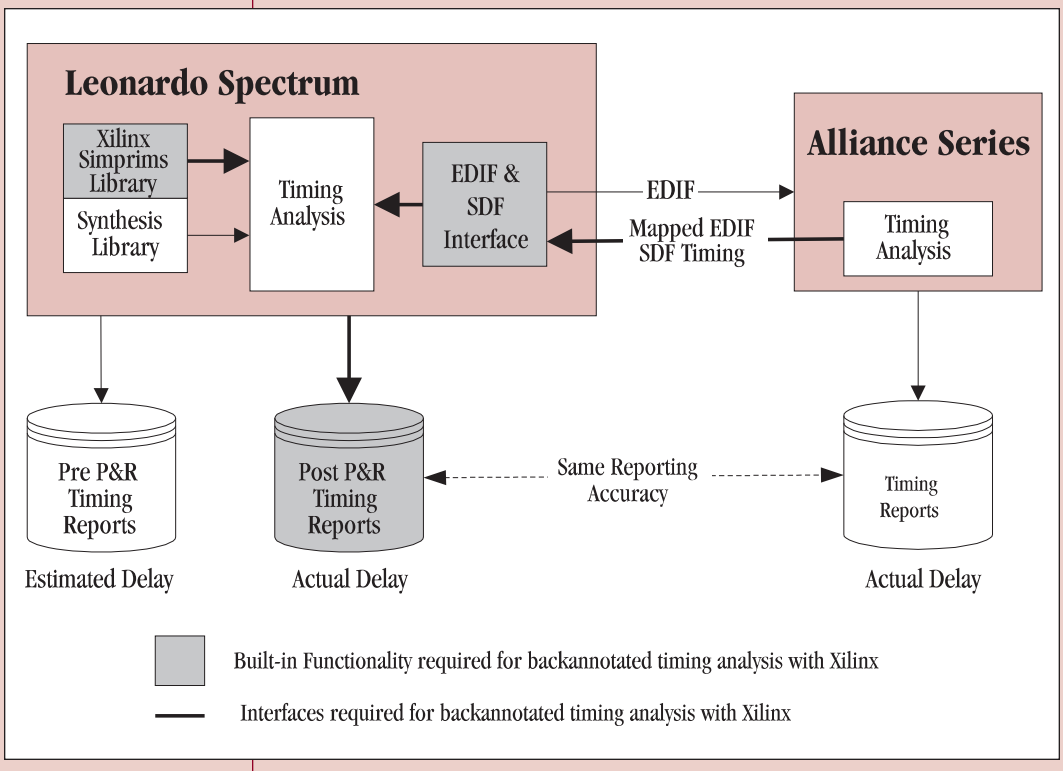
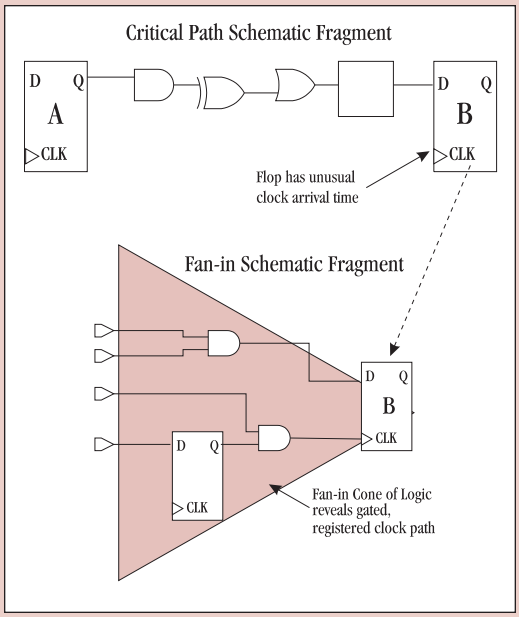


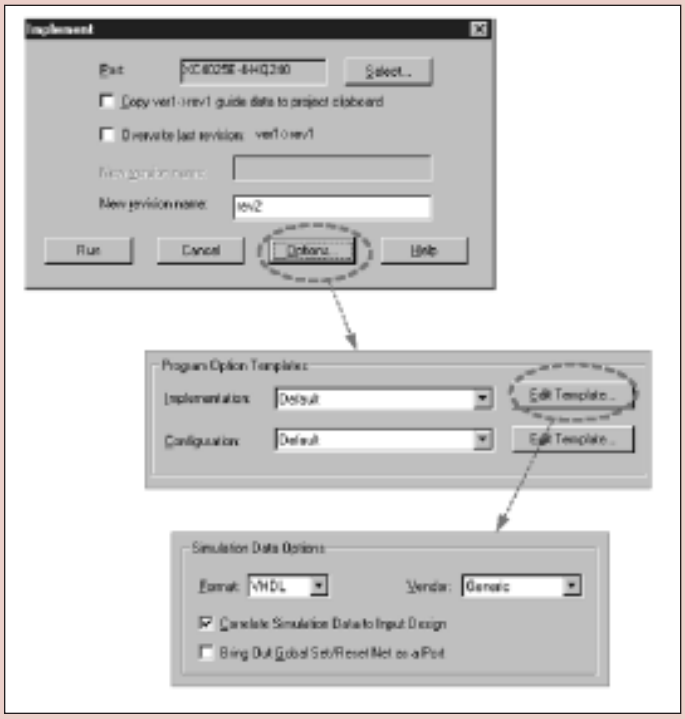
Figure 1 - The Leonardo Spectrum Timing Analysis Environment

with *Leonardo Spectrum*



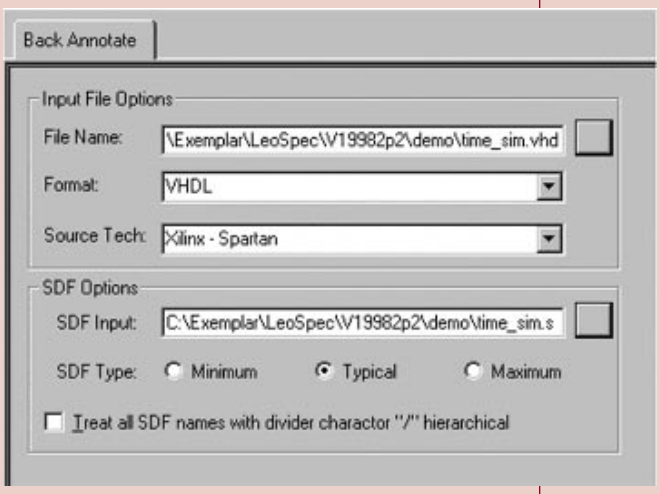
Performing Backannotated Timing Analysis

To perform backannotated static timing analysis with Leonardo Spectrum, the first step is to generate a gate level VHDL or Verilog netlist and an SDF file from the Alliance Series place and route software. To setup the Alliance Series software, perform the following steps:



The Alliance Series place and route software will produce a VHDL file and an SDF backannotation file that can be read into Leonardo Spectrum. Perform the following step to read in. Use Leonardo Spectrum's Back Annotation editor.

And enter in the fields as follows:



At this point you are ready to perform static timing analysis.

Conclusion

Leonardo Spectrum gives you a number of significant advantages when you verify the timing of your designs. ◆

We take you to the leaders.

HDL VERIFICATION SPECIAL SECTION

by Michael A. Bohm,
Exemplar Logic,
VP, Chief Scientist,
bohm@exemplar.com

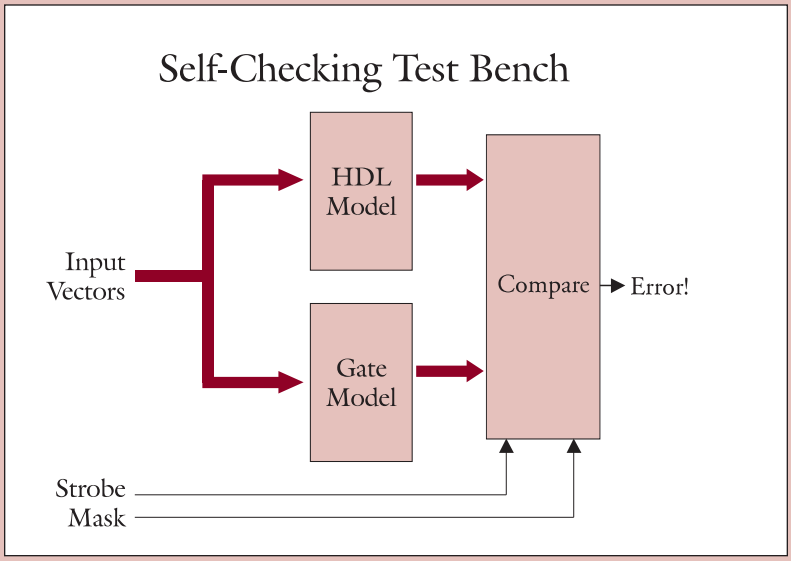
Verification Using a Self-checking Test Bench

As an FPGA designer, your life is basically one big debug cycle. From the moment you receive the first specification from marketing, until production silicon is ready, you are looking for problems.

Once you recognize a problem, then corrective action must be identified. This process of problem identification and correction is the slowest part of the design process, mainly due to the "human factor." Your ability to take in all available information, evaluate the data, and come to a conclusion is one of the main bottlenecks in producing complex FPGAs.

In the days when designs stayed below the 10K-gate barrier, you were capable of handling designs using schematic capture. The amount of design information that you had to control was manageable, with low risk. As designs got larger, Hardware Description Languages (HDL) allowed you to describe an FPGA in a more abstract and compact form. This new form, along with functional simulation, allowed you to define, understand, and build very complex FPGAs.

The HDL developed for an FPGA can be thought of as an executable specification for the device being designed. The problem with this specification is that it only contains the functional information. The AC, DC, and physical information about an FPGA come from the transformation performed during logic synthesis and place and route.



With all the changes that occur from transforming technology-independent HDL to a technology-dependent FPGA device, it is a good design practice to perform a final verification stage before creating silicon. This final verification will prove that the original HDL performs to specification and minimizes the risk in the final device.

One way to perform this verification is through a self-checking test bench. This test harness will not only prove design correctness, but will also provide a structure that is simple and easy to debug. The diagram below illustrates this principle.

The self-checking test bench has three main blocks: the original HDL model, the final gate-level model, and a comparison block. The test bench works by comparing the results from the two FPGA models, which both receive the same input stimulus, and then flags any discrepancies that are found. The real power in this method is that when an error is found, you can probe into both models and trace signals to see what is causing the problem.

The comparison block provides the brains in a testbench. This block performs a logical compare of the primary results of both models. The "strobe" pin on this block decides when data should be valid in both models. This is usually just before the end of the clock period. The "mask" pin tells the comparison block when to ignore data from the models. This is used during the initialization phase of the simulation. Usually the HDL model is in a known state from the beginning of simulation, while the gate level simulation will take a few clock cycles to settle down. The other feature of the comparison block is that it can stop the simulator when an error occurs by executing a VHDL "assert." This is nice feature because the simulator has stopped at the exact point of an error so you can then debug the design.

Conclusion

A self-checking test bench is a great method for performing that final verification stage before creating silicon. It gives you the extra confidence that the design will be correct and smoothly roll into production. ◆

Looking for the Best HDL Design Flow?

by Mary Brown,
Product Marketing,
Alliance Series,
maryb@xilinx.com

and

Paul Ingersoll, Regional
Marketing Manager,
paul.ingersoll@xilinx.com

Xilinx has the easiest, most efficient HDL design flow, as proven by customers in independent trials. To make the point, here is one story about the HDL Design Seminar that was held at Designcon, in San Jose (January 1998).

The objective of the design seminar was to demonstrate HDL design flows to designers that are new to HDL design methodologies and tools by having multiple design teams (a total of 16) design a thermostat controller. The following design flows and tool combinations were used:

Simulation	Synthesis	FPGA Technology
Veribest	FPGA Express	**Xilinx**
MTI	Exemplar	**Xilinx**
MTI	Synplicity	**Xilinx**
MTI	Exemplar	Altera
Viewlogic	Viewsynthesis	Altera
MTI	Synplicity	Altera

Each team was given the design specification and algorithm so they could describe the behavior of the design in either Verilog or VHDL, simulate, synthesize, place and route, and finally download the design bitstream into an FPGA. The target device and speed grade was based on the designers' best judgement. The FPGA would then be inserted into a demo board and a blow-dryer test would be used to judge the successful completion of the design. **Figure 1** illustrates the design flow.

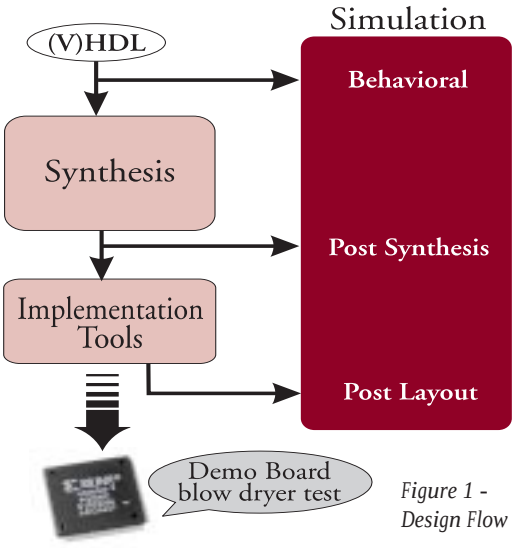


Figure 1 - Design Flow

> *"Each team was given the design specification and algorithm so they could describe the behavior of the design..."*

The two Xilinx-based teams were able to complete the design in record time - approximately two hours. The first Altera team required an additional hour and used twice as much device resources. The second Altera team could not complete the design because Altera's MAX PLUS II software kept issuing a "Device Does Not Fit" error.

Summary of the teams' results:

Rank	Team	Device	Utilization	Design Cycle*
1st	1	Xilinx 4005E	35%	120 mins
2nd	2	Xilinx 4005E	35%	120 mins
3rd	3	Altera 8282	70%	130 mins
4th	4	Altera 8282	Device Does Not Fit	Incomplete

** Design Cycle – Includes writing the HDL, simulation, synthesis, and device programming.*

Conclusion

So, as you can see, Xilinx has the easiest-to-use, the simplest, the most tightly-integrated HDL design flow, with faster runtimes. Of course, tests of this type are very subjective. However, based on these tests and other benchmarks, we are confident you will see similar results. Judge for yourself. ◆

XILINX®

In-Stat Analyst To Discuss ASIC Issues

by Gerald S. "Jerry" Worchel, Principal ASIC Market Analyst, In-Stat, jerryw@instat.com.

This represents the first in a continuing series of articles that will address the application-specific IC (ASIC) market, its design methodologies, and the trends and issues associated with its future. As this is my first column to appear in *XCell*, I feel it's only appropriate to take this opportunity to introduce myself to you. I will also highlight changes that have occurred in the semi-custom ASIC market over the past few years, give an overview of the market's future, and include a preview of what you can look forward to in upcoming issues.

To start, I have nearly 40 years of direct experience in the semiconductor industry. The majority of this time has been on the wafer facility and wafer processing side, in both a technical and managerial capacity. My experience spans the entire breadth of semiconductor process technology, from fab layout and implementation through assembly, test, and yield/failure analysis. I have been an analyst with In-Stat for nine years, covering both the memory and ASIC markets. During this time I have maintained my technical awareness through direct industry contact, attending technical conferences, technical consulting for In-Stat, and reading the plethora of technical literature that is available. Prior to In-Stat I was the founder and President of International Semiconductor Technology (IST), a company specializing in semiconductor technology transfers. So much for me.

The semi-custom ASIC market has undergone major change over the last few years. In-Stat defines the semi-custom ASIC market as being represented by the combined worldwide revenues of the following semi-custom design methodology categories:

- Non-Embedded Gate Array
- Embedded Gate Array
- Non-Embedded User Programmable Logic
- Embedded User Programmable Logic
- Standard Cell
- Linear Array

The semi-custom ASIC market (and technology) has undergone two key *paradigm shifts* in recent years. The first was that cell-based technology replaced Dynamic RAM, as the leading-edge technology driver. The second change to occur, and as important as the first, was that cell-based technology replaced array-based technology in the semi-custom ASIC market. These two shifts alone have forever changed the landscape of the semi-custom ASIC market. For instance, have you noticed that the next-generation process technology advances are now coming first from semi-custom ASIC manufacturers, and not the usual DRAM manufacturers? This fact will not change in the future.

The standard cell design methodology, which has trailed the gate array market in revenues for what once seemed like forever, is now more than twice its size. The requirement for high complexity, high density, high performance designs, with mixed-signal capability, is what has caused this shift in market fates.

These factors will all be explored in much greater depth in future issues of this publication. However, there are far more issues than just these that will affect the future semi-custom ASIC market, and this column will explore many of them. Some of the planned topics for future discussion are:

- Packaging (assembly)
- Test
- Intellectual Property (IP)
- Design
- Process technology
- Wafer fabrication
- Materials

Next issue: *"Embedded Designs; The Myth, The Reality."*

I look forward to our future communiqués, and would welcome your comments and recommendations, as well as your thoughts in general, regarding this column and its future. Please take the opportunity to e-mail me at jerryw@instat.com. ◆

How can I use the "clock enable pin" instead of gated clocks in my HDL designs?

The use of gated clocks can introduce glitches, increased clock delay, clock skew, and other undesirable effects, unless you pay close attention to your design. The first two examples in this article (VHDL and Verilog) illustrate a design that uses a gated clock.

Following these examples are VHDL and Verilog designs that show how you can modify the gated clock design to use the clock enable pin of the CLB.

VHDL Example of a Gated Clock

```
-------------------------------------------------
-- GATE_CLOCK.VHD Version 1.1                   --
-- Illustrates clock buffer control             --
-- Better implementation is to use              --
-- clock enable rather than gated clock         --
-- May 1997                                     --
-------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity gate_clock is
port (IN1,IN2,DATA,CLK,LOAD: in STD_LOGIC;
OUT1: out STD_LOGIC);
end gate_clock;

architecture BEHAVIORAL of gate_clock is
signal GATECLK: STD_LOGIC;
begin
GATECLK <= (IN1 and IN2 and CLK);
GATE_PR: process (GATECLK,DATA,LOAD)
begin
if (GATECLK'event and GATECLK='1') then
if (LOAD='1') then
OUT1 <= DATA;
end if;
end if;
end process; --End GATE_PR
end BEHAVIORAL;
```

Verilog Example of a Gated Clock

```
//////////////////////////////////////////
// GATE_CLOCK.V Version 1.1             //
// Gated Clock Example                  //
// Better implementation to use clock   //
// enables than gating the clock        //
// May 1997                             //
//////////////////////////////////////////
module gate_clock(IN1, IN2, DATA, CLK, LOAD,
 OUT1) ;
input IN1 ;
input IN2 ;
input DATA ;
input CLK ;
input LOAD ;
output OUT1 ;
reg OUT1 ;
wire GATECLK ;
assign GATECLK = (IN1 & IN2 & CLK);
always @(posedge GATECLK)
begin

if (LOAD == 1'b1)
OUT1 = DATA;
end
endmodule
```

VHDL Example of a Clock Enable Pin

```
-------------------------------------------------
-- CLOCK_ENABLE.VHD                           --
-- Illustrates clock use of clock             --
-- enable rather than gated clock             --
-- May 1997                                   --
-------------------------------------------------
-- CLOCK_ENABLE.VHD
-- May 1997
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity clock_enable is
port (IN1,IN2,DATA,CLOCK,LOAD: in STD_LOGIC;
DOUT: out STD_LOGIC);
end clock_enable;
architecture BEHAV of clock_enable is
signal ENABLE: STD_LOGIC;
begin
ENABLE <= IN1 and IN2 and LOAD;
EN_PR: process (ENABLE,DATA,CLOCK)
begin
if (CLOCK'event and CLOCK='1') then
if (ENABLE='1') then
DOUT <= DATA;
end if;
end if;
end process; -- End EN_PR
end BEHAV;
```

Verilog Example of a Clock Enable Pin

```
/*
//////////////////////////////////////////
// CLOCK_ENABLE.V                       //
// Example of use of clock enables      //
// rather than gating the clock         //
// May 1997                             //
//////////////////////////////////////////
*/
module clock_enable (IN1, IN2, DATA, CLK,
 LOAD, DOUT);
input IN1, IN2, DATA;
input CLK, LOAD;
output DOUT;
wire ENABLE;
reg DOUT;
assign ENABLE = IN1 & IN2 & LOAD;
always @(posedge CLK)
begin
if (ENABLE)
DOUT <= DATA;
end
endmodule
```

◆

THE XILINX HDL ADVISOR

by Roberta Fulton, Technical Marketing Engineer, Alliance Series, roberta.fulton@xilinx.com

LogiBLOX

I get an error from NGDBUILD about X_FF or some other SIMPRIM type primitive being an invalid module from NGDBUILD in my LogiBLOX design — what could be the cause?

If you specified "EDIF simulation netlist" as one of your output formats for your LogiBLOX module, you may see such an error. For the version 1.4 release, you will need to delete the EDIF file that you generate using LogiBLOX for functional simulation before trying to implement the design.

The version 1.5 of LogiBLOX will write the implementation netlist to a file with the ".ngc" extension instead of ".ngo". This change was made because otherwise NGDBUILD finds the EDIF simulation netlist (".edn" file) created by LogiBLOX for that module, and tries to use it as the source netlist to recreate the ".ngo" file. Since the LogiBLOX-created EDIF netlist is intended only for simulation, and does not contain the required implementation information, NGDBUILD (EDIF2NGD) will reject the EDIF and issue an error.

To eliminate this problem, the version 1.5 of LogiBLOX will write the implementation netlist to a file with the ".ngc" extension (instead of ".ngo"). When NGDBUILD finds a ".ngc" file to define a module, it will use this file directly, and not look for any associated "source" netlists. In all other respects, the ".ngc" file serves the same purpose as the ".ngo" file.

Reference: http://www.xilinx.com/techdocs/3904.htm

COREgen

What can I do if I'm having problems with, or can't remember, my WebLINX/CoreLINX password, or if I have problems registering on the CoreLINX website?

If you are having trouble registering on the WebLINX site, please send an email to webmaster@xilinx.com describing the problem. Be sure to include details on the exact sequence of steps you are going through, and note exactly where it fails.

If you are having trouble logging in to the Xilinx website, remember that passwords are case sensitive. If you cannot remember your password, you can go to the Xilinx site, click on the **Agent's** link in the top button bar, hit **Cancel** in the password dialog, then click on the link to "Please send me a new password."

This will bring up a form where you can enter your email address. WebLINX will automatically send you a new password.

Reference: http://www.xilinx.com/techdocs/3935.htm

What versions of Viewlogic, Foundation, Synopsys, Express are supported by COREGEN?

Versions of Viewlogic, Foundation, Workview Office, FPGA Express, and Synopsys FPGA Compiler required by COREGEN were inadvertently omitted from the User Guide.

COREGEN v1.4.0 supports the following platforms:

- Viewlogic Powerview v6.0 or later
- Workview Office v7.4
- Foundation vF1.4 (REQUIRED)
- Synopsys FPGA Express v2.0 (v2.1 preferred)
- Synopsys FPGA Compiler v1997.01

Reference: http://www.xilinx.com/techdocs/3884.htm

Using Foundation, what can I do when I encounter Lmacs or Btrieve errors?

Foundation Series Software

Most often, Btrieve and Lmacs errors are a result of conflicts associated with the Btrieve software. Btrieve is a Windows database software program, which is used by the Foundation Library Manager. Btrieve may also be used by other Windows software, which are unrelated to the Foundation software. If this other software uses a different version of the Btrieve software than the Foundation software uses, conflicts may exist, and Lmacs or Btrieve errors may be issued by Foundation. Often, the errors involve Foundation not being able to locate the proper library files.

First, check your WINDOWS directory for any of the following files:

- Wbt32res.dll
- Wbtrlocl.dll
- Wbtr32.exe
- Wbtrcall.dll
- Wbtrvres.dll

The Foundation install program writes the above files to the c:\windows directory by default.

Search for the same .dll and/or .exe files in the c:\windows\system directory. If they are also found here, there is a conflict between the differenct versions of Btrieve on your PC. Remove these files from c:\windows\system.

It is also possible that there is an incompatible version of Btrieve being loaded by Windows. This incompatible version may have been installed by another windows program. An easy way to attempt to resolve this problem is to copy the Btrieve files directly from the Foundation CD-ROM into the Windows directory.

From the Foundation Design Entry Tools CD-ROM, go to the FNDTN\ACTIVE\BTRIEVE directory, and copy the following all of the files in that directory into your local Windows directory.

How do I compile the Simulation Libraries for the Model Technology Simulator?

Model Technology

If you are using Modelsim to simulate VHDL or Verilog, the Xilinx simulation libraries need to be compiled first, before a simulation may be performed. The simulation libraries may be compiled to the project's working directory; however after creating a few Xilinx projects, this redundant library compilation may take up more time and disk space than necessary. Generally it is much more efficient to compile the libraries to a central area and point to the libraries via the modelsim.ini file. Solution record #1923 available on the Xilinx website at http://www.xilinx.com/techdocs/1923.htm explains how to compile the simulation libraries in this more efficient manner.

I want to create a group of FFs which power-up in a certain state. How can I do this in HDL without creating an extra port in my design using Alliance1.4?

HDL State Machine Technique

The ROCBUF was created for synthesis users who needed to create FFs which would power-up in a '1' or '0' state, but the FF would not have an asynchronous reset or set pin. FFs in XC4000 type devices with an asynchronous reset pin will power-up as a '0'. FF's in XC4000 type devices with an asynchronous set pin will power-up as a '1.'

By describing FFs with an asynchronous set or reset pin, you can create a group of FFs that power-up in a known pattern, like "10101111." If you want FFs with asynchronous reset or set pins, this is an easy task, because the HDL will describe this behavior. But, if you do not want or do not need these FFs to have a asynchronous reset or set pins, you must still describe, in the RTL code for the FFs, an asynchronous reset or set pin.

By connecting the HDL code which describes the asynchronous reset or set pin of an RTL described FF to the ROCBUF, you can create FFs that power-up in a known state. The ROCBUF will not synthesize to logic. So, even though the ROCBUF is connected to a top-level port, no extra pin will be added to a design. The top-level port the ROCBUF is connected to will not be implemented.

Continued on the following page

HDL State Machine Technique

Continued from the previous page

VERILOG EXAMPLE OF USING THE ROCBUF:

```
module
stmchine(CLK,RESET,STRTSTOP,CLKEN,RST);

input CLK;
input RESET;
input STRTSTOP;
output CLKEN;
output RST;

reg CLKEN;
reg RST;
wire rstWire;

parameter [5:0] //synopsys enum
STATE_TYPE
 clear=6'b000001,
 zero=6'b000010,
 start=6'b000100,
 counting=6'b001000,
 stop=6'b010000,
 stopped=6'b100000;

reg [5:0] current_state;
reg [5:0] next_state;

always@(current_state or STRTSTOP)
begin
    case(current_state) //synopsys
full_case parallel_case
    clear:begin
        next_state<=zero;
        CLKEN<=1'b0;
        RST<=1'b1;
        end
    zero:begin

 next_state<=(STRTSTOP)?start:zero;
        CLKEN<=1'b0;
        RST<=1'b0;
        end
    start:begin

next_state<=(STRTSTOP)?start:counting;
        CLKEN<=1'b0;
        RST<=1'b0;
        end
    counting:begin

next_state<=(STRTSTOP)?stop:counting;
        CLKEN<=1'b1;
        RST<=1'b0;
    end
    stop:begin

next_state<=(STRTSTOP)?stop:stopped;
        CLKEN<=1'b0;
        RST<=1'b0;
    end
    stopped:begin

next_state<=(STRTSTOP)?start:stopped;
        CLKEN<=1'b0;
        RST<=1'b0;
    end
    endcase
 end

 always@(posedge CLK or posedge
rstWire)
 begin
    if(rstWire==1'b1)
    current_state = clear;
    else
    current_state = next_state;
 end

ROCBUF U1 (.I(RESET),.O(rstWire));

endmodule
```

VHDL EXAMPLE OF USING THE ROCBUF:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_unsigned.all;

entity smallcntr is

 port ( CE : in STD_LOGIC;
    CLK : in STD_LOGIC;
    CLR : in STD_LOGIC;
    QOUT : out STD_LOGIC_VECTOR(9
downto 0)
    );
end smallcntr;

architecture inside of smallcntr is

signal qoutsig : STD_LOGIC_VECTOR(9
downto 0);
signal clrSig: STD_LOGIC;

component ROCBUF
port(I: in STD_LOGIC; O: out
STD_LOGIC);
end component;

begin

process(CE,CLK,clrSig)
 begin

 if(clrSig='1') then
   qoutsig <="0100010001";
 elsif(CE='1') then
   if(CLK'event and CLK='1') then
      qoutsig<=qoutsig +
"0000000001";
   end if;
 end if;
 end process;

QOUT<=qoutsig;

U1: ROCBUF port
map(I=>CLR,O=>clrSig);

end inside;
```

How does the EXTEST instruction work in the XC4000/XC5200 series devices?

The XC4000 and XC5200 series devices implement the *IEEE 1149.1*-compatible EXTEST instruction. Loading a bit sequence "000" in the Boundary Scan Instruction register (IR) will enable the EXTEST instruction.

Figure 1 shows the Boundary Scan Logic in a typical IOB. The Boundary Scan Data register (DR) is a serial shift register implemented in the IOBs. Each IOB can be configured as an independently controlled bi-directional pin. Therefore, three data register bits are provided per IOB: for input data, output data, and 3-state control.

An update latch accompanies each bit of the DR, and is used to hold test data during shifting of new test data. The update latches get updated during the *Update-DR* State of the TAP controller.

To execute the EXTEST instruction, shift the bit pattern "000" into the IR in the *Shift-IR* state of the TAP controller via the TDI pin. This instruction will become current in the Update-IR State and the EXTEST line will get asserted. At this time, data in the input bit of the DR gets driven on to the FPGA interconnect (IOB.I) and data in the output and 3-state control bits gets driven to the device pins.

In the *Capture-DR* State of the TAP controller, data from the device pins goes to the DR input bit (Shift/Capture line de-asserted); it gets captured in the IOB flip-flops. Care should be taken to make sure that the output bit of the DR has been 3-stated at this point, otherwise there will be contention on the pin and unknown data will get captured. The output and 3-state bits of the DR capture the data coming from the FPGA interconnect in this state (IOB.O and IOB.T).

This captured data can be shifted out for inspection on the TDO pin during the *Shift-DR* State of the TAP controller (Shift/Capture line asserted).

Note 1: The *IEEE* standard *1149.1* does not require an internal injection of data to the device interconnect during the *Update-IR* state. However, this capability helps to compensate for the lack of INTEST support.

Note 2: The Update latches are accessed every time the TAP controller is in the *Update-DR* state, regardless of the instruction. Care must be taken to ensure that appropriate data is contained in the update latches prior to initiating an EXTEST instruction. Any instruction, including BYPASS, that is executed after the test data has been loaded, but before the EXTEST instruction becomes current, changes the test data. ◆

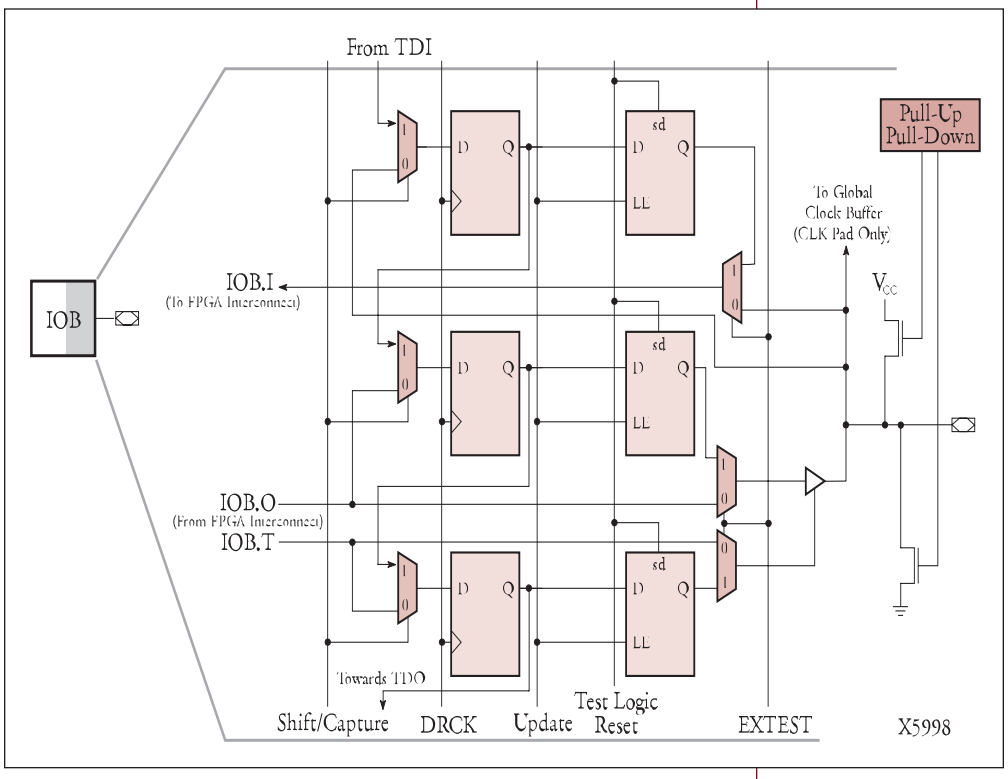


Figure 1 – Boundary Scan Logic

Corporate Headquarters
Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Tel: 408-559-7778
Fax: 408-559-7114

Europe
Xilinx, Ltd.
Benchmark House
203 Brooklands Road
Weybridge
Surrey KT14 0RH
United Kingdom
Tel: 44-1-932-349401
Fax: 44-1-932-349499

Japan
Xilinx, KK
Daini-Nagaoka Bldg. 2F
2-8-5, Hatchobori,
Chuo-ku, Tokyo 104
Japan
Tel: 81-3-3297-9191
Fax: 81-3-3297-9189

Hong Kong
Xilinx Asia Pacific
Unit 4312, Tower II
Metroplaza
Hing Fong Road
Kwai Fong, N.T.
Hong Kong
Tel: 852-2424-5200
Fax: 852-2494-7159

Rome wasn't built in a day. Your ASIC can be.

SPARTAN™

FPGA

www.xilinx.com

PN:XLQ398

XILINX®

2100 Logic Drive
San Jose, CA 95124-3450

First Class Presort
U.S. Postage
PAID
Permit No. 2196
San Jose, CA