XAPP408 (v1.2) February 15, 2002

Rethinking Your Verification Strategies for Multimillion-Gate FPGAs

Author: Thomas D. Tessier, T2 Design

Summary

Verification is an integral part of any FPGA design project. Many older verification models are no longer appropriate to the new multimillion-gate FPGAs, and more modern methods must be brought to bear if verification is to positively affect product time to market. The methodologies used for designing and implementing a good verification plan are discussed in detail, in the context of a real-world verification case study.

Introduction

FPGA verification is essential for successful time-to-market product delivery. But how do you know if your current verification techniques are the best choices for today's high-density FPGAs? Million-gate FPGAs require designers to rethink their verification strategies. The most effective way to alter the validation procedure to meet today's high gate-count requirements depends on the designer's background and experience.

Traditionally, engineers who have been using FPGAs since technology's migration from schematics to HDLs continue to use simulator-specific approaches to verification. The simulation tools are primarily used for module testing, while the lab is used for "in-system" test. This approach often requires the engineer to manually stimulate signals by toggling them and viewing the waveform responses. Since this process is time consuming, error prone, and difficult to repeat, designers will often spend minimal time in simulation, moving quickly to the lab where they can debug and modify until release. The functions implemented in million-gate FPGAs are far too complex to permit continued reliance on this ad hoc method.

Designers are choosing million-gate FPGAs because they are fast enough and large enough to handle complex designs that were previously only achievable with an ASIC. When engineers experienced in ASIC design move to high-density FPGAs, they take their verification approaches with them. Those who use a validation process featuring robust tools and a complete self-checking testbench environment find that hanging on to their familiar testing approaches now causes them to lose valuable design cycle time. ASIC designers can benefit from a carefully defined and executed verification plan that takes into consideration the ability to reprogram the FPGA. Time that was once beneficially spent in exhaustive verification of an ASIC at the RTL level now becomes less advantageous, and is actually costly in time-to-market terms for a high-density FPGA.

Throughout this paper, the term *ASIC* is defined as a traditional gate-array or standard-cell ASIC. *FPGA* is defined as a Field Programmable Gate Array such as a member of the Xilinx Virtex™ Family. Although the terms ASIC and FPGA can sometimes be used interchangeably, in the context of this paper they are not.

What Is Verification?

Verification is not synonymous with simulation. Verification is a *strategy* to make sure all aspects of the system meet the specification document, while simulation is a *tool* used in the verification effort. The basic components of verification are shown in Figure 1, page 2.

Specification

The specification document is the foundation of the test plan. Typically, specifications for a new product will be introduced as "the same thing we did last time, only faster and with some added features." The details of what was done last time are in someone's head and not written down anywhere. The specification describes the features to be implemented, under what conditions they occur, and what their expected output should be. This documentation should stop short of dictating implementation; that is left for the experience of the RTL designers. Nevertheless, a complete, detailed written specification is essential to meeting time to market.

Verification Plan

From the specification, the Verification Plan is developed. As the design goes through its various stages of evolution, the verification strategy insures *Design Correctness (Functionality)* and *Design Performance (Timing)*.

Designers familiar with ASIC verification approaches will realize that design size and power consumption are not issues with FPGAs. With an ASIC, the NRE is dependent upon the area of the design; with an FPGA, the designer often has the ability to increase to the next size part with only a minimum increase in unit cost. With an ASIC, power dissipation is often a problem in larger designs; with an FPGA, the FPGA vendor has already determined the maximum operating conditions, including power. The designers using FPGAs have fewer manufacturing-specific details to worry about.

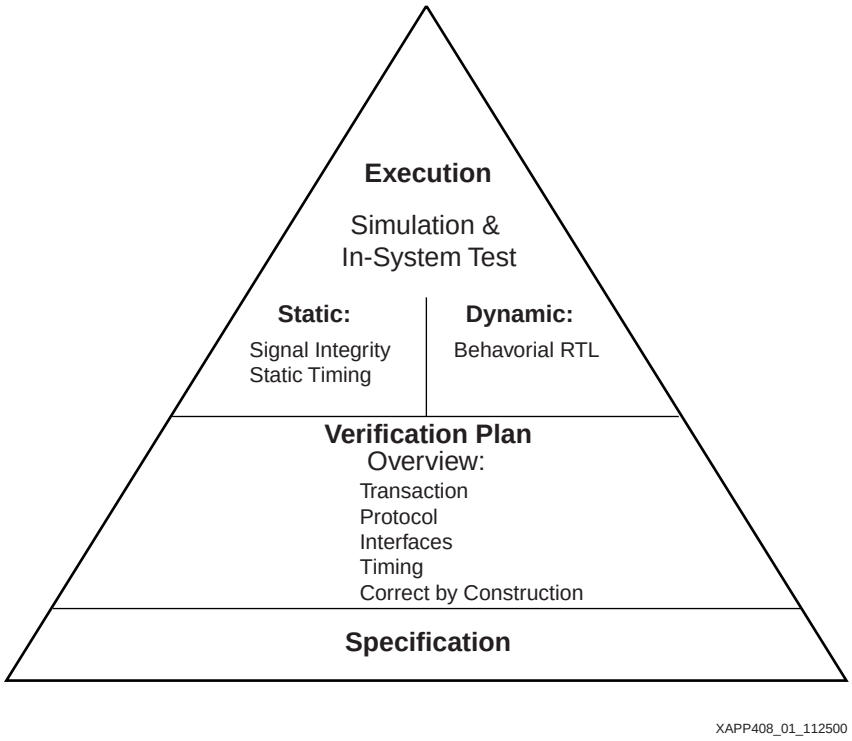

Figure 1: **The Verification Pyramid**

Both the design and verification teams need to be working from the same thorough specification. RTL engineers and verification engineers share the responsibility for implementing the test plan. The level of test granularity (detail) is outlined at the transaction, protocol, interface, and timing levels. Essential functions are identified. The conditions under which key features come into play, and their expected responses, are described. A determination is made as to the number of testbenches needed, their complexity, and test module dependencies.

Testbenches are written to validate the specification, not to verify the design implementation. This is an important point. Any discrepancies in design implementation versus testbench results should be referred back to the specification for clarification. This is not a new concept, but it is often overlooked in the rush to produce a product. When all elements described within the test plan are checked off, the verification effort has been completed to the required level of confidence.

To optimize the verification effort, the following list offers examples of the type of information you need to identify:

- External interfaces
  - Stimulus and response
  - Transaction level — e.g., Read versus Write operations
  - Timing requirements
- HDL models available to assist in testbench development
  - Packaged with proposed intellectual property (IP)
  - Bus functional models
  - Vendor supplied — e.g., memories
- Tools available to the project
  - Simulators
  - Static analysis
  - Testbench tools
  - Lab-based tools
  - Scripting and makefiles for data management
  - Computer resources adequate to do the job
- Functional interaction
  - Which internal interfaces interact, and is the interaction observable via the outside interfaces?
- Performance requirements — e.g., need 32-block data write @ 66 MHz with a latency of less than 300 as.
- Tradeoff: simulation versus in-system

Execution

As part of developing a verification strategy that best suits your design, try to break out those functions that are essential to simulate from those that can be tested during in-system test. The execution of the verification plan requires simulation *and* in-system test on the target PCB—the final stages of the pyramid.

Dynamic simulation describes what most designers visualize when they think of simulation: Behavioral HDL, RTL, and Gates.

- **Behavioral HDL:** non-synthesizeable constructs that are used as executable specifications and also for testbenches
- **RTL:** the functional logic of the design using synthesizeable constructs
- **Gates:** result of the synthesis and P&R process

Behavioral HDL is implemented more quickly than RTL, is easier to understand, and can be simulated faster. Behavioral simulation makes sure your HDL code is valid, and detects functional problems before synthesis. Million-gate FPGAs take a long time to simulate at the gate level and, depending upon the complexity of function, even at the RTL design level. Working out issues in pre-synthesis using a behavioral model saves project time. "What-if" scenarios are easily created at the behavioral level, and execution tradeoffs are more quickly evaluated. Testbenches should be written in behavioral-based HDL.

RTL is the code that is synthesized to produce gates. The level of detail can extend down to the bits in registers and combinatorial gate relationships. A behavioral testbench allows the designer to dynamically simulate the RTL until it meets the functionality specified in the test plan. Once validated, the RTL is ready for synthesis.

Gates are the lowest level of abstraction, and contain the highest level of detail. The same behavioral testbench is used to simulate gates after synthesis and Place and Route. The results of the gate simulation are compared to the RTL simulation output to ensure that the design still meets the criteria specified in the test plan. The design team may pick a subset of the test cases to run at the gate level; the equivalency of the RTL and gates can only be guaranteed if the entire suite is run or formal verification is employed. Simulations at this level run much slower than at pre-synthesis; therefore, it may not be time-feasible to run the entire testbench suite.

Static Analysis includes: Static Timing Analysis (STA), Formal Verification and Signal Integrity Analysis.

- **Static Timing Analysis:** at the PCB Level and the output of P&R for the FPGA only
- **Formal Verification:** quickly validating the design functionality at the transformation points, Synthesis, and P&R
- **Signal Integrity Analysis:** at the PCB Level

All of these techniques use mathematical approaches (i.e., no test vectors) to validate the functionality and timing of the finished design.

For PCB-level *Signal Integrity* and *Timing Analysis,* Xilinx provides IBIS models that define I/O characteristics used by signal integrity applications, as well as STAMP models for specifying setup/hold times, output delay times, and timing relationships.

Static Timing analysis is used to validate the timing of the design. Each P&R of the design will result in its own “timing environment” which needs to be validated to ensure that the design will work in the system. It is imperative that the designer develop good timing constraints that represent the actual environment in which the FPGA will operate. These timing constraints will drive both the synthesis process and timing-driven P&R (if used), and will validate the FPGA timing using STA. The post P&R STA is often the downfall of many designs. Engineers tend to look only at the maximum operating frequency, but ignore the I/O timing. A functional design will not work if it does not meet the system-level timing constraints.

The implementation of the timing constraints is driven by the tools available. Timing constraints can be specified, however, independently of the tools available. An example of a written timing constraint specification is:

> Address will be stable 4 ns before the rising edge of clock133 (setup) and 1 ns after the rising edge of clock133 (hold).
>
> Clock133 is a 133 MHz clock with a 50% duty cycle.

Formal verification is considered an emerging technology and beyond the scope of this article.

In-System Test

How do you decide what to simulate and what to validate at in-system test? ASIC design teams use the rule “If it isn’t tested, it’s broken” to guide the simulation approach of testing every register bit. Using HDL approaches for FPGAs allows a design team the flexibility of dividing the function verification tasks into traditional simulation and in-system testing. This approach will save valuable time by eliminating the extensive gate-level simulation of the complete regression suite. It is not necessary that every gate is toggled; we will explore a test case later that highlights a way to look at the verification problem where in-system testing is a valuable component. Manufacturing (ATPG) or JTAG vectors are not an issue for FPGAs because the silicon is already verified by the vendor.

We are expecting to utilize the reprogrammability feature of the FGPA in the target system during this verification phase. At this point, it is expected that module-level partitions have been tested for functionality, and that module interfaces are stable and well defined. The focus of the

verification effort here is the transaction and protocol of the interfaces. The exhaustive external interface testing is done using in-system testing.

During in-system testing, designers have a distinct advantage when using FPGAs over ASICs. An obvious benefit is the ability to reprogram the FPGA until the desired functionality is achieved. Xilinx FPGAs provide an additional advantage: With ChipScope ILA, the user can observe internal nodes of the chip, on the PCB, while running at system speeds. For ASIC-targeted designs, additional test structures are needed to provide this level of observability, often combined with scan chains for silicon validation. In addition to increased design complexity, the test-specific circuitry must be simulated.

Interaction of Verification Components

Once the design's executable specification and the testbench (both written in behavioral HDL) meet requirements, the design is replaced with RTL code. The RTL is then verified with the system-level testbenches to make sure it meets the written specification conditions. After the RTL is validated, it is synthesized and processed by the P&R tools. The resulting gates are plugged into the system verification testbenches, or into formal verification if it is available. This insures the tools have correctly implemented the design. In addition, generated gates are run through static timing analysis. This step verifies that the system-level timing is met. (Figure 2)

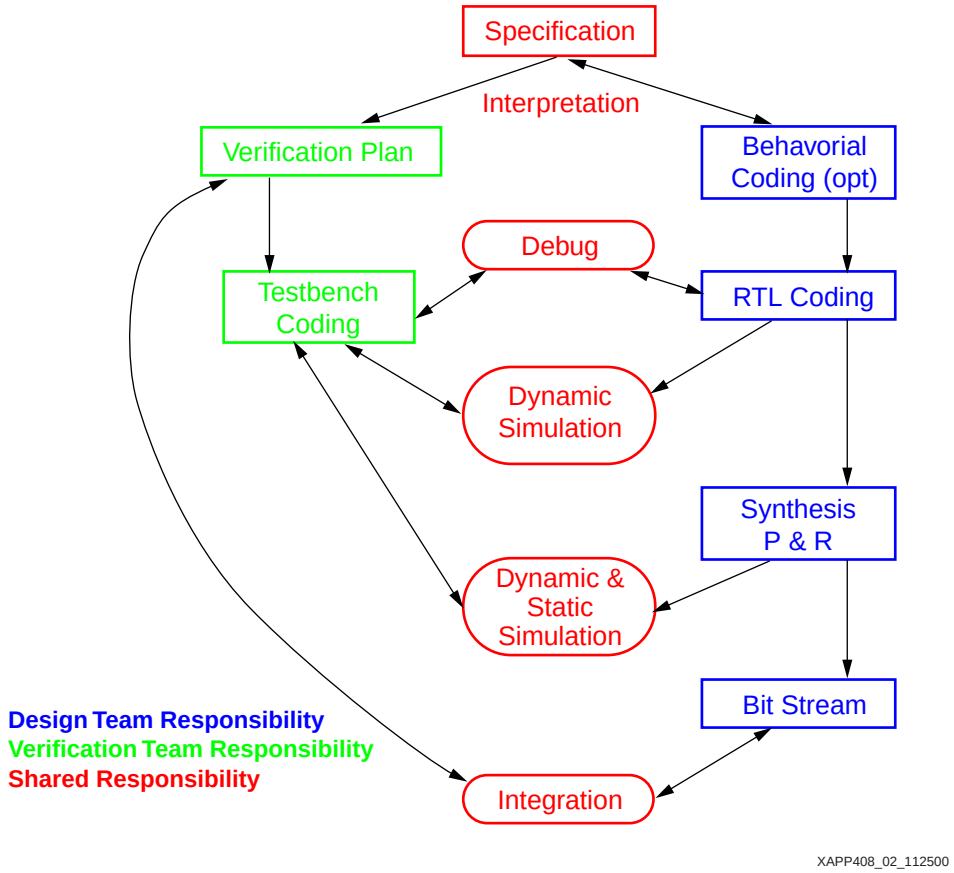

Figure 2: **Interaction of the Verification Components**

System integration is typically referred to as *power on*. This is the time when project teams come up with creative answers to the question, "Is it working yet?" Projects are ready for in-system test when they have validated RTL code, have been successfully placed and routed, and can create a bitstream to program the FPGA on the physical PCB. At this point, it is expected that module-level partitions have been tested for functionality, and that module interfaces are stable and well defined. The design has been simulated as a chip at both RTL and gate levels, with the minimum functionality necessary for power on.

Simulation of the chip is frequently not achieved by FPGA design teams when simulator-specific approaches are used. The case study will outline why this step is important to meeting time to market.

Verification Case Study

Telecom Header Processor

To explore available verification techniques in a real situation, we developed a case study. This example FPGA will highlight successful approaches for verification.

The design is an ATM statistics-gathering processor. The ATM front end captures a portion of the header. Once the header is captured, it is placed in a FIFO to be stored in the SDRAM data space via a DMA or burst function. The memory controller arbitrates between the SPARC, the PCI bus, and the FIFO for memory access. The PCI bus is provided to preload the type of data that will be accessed as well as the actual SPARC code. The PCI bus will interface to a set of registers targeted at the SPARC. These registers provide communication between the SPARC and the PCI host controller.

Although there is a defined process and order to putting together a verification strategy, there are design-dependent decisions to be made that may reduce the verification time and effort without compromising functionality. The goal is to allow insight into the verification methodology by creating a strategy and applying it to the test case (Figure 3).

The intended design meets the challenges facing today's engineers. The design contains several pieces of IP: a SPARC, the PCI bus, and CoreGen memories. The Utopia interface and the SDRAM controller are original designs.

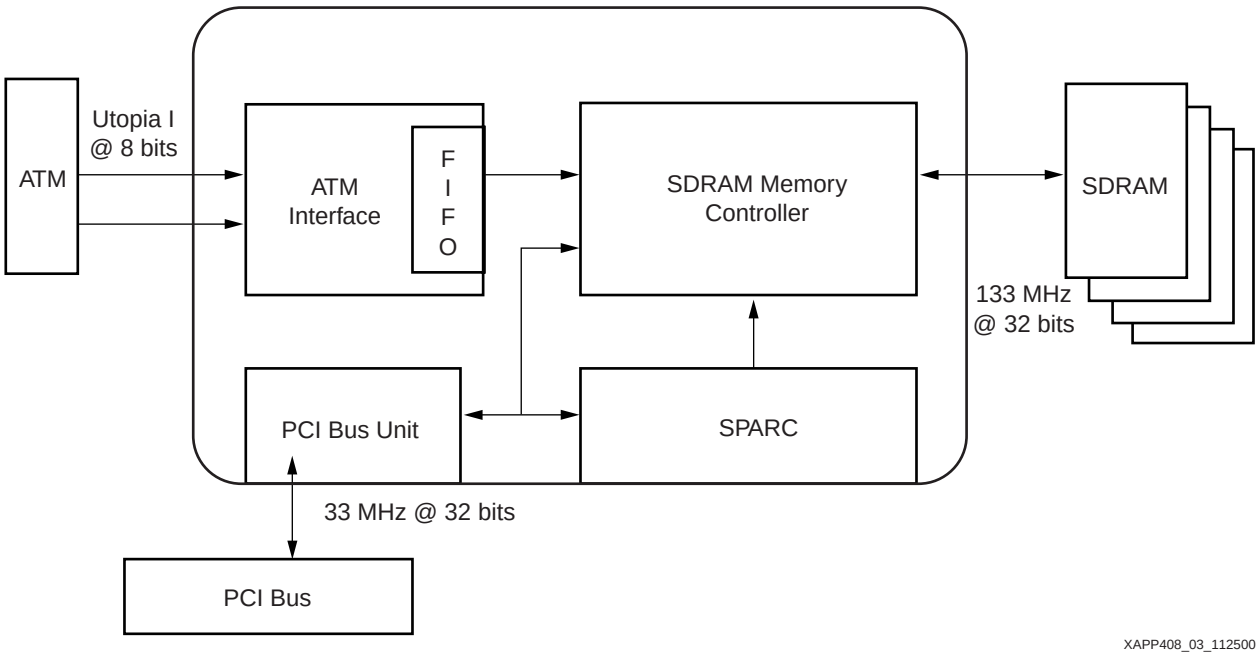

Figure 3: **Block Diagram of Testcase**

Specification

A system-level specification (see references) was developed for the test case project. The specification was reviewed by the architects, design team, and verification members. Our philosophy is to always have an engineer other than the RTL designer write the verification testbench. The specification guides both the design and the verification effort. By applying two different teams to the task, we increase the likelihood that the design will be completed as specified. Once the specification has gone through a review process, we are ready for the verification plan.

Verification Plan

Using the specification as the baseline, a verification plan is developed (see references). The verification plan includes what will be tested and how. For this project, our decision process consists of “thinking out loud” about how to split the verification plan into simulation and PCB verification efforts.

Verification Decisions

The case study will provide insight into the verification discovery process. This section includes the tradeoffs made on the project while developing the verification plan. Furthermore, several interesting aspects of simulating and integrating the project are included.

External Interfaces

The FPGA has several external interfaces (see Figure 4): a PCI bus running at 33 MHz, an SDRAM interface running at 125 MHz, and the Utopia interface running at 8 MHz. The timing can easily be described by the devices that communicate over these interfaces to the FPGA, so the development of the timing constraints is straightforward.

The transactions are identified as burst read/write, single read/write, and retry/failure modes on the PCI bus, as well as the handshaking signals that control the data flow on the synchronous Utopia interface. The SDRAM interface is synchronous, and the command is already built as a transaction.

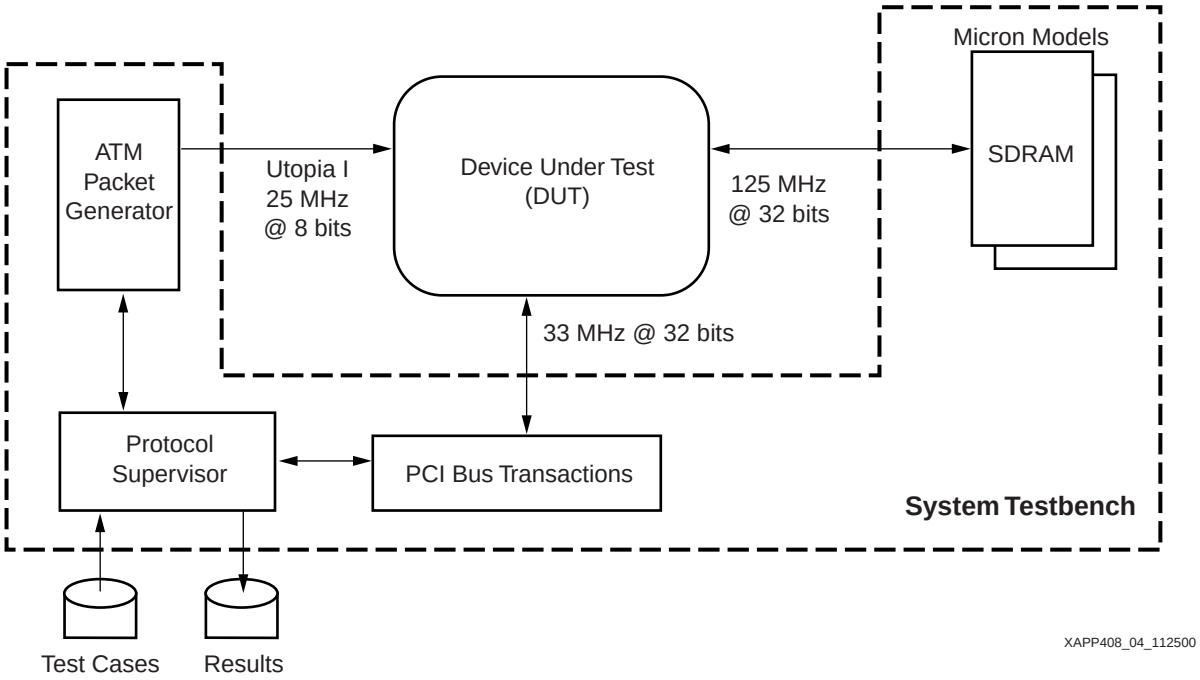

Figure 4: **Simulation Testbench**

Identify HDL Models Available

IP most often arrives with its own prepackaged, HDL-based testbench, allowing the user to get a feel for how it works. The IP-specific testbench can be used to develop the system-level testbench.

Each external functional interface—PCI bus, SDRAM and Utopia interface—will have to be verified. We will explore several approaches to verifying individual and system-level interaction of these interfaces.

The incorporation of already-designed intellectual property (IP) is extremely useful in speeding up the development of large systems. The IP may function as a module, but the user still needs

to integrate it into the targeted system and validate its interfaces. IP in general reduces the overall verification effort, but the designer must have a trustworthy source and be careful to ensure that the interfaces to the IP are thoroughly validated.

Each piece of IP should be simulated, synthesized, and P&R'ed as a module. From this effort the design engineer will gain further insight into how the IP works, and whether the module testbench can be used in the system-level validation. Modifications to the design can be easily verified against the IP-based testbench. The post-P&R evaluation answers the question of whether the IP small enough and fast enough for us to consider.

One problem with IP designs, specifically free ones, is that the design often is not delivered with an extensive testbench. This does not mean the IP is not usable, only that the user must spend more time validating the IP. The SPARC IP is an example of a design provided with a weak testbench.

The SPARC IP was delivered with a testbench that validated the processor's ability to fetch and operate on instructions. The testbench was by no means exhaustive, but the base functionality was provided. The PCI bus was delivered with a complete compliance-checking testbench. The testbench was extensive in that it covered all operational modes of the PCI bus.

Another IP which was considered for this project was the SDRAM reference design provided by Xilinx. The base testbench provided stimulus only, with the data never changing. The testbench was designed to show the base functionality, but assumed the designer was reviewing the resulting waveforms to understand the design.

Tools Available to Project

This project was written in VHDL and functionally simulated in the Model Technology ModelSim Simulator. For static timing analysis we used TRCE, the built-in timing analyzer for Xilinx FPGAs. The ILA ChipScope tool was available for this project, and we will show in a later section how we used this tool. The design flow was managed with a combination of makefiles and batch scripts.

Identifying Functional Interaction and Choosing What to Simulate

Major interfaces should be tested during simulation, but to what extent? Here we outline some key interfaces and our decision process that derived them.

How do you choose what functions to simulate now, and what portions to validate during in-system testing? Simulation is essential when preparing for in-system testing; without validated functional interfaces, in-system testing cannot proceed. During in-system test with a PCB, tests can be run at speed, but basic system interfaces must be working prior to power on. Trade-off decisions concerning how much time to spend in simulation, and when to progress to in-system test, are design dependent.

Our thought process starts with these two questions: What functions and interfaces do we need to have up and running before in-system test can start? What are the function and interface dependencies?

To debug our testcase PCB, we must be able to execute instructions on the SPARC. Before running code on the SPARC, we need to load instructions from the PCI bus into the SDRAM memory. Correct and timely operation of these functions are key to our ability to debug the

design at the PCB level. These interfaces must be simulated as modules and then as part of the system. An outline which lists test order dependencies is shown in Figure 5.

I. **Run SPARC code**

A. **Memory write/read from PCI bus**

1. SDRAM write/read to SDRAM memories

B. **PCI bus write to BAR1 burst.**

C. **PCI bus read from BAR1 burst**

1. PCI time-outs handled

D. **PCI bus write SPARC registers**

1. PCI bus write any register (BAR0) working.

Figure 5: **Essential Simulated Items**

Within the outline, the third-level indented items (numbers) are unit testbench requirements, whereas the top-level items (Roman numerals) are functions that must be validated to have an operational system.

A second part of the verification plan, described in Figure 6, lists the functions that can be checked at the PCB level.

II. **Process ATM Headers**

A. **Run code on SPARC**

1. Download code to SDRAM
2. Initiate Go command

B. **Generate ATM Traffic**

C. **Validate Correct number of headers processed**

1. SPARC Communicates with PCI bus.

Figure 6: **In-System Testing**

While developing the outline shown in Figure 6, we discovered an item not covered in the simulation outline. The SPARC needs to communicate with the PCI bus, but we didn't identify that interface to be simulated. When we arrive at the PCB level, we would not be able to determine if the system is running, because we would have no feedback regarding its operation.

To remedy this problem, we add the following test to Figure 5.

E. **PCI bus read SPARC registers**

1. PCI bus read any register (BAR0) working.

Figure 7: **Additional Simulated Items**

Thinking through the verification plan in terms of where to put the simulation effort, and when to transition to in-system test, enabled us to identify an important interface before we started the testbench development. Identifying key interfaces is required to achieve timely validation of this design, both in simulation and in system.

In-System Test Issues

No amount of simulation can make testing the physical PCB unnecessary. Each PCB design has it own timing variants and electrical environment.

Power on means different things to the ASIC and FPGA design efforts. The ASIC verification effort doesn't usually include in-system test of the device on the PCB, as such testing often occurs months after the ASIC design has been validated with simulation. For ASICs, there is a significant schedule lag from design to in-system test, and oftentimes the majority of the design team has already moved on to another project. Power-on integration for an ASIC, therefore, is typically started after the part is manufactured. For an FPGA, on the other hand, in-system test with the PCB is a critical part of the verification effort.

Several approaches can be applied to make this critical phase a success. An FPGA can (and will) be reprogrammed on the target PCB design. Reprogrammability is one feature of FPGAs that gives them an advantage over ASICs, allowing for a reduced simulation effort. A key way to use the target PCB is to think of it as a simulation accelerator. The effective simulation rate equates to about 10 Hz using a simulator, whereas a PCB system will run at the design's system frequency. It is possible to run many more effective clock cycles on the PCB then in simulation. This fact has led to the ad hoc simulation approaches used by FPGA designers in the past, but this does not work well with today's multimillion-gate designs. Simulation is necessary *before* the design is moved into the PCB phase.

Embedded processors provide the designer with increased functionality and product growth, but present unique problems when embedded into an FPGA. When designing with a chip-level embedded processor, the design team will often have an ICE-based debug environment which often requires access to the chip's pins or JTAG port. With a processor core embedded into an FPGA, the design team may not have the observability or the accessibility necessary for an ICE debugging tool, and new approaches will have to be explored.

Xilinx ChipScope ILA

In the past, FPGA engineers often pinned out critical internal signals so they could be viewed on a logic analyzer. This approach was workable until FPGA densities went over 100,000 gates. Now, with FPGAs exceeding 1 million gates, such specially pinned-out signals are simply not enough.

Xilinx has added a new tool, ChipScope ILA, to assist engineers working at the PCB level. It is a module that is compiled into the design, then hooked up to an HP Logic Analyzer. ChipScope gives you visibility into the design with up to 2048 signals. It has a very intelligent triggering mechanism, which will allow engineers the ability to fine-tune the signal and time of interest.

Summary

The key to developing the verification plan is to be complete and straightforward. The timeframe in which to develop the verification plan is after the specification is finished but before real design work has started. The verification plan will provide insight into the effort involved in simulating and validating the design. The verification plan often will have impact on the system specification. There may be cases where changing the system will make verification easier, and that will often result in a more timely delivery of the product.

Testbench Development

Creating a Valuable System-Level Testbench

Every design has performance goals. One of the criteria of a good system-level testbench is whether it assists the design team in assessing attainment of the design's performance goals. Not only should the system-level testbench exercise each interface; it should also operate the device as it was intended to be used. Performance goals are stated in terms of operating frequency, throughput, and latency. A good verification plan will allow the design team to explore these issues with a variety of tools. The system approach often presents a problem to

design teams who are accustomed to doing only module-level and rudimentary system validation.

Verification is the most active component of EDA today. Emerging technologies include formal verification and new testbench-specific languages like Vera and Specman "e". Many tools have been developed to aid the designer in module-level testbenches such as ILE Testbencher. We will not discuss emerging technologies in this paper, but instead focus on what engineers can do today with the tools they currently own.

This project was expected to leverage many pieces of IP. Each piece of IP had a testbench which may have some value at the system level. We evaluated each IP and its testbench in terms of its usability at the system level.

Testbench Evaluations

The SPARC testbench, as indicated previously, is rudimentary. Given that none of the SPARC process pins were going to be accessible in this FPGA, the testbench was not a good candidate as the basis for the system level testbench. Furthermore, processor code is always difficult to write when targeting a simulation environment. The code needs to be compact and quick-running. If an RTOS is being used in the real system, this code cannot be expected to be run during simulation, and an approximation needs to be developed. The problem is that whenever an approximation is made, resolution to the real system is lost. In this project, the code only needed to execute an infinite loop testing a register bit. When the bit changed, the SPARC jumped to the value provided by a register. With the code used for hardware validation so simple, we chose to rely on the IP-developed testbench for any modification we made to the SPARC.

A reference design provided by Xilinx for the SDRAM controller was reviewed by the team. This design provided a working SDRAM controller, but was unsuitable for our design. The Micron SDRAM memory models were used from this reference design for our system level testbench.

The PCI testbench was provided with the PCI core. Since the PCI bus testbench bolted onto the external interface of the FPGA design, it was used as the basis for the system-level testbench. The PCI testbench is used for compliance checking, but further review indicated that it could easily be extended to support functional testing. Because of the clean and extensible design, we based our system-level testbench on the PCI framework. The PCI interface controlled the operation of the FPGA and therefore was the logical choice as the central point of control.

The Utopia I testbench and interface is an original design. This interface had to follow the specification for Utopia I (available at **http://www.atmforum.com**).

A wrapper was developed which instantiated the PCI testbench, the Micron SDRAM memory models, the Utopia I testbench, and the FPGA. The Utopia I and the PCI testbench communicate with each other to synchronize events. The PCI bus reads the SDRAM to validate (self-check) that the expected data has arrived at its destination.

PCB In-System Testing Using ChipScope

The FPGA design depends upon Utopia headers being stripped off and placed into a FIFO. The FIFO signals the SDRAM controller to place the data into the SDRAM. The SPARC processor sets up a register with the starting point of the data location. Once the data has been moved, it is the responsibility of the SPARC processor to process all the data or move the pointer to a new location. During simulation, the system was never validated at the full data rate, as this would have taken many thousands of simulation cycles.

It was discovered that the design was not processing all the headers. Using the ChipScope package, we could set up trap points on the pointer register being updated, and review the SDRAM controller-processed data from the Utopia FIFO. With this information, it could be easily shown that the SPARC was not moving the pointer quickly enough to support the maximum data rate. Either the SPARC code needed modification or the hardware needed to be

changed to support automatic address incrementing. Without ChipScope, it would have taken much more time to discover a workable solution.

Reaching the Top of the Pyramid

A verification strategy combining simulation, static analysis, and in-system testing is key to success with high-density FPGAs. The engineer is bombarded with many different choices for verification of a design. To meet time-to-market pressures, designers need to leverage multiple approaches.

The productivity gained with a good verification strategy for FPGAs includes:

- Buy-in of team members that ensures they are working from the same set of requirements and understand the interaction at the various levels of abstraction
- Misinterpretations of the specification are caught early on in the verification planning phase
- Confidence that RTL and gate-level simulation is catching design flaws
- Working more efficiently at the RTL level by focusing on the "must-simulate" features of the design necessary for in-system test, instead on trying to toggle every register bit
- Validating interface timing and functionality with the post-P&R gates so that the designer only needs to simulate a subset of the full RTL test suite
- A reduction of the simulation scope at each detailed level of abstraction, meaning gates (lowest level of detail) are simulated less then RTL
- Applying the simulation environment to aid in the debug during in-system testing
- Designers are prepared for in-system test instead of reacting to it
- Using the reprogrammability of the FPGA instead of exploiting it—minimizing the thrashing: e.g., Change RTL → Synthesis → P&R → In-System Test

References

For more information, or for assistance in fulfilling your multimillion-gate FPGA design needs, contact Thomas Tessier, directly via e-mail: **tomt@hdl-design.com**. The full application note can be found on our website **http://www.hdl-design.com**, and on the Xilinx website at **http://www.xilinx.com**.

See Xilinx Design Guide to FPGAs at:

http://support.xilinx.com/support/sw_manuals/3_li/download/gensim.zip

Xcell Articles:

Xilinx: *Using IBIS Specifications* (Xcell Journal 27 article, Q1 98) available at: **http://www.xilinx.com/xcell/xl27/xl27_10b.pdf**

Xilinx: *FPGA-on-Board Timing Verification Using Tau* (Xcell Issue 33 Quarterly Journal, Q3 99) available at: **http://www.xilinx.com/xcell/xl33/xl33_54.pdf**

Xilinx: *On-Chip, Real-Time Logic Analysis with ChipScope ILA* (Xcell Journal 36, Q2 00) available at: **http://www.xilinx.com/xcell/xl36/xl36_19.pdf**

Xilinx: Special section on verification (Xcell Journal 29, Q3 98) available at: **http://www.xilinx.com/xcell/xcell29.htm#verify**

Books:

Janick Bergeron, *Writing Testbenches: Functional Verification of HDL Models*. Kluwer Academic Publishers

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/15/00	1.0	Initial Xilinx release.
11/22/00	1.1	Updated and edited to current format standards
02/15/02	1.2	Updated to include disclaimer.