Re-Thinking Your Verification Strategies for Multimillion Gate FPGAs

XAPP408 (v1.02) October 16, 2000

Author: Thomas D. Tessier, T2 Design

Summary

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 re-think their verification strategies. The most effective way to alter your 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 spend minimal time in simulation and quickly move to the lab where they can debug and modify until release. Million-gate FPGAs implement functions far too complex to make it feasible to continue to rely on this ad-hoc method.

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

Throughout the paper references to ASIC are defined as a traditional gate-array or standard cell ASIC. FPGA indicates a Field Programmable Gate Array like the Xilinx Virtex™ Family. Although the terms, ASIC and FPGA can be used interchangeably, for the context of this paper they are not.

What is Verification?

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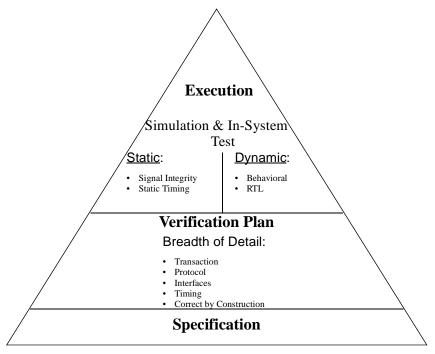
Figure 1: **The Verification Pyramid**

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. A detailed and complete specification in written format is essential to meet 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 verification approaches for ASICs will realize that design size and power consumption are not issues will 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 less manufacturing specific details to worry about.

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 (or detail) is outlined at: transactions, protocol, interfaces and timing levels. Essential functions are identified. The conditions of when key features occur and their expected responses is described. A determination of the number of test benches needed, their complexity, and test module dependencies is made. Test benches 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 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 your verification effort the following list offers examples of the type of information you need to *identify*:

- External interfaces
  - Stimulus and Response
  - Transaction level, i.e., Read vs. Write operations
  - Timing Requirements
- HDL models available to assist in Testbench Development
  - Packaged with proposed Intellectual Property (IP)
  - Bus Functional Models
  - Vendor Supplied, i.e., 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, i.e., need 32 block data write @ 66 MHz with a latency of less than 300 as.
- Simulation vs. In-System trade-off

Execution

A verification strategy that best suits your design means breaking out those functions that are essential to simulate and 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-includes non-synthesizeable constructs which are used as executable specifications and also for testbenches.
- RTL-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, easier to understand, and 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 the RTL designs. 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 trade-offs 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 be down to the bits in registers and combinational 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 test bench 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. 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.

Static Analysis comprises: Static Timing Analysis, Formal Verification and Signal Integrity Analysis. All of these techniques use mathematical approaches (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 and 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, timing driven P&R (if used) and 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 are driven by the tools available. Timing constraints can be specified independent 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.

How do you decide what to simulate and what to validate at In-System test? ASIC design teams use the rule of "If it isn't tested it's broken" to guide the simulation approach 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 testcase later which highlights a way to look at the verification problem with In-System Test as a valuable component. Manufacturing (ATPG) or JTAG vectors are not an issue for FPGAs because the silicon is already verified by the vendor.

In-System Test

We are expecting to utilize the re-programmability 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 enabling the user to 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 executable specification (of the design) and testbench, both written in behavioral HDL, meet the 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 formal verification if it is available. This insures the tools have correctly implemented the design. In addition generated gates are run thorough static timing analysis. This step verifies that the system level timing is met. (Figure 2)
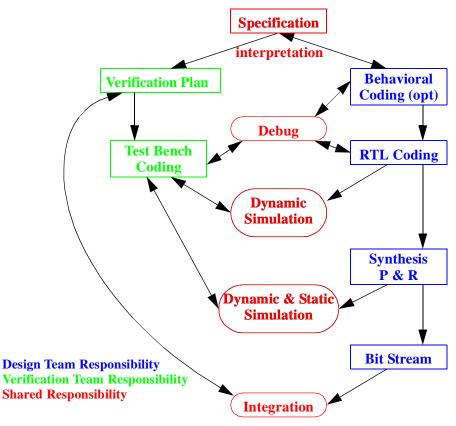

Figure 2: **Interaction of the Verification Components**

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, successfully placed and routed and can create a bit stream 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 gates levels with minimum functionality necessary for power on. The simulation of the chip is often not achieved by FPGA design teams still using simulator specific approaches. 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 pre-load the type of data which will be accessed and 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 process and an 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.
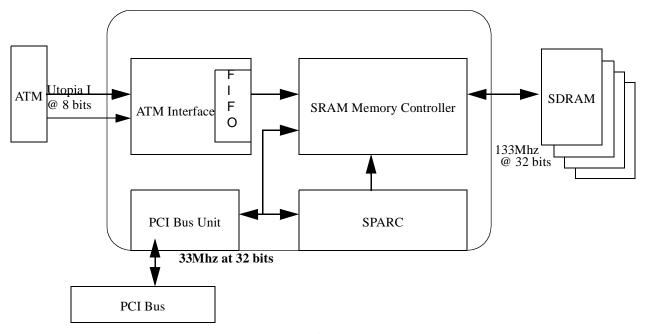

Figure 3: **Block Diagram of Testcase**

Specification

A system level specification (see references) was developed for the testcase project. The specification was reviewed by the architects, design team and verification members. Our philosophy is to always have a different engineer write the verification testbench than the RTL designer. 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 we "think out loud" for our decision process that splits 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 trade-offs 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 as shown in Figure 4. A PCI bus running a 33 MHz, a 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 that the development of the timing constraints is straight forward. The transactions are identified as: burst read/write, single read/write and retry/failure modes on the PCI bus and handshaking signals which control the data flow on the synchronous Utopia interface. The SDRAM interface is synchronous and the command is already built as a transaction.
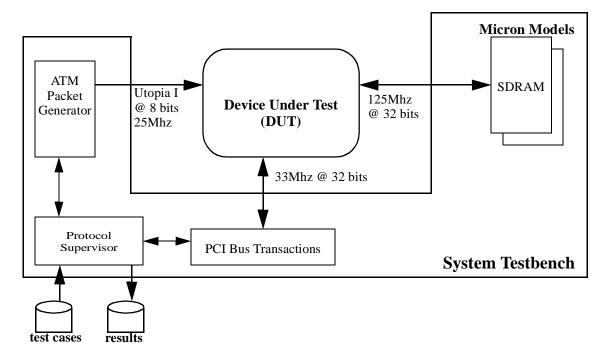

Figure 4: **Simulation Testbench**

Identify HDL Models Available

IP most often arrives with its own prepackage 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 I 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 as a module. From this effort the design engineer will gain further insight into how the IP works and if 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 evaluation after P&R provides the answers to is the IP small 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, just that the user must spend more time validating the IP. The SPARC is an example of a design provided with a weak testbench.

The SPARC IP was delivered with a testbench which validated that the processor could 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 utilized 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 in preparing for in-system testing. Without validated functional interfaces, in-system test 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 the 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 is key to the success of the project and the 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 listing 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 it can be noted that the items that are at the third level (numbers) indented are unit test bench 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 discover 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 not have feedback of its operation.

To remedy this problem we add the following test to Figure 7.

> > **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 be successful in timely validating this design in both simulation and in-system test.

In-System Test Issues

No amount of simulation can make up for using the physical PCB. Each PCB design has it own timing variants and electrical environment. Power on means different things to the ASIC and FPGA design efforts. The verification effort for an ASIC doesn't usually include in-system test of the part on the PCB; as this 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; often the majority of the design team has moved onto another project. Power on integration for an ASIC is started after the part is manufactured.   For an FPGA, in-system test with the PCB is a critical part of the verification effort.

This effort is part of the verification task and several approaches can be applied to make this critical phase a success. An FPGA can (and will) be reprogrammed on the target PCB design. The re-programmability is one aspect that gives FPGAs an advantage over ASICs and allows 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 about 10Hz 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 it does not work with today's Multi-Million gate designs. Simulation is necessary before the design is moved into the PCB phase.

Embedded processors provide the designer increased functionality and product growth but present unique problems when imbedded into an FPGA. When designing with a chip level

embedded processor the design team will often have an ICE based debug environment. The debugging environment often requires access to the chips 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. New approaches such as ChipScope ILE will have to be explored. Xilinx has added a new tool to assist engineers working at the PCB level. In the past FPGA engineers have often pinned out critical internal signals to view on a logic analyzer. This approach has been workable until FPGA densities went over 100K gates. Now with FPGAs exceeding 1Mgates the pinned out signals is not enough. Xilinx has developed ChipScope ILA for this purpose. It is a module which you compile into your design then hook 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.

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 that changing the system will make the verification easier which will 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 if it assists the design team in assessing if the performance goals are met. Not only should the system level testbench exercise each interface but it should operate the device as it was intended to be used. Performance goals are stated in the 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 accustom 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 like ILE Testbencher. We will not discuss emerging technologies in this paper but instead focus on what engineers can do today with their tools they currently own.

This project 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 to it usability in the system level.

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 bases 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 run quickly. If a RTOS is being used in the real system this code cannot be expected to be run during simulation. An approximation needs to be developed. The problem is whenever you make an approximation you loose resolution to the real system. 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 on 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 extendable 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 communicated with each other to synchronize events. The PCI bus read the SDRAM to validate (self-check) that the expected data had 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 with the full data rate as this would take many thousands of simulation cycles. It was discovered that the design was not processing all the headers. Using the ChipScope package we could setup trap points on the pointer register being updated and review the SDRAM controller processed data from the Utopia I FIFO. With this information it could be easy shown that the SPARC was not moving the pointer fast enough to support the maximum data rate. Either the SPARC code needed modification or the hardware was 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 and ensures they are working from the same set of requirements and understanding of 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 focussing on the "must simulate" features of the design which are 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.
- Utilizing the re-programmability of the FPGA instead of exploiting it - minimizing the thrashing; i.e. Change RTL -> Synthesis -> P&R -> In-System Test.

References

For more information or to assist you in your Multimillion Gate FPGA design needs contact Thomas Tessier, directly via email: tomt@hdl-design.com. The full application note can be found at our web site: **http://www.hdl-design.com** and at Xilinx web site **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 (Q399) 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**

Special Section on Verification (Xcell 29 - Third Quarter, 1998) 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.