

# What You Should Expect the System Simulation Engineer You're Going to Hire to Do

Matt Hsu  
SynOptics Communications, Inc.  
4401 Great America Pkwy  
Santa Clara, CA 95052

## Abstract:

Surprisingly, one of the things that a system simulation engineer is least likely to be doing is simulating your system. This is partly because modern system simulation environments must constantly be evolved to include the fastest, latest tools in their attempt to verify as much of complex systems as possible prior to fabrication. This is especially true of systems containing ASICs. Frequently, environments must also be customized for each system by the addition of special data generation and checking algorithms. System simulation methodologies that use VHDL as a backbone can leverage its standard nature when they are evolved to include new tools and features. The extensive use of VHDL also simplifies the verification of the many levels of a system including mother boards, daughter cards, boot code, and interfaces, as well as ASICs, FPGAs, and their interfaces to each other and the rest of the system. This paper describes some of the infrastructure the system simulation engineer is responsible for building, maintaining, and evolving, some of the techniques we used, and some of the places we expect to go in the future.

## 1.0 Introduction

Throughout this paper pieces of system simulation infrastructure and verification techniques will be illustrated showing how and where VHDL can be leveraged. It is the responsibility of the system simulation engineer to architect, maintain, and evolve this infrastructure by defining

methodology, guiding VHDL coding style, specifying tools, and implementing design and verification techniques discovered along the way. Since our group was newly formed we had the luxury of picking many of the tools that we wanted to use. We also had a charter to develop and integrate new design paradigms for the company.

The position most system simulation engineers will be in however, is with fixed ASIC and board design flows and, more importantly, the fixed mindsets of designers and managers. The system simulation engineer must not only be persuasive enough to get buy-in from management to purchase "expensive new tools" and get the respect of designers to get them to use them, but must also champion the methodology in its infancy so that the growing pains and inevitable learning curves don't cause the process to lose momentum. To this end it is of prime importance that system simulation infrastructure is built on the simulation environment already in place. Not only to overcome the inevitable resistance, but because there are usually many man years of expertise and design database that could be used to minimize the impact of the new methodologies.

## 2.0 System Simulation Goals

The obvious goals of system simulation are to verify the functions, features, and interoperability of ASICs and FPGAs, the functionality and timing of boards, and to

some extent, the manufacturability of everything. Emphasis is placed on ASICs as they tend to be the most risk because of purchase commitment and long in circuit debug and turnaround times. The unspoken goals, though not usually achievable, are *no second spins on ASICs* and *no board turns*.

There are some not so obvious goals as well. The environment should respond quickly to design changes. The environment should be easy to understand and use requiring as little maintenance as possible. Generation of tests and reduction of test results should be automated wherever possible and simplified where automation is too cumbersome. The environment should be easy to evolve without requiring complete abandonment or redesign of databases, tools, or methodologies. As many people as possible should be able to become proficient at using the environment in a relatively short time period.

Although these goals do not specifically imply VHDL based design, VHDL is constant - at least compared to the tools and associated methodologies. This implies that VHDL databases will generally be reusable. Further, the current use of synthesizable RTL VHDL for ASICs and FPGAs meshes very well with system simulation techniques because you inherently get relatively fast, correct-by-construction simulation models.

### **3.0 Minimum Things Your Environment Should Include**

Everyone recognizes the need for the high power tools that actually do the work, such as simulators and synthesizers. What is usually left behind, are the support tools that prevent thrashing during the inevitable bug fix iterations. Three support tools are essential to successful design: revision control, bug tracking, and batch queuing.

Aside from a system simulation engineer, having an experienced person dedicated to writing tools and scripts is just as invaluable. The tools support person off

loads the system simulation engineer and the designers from dealing with the repetitive nature of design by writing automation scripts and dealing with all the variances and varieties of maintaining the different tools such as managing licensing and networking, disk space, and memory and workstation configurations.

### **3.1 Revision Control**

VHDL is very much like software in that there are frequently many units of code that will go through varying degrees of revision. Software designers learned long ago what a nightmare building and linking uncontrolled code can cause. This is true not only for VHDL design code, but also for testbenches, tests (perhaps in the form of stimulus and response files), support code, simulation and synthesis scripts, initialization files, and even schematics. All these types of code should be controlled in a clean and orderly manner. Software revision control tools and techniques are directly applicable to the VHDL system simulation environment.

We use "rcs" instead of "scs" because it saves the current revision on top and previous revisions as changes to it providing some protection against corruption of the change stream. Another safety technique we use that we delete all old source files prior to a regression run and check out fresh source. This prevents residual files from either masking new errors or regenerating old ones. It also guarantees that the latest designs continue to play together and constantly exercises their controlled databases. One of the roles of the system simulation engineer is to enforce and protect the "golden" controlled code. Designers should be discouraged from keeping their own private code. Code ready for "checking in" should have a minimal set of tests run against it.

### **3.2 Bug Tracking**

Like every other support tool, the bug tracking system must be easy to use or it

won't be used. We use ours for tracking VHDL simulation bugs, VHDL synthesis bugs, internal and external tool bugs, bugs found in simulation, bugs found in the lab, documentation bugs - essentially every bug that comes up. In addition, we also use our bug tracking system for revision enhancements and questions on system and design functions. One of the real reasons the bug tracking system works, however, is that it is used to gate tapeout of ASICs or release of boards.

Any designer who is assigned an "open" bug can mark it as "fixed" but cannot "close" it. Only the person who opened a bug can close it. In this way bug loops get closed, relevant bugs have visibility to management, circumstances don't get lost from designers memories, and they can be reviewed during product "post mortems" to improve the design process.

### 3.3 Regression Testing

Frequently, designers only have the time to test logic around a particular area where they have made changes. The most important function of system simulation is to address this by applying the hundreds of feature and function tests which guarantee that the design changes didn't break other parts of the design, interfaces, or other designs. For this reason it is important that testbenches and tests be written with regression testing in mind. It is also important to gate the release for fabrication by the regression test suite for that design.

Typically a designer tests a design function by using a simulator's port or signal driving commands to stimulate a design and verify correct function by visually analyzing the response waveforms. This information is captured in system simulation tests by building data generation and checking structures into the environment and using VHDL textio to read it in.

Stimulus should not be of the "test vector per clock" format but should be more general to an interface. For example, in one

of our test stimulus files one line of code represents a cpu cycle (location, read or write operation, data) and the testbench is responsible for generating the necessary chip select, address and data strobes and wait for the device acknowledge. Another line contains the information necessary to generate and check a data packet (arrival time, length, source and destination) and the testbench creates random data to fill the packet, generates a CRC and appends it, and checks the packet when it comes out of the system.

System regression tests should be as self contained as possible and able to be run without undue setup. They should not take advantage of specific simulator features as changes to these features or multi-simulator environments may invalidate an entire test suite. The simulation environment should not require or produce excessive amounts of information and simple scripts should be developed to indicate whether an entire test suite has run properly and when it hasn't, status of the tests that failed.

Finally, built into the test environment and *especially* into the ASICs themselves are VHDL assertion warnings that produce messages if state machines transition into illegal states or signals to erroneous values. Synthesis of this code is disabled. Their function is to wait for an unpredicted condition in the infinite state space of a system to trigger them *any time* the design is used.

### 3.4 Batch Queuing

There are two concepts for running the regression test suites. They are the "super-machine" and "distributed processing." For the super-machine, the fastest workstation available is loaded with gate-level accelerators, hardware modelers, VHDL accelerators, lots of RAM, and sufficient swap. Tests are run in series until completed. Using distributed processing, unused or idle workstations are used and tests are run in parallel on whichever ones are available.

There are advantages and disadvantages to both. Accelerators and modelers are expensive and have size limitations. Distributed processing absorbs tool licenses. We use distributed processing to run our regressions. We have a batch queue that looks for machines, determines their configurations, and queues tests on those machines. The queue also checks responses and generates log and error files.

#### 4.0 Testing

Tests for system simulation are not that different from those for ASICs. We apply the following concepts:

- 1) Tests should check themselves giving a pass/fail indication.
- 2) Every feature and function should be testable and tested.
- 3) Search through the ASIC, FPGA, and system specifications and list all the features and functions.
- 4) Have test development brain-storming sessions with designers, test and manufacturing engineers, etc.
- 5) Write a lot of tests that check a function separating significant variations into different tests.
- 6) Run known sequences before, between, and after the function being tested.
- 7) Within the same test, test the same function more than once.
- 8) Where possible, tests should not be "tuned" (stimulus arriving at precise clock cycles) to prevent "re-tuning" for design iterations.
- 9) Suspect everything. Digging into suspicious areas and anomalies usually turns up something that requires attention.
- 10) Verification people need only discover who owns the bug; it is the designer's responsibility to find and fix it.
- 11) Don't trust designer's guarantees that the bugs are fixed and the design can be sent for fabrication tomorrow until all the regression tests for that design are run.

#### 5.0 What is a System Simulation?

Although difficult to define, system simulation has some generally accepted characteristics. Most people would agree that it is verification at a level of complexity higher than a single ASIC. For practical reasons we consider it to be near the level of complexity of a "product." For example, a product that we wish to verify through simulation might contain a motherboard, multiple flavors of ASICs and FPGAs, a CPU and associated memory, glue logic, a backplane, and several daughter cards.

#### 5.1 The Hand Stitched System Simulation Model

The hand stitched system model is a netlist that contains a minimum number of components, generally limited to ASICs, FPGAs, and data generating and checking code. Testbenches are used to apply stimulus, check data, and generate response. Stimulus and response data are generally at a high level of abstraction. This method is useful when a schematic is unavailable.

We used a hand stitched simulation to verify the ASICs that make up the core fabric for a family of products. By nature our products have functions that are repeated on a port-by-port basis. Using generics as generate block parameters, we controlled the size of the simulation instantiating only the minimum number of ASICs required to test a particular function. Note that by instantiating one ASIC, one can leverage the system simulation platform to verify individual ASIC behavior as well. By instantiating dozens of ASICs, bugs were uncovered by the sheer complexity of the simulation.

#### 4.0 The Converted Schematic System Model

Very few designers would capture schematics for boards using a text editor when good schematic capture tools are

available. Further, the fabrication of boards using the output of schematic capture tools is a mature and streamlined process. Since the schematic is the ultimate source for the connection of all components of a product, it is the ideal vehicle for "correct-by-construction" system simulation. We convert this database to a VHDL netlist and hang all the models necessary for simulation to it. Since our ASIC and FPGA designs were captured in VHDL they fit right in.

One of the deficiencies in system simulation today is the availability of simulation models for the latest off-the-shelf components. Some SSI/MSI/LSI models are available commercially but required the creation of "wrappers" to map our schematic symbol ports to theirs. For other models we wrote full or reduced function VHDL. Many models can be left with empty or pass-through architectures. Note that at this point, the simulation verifies functionality and connectivity only, timing is not checked. Like the hand-stitched system, the simulation size can be controlled but this time by using configurations to push down through hierarchy and bind in reduced function, pass-through, or empty architectures for large components. Using VHDL libraries, these models are compiled in a centralized location.

One of the advantages to using VHDL is that you can implant data generation and checking processes. These processes can contain assertion warnings or text to input stimulus or output response and hang like logic analyzer probes waiting to be triggered. We use "sed" scripts to surgically implant this code. In this area the full power of VHDL is applicable.

Regression tests for the converted schematic model are controlled by the CPU and the data generation and checking code. In our case the CPU is a bus functional model with standard subroutines for initialization, data flow control, and pass/fail indications.

#### 4.1 Random Data Testing

For complex systems the "state space" that they represent is so large that feature and function tests do not guarantee first pass silicon success. We have developed parameterized data generation code that allows us to run infinite simulations using randomly generated data arriving at random times. Using VHDL this model contains linked lists that keep track of data throughout the system. These lists grow and contract using VHDL access types for dynamic memory allocation and deallocation. At any time *while the simulation is running* an indicator a file is used to instruct the simulation to output it's status. Using this technique specific data was also injected into the simulation taking advantage of the random state space that had been generated. All the time there are assertion warnings looking for problems. Note that this is performed purely with VHDL constructs so that the simulator interface is not used.

#### 5.0 What About ASIC Methodology?

As expected, our VHDL based ASIC methodology integrates efficiently with the VHDL based system simulation. We design ASICs and FPGAs using synthesizable RTL from the start. The system simulation model acts as the testbench for these designs with which designers generate regression tests as they are checking them. The regression tests then become a gating item for ASIC signoff. We also have tools that generate VHDL wrappers for our ASICs which extract and output test vectors that are easily converted into manufacturing functional test vectors. We are waiting for reliable VITAL gate level models from foundries so that we can use gate level netlists in the simulation model if necessary. After fabrication, the ASIC and FPGA models can be compiled into centralized libraries.

#### 6.0 Evolutionary Steps in the Future

Using the converted schematic we expect to

add several features. Foremost is verification of timing behavior. Wrappers for the VHDL designs that map schematic symbols to VHDL ports would contain timing checking and delay generation. This would decouple the system verification process from the synthesis flow. Timing prior to synthesis would come from requirement specifications while real numbers could be used after fabrication.

Other plans include getting a jump on diagnostic and boot code development by using the system simulation to run real code. It would also be easy to add processes in the converted schematic to check that every net gets exercised producing a fault grade for the board.

## **7.0 Conclusion**

System simulation has proven to be invaluable in ferreting out bugs in our systems. For one product we have over 400 tests that run for over 200 us of simulation time which uncovered over 100 bugs in the ASICs, FPGAs, and motherboard. Even so we found bugs in the lab every one of which we were able to duplicate in simulation.

To meet the demands of verification of complex systems, the system simulation environment will be stressed to its limits. The system simulation engineer you hire must be well rounded in all aspects of the design process in order to understand the ways to evolve the environment to verify as much of your systems as possible. Since many people will be relying on the methodologies developed they should be streamlined, clean, and orderly. The system simulation engineer must take the high ground and be firm enough to resist shortcuts and hacking but allow them where the better solution can later be put in place for the next iteration of the process. Remember, unlike your designers, your system simulation engineer's main job is finding bugs. The more bugs found the more valuable you should consider the position.