

# PHY Verification – What’s Missing?

Thomas J. Sheffler, Kathryn M. Mossawir, Kevin D. Jones

Rambus Inc.  
Los Altos, CA

[sheffler@rambus.com](mailto:sheffler@rambus.com), [kmossawir@rambus.com](mailto:kmossawir@rambus.com), [kdj@rambus.com](mailto:kdj@rambus.com)

**Abstract**— PHYs are devices composed of digital and analog circuits that transmit data over a physical medium. These devices present unique verification challenges. While a PHY contains significant amounts of digital logic, the representation of analog blocks used for verification can make real bugs invisible in simulation. Special modes whose sole purpose is to allow manufacturing test and characterization of analog components add complexity to a device whose normal-mode operation is straightforward. In this paper we share some mixed-signal design patterns that present verification challenges and illustrate limitations of standard approaches. Ultimately, however, we believe new tools, representations or methodologies could help, and we suggest some approaches EDA vendors might investigate.

## I. INTRODUCTION

A PHY is a device that combines digital and analog circuits to communicate data over a physical medium. A typical PHY presents an ASIC (Application Specific IC) side supporting a synchronous interface to on-chip logic, and a "channel" side that may include the pads of the chip (see Figure 1). The job of the PHY provider is to implement a robust digital interface for the ASIC designer and to contain many of the complexities of off-chip communication.

This paper focuses on the verification challenges facing the PHY provider. A PHY contains both digital and analog

circuits, both of which must operate correctly. Our job as logic verifiers begins with the assumption that circuits are correct and implement their specifications. From this starting point, we need to ensure that there are no errors in PHY logic.

To illustrate our points, we present a series of difficult design and verification patterns that occur in PHYs. For each such pattern, we describe the types of logic errors it may introduce and why they are difficult to detect. Where appropriate we describe our current verification approach and identify what is lacking in current tools or methodologies. We hope that these pieces of information can be helpful to tool builders in the development of new systems.

### A. Typical Verification Approach

A typical design and verification flow for a mixed-signal device like a PHY clearly separates the design of key analog blocks from module-level logic design and top-level assembly of the circuits and logic. The analog tasks include detailed block-level circuit design, specification and simulation using a SPICE variant. These labor and time intensive tasks result in block-level schematics of sized transistor-level circuits.

Logic blocks are designed at the gate or RTL level. While the end-to-end data transport behavior of the PHY is specified, there often are not block-level specifications for every logic block. As a result, verification of the PHY as a whole is relied upon to catch both block interconnection errors and intra-block logic errors.

The extracted netlist representing the PHY consists of a network of logic and circuit blocks, each of which has one of two representations. Logic blocks are expressed as RTL or gates. Circuit blocks are normally abstracted as behavioral Verilog modules, but if an AMS (Analog/Mixed-Signal) simulator is used, circuit blocks may be modeled as a SPICE-level netlist.

To develop a verification testbench that supports the simulation of any of these four variants, we construct the "PHYtoPHY" configuration shown in Figure 2. This verification environment uses Specman or System Verilog for the testbench and communicates to the ASIC interfaces

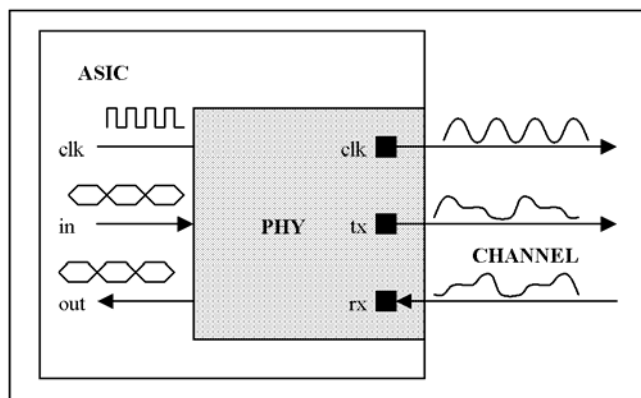


Figure 1. A PHY bridges the On-Chip Logical and the Off-Chip Physical

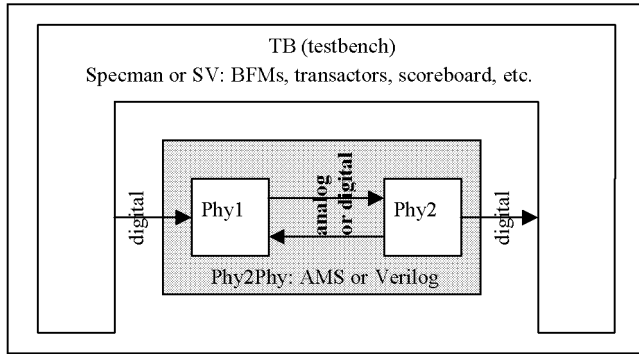


Figure 2. PHY-to-PHY Verification Environment

of the PHYs using two- or four-valued logic. By constructing the PHYtoPHY configuration, the testbench only needs to attach to the logic interfaces of the PHYs, regardless of the DUT (device under test) simulation representation.

This approach works well for exercising the data transport behavior of the PHY and for exercising the higher-level functionality of advanced protocol stacks. Much of the current verification technology [1,2,3] addresses how to build testbenches like these and how to effectively exercise hidden corner cases in these protocols. Assertion technology also helps by giving visibility to errors where they occur rather than relying on data mismatches after data transport [4]. We fully advocate the use of these approaches and techniques.

What one finds is that this approach does not help find logic errors related to subtle effects of PHY behavior. These effects include control of signal conditioning circuits like output drivers and programmable termination impedance, analog test structures and programmable clocking circuits, to name a few such areas. Section III.A expands on this particular problem and shows how the logic controlling these subtle electrical characteristics can harbor errors that are difficult to find.

This paper reviews some of the difficult PHY design patterns we presented in [5] and considers how current tools and methodologies do not directly offer approaches for catching typical logic errors in a systematic manner. To date, we are somewhat successful at finding some classes of these errors, but only because we know what to look for in our designs. We would prefer a unified methodology that detects errors in all of the patterns presented here and is extendible to other patterns as well.

## II. CHARACTERISTICS OF HIGH-SPEED MIXED-SIGNAL PHY DESIGN

In the simulation literature, the term "mixed-signal" has become an umbrella term for many types of systems [6,7,8]. Verilog-AMS introduces the concept of natures and disciplines allowing the description of not only electrical circuits, but motor control systems and optics. Our class of

mixed-signal problems is somewhat specific. Here we identify some of the characteristics of a high-speed mixed-signal PHY design.

- Core logic running at speed reasonable for implementation technology.
- Complexity of logic relatively low: hundreds of registers, tens of state machines, possibly one simple processor.
- "Thin" logic layer over circuitry. Many aspects of circuitry control are exposed to the client in the form of protocol constraints.
- IO pipeline of custom digital logic running at 2x, 4x or 8x over core logic speed. It is tightly-controlled synchronous logic of simple function. Custom layout.
- IO Drivers with complex link conditioning circuitry. Circuit characteristics tightly coupled to package design, signal integrity and power delivery.
- Aggressive design of on-PHY clocking employing DLLs (delay locked loops) and PLLs (phase locked loops) under digital control. Clocking may be adaptable. Avoidance of latency-inducing synchronizer structures in main data path.
- Primary functional analog components: DLL, PLL, CDR (clock/data recovery), clock mixer, phase detector, precision voltage and current references, programmable drivers, programmable on-die-termination, programmable equalization and decision-feedback-equalizer samplers. Complexity of each analog block on the order of 20 or fewer transistors. Presently, no RF (radio frequency) mixers.
- Complicated initialization procedure involving the fine-tuning of circuit parameters to match operating environment. Normal mode operation is mostly serialization of data. Periodic update of circuit parameters may be performed to optimize performance.
- Simple data path. Minimal or no data coding.
- Structures supporting lab characterization.
- Structures for Analog DFT (design for test).
- Multiple clock domains.
- Possible sensing of electrical or physical link conditions.

## III. PHY VERIFICATION CHALLENGES BY DESIGN PATTERN

In this section, we present difficult design patterns encountered in PHY design and describe why each of them introduces hard to find logic errors. For each, we describe some possible techniques that can be used to avoid these errors and identify what is missing in current tools.

### A. Control of Signal Conditioning Circuits

A PHY modulates digital data into electrical signals for transmission and demodulates electrical signals into digital data on the receiving end. To enable the use of the PHY in many different electrical environments, many of the

characteristics of the "analog" transmitters and receivers are under digital control. Characteristics of a transmitter, such as signal swing, driver current or slew rate are digitally programmable. Receivers may offer programmable on-die termination, or may employ comparators with digitally adjustable offsets for compensation. (Compensation is a technique that helps tune a circuit to its near-center "ideal" behavior in the face of fabrication variances.)

The logic controlling these electrical characteristics directly shapes the magnitude, slew rate or timing of the data traveling over the channel, but does not directly affect the binary value of the data on the channel. For this reason, the straightforward PHYtoPHY verification testbench described earlier may not detect errors in this type of control logic. A more directed approach is required. We discuss two options below.

### 1) Using Behavioral Models

One approach is to write behavioral Verilog models for each analog block. Figure 3 illustrates an inverting output driver with a programmable strength. The figure poses the question: "What is an appropriate logic representation for such a circuit?"

One may partition the inputs of these models into one of two classes: "data path" and "control path." Data path inputs are those that transmit data to the channel. Errors on these inputs can be detected in the normal data path checking of the PHYtoPHY testbench.

Control path inputs are those that adjust electrical characteristics, and these need to be handled specially. A common type of error that occurs on these ports is that the logic driving them has an encoding error of some type. Assertions do not necessarily reveal these because the individual values are not illegal, but they are not what is intended.

To check that the values delivered into the control ports are what is intended requires the testbench being able to observe the control port values, and the test-writer knowing the expect values at these ports. Both requirements present difficulties. The first is the problem of connecting the testbench to the observation point. The second problem is that one must write a predictor for the value at the observation point. The first point has been more problematic for us than the second.

In order for the testbench to be able to check the control ports of analog blocks, we require a way to create a connection from the testbench to the port. Our environment combines a Specman testbench with a Verilog netlist derived from a schematic entry environment. Using absolute path names to reference internal nodes is possible in this environment, but becomes unmaintainable in practice.

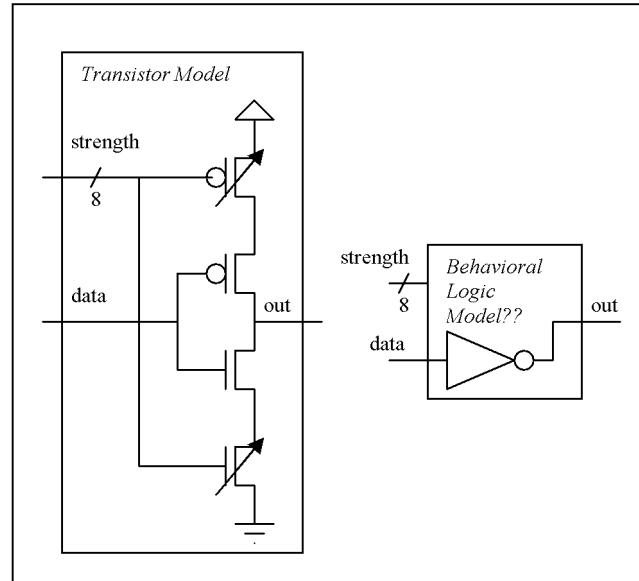


Figure 3. A example circuit model and a dangerous behavioral substitution.

The Verilog path names to the control ports change as the design evolves, and worse, are not even uniform in our designs. That is, the path names for ports in two similar blocks may have slightly different depths. What we do usually know is that all instances of the same analog block have the same type of behavior at their control ports, although their values are independent. We would like to be able to write one piece of code that connects the testbench to all of these instances and have the run-time system of the simulator provide an instance of each checker as needed. We would also like the run-time to provide a means to relate each instance to a unique index or name that remains constant as the design undergoes small permutations. Using this index or name, we could write code that checks the value of each instance in a pathname-independent way using information from the testbench.

### 2) Using Verilog AMS

Using a SPICE-level representation of the "analog" blocks is another approach. With an AMS simulator, we should be able to observe the electrical values on the channel directly. While this approach sounds appealing, it is not very useful.

As a logic verifier, we want to show that for all possible digital values, the logic does the right thing. Thus, we would want to see that for all control values, the link characteristics are what is intended. These characteristics include signal swing, slew rate, timing and other parameters that are a combination of input data and control port programming. What does not exist yet are well-developed techniques for computing analog "expect" values of these types of characteristics and the means to check them, although some tools have recently been offered [9]. The field of mixed-signal verification has yet to invent standardized ways to

perform "analog response checking" in these types of values. Analog response checking will also differ from digital response checking in that values will be approximate, or within a suitable window rather than an absolutely right or wrong digital value. We do not yet have a good idea of what is needed, but believe this will be a fertile area of research in the future.

### B. Environment Detection

A PHY may be responsible for reporting characteristics of the transmission medium not directly related to data transmission. The PHY may be required to sense a quiet link, an unconnected link or a particular thermal condition. Detection of these characteristics requires special circuits that report a physical condition as a logical value. As a verification engineer, we must ensure that the consuming logic is correct for all values produced by the sensor.

To do this, the output value of the sensor should be under control of the testbench. In practice, creating connections from the testbench to each such source is difficult because the "pseudo-wire" that provides a path from the output of the sensor to the testbench is not a real wire in the netlist, and the use of absolute pathnames for this type of "stimulus injection" suffers from the same difficulties the observation points of the previous section do.

What we would prefer is a more natural means to model the conditions that the sensor responds to. In the case of the temperature sensor, we would prefer the ability to model the ambient temperature as a quantity that can be established by the testbench and reported in the coverage report. We would prefer that temperature were modeled using natural units to ease understanding and to clarify communication between engineers.

The modeling of the link-presence detection circuit could benefit from some enhanced behavioral modeling abilities. At the behavioral level, a link (wire or differential pair) has a logical data stream and may be connected properly or disconnected. Long before the circuit exists that detects link-presence and decodes the logical data stream, we would like to have the ability to model these two concerns so that logic design and verification can proceed. We would like to have a seamless way for the testbench to send a data stream, and a way for it to inject the "link disconnected" situation as a condition different from a stream of all zeros. We would like the module interface of this abstract behavioral model to be the same as the eventual circuit implementation. Such a seamless approach to behavioral modeling is not currently possible.

### C. Analog Test Structures

In contrast to digital designs that employ a structured DFT flow, a PHY may include special structures and modes to test non-digital circuits like IO drivers and PLLs. IO test modes add small amounts of logic to determine if drivers and receivers are functional. PLLs may require bypass structures

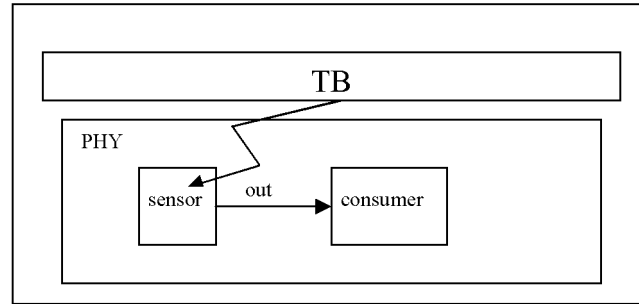


Figure 4. Environment Sensor within PHY

for the testing of the rest of the PHY. Each of these features introduce difficulties for verification.

In a transceiver pair connected to a pin, normally only the driver or receiver are active at a given time. By adding a model to the logic preceding the transceiver, a "loopback" test mode can be created to check the driver/receiver pair. For pins that are transmit only, an additional receiver may be added - solely to be used in IO test mode. Figure 5 shows two variants on an "Internal Loopback" test mode.

External Loopback modes test loop functionality out to the pins of the device. These modes pair a driver on one pin with a receiver on another pin. Exercising this mode in the lab requires a test board. For simulation, it requires an appropriate wiring harness. Because only half the drivers and receivers are tested in this type of test, a second test mode may be defined to test the other pairing of drivers and receivers. Each mode may require a different wiring harness. Figure 6 illustrates an "External Loopback" test mode.

Each of these IO test modes use the normal data path of the PHY and overlays test functions. This fact complicates both the design and verification phase, and the post-silicon validation phase. Functional verification becomes responsible for testing what is a test mode, while all other test modes are examined by the DFT team. At post-silicon validation, the part must be put into full operational mode to perform analog test. This procedure differs significantly from normal DFT flows in which simplified access through scan pins suffices.

What would be better is a structured analog-DFT methodology that clearly compartmentalizes normal operation modes and circuitry and "test" modes and circuitry. We believe that such a methodology is absolutely necessary, but that it will be a difficult sell. It is conceivable that such a methodology will have an area overhead and maybe even a performance impact. The benefits to yield and test throughput will have to be significant enough to justify the overhead for such an approach to be adopted.

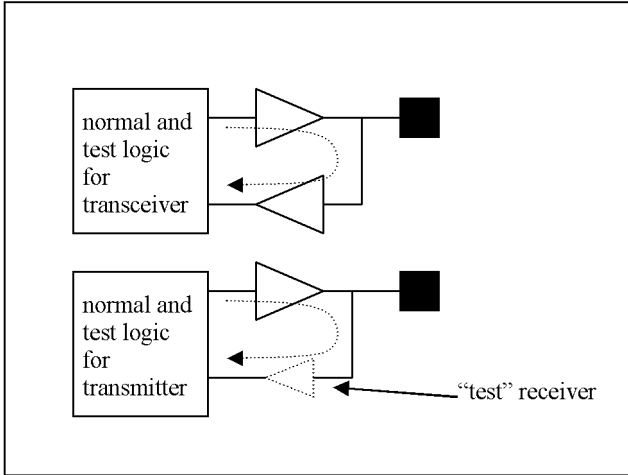


Figure 5. Internal Loopback for Transceiver and for Transmitter

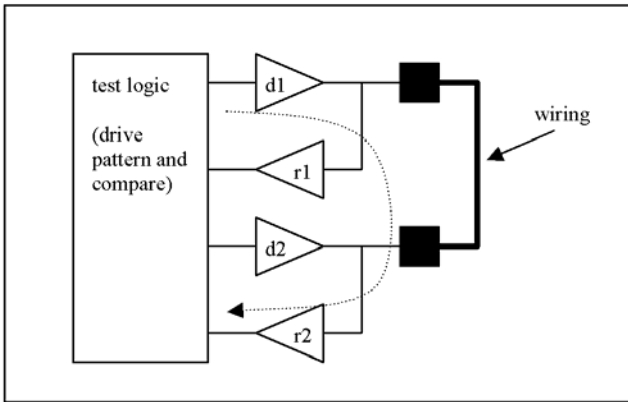


Figure 6. External Loopback

#### D. Embedded Logic

Some of the circuits employed by PHYs embed small portions of logic in blocks considered to be "circuit" blocks. One particular analog design technique that we encounter is a circuit block that uses a small amount of digital logic as a decoder on a digital input bus, as shown in Figure 7. The circuit implements a transform on analog signals based on a digital control. The digital portion is small, but is an integral part of the design.

This simple structure is a challenge in practice. The encoder is ordinary digital logic which may have errors, so it must be checked. It is tempting for some to abstract the behavior of the pair as a behavioral Verilog block. Doing so obscures the encoder from functional verification. A behavioral representation of the entire module is safe only if it is verified at the circuit level, but we have found that the tools and techniques used by circuit designers do not fully verify logic like this.

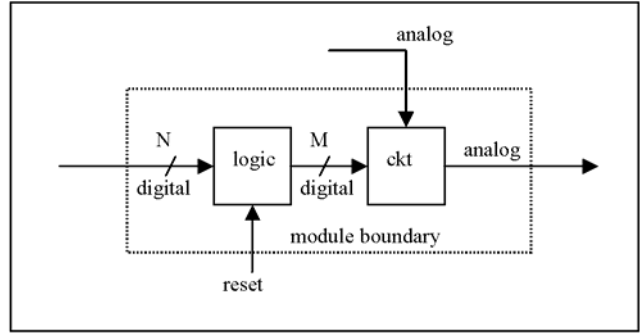


Figure 7. Embedded Logic

By choosing to model only the analog block in behavioral Verilog, the digital logic is exposed to the verification environment and can be verified. However, this approach also has limitations. The end-to-end data checking of the Phy-to-Phy verification environment may not detect errors in the encoder because this logic feeds an analog block, which may introduce a digital observability issue.

This structure is a good candidate for a localized assertion checker at the digital input to the analog block, but even that becomes complicated in practice. In correct circuit operation, there are constraints on the allowed digital codes, but these constraints depend on the context (global device state) in which the device is being operated. Some codes may normally be disallowed, but may be allowed during reset, initialization or in mode transitions. The device state that signifies that the device is undergoing reset may be distributed, or in the worst case, the state indicating the "intent" to switch modes may exist solely in the testbench.

While the temporal language aspects of current assertion technology are sufficient to express the properties we would like to express, what seems to be missing is a mechanism to collect the necessary global state to sufficiently contextualize the assertion checker. If the verification environment with DUT is considered a hierarchy, then the only point in the hierarchy that has visibility over all state elements is the testbench itself. However, the locations of the various states and signals that should be checked by the assertions are replicated HDL structures. What we would like is a flexible mechanism to combine non-local state data with assertion checkers placed near the properties they check.

#### E. Clocking

The example shown in Figure 8 is a simplified version of a design pattern found in a PHY for centering a sampling clock around off-chip data. Input data arrives at a port at a system defined frequency, but unknown phase. In order to center the sample clock to maximize setup and hold, the PHY calculates whether to sample the input data on the rising or falling edge of the clock. This simple edge selection mechanism selects between sampling at 0 or 180

degrees. By operating the main logic block at the same phase, a synchronizer insertion can be avoided.

One of the interesting points about a straightforward RTL modeling of this structure is that such a model does not readily reveal errors. Consider an error in which the edge selection logic always selects the rising edge. In an idealized logic simulator with idealized clock and data edges, the input data will be sampled reliably no matter what its phase, and a verification testbench that relies on detecting end-to-end data transmission errors might not catch this simple error. A directed test that targets the calculation of the edge select may be necessary.

Another option involves redefining clock and data edge transitions so that they pass through an intermediate "X" period. In practice, the library models of standard cells have very pessimistic "X" behavior, and the "X's" introduced by this approach usually introduce unforeseen side-effects, making it difficult to deploy in practice. The use of "Z" to model "no clock driven" introduces a similar problem, since this use of the "Z" symbol is not common to all of the underlying models.

What is missing is a structured discipline for reasoning about computations on clocks. The example above showed selection of a clock at 0 or 180 degrees, but we see more detailed control in practice. In these types of designs, logic not only gates the clocks - turning them on or off - but adjusts frequency and phase under programmable and adaptable control. RTL-based logic simulation is not very well suited for finding errors in these types of structures.

We imagine one of two types of tools helping here: one dynamic, the other static. A dynamic approach might analyze our designs for our company-specific clock-domain crossing patterns and produce a new netlist that helps illustrate logic-decided clock centering issues. A static

#### F. Feedback Control System

PHY designs are introducing increasingly refined digital control of circuit structures. We are seeing a proliferation of circuit structures that use feedback to adapt to many different device and system characteristics. An example is shown in Figure 9. The figure shows the major components of a feedback system that senses the drive strength of a PHY

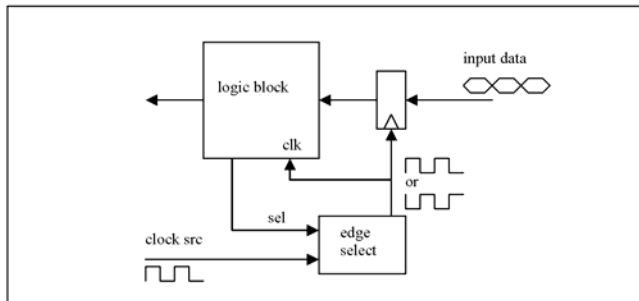


Figure 8. Computed Clock

using a comparator and adjusts the drive strength to match the reference value. The interesting thing about this structure, is that depending on the implementation transistors, the channel impedance and termination impedance, the optimal drive strength may vary.

Ideally, we want to ensure that the logic operates properly for any drive strength that a system may require. One approach to this end in an AMS setting would be to define all possible system configurations and use a pseudo-random approach to explore some subset of these configurations. A configuration is defined by device characteristics (transistor sizes and channel impedances), which are typically SPICE netlist compile-time constants. Thus, they are not under control of a CRT (constrained random testing) verification testbench.

Our configuration exploration approach allows netlist generation from a circuit template. In order to clearly explain the sequence of steps at which various parameters are bound to specific values, we have extended the simulation model of the Specman eRM and the System Verilog VMM and developed the simulation lifetime model shown in Figure 10. This has allowed us to recognize configuration parameters that are bound at netlist text file production, compile-time, elaboration time, and dynamically during simulation. The spans labeled "Configuration" and "CRT" show the influence of parameter binding decisions as they relate to the simulation lifetime.

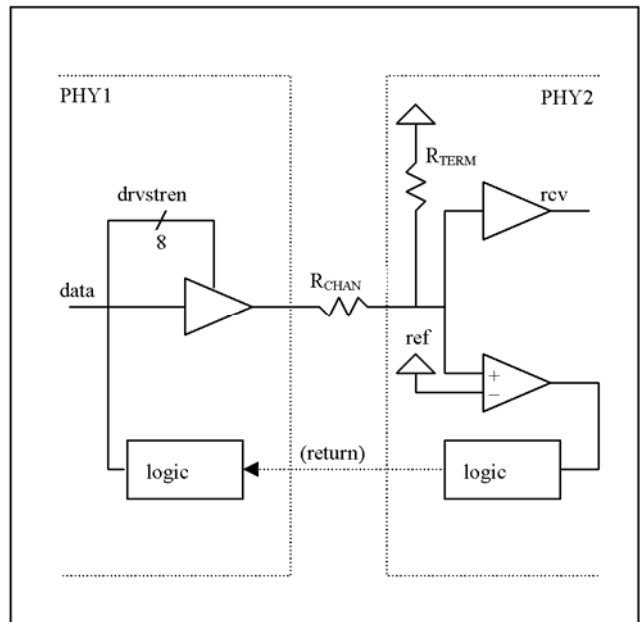


Figure 9. Feedback Control System

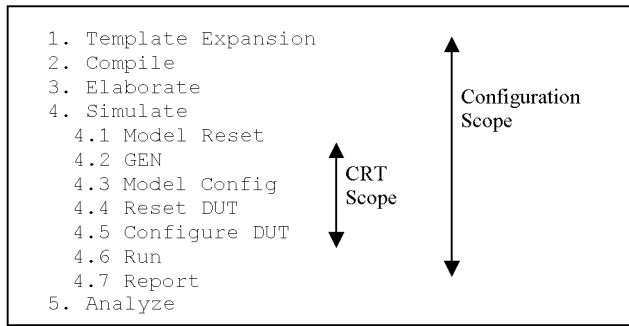


Figure 10. Simulation Lifetime

Because many netlist configuration decisions are made before simulation starts, CRT is not an effective means to explore a subset of our configurations. What we would prefer is a way to use CRT techniques to generate and monitor (cover) hardware configurations in a way that is completely integrated with simulation runtime parameter generation. We believe that an enhanced simulation lifetime model that encompasses hardware instance definition and elaboration is an important first-step to defining how constrained-random testing can be applied to sampling of architectural variants. We also believe that orchestrating the tools necessary to provide this capability in an AMS environment will be a complicated engineering problem.

We note that a two-phase approach has been suggested in which Specman or System Verilog is run once to generate the parameters of a hardware configuration, and then a second time to simulate that configuration [10]. This technique approximates pseudo-random configuration exploration, but is not the full solution. Future work in this area will need to address some of the following questions: Can coverage databases be merged when structural elements are present in one configuration but not another? If a circuit structure (such as a register) is present in some configurations but not others, what does it mean to cover it well? And lastly: this technique generates a number of configurations, using CRT as a random sampling of the state space of each configuration. What is the correlation between this type of coverage and random sampling of the entire configuration/state-space universe?

#### IV. CONCLUSION

This paper has presented a number of verification challenges that occur in PHY designs as a means of motivating a discussion about tools and methodologies that could be of assistance. We have touched on a wide variety of tools that could each help build a more robust verification system to pinpoint specific problems. This wish-list has included tools to better integrate testbench code with DUT-internal models, analog stimulus generation and response checking, a separate analog DFT flow, assertions that can use non-local information and a more flexible hardware configuration modeling capability under the control of the CRT scenario generator.

Each of these tools could help in the near term, but what we really need are disciplines that help separate the concerns that are intermingled in a PHY design. We need a way to design analog blocks, abstract their behavior for assembly into mixed-signal systems and verify the result in a way that does not require simulation at the SPICE level. We need an analog DFT discipline that separates normal and test modes as much as possible. We would also like tools that help maintain the correspondence of pieces of distributed structures - like the feedback control system archetype - so that as design proceeds and the pieces are distributed across different physical blocks, we can still relate the pieces back to the original design intent.

Ultimately, we believe new tools and representations will be necessary to simplify PHY design and verification. Schematic-driven layout - and even RTL-based design - are implementation approaches that do not do enough to capture architectural intent of our mixed-signal systems. The Verilog and SPICE level netlists we use for verification are reaching their limits, in terms of the types of things they model. Multiple design representations, architectural assertions, and enhanced automation for managing correspondences between logical and physical hierarchies are just some of the things we believe will be necessary to ensure correctness.

A PHY is one example of a type of device that blends digital and analog circuits. As more chip designs incorporate PHYs and mixed-signal designs become pervasive, we believe that the challenges presented here will become mainstream. We do not necessarily believe that we have the right answers yet as to how to manage these challenges, but we hope that by sharing our experiences we can encourage the development of stronger tools and methodologies.

#### REFERENCES

- [1] Janick Bergeron, Eduard Cerny, Alan Hunter, and Andy Nightingale. Verification Methodology Manual for System Verilog. Springer, 2006.
- [2] Mark Glasser, Adam Rose, Tom Fitzpatrick, Dave Rich and Harry Foster. Advanced Verification Methodology Cookbook, 2.0. Mentor Graphics Corp. 2006.
- [3] <http://www.verisity.com/products/erm.html>
- [4] Lionel Bening and Harry Foster. Principles of Verifiable RTL Design - A Functional Coding Style Supporting Verification Processes in Verilog. Kluwer Academic Publishers, 2000, p. 25.
- [5] T. J. Sheffler, K. M. Mossawir and K. D. Jones, "PHY verification - still an open problem," DesignCon 2007, Jan. 2007, in Press.
- [6] Ken Kundert and Olaf Zinke. The Designer's Guide to Verilog-AMS. Kluwer Academic Publishers, 2004.
- [7] Peter J. Ashenden, Gregory D. Peterson and Darrell A. Teegarden. The System Designer's Guide to VHDL-AMS. Morgan Kaufmann Publishers, 2002
- [8] The Designer's Guide to Analog and Mixed-Signal Modeling. <http://www.openmast.org>.
- [9] <http://www.knowlent.com>
- [10] <http://verificationguild.com/modules.php?name=Forums&file=viewtopic&t=1458>

