

DesignCon 2007

PHY Verification – Still an Open Problem

Thomas J. Sheffler, Rambus Inc.
sheffler@rambus.com

Kathryn M. Mossawir, Rambus Inc.
kmossawir@rambus.com

Kevin D. Jones, Rambus Inc.
kdj@rambus.com

Abstract

PHYs are devices that combine digital and analog circuits to transmit data over a physical medium. To reach higher data rates, PHYs are employing increasingly complex digital control of a greater number of analog characteristics. In addition to interacting digital and analog, complexity in a PHY results from interactions between other domains that are not easily compartmentalized. The verification engineer must consider interactions between analog and digital, discrete and continuous, normal and test-mode, clocks and logic, and physical and logical. For this reason, verification is a difficult problem.

In this paper, we share our experience in PHY verification by explaining a number of design patterns that introduce cross-domain complexity. For each pattern, we suggest some techniques that can help others avoid pitfalls. We show why we feel that good solutions do not exist and indicate directions that might lead to a more unified solution to the PHY verification problem.

Author(s) Biography

Tom Sheffler is a Senior Principal Engineer at Rambus Inc. He joined Rambus eleven years ago and has served in the verification and architecture groups. Tom received his Ph.D. and M.S. in Computer Engineering from Carnegie Mellon, and his B.S.E.E. from the University of Virginia.

Kathryn Mossawir is a Senior Member of Technical Staff at Rambus Inc., where she has worked in the verification group since joining the company in 2003. Kathryn received her M.S. in Electrical Engineering and B.S. in Computer Systems Engineering from Stanford University.

Kevin D. Jones is an Engineering Director at Rambus Inc., Los Altos, CA. He holds a Ph.D. in Computing Science from the University of Manchester, a M.Sc. in Computation from Oxford University and a B.Sc. in Computer Science from the University of Reading. Dr. Jones's areas of interest cover practical and theoretical approaches to design and verification of digital and analog circuits and systems. For the last 12 years at Rambus, he has been working on, and leading groups working on, practical verification for high speed digital/analog designs. He is currently leading a technology development

team focusing on novel design and verification tools and methods for this class of designs.

1. Introduction

A PHY is a component that transmits a raw bit stream over a "PHYSical" transport medium. Advanced PHYs have programmable and adaptable control logic that adjust the timing of clock edges and amplitude of transmitted signals. PHYs are fundamentally mixed-signal devices, containing a mix of synthesizable RTL control structures, custom analog circuits and custom digital IO pipelines.

Functional Verification has come to focus on the construction of Coverage-Directed Constrained-Random testbenches for Synthesizable-RTL designs. Increasing complexity in these types of designs arises from the layering of functionality that results from the integration of different functional blocks. In contrast, PHYs present increasing complexity as they provide more digital control of sub-cycle phenomena, such as clock phase and electrical characteristics of the bits transmitted over a physical medium. While tools such as Specman and frameworks such as VMM [1] and AVM [2] developed for logic verification are generally applicable to any scenario requiring stimulus generation, there remains other work to do in the verification of a PHY to address their particular type of complexity. Whereas in this paper we focus on PHY designs, we suspect that many of the problems presented here have similar counterparts in other types of mixed-signal designs and thus may be of interest beyond the realm of PHYs.

This paper discusses some of the challenges PHY verification presents to the Verification Engineer. A PHY is a mixed-signal (analog/digital) device, but it mixes other concerns as well. A verification approach based upon straightforward modeling of the logical behavior of the PHY with end-to-end data transport checking misses many of the interesting properties that a PHY must implement for correct behavior. A PHY bridges the physical and logical worlds, intertwines the continuous and discrete, and melds circuit and logic. Each such interaction must be considered in a successful verification approach.

In this paper, we discuss our experiences with these devices and share some advice regarding common pitfalls. We attempt to generalize each example to a class of problem not addressed by the current state of the art, and share why we believe PHY verification remains an open problem.

1.1 Interfaces and Complexity

In both software and hardware, complexity is managed through interfaces and abstractions. Interfaces are boundary definitions that allow the user of a software function or hardware module to understand its function without needing to know its implementation. Abstractions allow one to describe a design in a way that elides many implementation details. High performance PHY designs are not composed of modules

with clean interfaces and do not make use of common abstraction mechanisms. This contributes to their complexity.

In the software world, interfaces describe function and module boundaries. Abstractions such as "object-oriented" languages help to group data elements and functions in meaningful ways relevant to the task at hand. It is the job of the compiler and runtime system to restructure the code in a way that is efficient for execution.

In hardware design, interfaces describe module boundaries. Logic abstractions like "synthesizable RTL" and behavioral languages such as BlueSpec [3] and Synthesizable System-C [4] allow designers to describe algorithms and not think about gates. Synthesis is the automated process that refines the abstract design and produces gates.

Good interfaces and abstractions are transparent so that the final implementation may be predicted by a designer. This kind of transparency allows a designer to optimize a design knowing the transformations compilers and synthesizers will make. In this way, the programmer or designer retains control over the implementation, but does not need to track every implementation detail. The compiler or synthesizer does the detailed work and does not make mistakes.

PHY design confounds attempts to use logical interfaces and abstractions. In order to meet absolute maximal performance targets, clean logical interface boundaries are disrupted whenever a cross-boundary optimization can improve performance. Layout concerns (also driven by performance) can severely impact module boundaries as well.

Although digital synthesis is mainstream and analog synthesis is becoming available, hardware description abstractions simply do not yet exist for high-performance analog/digital systems. Without interfaces and abstractions, complexity in PHY design and verification is pervasive.

Another way to approach complexity is to understand design patterns. In the book "Design Patterns: Elements of Reusable Object-Oriented Software" [5], the authors state their goal as :

"The purpose of this book is to record experience in designing object-oriented software as design patterns. Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems."

This paper performs a similar function by describing design patterns that occur in PHY design and verification. These design patterns do not correspond to single modules, nor do they represent idioms that are repeatable enough that their implementation can be automated. However, by recognizing patterns that occur in PHY design, a collection of techniques can be developed that can be used for verification. A systematic approach towards verification of these individual patterns greatly reduces the complexity of verifying an entire PHY design.

1.2 Organization of this Paper

The rest of this paper is organized as follows. Section 2 further identifies the characteristics of Rambus PHYs. Section 3 identifies specific design patterns observed in PHY design and verification and notes how their complexity results from a combination of concerns that are normally compartmentalized. Where possible, we describe the failing of a common verification approach and suggest a better one. Section 4 concludes and suggests future research directions.

2. Characteristics of Rambus Mixed-Signal PHY Design

This section identifies a class of IP that characterizes PHY designs. 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 Rambus 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 and PLLs 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 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 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.
- Multiple clock domains.
- Possible sensing of electrical or physical link conditions.

3. PHY Challenges by Function

This section further refines the characteristics of PHY designs by calling out specific design patterns. For each of these patterns, we describe how complexity results from the blending of concerns that are normally compartmentalized. We then present some suggestions on how to handle the challenges presented by each of these patterns.

These examples also illustrate how straightforward component models of the devices in the examples do not make good verification models. That is, good models of normal-case behavior do not necessarily reveal simple design errors. Where appropriate, we will note a specific type of design error and show how a verification model does or does not detect it.

3.1 Digital Control of Signal Conditioning Circuits

Many analog aspects of PHY link-level driver circuits are under direct digital control. Transmitter signal swing and driver current are programmable. The symmetry of a signal's swing about a reference voltage may be software-controlled. Data edge rates are adjustable. Using programmable coefficients, pre-emphasis (equalization) may be used to tune the characteristics of the transmitted signal to the transmission medium.

Link level receiver circuits are also under digital control. Termination impedance is adjustable through the use of an on-die termination (ODT) circuit. When a Decision-Feedback Equalizer (DFE) is present, the feedback coefficients are under software control. In some PHY designs, a voltage comparator is part of the receiver circuit and is used in special modes to detect the levels of the received signal.

Although these digitally-controlled signal conditioning circuits are simple forms of digital/analog interaction, they introduce significant verification complexity. This particular type of digital/analog interaction obscures our ability to observe logic errors present in the digital control. Both constrained-random functional simulation with end-to-end checking and mixed-signal simulation suffer from observability problems.

An example of this design pattern is shown in Fig. 1. The transmitter of PHY1 has a driver of programmable strength; a digital value adjusts the strength of the signal transmitted over the channel. The "adjustable" pull-up and pull-down transistors shown are composite structures implemented as a weighted tree of transistors, but that detail is omitted here. On the receiving side, PHY2 implements on-die termination with a fixed pull-up impedance.

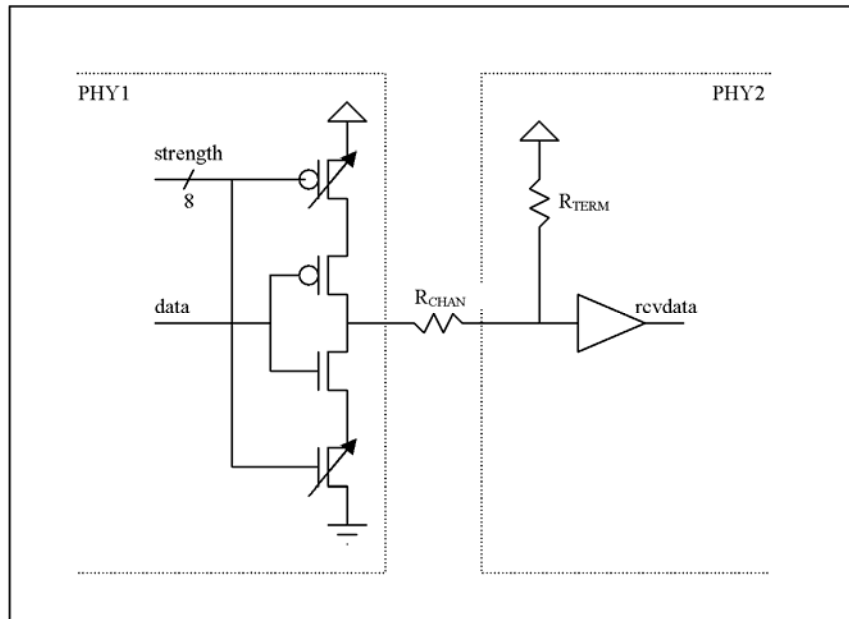


Figure 1. Details of Transmitter/Receiver Pair.

For functional verification of the PHY, we assume that there are no errors in the analog circuitry, and the job of the verifier is to ensure that the digital control values are delivered correctly to the transistor ports where they are consumed. A good mental check as to the suitability of a modeling and verification approach is to evaluate whether common errors such as crossed control wires, stuck-at control wires or inverted control wires would be detected. It is desirable that such simple errors are detected directly.

A common approach to functional verification is digital simulation using Verilog or VHDL (mixed-signal simulation is discussed separately below). To use this approach with this design pattern, we must replace the analog portions of the circuit. One modeling option replaces the driver and receiver with behavioral blocks that abstract their function to two-valued (or four-valued) logic. Because the channel is a one-bit wire, the straightforward behavioral representation of the programmable driver is a Boolean value that does not encode the drive strength. Since the drive strength information is not propagated beyond the digital-to-analog transition within the driver, we cannot observe errors in the drive strength control using end-to-end checking. The verifier should instead include a direct check of the drive strength at the digital-to-analog boundary by probing the digital control value directly while checking end-to-end data transfer as usual. This additional observation point, as illustrated in Fig. 2, allows the digital control value to be verified without losing the speed and abstraction of digital simulation. The cost of this approach is that the testbench must now include the ability to predict the correct value for the digital control at the added observation point.

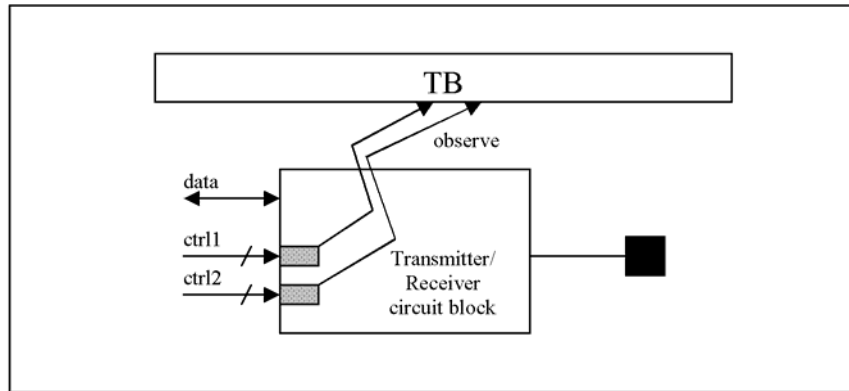


Figure 2. Observation of Signal Conditioning Circuit Control Ports

An alternate Verilog substitution replaces the channel wire with a bus that encodes the strength value. This method strives to propagate strength control errors beyond the digital-to-analog transition. Such a structural replacement is possible, but disruptive to normal tool flows. Assuming one does encode the strength as a bus, verification of the strength control logic also requires that the strength value is consumed in some way by the receiver. This will introduce some type of strength value checker on the receiving side. This approach is an intermediate step between looking at the values at the boundary of the digital domain and the fully analog modeling described below. While there are some circumstances where it is beneficial, we believe that in most cases, detecting logic errors can be done more efficiently using the approach in the preceding paragraph.

3.1.1 Inefficiency of Mixed-Signal Verification

Using Verilog-AMS to more accurately model the DUT can help find logic errors, and we do use it in our flow. However, for many types of errors that occur in this design pattern, the approach is not efficient.

The reason for a programmable strength driver is to compensate for process, voltage and temperature (PVT) variances in the PHY driver, receiver and data channel. This critical design feature turns out to be a liability for functional verification, wherein we strive to catch logic errors without modeling PVT variances. When using mixed-signal simulation with this design pattern, PVT variances must be considered in order to catch some simple logic errors in the digital control value.

For normal case mixed-signal simulation, the driver transistors and termination resistance are sized such that there is a digital strength value $0 < S_{nom} < S_{max}$ that produces a nominal case working system that transmits data correctly. If $S_{nom} > 1$, then there is a value S_{min1} that is the minimum strength value that permits successful data transmission where the value $(S_{min1}-1)$ does not transmit data.

Call the interval $[0..S_{\min1}-1]$ the "failing" set of strengths and the interval $[S_{\min1}..S_{\max}]$ the "passing" set. By constructing a directed test that varies the strength value and checks whether data transmission fails or succeeds, we can detect logic errors that manifest themselves as a mapping of a passing strength to a failing strength, or vice versa. However, logic errors in which the correct and incorrect values are both in the failing set or both in the passing set are not detected. This observability problem can be attributed to the design pattern having a multi-valued digital strength control combined with a two-valued data response. Since there are more input values than output values, there can be incorrect strengths that produce the same output as the correct strength, and are therefore undetectable. Moreover, the strength threshold at which the data is successfully transmitted or not varies with PVT (which is why the strength is programmable in the first place), and although the error may not cause a problem at the typical PVT setting, it may result in a system failure at a corner-case PVT setting.

Instead of running many mixed-mode simulations at various PVT settings, which would be too slow, we can develop a decision procedure to search for this design error. First, we test the system with $S_{\min1}$ as the minimum strength value that permits successful data transmission. Next, with a different configuration of transistor sizes, channel and termination impedance, etc., we can find another value, $S_{\min2} \neq S_{\min1}$, and repeat the process of examining all of the strength values. This step will partition the strengths differently and rule out other logic permutations. Continuing the process, we can rule out all logic permutation errors by designing a separate test system for each possible strength value.

This process, while complete, is inefficient in practice. The construction of the different variants of the system along with a test specific to the logic being verified is a large overhead to pay to detect a basic logic error. A more efficient solution might be possible if the received voltage level is checked directly, but this would require that the testbench include the ability to accurately predict the correct voltage level, where the definition of "correct" may be more complex than a single voltage at one instant in time.

This section demonstrates that sometimes the "obvious" approaches to mixed-mode simulation do not give optimal results either in finding errors, or in finding them efficiently. It is important to carefully select the modeling and verification approach, including the level of abstraction and observation points, in order to be able to detect critical logic errors.

3.2 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. This design pattern bridges the physical and logical, and in the case of triggered sensors, the continuous and discrete. Each interaction introduces verification complexity.

The input to this type of sensor circuit is represented by a pure source in the netlist, i.e. something that produces a value that appears non-deterministic to the rest of the system. The job of the verification engineer is to ensure that the logic is correct for all values produced by the source. The output value of the sensor should be under control of the testbench so that it can be varied, and witnessed by system checkers (Fig. 3). 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 must be added. Tools such as Specman allow the verifier to connect directly to an internal wire using a pathname, but problems arise when the design hierarchy changes or the block is re-used in another design, adding significant maintenance complexity.

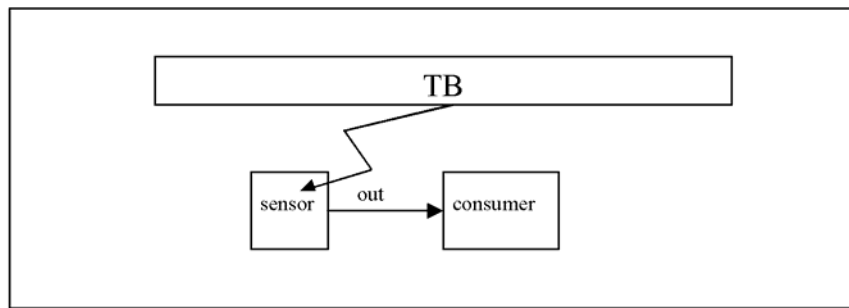


Figure 3. Sensor with Input Value Provided by Testbench

A related structure is the triggered sensor (Fig. 4). In this variant, an enable signal causes the sensor to begin detecting an environmental condition. At some time later, the output value is sampled by client logic. With a triggered sensor, the verification engineer has the additional job of ensuring that the client logic only consumes the output of the sensor when it is valid, and not before.

The amount of time that must elapse between the trigger and the sample is a continuous time value. Call this time " t_{STABLE} ". The system cycle time is " t_{CYC} ". The client logic is designed to count out a discrete " N " cycles after the trigger before it samples the sensor output. It should be the case that " $N * t_{\text{CYC}} \geq t_{\text{STABLE}}$ " so that the client logic does not sample the sensor before its output is valid. The constraint that " $N * t_{\text{CYC}} \geq t_{\text{STABLE}}$ " should be captured in the design, such that if it is violated an error is signaled. Such a violation can occur because of a logic error, or because a design is improperly scaled to a higher system frequency.

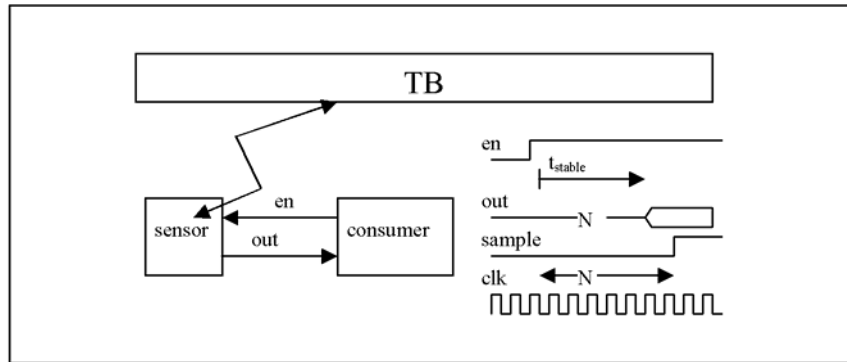


Figure 4. Sensor with Output-Sample Constraint

Modeling the sensor and designing the client logic such that violations of the sample constraint are detected requires care. For reliable detection of violations, the sensor must send an "invalid" value when the output should not be relied upon - measured in continuous time - and the consumer must explicitly raise an error if it attempts to sample the value when it is invalid. It might seem obvious to model the sensor so that it presents an "X" when its value should not be sampled, but this technique works only if the receiving logic detects or propagates the "X" value. For example, if the sensor value is used as the conditional in an "if" statement in the RTL, the "X" value is treated as "false", and the violation is hidden. Often, the modeling of the sensor is left to a circuit designer and the data capture is implemented by a logic designer. Detecting the capture error requires clear communication of the assumptions made by each engineer.

3.3 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. Complexity is introduced by the lack of separation between "normal" and "test" modes.

In a transceiver pair connected to a pin, normally only either the driver or receiver is active at a given time. By adding a mode to the logic preceding the transceiver, a "loopback" test 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. Fig. 5 shows two variants of an "Internal Loopback" test mode design pattern.

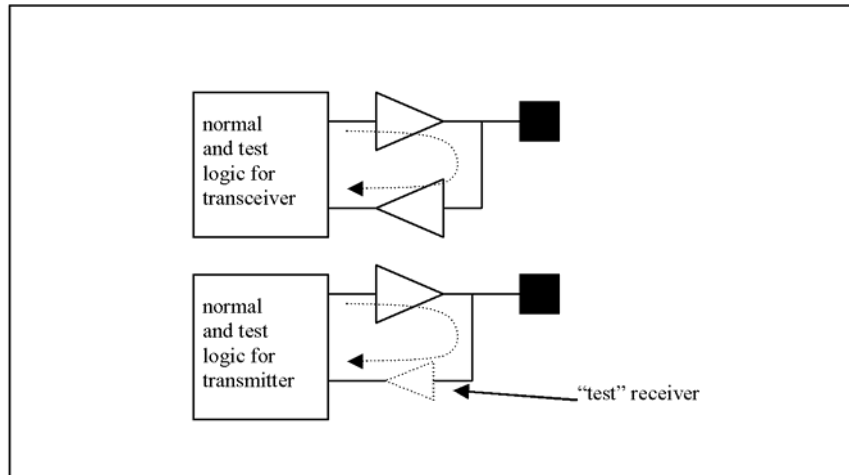


Figure 5. Internal Loopback for Transceiver and Transmitter.

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 at one time, a second test mode may be defined to test the other pairing of drivers and receivers. Each mode may require a different wiring harness, adding some manual work and complexity. Fig. 6 illustrates an "External Loopback" test mode design pattern.

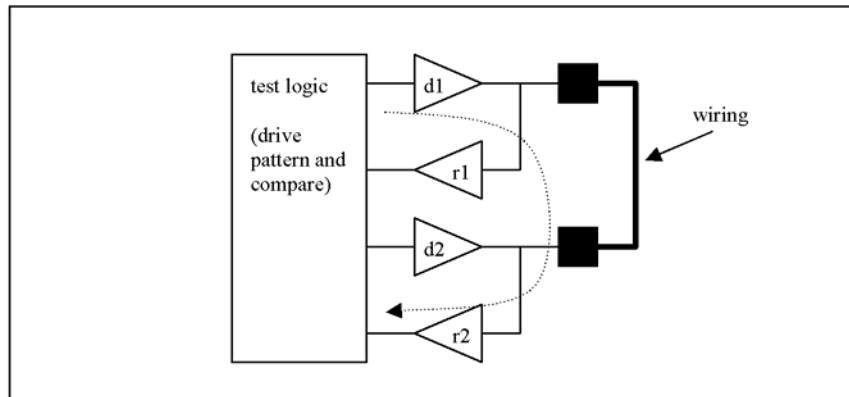


Figure 6. External Loopback for Typical Transceiver and Transmitter Structures.

Another variant on external loopback implements a form of self-test. In this mode, a test pattern is driven and the received value is compared to the value driven using self-test logic in the PHY (Fig. 7).

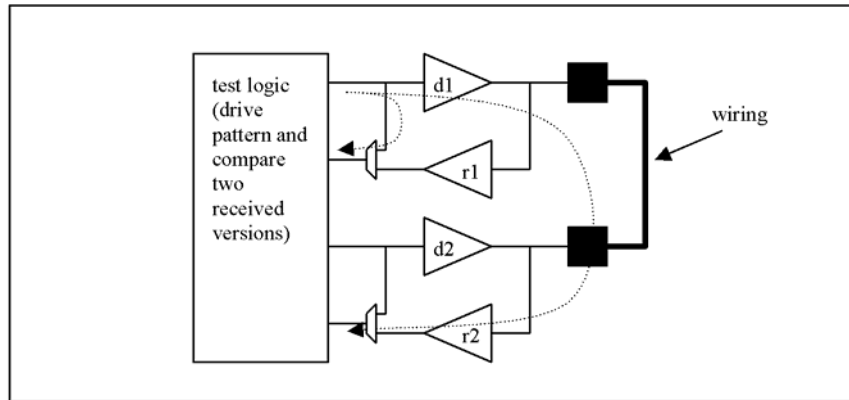


Figure 7. External Loopback Variant.

Each of these IO test modes uses the normal data path of the PHY and overlays test functions. The disciplines of logic design and DFT are intertwined as are the roles of verifier and test engineer in regards to IO test. In practice, exercising each of these test modes requires a separate initialization sequence and wiring harness.

PLLs also introduce test problems. Most PLLs do not operate at very low frequencies. Thus, for low-speed lab debug it is important to provide modes that disable the PLLs and operate the PHY using externally supplied clocks. This is another example of a test structure (PLL bypass) that is intertwined with "normal mode" operation (Fig. 8).

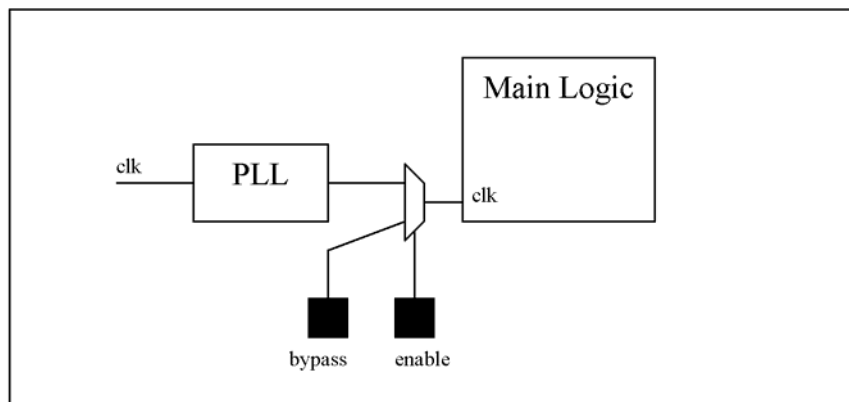


Figure 8. PLL Bypass Logic and Pins

It is important that logic verification ensures the correctness of the muxes, logic and wiring added for this type of test mode. The addition of this small amount of circuitry can double the number of behaviors that need to be verified. Unless the effects of the Bypass mode can be clearly separated from Normal mode, then all Normal mode behavior must be checked again with Bypass mode enabled.

The extra complexity introduced by this design pattern is necessary because of the current lack of well-developed techniques and tools for DFT-like functionality for mixed mode designs. The added complexity described here both obscures normal-mode verification and introduces a potential new set of design errors. It would be desirable to have the orthogonality between normal and test modes that automated test pattern generation (ATPG) provides for digital logic. However, there is currently no DFT-analogous approach for designs involving analog blocks.

3.4 Embedded Logic in Circuit Blocks

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 to use a small amount of digital logic within a circuit block as a decoder on a digital input bus. 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 (Fig. 9).

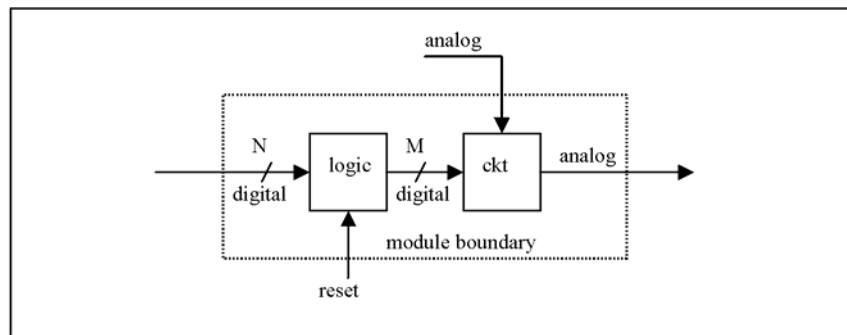


Figure 9. Embedded Logic

This simple structure presents a hazard. The decoder is ordinary digital logic which may have errors, so it must be verified. For functional verification of the design, it is tempting to abstract the behavior of the pair as a single block, because the behavioral description of the two together is simpler than the description of the analog block alone. Doing so, however, obscures the encoder from functional verification. Such a substitution is safe only if the pair is completely verified at the circuit level. However, the circuit designer's tools are relatively inefficient for functional verification, so this is not a good choice.

Choosing to model only the analog block in behavioral Verilog also presents a challenge. A simple assertion checker at the digital input to the circuit would seem straightforward, but becomes complicated in practice because it produces too many "false failures". In our designs, "illegal" digital codes are allowed for performance reasons, but only in specified short periods of time, such as reset, during initialization, or in major mode transitions. Unfortunately, detecting the intent to switch modes requires knowing what the user (or testbench) is intending to do. Writing this type of internal checker requires tight integration with the testbench.

In cases where digital and analog circuitry is tightly coupled, the boundary at which analog blocks are modeled for functional verification must be chosen carefully. With current tools there is an unrewarding tradeoff between time spent in circuit-level verification and time spent developing testbench capabilities.

3.5 Clocking

The PHY is often responsible for synchronizing on-chip and off-chip communications. In structured digital logic design, simplified rules specify that one should avoid computed clocks. However, to meet performance goals when synchronizing on-chip and off-chip communications, high-speed PHY designs violate these rules to reduce latency and to align key clocks. Complexity arises from the resultant intertwining of logic and clocks.

The example below is a simplified version of a design pattern found in real devices. 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.

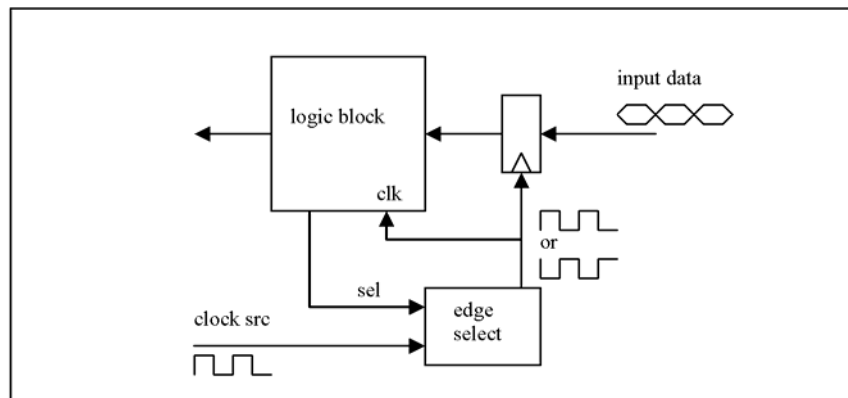


Figure 10. Computed Clock

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 error 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 go through an intermediate "X" period. The "X's" of this approach usually introduce unforeseen side-effects, making it difficult to deploy in practice.

This design pattern represents a way of synchronizing data that is not supported by many digital logic tools designed to look for timing errors, such as static timing analysis tools and clock-domain-crossing checkers. It is another style of design altogether that requires its own verification technique.

3.6 Feedback Control System

PHY designs are introducing increasingly refined digital control of circuit structures [9]. A design pattern that occurs in this context is that of the feedback control system. An example of feedback control system that uses a comparator and digital logic to adjust the drive strength of a communication channel is shown in Fig. 11. In the generalization of this design pattern, the return path may be physical, or may be a virtual path multiplexed over another signal. This design pattern also shows up in feedback control of other link characteristics, such as clock phase, where the voltage comparator is replaced with a phase comparator.

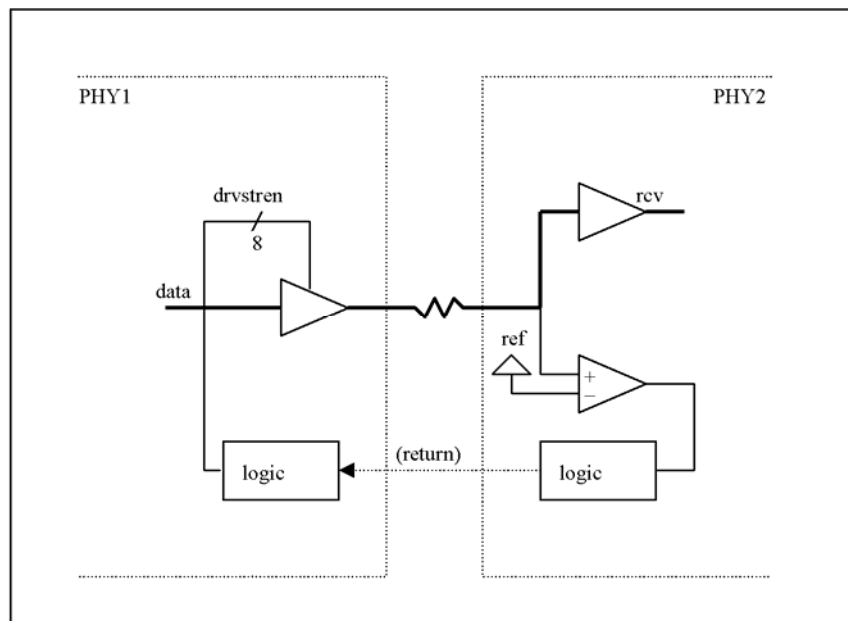


Figure 11. Feedback Control System

This structure introduces complexity in a number of different forms. It blends analog and digital behaviors and the system behavior spans blocks in different devices. Such "feedback" structures are becoming more prevalent as the idea of digital correction of analog circuit imperfections proves useful.

For functional verification, behavioral Verilog block replacements have all the limitations of section 3.1, and this pattern introduces additional difficulties. To exercise the feedback control system, we also require a means to introduce perturbations into the communication channel to check the behavior of the control algorithm. For this purpose, we recommend a merged channel model, where one model abstracts the driver, channel

and receiver, with explicit perturbation control from the testbench. Using this type of verification model allows the testbench explicit control over channel transmission characteristics. Because the verification model spans multiple devices, it is not a simple block substitution, and its implementation may be time-consuming and error-prone.

In the extreme case, the feedback system would involve software running on a microcontroller which monitors the system and implements the feedback control. This system would contain the following: software monitoring and controlling digital logic, digital logic controlling and reporting about analog signal values, and analog circuitry interacting with the physical channel. There is no simplistic way of modeling and simulating such a system which gives both speed and accuracy.

4. Conclusions

PHY designs introduce complexity in the form of many pair-wise interactions between domains that people would prefer to separate. We have shown examples of such interactions between digital and analog, discrete and continuous, normal and test, logic and circuit, clocks and logic, as well as for feedback systems adjusting these interactions. Each of these introduce small amounts of complexity on their own. But the presence of all of them in one design means that PHYs are very complex devices.

PHY complexity requires that designers juggle many concerns at once because specialized tools and automation do not exist for their particular types of difficulties. Verifying that all of the decisions a PHY designer made are correct is at least as complex as making them correctly the first time. Thus, we require a way to record the intent of the designer and check it repeatedly.

We do not believe that a single design model or representation is suitable and efficient for verifying all of these interactions. Rather, specific models, substitutions or techniques are best for exercising and testing each design pattern. Relying only on end-to-end data checking misses many important classes of errors. Thorough verification of a PHY design requires careful modeling and tight links between the testbench and simulation model. While today's modeling and verification approaches using digital and mixed-signal techniques can reveal common errors in PHY designs, they work only if such errors are anticipated and handled as special cases. We would prefer a more unified approach.

There are clearly many areas of research that can help to close this open problem. We believe the following are needed to accurately verify a PHY: collections of models representing different abstractions of the PHY, each useful for finding a different type of error; information on how each model relates to the others such that they may be shown to be consistent; and finally assertions annotating the design with information on the intent of the structures within the design, whether analog or digital, physical or logical. We are currently working in this direction. Identifying and characterizing the issues is a necessary step to evaluating potential solutions and we believe this paper contributes to that effort.

References

- [1] Verification Methodology Manual for SystemVerilog. Janick Bergeron, Eduard Cerny, Alan Hunter, Andy Nightingale. 2006 Springer.
- [2] AVM Cookbook. http://www.mentor.com/products/fv/events/avm_cooking.cfm
- [3] <http://www.celoxica.com>
- [4] <http://www.bluespec.com>
- [5] Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. Published by Addison Wesley Professional. Series: Addison-Wesley Professional Computing Series. Published oct 31, 1994 Copyright: 1995.
- [6] The Designer's Guide to Verilog-AMS by Ken Kundert, Olaf Zinke Copyright 2004 Kluwer Academic Publishers Norwell, MA 02061 ISBN: 1-4020-8044-1.
- [7] The System Designer's Guide to VHDL-AMS Peter J. Ashenden, Gregory D. Peterson and Darrell A. Teegarden Published by Morgan Kaufmann Publishers, San Francisco, September 2002, ISBN 1-55860-749-8.
- [8] The Designer's Guide to Analog and Mixed-Signal Modeling, download from <http://www.openmast.org>
- [9] A Serial-Link Transceiver Based on 8-GSa/s A/D and D/A Converters in 0.25- μ m CMOS. Chih-Kong Ken Yang, Vladimir Stojanovic, Siamak Modjtahedi, Mark A. Horowitz, William Ellersick. Solid-State Circuits, IEEE Journal of Nov 2001, Vol. 36, no. 11, pp. 1684-1692.