Functional Verification of a SCI-PHY Level 2 Protocol Engine

Leila Barakatain[1], Sofiène Tahar[1], Jean Lamarche[2] and Jean-Marc Gendreau[2]

[1] Department of Electrical and Computer Engineering, Concordia University
Montreal, Quebec, H3G 1M8 Canada

[2] PMC-Sierra, Inc., Montreal Design Center
Ville Mont-Royal, Quebec, H3R 3L5 Canada

Abstract

In this paper we present our results on verifying the implementation of a SCI-PHY (Saturn Compatible Interface for ATM-PHY devices) Level 2 protocol engine, commercialized by PMC-Sierra, Inc. Using the FormalCheck tool, we first established a proper verification environment, then defined a number of relevant liveness and safety properties, which we checked on an abstracted (8 PHY devices) as well as the original (32 PHY devices) hardware model. Throughout our verification efforts, we uncovered a number of mismatches between the hardware design in VHDL and the SCI-PHY protocol specification. Some of these are pertinent design bugs which were not caught during the simulation process.

1. Introduction

With the increasing reliance of digital systems, design errors can cause serious failures, resulting in the loss of time, money, and long design cycle. Large amounts of effort are required to correct design errors, especially when the error is discovered late in the design process. For these reasons, we need approaches that enable us to discover errors and validate designs as early as possible. Conventionally, simulation has been the main debugging technique. However, due to the increasing complexity of digital systems, it is becoming impossible to simulate large designs adequately. Therefore, there has been a recent surge of interest in alternative/complementary techniques, such as formal verification [4]. In formal verification, a mathematical model of the design is compared with a formal specification describing the correctness criteria for the design. The verification is exhaustive: all possible behaviors of the model are considered.

In this paper, we describe the functional verification of a protocol engine implementing SCI-PHY—Saturn Compatible Interface for ATM-PHY (Asynchronous Transfer Mode physical layer devices). The SCI-PHY Level 2 protocol is a superset of UTOPIA (Universal Test & Operations PHY Interface for ATM) Level 2 protocol [1]. The design we considered implements the output port of a cell interface and is product of PMC-Sierra, Inc. [6]. It can be configured to operate either as a bus master or bus slave. In this paper, we present our results of formally verifying the SCI-PHY protocol engine using FormalCheck [3]. FormalCheck is a model checking tool from Cadence, that is used to verify synthesizable Register Transfer Level (RTL) VHDL or Verilog design models.

The rest of the paper is organized as follows. In Section 2, we briefly describe the structure of the SCY-PHY protocol hardware design and behavior. In Section 3, we overview the FormalCheck tool. In Section 4, we present the verification environment we adopted for verifying the SCI-PHY protocol design. In Section 5, we describe sample properties we checked using FormalCheck. In Section 6, we display our experimental results and discuss some of the errors we uncovered. In Section 7, we conclude the paper.

2. The SCI-PHY Protocol Engine

The SCI-PHY protocol was defined within the SATURN Group [5] as a standardized cell-based interface between ATM layer and PHY layer devices to support single-PHY and multi-PHY applications. The SCI-PHY Level 2 protocol is a superset of UTOPIA (Universal Test & Operations PHY Interface for ATM) Level 2 protocol [1]. SCI-PHY Level 2 is an extension of SCI-PHY, that leaves all of the basic specifications and operations unchanged, but adds two important interface specifications (1) a Physical Medium Dependant (PMD) to Transmission Convergence (TC) interface specification that is compatible with all major vendors of 155 and 622 Mbit/s PMD and TC devices, and (2) an ATM Layer to Switch interface specification that provides a general purpose “extended-cell” format that will accommodate most ATM Layer implementations [5].

The SCI-PHY protocol engine we consider here is a Telecom System Block (TSB) designed by PMC-Sierra, Inc. It implements the output port of a cell interface and can output the cell data either in 8-bit or 16-bit wide format at clock rates up to 52 MHz [6]. Data transfers are cell-based, that is an entire cell is transferred to one PHY device before another is selected.

2.1. SCI-PHY Cell Format

The 8-bit wide, variable data structure at the SCI-PHY interface is shown in Figure 1. A user defined (UDF) byte is included in the data structure to allow Header Error Control (HEC) generation to be performed either in the ATM layer device or the PHY layer device. The prepended bytes are used by ATM switch cores in system specific ways to route the cell through those cores [5].

The SCI-PHY protocol TSB is designed to interface directly to a Multi-channel Cell FIFO (MCF). It directly supports up to 32 logical channels each corresponding to a physical layer (PHY) ATM device. Each logical channel corresponds to a FIFO channel in the external FIFO. When the TSB is operated as a bus slave, it autonomously multiplexes the traffic from up to 32 logical channels and presents them as a single

cell stream. The logical channel is identified by the first word of the cell data received from the FIFO.

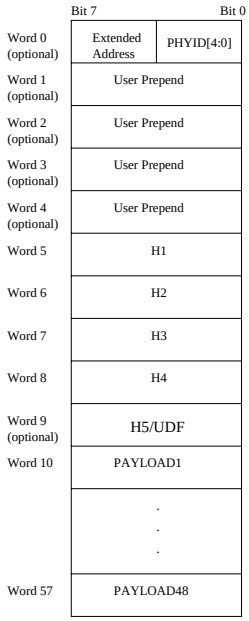

Figure 1: 8-bit SCI-PHY Cell Format

2.2. SCI-PHY Protocol TSB

The SCI-PHY protocol TSB design consists of two major blocks, the CBI block and SCAN block (Figure 2). The CBI block is used only for the configuration and the test interface and the SCAN block drives the main functionality. The SCAN block consists of four main blocks, namely, Datapath, Polling controller, FIFO interface controller and Transfer controller. The Polling controller block handles the control signals related to the polling of the protocol engine in the Receive-Slave mode and polling the PHY devices in Transmit-Master mode. The FIFO interface controller block decides if any of the 32 FIFOs has a cell available and also decides which FIFO channel should be selected. The Transfer controller block determines when the transmission of a cell starts, and the Datapath block actually handles the transmission of data.

3. The FormalCheck Tool

FormalCheck [3] is a functional verification (model checking) tool commercialized by Cadence intended to augment conventional simulation techniques. It is used to verify synthesizable Register Transfer Level (RTL) VHDL or Verilog design models using model checking. FormalCheck provides an intuitive graphical interface to simplify the verification process.

A model checker verifies that a design model exhibits specific behaviors (*properties*) that are required by the design specification. Properties that form the basis of a model checker's queries fall into two categories: *safety* and *liveness*. *Safety* properties describe behaviors that can be shown to be false by a finite simulation trace. Safety properties use one of the two formats: the *always* format and the *never* format. *Liveness* properties describe behaviors that are eventually exhibited. Liveness properties cannot be checked with a simulation tool, unless the maximum number of steps before the fulfillment of the eventuality is known. Liveness properties use one of the three formats: the *Eventually* format, the *Eventually Always* format, and the *Strong Liveness* format.

FormalCheck works by checking a *query*. A query is a group of properties that are to be checked together subject to specified *Constraints*. FormalCheck provides run options that are used to select which algorithms are used in a verification run. Proper use of run options will ensure an efficient verification.

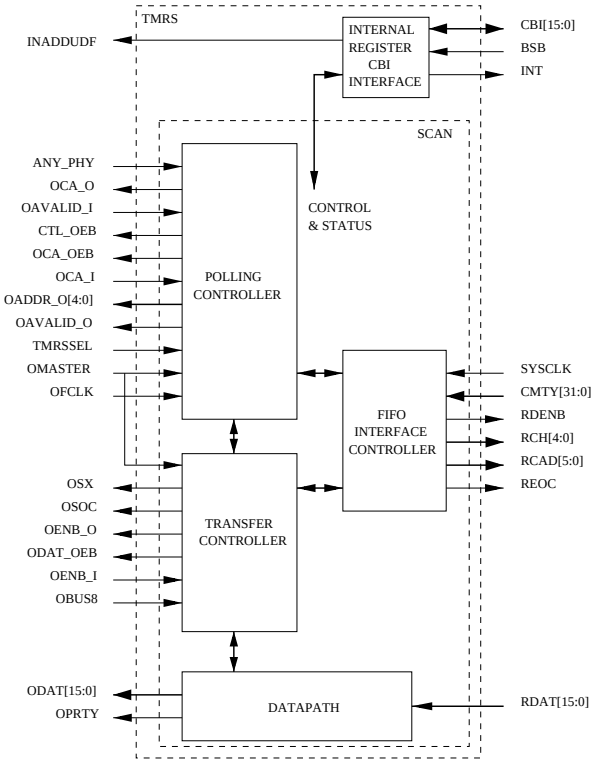

Figure 2: SCI-PHY Protocol TSB

4. Verification Environment

In following we describe the environment set-up we built in order to verify the SCI-PHY protocol design within FormalCheck in an efficient way.

4.1. Model Reduction and Abstraction

The SCAN block of the SCI-PHY protocol TSB consists of four main blocks (Figure 2), namely, Datapath, Polling controller, FIFO interface controller and Transfer controller, where the Polling controller, FIFO interface controller and Transfer controller were of interest in our study. To reduce the state space and speed-up the verification, we tried to trim the protocol design by eliminating the blocks that did not have any or very little effect on the three control blocks. The abstractions and reductions adopted are as following:

Table 1: Properties for the Receive Slave SCI-PHY mode

Property	Source(s)	Brief Description of Property
Property_1	SCI-PHY Doc. & Test bench	OENB_I = 0 & TSB Selected then OSOC = 1 & @StartTxState
Property_2	SCI-PHY Doc. & TSB Spec.	If TSB is polled and all CMTY[31:0] lines are high or the prefetch of the next FIFO is not finished, OCA_O will be deasserted
Property_3	TSB Spec.	In Back To Back Transfer Mode: @TransferDone & OENB_I = 0 & AllFifoEmpty = 1 then ODAT_OEB = 1
Property_4	SCI-PHY Doc.	No PHY shall drive RCA (OCA_O) upon sampling RAVALID (OAVALID_I) low
Property_5	SCI-PHY Doc. & TSB Spec.	@TSBpolled & (more cell available to transfer & @PrefetchDone) then OCA_O = 1
Property_6	TSB Spec.	In Back To Back Transfer Mode: @TransferDone & OENB_I = 0 & New_Cell_Transf_Rdy = 1 then Eventually OSOC = 1 & @StartTxState
Property_7	SCI-PHY Doc., TSB Spec. & Test bench	TSB always expects a complete cell transfer but supports transfer interruption by deassertion of OENB_I
Property_8	SCI-PHY Doc.	If master selects the TSB before the prefetch cycle is done, the TSB will not start a cell transfer in the next C.C.
Property_9	SCI-PHY Doc. & TSB Spec.	When operating in the Slave mode the TSB monitors the input signal OENB_I to validate the data transfer
Property_10	SCI-PHY Doc. & TSB Spec.	TSB Selected & OENB_I = 0 then ODAT_OEB = 0
Property_11	TSB Spec.	TSB not Selected & @TransferComplete then ODAT_OEB = 1

- We isolated the SCAN block, hence, eliminating the CBI block used for test purposes.
- We removed the INPUT_MUX module from the SCAN block. We modeled the outputs of this block which were used by the control blocks, inside the SCAN block.
- We removed the DATAPATH module from the SCAN block.
- We made an abstracted model of the SCAN block to support only 8 PHY devices, to reduce the verification time and also the memory used by FormalCheck. Once we succeeded verifying the properties for the abstracted model, we verified the same properties on the model which supports 32 PHY devices.

4.2. Environment Constraints

In FormalCheck, environment abstraction is possible using *Reduction Seeds*, *Electronic Scissors*, and *Constraints* [3]. By adding Constraints to the properties, we reduce the state space explored in verification and therefore improve the overall performance, which means less CPU time and/or memory usage [3].

Correct design behavior requires certain input behavior. In FormalCheck, primary signals are assumed to be non-deterministic, meaning they could acquire any value within their range on any edge of the clock. However, in most cases correct design operation is allowed on a single edge of the clock. For this reason, properties should be observed using the appropriate clock edge. Examples of Constraints we defined for properties checking on the TSB are:

- A *Clock* Constraint on OFCLK and SYSCLK signals (refer to Figure 2), starting with high for one cycle and then low for one cycle.
- A *Reset* Constraint on the RST signal (Figure 2), starting with high for 2 clock cycles and then goes to low forever.
- A *Group* Constraint to drive to SCI-PHY slave mode.
- A *Group* Constraint to drive the TSB to default mode of operation.

5. Properties Specification

Once the proper environment established, we defined a set of relevant properties based on the SCI-PHY Level 2 protocol specification. Table 1 shows a brief description of the properties established to verify the Receive Slave SCI-PHY mode of the protocol engine TSB. In this section, we describe three sample properties, including liveness and safety properties. A complete description of all properties is reported in [2]. The experimental results of the properties verification are summarized in Tables 2 and 3 (Section 6).

Property_2: According to the SCI-PHY protocol: “Each PHY link shall have a unique address corresponding to a value between 0 and 31. Upon sampling its address with the rising edge of the RFCLK, a PHY must drive RCA to indicate whether it has an entire cell in its buffer” [6]. For the TSB design, this property is expressed as following: “When OAVALID_I and TSBSEL are asserted (TSB is polled) and all CMTY[31:0] lines are high or the prefetch of the next FIFO is not finished, OCA_O will be deasserted” [5]. In FormalCheck, this property is described as follows.

```
Property: property_2
Type: Always
After: (@TSBpolled) and
       (Polling_Sm_Inst:Allfifoempty = 1)
       and (@RstDone) and (@CLKrising)
Always: (not @RCAhigh)
```

Property_7: According to the SCI-PHY protocol: “The ATM layer device may pause the transfer at any time by deasserting RRDENB” [6]. For the TSB design, this property is expressed as following: “The Slave Transfer State

Machine always expects a complete cell transfer but supports transfer interruption by deassertion of the OENB_I signal" [5]. In FormalCheck, this property is described as follows.

```
Property: property_7
Type: Eventually Always
After: (not @TransferDone) and (OENB_I =1)
       and (@RstDone) and (@CLKrising)
Eventually Always: @TransferDone
Unless: (OENB_I =1)
```

Property_10: According to the SCI-PHY protocol: "When RRDENB is sampled low by the PHY layer device, the RDAT bus will be accepted by the ATM layer device on the next rising edge of RFCLK" [35]. In the TSB when operating in Slave mode, this property is expressed as following: "ODAT_OEB is low when OENB_I was asserted low in the previous clock cycle, and the TSB is selected for a cell transfer" [34]. In FormalCheck, this property is described as follows.

```
Property: property_10
Type: Always
After: (@TSBselected) and (@RstDone) and
       (@CLKrising)
Always: ODAT_OEB = 0
Unless: (OENB_I =1)
```

6. Experimental Results

After completing the set-up of the proper environment and the specification of the properties, we verified both an abstracted model (with 8 PHY devices) and the original (with 32 PHY devices) TSB models. The experimental results are shown in Tables 2 and 3, respectively, including the reduction algorithm used, the status of the property verification, the number of reached states, the number of states in the model, the average state coverage, the CPU time (real time) in seconds, and the memory usage in megabytes. All experiments were conducted on a HP9000 (440MHz) with 6144 MB RAM and HP-UX11 operating system.

Tables 2 and 3 report that the verification of Property_1 and Property_6 were "Terminated". To discover the cause of the termination, these properties were decomposed into two sequential sub-properties each (e.g., Property_1A and Property_1B), which provide the same behavior as the original properties and could be proved independently [2].

During the verification process, we found several mismatches between the RTL implementation of TSB and the TSB specification with respect to the SCI-PHY protocol (e.g., Property_3). We hence suggested concrete modifications to the design and specification of the TSB which were considered acceptable by the designers, and the specification was revised to reflect these corrections. An example of such errors is given below (a full list of the uncovered errors can be found in [2]).

According to the SCI-PHY protocol: "To ensure backwards compatibility with single-PHY devices, the PHY for which a cell transfer is in progress shall not be polled until completion of the cell transfer" [5]. The TSB was basically designed to be polled while transmitting a cell, so naturally it is not compatible with single-PHY devices. In Receive Slave SCI-PHY mode the TSB relies on the master device. If the master device is not compatible with single-PHY devices and it polls the TSB during a cell transfer, the TSB will reply and will not care being backwards compatible with single-PHY devices.

Table 2: Experimental results of model checking of the abstracted (8 PHY) design

Properties	Reduction Algorithm	Status	States Reached	State Variables	State Var. Avg. Coverage	Real Time (seconds)	Memory Usage (MB)
Property_1	Iterated	Terminated	N/A	N/A	N/A	N/A	N/A
Property_1A	1-Step	Verified	5.09e+09	114	97.37%	493	34.61
Property_1B	1-Step	Verified	5.09e+09	114	97.37%	189	34.59
Property_2	1-Step	Verified	5.09e+09	114	97.81%	126	34.73
Property_3	1-Step	Failed	3.28e+08	115	95.22%	160	36.23
Property_4	1-Step	Verified	5.09e+09	114	97.81%	84	34.20
Property_5	1-Step	Verified	5.09e+09	115	97.39%	256	36.82
Property_6	1-Step	Terminated	N/A	N/A	N/A	N/A	N/A
Property_6A	1-Step	Verified	3.83e+08	114	98.25%	161	29.90
Property_6B	1-Step	Verified	3.83e+08	114	98.25%	256	29.51
Property_7	1-Step	Verified	5.09e+09	114	98.25%	208	34.74
Property_8	1-Step	Verified	5.09e+09	114	97.37%	34	5.27
Property_9	1-Step	Verified	5.45e+09	115	97.39%	23	5.27
Property_10	1-Step	Verified	4.61e+03	20	97.50%	18	21.68
Property_11	1-Step	Verified	5.14e+09	116	97.41%	136	34.63

Table 3: Experimental results of model checking of the full (32 PHY) design

Properties	Reduction Algorithm	Status	States Reached	State Variables	State Var. Avg. Coverage	Real Time (seconds)	Memory Usage (MB)
Property_1	Iterated	Terminated	N/A	N/A	N/A	N/A	N/A
Property_1A	Iterated	Verified	2.93e+18	195	97.95%	29	7.22
Property_1B	Iterated	Verified	2.94e+18	195	97.95%	31	7.22
Property_2	Iterated	Verified	4.12e+25	193	99.48%	21	7.21
Property_3	1-Step	Failed	1.76e+18	251	97.41%	10370	126.79
Property_4	Iterated	Verified	2.93e+18	196	98.21%	76	42.50
Property_5	Iterated	Verified	2.93e+18	197	97.97%	73	38.87
Property_6	Iterated	Terminated	N/A	N/A	N/A	N/A	N/A
Property_6A	Iterated	Verified	2.06e+17	195	98.46%	86	42.03
Property_6B	Iterated	Verified	2.06e+17	195	98.46%	92	43.16
Property_7	Iterated	Verified	2.93e+18	195	98.46%	73	42.50
Property_8	Iterated	Verified	2.93e+18	195	97.95%	72	37.79
Property_9	Iterated	Verified	3.09e+18	196	97.96%	69	38.33
Property_10	Iterated	Verified	2.93e+18	195	98.21%	75	38.96
Property_11	Iterated	Verified	2.94e+18	196	98.21%	380	38.90

7. Conclusions

In this paper, we investigated the formal verification of a SCI-PHY (Saturn Compatible Interface for ATM-PHY devices) Level 2 protocol engine, commercialized by PMC-Sierra, Inc. Using the FormalCheck tool, we first established a suitable verification environment, then defined a number of relevant liveness and safety properties covering essential behaviors of the SCI-PHY protocol. We checked these properties on an abstracted (8 PHY devices) as well as the original (32 PHY devices) hardware model. Throughout our verification efforts, we uncovered a number of mismatches between the hardware design in VHDL and the SCI-PHY protocol specification. Some of these are pertinent design bugs which were not caught during the simulation process.

Human time is a very important factor in formal verification. In this study as shown in Table 4, a lot of time was spent for understanding the specifications to define the right queries and also for understanding the implementation to be able to apply the features of FormalCheck like Reduction Seeds, Constraints and State Variables. Since the designer has a thorough knowledge of the design and the specification, assuming he/she is trained to use FormalCheck, it would take him/her only 3-4 weeks to define the queries and formally verify this design. This is much shorter in comparison to the 13 weeks spent for simulation.

Acknowledgments

This work was supported by an industrial research grant from PMC-Sierra Inc. We would like to thank PMC-Sierra colleagues for their cooperation.

References

[1] The ATM Forum Technical Committee, UTOPIA Level 2; Version 1.0, June 1995.

[2] L. Barakatain, Practical Approaches to Model Checking using FormalCheck. Ma.Sc. Thesis, Dept. of Electrical & Computer Engineering, Concordia University, April 2000.

[3] Cadence, Formal Verification Using Affirma FormalCheck. Manual, Version 2.3, October 1999.

[4] C. Kern and M. Greenstreet, "Formal Verification in Hardware Design: A Survey," ACM Transactions on Design Automation of E. Systems, Vol. 4, pp. 123-193, April 1999.

[5] PMC-Sierra, Inc., Saturn Compatible Interface Specification for PHY Layer and ATM Layer DEVICES, Level 2. Application Note, Issue 4: August 1997.

[6] PMC-Sierra, Inc.: SCI-PHY Transmit Master and Receive Slave TSB Specification; Issue 2, May 10, 1999.

Table 4: Detailed time frame for the design and verification project

Design and Verification Phases	Time (weeks)
Designing the RT level model	4
Writing test benches and simulation	13
Reading documents (SCI-PHY, UTOPIA, and TSB Spec.)	2
Reading the VHDL RTL code	3
Verifying FSM and Timing Diagrams	2
Defining and verifying properties on the complete TSB	5
Total time spent for model checking	12