Model Checking of the Transmit Master/Receive Slave (TMRS) using FormalCheck

Leila Barakatain and Sofiène Tahar

Technical Report

October 1999

Concordia University, ECE Dept., Montreal, Quebec, H3G 1M8 Canada
E-mail: {l_baraka, tahar}@ece.concordia.ca

Abstract. *In this paper we present our methods and results on formally verifying the implementation of the Receive Slave SCI-PHY mode of the Transmit Master/Receive Slave (TMRS) design using FormalCheck. TMRS is made of two major modules namely "Internal Register CBI interface" and "SCAN Interface". The SCAN module is responsible for the main functionality of the TMRS block, whereas the CBI module handles the configuration and the test route of it. We produced an abstracted model of the TMRS block. We then provided a number of relevant liveness and safety properties expressible in FormalCheck, and accomplished their verification on the abstracted model of the TMRS and also on its original model in reasonable CPU time. We found a number of mismatches between the RTL design and the SCI-PHY protocol as well as the TMRS specification. The experience on the verification of the TMRS demonstrates that FormalCheck is powerful enough to be used for verification of real size circuits.*

1 Introduction

In today's red-hot economy, staying on schedule to hit a narrow window of market opportunity often means the difference between product dominance and product death. With the volume and complexity of logic required to satisfy function-hungry consumers comes exponential growth in the difficulty of making sure it all works. Verification is on the critical path for today's IC designers, no matter what type of system they are building.

Formal methods have been around in academia for sometime [5]. However it is now becoming prevalent in the Electronic Design Automation (EDA) industry as a means to combat the explosive complexity of current Integrated Circuit (IC) designs. As an IC functionality increases linearly the amount of vectors required to fully testing this functionality increases exponentially. It therefore becomes impossible for a human being to fathom all the vectors required to fully test an IC. By using formal verification in parallel with the design efforts, the overall design cycle can be reduced [5].

The TMRS Telecom System Block (TSB) (designed by PMC-Sierra Inc.) [1], was chosen to be verified as a research case to experiment the benefits of formally verifying a design using FormalCheck [4] over using simulations to find bugs inside a design. The TMRS implements the output port of a cell interface. It can be configured to operate either as a bus master or bus slave. Also, the TMRS supports SCI-PHY (Saturn Compatible Interface for ATM-PHY devices) Level 2 [2] and ANY-PHY (proprietary protocol of PMC-Sierra) [1] protocols. SCI-PHY Level 2 protocol is a superset of UTOPIA (Universal Test & Operations PHY Interface for ATM) level 2 protocol [6].

In this paper we present our results of formally verifying the TMRS block when it is in the Receive Slave SCI-PHY mode of operation. The rest of the paper is structured as following: In Section 2, we describe the behavior of the TMRS block and its interface signals in Receive Slave SCI-PHY mode. In Section 3, we show the methodology and results of the property checking using FormalCheck and about the errors we found in the specification of the TMRS. We conclude the paper in Section 5.

2 The TMRS Telecom System Block

The TMRS [1] implements the output port of a cell interface. It can output the cell data either in 8-bit or 16-bit wide format at clock rates up to 52 MHz. Data transfers are cell-based, that is an entire cell is transferred to one PHY device before another is selected. It outputs cells on SCI-PHY/ANY-PHY compatible interface.

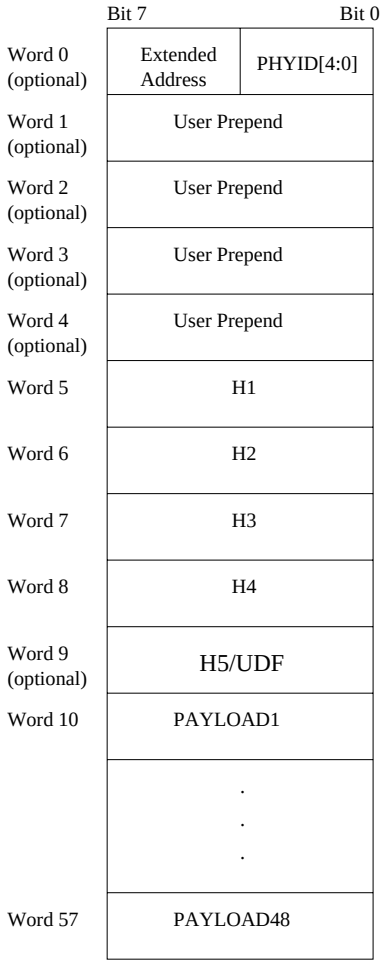

Figure 1: 8-bit SCI-PHY Cell Format

The 8-bit wide, variable data structure in 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 [2].

The TMRS block 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 ATM (PHY) device. Each logical channel corresponds to a FIFO channel in the external FIFO. When the TMRS 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.

2.1 The TMRS Icon and Pin Description

The Icon of the TMRS is presented in Figure 2. In this study only the control part of TMRS (not the data-path) in Receive-Slave SCI-PHY mode is of interest, therefore we only describe the control pins used in this mode of operation.

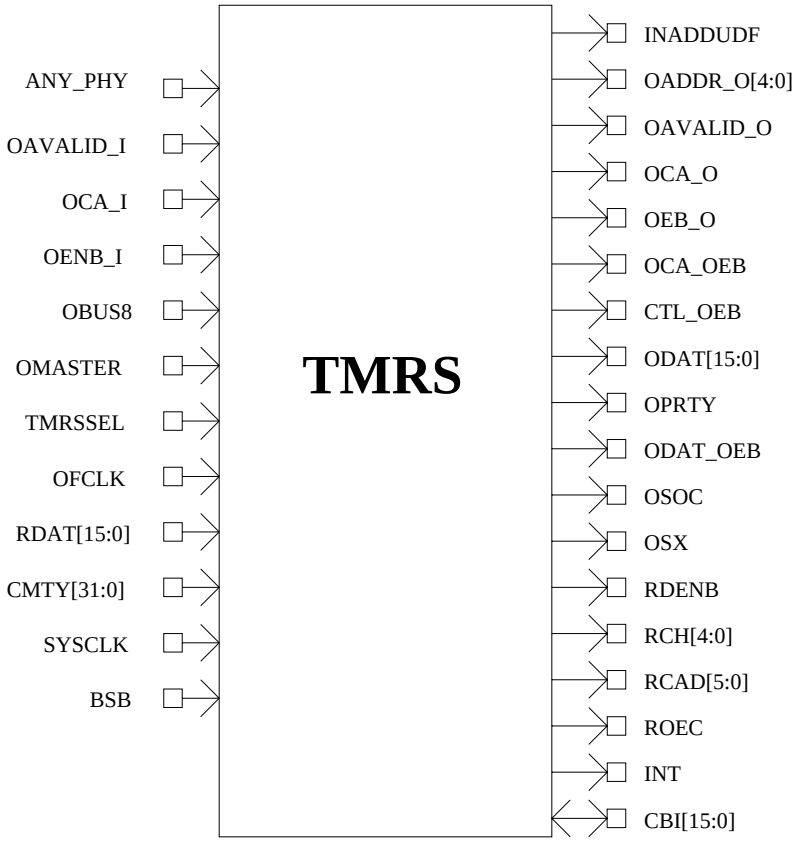

Figure 2: TMRS Icon

OMASTER: The Output Port Master Select (OMASTER) pin determines the direction of the output port control signals. When OMASTER is low, the output port of the device in which the TMRS is integrated is a bus slave and complies with the SCI-PHY or ANY-PHY receive protocol. The TMRSSEL, OAVALID_I, OENB_I, OCA_O, and OCA_OEB signals are used.

ANY_PHY: When set to low, the TMRS supports SCI-PHY (UTOPIA Level 2) protocol, otherwise it supports ANY-PHY protocol.

TMRSSEL: The TMRS Select (TMRSSEL) pin selects the TMRS for polling and cell transfer operations. This pin is only used when the TMRS is configured as a slave, i.e., OMASTER is low. If the TMRSSEL is sampled high when OAVALID_I is high, the TMRS is enabled for polling. The TMRS is selected for a cell transfer when TMRSSEL and OENB_I were sampled high in the previous clock cycle and OENB_I is sampled low in the current clock cycle. The cell transfer is performed while OENB_I is maintained low.

OAVALID_I: The Output Port Address Valid (OAVALID_I) output indicates that the bus master is asserting a valid PHY address for polling purposes. In SCI-PHY mode, OAVALID_I is active high. In slave mode (OMASTER is low), the OCA_O output is driven on the clock cycle following OAVALID_I and TMRSSEL being sampled high. If OAVALID_I is sampled low, OCA_O becomes high impedance. The TMRS supports polling in contiguous cycles if OAVALID_I is held high.

OENB_I: The active low Output port Enable (OENB_I) input is used to initiate the transfer of cells from the cell output port to a PHY device when TMRS is in Slave mode. In SCI-PHY mode, when OENB_I is sampled low and the TMRS is selected, a word is output on bus ODAT[15:0] on the following first clock cycle. The selection occurs when OENB_I and TMRSSEL were last sampled high. To transfer a complete cell OENB_I has to be maintained low for the period of cell transfer. The period varies depending on the bus width and cell format. OENB_I can be deasserted any time during the cell transfer to perform word level flow control, but it is assumed that no other devices will be selected when OENB_I is reasserted.

OCA_O: The Output Cell Available (OCA_O) output provides cell level flow control. OCA_O is only used if the MASTER input is low (Slave mode). The TMRS indicates the presence of a cell to transfer via the OCA_O output. When OAVALID_I and TMRSSEL are asserted, OCA_O is driven to high if at least one line of the CMTY[31:0] status bus is low and the prefetch of the next FIFO is finished. In SCI-PHY mode, the OCA_O assertion occurs on the first clock cycle following OAVALID_I and TMRSSEL assertion. If all CMTY[31:0] lines are high or the prefetch of the next FIFO is not finished, OCA_O is deasserted. Note that the CMTY line corresponding to the FIFO of the current transfer is always masked (i.e., always considered to be set to high).

OCA_OEB: The Output Cell Available Output Enable (OCA_OEB) is an active low signal intended to be wired directly to an active low pad output enable. It is used to control the direction of the OCA pin at the device level. In Slave mode (OMASTER low), OCA_OEN becomes low if the TMRS is selected for polling; otherwise, OCA_OEB is high.

OBUS8: The Output port Bus width select (OBUS8) selects the interface bus width. When OBUS8 is high, only ODAT[7:0] present valid data and ODAT[15:8] are forced low. When OBUS8 is low, all ODAT[15:0] inputs are used.

OFCLK: The Output FIFO Clock (OFCLK) is used to read words from the TMRS cell output port. OFCLK must cycle at a 52 MHz or lower instantaneous rate. All output port SCI-PHY bus timing is relative to the rising edge of OFCLK.

CMTY[31:0]: The Channel Empty bus (CMTY[31:0]) input reports the cell availability of external FIFOs or of an external FIFO channel. A logic low on a line of the CMTY[31:0] indicates the corresponding channel of the FIFO has at least one complete cell available to transfer. CMTY[31:0] is sampled on the rising edge of OFCLK.

SYSCLK: The System Clock (SYSCLK) input is used to synchronize the FIFO interface. SYSCLK must cycle at a 52 MHz or lower instantaneous rate.

OSOC: The Output Start Of Cell (OSOC) marks the start of the cell on the ODAT[15:0] bus. When operating in SCI-PHY mode, OSOC assertion indicates that the first word of the cell structure is present on the ODAT[15:0] stream. When operating in Slave mode, OSOC is valid when OENB_I was asserted in the previous cycle.

ODAT_OEB: The Output Data Enable (ODAT_OEB) is an active low signal intended to be wired directly to an active low pad output enable for the ODAT[15:0] bus, OSOC and OPRTY signals.

PRELEN[1:0]: The Prepend Length (PRELEN[1:0]) bits determine the number of prepended bytes or words in each cell. The prepended bytes/words are added in addition to the address prepend used in Slave or ANY-PHY mode. When OBUS8 is high, PRELEN can have the values 0, 1, 2, and 4, which is the number of prepended bytes. When OBUS8 is low, PRELEN can have values 0, 1, and 2, which is the number of prepended words.

HECUDF: The HECUDF bit determines whether or not the HEC/UDF octets are included in the cell received from the FIFO. When set to high (the default), the HEC and UDF octets are included. When set to low, the third word of the 27-word ATM cell is omitted and a 26-word cell (plus appended words) is transferred. If OBUS8 is high, the fifth byte of the 53-octet ATM cell is omitted and a 52-byte cell (plus appended bytes) is transferred.

INADDUDF: The INADDUDF output indicates the state of the INADDUDF bit in the configuration and status register. This bit relocates the word identifying the logical channel in the H5/UDF field when operating in SCI-PHY slave mode. When this bit is set to high and the 8-bit cell format is used, the logical channel is identified in the H5 byte (Figure 1).

2.2 Block Diagram of the TMRS

The TMRS consists of two major blocks, the CBI block and SCAN block (Figure 3). It consists of about 7500 gates. The CBI block is used only for the configuration and the test interface and the SCAN block drives the main functionality of the TMRS. The SCAN block of TMRS 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 TMRS 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 have 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.

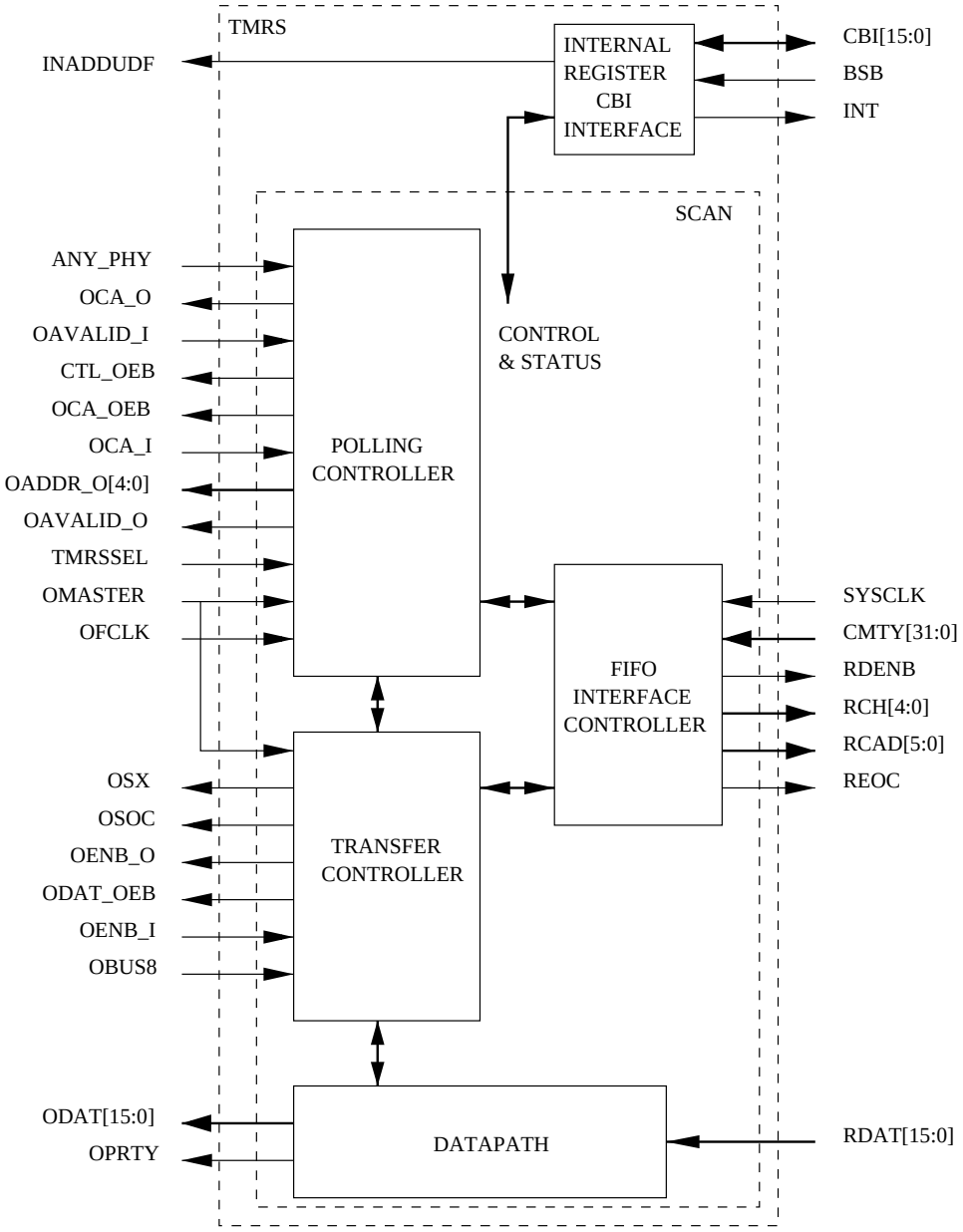

Figure 3: Block diagram of the TMRS

2.3 SCI-PHY Receive Slave Operation

When operating in Receive Slave SCI-PHY mode, the TMRS works as a single PHY. The TMRS will respond to a polling request when OAVALID_I and TMRSSEL pins are asserted. The TMRS autonomously reflects the cell availability of all FIFO channels as indicated by the CMTY[31:0] input bus. To compensate for the PMCF latency, the TMRS prefetches the first five bytes/words of the next non-empty FIFO before the next transfer occurs. So if it cannot finish the prefetch cycle before the end of the current transfer, OCA_O will not be asserted upon a polling request, even if some FIFOs are non-empty. When a polling request occurs simultaneously with a cell being transferred, the TMRS will mask the CMTY line associated with the FIFO of the current transfer. Upon a polling selection, OCA_OEB signal is synchronous to the OCA_O.

A cell transfer is affected by the assertion of OENB_I. The TMRS is selected for transfer when the TMRSSEL input is high on the clock cycle prior to OENB_I assertion. The first word of the cell data received from the FIFO identifies the logical channel. Alternatively if INADDUDF field in the control register is set to one, the word identifying the FIFO channel is placed in the H5/UDF field (Figure 1). Upon selection, the TMRS transfers a cell from the next available FIFO channel. The TMRS services non-empty FIFO channels in a round robin fashion.

3 Model Checking of the TMRS Based on SCI-PHY Level 2 Protocol

The SCI-PHY protocol was defined within the SATURN Group [2] as a standardized cell-based interface between ATM layer and PHY layer devices to support single-PHY and multi-PHY applications. 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 [2].

To be able to match the TMRS interface signals with SCI-PHY Level 2 interface signals, we based our verifications on the following environment assumptions.

1. The interface signal RCA in SCI-PHY is assumed to be the output of a buffer which its inputs are OCA_O and OCA_OEB, two interface signals of the TMRS. It is assumed that the OCA_O signal is the input of the buffer and the OCA_OEB is an active low enable signal of that buffer, as shown in Figure 4.

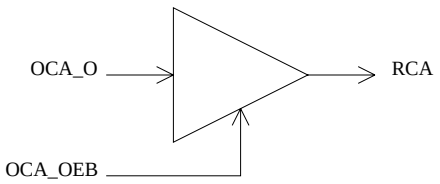

Figure 4: The OCA_O and OCA_OEB signals in the TMRS, and RCA signal in SCI-PHY protocol

2. TMRSSEL input of the TMRS is assumed to be one of 32 output bits of a 5x32 decoder, which decodes RADDR[4:0] of SCI-PHY interface signal, as shown in Figure 5.

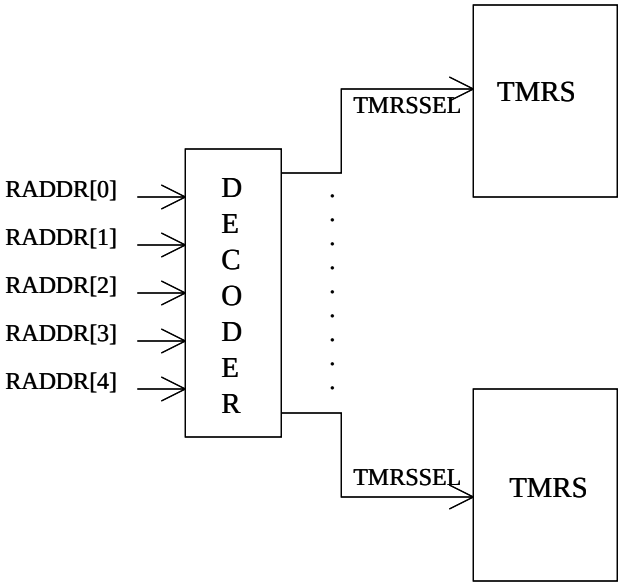

Figure 5: RADDR[4:0] bus to TMRSSEL signal

Table 1 gives the name cross-reference between the TMRS pins when in Receive Slave SCI-PHY mode, and Receive interface signals in SCI-PHY protocol. Other TMRS interface signals are set to specific values to work in Receive Slave SCI-PHY mode. They are explained in Section 3.1.3.

Table 1: SCI-PHY Signal Cross-Reference

SCI-PHY Receive	TMRS Receive Slave
RFCLK	OFCLK
RRDENB	OENB_I
RSOC	OSOC
RCA	OCA_O, OCA_OEB
RDAT[15:0]	ODAT[15:0]
RxPRTY	OPRTY
RADDR[4:0]	TMRSSEL
RVALID	OAVALID_I

3.1 Environment for The SCAN Block

State space explosion is a well-known problem in FSM-based verification approaches [5]. In digital hardware designs, the state space increases exponentially with the number of latches of a design. Because model-checking algorithms are based on state space exploration, their efficiency is also limited by this phenomenon. In this section, we describe our methodology and results to deal with the state space explosion we faced in the property checking.

The SCAN block of TMRS consists of four main blocks (Figure 3), 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. In order to make the proper environment for these three blocks and also avoid the state space explosion, we used the following techniques: model abstraction and reduction, tool guided reduction, and tool guided *Constraints* in FormalCheck.

3.1.1 Model Abstraction and Reduction

As we explained before, we are only interested in three control blocks of TMRS. To reduce the state space and speed up the verification, we tried to trim the TMRS block 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:

- Isolated the SCAN block of the TMRS. Hence, we eliminated the CBI block used for test purposes.
- 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.
- Removed the DATAPATH module from the SCAN block.
- Made an abstracted model of the SCAN block to support only 8 PHY devices, to reduce the verification time and also to reduce the amount of memory used by FormalCheck. Once we succeeded verifying the properties for the abstracted model, we verified the same properties on the model of TMRS which supports 32 PHY devices.

3.1.2 Tool Guided Reduction

In FormalCheck there are two reduction algorithms, the *1-Step algorithm* and *Iterated algorithm*. The *1-Step algorithm*, which is the default reduction algorithm in FormalCheck, performs a single reduction and then verifies the query (a query consists of a property/properties and *Constraints*). This algorithm looks at the dependency graph and removes any portion of the design from the verification proof that cannot affect the outcome of the query. The *Iterated algorithm* iteratively reduces the design model during the verification process. This algorithm takes an attempt to find a small portion of the design that can be used to verify the current query. This technique guarantees that queries proven to be true on the small portion of the design would also be true for the entire design. The *Iterated* algorithm can be run with or without a user-supplied starting point (*Reduction Seed*) and can reduce the state space of the design by several orders of magnitude. The *1-Step algorithm* performs the reduction quicker than the *Iterated algorithm*, but does not reduce the design model as much. For this reason, the *Iterated algorithm* is better suited to larger models where there is a risk to run out of memory during verification[4].

In our design, we used the *Iterated algorithm* for the properties which were consuming too much of memory. Also to reduce the memory usage even more in those properties, we introduced *Reduction Seeds* with the *Start as Input* option [4]. By defining some of the signals as *Reduction Seeds*, we are reducing the explored state space for model checking and hence improving performance. Assigning *Start as Input* reduction value to an element will initially free it. It starts as a primary input but FormalCheck can make it active again if it is determined that the logic driving the design element cannot be pruned [4].

We used the *Iterated* reduction algorithm in property_1, property_2, property_4, property_5, and property_6 Section 3.3. We will explain the *Reduction Seeds* used for each one of these properties in Section 3.3.

3.1.3 Tool Guided Constraints

In FormalCheck a query is made up of both properties and *Constraints*, which the *Constraints* establish the operating environment for the design. In formal verification all reachable states are explored. 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.

In our experiment, we defined the following *Constraints* as default on all of the properties used to verify the TMRS:

- A clock *Constraint* on OFCLK signal, starting with high for 1 crank and then low for 1 crank (In FormalCheck, a crank is considered as the propagation delay of a flip-flop. Since there is no concept of the absolute time, FormalCheck uses a crank as the unit of time and observes the events relative to this unit).
- A *Clock Constraint* on SYSCLK signal, the same as OFCLK.
- A *Reset Constraint* on RST signal, starting with high for 2 cranks and then goes to low forever.
- A *Group Constraint* named SCIPHY_Slave. This group is used to drive the TMRS to SCI-PHY slave mode. The two signals under this group, ANYPHY and OM ASTER signals, are set to low.
- A *Group Constraint* named Default_Mode. This group is used to put TMRS in a default mode of operation. The signals used in this group are:
  1. HECUDF set to high.
  2. INADDUDF set to low.
  3. OBUS8 set to high.
  4. PHYDEV set to 31 (set to 7 for the abstracted model of the TMRS).
  5. PRELEN set to “00”.
- A *Group Constraint* named STABLE_INPUTS. This *Constraint* was defined to create a suitable environment for the TMRS. By adding this *Constraint* the input signals to TMRS will stay stable for the whole duration of the OFCLK and they only change with the rising edge of OFCLK. The signals included in this group are as following:
  1. CMTY[31:0] (or CMTY[7:0] in the abstracted model of the TMRS)
  2. OENB_I input signal
  3. TMRSSEL input signal
  4. OAVALID_I. We have defined two *Constraint*s on this signal: (1) OAVALID_I has to be stable during one clock cycle to meet the required timing limits. (2) Based on SCI-PHY Level 2 standard, “*To avoid contention while polling, the ATM layer device shall present a valid address no more than once every other RFCLK period. It is recommended that RADDR[4:0] be set to 0x1F when invalid. When supporting 32 PHY links, RAVALID is asserted simultaneously with a valid address on RADDR[4:0]*” [2]. That means in the TMRS, the OAVALID_I signal should not stay high for two consecutive clock cycles. This is the second *Constraint* for this signal.

3.2 Expression Macros

Often, the same complex expression is needed in several behaviors. FormalCheck supports expression reuse through Macros. Any expression that is given a user supplied name becomes a Macro and is globally available. We created some expression Macros inside FormalCheck to make the properties more comprehensive. The following lists the Macro names and also their contents. The "@" sign in front of the Macro shows that this is a Macro not a signal.

1. *@CLKrising:* scan:OFCLK = rising
2. *@FIFOnotEmpty:* scan:Polling_Sm_Inst:Allfifoempty = 0
3. *@PrefetchDone:* scan:Transfer_SM_Inst:Prefetch_Not_Rdy = 0
4. *@RCAhigh:* (scan:Oca_O = 1) and (scan:Oca_Oeb = 0)
5. *@RstDone:* scan:Rst = 0
6. *@TMRSpolled:* (scan:Tmrssel = 1) and (scan:Oavalid_I = 1)
7. *@TMRSselected*: (scan:Oenb_I = 0) and (scan:Transfer_Sm_Inst:Oenb_I_Reg = 1) and (scan:Tmrssel_Reg = 1)
8. *@TMRSselectedLastCC*:((scan:Transfer_Sm_Inst:Oenb_I_Reg2 = 1) and (scan:Tmrssel_Reg2 = 1)) and (scan:Oenb_I_Reg = 0)
9. *@TransferDone*: (scan:Transfer_Sm_Inst:Rec_Cpt = scan:Transfer_Sm_Inst:Cell_Length)
10. *@WaitSelectState:* scan:Transfer_Sm_Inst:Slave_State=Sms_Wait_Sel
11. *@StartTxState:* scan:Transfer_Sm_Inst:Slave_State=Sms_Start_Transf
12. @Latch4State: scan:Transfer_Sm_Inst:Slave_State=Sms_Latch4

3.3 Properties

After establishing a proper environment, we consider 7 properties of the TMRS, including liveness and safety properties. The following properties have been defined based on either the SCI-PHY Level 2 protocol or the specification of the TMRS.

Property_1: According to the SCI-PHY protocol: "When RRDENB is sampled low by the PHY layer device, the RSOC signal will be accepted by the ATM layer device on the next rising edge of RFCLK" [2]. We expressed this property for the TMRS as following: "If OENB_I is sampled low by the PHY layer device, and the TMRS was selected (TMRSSEL and OENB_I signals were sampled high) in the previous clock cycle, on the next rising edge of OFCLK the transmission of a full cell will start and the OSOC signal will be set to high". In FormalCheck, this property is expressed as follows.

```
Property: property_1
Type: Always
After: (@TMRSselected) and (@WaitSelected) and (@CLKrising)
Always: OSOC = 1 and @StartTxState
Options: Fulfill Delay: 0  Duration: 1  Count of @CLKrising
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Fifo_Ctl_Inst
```

Tables 2 and 3 report that the verification of Property_1 was “Terminated”. Based on the specification of the TMRS “the START_TRANSF state, starts the transfer by asserting OSOC signal”, therefore, Property_1 expects the FSM to move to START_TRANSF state and the OSOC signal to be set to high in the same clock cycle. By examining the “verify.out” file, it was observed that no transition was enabled for this property during the verification process. This means the enabling condition for this property was irrelevant to the condition to be fulfilled. The Transfer_Slave state machine is actually made of two parallel processes (state machines), which one of them generates the OSOC as well as other control signals and the other one determines the state of the Transfer_Slave state machine in the next clock cycle. To discover the cause of termination, this property was decomposed to the following two properties, Property_1A, which checks for the state transition to START_TRANSF state, and Property_1B, which checks for the assertion of the OSOC signal. These two properties are stated as following.

Property_1A: If the TMRS is in the WAIT_SEL state and the TMRS was selected in the previous clock cycle (TMRSSEL_REG_2 and OENB_I_REG2 signals are sampled high), and OENB_I signal was sampled low in the previous clock cycle (OENB_I_REG is low), on the next rising edge of OFCLK the TMRS will be in START_TRANSF state. In FormalCheck this property is expressed as follows.

```
Property: property_1A
Type: Always
After: (@TMRSselectedLastCC) and (@WaitSelected) and (@CLKrising)
Always: @StartTxState
Options: Fulfill Delay: 0  Duration: 1  Count of @CLKrising
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Fifo_Ctl_Inst
```

Property_1B: If the TMRS is in the LATCH4 or WAIT_SEL states and OENB_I signal is sampled low and the TMRS is selected (TMRSSEL_REG and OENB_I_REG signals are sampled high), on the next rising edge of OFCLK the transmission of a full cell will start (the OSOC signal will be set to high) and the TMRS will be in WAIT_SEL state. In FormalCheck, this property is expressed as follows.

```
Property: property_1B
Type: Always
After: (@TMRSselected) and (@Latch4State or @WaitSelected) and (@CLKrising)
Always: Osoc = 1 and @WaitSelectState
Options: Fulfill Delay: 0  Duration: 1  Count of @CLKrising
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Fifo_Ctl_Inst
```

Verification of Property_1A and Property_1B revealed the origin of the problem, which caused terminating the verification of Property_1. The Transfer_Slave state machine proceeds to START_TRANSF state one clock cycle after the OSOC signal (Output Start Of Cell) is asserted. Therefore, the OSOC signal and START_TRANSF state are not synchronous.

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” [2]. For TMRS, this property is expressed as following: “When OAVALID_I and TMRSSEL are asserted (TMRS 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” [1]. In FormalCheck, this property is expressed as follows.

```
Property: property_2
Type: Always
After: (@TMRSpolled) and (Polling_Sm_Inst:Allfifoempty = 1) and(@RstDone) and
         (@CLKrising)
Always: (not @RCAhigh)
Options: (None)
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Transfer_Sm_Inst
```

Property_3: In Receive Slave SCI-PHY back to back transfer mode, when the external master device does not deassert OENB_I at the end of a transfer, the TMRS remains selected for another cell transfer. If all FIFOs are empty the TMRS will deassert ODAT_OEB and will wait to be reselected [1]. In FormalCheck, this property is expressed as follows.

```
Property: property_3
Type: Always
After: (@TransferDone) and (Oenb_I = 0) and (Polling_Sm_Inst: Allfifoempty = 1)
         and (@RstDone)
Always: Odat_Oeb = 1
Options: Fulfill Delay: 0 Duration 1 count of @CLKrising
```

Reduction options:

```
Reduction Technique: 1-Step
Reduction Seed: Empty
```

It is observed from Tables 2 and 3, that the verification of Property_3 failed in both models. This means even after a cell transfer is done and there are no more cells to transfer, ODAT_OEB will still be asserted. This problem was taken into consideration by the designer of the TMRS and was fixed. In the new version of the design, when in the back-to-back transfer mode, after completing a cell transfer the ODAT_OEB will stay asserted and the data on the ODAT bus will be the byte/ word which was transferred last. The master is capable of determining the end of a cell (by counting the number of bytes/words it has received), therefore, the master waits for the assertion of the OSOC signal (from the TMRS).

Property_4: No PHY shall drive RCA upon sampling RAVALID low [2]. In FormalCheck, this property is expressed as follows.

```
Property: property_4
Type: Never
After: (Oavalid_I = 0) and (@RstDone) and (@CLKrising)
Never: @RCAhigh
Options: (None)
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Fifo_Ctl_Inst
```

Property_5: 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” [2]. In TMRS this property is expressed as following: “When Oavalid_I and TMRSSEL are asserted, OCA_O is driven to logic 1 in the next clock cycle, if at least one of the CMTY[31:0] status bus is low and the prefetch of the next FIFO is finished” [1]. In FormalCheck, this property is expressed as follows.

```
Property: property_5
Type: Always
After: (@TMRSpolled) and (@FIFOnotEmpty) and (@PrefetchDone) and (@CLKrising) and
          (@RstDone)
Always: @RCAhigh
Options: Fulfill Delay: 0 Duration: 1 Count of @CLKrising
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Fifo_Ctl_Inst
```

Property_6: In Receive Slave SCI-PHY back to back transfer, when the external master does not deassert OENB_I at the end of a transfer, and at least one of the FIFOs have a cell to transfer, the TMRS will eventually start transmitting the new cell.

We acquired from Property_1, that the OSOC signal and START_TRANSF state of the Transfer_Slave state machine are not synchronous. Accordingly, we had to decompose Property_6 to two properties to be able to check for both the OSOC signal and the START_TRANSF state of the FSM, while in back-to-back transfer mode. The queries of these two properties contain another Constraint, named BackToBackTx, in addition to the default Constraints explained in Section 3.1.3. This Constraint puts the TMRS in the back-to-back transfer mode. Also to make matters easier, we assumed no interruptions from the master will occur while transferring a cell, as well as after the completion of a cell transfer (since a cell transfer with interruptions from the master was verified through other queries, this assumption is considered safe). The BackToBackTx Constraint is expressed in FormalCheck as follows.

```
Constraint: BackToBackTx
Type: Always
After: (@IncCptState) and (@CLKrising)
Assume Always: Oenb_I = 0
Unless: (@StartTxState) or (@WaitSelectState)
Options: (None)
```

In FormalCheck, Propery_6A and Property_6B are expressed as following.

Property_6A:

```
Property: property_6A
Type: Eventually
After: (@TransferDone) and (Transfer_Sm_Inst:New_Cell_Transf_Rdy = 1) and
       (@Clkrising)
Eventually: @StartTxState
Options: (None)
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as Input: Fifi_Ctl_Inst
```

Property_6B:

```
Property: property_6B
Type: Eventually
After: (@TransferDone) and (Transfer_Sm_Inst:New_Cell_Transf_Rdy = 1) and
       (@Clkrising)
Eventually: (Osoc = 1) and (@LatchNext2State)
Options: (None)
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as Input: Fifi_Ctl_Inst
```

Property_7: According to SCI-PHY protocol: “The ATM layer device may pause the transfer at any time by deasserting RRDENB” [2]. In TMRS this property is expressed as following: “The Slave Transfer State Machine always expects a complete cell transfer but supports transfer interruption by deassertion of Oenb_I” [1]. In FormalCheck, this property is expressed 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)
Options: (None)
```

Reduction options:

```
Reduction Technique: Iterated
Reduction Seed: New (User defined seed)
Start as input: Fifo_Ctl_Inst and Polling_Sm
```

4 Experimental Results

All verifications in this paper were executed on a HP9000 (440MHz) with 6144 MB RAM and HP-UX11 operating system. For the rest of this paper, we will call the model of the TMRS which supports 32-PHY devices, the original model, and the model of the TMRS which supports 8 PHY devices, the abstracted model to simplify the text. As mentioned in Section 3.1.1, we first designed an abstracted model of the TMRS (which supports 8 PHY devices) and verified it. After completing the verification of the abstracted model of the TMRS, we verified the original model of the TMRS using reduction techniques. The experimental results are shown in Tables 2 and 3, respectively. These tables include 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.

Table 2: Verification results of model checking on the abstracted model of TMRS using FormalCheck

Properties	Reduction Algorithm	Status	States Reached	State Variables	State Var. Avg. Coverage	Real Time (seconds)	Memory Usage (MB)
Property_1	1-Step	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_6A	1-Step	Verified	3.83e+08	114	98.28%	161	29.90
Property_6B	1-Step	Verified	3.83e+08	114	98.28%	256	29.51
Property_7	1-Step	Verified	5.09e+09	114	98.28%	208	34.74

Table 3: Verification results of model checking on the original model of TMRS using FormalCheck

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	3.50e+17	150	99.00%	39	32.51
Property_1B	Iterated	Verified	3.50e+17	150	99.00%	38	31.97
Property_2	Iterated	Verified	6.01e+10	41	98.78%	16	25.18
Property_3	1-Step	Failed	2.07e+17	196	97.96%	15685	213.77
Property_4	Iterated	Verified	3.50e+17	150	99.33%	63	33.01
Property_5	Iterated	Verified	3.50e+17	151	99.01%	48	30.91
Property_6A	Iterated	Verified	2.57e+16	150	99.67%	40	32.49
Property_6B	Iterated	Verified	2.57e+16	150	99.67%	44	30.46
Property_7	Iterated	Verified	1.76e+14	76	100.00%	22	6.85

We can see in Tables 2 and 3, the 1-Step reduction algorithm was used for the verification of Property_3 in both models. The reason for that is, when the Iterated algorithm was used the result of the verification was “Terminated”, but with 1-Step algorithm the verification would finish with no problem. This shows that the Iterated algorithm is not powerful enough to reduce the model efficiently for all kinds of queries, and in that case the user has no choice but using the 1-Step algorithm. It must be added here that, if a query is verified using the Iterated algorithm, the verification result is guaranteed to be valid.

By comparing the results of Property_2 for the abstracted model and the original model, we notice that the number of state variables, memory usage, and the real time for the original model is much less than the abstracted model! The reason for this difference is that FormalCheck used Autocheck Restrictions on the RDAT[15:0], CMTY[31:0], and PTYP inputs of the original model and not for the abstracted model. By doing that, the number of state variables for the original model was automatically reduced, and as a result the memory usage and the real time was reduced as well.

Properties 1A, 1B and 5 consume less CPU time and memory for the original model compared to the abstracted model. The reason for that is, we used the *Iterated* algorithm and *Reduction Seed* for the original model, whereas for the abstracted model the default (*1-Step* algorithm) was used.

By looking at the verification results in Tables 2 and 3, we can see that the reduction algorithm used for each property is an important key to reduce the CPU time and the memory usage. Also we can see that if the properties are verified under the same conditions, by increasing the number of PHY devices supported by the TMRS, the number of state variables will increase. The CPU time for property checking is related to different parameters such as memory usage, number of state variables, and the reduction method used for that specific property. Feedbacks from the internal blocks can increase the CPU time also.

4.1 Further Errors Found

In our study and during the verification process, we found several errors in the specification of the TMRS. These errors are not only related to the Receive Slave SCI-PHY mode of TMRS, but also to the other modes such as Transmit Master for SCI-PHY/ANY-PHY and also Receive Slave

ANY-PHY modes of operation. These error were found by property checking and also by code inspection. The errors are listed as following:

- The following fundamental mismatch was found between the specification of the TMRS [1] and the SCI-PHY protocol [2].

  According to 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*" [2]. The TMRS 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 TMRS relies on the master device. If the master device is not compatible with single-PHY devices and it polls the TMRS during a cell transfer, the TMRS will reply and it will not care about being backwards compatible with single-PHY devices.

- In the Transmit Master SCI-PHY/ANY-PHY mode we found the following error in the specification of the TMRS [1].

  The Master Transfer state machine is missing the "Prefetch #3 State" and "Prefetch_Next_FIFO 3" states.

- In the Receive Slave SCI-PHY/ANY-PHY mode we found the following eight errors in the specification of the TMRS.

  - The Slave Transfer state machine is missing "Latch_3", "Latch_4", and "Prefetch_Next_FIFO _3" states.
  - The Slave Transfer state machine shows while in "Empty_Pipe" state,

    `if the new_cell_transfer_rdy = 0 => Next_State = Load_Pipe_Cache_Pre_State`

    whereas in the TMRS design we have,

    `if the new_cell_transfer_rdy = 1 => Next_State = Load_Pipe_Cache_Pre_State`
  - The Slave Transfer state machine shows while in "Empty_Pipe" state,

    `if transfer_end = 0 => Next_State = Wait_State`

    whereas in the TMRS design we have,

    `if transfer_end = 1 => Next_State = Wait_State`
  - The Slave Transfer timing diagram for SCI-PHY and ANY-PHY are missing "Pre #3", "Latch #3", and "Latch #4" states.
  - The Slave Transfer state machine shows, while in "Start_Trans" state,

    `if ANY-PHY = 0 and Transf_En = 1 => Next_State = INC_CPT_State`

    but this FSM does not specify the case of `ANY-PHY = 1`.
  - The Slave Transfer timing diagram (ANY-PHY) is showing "Asser_OSOC" state, which does not exist in the TMRS design.
  - The Slave Transfer timing diagram (ANY-PHY) shows that the OSX signal is asserted while in "Start_Transfer" state. In reality the OSX signal is asserted while in "Wait_Sel" state.
  - The Slave Transfer timing diagram (ANY-PHY) shows that the OSOC signal is asserted while in "Asser_OSOC" state. In reality this signal will be asserted while in "Start_Transfer" state.

- In the Icon specification of the TMRS [1], we found the following errors:

  - The CBI[15:0] bus is shown as an input bus, whereas it is an I/O bus.
  - ODAT[15:0] data bus is shown as I/O bus, whereas it is actually an output data bus.
  - Signals SCAN_IN and SCAN_EN are shown as input signals to the SCAN block and also SCAN_OUT is shown as an output signal from this block. These signals are defined in the code of the TMRS, but they have never been used inside the SCAN block.

All of the suggested modifications over the specification of the TMRS, which were specified in this section, were considered acceptable by the designer of the TMRS, and the specification was revised to reflect these corrections.

5 Conclusions

In this study, we explored model checking for the Receive Slave SCI-PHY mode of operation of the TMRS TBS. The main contributions of this work are (1) the establishment of the abstracted model of the TMRS, (2) the definition of a suitable environment with FormalCheck for the TMRS interface signals, (3) the definition of a set of properties on the TMRS in FormalCheck, (4) the application of reduction techniques and reduction seeds to the model to reduce the state space, (5) the verification of the original model of the TMRS, (6) the discovery of several mismatches between the TMRS design, its specification, and the SCI-PHY protocol, and (7) the uncovering of some errors in the specification of the TMRS.

FormalCheck provides various algorithms to perform verification, such as “Symbolic State Enumeration” (using ordered Binary Decision Diagrams or BDDs), “Explicit State Enumeration” and “Auto Restrict” options [4]. To verify large circuits and to avoid the state space explosion there are several techniques to use: (1) choosing the suitable run option can reduce the run time when verifying large circuits, (2) using a suitable reduction method and reduction seed in FormalCheck can reduce the state space. The only drawback to using the reduction seed is that the person who is verifying the design has to know the design well enough to be able to introduce reduction seeds. The danger behind over constraining a design is false verification results (3) If the reduction techniques fail in reducing the state space, an abstracted model of the circuit needs to be developed.

Using the Iterated reduction algorithm in FormalCheck does indeed make the verification much faster than the 1-Step algorithm and the verification results are guaranteed to be valid for the entire design. The only problem with this algorithm is that it does not seem to be able to reduce the design efficiently for all kinds of queries. Using this algorithm for some queries can cause the verification result to be “Terminated”, whereas using the 1-Step algorithm, even though in a longer time, will proceed to complete the verification of the same queries (provided having enough resources to avoid state space explosion).

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 a lot of time was spent 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 2-3 weeks to define the queries and formally verify this design

.

Table 4: Detailed time frame of verification of the TMRS

Design and Verification phases	Time (week)
Designing the RT level model	4
Writing test benches and simulation	13
Reading documents (SCI-PHY, UTOPIA, and TMRS Spec.) for model checking purposes	2.5
Reading the code for model checking purposes	3
Verifying FSM and Timing Diagrams	2
Making the abstract model of TMRS	0.5
Defining properties and verifying the abstract model	3
Verifying the original model	2
Total time spent for model checking	13

The “Back Reference” feature in FormalCheck can help the designer debug the problems found during verification. The trick to finding these problems is to isolate which sequence of events caused the erroneous behavior. This is time-consuming with simulation, because it is difficult for designers to sort through all the events and determine which ones matter. FormalCheck can automatically find a minimal sequence of events that cause the error, when a user asserts a property that expresses that the error condition should never occur.

Acknowledgments

This work was supported by an industrial research grant from PMC-Sierra Inc. We would like to thank PMC-Sierra colleagues whose cooperation contributed to the maturity of this work. In particular special thanks go to Jean Lamarche and Jean-Marc Gendreau for many (and lengthy) discussions on the TMRS block and its verification.

References

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

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

[3] Cadence: *Formal Verification Using Affirma FormalCheck*; Manual, Version 2.3, October 1999

[4] Bell Labs Design Automation, Lucent Technologies: *FormalCheck Users Guide*; V2.1, July 1998

[5] R. P. Kurshan: Formal Verification in a Commercial Setting, *Proc. Design Automation Conference (DAC’97)*, Anaheim, California, June 1997, pp 258-262.

[6] The ATM Forum Technical Committee: *UTOPIA Level 2*; Version 1.0, June 1995