ON THE MODEL CHECKING OF THE RCMP-800 INPUT FIFO

Jianping Lu and Sofiène Tahar
Concordia University, ECE Dept., Montreal, Quebec, H3G 1M8 Canada
{jianping, tahar}@ece.concordia.ca

Abstract

In this paper we display several practical approaches adopted for the formal verification of an industrial case study using model checking. The device under investigation is the Routing Control, Monitoring and Policing 800 Mbps (RCMP-800), a product from PMC-Sierra, Inc. RCMP-800 is an integrated circuit that implements ATM (Asynchronous Transfer Mode) layer functions including fault and performance monitoring, header translation and cell rate policing. In particular, we present our experience on model checking of the input FIFO of RCMP-800 using the VIS tool. We successfully established the environments and verified a number of relevant properties in the input process module of RCMP-800, which led to the discovery of a few errors.

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 the error, 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 formal verification [5].

One very successful formal verification approach is model checking [5] which enables to check a design model against temporal logic properties. Model checking is an automatic technique for verifying finite-state reactive systems, such as sequential circuit designs and communication protocols. Specifications are expressed in a propositional temporal logic, and the reactive system is modeled as a state-transition graph. An efficient search procedure is used to determine automatically if the specifications are satisfied by the state-transition graph. While some approaches such as symbolic representations have greatly increased the size of the system that can be verified, most realistic systems are still too large to be handled.

In this paper we present our experience on model checking of the input FIFO of RCMP-800 using the Verification Interacting with Synthesis (VIS) tool [1]. The RCMP-800 (Routing Control, Monitoring and Policing 800 Mbps [8]), a product from PMC-Sierra, Inc., is an integrated circuit that implements ATM (Asynchronous Transfer Mode) [3] layer functions including fault and performance monitoring, header translation and cell rate policing). We wrote the RTL (Register Transfer Level) description of the input FIFO in Verilog. Since, the input FIFO has 128 x 16 bit memory which could contain 4 ATM cells, its verification could not be handled by VIS. Therefore, we abstracted away the memory to concentrate on the functionality of the control circuitry, which is usually the critical part in the verification. We then established an environment for the input FIFO, which gives the inputs as random variables and defines registers as a default value. Thereafter, we defined a set of safety and liveness properties, which we verified against the Verilog model. While most properties could be verified with no problems, some yielded a state space explosion. We hence applied a number of further abstraction and reduction techniques [6] to reduce the state space. This enabled us to verify all properties with a reasonable CPU time. We have been able to find a few bugs in the design, which were not caught during the simulation. Our experience demonstrates the applicability of formal verification for a commercial digital design.

The rest od the paper is organized as follows. In Section 2, we introduce the RCMP-800 device in detail. In Section 3, we describe the behavior of the RCMP-800 Input FIFO. In Section 4, we present the verification environment and the set of properties we verified. In Section 5, we display our experimental results and point to a few design errors detected. Finally, in Section 6, we conclude the paper.

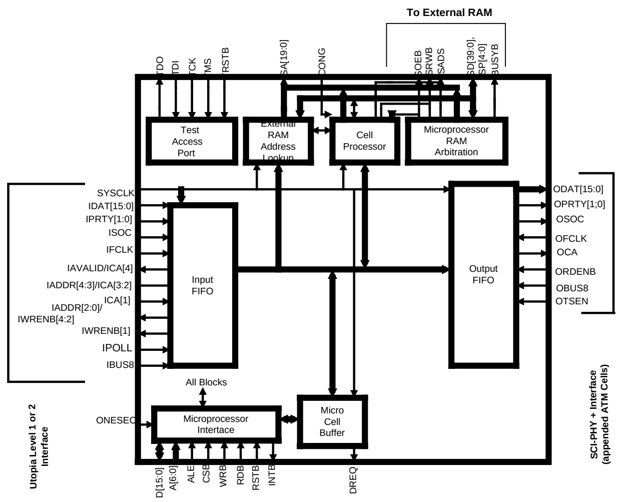

Fig. 1. Structure of the RCMP-800

2. THE RCMP-800

2.1 Function of RCMP-800

The Routing Control, Monitoring and Policing 800 Mbps (RCMP-800) device is an integrated circuit that implements ATM layer functions including fault and performance monitoring, header translation and cell rate policing. The RCMP-800 is intended to be situated between a switch core and the physical layer devices in the ingress direction. It supports a sustained aggregate throughput of 1.42 x 10^6 cell/s. The RCMP-800 uses external SRAM to store per-VPI/VCI (Virtual Path Identifier/ Virtual Channel Identifier) [3] data structures. The device is capable of supporting up to 65536 connections. The input cell interface can be connected to up to 32 physical layer devices through a SCI-PHY [9] compatible bus. The 53 byte ATM cell is encapsulated in a data structure which can contain pre-pended or post-pended routing information. Received cells are buffered in a four cell deep FIFO. All physical layer and unassigned cells are discarded. For the remaining cells, a subset of ATM header and appended bits is used as a search key to find the VC (Virtual Channel) Table Record for the virtual translation. If a connection is not provisioned and the search terminates unsuccessfully, the cell is discarded and a count of invalid cells is incremented. If the search is successful, subsequent processing of the cell is dependent on contents of the cell and configuration fields in the VC Table Record.

Cell rate policing is supported through two instances of the Generic Cell Rate Algorithm (GCRA) for each connection. Each cell that violates the traffic contract can be tagged (CLP (Cell Loss Priority) [3] bit set high) or discarded. To offer more flexibility, each GCRA instance can be programmed to police any combination of user cells, OAM cells, Resource Management (RM), high priority cells or low priority cells.

The RCMP-800 supports multicasting. A single received cell can result in an arbitrary number of cells presented on the Output Cell Interface, each with its own unique VPI/VCI value and appended bytes. The ATM cell payload is duplicated without modification.

The Output Cell Interface can be connected to the switch core through an extended cell format SCI-PHY compatible bus. Cells are stored in a four cell deep FIFO until the downstream devices are ready to accept them. The details of how cells are handled in this FIFO depends on the particular application of the RCMP-800.

The Microprocessor Interface is provided for device configuration, control and monitoring by an external microprocessor. This interface provides access to the external SRAM (Static RAM) of the data structure, configuration of individual connections and monitoring of the connections. The Microprocessor Cell Buffer gives access to the cell stream, either directly by a DMA (Direct Memory Access) controller. Programmed cell types can be routed to a microprocessor readable 16 cell FIFO. The microprocessor can send cells over the Output Cell Interface.

2.2 Structure of RCMP-800

The structure of the chip is shown in Figure 1. It consists of an input FIFO, output FIFO, Microprocessor Interface, Micro Cell buffer, Cell Processor, Microprocessor RAM arbitration, External RAM address Look up and JTAG (Joint Test Action Group standard) Test access port. Both Input and Output FIFO are four cell depth FIFO. The basic function of the input FIFO is to receive the data cells into the RCMP, and that of the output FIFO is to transmit the data cells to the fabric. ATM cells are transferred to Micro Cell buffer from the input FIFO, and the microprocessor will read the ATM cells in the Micro Cell, and checks the cell types, cell header, VCI/VPI and CI (Connection Identifier). Depending on that information, the microprocessor looks for the VC table, and determines the pre-pend and post-pend bytes, or tags CLP bit or discards the cell if GCRA is violated, or inserts RM or OAM cell which will be written in Micro Cell Buffer. Cell Processor and Microprocessor RAM arbitration, External RAM address look-up are used to look up VC routing table, so this part involves both hardware and software. We are interested in the input and output FIFOs of the RCMP-800 hardware. The whole chip verification which involves hardware and software verification is not possible for model checking due to the limitation of current tools.

However, model checking can be used to verify an individual block of the RCMP-800, here we use the input FIFO as an example.

3. BEHAVIOR OF RCMP-800 INPUT FIFO

3.1 Interface of RCMP-800 Input FIFO

Cells received on the extended cell format SCI-PHY compatible Input Cell Interface are buffered in a 4 cell deep FIFO. The input buffer provides for the separation of internal timing from asynchronous external devices. The SCI-PHY cell interface operates at clock rates up to 52 MHz and supports 16 bit and 8 bit wide data structures with programmable lengths. The 16 bit data structure contains 26 (HEC and User Defined Field excluded) or 27 words allocated to carry an ATM cell and up to 5 appended words. The 8 bit data structure contains a 52 bytes (HEC excluded) or 53 bytes ATM cell and up to 10 appended bytes.

The input FIFO filters cells both unassigned and reserved for the use of the Physical Layer. Unassigned cells are identified by an all zero VPI/VCI value and CLP=0. They are filtered without notification. Physical layer cells are identified by an all zero VPI/VCI value and CLP = 1. They are filtered with a resulting maskable interrupt indication and a Physical Layer cell count increment. By default, the cell coding is assumed to be for a Network-Network interface (NNI); therefore the VPI is taken to be twelve bits.

The RCMP-800 is a bus master and services the PHY devices as one of two ways: direct status arbitration or address line polling. For direct status arbitration, the RCMP-800 monitors cell available signals (*ICA[4:1]*) from up to four physical (PHY) layer devices and generates write enables (*IWRENB[4:1]*) in response. For address line polling, *ICA[1]* and *IWRENB[1]* are shared between up to 32 PHY devices and signals *IADDR[4:0]* and *IAVALID* are used to address the latter individually. The RCMP-800 performs round-robin polling of the PHY devices to determine which have available cells. The RCMP-800 will read an entire cell from one PHY device before accessing the next PHY device. No fixed cell slots exist, but instead the RCMP-800 maximizes throughput by servicing a PHY device as soon as the bus is free and PHY device’s cell available signal is asserted.

3.2 Functions of RCMP Input FIFO

The main functions of the input FIFO are the following:

1. Storing ATM cells in the input FIFO. There are three counters: read counter, write counter and FIFO counter. Read counter and write counter are used to control the read and write of the FIFO, and FIFO counter indicates the depth of the FIFO.
2. Checking parity for the input data, and the parity check result is stored in the register.
3. Discarding the unsigned cells.
4. Discarding physical layer OAM cell with a notification
5. Implementing two modes of the PHY devices services:
   - One is direct status arbitration. Only four physical devices are on the transceiver board, and each physical device has an individual *ICA* (Input Cell Available) and *IWREN* (Write Enable) signals.
   - The other is address line polling. There are 32 physical devices which are accessed by 5-bit physical address and share the same *ICA* and *IWREN* signals.

4. VERIFICATION OF THE INPUT FIFO

We used the same methods that we described in [7] to do the verification. We wrote the RTL description of the input FIFO in Verilog, with some minor changes on the model. The difference between the original model and our verification model was that we only used 16-bit datapath while the original design used either 16-bit or 8-bit datapath. In addition, the input FIFO had 128 x 16 bit memory which could contain 4 ATM cells, but VIS could not handle such big memory verification. Therefore, we verified the control circuits of the input FIFO excluding the memory. Such reduction is practical because usually the control circuit is the critical part in the verification.

4.1 The Environment of the Input FIFO

Similar to [7], we established an environment for the input FIFO, and the environment gives the inputs as random variables and defines registers as a default value. Figure 2 gives the code for the required environment.

In Figure 2, lines 2 to 9 are inputs from transceiver board and the block of Micro cell buffer, so we define them as nondeterministic variables. Lines 10 to 14 are registers, so we give them as their default values. Because our verification is to focus on the critical behavior of the block, the constant register values will not have an influence on the verification. However, to further verify the block, we could apply other constant values for these registers.

```
always @(posedge clock) begin
idat = idat_ran;
prty = prty_ran;
isoc = isoc_ran;
ica4 = ica4_ran;
ica1 = ica1_ran;
ica32 = ica32_ran;
ipoll = ipoll_ran;
ifrdb = ifrdb_ran;
hecudf = 1;
icainv = 0;
cellpost = 0;
celllen = 0;
ibyteprty = 0;
icalevel0 = 1;
ifrst = 0;
iptyp = 0;
end
```

Fig. 2. Environment of the input FIFO

4.2 Properties Description

According to the functions of the input FIFO described in Section 3, we give 8 properties. In the following CTL (Computational Tree Logic) expressions, "*", "->" and "^" mean logical "and", "imply" and "xor", respectively. "AG" and "AX" are CTL operators meaning for all paths in all states, and for all paths in the next state, respectively.

Property 1. In normal operation (not in *discard* operation), the write counter will increment by 1 whenever *writeB* is deasserted. The CTL expression is the following:

```
AG (discard = 0 * writeB=0 -> AX (wr_ptr ==
 wr_ptr_plus1))
```

where *discard* is an internal signal which determines whether the FIFO is in normal operation or discard operation, *writeB* is a write enable signal, and *wr_ptr* is a write pointer. We introduce the assistant variable *wr_ptr_plus1* in the design module, which will always be *wr_ptr + 1* with one clock cycle delay.

Property 2. Whenever the signal *ifrdb* is deasserted, then the read counter will be incremented by 1. The property can be expressed as follows:

```
AG(ifrdb = 0  -> AX(rd_ptr == rd_ptr_plus1))
```

where *ifrdb* indicates Micro cell FIFO has enough space to receive a cell, and *rd_ptr* is read pointer of the input FIFO. Similar to *wr_ptr_plus1*, *rd_ptr_plus1* is introduced to have the value of *rd_ptr + 1* with one clock cycle delay.

Property 3. In a normal operation, the signal *ifcounter* will be incremented by 1 whenever *writeB* is deasserted and *ifrdb* is asserted; and *ifcounter* will be decrement by 1 whenever *writeB* is asserted and *ifrdb* is deasserted. Formally:

```
AG  (discard=0  *  writeB  =0  *  ifrdb  =  1  ->
 AX(ifcounter ==ifcounter_plus1))

AG(discard=0  *  writeB  =  1  *  ifrdb  =  0   ->
 AX(ifcounter == ifcounter_minus1)
```

Here, *discard*, *writeB* and *ifrdb* have the same meaning as Property 2. *Ifcounter* indicates the depth of the input FIFO. Similarly, *ifcounter_plus1* and *ifcounter_minus1* are introduced to represent the values of *ifcounter + 1* and *ifcounter - 1* with one clock cycle delay, respectively.

Property 4. The parity check is correct and the result will be stored in the register. Since we define *ibyteprty* as default value "0" and *iptyp* as a default value "0", it is a word-basis odd parity checking. With *iprty[1] = 1* indicating the parity error over the *IDAT[15:0]* bus, the specification of the property is:

```
AG(ibyteprty = 0 * iptyp=0 -> (iprty[1] = !
 (prty[1] ^ idat[0] ^ idat[1] ^ idat[2] ^ … ^
 idat[15])).
```

We use division [7] to verify this property using sub-properties:

```
AG (prtychkl == idat[0] ^ idat[1] ^ idat[2] ^
 idat[3]  ^  idat[4]  ^  idat[5]  ^  idat[6]  ^
 idat[7])

AG  (prtychk2 ==idat[8]  ^idat[9]  ^  idat[10]  ^
 idat[11]  ^  idat[12]  ^  idat[13]  ^  idat[14]  ^
 idat[15])

AG(iprty[1]= !(prtychkl ^prtychk2 ^ prty[1]))
```

Property 5. Any unassigned cells will be discarded by the input FIFO. Actually, the ATM header determines whether a cell is unassigned cell or not. If all the bits of VPI and VCI and CLP bit are zero, then the cell is unassigned. Since we consider NNI here, the format of an unsigned cell is similar to the one in Table 1 [4].

Property 5 can be deduced through the formulas:

```
AG (idat[15:0]=0  * cellcount = 0 * writeB = 0
 -> AX (vpi_vci[27:12] = 0 * cellcount = 1))

AG (idat[15:0]=0  * cellcount = 0 * writeB = 0
 -> AX AX (vpi_vci[27:12] = 0))

AG(idat[11:0]=0 * idat[15]=0 * cellcount= 1 *
 writeB = 0 -> AX (vpi_vci[11:0] =0 * clp = 0))

AG(vpi_vci[27:0] * clp = 0 * cellcount=2 -> AX
 discard = 1)
```

where *idat* is a 16 bit datapath which receives data from transceiver board, *vpi_vci* is a 28-bit registers which store VPI and VCI value for each cell, and *cellcount* indicates how many data bytes have been transferred into RCMP for each cell.

Table 1. Format of an unassigned cell

ATM header	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
unassigned cell	00000000	00000000	00000000	0000xxx0	HEC byte

Property 6. Any physical cells will be discarded by the input FIFO with a notification.

Similar to unassigned cell, physical cell is determined by its ATM header, and the format of a physical cell is in Table 2 [4].

Like Property 5, Property 6 can be deduced by the following three CTL expressions:

```
AG(idat[15:4] = 0 * cellcount = 0 * writeB = 0
 -> AX vpi_vci[23:12] = 0 * cellcount = 1)
```

```
AG(idat[15:4] = 0 * cellcount = 0 * writeB = 0
 -> AX AX vpi_vci[23:12] = 0 )

AG(idat[11:0]=0  *idat[15]=1  *  cellcount=1  *
 writeB= 0 ->AX(vpi_vci[11:0]=0 * clp=1))

AG(vpi_vci[23:0] * cellcount = 2 * clp = 1 ->
 discard = 1 * phycell = 1)
```

Table 2. Format of a physical cell

ATM header	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
Physical cell	xxxx0000	00000000	00000000	0000xxx1	HEC byte

Property 7. If *ipoll* is low, the RCMP is receiving data from PHY device 1 and PHY device 2 has a cell available, then PHY device 2 will transmit a cell to RCMP next.

Here, switching a receiving PHY device from one to another only happens at the state *cellcount = 0*. When a PHY device gets permission to transfer a cell into RCMP, the write enable signal (*iwren*) will be asserted. And also the variable *p_state* stores the number of PHY devices. Therefore, the CTL expression is as follows:

```
AG ( ipoll = 0 * cellcount = 0 * p_state = 1 *
 ica2 = 1  -> AX (iwren2 = 1))
```

Here, *p_state* stores the number of current receiving PHY devices, *ica2 = 1* means that PHY device 2 has a cell to send, *iwren2 = 1* means PHY device 2 gets the permission to send.

Property 8. If *ipoll* is high, the RCMP is receiving data from the PHY device of address 10 and the PHY device of address 11 has a cell to send, RCMP will transmit the cell from PHY device of address 11 next.

```
AG (ipoll = 1 * cellcount = 0 * iaddr = 10 * ica
 = 1 -> AX AX (iaddr = 11))
```

where, *iaddr = 10* expresses that the address of the current receiving PHY device is 10. Because RCMP has a pipeline searching process, *iaddr* will be equal to 11 in two clock cycles.

While the above properties do not cover all functions listed in Section 3, other properties can easily be described in a similar way.

5. EXPERIMENTAL RESULTS AND ERROR DETECTION

We did meet state explosion problem when verifying these properties, and the machine gave the error indicating that the memory cannot be allocated. So we applied a reduction method in which we “hide” unrelated design details when verifying a property [6]. Because a hardware design is “process-based”, we could simply comment unrelated process when verifying a property, and it is also possible to write a program to search unrelated processes and comment them automatically. Although the method seems very obvious, it is very effective. By this method, all the properties were verified in VIS with a reasonable CPU time. Table 3 summarizes the experimental results including the number of CTL formulas involved, CPU time in seconds, memory usage and number of BDD nodes generated.

Table 3. Model checking results

Properties	**Number of CTL**	**CPU time (seconds)**	**Memory Usage (MB)**	**Nodes Allocated (K)**
Property 1	1	75	97	103, 907
Property 2	1	57	68	87,103
Property 3	2	63	59	91,034
Property 4	3	56	87	79,308
Property 5	3	62	61	71,805
Property 6	3	74	102	174,830
Property 7	1	23	42	34,049
Property 8	1	12	34	20,911

It is to be noted that before performing the model checking, we carried out an extensive simulation of the RCMP-800 Input FIFO, but we still have found a number errors. For instance, we identified a bug in the "address line polling circuitry" while model checking Properties 7 and 8. After fixing the RTL code, the properties succeeded the model checking.

6. CONCLUSIONS

In this paper, we applied model checking to verify a block of an ATM commercial product. We show how to described the properties in CTL. To save state space, we defined register variables as their default values in the environment. However, we still encountered state space explosion problem. This has been solved by adopting a reduction method in which we comment out property unrelated HDL design code before running model checking. In this work, model checking of all the properties are done with a reasonable time, and we did detect a set of design errors by the model checking.

References

[1] R. Brayton et.al., "*VIS: A System for Verification and Synthesis*," Technical Report UCB/ERL M95, Electronics Research Laboratory, University of California, Berkeley, December 1995.

[2] H D. Ginsburg, "*ATM Solutions for Enterprise Internetworking*," Addisson-Wesley, 1996.

[3] N. Giroux and S. Ganti. "*Quality of Service in ATM Network*," TK5105.35.G58, 1998.

[4] F. Halsall, "*Data Commutations, Computer Networks and Open Systems*," Addison-Wesley, 1996.

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

[6] J. Lu, "*On the Formal Verification of ATM Switches*," M.A.Sc. Thesis, Concordia University, Department of Electrical and Computer Engineering, May 1999.

[7] J. Lu and S. Tahar. "Practical Approaches to the Automatic Verification of an ATM Switch Fabric using VIS," *Proc. IEEE 8th Great Lakes Symposium on VLSI*, Lafayette, Louisiana, USA, pp. 368-373. February 1998.

[8] PMC-Sierra, Inc. "*ATM Layer Routing Control, Monitoring and Policing 800 Mbps*," PMC-940904, August 1997.

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