MODELING AND VERIFICATION OF AN ATM PORT CONTROLLER IN VIS

Jianping Lu and Sofiène Tahar
Dept. of ECE, Concordia University
{jianping, tahar}@ece.concordia.ca

Abstract

In this paper we display a practical approach adopted for the formal verification of Fairisle ATM (Asynchronous Transfer Mode) switch port controller using model checking. The ATM port controller is part of the Cambridge Fairisle ATM network and plays a key role in the ATM switching process. In particular, we present our experience on the model checking of the ATM port controller using the VIS tool from UC Berkeley. To this end, we successfully modeled the port controller behavior and structure in Verilog HDL, established the necessary verification environments and verified a number of relevant temporal properties on the port controller.

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 bugs, 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 [3].

One very successful formal verification approach is model checking [3] 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. However, the specifications are not always easy to be expressed in the given temporal logic. In this paper, we display practical approaches to represent the specification in the temporal logic and present our experience on the model checking of the Fairisle ATM (Asynchronous Transfer Mode [2]) port controller using the VIS (Verification Interacting with Synthesis) [1] tool from UC Berkeley.

The Fairisle port controller (Fig 1) is a real design from Cambridge University. It is at the heart of Fairisle ATM network switch [4]. In the ingress [2], the port controller receives ATM cells from the transmission board and performs the ATM switching on the received cells. It also sends the ATM cells to the switch fabric [5]. In the egress [2], the port controller receives ATM cells from the fabric and sends the acknowledgment signals to the switch fabric. The port controller assigns priorities to ATM cells, by preloading priority bits into the memory. The priority bit will be used for arbitration in the switch fabric.

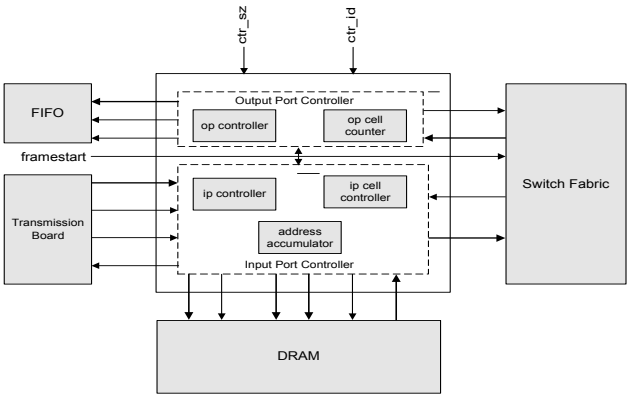

Fig 1. The Fairisle ATM switch

In this work, we modeled the port controller at the RTL (Register Transfer Level) following some documentation and incomplete structural code we have obtained from Cambridge. The RTL description of the port controller is written in Verilog HDL (Hardware Description Language). To verify the port controller in VIS, we established a proper environment, and defined a number of related CTL (Computation Tree Logic [3]) properties.

CCECE 2003 - CCGEI 2003, Montréal, May/mai 2003
0-7803-7781-8/03/$17.00 © 2003 IEEE

In following sections, we will introduce the behavior and structure of the port controller in Section 2. Section 3 describes the properties we established on the port controller. Section 4 illustrates a practical method on verifying CTL properties using model checking, and Section 5 summarizes the paper.

2. THE FAIRISLE PORT CONTROLLER

Fig 2 shows the format of an ATM cell. Received cells have 52 bytes: 48 data bytes, 2 VCI (Virtual Channel Identifier) bytes and 2 FAS (Frame Assignment Sequence) bytes. Transmitted cells have 54 bytes: 48 data bytes, 1 Fabric Routing Byte (FRB), 1 Port controller Routing Byte (PRB), 2 VCI bytes and 2 FA bytes.

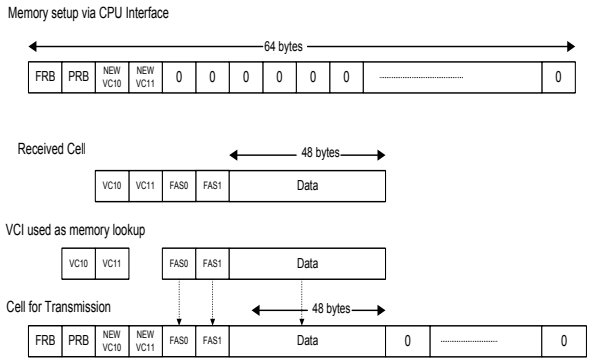

Fig 2. Format of received and transmitted cells

2.1. Behavior of the Fairisle Port Controller

The Fairsile port controller consists of input port controller and output port controller. The input port controller receives ATM cells from the transmission board, and writes them into the memory at an address based on the value of the VCI [2]. In addition, the input port controller reads ATM cells out of the memory and transmits them into the switch fabric. Once it receives positive acknowledgement signals, the input port controller will continue transmitting data; otherwise, it will stop sending data. The output port controller receives data cells from the fabric, and sends acknowledgment signals back to the fabric. If the output port controller receives a data cell, it gives a positive acknowledgment signal; otherwise, it sends a negative acknowledgment.

The input port controller always monitors the *framestart* signal (Fig 1). On an active *framestart* signal, the input port controller will assert a write enable signal to the memory. After the *framestart* signal is received, the input port controller will latch the first two bytes which build the VCI field of the receiving cell. (Fig 1) is the conversion between the VCI of the receiving cell and its memory location, *c* means the column address and *r* means the row address. The decimal digit indicates the position in the binary address (e.g. *r4* means bit 4 of the row address).

On the other hand, when the *framestart* signal is asserted and the input port controller has a cell to send, the input port controller will read the data cell from the memory into the fabric. After a certain number of clock cycles, if the input port controller receives the positive acknowledgment signal through the switch fabric, it will continue sending the ATM cell; otherwise, it will stop the transmission.

Table 1. VCI to memory location conversion

VCI 0 byte	0	1	2	3	4	5	6	7
addr_r	r1	r2	r3	r4	r5	r6	r7	r8
VCI 1 byte	0	1	2	3	4	5	6	7
addr_r	-	-	-	-	c6	c7	c8	r0

While the input port controller receives data from the transmission board and transmits the data into the fabric, the output port controller receives data from the switch fabric and gives the acknowledgment signal to the input port controller through the switch fabric. After the *framestart* signal is asserted, the output port controller will detect the active bit in the port controller header. If the active bit is asserted, the output port controller will generate the positive acknowledgment signal which will be transmitted into the input port controller through the fabric; otherwise, the output port controller will generate the negative acknowledgment signal. If the output port controller receives a data cell, it will write the data into the output FIFO, and the first byte of the data, which is the Port controller Routing Byte (PRB), will be stripped.

2.2. Structure of the Fairisle Port Controller

Fig 3 shows the structure of the port controller. The signals *ip_mem_data* and *mem_ip_data* mean the data outputs to the cell memory and the data inputs from the cell memory, respectively. Both signals have 8-bit bus width. The signals *ip_mem_wr_en* and *ip_mem_rd_req* are the memory write enable and memory read request signals, respectively. The memory row and column addresses are provided by *ip_mem_addr_r* and *ip_mem_addr_c*, respectively. The *rx_ip_data* is an 8-bit data bus which is the data inputs from the transmission board. The signals *rx_rd_req* and *rx_ip_soc* indicate cell availability in the transmission board and the start of a cell, respectively. The *rx_ip_soc* signal corresponds to the *framestart* mentioned

above. The signal *ip_rx_wr_en* demonstrates whether the input port controller is able to accept a cell or not. The *ip_fab_data* is an 8-bit data bus which transfers data from the input port controller to the fabric.

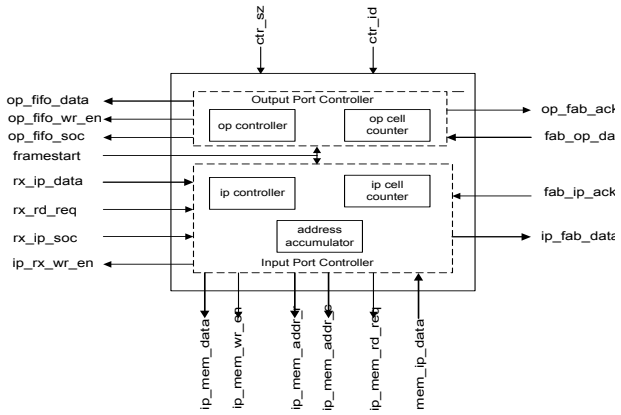

Fig 3. The structure of the port controller

The *fab_ip_ack* is the acknowledgment signal which indicates whether the current cell succeeded the transfer to the destined output port controller. The *fab_op_data* is an 8-bit data inputs from the fabric to the port controller and the *fab_op_ack* is the acknowledgment signal generated by this latter. The op_fifo_data is an 8-bit datapath from the output port controller to the FIFO. The op_fifo_wr_en is the write enable signal for the output FIFO. The signal op_fifo_soc indicates the start of a cell, and it is asserted before the first byte data transfer. The npc-rst_n is the reset signal.

Inside the port controller, there are two control registers (*ctr_id* and *ctr_sz*) and one status register (*ip_empty*). The *ctr_id* disables the inputs when it is asserted. The register *ctr_sz* is for debugging purposes. The register *ip_empty* is used to indicate the status of the port controller.

3. PROPERTIES OF THE PORT CONTROLLER

Based on the expected function of the port controller, we defined the following six major properties.

Property 1: The Fairisle port controller will be reset properly when the reset signal (*npc_rst_n*) is zero .

Property 2: When the input port controller can accept a cell, the transmission board has a cell to send, and the input port controller is in debugging state (*ctr_sz = 1*), then the cell will be transferred to the input port controller and stored in the memory at the right location.

Property 3: When the input port controller can accept a cell, the transmission board has a cell to send, and the input port controller is in the normal operation state (*ctr_sz=0*), then the cell will be transferred to the input port controller and stored in the memory at the right location.

Property 4: When the input port controller has a cell to send, it will send the cell to the fabric. If the input port controller does not receive a positive acknowledgment signal, it will stop sending the cell; otherwise, it will send the data cell completely.

Property 5: The memory cannot be read and write at the same time.

Property 6: The output port controller will send an acknowledgment signal after it detects an incoming cell.

Each of the above properties will be described formally in CTL. Due to limited space, we will report in detail the formal specification and verification of one sample property, Property 3, in next section. The CTL specification of the other properties can be found in [6].

4. SPECIFICATION AND VERIFICATION

In this section we will demonstrate by example (using Property 3) how the specification and verification is processed in a practical way. Property 3 has the following assumptions:

1. The input port controller can accept a cell, expressed as "*ip_empty* = 1";
2. The transmission board has a cell to send, expressed as "*rx_ip_rd_req* = 1";
3. The port controller is in normal operation state, expressed as "*ctr_sz* = 0" and "*ctr_id* = 0";
4. The port controller receives the *framestart* signal, expressed as "*rx_ip_soc* = 1";
5. The input port controller is not in reset state, and it can be expressed as "*npc_rst_n* = 1" .

The input port controller first detects the signals *ip_empty*, *rx_ip_rd_req*, *ctr_sz, npc_rst_n,* and *ctr_id*. If these signals are satisfied with the above assumptions, the port controller will start monitoring *rx_ip_soc* in the following clock cycles. If the *rx_ip_soc* is asserted as well, the cell is transferred to the port controller. This behavior can be expressed formally in CTL as the follows, where the symbols "*", "+", "->" represent logical and, or and implication, respectively, and the symbols "AG" and "AX" are temporal operators meaning for all paths in all states and for all paths in next state, respectively.

```
AG(npc_rst_n=1 * ctl_id=0 * ctr_sz=0 * ip_empty=1
* rx_ip_rd_req=1 * rx_ip_soc=1  -> AX AX
ip_mem_addr_r[8:1] == rx_ip_data)
```

The above CTL expression is not fully correct because the assumptions (*ip_empty*=1 and *rx_ip_rd_req*=1) do not happen at the same state as the fourth assumption (*rx_ip_soc*=1). Therefore, we need to put the assumptions into the environment. In fact,

because the port controller has a cyclic period synchronized by the *rx_ip_soc* signal whose period is 64 clock cycles, we could establish an environment state machine with 64 states (Fig 4).

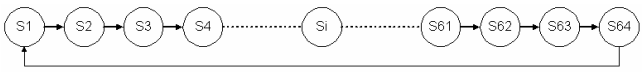

Fig 4. Environment machine for the port controller

The Verilog code for the above environment of the port controller for Property 3 is shown in Fig. 5. In this environment, the correct assumption *npc_rst_n*=1, *ctl_id=0, ctr_sz*=0, *ip_empty*=1, *rx_ip_rd_req*=1, and *rx_ip_soc*=1 are set properly.

```
1  typedef num {S1, S2, S3, …, Si, … , S64} state;
2  assign rx_ip_data_ran = $ND(0, 1, 2, …, 255);
3  always @ (posedge clock) begin
4  case (state)
5     S64: state = S1;
6     S1: state = S2;
7      S2: state = S3;
8     S3: state = S4;
9        ……
10    Si: state = Si+1;
11     ……
12    S63: state = S64;
13 endcase;
14 if (state== S1)
15    framestart = 1;
16 else
17    framestart = 0;
18 if (state == S3)
19    rx_ip_soc = 1;
20 else
21    rx_ip_soc = 0;
22 ip_empty = 1;
23 rx_ip_rd_req=1 ;
24 ctr_sz = 0;
25 ctr_id = 0;
26 npc_rst_n = 1;
27 rx_ip_data = rx_ip_data_ran;
28 if (state = S4)  rx_ip_data_s4 = rx_ip_data_ran;
29 else if (state=S5) rx_ip_data_s5 = rx_ip_data_ran;
30 else if (state=S6) rx_ip_data_s6 = rx_ip_data_ran;
31 end
```

Fig 5. Verilog code of Property 3 environment

Next, we divide the verification into the two steps. The first step is to verify whether the environment represents the assumption, and the second step is to check if the conclusion is valid.

Step 1. Verify the assumption

The five assumptions are expressed using the following CTL expressions:

```
AG(npc_rst_n=1 * ctr_id=0 * ctr_sz=0)                    (1)

AG(state=S1 -> ip_empty =1 * rx_ip_rd_req=1)             (2)

AG(state=S3 -> rx_ip_soc=1)                              (3)
```

Step2. Verify the conclusions.

In Property 3, we have to verify two aspects: one is to ensure that two bytes of VCI become the memory address and the memory address is incremented by 1 per byte data transfer. And the other is to verify that the data is transferred from transmission board to the memory with one clock cycle delay and the memory write enable signal is asserted during the data transfer. The formulas (4) and (5) below specify that the two bytes of VCI are transferred to be memory address correctly.

```
AG(state=S5 -> ip_mem_addr_r[8:1]==rx_ip_data_s4)   (4)

AG(state=S6 -> ip_mem_addr_r[8:1]== rx_ip_data_s4
* ip_mem_addr_r[0]==rx_ip_data_s5[7] *
ip_mem_addr_c[8:6]==rx_ip_data_s5[6:4] *
ip_mem_addr_c[5:0]=6'b000100)                        (5)
```

The correct memory addresses increment can be specified by formulas (6), (7), (8) and (9) below. Due to the space limitation, the CTL properties for the address increment between S8 and S56 are not listed in here, but they are very similar to (7) and (8).

```
AG(state=S7 -> ip_mem_addr_r[8:1]== rx_ip_data_s4
* ip_mem_addr_r[0]==rx_ip_data_s5[7] *
ip_mem_addr_c[8:6]==rx_ip_data_s5[6:4] *
ip_mem_addr_c[5:0]=6'b000100)                        (6)

AG(state=S8 -> ip_mem_addr_r[8:1]==rx_ip_data_s4 *
ip_mem_addr_r[0]==rx_ip_data_s5[7] *
ip_mem_addr_c[8:6]==rx_ip_data_s5[6:4] *
ip_mem_addr_c[5:0]=6'b000101)                        (7)

AG(state=S56 -> ip_mem_addr_r[8:1]==rx_ip_data_s4
* ip_mem_addr_r[0]==rx_ip_data_s5[7] *
ip_mem_addr_c[8:6]==rx_ip_data_s5[6:4] *
ip_mem_addr_c[5:0]= 6'b110101)                       (8)

AG(state=S57 + …… +  state=S64 ->
ip_mem_addr_r[8:1]==rx_ip_data_s4 *
ip_mem_addr_r[0]==rx_ip_data_s5[7] *
ip_mem_addr_c[8:6]==rx_ip_data_s5[6:4] *
ip_mem_addr_c[5:0] = 6'b000000)                      (9)
```

Next, we verify that the data is transferred from the transmission board to the memory with one clock cycle delay and the memory write enable signal is asserted during data transfer process. This sub-property involves two signals. One is the memory write enable signal (ip_mem_wr_en) and the other is the data output signal (ip_mem_data). The ip_mem_wr_en signal, which should be asserted during the data transfer period (S7 to S56), is expressed by formulas (10) and (11) below. Also during the data transfer period, the ip_mem_data should equal the value of rx_ip_data with one clock cycle delay. The first and last byte data transfers are represented by formulas (12) and (13) below. Due to the space limitation, the CTL properties for the rest of data transfer are not enumerated in here, but they are very similar to (12) and (13).

```
AG(state=S1 + state=S2 + … + state=S6 + state=S57
+ … + state=S64 -> ip_mem_wr_en=0)            (10)

AG(state=S7 + state=S8 + … + state=S56 ->
ip_mem_wr_en=1)                               (11)

AG(state=S7 -> ip_mem_data==rx_ip_data_s6)     (12)

AG(state=S56 -> ip_mem_data==rx_ip_data_s55)  (13)
```

Using the established environment and combining the property assumptions and conclusions, Property 3 is successfully verified through model checking in VIS. Following the above method, all other 5 properties of the port controller have been similarly specified and verified. Experimental results on the model checking of all 6 properties are shown in Table 2, including CPU time, memory usage and number of BDD nodes generated. The experiments were performed on a Sun Ultra Sparc (300MHz/500 MB) machine.

Table 2. Model checking experiment results

Properties	CPU time (sec)	Memory usage (MB)	BDD nodes allocated (K)
Property 1	52	92	203,493
Property 2	256	198	284,563
Property 3	209	156	293,354
Property 4	378	201	304,731
Property 5	34	77	153,980
Property 6	76	89	197,091

5. CONCLUSION

In this paper, we have presented the modeling and formal verification by model checking of an ATM switch port controller. This is a real design of a telecommunications component used in the Cambridge Fairisle ATM network. While some specification properties cannot be concisely expressed using single temporal logic formulas, we have shown how we make use of an environment state machine to enable a proper specification. To enable the model checking process, properties are further subdivided into a set of assumption and conclusion subformulas which are combined by conjunction. Using such an approach, we succeeded the model checking of all specification properties of the port controller within the reasonable time. The method presented could be enough in order to verify larger designs. To this end, we may have to apply some more advanced techniques, such as symmetry reduction or compositional verification [5].

6. 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", Addison-Wesley, 1996.

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

[4] I. Leslie and D. McAuley, "Fairisle: An ATM Network for the Local Area", ACM Communication Review, Vol. 19, No. 4, Sep. 1991, pp. 327-336.

[5] 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, Feb. 1998, pp. 368-373.

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