# Design and Verification of an ATM Knockout Switch Concentrator

Jianping Lu and Sofiène Tahar

Electrical & Computer Engineering Department, Concordia University
Montreal, Quebec, Canada
Email: {jianping, tahar}@ece.concordia.ca

## Technical Report

February 2001

**Abstract.** *In this paper we describe the design and verification of the concentrator of a Knockout ATM (Asynchronous Transfer Mode) switch fabric using the VIS tool. The Knockout is a popular ATM switch fabric which has application in both datagram and virtual circuit packet networks. The concentrator is the most difficult component in the Knockout ATM switch fabric. We developed an RTL structural design as well as a higher-level behavioral model of the Knockout switch concentrator in Verilog HDL. We then used equivalence checking within VIS to verify the concentrator structure against its behavioral model. While sequential equivalence checking failed, we succeeded the combinational equivalence checking of a latch-reduced model of the concentrator.*

## 1. Introduction

With the rapid development of EDA tools, a top-down design phase has been used in industry. An ASIC designer usually describes a design as RTL (Register Transfer Level) code in some Hardware Description Language (HDL) such as Verilog or VHDL. The RTL design is then synthesized into gate-level netlist automatically using synthesis tools. The gate level design will be further converted into transistor-level design by using standard cells. Throughout these conversions, it is mandatory to ensure the correctness of each design phase. In addition to the RTL design, sometimes a behavioral description is developed in order to prepare test benches and proper simulation environments. However, it would be interesting to directly compare the RTL and behavioral models. This can be achieved by giving the same inputs to the RTL and behavioral code and checking whether the outputs of both models are the same. For a large design, such exhaustive simulation is impossible, so the practical way to do this is using random simulation. However, some special scenarios cannot be detected by random testing. Using equivalence checking based on formal methods [7], such verification is made possible. In fact, if equivalence checking can replace random simulation, such comparison will be more reliable. Because of the above reasons, equivalence checking becomes quite useful in hardware verification.

Equivalence checking can be divided into two categories: combinational equivalence checking [6] and sequential equivalence checking [4]. Combinational equivalence checking is based on Reduced Ordered Binary Decision Diagrams (ROBDDs) [2] which represents a circuit as a binary decision diagram. Bryant [2] proved that a circuit can be described as a reduced binary diagram

which is a canonical form, i.e., every circuit (function) has a unique representation. Hence, equivalence checking simply involves testing whether the two binary diagrams match exactly. This method has been efficiently applied in almost every equivalence checking tool. Some commercial tools have also been used in industry to verify the equivalence between RTL and gate-level circuits. However, since current designs are mainly synchronized and clock-driven, to perform the combinational equivalence checking between two different level sequential circuits, we have to cut the designs into pieces, and map each latch (i.e., 1-bit register or flip-flop) of one level design into another, and compare the combinational circuits between every two consecutive latches. Therefore, combinational equivalence cannot be applied to compare an RTL design with its behavioral counterpart because their internal latches cannot be mapped correspondingly.

Instead, sequential equivalence checking [4] can be used to verify the equivalence between two levels of a design without latch mapping, that means it can be applied to verify the equivalence between an RTL design and its behavioral model. But the drawback of sequential equivalence checking is the so-called state space explosion problem [7] so that it is hard to be applied in a large design.

To make use of sequential equivalence checking, in [8] we applied *modular* sequential equivalence checking on the verification of Fairisle ATM (Asynchronous Transfer Mode) switch [9]. Modular means to verify the equivalence between *submodules* of the behavioral and RTL models, or *submodules* of RTL and synthesized gate-level models. Although such equivalence checking cannot ensure the overall behavior correctness of a system, a reliable submodule will reduce the effort in a higher-level module verification. The overall behavior can be further verified by model checking or simulation.

Other related work in the area of the formal verification of communications devices using equivalence checking include the verifications in the MDG (Multiway Decision Graphs) tool [3] of the Fairisle ATM switch fabric [10] and a Telecom System Block from PMC-sierra Inc. [12]. In [10], the authors achieved a three level equivalence checking between behavioral, RTL and gate levels. In [12], the sequential equivalence checking between behavioral model and RTL is described. In both cases, the authors were making use of the data abstraction features of the MDG representation and tool [3].

In this paper, we will apply both combinational equivalence checking and sequential equivalence checking to verify the concentrator block of a Knockout ATM switch [5]. The rest of the paper is organized as follows: We present the general functionality of an ATM Knockout switch in Section 2. and the architecture of a Knockout concentrator in Section 3.. In Sections 4 and 5, we describe the design and Verilog modeling of the structure and behavior of a 4x4 Knockout concentrator, respectively. In Section 6., we discuss the equivalence checking verification. Finally, Section 7. concludes the paper.

## 2. The Knockout ATM Switch

The Knockout switch is an N-input N-output packet switch with all inputs and outputs operating at the same bit rate. Fixed-length packets arrive on the N-input in a time-slotted fashion, with each packet containing the address of the output port. The Knockout switch has application in both datagram and virtual circuit packet networks.

Aside from having control over the average number of packet arrivals destined for a given output, we assume no control over the specific arrival time of packets on the inputs and their associated output addresses. In other words, there is no time-slot specific scheduling that prevents two or more packets from arriving on different inputs in the same time slot destined for the same

output. Hence, to reduce at minimum (or at least provide a sufficiently small probability of) packet loss, packet buffering must be provided in the switch to smooth fluctuations in packet arrivals destined for the same output.

The interconnection fabric for the Knockout Switch has two basic characteristics: (1) each input has a separate broadcast bus, and (2) each output has access to the packets arriving on all inputs. Figure 1(a) illustrates these two characteristics where each of the N inputs is placed directly on a separate broadcast bus and each output passively interfaces to the complete set of N buses. This simple structure provides several important features.

First, with each input having a direct path to every output, no switch blocking occurs where packets destined for one output interfere with (i.e., delay or block) packets going to other outputs. The only congestion in the switch takes place at the interface to each output where, as mentioned, packets can arrive simultaneously on different inputs destined for the same output. Without a priority scheduling of packet arrivals, this type of congestion is unavoidable, and dealing with it typically represents the greatest source of complexity within the packet switch. The focus of the Knockout Switch architecture is one of minimizing this complexity.

In addition to the above property, the switch architecture is modular: the N broadcast buses can reside on an equipment backplane with the circuitry for each of the N input/output pairs placed on a single plug-in circuit card. Hence, the switch can grow modularly from 2 x 2 up to N x N by adding additional circuit cards.

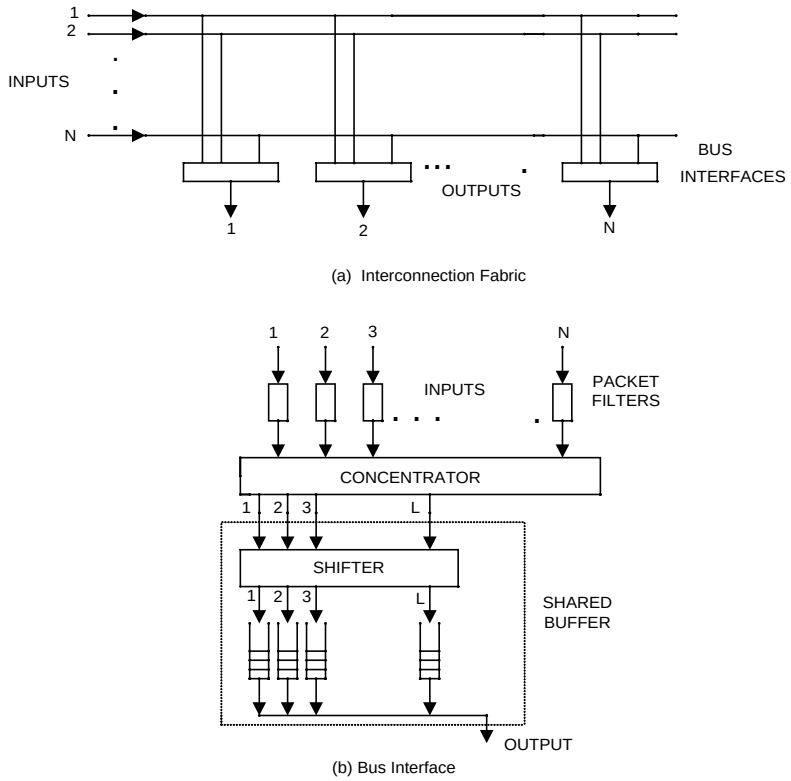


**Figure 1: The Knockout ATM switch**

## 3. Architecture of the Knockout ATM Switch

Figure 1(b) illustrates the architecture of the bus interface associated with each output of the switch. The bus interface has three major components: packet filter, concentrator and shifter buffer. At the top of Figure 1(b), there is a row of N packet filters. Here the address of every packet broadcast on each of the N buses is destined, with packets addressed to the output are allowed to pass on to the concentrator, others are blocked. The concentrator then achieves an N to L (L << N) concentration of input lines, wherein up to L packets making it through the packet filters in each time slot will emerge at the outputs of the concentrator. These L concentration outputs then enter a shared buffer composed of a shifter and L separate FIFO buffers. The shared buffer allows complete sharing of the L FIFO buffers and provides the equivalent of a single queue with L inputs and one output. Operating under a first-in first-out queuing discipline.

In Figure 1(b), the circuits for packet filter and shifter are very simple and common, and they are quite similar to other ATM switches. However, the concentrator circuit is very special. Specifically, the function of the concentrator is: if there are k packets arriving at a time slot for the same output address, these k packets, after passing through the concentrator, will emerge from the concentrator on outputs 1 to k, when k < L+1. If k > L, all L outputs of the concentrator will have packets, and k-L packets will be dropped (i.e., lost) within the concentrator.

The basic building block of the concentrator is a simple 2 x 2 contention switch shown in Figure 2(a). The two inputs contend for the "winner" output according to their activity bits. If only one input has an arriving packet (indicated by the active bit = 1), it is routed to the winner (left) output. If both inputs have arriving packets, one input is routed to the winner output and the other input is routed to the looser output. If both inputs have no arriving packets, we do not care except that the active bit for both should remain at logic 0 at the switch outputs.

The above requirements are met by a switch with the two states as shown in Figure 2(b). Here, the switch examines the active bit for only the left input. If the active bit is at logic 1, the left input is routed to the winner output and the right input is routed to the loser output. If the active bit is a logic 0, the right input is routed to the winner output, and no path is provided through the switch for the left input.

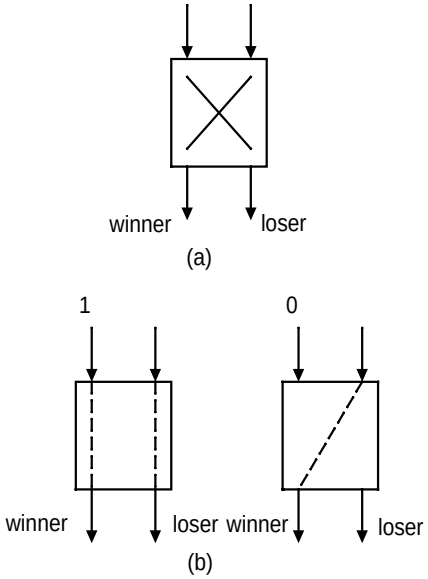


**Figure 2: 2 x 2 contention switch**

## 4. Knockout Switch Concentrator Structure

Figure 3 shows the design of an 8-input/4-output concentrator composed of simple 2 x 2 switch elements and single-input/single-output 1-bit delay element (marked by "D"). This design can be easily expanded to N-input/L-output concentrator with the same two elements, so it has the advantage of easy implementation. However, the output function is not easily deduced from the circuits, and also the connection is complex so that bugs tend to be hidden in the implementation. For these two reasons, we need to verify the above structure against a behavioral model. In the following section, we give our methods to verify the circuit by equivalence checking.

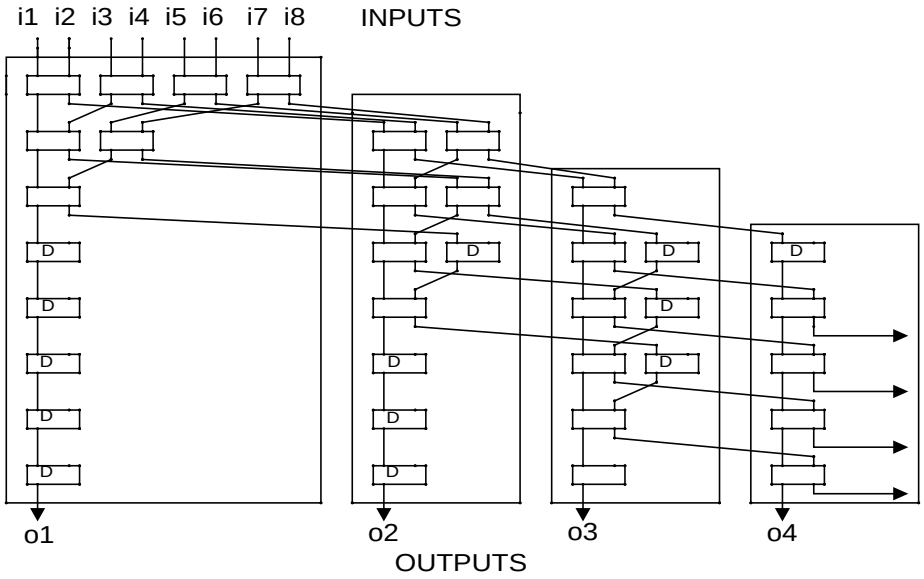


**Figure 3: Structure of the 8-input/4-output concentrator**

To verify the structure of the Knockout concentrator in VIS, we first modeled it in Verilog, which is the input language of the VIS tool. In the Appendix, we give the whole Verilog code of the structural (RTL) model of the concentrator.

## 5. Knockout Switch Concentrator Behavior

Although the structure of the Knockout concentrator (Figure 3) seems complicated, its specification is simple: if there is only one active cell in the input ports, it will be transmitted to the left most output port; if there are only two active cells in the input ports, they will be transmitted to the two left most output ports; if there are three active cells in the input ports, they will be transmitted to the three left most output ports; if there are four or more than four, they will be transmitted to all the four output ports. This behavior can be described as shown in Figure 4.

In Figure 4, *i1*, *i2*, …, *i8* are the 8 inputs of the concentrator. *ik* = 1 means there is a cell at input *k* (*k* =1, 2, …, 8), and *ik* = 0 indicates there is no cell at input *k*. The signal *sum* means the sum of the 8 inputs, so the value of *sum* indicates the number of inputs which have cells to send. Because there are 8 clock cycles delay in the structural model of the Knockout concentrator (Figure 4), we also give 8 clock cycles delays in the behavioral model to synchronize both models.

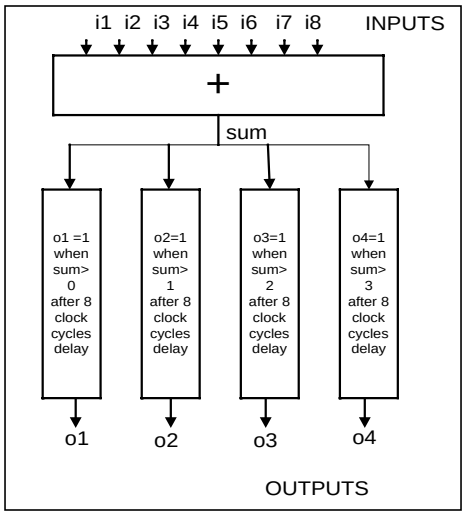


**Figure 4: Behavior of the 8-input/4-output concentrator**

For the verification in VIS, we also modeled the above behavior of the concentrator in Verilog (see Figure 5 below, where the notation "x ? y : z" means "if x then y else z").

```
module ko_behav (i1, i2, i3, i4, i5, i6, i7, i8, o1, o2, o3, o4);
input i1, i2, i3, i4, i5, i6, i7, i8;
output o1, o2, o3, o4;
wire [3 : 0] sum;

assign sum = i1 + i2 + i3 + i4 + i5 + i6 + i7 + i8;
assign o1 = (sum < 1) ? 0 : 1;
assign o2 = (sum < 2) ? 0 : 1;
assign o3 = (sum < 3) ? 0 : 1;
assign o4 = (sum < 4) ? 0 : 1;
endmodule
```

**Figure 5: Verilog model of the concentrator behavior**

## 6. Verification by Equivalence Checking

The Knockout concentrator is a sequential design containing about 64 latches (1-bit registers/flip-flops). Hence we started with a sequential equivalence checking in VIS between the behavioral and structural Verilog models. However, we could not get any result because the verification crashed due to a failure in allocating memory (out-of- memory error). This experiment was conducted on a Sun UltraSparc (300MHz/512MB) workstation.

After close inspection, we noticed that the latches in the structural Knockout concentrator are only for distributing the operation which could be done with one clock cycle instead of eight clock cycles so that the clock frequency of the whole system could be higher. In other words, these latches do not have the function of state transition. Therefore, reducing these latches will not have an influence on the function of the design.

Once the latches were removed from the Knockout concentrator, it becomes a combinational circuit. Hence it is now possible to apply combinational equivalence checking. Practically, it is fairly easy to reduce the latches in a Verilog module. In the structural Knockout concentrator description, we only need to modify the two submodules: 2x2 switch element and D-flip-flop while keeping the same concentrator structure. In the behavioral Verilog module, we only need to remove the eight clock cycles delay.

The combinational equivalence checking on Knockout concentrator was very efficient, and the verification in VIS succeeded. Table 1 shows the experimental results for the combinational equivalence checking in VIS, including CPU time, memory usage and number of BDD nodes allocated. This experiment was conducted on the same workstation indicated above.

**Table 1: Combinational equiv. checking results in VIS**

| CPU time (sec.) | Memory usage (MB) | Nodes allocated (K) |
|---|---|---|
| 32 | 45 | 32,476 |

From Table 1, it is interesting to notice that the combinational equivalence checking in VIS used little CPU time and memory to verify the Knockout concentrator while the sequential equivalence checking of the same models failed running out of memory. This is actually illustrating the reason why sequential equivalence checking has not been adopted in industry while combinational equivalence checking is routinely used in checking the correctness of synthesized models.

## 7. Conclusions

This paper introduced combinational equivalence checking and sequential equivalence checking as means for the verification of digital designs. Combinational equivalence checking can handle a relatively large design, and with the help of some techniques such as latch mapping, it also can partially verify a sequential circuit.

In particular, we described the design, modeling and verification of the concentrator of a Knockout ATM switch fabric using Verilog HDL and the VIS tool. We developed models for both the structure and behavior of an 8 input/4 output Knockout switch concentrator. We then used equivalence checking within VIS to verify the structure of the concentrator against its behavioral model. This was made possible through the transformation (via latch-reduction) of the sequential design into a combinational one. Hence, we turned the equivalence checking problem into a pure combinational equivalence checking.

Since the 8 input/4 output Knockout concentrator is relatively simple, we did not find any errors in the design model. However, the advantage of the equivalence checking approach proposed in this work would be more obvious if we design and verify a 32/16 or a 64/32 Knockout concentrator.

## 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] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677-691, August 1986.

[3] F. Corella, et. al, "Multiway Decision Graphs for Automated Hardware Verification," *Formal Methods in System Design*, Vol. 10, pp. 7-46, February 1997.

[4] O. Coudert, C. Berthet, J. Madre, "Verification of Synchronous Sequential Machines using Boolean Functional Vectors," *Proc. IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, Leuven, Belgium, pp. 111-128, November 1989

[5] D. Ginsburg, "*ATM Solutions for Enterprise Internetworking*", Addison Wesley, 1996.

[6] J. Jain, A. Narayan, M. Fujita, and A. Sangiovanni-Vincentelli, "Formal Verification of Combinational Circuits," *VLSI Design*, 1997.

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

[8] 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.

[9] I. Leslie and D. McAuley, "Fairisle: An ATM Network for the Local Area," *ACM Communication Review*, Vol. 19, No. 4, pp. 327-336, September, 1999.

[10] S. Tahar et. al., "Modeling and Verification of the Fairisle ATM Switch Fabric using MDGs," *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 18, No. 7, pp. 956-972, July 1999.

[11] Y.S. Yeh, M. Hiuchyj, and A. Acampora, "The Knockout Switch: a Simple Modular Architecture for High Performance Packet Switching," *IEEE J. Selected Areas in Communications*, Vol. SAC-5, No. 8, pp. 1274-1283, October 1987.

[12] M.H. Zobair and S. Tahar, "On the Modeling and Verification of a Telecom System Block Using MDGs," Technical Report, Concordia University, Department of Electrical and Computer Engineering, December 2000.

## Appendix: Verilog Model of the Concentrator Structure

```
module ko_struct (i1, i2, i3, i4, i5, i6,,i7, i8, o1, o2, o3, o4);

input i1, i2, i3, i4, i5, i6, i7, i8;
output o1, o2, o3, o4;

wire m11, m12, m13, m14, m15, m16, m17, m18;
wire m21, m22, m23, m24, m25, m26, m27, m28;
wire m31, m32, m33, m34, m35, m36, m37, m38;
wire m41, m42, m43, m44, m45, m46, m47, m48;
wire m51, m52, m53, m54, m55, m56, m57, m58;
wire m61, m62, m63, m64, m65, m66, m67;
wire m71, m72, m73, m74, m75, m76;
wire m85;

se se11(i1, i2, m11, m12);
se se12(i3, i4, m13, m14);
se se13(i5, i6, m15, m16);
se se14(i7, i8, m17, m18);
se se21(m11, m13, m21, m22);
se se22(m15, m17, m23, m24);
se se23(m12, m14, m25, m26);
se se24(m16, m18, m27, m28);
se se31(m21, m23, m31, m32);
se se32(m25, m27, m33, m34);
se se33(m22, m24, m35, m36);
se se34(m26, m28, m37, m38);
dr dr41(m31, m41);
se se42(m33, m35, m42, m43);
dr dr43(m32, m44);
se se44(m37, m34, m45, m46);
dr dr45(m36, m47);
dr dr46(m38, m48);
dr dr51(m41, m51);
se se52(m42, m44, m52, m53);
se se53(m45, m47, m54, m55);
dr dr54(m43, m56);
se se55(m48, m46, m57, m58);
dr dr61(m51, m61);
dr dr62(m52, m62);
se se63(m54, m56, m63, m64);
dr dr64(m53, m65);
se se65(m57, m55, m66, m67);
dr dr71(m61, m71);
dr dr72(m62, m72);
se se73(m63, m65, m73, m74);
se se74(m66, m64, m75, m76);
dr dr81(m71, o1);
dr dr82(m72, o2);
dr dr83(m73, o3);
se se84(m75, m74, o4, m85);
endmodule
```

```
module se (i1,i2,o1,o2);
input i1, i2;
output o1, o2;
assign o1 = (i1 ==0) ? i2 : i1;
assign o2 = (i1 ==0) ? i1 : i2;
endmodule

module dr (i1,o1);
input i1;
output o1;
assign o1 = i1;
endmodule
```