Technical Report No. IRO-1021
IRO Dept., University of Montreal
Montreal, Quebec, Canada, March 1996

Behavioral Verification of an ATM Switch Fabric using Implicit Abstract State Enumeration

Michel Langevin[§], Sofiène Tahar, Zijian Zhou, Xiaoyu Song and Eduard Cerny

University of Montreal, IRO Dept., Montréal, Québec, H3C 3J7 Canada
E-mail: {tahar, zhouz, song, cerny}@iro.umontreal.ca

[§] GMD-SET, Schloss Birlinghoven, 53757 St. Augustin, Germany
E-mail: langevin@borneo.gmd.de

Abstract. We investigate the equivalence checking of the RTL hardware implementation of the Cambridge Fairisle Asynchronous Transfer Mode (ATM) 4 by 4 switch fabric against a high-level behavioral specification which has no restrictions with respect to the frame size, cell length or word width. The verification is based on the reachability analysis of the product machine of the implementation and the specification, both modeled as Abstract State Machines (ASM). Multiway Decision Graphs (MDG) are used to encode both the output and transition relations of ASMs and the set of reachable abstract states, allowing implicit abstract state enumeration. Since MDGs avoid model explosion induced by data values, this experiment confirms the effectiveness of the MDG-based verification methodology as an extension of ROBDD-based approaches.

1 Introduction

Design errors in network components may have disastrous effects, in particular if networks are used in safety-critical applications where communications errors could cause loss of life. Simulation is traditionally used for checking the correctness of the systems. However, it is practically impossible to achieve exhaustive simulation for complex communication systems. Therefore, formal verification of the correctness of digital systems is thus gaining interest, as the correctness of a formally verified design implicitly involves all possible input values [9].

ATM (*Asynchronous Transfer Mode*) has been considered as an appropriate network technology to address the variety of needs for new high-speed, high-bandwidth applications. Although ATM is being hailed as the most important communication mechanism of the foreseeable future, there is currently little experience on the application of formal verification to ATM network hardware.

In this paper, we present our results on formally verifying an ATM network component using a new class of decision graphs, called *Multiway Decision Graphs* (MDG) [3]. The device we verified is the Fairisle 4 by 4 switch fabric which is part of the Fairisle ATM network [11] designed and in use at the Computer Laboratory of the University of Cambridge. It was fabricated with no consideration for formal verification. The main contributions of our work are the specification of the expected behavior of the ATM network switching fabric, and its equivalence checking against the hardware implementation. The high-level behavioral specification and thus the verification have no restrictions with respect to the frame size, cell length or word width. The verification is based on reachability analysis of the product machine of the implementation and the specification,

modeled as networks of Abstract State Machines (ASM) [3]. Multiway Decision Graphs (MDG) are used to encode the output and transition relations of ASMs and the set of reachable abstract states, allowing implicit abstract state enumeration. This experiment confirmed the effectiveness of the MDG-based verification methodology as an extension to ROBDD-based approaches [1], since model explosion induced from data values expanded to their binary form is largely avoided. The verification was performed automatically in a reasonable amount of time. Our results demonstrate the feasibility of applying the MDG-based verification method to a real hardware design.

The organization of this paper is as follows: in Section 2, we review some related work on the formal verification of ATM hardware. In Section 3, we describe the behavioral specification of the switch fabric as a state machine and present the descriptions of the hardware implementation. In Section 4, we show how to model the specification and the implementation as abstract state machines (ASM) represented by MDGs. Using the implicit abstract state enumeration, we present in Section 5, the equivalence proof of the behavioral specification against the RTL hardware description. In Section 6, we report experimental results and then conclude the paper in Section 7.

2 Related Work

Although the principles of ATM are viewed as the base technology for the next generation of global communications, there exist few published results addressing formal verification of ATM related circuits. To our best knowledge, there are only two such cases in the open literature.

P. Curzon [4] formally verified the 4 by 4 fabric of the Fairisle switch using the HOL theorem prover [8]. He hierarchically verified each of the modules used in the hardware design of the switching element, by describing the behavioral and structural specifications down to the gate level, and then proving the related correctness theorems in HOL. The separate proofs were then combined to prove the correctness of the whole switch fabric.

B. Chen, et. al at Fujitsu Digital Technology Ltd. [2] exploited symbolic model checking to detect a design error in an ATM circuit. The ATM circuit is composed of about 111K gates and supports high-speed switching operations at 156 MHz. Using the SMV (Symbolic Model Verifier) [10], they identified the design error by checking some properties described in CTL (Computational Tree Logic) [10]. With the restriction of the Boolean representation in SMV and to avoid state space explosion, they abstracted the data width of addresses from 8 bits to 1 bit, and the number of addresses in the WAF (Write Address FIFO) from 168 to 5. However, in some cases, a property could not be checked because of this reduction and a more detailed (gate-level) datapath model was needed for certain blocks to pinpoint the source of the error. Although the design error was diagnosed, there is no proof showing that the abstracted circuit was itself correct.

The Curzon's work used HOL which is interactive and requires lots of expertise to guide the verification process [9]. The work at Fujitsu Ltd. used a model checker which is automatic, but the required data abstraction (e.g., using 1 bit to represent 8-bit data width) for avoiding the state explosion is not always applicable.

To overcome these drawbacks, we attempt to raise the level of abstraction of automated verification methods to that of interactive methods, without sacrificing automation. The MDGs [3] are based on a subset of a many-sorted first-order logic with abstract sorts and uninterpreted function symbols. They subsume the class of Bryant's Reduced Ordered Binary Decision Diagrams (ROBDD) [1]. In our previous work [12], we produced the description of the Fairisle switch fabric hardware at two different levels of abstraction—the original gate-level

implementation and an abstract RTL implementation which holds for any arbitrary word width n. We then performed the verification of the RTL hardware against the gate-level implementation where the generic word size was aligned to be 8-bit wide. In this paper we report on the equivalence checking of the hardware implementation of the ATM network switch fabric against a high-level behavioral specification, with no restrictions on the frame size, cell length or word width.

3 The Fairisle ATM Switch

The Fairisle switch [11] consists of three types of components: *input port controllers*, *output port controllers* and a *switch fabric* (Figure 1).

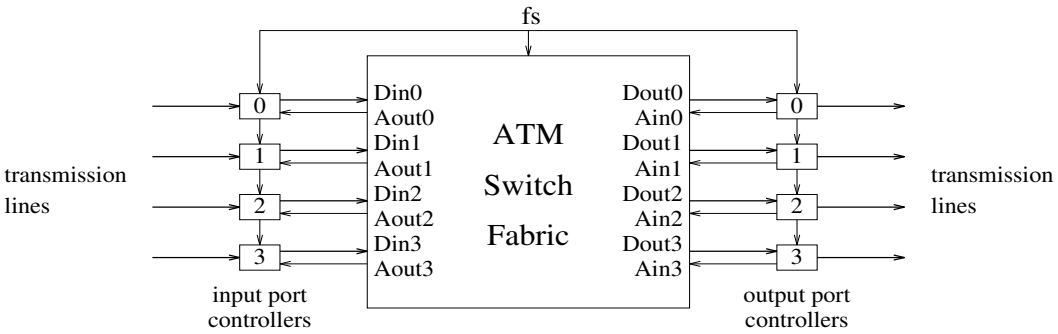


Figure 1: The Fairisle ATM switch

The port controllers synchronize and process incoming and outgoing data cells, appending control information in the front of the cells in a *routing tag* (Figure 2). This tag is stripped off before the cell reaches the output stage of the fabric. A cell consists of a fixed number of data bytes which arrive one at a time. The fabric switches cells from the input ports to the output ports according to the routing tag. If different port controllers inject cells destined for the same output port controller (indicated by the *route* bits) into the fabric at the same time, then only one will succeed. The others must retry later. The routing tag also includes priority information (*priority* bit) that is used by the fabric for arbitration: high priority cells are given precedence, and the final choice is made on a *round-robin* arbitration basis. The input controllers are informed of whether their cell was successful using acknowledgment lines. The fabric sends a negative acknowledgment to the unsuccessful input ports, but passes the acknowledgment from the requested output port to the successful input port. The port controllers and switch fabric all use the same clock, hence bytes are received synchronously on all links. They also use a higher-level cell frame clock—the *frame-start* signal (*fs*). It ensures that the port controllers inject data cells into the fabric synchronously so that the routing tags (bytes) arrive at the same time. In this paper, we are concerned with the verification of the switch fabric which is the core of the Fairisle ATM switch.

Figure 2: The routing tag (header) of a Fairisle ATM Cell

3.1 Derivation of a behavioral specification

Starting from a set of timing-diagrams describing the expected input-output behavior of the switch fabric, we derive a complete behavior specification in the form of a finite state machine. Although an implementation of the ATM switch fabric already existed, it was done by one of the co-authors without consulting the implementation, i.e., the specification would be the same even if it were described before any implementation. The state machine of the specification consists of a complete description of the expected behavior of the fabric, including the assumptions on the environment.

The behavior of the switch fabric is cyclic. In each cycle or *frame*, it performs the following actions: The switch waits for cells to arrive. Once they arrive, it reads the headers, and establishes paths to the appropriate output ports and produces the acknowledgments.

The cells from all the input ports start when a particular bit (the *active* bit, Figure 2) of any one of them goes high. The fabric does not know when this will happen. However, all the input port controllers must start sending cells at the same time within the frame. If no input port raises the active bit throughout the frame then the frame is *inactive*—no cells are processed. Otherwise it is *active*.

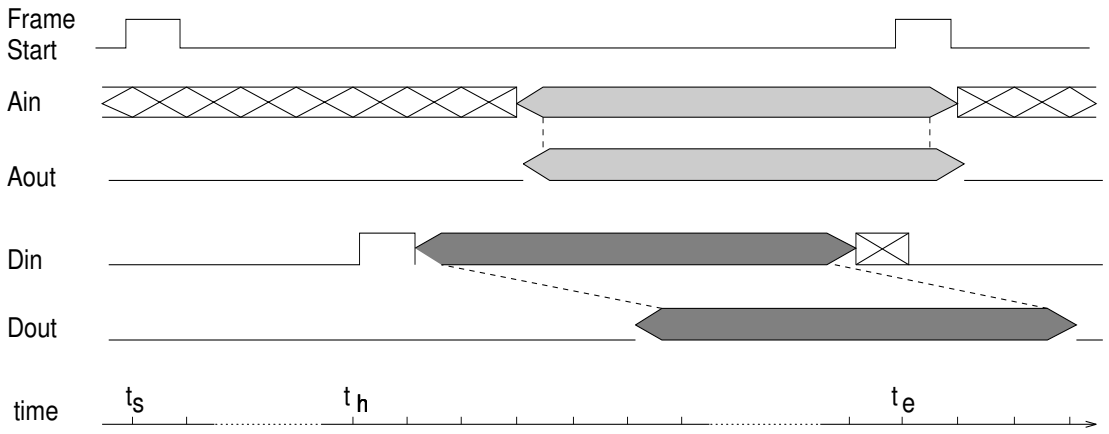


Figure 3: Expected behavior during an active frame

A timing-diagram on the expected input-output behavior of the switch fabric during an active frame is given in Figure 3. After the frame starts (at time t_s), the switch waits for the headers that will appear on the data input lines *Din*. After the arrival of the headers (at time t_h), an arbitration is done in at most 2 cycles. The successful cells (following bytes of the headers on *Din* lines) are transferred to the corresponding output ports (*Dout* lines) with a delay of 4 cycles, while acknowledgments traverse in the inverse direction, without synchronous delay, starting at time t_h+3. Notice that the last cycle of a frame (at time t_e-1) does not transfer data. When there is no data or acknowledgment to transfer, the switch forces *zero* values on output data lines (thus, the value of *Ain* are don't care).

The state machine representation in Figure 4 is inspired by the top-level behavioral description of the switch fabric as given in the design documentation (in the form of timing diagrams). The description is based on the following four assumptions on the environment of the fabric:

1. At start up (t_0) the frame-start (t_s) waits at least two cycles before going high, i.e., $t_s \geq t_0$+2
2. Headers arrive (t_h) at least three cycles after frame-start (t_s), i.e., $t_h > t_s$+2
3. Next frame-start arrives (t_e=t'_s) at least three cycles after frame-start (t_s), i.e., $t_e > t_s$+2
4. Headers arrive (t_h) at least three cycles before next frame-start (t_e=t'_s), i.e., $t_e > t_h$+2

In Figure 4, there are 14 conceptual states in the machine. Some symbols are used to keep the presentation simple. The symbols *s* and *h* denote a frame-start (*fs* = 1) and the arrival of headers (active bit set in at least one *Din*), respectively. The symbol "~" denotes negation. The symbols *a*, *d* or *r* inside a conceptual state represent the computation of acknowledgment output (*Aout*), data output (*Dout*) or round-robin arbitration, respectively; they are part of all transitions emerging from the state. Note that the absence of an acknowledgment or data symbol in a conceptual state means that the default value of 0 is output.

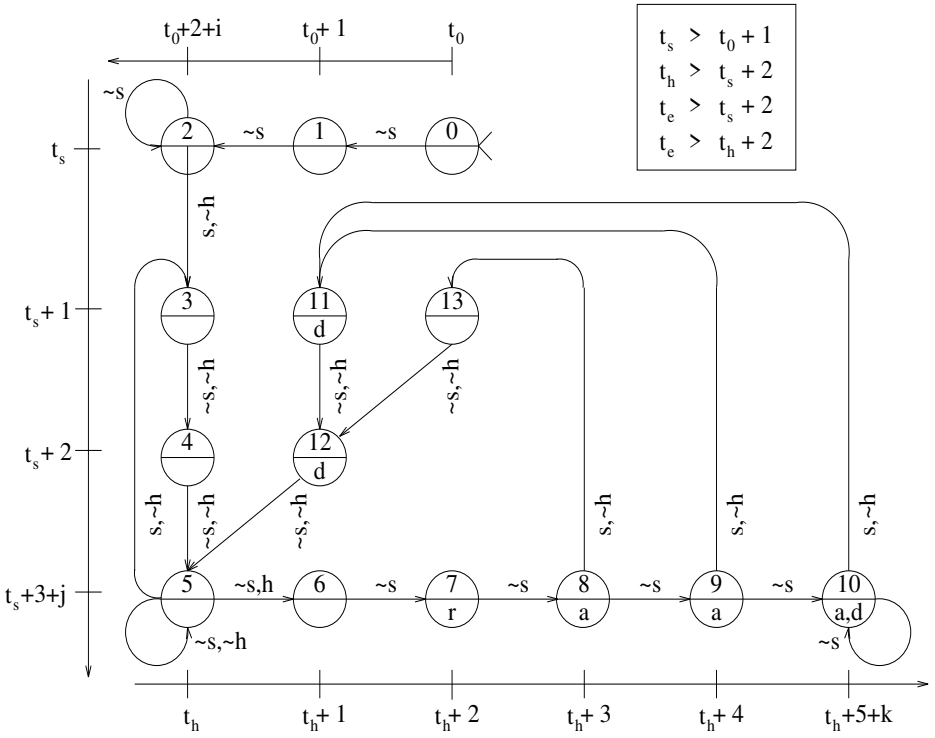


Figure 4: The expected Fairisle switch fabric behavior

Three time axis illustrate the time units of a frame to which the transitions correspond. The symbols t_0, t_s and t_h in the axis represent the initial time, the arrival time of a frame-start signal and the arrival time of a header, respectively. Note that the ending time (t_e) of a frame is not given, since it is the same as the time t_s of the next frame.

State 0 is the initial state from which there must be two time steps without the arrival of the frame-start signal (states 1 and 2). This complies with the first constraint on the environment of the switch. The three states 0, 1 and 2 are related to the time axis t_0. The waiting loop for the first frame-start signal in state 2 is shown in the figure by a natural number *i*.

States 3, 4 and 5 describe the behavior of the switch after the arrival of a frame-start, with at least a three-cycle delay before the arrival of either the headers or the next frame-start. These are the second and third constraints on the environment. The states 3, 4 and 5 are related to the time axis t_s. The waiting loop for the arrival of either the headers or the next frame-start in state 5 is shown in the figure by a natural number *j*. Note that the arrival of the next frame-start corresponds to the end of an empty frame.

States 6 to 13 describe the behavior of the switch fabric after the arrival of the headers. When the headers arrive, the frame-start signal must not arrive before at least three cycles to comply with the last constraint on the environment. The states 6 to 10 are related to the time axis t_h. After arbitration is done (state 8), the switch transfers the acknowledgments in each cycle of a frame, and switches data delayed by two cycles. This delay is represented using the sequence of transitions from state 8 to 10. The self loop in state 10 represents the transmission of data and acknowledgments in the remaining cycles of the cell (illustrated in the figure by a natural number k). The arrival of a frame-start signal in states 8, 9 or 10 marks the beginning of another frame period. Here, a new sequence of state transitions along the t_s axis progresses similarly in states 3, 4 and 5 described above, but considering possibly different scenaria for completing the transmission of preceding cells.

To compute the acknowledgment outputs, the data outputs and the round-robin arbitration, the following state variables are parts of the conceptual states:

- co_i (i= {0,…,3}) of type {0,1}: co_i is 1 iff the output port i is connected.
- ip_i (i={0,…,3}) of type {0,…,3}: ip_i is the input port connected to the output port i (during arbitration, it is the last input port connected to the output port i).
- $sr_{i,j}$ (i={0,…,3}; j={1,…,4}) of type {0,…,255}: $sr_{i,j}$ is the value of Din_i delayed by j clock cycles. That is, during each transition of the state machine, the data input Din_i is shifted in.

In states annotated by a (8, 9 and 10) the value of $Aout_i$, i={0,…,3} is computed as follows:

if ((co_0=1) **and** (ip_0=i)) **then** ($Aout_i$=Ain_0)
ef ((co_1=1) **and** (ip_1=i)) **then** ($Aout_i$=Ain_1)
ef ((co_2=1) **and** (ip_2=i)) **then** ($Aout_i$=Ain_2)
ef ((co_3=1) **and** (ip_3=i)) **then** ($Aout_i$=Ain_3)
else ($Aout_i$=0)

where **ef** means **else if**. That is, if input port i is connected, $Aout_i$ takes the value of the corresponding Ain_j; otherwise, $Aout_i$ is 0. In states annotated by d (10, 11 and 12), the value of $Dout_i$, i={0,…,3} is computed as follows:

if (co_i=0) **then** ($Dout_i$=0)
ef (ip_i=0) **then** ($Dout_i$=$sr_{0,4}$)
ef (ip_i=1) **then** ($Dout_i$=$sr_{1,4}$)
ef (ip_i=2) **then** ($Dout_i$=$sr_{2,4}$)
else ($Dout_i$=$sr_{3,4}$)

That is, if the output port i is connected, $Dout_i$ takes the value of the corresponding Din_j delayed by 4 cycles; otherwise, $Dout_i$ is 0. The values of co_i and ip_i are modified only during arbitration, i.e., during the transition from state 7 to state 8; each (co_i, ip_i) value-pair is computed from the values of all $sr_{j,2}$ (the cell headers), considering the active, priority and route fields, and the current value of ip_i for the round-robin arbitration. This computation can be described as a case analysis using **if-then-else** constructs, but it is too long to be shown here.

3.2 The switch fabric hardware implementation

Figure 5 shows a block diagram of the switch fabric implementation. It is composed of an arbitration unit (timing, decoder, priority filter and arbiter), an acknowledgment unit and a dataswitch unit. The timing block controls the timing of decisions with respect to the frame-start signal and the arrival time of the headers. Arbitration is implemented in two stages. The decoder reads the routing tags of the cells and decodes the port requests and priorities. The priority filter filters out the requests with low priority and those from inactive inputs, and passes the actual request situation for each output port to the arbiters. The arbiters (in total four—one for each port) make arbitration decisions for each output port i by setting values for the corresponding $outDis_i$, $xGrant_i$ and $yGrant_i$ Boolean signals (12 outputs of arbiter in Figure 5). The dataswitch unit performs the actual switching of data from input ports to output ports (a set of registers is used in the dataswitch, and they are reset when their corresponding *outDis* signal is 1). The ack unit passes appropriate acknowledgments to the input ports. Remark that the pair of $xGrant_i$ and $yGrant_i$ signals corresponds to the ip_i variable of the specification, but this simple relation does not hold between the $outDis_i$ signals and the specified co_i variables.

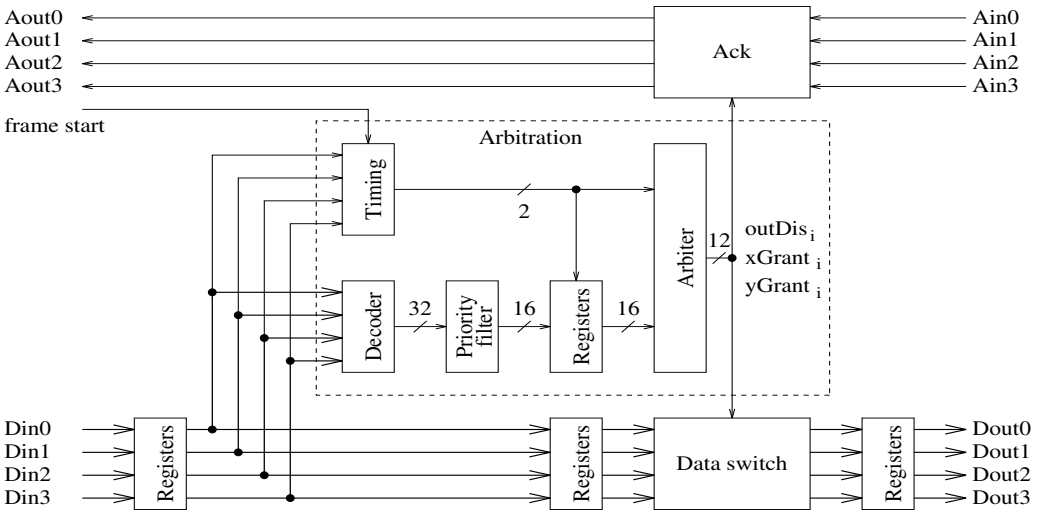


Figure 5: A Fairisle switch fabric implementation

Each of these units is repeatedly subdivided until eventually the logic gate level is reached, providing a hierarchy of modules. The hardware design has a total of 441 basic components where a basic component is a logic gate with two or more inputs, or a 1-bit flip flop. The switch fabric is built on a 4200 gate Xilinx FPGA. The switching element can be clocked at 20 MHz and currently frame-start pulses occur every 64 clock cycles. The design was described in Qudos HDL [6] hardware description language which was used for generating the Xilinx netlist. The Qudos simulator [6] was used to perform the original (non-formal) validation.

4 Behavioral Representation using Multiway Decision Graphs

In the previous sections we presented a specification of the expected behavior of the ATM switch fabric and a gate-level implementation. To compare these two descriptions, we need to model their behaviors. The conventional method is to use finite state machines and represent them using Reduced Ordered Binary Decision Diagrams (ROBDD) [1]. However, the presence of data in both

descriptions (16 8-bit wide state variables in the specification and 20 registers of the same width in the implementation) makes the procedure very explosive and difficult. To alleviate the problem, we use *Multiway Decision Graphs* (MDG) [3], proposed as a new class of decision graphs.

Multiway Decision Graphs subsume ROBDDs and accommodate abstract types and uninterpreted function symbols, while providing and exploiting structure sharing. The formalism underlying MDGs is a many-sorted first-order logic with a distinction between *abstract sorts* and *concrete sorts*. Concrete sorts have *enumerations*, while abstract sorts do not. A data value can be represented by a single variable of abstract sort, rather than by a vector of Boolean variables, and a data operation can be represented by an *uninterpreted function* symbol. We distinguish two kinds of function symbols (operators): *abstract operators* which are used for data operations, e.g. addition of two abstract variables; *cross- operators* which are used for feed-back from datapath to control circuitry, e.g. a comparator *equal*. For more details, see [3] or [13].

For circuits with large datapaths, as the ATM switch fabric, MDG-based modeling is much more compact than using ROBDDs. It allows us to consider the data input, state and output variables of a state machine as values of an abstract (i.e., non-specified) sort. For instance, the 8-bit-wide signals in both ATM descriptions can be described as values of an abstract sort (*wordn)*. The output and next state relations of the state machine can be encoded in MDGs, similar to ROBDDs [3]. This MDG-based modeling for *Abstract State Machine* (ASM) [3] can represent an unbounded class of FSMs, depending on the interpretation of the abstract sorts and operators.

In the following, we show how to model the specification and implementation of the ATM switch fabric as ASMs using MDGs. The two descriptions are in a prolog-style MDG-HDL, supported by the MDG software package. MDG-HDL allows the description of hierarchical hardware structures using module constructs and comes with a large library of predefined basic components (such as logic gates, multiplexors, registers, bus drivers, ROMs, etc.). For behavioral specifications, MDG-HDL allows the use of high-level constructs as ITE (**if-then-else**) formulas, CASE formulas or tabular representations. A translator from a subset of VHDL is under development.

4.1 MDG-ASM specification model

We must first define some basic sorts and operators:

- concrete sort *bool* = {0,1}
- concrete sort *port* = {0,…,3}
- concrete sort *Ctl* = {0,…,13}
- abstract sort *wordn* (representing data bytes)
- generic constant *zero* of sort *wordn* (representing the default 0 8-bit value)
- cross-operator *act* of type [*wordn* → *bool*] (representing the active field of a Byte)
- cross-operator *pri* of type [*wordn* → *bool*] (representing the priority field of a Byte)
- cross-operator *rou* of type [*wordn* → *port*] (representing the route field of a Byte)

The ASM is composed of

1) the input variables *fs*, Ain_i (i = {0,…,3}) of sort *bool*, and Din_i (i = {0,…,3}) of sort *wordn*;

2) the output variables $Aout_i$ (i = {0,…,3}) of sort *bool* and $Dout_i$ (i = {0,…,3}) of sort *wordn*;

3) the state variables *c* of sort *Ctl*, co_i (i= {0,…,3}) of sort *bool*, ip_i (i={0,…,3}) of sort *port*, and $sr_{i,j}$ (i={0,…,3}; j={1,…,4}) of sort *wordn*.

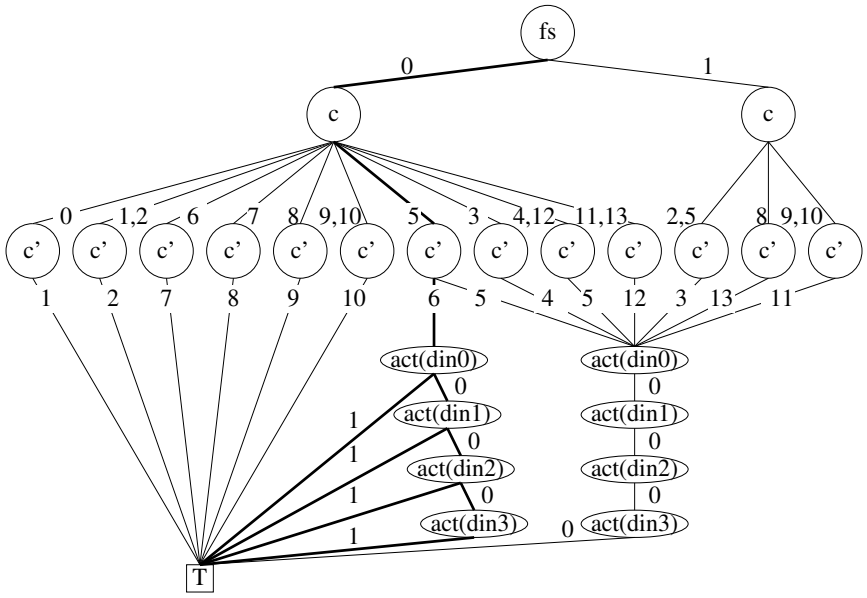


Figure 6: The MDG associated with the next-state value of *c*

The description of the ASM is completed by giving its output and next-state relations. Similar to ROBDD-based FSM modeling, an MDG is associated with each output and next-state variable, encoding its value as function of the input and state variables. For instance, the MDG associated with the next-state variable *c'* (next value of *c*) is in Figure 6: The transition from state 5 to state 6 under the meta-symbols ~*s* and *h* of Figure 4 is encoded by the set of highlighted paths ($fs = 0$) and ($c = 5$) and ($c' = 6$) and at least one ($act(Din_i) = 1$) (representing the arrival of a header). Remark that the formula represented by the set of MDG paths is similar to the one represented by the set of ROBDD paths leading to the true leaf, excepted that first-order terms can appear along the paths. The terms $act(Din_i)$ are the *cross-terms*, and they encode data-dependent decisions. The MDG associated with the output variable $Dout_0$ is in Figure 7. $Dout_0$ is equal to the corresponding $sr_{i,4}$ value, depending on ip_0, if the output port 0 is connected ($co_0 = 1$) and if the conceptual state *c* is 10, 11 or 12; otherwise, $Dout_0$ equals *zero*.

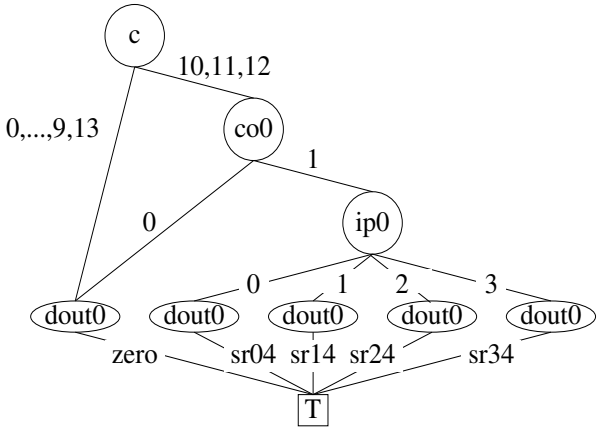


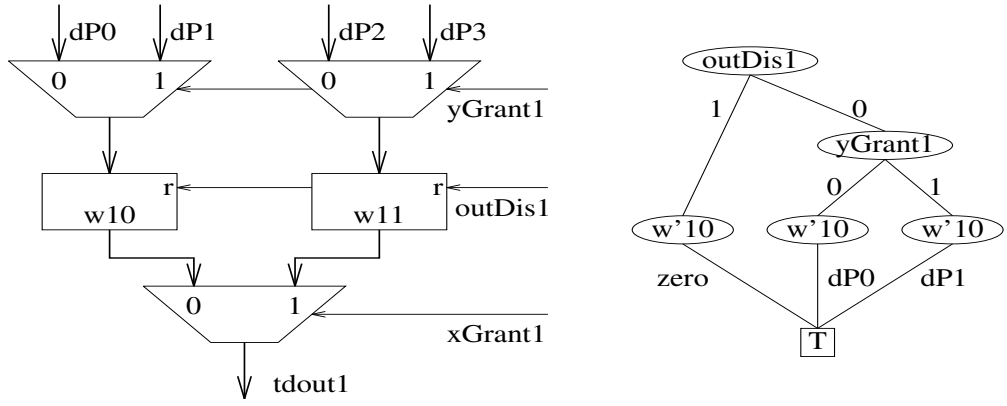
Figure 7: MDG associated with the value of $Dout_0$

The MDGs of the other output and next-state variables can be derived in a similar way.

4.2 MDG-ASM implementation model

To formally verify the fabric using MDGs, we first translated the description into very similar descriptions using our HDL (MDG-HDL), yielding a gate-level description. We also provided an RTL description of the switch implementation. Here, all byte signals are modeled in MDG-HDL as signals of abstract sort *wordn*, and the active, priority and route fields are accessed using the cross-operators *act*, *pri* and *rou*, respectively. The dataswitch unit is implemented using simple data multiplexors instead of collections of logic gates. This leads to a simpler abstract implementation of the dataswitch, reflecting the switching behavior in a more natural way and using a smaller network of components and signals. For more details about the abstraction techniques used, refer to [12].

Each RTL component has a behavioral model described by MDGs. Given the RTL netlist, an ASM-MDG model of its behavior is extracted by composing, for each output or register, the MDGs of the components in its cone of influence, and abstracting the internal signals [13], similarly as extracting FSM-ROBDD models from gate-level netlist. For instance, Figure 8 (a) presents the RTL netlist for the word-slice of the dataswitch component that computes the output data for port 1. The dP_i signals come from the registers at the input of the dataswitch (Figure 5), corresponding to the data inputs of two cycles before. The output of the dataswitch is fed into registers before reaching the output of the fabric, corresponding to the outputs of one cycle before. The registers inside the dataswitch, e.g., w_{10} and w_{11} in Figure 8 (a), are used to partially compute the output, given the value of $yGrant_i$ (selection between odd or even input ports). The selection is completed later when the value of $xGrant_i$ is known. However, if $outDis_i$ is 1, the corresponding intermediate registers are forced to *zero*. Given the descriptions of word-wide multiplexors and word-wide registers with synchronous reset, the extracted MDG behavioral model for w'_{10} (next value of w_{10}) is shown in Figure 8 (b).



(a) RTL schematic (b) MDG model for w10

Figure 8: RTL implementation and behavior of a dataswitch component part

The MDGs associated with the other registers and the output signals are obtained in a similar way.

5 Formal verification of the Switch Fabric Behavior

Two ASMs are behaviorally equivalent if and only if they produce the same output values for all input sequences. We feed the machines of the behavioral specification and RTL implementation with the same input sequences and check the equivalence of their outputs using reachability

analysis of the product-machine (iterative fixed-point computation of the reachable states). As with ROBDDs, we use MDGs to represent the sets of states, except that they are abstract, and perform implicit abstract state enumeration automatically using graph-based manipulations of MDGs (conjunction, existential abstraction, renaming, and what we call pruning-by-subsumption to approximate difference) [3]. MDG-based reachability analysis, however, avoids state explosion induced from data values, since the verification is independent of the datapath width.

For the product of the ASMs presented in Sections 4.1 and 4.2, an MDG representing a set of total-states encodes a relation between 39 concrete state variables and 36 abstract state variables. The relation may depend on data values, encoded using cross-terms. For instance, parts of the MDG encoding the set of total-states reached at the 9^{th} iteration of the reachability analysis (i.e., all the states reachable one transition after arbitration) are depicted in Figure 9.

In comparison with ROBDDs, 8 Boolean variables would be needed for each abstract state variable of the MDGs (i.e., 288 Boolean variables for data). Due to the fact that in the set of reachable states many non-disjoint combinations of data state variables have the same values, it is not possible to interleave these Boolean variables to avoid explosion caused by the binary encoding of data values. Instead, in MDGs, the encoding is performed using abstract data, yet isomorphic graph sharing is exploited as in ROBDDs (see the left hand side of Figure 9). Since abstract data are used, decisions on their values are encoded by cross-terms which also contribute nodes in the MDGs (Figure 9). Although cross-terms add complexity to the graph structure in general, the overhead is much smaller than the explosion induced from encoding data in binary form.

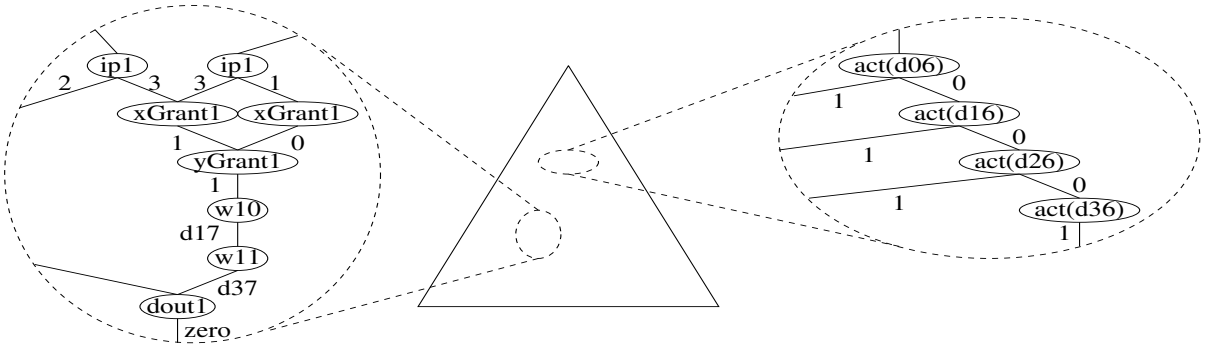


Figure 9: MDG encoding a set of reachable abstract states

Using abstract reachability analysis, the verification of the behavioral model is done for an arbitrary word width *n* and any frame size and cell length that respect the environment assumptions of the specification. In a previous work [12], we have accomplished the verification of the abstract RTL description against the gate-level one where the data width of the abstract model is instantiated to 8 bits using *uninterpreted functions* which encode and decode abstract data to Boolean data and vice-versa. By combining these two verification results, we obtain a complete verification from high-level behavior down to the gate-level implementation.

6 Experimental Results and Discussion

We report the experimental results of comparing the ASM specification model of the Fairisle switch fabric against the ASM RTL hardware model using the MDG-based techniques. The prototype of the MDG package is implemented in Quintus Prolog. The experiments were

performed on a SPARC station 10 of 128 MB memory. Table 1 shows the size and the CPU time for some significant MDGs generated during the reachability analysis.

Table 1: Size and generation time of some MDGs

MDG	#nodes	#paths	CPU time (sec.)
q_0	71	256	1
q_7	8321	1675264	80
v_7	8572	1677824	5
q_8	11665	128128	150
v_8	20225	1805952	10
q_9	25985	256256	300
v_9	46160	2062208	10
q_{10}	4995	60360	1100
v_{10}	51106	2122568	60
q_{11}	1	1	60

The MDG q_j encodes the frontier set of reachable states at the end of iteration j, while v_j the set of all reached states. q_0 is the MDG encoding the initial total states. There are 256 paths because the initial values of each ip_i can vary. Note that, due to subgraph sharing, all these paths are encoded using only 71 nodes. The MDG q_7 encodes the frontier set of states after the first arbitration phase is performed in the implementation (i.e., at $t_h + 2$), while q_8 encodes those after the arbitration is completed in both machines (i.e., in which acknowledgments should start to be computed). q_9 and q_{10} encode states where the frame may be terminated. q_{11} is the false MDG (representing the empty set) meaning that all reachable states have been visited. Output values of both machines are compared at each iteration, which took up to 100 sec. in some cases. Since no differences were detected, both ASMs are equivalent.

Table 2 summarizes the experimental results of the equivalence verification. It is apparent that a large set of states are encoded because of MDG's abstract representation and graph sharing, which results in an acceptable number of nodes. This supports the conclusion that the verification would be impossible if ROBDD representation in the reachability analysis were used.

Table 2: Experimental results on behavioral verification

Equivalence verification	CPU time (sec.)	Memory (MB)	#nodes
Compiling	120	10	25417
Reachability analysis	2800	140	295139
Total	2920	150	320556

Our verification confirms the results done by P. Curzon using HOL [4], that no errors were discovered in the implementation of the switching element. In fact, the fabric has been in service for some time before it was verified by formal methods. To test the effectiveness of our approach, we experimented with three erroneous implementations: 1) we exchanged the inputs to the JK Flip-Flop that produces the $outDis_3$ signal; 2) we used priority information on input port 0 instead of input port 2; 3) we used an AND gate instead of an OR gate to produce the $Aout_0$ signal. The non-equivalence of each erroneous implementation with respect to the specification was detected automatically during the reachability analysis, and counter-examples were generated to help diagnose the errors. Table 3 shows the results for these three experiments including separate CPU time columns for performing the reachability analysis and for generating the counter-example,

respectively. These results manifest the practicality and the effectiveness of MDG-based verification methods.

Table 3: Verification of some faulty implementations

Experiment	rean (sec.)	counter-example (sec.)	#nodes	Memory (MB)
Error 1	11	9	2462	1
Error 2	850	450	150904	120
Error 3	600	400	147339	105

Comparison with the verification in HOL

While Curzon's behavioral description [4] is using the expressiveness of HOL to describe the behavior at the frame level, our specification is based on a state machine model which is much more readable and commonly used by designers to model circuit behavior than a set of logic formulas. We impose two more assumptions than Curzon on the environment of the switch fabric (the second and third ones in Section 3.1). These assumptions are parts of the informal description of the switch fabric, and permit to reduce by a factor of 2 the complexity of our automatic verification.

P. Curzon reported that approximately two man-months were needed to perform the verification of the whole fabric [4]. However, this time includes the verification of each of the modules and the verification of their combinations. Using the HOL theorem prover, much work was needed in proving a large number of lemmas and in setting-up the proof scripts interactively, e.g., the time spent for the verification of the dataswitch unit was about one week [4]. The proof script was about 530 lines long (17 KB). Using our state machine models and MDGs, the verification was achieved automatically without the need of any proof script, except the careful management of node ordering (similar to ROBDDs).

P. Curzon reported in [5] that the detection and correction of errors in one erroneous 4 by 4 switch fabric using HOL took 3 man-months and consumed a large portion of the verification time [5]. With our verification tool, errors are detected automatically and can be diagnosed with the help of the counter-example facility.

7 Conclusions and Future Work

We have shown the applicability of formal verification techniques based on a new class of decision graphs, the Multiway Decision Graphs, to a real circuit. We have thus demonstrated that formal verification of a real piece of communication hardware can be conducted automatically using the MDG tools. We investigated the equivalence checking of the RTL hardware implementation of the Fairisle ATM switch fabric against a high-level behavioral specification with no restrictions on the frame size, the cell length and the word width. The verification was based on the reachability analysis of the product machine of the implementation and the specification both given as Abstract State Machines (ASM). We found no errors in the current implementation. However, we were able to verify several faulty implementations and the injected errors were successfully identified. In a previous work [12], the abstract RTL model was verified equivalent to the original gate-level implementation where the n-bit words of abstract sort were aligned with 8 bits. The results of the current and previous work illustrate the practicability of such a complete formal verification down to the gate level using tools exploiting MDGs, a methodology that would be impossible in this case using only ROBDD-based reachability analysis.

The ROBDD-based symbolic model checking technique developed by K. L. McMillan [10] is widely used for the verification of safety as well as liveness properties. Since it requires a Boolean representation of the circuit, it must cope with the state explosion problem when the number of variables is large. MDGs have the ability of representing a data value with a single variable of abstract sort and hence are generally more compact than ROBDDs. We are in the course of developing a model checking algorithm for a restricted first-order temporal logic. This additional feature would allow us to perform the verification of properties of abstract specifications.

References

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

[2] Chen, B.; Yamazaki, M.; Fujita, M.: Bug Identification of a Real Chip Design by Symbolic Model Checking; *Proc. International Conference on Circuits And Systems (ISCAS'94)*, London, UK, June 1994, pp. 132-136.

[3] Corella, F.; Zhou, Z.; Song, X.; Langevin, M.; Cerny, E.: Multiway Decision Graphs for Automated Hardware Verification; To appear in the journal of *Formal Methods in System Design*. Available as IBM research report RC19676(87224), July 1994.

[4] Curzon, P.: The Formal Verification of the Fairisle ATM Switching Element; Technical Reports No. 328 & No. 329, University of Cambridge, Computer Laboratory, March 1994.

[5] Curzon, P.: Problems Encountered in the Machine-assisted Proof of Hardware; In: Joyce, J. and Seger, C. (Eds.), *Higher Order Logic Theorem Proving and Its Applications*, Lecture Notes in Computer Science 780, Springer Verlag, 1994.

[6] Edgcombe, K.: T*he Qudos Quick Chip User Guide*; Qudos Limited.

[7] Gordon, M.; Herbert, J.: Formal Hardware Verification Methodology and its Application to a Network Interface Chip; *IEE Proceedings*, 133, Part E(5), September 1986, pp. 255-270.

[8] Gordon, M.; Melham, T.: *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*; Cambridge, University Press, 1993.

[9] Gupta, A.: Formal Hardware Verification Methods: A Survey; Journal of *Formal Methods in System Design*, Vol. 1, No. 2/3, 1992, pp. 151-238.

[10] McMillan, M.: *Symbolic Model Checking*; Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[11] Leslie, I.; McAuley, D.: Fairisle: An ATM Network for Local Area; *ACM Communication Review*, Vol. 19, No. 4, September 1991, pp. 237-336.

[12] Tahar, S.; Zhou, A.; Song, X.; Cerny, E. and Langevin, M.: Formal Verification of an ATM Switch Fabric using Multiway Decision Graphs; *Proc. IEEE Sixth Great Lakes Symposium on VLSI (GLS-VLSI'96)*, Ames, Iowa, USA, March 1996, IEEE Computer Society Press.

[13] Zhou, Z.; Song, X.; Corella, F.; Cerny, E.; Langevin, M.: Description and Verification of RTL Designs using Multiway Decision Graphs; *Proc. Conference on Computer Hardware Description Languages and their applications* (CHDL'95), Chiba, Japan, August 1995.