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

Michel Langevin[§1], Sofiène Tahar[2], Zijian Zhou[1], Xiaoyu Song, Eduard Cerny
University of Montreal, IRO Dept., Montreal, Quebec, Canada
[§] GMD-SET, Schloss Birlinghoven, St. Augustin, Germany

Abstract

We investigate 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 unrestricted frame size, cell length and 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 the ASMs and of the set of reachable abstract states, allowing implicit abstract state enumeration. Since MDGs avoid model explosion induced by data values, this experiment demonstrates the effectiveness of MDG-based verification 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. Simulation is traditionally used for checking system correctness. However, it is practically impossible to achieve exhaustive simulation for such complex systems. Therefore, formal verification of the correctness of digital systems is gaining interest, as the correctness of a formally verified design implicitly involves all possible input values [7].

ATM (*Asynchronous Transfer Mode*) is considered as the network technology for addressing the variety of needs for new high-speed, high-bandwidth applications. Although ATM is being hailed as the most important communication mechanism in 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 component is the Fairisle 4 by 4 switch fabric which is part of the Fairisle ATM network [9]. It was fabricated with no consideration for formal verification. The main contributions of our work are the specification of the expected behavior of the switch fabric, and its equivalence checking against the hardware implementation. The behavioral specification and thus the verification have no restrictions on 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 confirms the effectiveness of the MDG-based verification methodology as an extension of ROBDD-based approaches [1], since model explosion induced by data values expanded to their binary representation is largely avoided.

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

2. Related Work

There exist so far 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 [6]. He hierarchically verified each of the modules used in the 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. Using SMV (Symbolic Model Verifier) [8], they identified the error by checking a number of prop-

[1] now by Nortel Technologies, Ottawa, Ontario, Canada
[2] now by Concordia University, Montreal, Quebec, Canada

1063-6404/96 $5.00 © 1996 IEEE

erties described in CTL (Computational Tree Logic) [8]. Given the Boolean representation in SMV, to avoid state space explosion, they abstracted the width of addresses from 8 bits to 1 bit, and the number of addresses in a FIFO from 168 to 5. However, in some cases a property could not be verified because of this reduction and a detailed gate-level model was needed for certain blocks to pinpoint the source of the error.

Curzon's work used HOL which is interactive and requires expertise to guide the verification process [7]. The work at Fujitsu Ltd. used a model checker which is automatic, but the adopted data abstraction for avoiding 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. 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 Reduced Ordered Binary Decision Diagrams (ROBDD) [1]. In a previous work [10], we produced the description of the Fairisle switch fabric 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 implementation against a behavioral specification with no restrictions on the frame size, cell length or word width.

3. The Fairisle ATM Switch

The Fairisle switch [9] consists of three types of components: *input port controllers*, *output port controllers* and a *switch fabric* (Figure 1).

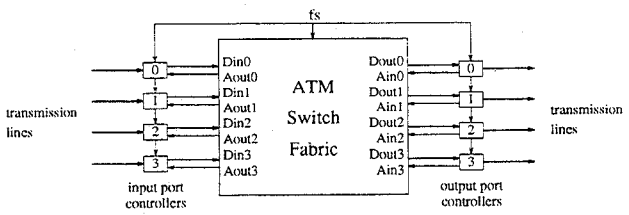


Figure 1. The Fairisle ATM switch

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 re-try later. The routing tag also includes priority information (*priority* bit) used for arbitration: high priority cells are given precedence, and the final choice is made on a *round-robin* 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 arrive at the same time. In this paper, we are concerned with the verification of the switch fabric only.

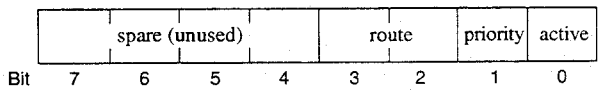


Figure 2. The routing tag of a Fairisle ATM Cell

3.1. Behavioral specification

Starting from a set of timing-diagrams describing the expected input-output behavior of the switch fabric, we derived a behavioral specification in the form of a finite state machine. The state machine 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*.

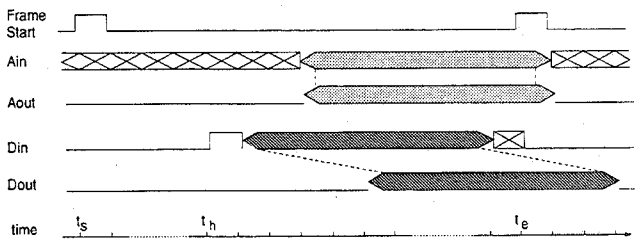


Figure 3. Expected behavior in an active frame

A timing-diagram of the expected input-output behavior during an active frame is in Figure 3. After the frame starts (at time t_s), the switch waits for the headers to appear on the 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 (bytes that follow the headers on *Din*) are transferred to the corresponding output ports (*Dout*) with a delay of 4 cycles, while acknowledgments traverse in the opposite 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 was inspired by this behavioral description as given in the design documentation (in the form of timing diagrams). The description is based on the following four assumptions about the environment of the fabric:

- At start up (t_0) the frame start (t_s) waits at least two cycles before going high, i.e., $t_s \geq t_0+2$
- Headers arrive (t_h) at least three cycles after frame start (t_s), i.e., $t_h > t_s+2$
- Next frame start arrives ($t_e=t'_s$) at least three cycles after frame start (t_s), i.e., $t_e > t_s+2$
- 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. To simplify the presentation, 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, "~" denotes negation, and the symbols *a*, *d* or *r* inside a conceptual state represent the computation of the acknowledgment output (*Aout*), the 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.

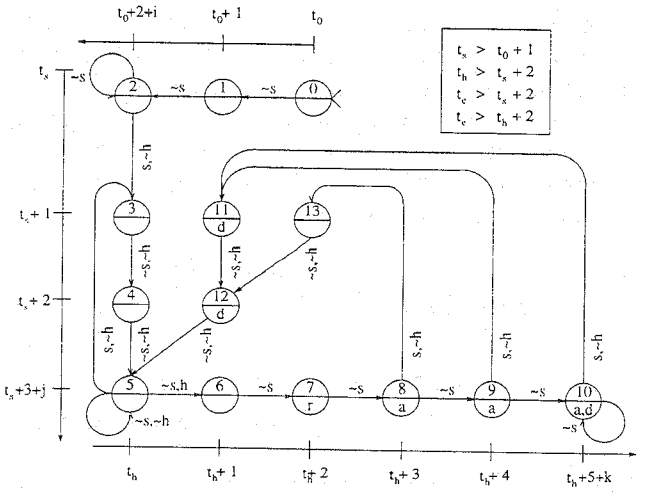


Figure 4. The expected switch fabric behavior

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

State 0 is the initial state from which there must be two transitions without the arrival of the frame start (states 1 and 2). This complies with the first constraint on the environment of the switch. The states 0, 1 and 2 are related to the time axis t_0. The waiting loop for the first frame start in state 2 is shown 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. These states 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 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 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. States 6 to 10 are related to the time axis t_h. After arbitration (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 state 10. The self-loop in state 10 represents the transmission of data and acknowledgments in the remaining cycles of the cell (indicated by a natural number *k*). The arrival of a frame start in states 8, 9 or 10 marks the beginning of another frame. Here, a new sequence of state transitions along the t_s axis progresses similarly as in states 3, 4 and 5 described above, but considering possibly different scenarios for completing the transmission of the preceding cells.

To compute the acknowledgments, the data outputs and the round-robin arbitration, we use the following state variables:

- 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 the states annotated by *a* (8, 9 and 10) the values of $Aout_i$, i={0,...,3} are computed as follows (**ef** stands for **else if**):

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)

In the states annotated by *d* (10, 11 and 12), the values of $Dout_i$, i={0,...,3} are 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 ($Dout_i$=$sr_{3,4}$)

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 can be easily described using **if-then-else** constructs, but it is too long to be shown here.

3.2. The switch fabric 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. The dataswitch performs the actual switching of data from input ports to output ports (a set of registers is used in the dataswitch; they are reset when their corresponding *outDis* signal is 1). The ack unit passes appropriate acknowledgments to the input ports. 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.

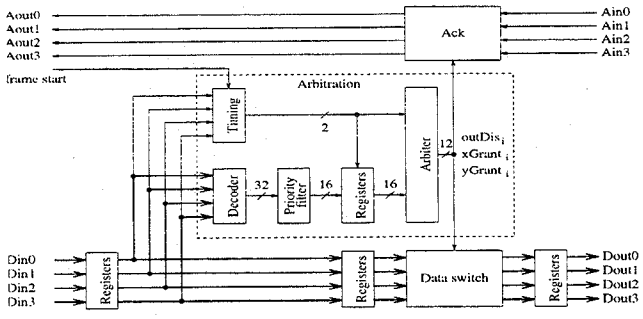


Figure 5. A Fairisle switch fabric implementation

Each of the units is repeatedly subdivided until eventually the logic gate level is reached. The design has a total of 441 basic components (a logic gate with two or more inputs, or a 1-bit Flip-Flop). The design was described in Qudos HDL [5] and the Qudos simulator was used to perform the original (non-formal) validation.

4. Representation using MDGs

To compare the two descriptions exposed in the preceding sections, we need to model their behavior. 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].

MDGs 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 [11].

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 data 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 then be encoded in MDG. MDG-oriented modeling using *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 which allows the description of hierarchical 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, it contains constructs such as ITE (**if-then-else**) and CASE formulas, or tabular representations. A translator from a subset of VHDL is under development.

4.1. ASM model of the specification

Definitions of 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*
- cross-operator *act* of type [*wordn* ⟶ *bool*] (representing the active field of header)
- cross-operator *pri* of type [*wordn* ⟶ *bool*] (representing the priority field of header)
- cross-operator *rou* of type [*wordn* ⟶ *port*] (representing the route field of header)

The ASM is composed of

1) Input variables *fs*, Ain_i (i = {0,...,3}) of sort *bool*, and Din_i (i = {0,...,3}) of sort *wordn*;
2) Output variables $Aout_i$ (i = {0,...,3}) of sort *bool* and $Dout_i$ (i = {0,...,3}) of sort *wordn*;
3) 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*.

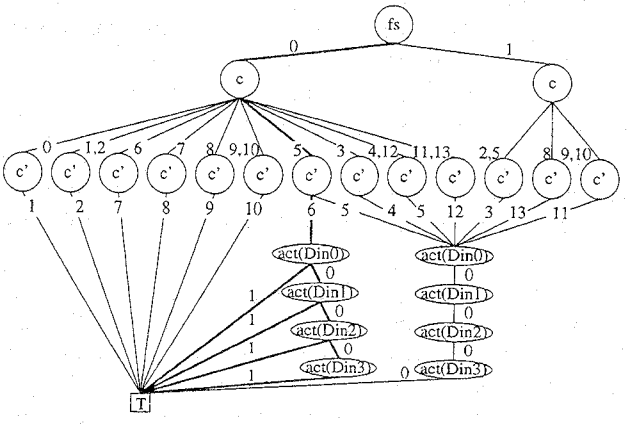


Figure 6. MDG associated with next-state *c'*

The description of the ASM is completed by giving its output and next-state relations. 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 of the next-state variable *c'* is shown 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). 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, except that first-order terms can appear along the paths. The terms $act(Din_i)$ are *cross-terms*; they encode data-dependent decisions. The MDG of the output $Dout_0$ 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, $Dout_0 = zero$.

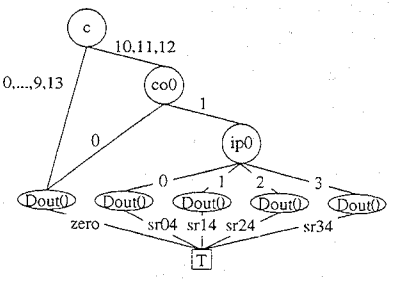
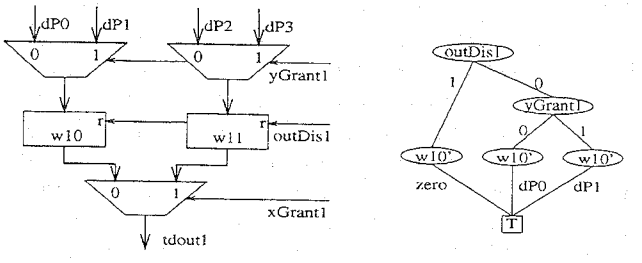


Figure 7. MDG associated with $Dout_0$

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

4.2. ASM model of the implementation

We first translated the Qudos HDL description into MDG-HDL. We also constructed an RTL description in which all byte signals are of abstract sort *wordn*, and the active, priority and route fields are accessed using the cross-operators *act*, *pri* and *rou*, respectively. The dataswitch is implemented using simple multiplexors instead of collections of logic gates. This leads to a simpler implementation of the dataswitch, reflecting the switching behavior in a more natural way and using a smaller network of components [10].



(a) RTL Implementation (b) MDG model for signal w_{10}

Figure 8. Implementation models of dataswitch

Each RTL component has an MDG model in our library. Given the RTL netlist, an ASM 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 [11], like in the ROBDD case. For instance, Figure 8 (a) shows the RTL netlist of a word-slice of the switch output port 1. The dP_i signals come from the registers at the input of the switch (Figure 5), thus delaying the data inputs by two cycles. The output of the dataswitch is fed into registers before reaching the output of the fabric. 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

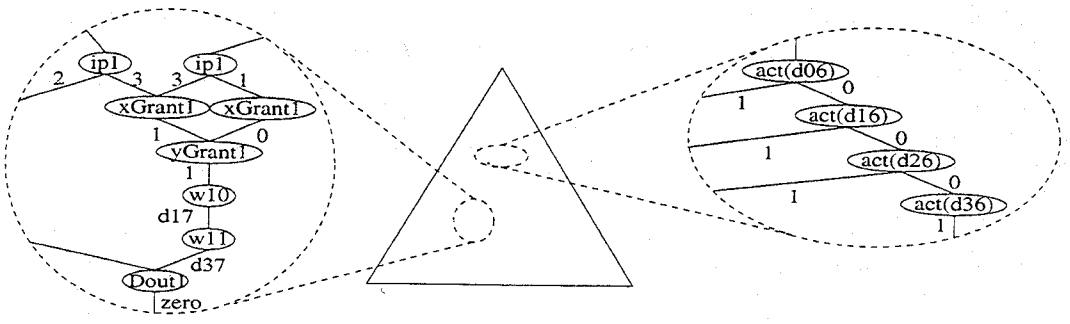


Figure 9. MDG encoding a set of reachable abstract states

input ports). The selection is then completed based on the value of $xGrant_i$ is known. If $outDis_i$ is 1, the intermediate registers are forced to *zero*. Given the MDG models of the *wordn* multiplexors and registers with synchronous reset, the MDG model for w_{10}' (next value of w_{10}) is shown in Figure 8 (b). The MDGs associated with the other registers and the output signals are obtained in a similar way.

5. Formal verification of the Fabric

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 in every state using reachability analysis of the product-machine. As with ROBDDs, we use MDGs to represent sets of states, except that they are abstract, and thus perform implicit abstract state enumeration automatically using graph-based operations on MDGs (conjunction, existential abstraction, renaming, and pruning-by-subsumption to approximate difference) [3]. MDG-based reachability analysis avoids state explosion induced by data, since the verification is independent of the datapath width.

For the product of the ASMs from Sections 4.1 and 4.2, an MDG representing a set of total-states encodes a relation between 39 concrete 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 ROBDDs, 8 Boolean variables would be needed for each abstract 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. In MDGs, the encoding is done using abstract data, yet isomorphic graph sharing is exploited as in ROBDDs (e.g., the left hand side of Figure 9, where the variables d_{ij} represent the values of Din_i at iteration j). Decisions on values of abstract data are represented by cross-terms which also contribute nodes in the MDGs (e.g., $act(d_{06})$ in 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.

Using abstract reachability analysis, verification 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 [10], we accomplished the verification of the abstract RTL description against the gate-level one. By combining these two results, we obtain complete verification of the design, from a behavioral specification down to the gate-level implementation.

6. Experimental Results and Discussion

We report the experimental results of comparing the ASM specification of the switch fabric against the ASM RTL model using our MDG techniques. A prototype of the MDG verification tools are implemented in Quintus Prolog. The experiments were done on a SPARC station 10 with 128 MB of memory. Table 1 shows the size and the CPU times for constructing some significant MDGs generated during the reachability analysis.

Table 1. Size and generation time of some MDGs

MDG	Number of Nodes	Number of 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 is the frontier set of states at the end of iteration j, while v_j is 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 is the frontier set of states after the first arbitration phase in the implementation (i.e., at $t_h + 2$), while q_8 encodes those after the arbitration is completed in both machines. q_9 and q_{10} encode states where the frame may be terminated. q_{11} is the final false MDG of the frontier set (representing the empty set) meaning that all reachable states have been visited. The output of the two machines were compared at each iteration, which took up to 100 sec. in some cases. No difference between the two ASMs were detected.

Table 2 summarizes the results. 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. Behavioral verification results

Equivalence Verification	CPU time (sec.)	Memory (MB)	Number of Nodes
Compiling	120	10	25417
Reachability Analysis	2800	140	295139
Total	2920	150	320556

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 during the reachability analysis, and counterexamples were generated to help with diagnosing the errors. Table 3 shows the results for these three experiments, including separate CPU time for performing the reachability analysis and for generating counterexamples.

Table 3. Verification of faulty implementations

Case	Reach. Anal. (sec)	Counter-example (sec)	Number of Nodes	Memory (MB)
Error 1	11	9	2462	1
Error 2	850	450	150904	120
Error 3	600	400	147339	105

7. Conclusions

We have shown the applicability of formal verification based on a new class of decision graphs, the Multiway Decision Graphs, to a real circuit. We have demonstrated that formal verification of a real piece of communication hardware can be conducted automatically using the MDG tools. We investigated equivalence checking of the RTL hardware implementation of the Fairisle ATM switch fabric against a 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 also verified several faulty implementations and the injected errors were successfully identified. The results of our current and previous work [10] illustrate the practicability of 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.

We are in the course of developing a model checking algorithm for a restricted first-order temporal logic. This feature will allow the verification of properties on Abstract State Machines.

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

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

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

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

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

[10] Tahar, S.; Zhou, A.; Song, X.; Cerny, E.; 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.

[11] 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.