Formal Verification of a SONET Telecom System Block

M. Hasan Zobair and Sofiène Tahar

Concordia University, Montreal, Quebec, H3G 1M8 Canada
Email:{mh_zobai,tahar}@ece.concordia.ca

Abstract. In this paper, we describe the formal verification of an industrial hardware design from PMC-Sierra, Inc. The design under investigation is a Telecom System Block which processes a portion of the SONET (Synchronous Optical Network) line overhead of a received data stream. We adopted a hierarchical modeling and verification approach which follows the natural design hierarchy. The formal specification and verification have been carried out based on MDGs (Multiway Decision Graphs), a new decision diagram subsuming the traditional binary decision diagrams and allowing abstract data and functions. The verification has been performed using both model checking and equivalence checking. To measure the performance of the MDG based model checker, we also conducted a comparative verification of the same design using Cadence FormalCheck tool.

1 Introduction

Simulation-based methods are currently used by the industrial community for system-level verification, since it can handle the entire design at a time. Simulation, however, cannot provide a high coverage ratio due to the exponential number of test cases to be developed and verified. This handicap is the reason, new methods are needed for the economical and reliable verification of digital systems. Formal verification [7] have recently paved a path, showing the utility of finding bugs early in the design cycle. Formal verification techniques are usually classified in two categories [7]: interactive theorem proving and automatic decision diagram based model checking and equivalence checking. In model checking, one checks if the design satisfies some properties (formal specification). With equivalence checking, we check if two designs exhibit the same behavior. The latter techniques have been successfully applied to real industrial design. However, since most tools are based on Binary Decision Diagrams (BDDs), they require the design to be described at the Boolean level. In practice, they often fail to verify a large-scale design because of the so-called state space explosion [7].

In this paper, we present a methodology for the formal verification of a real industrial design using the Multiway Decision Graphs (MDG) tools. MDGs subsume the traditional binary decision diagrams while extending them with abstract data sorts and uninterrupted function symbols [6]. The design we considered is a Telecom System Block (TSB) from PMC-Sierra, Inc., called RASE—

Receive, Automatic Protection Switch Control, Synchronization Status Extraction and Bit Error Rate Monitor [8]. The main aspect of this paper is to illustrate the ability to carry out the verification of an industrial size design using MDGs. Furthermore, we conducted a comparison between the experimental results obtained with the MDG model checker and the model checking of the same design using Cadence FormalCheck [5].

There exists a few related works on the application of MDG tools in hardware verification of real systems. For instance, Tahar *et al.* [9] verified the Fairisle ATM (Asynchronous Transfer Mode) Switch Fabric using MDGs which has about 4200 equivalent gates implemented in Xilinx FPGAs. In comparison to this design, our investigated design represents with its 11400 equivalent gates [8] a significant larger case study. Other major differences between our work and the one presented in [9] are as follows:

1) Unlike the design presented in [9] in which the authors used an academic ATM switch fabric designed at Cambridge University, our work presents a design which is a commercial product.

2) We applied both safety and liveness properties model checking on the investigated design, but for the work in [9], the authors applied only invariant (safety) property checking on the system. In difference to [9], we were hence able to make a direct comparison with model checking tools such as FormalCheck [5].

Another notable related work is the one done by Balakrishnan *et al.* [2]. The authors used the MDG tools to model and verify an Embedded System of a mouse control application based on the PIC 16C71 Microcontroller from Microchip Technology, Inc. Although this represents the sole commercial design verified by MDGs before our work, its application is concerned with software assembly aspects rather than hardware design and implementation.

The rest of this paper is organized as follows: Section 2 gives an overview of Multiway Decision Graphs (MDGs). Section 3 describes the functionality of the RASE TSB. Section 4 describes our hierarchical verification methodology. Section 5 presents a comparison of the verification process between MDG and Cadence FormalCheck. Section 6 finally concludes the paper.

2 Multiway Decision Graphs

Multiway Decision Graphs (MDGs) [6] have been proposed to solve the state space explosion problem of ROBDD (Reduced Ordered Binary Decision Diagram) [4] based verification tools. While accommodating a higher level of abstraction as with theorem proving, the MDG tools offer automation in the verification process like ROBDD based tools. An MDG is a finite, directed acyclic graph [6]. An internal node of an MDG can be a variable of concrete sort with its edge labels being the individual constants in the enumeration of the sort; or it can be a variable of abstract sort and its edges are labeled abstract terms of the same sort; or it can be a cross-term (whose function symbol is a cross-operator). Variables of concrete and abstract sort can be used to model control and data signals, respectively, while uninterpreted function symbols and cross-operators denote data operations and feedback from the datapath to the control circuitry, respectively. Hence, a data signal can be represented by a single variable of abstract sort, rather than a vector of Boolean variables, and data operations can be viewed as black boxes and represented by uninterpreted function symbols. An MDG may have only one leaf node denoted as **T**, which means all paths in an MDG are true formulae. Thus, MDGs essentially represent relations rather than functions in a canonical form. MDGs can also represent sets of states. Like ROBDDs, MDGs must be reduced and ordered. The underlying logic of the MDGs is a subset of many-sorted first-order logic [6].

In MDG-based verification, abstract descriptions of state machines, called Abstract State Machines (ASM) [6] are used to model the system. An ASM is obtained by letting some data inputs, states or output variables be of abstract sort, and the datapath operations be uninterpreted function symbols. They admit non-finite state machines as models in addition to their intended finite interpretations. This makes it possible to verify a circuit at the RTL functional model. The MDG tools [12] provide algorithms for equivalence checking, invariant checking and model checking, which are based on the reachability analysis of all states. The equivalence verification procedures are combinational and sequential verification. The model checking supports a first-order temporal logic, called $\mathcal{L}_{\mathcal{MDG}}$ ($next\_let\_formula$) [10], which is an extension of universally quantified branching time first-order temporal logic. The MDG tools run on a Prolog platform and accepts a Prolog-style HDL as its input language, called MDG-HDL [12], which allows the use of abstract variables for representing data signals. MDG-HDL supports structural descriptions, behavioral descriptions, or a mixture of structural and behavioral description. It also comes with a library of basic hardware components, such as logic gates, multiplexers, bus drivers, etc.

The MDG tools have some significant practical limitations: For instance, due to the non-interpretation of data operators, the reachability analysis of abstract states may not terminate. If this situation occurs, a dedicated heuristic has to be used from a set of algorithms developed in [1] and [11]. Another practical drawback of the MDG tools with respect to an industrial setting is that they do not accept VHDL or Verilog HDL as input language. However a project is currently underway to translate Verilog to MDG-HDL which will be available in the near future. This paper is to advocate MDG, we will hence use small examples from the RASE TSB model to illustrate the MDG modeling of behavior and structure.

3 The RASE Telecom System Block

The design under investigation is the RASE Telecom System Block (TSB). It processes a portion of the SONET (Synchronous Optical Network) [3] line overhead of a received SONET data stream. The RASE TSB consists of three types of components: Transport overhead extraction and manipulation, Bit Error Rate Monitoring (BERM) and Interrupt Server (see Figure 1) [8]. In addition to these blocks, it has an interface to a Common Bus Interface (CBI) block which is used mainly for the configuration and testing of the TSB interface and two inputs/outputs multiplexers. The transport overhead extraction and manipulation functions are implemented by three sub-modules (Transport Overhead bytes extractor, Automatic Protection Switch (APS) control and Synchronization Status filtering).

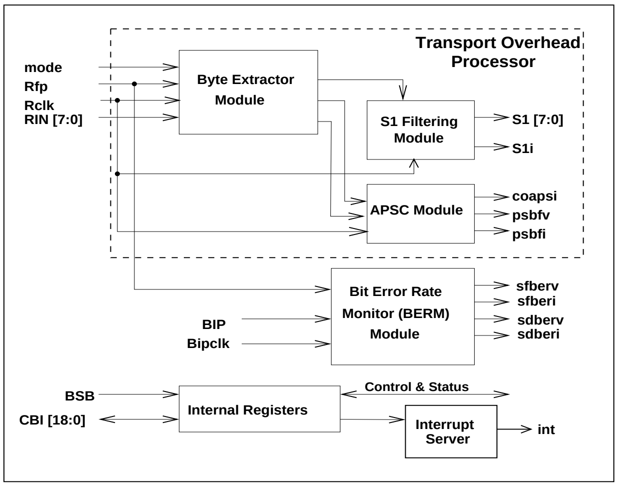


Fig. 1. The RASE Telecom System Block

The RASE TSB extracts the Automatic Protection Switch (APS) bytes, i.e., K1 and K2 bytes, and the Synchronization Status byte, i.e, S1 byte, from a SONET frame (see Figure 2). After extracting the above bytes, it processes them according to some requirements set by the SONET standard. The RASE TSB also performs Bit Error Rate Monitoring using the Bit Interleaved Parity (BIP)-24/8 line of a frame, i.e., B2 bytes (Figure 2). The received line BIP error detection code is based on the line overhead and synchronous payload envelope of the received data stream.

We propose a hierarchical approach to model the TSB behavior at different levels of the design hierarchy which in turn enables the verification process to be done at different levels. Inspired by [8], we derived a behavioral model of the RASE TSB which consists of five main functional blocks – Transport Overhead Extractor (TOH), Automatic Protection Switch (APS), Synchronization Status Filtering, Bit Error Rate Monitoring (BERM) and Interrupt Server.

3.1 MDG Modeling of the RASE Implementation

In this section, we describe the implementation of the TSB at the Register Transfer Level (RTL). For MDG-based verification we translated the original VHDL models into very similar models using MDG-HDL. One of the major

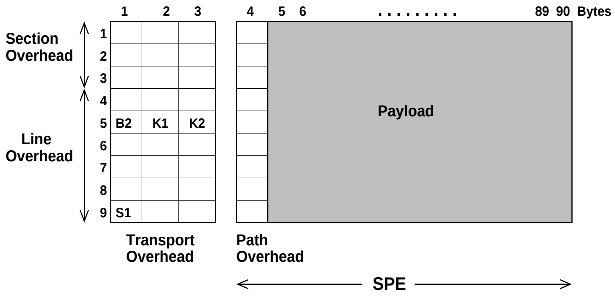


Fig. 2. The STS-1 SONET frame structure

advantages in using MDGs is the ability to handle abstract descriptions. This avoids all the ponderous procedure of defining each bit of a vector of Boolean variables. Rather, a vector of Boolean variable can be viewed as a single abstract variable. Another advantage in using MDGs is the ability to represent data operations by *uninterpreted* function symbols. This enables the arithmetic and logical blocks to be viewed as black boxes. To handle the complexity of the design, we hence adopted a module abstraction technique for the RTL model. We can illustrate the idea of module abstraction in MDGs using an example (see Figure 3) from our case study.

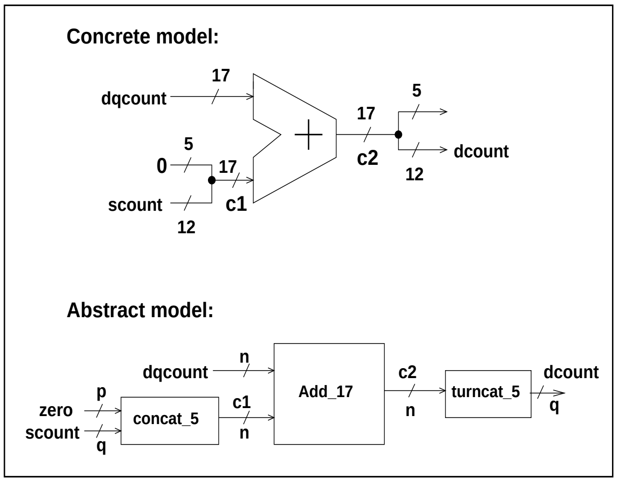


Fig. 3. An example of module abstraction

The circuit in the example is performing data operations over two operands of different size. It is concatenating 5-bits for matching the size of the operands to be used for addition and extracting twelve bits from the least significant bit positions of the output by truncating the upper bits. Using MDG-HDL, we can abstract the width of the datapath as well as the functionality of the original model. The data operations (e.g., addition, concatenation) can be modeled using *uninterpreted* function symbols applied to the operands (e.g. *Add_17*, *Concat_5*).

3.2 MDG Modeling of the RASE Behavior

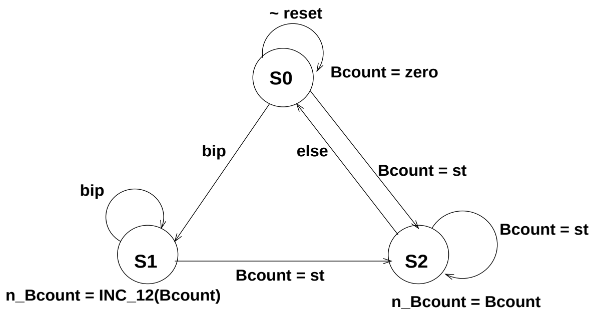
Based on the product documentation [8] provided to us, we derived a high-level MDG behavioral model of the RASE TSB using Abstract State Machines (ASMs). To illustrate the behavioral modeling approach adopted for the RASE TSB, we are presenting here only the BIP counting abstract state machine and its pseudo-MDG-HDL model (see Figure 4). The BIP line counter has three possible states—S0, S1 and S2. The state variable *Bcount*, of abstract sort *worda12*, stores the count value of the BIP line. The symbols *st* and *bip* are the inputs to the state machine. They represent the saturation threshold value of the counter and the received BIP line, respectively. In state S0, the counter has been initialized to *zero* which is a generic constant of abstract sort *worda12*. After initialization, if the input bip = “1” then the next state will be S1. The uninterpreted function *inc_12* of type[1] [*worda12* $\rightarrow$ *worda12*] denotes the increment-by-one operation of abstract words. When the count value is equal to the saturation threshold value *st*, there will be a transition to state S2. In state S2, the value of the counter will remain unchanged until *Bcount* is not equal to *st*.

An abstract state machine can have an infinite number of states due to the abstract nature of some variables and function symbols. The reachability analysis algorithm of the MDG tools is based on the abstract implicit state enumeration [6]. Due to the *non-termination* of abstract state enumeration all states may not be reached [1]. To illustrate this limitation of MDG-based verification, we can look at the example of Figure 4, where a generic constant *zero* of the abstract sort denotes the initial value of *Bcount*. The MDG representing the set of reachable states of the BIP counting ASM (see Figure 4) would contain states of the form (*Bcount*, *inc_2*(. . . *inc_2*(*zero*). . .)) for the number of infinite iterations. As a consequence, there is no finite MDG representation of the set of reachable states and the reachability algorithm will not terminate, since the structure of the MDG will become arbitrarily large. This typical form of non-termination can be avoided by using some heuristic techniques described, e.g., in [1]. One such methods is based on the *generalization* of initial state that causes divergence, like the variable *Bcount* in Figure 4. Rather than starting the reachability analysis with an abstract constant *zero* as the initial value of *Bcount*, a *fresh*[2] abstract variable (e.g., C) is assigned to *Bcount* at the beginning of the analysis.

[1] The notation f: $[\alpha \rightarrow \beta]$ implies that the function f has argument of sort α and range of sort β .

[2] A *fresh* variable is disjoint from all other variables.

ASM of the BIP line counter:



MDG-HDL model:

```
initialize Bcount = C
initialize state = S0
case state = S0: if bip = 1
                   then next(state) = S1
                        Bcount = inc_12(Bcount)
                   else if Bcount = st
                        then next(state) = S2
                             Bcount = Bcount
case state = S1: if bip = 1
                   then next(stste) = S1
                        Bcount = inc_12(Bcount)
                   else if Bcount = st
                        then next(state) = S2
                             Bcount = Bcount
case state = S2: if Bcount = st
                   then next(state) = S2
                        Bcount = Bcount
                   else next(state) = S0
```

Fig. 4. Example of an ASM with its MDG-HDL model

4 Verification of the RASE TSB

To carry out the verification of the system, we need to define a proper environment of the system. The functional simulation for the TSB was performed using a VHDL test bench. In the simulation, two different types of data structure had been used. These data structures are implemented for two different functions: the SONET frame data information and the BIP line data. For the formal verification, we are interested in five control blocks only to eliminate the possibility of state space explosion. During our modeling of the verification environment, we eliminated those blocks which have no effect on the functionality of the TSB. In contrast to simulation, formal verification has a built-in technique that allows a non-deterministic choice of values for the primary inputs. This non-deterministic choice of data structure will eliminate the use of deterministic data structure as used for simulation. After establishing a proper environment, we performed model checking and equivalence checking of the TSB. This will be described in the following subsections.

4.1 Model Checking of the RASE TSB

We first applied property checking to ascertain that both the specification and the implementation of the Telecom System Block satisfy some specific characteristics of the system. The verification of the properties has been carried out using the model checking facility of MDG tools [10].

The properties are described using a property specification language called $\mathcal{L}_{\mathcal{MDG}}$ [10], which is a universally quantified first-order temporal logic. Due to space limitations, in the following, we present one sample safety property (Property 7) of the RASE TSB. A complete description of a longer list of properties can be found in [13].

Property 7: When the calculated Bit Error Rate (BER) counters (*dcount* and *ccount*) exceed their programmable alarm declaration threshold value (*declare_th*) and alarm clearing threshold value (*clear_th*), respectively, an alarm (*berv*) will be triggered to indicate that the calculated BER on the SONET input data stream exceeds its programmable threshold value, i.e.,

$if\ ((Bipclk = 1)\ and\ (dcount > declare\_th)\ and\ (ccount > clear\_th))$
$then\ Always\ (berv = 1)$

where, *Bipclk* is the BIP line referenced clock and, *dcount* and *ccount* counters count up to 4k depending on the configuration of the threshold values.

We applied model checking on both the block and top levels of the TSB. As there are five blocks in our TSB design, we grouped the properties into five block level properties and finally merged them into a set of twelve properties featuring the top level verification of the TSB.

The experimental results from the verification of some specific properties for the implementation are given in Table 2, including the CPU time in seconds, memory usage in MB and the number of state variables. Experimental results of the full set of properties checked on both the behavioral specification and implementation models can be found in [13]. All experimental results have been carried out on a Sun Ultra Sparc 2 machine with 768 MB of memory.

4.2 Equivalence Checking of the RASE TSB

Once a certain confidence in the correctness of the models has been established, we proceed with the verification of the RASE TSB models using equivalence checking following a hierarchical approach. For instance, we verified that the RTL implementation of each module complied with the specification of its behavioral model. Thanks to the many abstractions we adopted to the overall RASE TSB, we also succeeded to verify the top level specification of the RASE TSB against its total implementation. Experimental results, including CPU time, memory usage and number of MDG nodes generated, for the equivalence checking between the behavioral models against their RTL implementations are given in Table 1, including the CPU time in seconds, memory usage in MB and the number of MDG nodes generated.

The verifications of the first four modules consumed less CPU time and memory, because they are less complex and have less abstract state variables and cross-operators than those of the last three modules (see Table 1). The BERM module consumed more CPU time and memory during the verification as it performs complex arithmetic operations on abstract data. On the other hand, the verification of the TOH Process module consumed less CPU time and memory, even though it needs more MDG components to model than the BERM module. This is because of the fact that the TOH Process module is a state machine based design and in contrast to BERM does not perform any complex data operation. To model the complex arithmetic operations of the BERM, we need more abstract state variables and uninterpreted functions, especially cross-operators, which have significant effects on the verification of this module. As the top level of the design comprises all the bottom level sub-modules, it obviously takes more CPU time and memory during the verification process than the other modules.

Table 1. Equivalence checking results

Property	CPU time (sec.)	Memory (MB)	MDG Nodes
TOH Extraction	3.88	2.33	2806
APSC module	17.37	7.91	9974
SSF module	22.22	6.81	14831
Interrupt server	0.48	0.09	180
BERM module	80.53	21.31	35799
TOH Pro. module	89.03	27.79	60068
RASE TSB	437.15	47.36	135658

5 Comparison with FormalCheck

One of the motivations of this work was to compare the model checking of the RASE TSB using the MDG model checker with existing commercial model checking tools, here Cadence FormalCheck [5]. The performance criteria of the comparison were CPU-time, memory usages and state variables. During our verification in FormalCheck, we setup the verification environment of the system in such a way that it reduces the state space and speeds up the verification process. To do so, we eliminated some blocks which do not affect the behavior of the system. Based on the design hierarchy, we organized the verification in FormalCheck such that model checking of the lower level modules will be done in first place. Finally, we integrated all sub-modules into a top level structural model of the RASE TSB and performed the model checking on it. In our performance comparison between FormalCheck and MDG model checking, we consider only the top level verification. The summary of the comparison between these two verification systems are given in Table 2, where '*' means that the verification did not terminate after a substantial run-time (several days). While performing the property checking on the top level model of the RASE TSB using FormalCheck, some of the properties verifications (Properties 5, 7, 8, 9, 10 and 12) did not terminate. These properties were taking too much CPU time and memory, even though we used different tool guided reduction and abstraction techniques in FormalCheck. The number of state variables used in FormalCheck are less than MDG, because FormalCheck has a built-in state reduction technique which was not available within the MDG system when we worked on this project.

Properties 7, 8 and 12 belong to the BERM module, which is the largest and most complex module of the RASE TSB. The BERM module has several control state variables which perform complex arithmetic operations between large sized data. The verification of Properties 7, 8 and 12 did not terminate within a substantial CPU time as these three properties are dealing with control signals having width of 24 bits to 12 bits. Moreover, some complex data operations between large sized state variables were involved. In general, if the control information needs n bits, then it is impossible to reduce the datapath width to less than n. Hence, in this case ROBDD based datapath reduction technique is no more feasible. On the other hand, using the MDG-based approach, we naturally allow the abstract representation of data while the control information is extracted from the datapath using cross-operators. ROBDD [4] have proved to be a powerful tool for automated hardware verification. However, they require a Boolean representation of the circuit and the size of an ROBDD grows exponentially with the width of the Boolean variables. Therefore, ROBDD-based verification cannot be directly applied to a mixed control-datapath design which has large data words. The verification of the BERM module falls into the above mentioned category which comprises lots of wide state variables, e.g., 12-bits, 24-bits. There are ways to deal with this problem: the first one is to reduce the complexity of the design by means of data abstraction and the second one is to represent hardware designs at different levels of abstraction, and verify the designs hierarchically. In contrast to MDG FormalCheck, which is an ROBDD based tool, does not support datapath abstraction. Because of this limitation in FormalCheck the verification of Properties 7, 8 and 12 encountered the state space explosion considering 12-bits and 24-bits state variables. In both cases we used a hierarchical verification approach depending on the design hierarchy.

Our experimental results show that FormalCheck whose underlying structure is automata oriented [7] is more efficient in verifying FSM-based modules, i.e., concrete variables, than the MDG tools. For instance, Table 2 shows that Properties 1, 2, 3, 4, 6 and 11 take less verification time in FormalCheck than in the MDG tools. These properties are related to the APSC, Synchronization status and TOH Extraction modules which are completely FSM-based designs. Properties 5, 9 and 10 did not terminate on the top level using FormalCheck, as these properties are verifying the integrated functionality of several modules.

Human effort to formal verification of any design is an important issue to the industrial community. In FormalCheck, we do not need manual intervention for variable ordering while for MDG tools, we used manual variable ordering since no heuristic ordering algorithm is available in the current version. The

Table 2. Model checking results using MDG and FormalCheck

Property	MDG			FormalCheck		
	Time (sec.)	Memory (MB)	State variables	Time (sec.)	Memory (MB)	State variables
Property 1	82.47	15.60	57	60	16.08	54
Property 2	82.62	14.98	57	32	12.81	71
Property 6	81.65	15.80	55	10	11.75	28
Property 7	82.54	15.86	57	*	*	*
Property 8	64.30	15.72	57	*	*	*
Property 9	78.06	16.65	55	*	*	*

translation of the original VHDL design description to MDG-HDL structural model took about one man-month. In contrast to this, no time was spent on the RTL modeling for FormalCheck which accepts the original VHDL structural model without any translation.

6 Conclusions

In this paper, we demonstrated that the MDG tools have the capability to verify a moderate size industrial Telecom hardware, the RASE TSB. Based on the product document provided by the PMC-Sierra Inc., we derived a behavioral model of the Telecom System Block. The specification was given as English text which was modeled in terms of Abstract State Machines using MDG-HDL. To handle the complexity of the RTL model, the module was abstracted by using MDG based abstract sorts and uninterpreted functions. We adopted a hierarchical verification approach to verify the whole TSB through MDG-based equivalence and model checking. Our verification did not find any errors in the existing design.

Although some of the ideas presented in this paper may be well known, this is the first time a prototype tool MDG has been applied to a commercial Telecom system. Unlike other conventional EDA tools, applying MDG to a design verification is not straight forward or well defined. To apply this tool, specially for a design like the RASE TSB, one needs to come up with a methodology "how to apply MDG" to this kind of large Telecom hardware. On the other hand, one of the motivations of this work was to compare the verification of the TSB using MDGs with the verification done by an existing commercialized formal verification tool, Cadence FormalCheck. The verification in FormalCheck has one major practical advantage over MDG, namely the VHDL/Verilog front-end which enables the integration with the design flow. The MDG-based approach can handle arbitrary data widths using abstract sort and uninterpreted functions which is a solution to the state space explosion problem. On the other hand, our experimental results show that in some cases FormalCheck fails short due to state space explosion for wider datapath.

7 Acknowledgments

We would like to thank PMC-Sierra, Inc. for providing us with the documentation and HDL models of the RASE TSB. We are in particular grateful to Jean Lamarche from PMC-Sierra, whose cooperation contributed to the maturity of this work.

References

1. O. Ait-Mohamed, X. Song, E. Cerny. On the nontermination of MDG-based abstract state enumeration. In Proc. IFIP Conference on Correct Hardware and Verification Methods, Montreal, Canada, October 1997, pp. 218-235.
2. S. Balakrishnan and S. Tahar. A Hierarchical Approach to the Formal Verification of Embedded Systems Using MDGs, In Proc. IEEE 9th Great Lakes Symposium on VLSI, Ann Arbor, Michigan, USA, March 1999, IEEE Computer Society Press, pp. 284-287.
3. Bell Communication Research (BellCORE). SONET Transport Systems: Common Generic Criteria. GR-253-CORE, issue 2, December 1995.
4. R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. In IEEE Transactions on Computers, Vol. C-35, No. 8, August 1986, pp. 677-691.
5. Cadence Design Systems, Inc. Formal Verification Using Affirma FormalCheck Manual. Version 2.3, August 1999.
6. F. Corella, Z. Zhou, X. Song, M. Langevin and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. Formal Methods in System Design, Vol. 10, February 1997, pp. 7-46.
7. C. Kern and M. Greenstreet. Formal Verification in Hardware Design: A Survey. ACM Transactions on Design Automation of Electronic Systems, Vol. 4, April 1999, pp. 123-193.
8. PMC-Sierra Inc. Receive, APS, Synchronization Status and BERM Telecom System Block. Engineering Document. Issue 4, January 29, 1998.
9. S. Tahar, X. Song, E. Cerny, Z. Zhou, M. Langevin and O. Ait- Mohamed. Modeling and Verification of the Fairisle ATM Switch Fabric using MDGs. IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 18, No. 7, July 1999, pp. 956-972.
10. Y. Xu, E. Cerny, X. Song, F. Corella, O. Mohamed. Model Checking for First-Order Temporal Logic using Multiway Decision Graphs. In Computer Aided Verification, LNCS 1427, Springer Verlag, 1998, pp. 219-231.
11. Z. Zhou, X. Song, S. Tahar, E. Cerny, F. Corella, M. Langevin. Formal Verification of the Island Tunnel Controller using Multiway Decision Graphs. In Formal Methods in Computer-Aided Design, LNCS 1166, Springer Verlag, 1996, pp. 233-246.
12. Z. Zhou and N. Boulerice. MDG Tools (v1.0) User's Manual. Dept. of Information and Operation Research, University of Montreal, Canada, 1996.
13. M.H. Zobair. Modeling and Formal Verification of a Telecom System Block using MDGs. M.A.Sc. Thesis, Concordia University, Department of Electrical and Computer Engineering, Montreal, Canada, April 2001.