High-Level Design and Validation of ATM Switch

(appears in proceedings of IEEE International High Level Design Validation and Test Workshop)

S.P. Rajan M. Fujita K. Yuan M. T-C. Lee

Fujitsu Laboratories of America, 3350 Scott Blvd. #34, Santa Clara, CA 95054

ECE Dept., University of Arizona, Tucson, AZ 85721

Avant! Corporation, 1208 E. Arques Ave., Sunnyvale, CA 94086

Abstract

Asynchronous Transfer Mode (ATM) has emerged as a backbone for high-speed broadband communications networks. In this paper we present ATM switch design starting from a parametric high-level model, validating the model using a combination of formal verification and simulation. We used a combination of theorem proving and model checking to discover bugs in the high-level model that was presumed correct using simulation. Parametric design validation obviates the need to validate specific ATM switch designs derived from the parametric model.

1 Introduction

Asynchronous Transfer Mode (ATM) technology has emerged as a backbone for high-speed broadband communications networks [CFFT96]. An ATM network backbone typically consists of a number of small ATM switches interconnected in a matrix topology. An ATM switch takes data from input ports and forwards the input data to the proper output ports in the same order as the input data. An ATM switch is typically designed as a RAM-embedded Application Specific Integrated Circuit (ASIC). High-level modeling and hardware synthesis has been shown to deliver good results in ATM switch design [LHCF96]. However, the high-level model presented earlier [LHCF96] works only for specific design requirements such as a fixed clock ratio and a fixed number of input/output ports. Furthermore, the model was presumed correct using simulation. However, by formal verification we discovered several bugs in the control part of the model. The difficulty in validation comes about because of complex control module design for a parametric number of concurrent processes, corresponding to an arbitrary number of input/output ports in the high-level model, communicating through shared signals. While simulation is efficient for validating portions of design that do not have many interacting concurrent processes, it is only by general purpose theorem proving that we can prove properties of designs with generic parameters, and model checking can efficiently handle the control part of the design with a small state space. Parametric ATM switch validation by exclusively using either simulation or one of the formal verification methods such as theorem proving and model checking would be tedious and inefficient. We employ a pragmatic combination of theorem proving and model checking [RSS95] in conjunction with small-scale conventional simulation.

There have been other efforts in ATM switch verification earlier [TZS+96, Cur94]. However, a major drawback with earlier efforts is that with any change of switch design requirements such as the clock ratio and the number of input/output ports, one needs to repeatedly iterate through high-level modeling, validation, and synthesis cycle until the design satisfies the design objectives. Furthermore, earlier efforts [CYF94, TZS+96, Cur94] do not introduce formal validation early in the design cycle, rather describe application of formal verification on a completed design *post-facto*. In order to cut down the modeling-validation-synthesis iteration cycle, the solution we propose in this paper is the development of a validated parametric high-level model that can be synthesized with concrete values for the generic parameters. The validation of the parametric model is done by a combination of simulation and formal verification early in the design cycle, before attempting high-level synthesis. We use a combination of theorem proving and model checking within the framework of Prototype Verification System (PVS) [ORR+96].

The rest of the paper is organized as follows: in Section 2 we discuss parametric model of the ATM switch. The validation process using a combination of theorem proving and model checking with an outline of simulation is discussed in Section 3. Finally, Section 4 presents the conclusions.

2 ATM Switch: Parametrized High-Level Model

An ATM switch forms the basic component of an ATM network, in which several switches could be connected in the form of a matrix for larger bandwidth. The modeling and functionality can be illustrated by a *small capacity ATM switch*, whose architecture is shown in Figure 1. The ATM switch has 2 input ports, 2 expanded ports, and 2 output ports. The incoming data are packed into 27 16-bit words called *cells*. The first 3 words of a cell form a `TAG`, which contains the switching information. The `TAG`

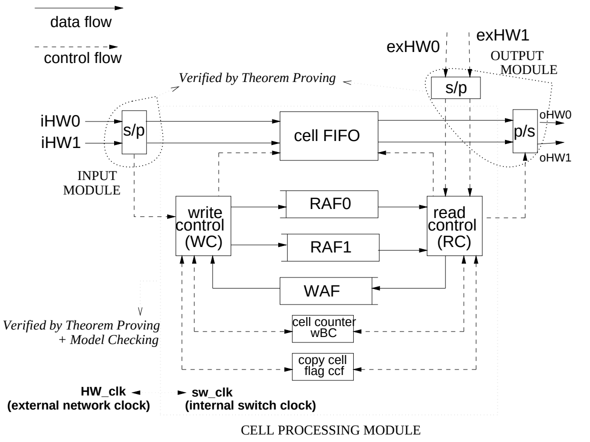

Figure 1: Architecture of ATM switch with 2 input and 2 output ports

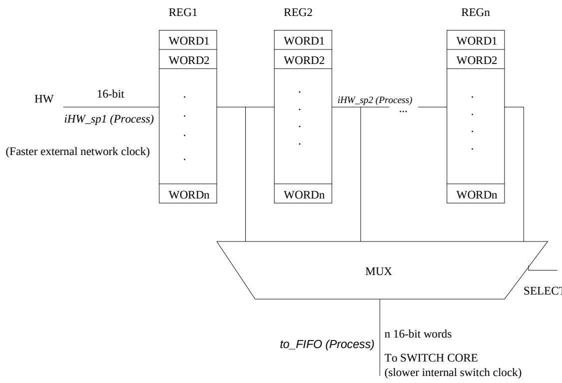

Figure 2: Buffering for Serial-to-Parallel conversion with clock ratio n

includes the type of cell, the cell ID, and output routing information. ATM cells from the network arriving at the input ports iHW0 and iHW1 are collected by the INPUT MODULE. The cells are forwarded to the CELL PROCESSING MODULE, which stores the cells in the proper addresses and routes them to the corresponding output ports through the OUTPUT MODULE. The different formal verification methods applied to various modules are also shown in Figure 1. The ratio of the external network clock to the internal switch clock determines the size of the buffer for Serial-to-Parallel conversion at input.

In the parametric model of the ATM switch, the clock ratio, the number of input/output ports, and the buffer size, are declared as generic parameters. The model is then ready be used for hardware synthesis and simulation with concrete instantiations of the generic parameters.

2.1 Parametric Clock Ratio

A parametric ratio entails a matching parametric size of the buffer for S/P conversion (see Section 2). Thus if we choose a ratio n, we need an n x n shift-register, i.e. n registers, each of which can store n words. Every external network clock cycle brings in a new incoming word to be stored in the first register. As shown in Figure 2, after every n external network clock cycles, i.e. at every internal switch clock cycle, all the contents of register i are shifted to register i+1, where i ranges from 1 to n-1. A generic n-bit counter is used to keep track of the next set of n words that need to be processed by the Write-Control (WC) for writing into the cell FIFO.

The S/P conversion module for a generic clock ratio n is modeled as a set of two concurrent processes. The processes iHW_sp1 and iHW_sp2 in Figure 2, both activated by the external network clock, execute as follows:

- Process iHW_sp1 stores incoming cell words into the first register iHW_bword(1). Thus the ith cell word is stored to the ith slot of register 1, with i ranging from 1 to n. Every register is associated with a bit-vector for checking parity of the incoming cell words. Immediately after the incoming nth cell word has been stored in the first register, the first bit of the n-bit counter iHW_cycle_counter, which keeps track of the register that contains set of words ready to be fetched by the next module, is set to bit value 1. Every other bit has the value 0. It should be noted that such a cycle counter is needed as a sentinel [LHCF96] because the internal switch clock (sw_clk) and the external network clock (HW_clk) may not be synchronized with respect to the rising/falling clock edges.

- Process iHW_sp2 shifts contents of register i to register i+1 at every network clock cycle. The bit value 1 in iHW_cycle_counter is shifted to the left as the clock advances.

- Process to_FIFO is activated by the slower internal switch clock. The ith bit in iHW_cycle_counter is set to value 1, which indicates that the ith register contains the set of cell words ready to be fetched for further processing. Exactly one set of words is generated for every n external network clock cycles, or one internal switch clock cycle.

The parallel-to-serial (P/S) conversion module is modeled as a set of communicating processes similar to the S/P module, and is not discussed in this paper.

2.2 Parametrizing the number of ports

In the model of the ATM switch presented earlier [LHCF96], the two ports are each associated with a process, and each process communicates with other processes, such as the WC, using synchronization signals. In the parametric model, we must construct an arbitrary number of processes and corresponding synchronization signals corresponding to a generic number m of ports.

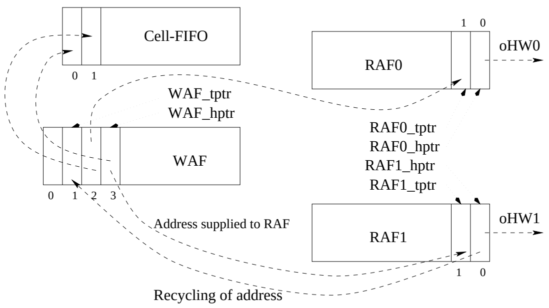

Figure 3: Address Recycling in a 2-port ATM switch with generic buffer size for WAF, cell-FIFO, RAF0/1

2.3 Address Recycling Control Module

At power-on or reset, the Write-Address FIFO (WAF) unit is initialized to consist of new addresses in which the incoming cells are to be stored. As a cell arrives from the input ports, the WAF unit supplies a new address in which the cell word must be stored, and supplies that same address to the RAF units. The new address in WAF is obtained from the pointer `WAF_hptr`, and the address is stored in RAFs in the location pointed to by `RAF_tptr`s. When a cell word is to be read out for output, the RAF units supply the address from which the cell word must be read. The pointer `RAF_hptr`s points to the address in RAFs from which the next cell word is to be read out. When the cell word is read out, the RAF units recycle the address back to the WAF unit. In the case of a copy cell, the cell must be read out of each RAF unit corresponding to an output port before the cell address is recycled back to the WAF. The recycling process in an ATM switch with 2 ports is shown in Figure 3. In summary,

- `WAF_hptr` points to the address in the WAF to be supplied to the current incoming cell word.
- `WAF_tptr` points to the WAF location in which the next recycled address must be stored.
- `RAF_hptr` points to the RAF location in which the cell word must be output next, and
- `RAF_tptr` points to the RAF location in which the cell word currently supplied by the WAF must be stored.

The pointers are incremented by 1 (modulus the buffer size) for accessing the next location in the FIFOs.

3 Formal Verification

The goal of formal verification of the ATM switch model is to show the *overall functional correctness property* that the *input cells are switched to the proper output ports, and that the order of input cells is maintained at the output.* The overall correctness property subsumes the properties that input cells are not lost, and that the cells without a copy flag are not duplicated at the output. Furthermore, it should be noted that the overall correctness property for the parametric ATM switch model holds for any arbitrary clock ratio, an arbitrary number of ports, and an arbitrary buffer size of FIFOs.

The parametric ATM switch model described in Section 2 is verified by first verifying the properties of its components. The INPUT MODULE is verified by theorem proving, the CELL PROCESSING MODULE is verified by a combination of model checking and theorem proving, and the OUTPUT MODULE is verified by theorem proving. The verified component properties are then composed to show the overall correctness property. The components correspond to the processes of the high-level model. The model is first translated into PVS specification language. The verification of the properties are performed by invoking appropriate built-in proof strategies. The proof strategies consisting of a combination of induction, rewriting, and special purpose decision procedures for linear arithmetic and model checking using BDDs provide a semi-automatic verification of the parametric ATM switch model.

There are essentially 3 modules in the ATM switch: input serial-to-parallel conversion (S/P), address recycling (AR), and output parallel-to-serial conversion (P/S). The overall correctness property can be shown by composing the correctness properties of each of the 3 components. First, Section 3.1 gives an overview of specification and verification using PVS. The verification of the correctness properties of the serial-to-parallel conversion module is discussed in Section 3.2, and the address recycling module in Section 3.3. The verification of the correctness of the output parallel-to-serial conversion is similar to the input module.

3.1 Overview of PVS

The Prototype Verification System (PVS) [ORR+96] is an environment for specifying entities such as hardware/software models and algorithms, and verifying properties associated with the entities. The specification language features common programming language constructs such as arrays, functions, and records. It has built-in types for reals, integers, naturals, and lists. The language also allows hierarchical structuring of specifications. Besides features such as subtyping and dependent typing, it permits overloading of operators, as in some programming languages and hardware description languages.

The PVS verifier is used to determine if the desired properties hold in the specification of the model. The user interacts with the verifier by a small set of commands. The verifier contains procedures for Boolean reasoning, arithmetic and (conditional) rewriting. In particular, Binary Decision Diagram (BDD) [Bry92] based simplification may be invoked for Boolean reasoning. Model checking, in which properties are stated in fair-CTL and models are written as state transition relations, can be invoked within PVS as another proof rule. The PVS verifier also features a variety of general induction schemes to tackle large-scale verification. Moreover, different verification schemes can be combined into general-purpose strategies for similar classes of problems, such as microprocessor verification [Sha96].

3.2 Formal Verification of Serial to Parallel conversion Module

The correctness property to be verified is that an incoming cell word from the network to an ATM input port appears at the output of the S/P buffer (i.e. the input to cell FIFO) at least after `n` (`n` is the clock ratio) external network clock cycles, i.e. 1 internal switch clock cycle and at most before `2n` external network clock cycles:

Theorem 3.1 (S/P Correctness Property)

```
∀ (t:time),(n > 0):
∃ m:  {m | n ≤ m ≤ 2n}:
sp1_sp2(n,t) ∧ to_fifo(n,t)
⟹
iHW0_cell_word(t) = iHW0_word(t+m)

%% Note:  Specification of the S/P modules:
   iHW_sp1/2 and to_fifo.
%% iHW0_cell_word is the cell word at input
    port iHW0 and iHW0_word is input to cell
    FIFO.
```

The proof in PVS follows by induction on clock ratio parameter "n", rewriting, arithmetic simplification, and propositional simplification using BDDs. The verification is semi-automatic with 5 interactive steps. The *run-time* of the proof on a Sparc-20 with 64 MB is **90 seconds**. The property could not have been checked using either simulation or finite-state model checking, because the clock ratio is a generic value.

3.3 Formal Verification of the Address Recycling (AR) Module

The address recycling module described in Section 2.3 consists of the WC, the WAF, the RC, and one RAF for each output port. In the parametric model, the sizes of the WAF and RAFs are arbitrary. The corresponding PVS model of the recycling module is a state machine parametrized with respect to the buffer-size of WAF/RAFs/cell-FIFO and number of input/output ports. The property that we would like to verify is that the cell contents of the address recycled from RAFs to WAF should be processed by the output module prior to recycling the address. The correctness of recycling depends on whether the concurrent processes WAF, RAF, WC, and RC, correctly update the shared signals corresponding to `RAF_hptrs`, `RAF_tptrs` and `WAF_hptr`, `WAF_tptr`.

The recycling correctness property can be recast as a property that the address supplied by the WAF unit to store the next incoming cell is not contained in any of the RAFs corresponding to the output ports. In the PVS state machine model, WAF and RAF buffer sizes are arbitrary (uninterpreted) and the number of ports is also arbitrary. Thus, we first instantiate the model with

- buffer size of 2 and
- number of input/output ports to 2

and verify the property that the address supplied from the WAF for the next incoming cell to be stored i.e. `iHW0_write_addr` or `iHW1_write_addr` corresponding to the 2 ports iHW0 and iHW1 is not contained in either of the 2 locations in either of the RAF buffers corresponding to the 2 output ports: oHW0 and oHW1 (see Figure 3). However, the verification did *not succeed* for our original model, and produced a counterexample in the form of a series of unprovable sub-properties. A closer examination of the sub-properties **revealed bugs** that the pointers to the WAF locations (`iHW0_WAF_hptr`, `iHW0_WAF_tptr`, `iHW1_WAF_hptr`, and `iHW1_WAF_tptr`) were not initialized to 0, and were not incremented at every clock step. The model was updated and the property was checked to hold in the model. Using CTL model checking in PVS, we specify the property using the `AG` modality to cover all the states and all the paths starting from the initial state as shown in Theorem 3.2:

Theorem 3.2 (Address Recycling (AR))

```
AG(¬ (iHW0_write_addr = RAF0(1))
     ∧ (iHW0_write_addr = RAF0(0)))
   ∨ ((iHW1_write_addr = RAF1(1))
     ∧ (iHW1_write_addr = RAF1(0))))
```

The state-space is small for a buffer size of 2, and the run-time for the verification is under **20 seconds** on Sparc-20 with 64 MB. We can characterize property 3.2 as a *safety* property.

Following earlier work on verification of Peterson's mutual exclusion algorithm for an arbitrary number of processes [Sha96], we can show using simple induction, rewriting, and arithmetic proof rules of general purpose theorem proving in PVS that property 3.2 holds for

- an arbitrary buffer size, and
- an arbitrary number of ports.

The verification is semi-automatic with a small number of interactive steps.

3.4 Verification of Overall ATM Correctness Property

In order to verify the overall correctness property that the cells are output in the same order as the input, we must verify 3 more properties corresponding to the RC and RAF units: "RC Liveness", which states that every RAF unit is read out eventually (i.e. no RAF unit is starved); "RAF Behavior", which states that every RAF unit behaves like a FIFO queue, and **Cell Tag Match** which asserts that the port tag bit of a cell word matches with the port number corresponding to the RAF unit in which the cell is stored. However, we employ a combination of theorem proving and model checking only for the first 2 properties, and check for **Cell Tag Match** by *simulation*. It is laborious to use formal verification for checking **Cell Tag Match** because of the data processing involved in extracting the proper TAG bits from the cell word. While simulation is efficient we can assert that the port tag bit of a cell word matches with the port number corresponding to the RAF unit in which the cell is stored.

Theorem 3.3 (RC Liveness)
`AF(output_port_read_address = RAF(RAF_hptr))`

Theorem 3.4 (RAF Behavior)
`AG(increment(RAF_hptr)`
$\Rightarrow$ `AF(increment(RAF_tptr)))`

The verification of properties above are similar to that of property 3.2 and use a combination of theorem proving and model checking. By composing the individual correctness theorems 3.1, 3.2, 3.3, 3.4, and **Cell Tag Match** property checked by simulation, we obtain the following overall correctness property:

Theorem 3.5 (ATM Switch Correctness)
Every input cell is switched to the proper output port, and the order among the cells at an output port is identical to that at the corresponding input port.

4 Summary

In earlier work [CYF94], a number of abstractions had to be manually applied without any computer-aided tool applied to formally verify an industrial ATM switch design using SMV-based model checking [McM93]. The verification successfully revealed the bugs and the design was corrected. But, such a post-facto verification would not be economically feasible in an industrial setting.

In this paper we have presented ATM switch design beginning with a parametric high-level model, using a combination of formal verification and simulation early in the design cycle to remove as many bugs as possible in the high-level model. The model has been used to obtain hardware implementations of customers' choice by hardware synthesis. Our work uses a pragmatic combination of theorem proving and model checking within a single framework, in conjunction with simulation for efficient verification of large industrial software/hardware designs.

References

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[CFFT96] Tom Chaney, J. Andrew Fingerhut, Margaret Flucke, and Jonathan Turner. Design of a gigabit ATM switching system. Technical Report WUCS-96-07, Computer Science Department, Washington University, St. Louis, Missouri, February 1996.

[Cur94] Paul Curzon. The formal verification of the fairisle ATM switching element. Technical Report 328 and 329, Computer Laboratory, University of Cambridge, Cambridge, UK, March 1994.

[CYF94] B. Chen, M. Yamazaki, and M. Fujita. Bug identification of a real chip design by symbolic model checking. In *Proceedings of the European Conference on Design Automation, the European Test Conference*, pages 132–136, Paris, France, February 1994. IEEE Computer Society.

[LHCF96] Mike T-C. Lee, Yu-Chin Hsu, Ben Chen, and Masahiro Fujita. Domain-specific high-level modeling and synthesis for ATM switch design using VHDL. In *Proceedings of the 33th Design Automation Conference*. Association for Computing Machinery, 1996.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Pub., Boston, MA, 1993.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[RSS95] Sreeranga P. Rajan, N. Shankar, and M. Srivas. An integration of model-checking with automated proof checking. In *7th Conference on Computer-Aided Verification*, July 1995.

[Sha96] N. Shankar. PVS: Combining specification, proof checking, and model checking. In M. Srivas and A. Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 257–264, Palo Alto, CA, November 1996. Springer-Verlag.

[TZS+96] S. Tahar, A. Zhou, X. Song, E. Cerny, and M. Kangevin. Formal verification of an ATM switch fabric using mutiway decision graphs. In *Proceedings of IEEE Sixth Great Lakes Symposium on VLSI (GLS-VLSI'96)*, Ames, Iowa, March 1996. IEEE Computer Society.