Invited Talk

H/W-S/W Co-Verification in ATM

Giovanni Mancini

Bell-Northern Research
Ottawa, Ontario, Canada, K1Y-4H7

Abstract

This paper presents a verification strategy for the hardware-software interface of a large ATM switch. The complexity of such a system and the requirements are discussed. A modeling strategy based on software emulation is presented which will allow for the functional verification of the interface prior to lab samples being available, resulting in reduced system integration times.

1. Introduction

Stored program or software controlled switching systems are one of the more complex electronic systems being designed today. Further adding to the complexity is the market pressures which virtually dictate concurrent engineering of hardware and software elements within the system.

The above is certainly true for ATM (Asynchronous Transfer Mode) telecommunications. ATM is presently viewed as the next big growth market in telecommunications with 100% compounded growth over the next 5 years (source: the Yankee Group, 1993). Therefore it is a very competitive market with suppliers designing their systems while standards are still evolving.

What poses a particular challenge in the development of ATM switches is that the nature of the switching function and customer requirements will require systems with greater levels of hardware-software interaction.

2. The System

A large ATM backbone switch is generally intended for the public networks market (e.g., telephone companies). This implies a class of functional and reliability requirements not normally present in standard electronic systems. The system itself consists of three major components (Figure 1), the hardware, the embedded control software (ECS) and the system control software (SCS).
The SCS software provides control functions for one or more switches and the required network management functions. The software may run on standard computer platforms such as UNIX based workstations. These processors are located in the network control centers of the operating companies or may be rack mounted in a central office facility, in close proximity to the actual hardware. The SCS can be of the order of hundred of thousands to millions of lines of code.

The hardware which provides the low level, real time switched datapath and the ECS are referred to as the Switching Element (SE). The SE consists of proprietary hardware, commercial components and industry standard embedded microprocessors. The ECS is of the order of 20K-40K lines of C/C++ code running under a commercially available real time operating system (RTOS). Numerous SEs can be grouped into clusters to increase the number of ATM cell streams switched and hence system throughput. SEs are usually configured with full redundancy to meet stringent reliability requirements.

Figure 1. A functional representation of an ATM switch architecture.

The SE hardware is considerable in size. One instance of an SE, including the redundant plane, may span up to three shelves of a rack. Each shelf may contain on average 12 full boards. Each board may contain up to 20 ASIC instances plus commercial components and possibly a microprocessor. The exact component count depends on the specifics of the transport layers terminated. One SE can have well over 2 million design gates instantiated within its custom ASICs, in addition to commercial components.

The SE architecture has the entire switching function (i.e. datapath) realized in H/W with virtually all higher level control functions implemented in software.

The components of the SE consist of a front end processing function which terminates the ATM cell streams from the network and carries out all ATM processing on the ingress and egress of cells. The cells are then relayed through a time switch function which switches the cells from the input port they arrived to the output port from which they are to leave the switch. Both functions are connected to embedded microprocessors which not only control the hardware but also carry out a number of operations for the SCS.

In ATM, as with any large switching system, there is a considerable amount of control information which must be transmitted to the SEs in the network. Part of the SE design is the mechanism by which it will receive control information and route to the appropriate microprocessor in an SE for action. The SCS may have to download data required to properly configure the switch for full operation. Control requests (e.g. a request to set up a new virtual channel) are interpreted by the ECS which will then execute the required bus transactions to configure the hardware registers and memories within the ASICs. Control data may also flow from the SE to the SCS. For purposes of this paper, we limit ourselves mainly to the verification of the hardware-software system within a SE.
3. Design Process

A typical system design process is represented by the diagram in Figure 2. The process starts with the initial architecture team developing a system architecture and specification. Using the system design specification, the design is partitioned into hardware and software design flows. The hardware flow is further divided into an ASIC design flow and a PCB design flow. Each of these three flows are tailored to the specific requirements of software design, ASIC design and PCB design. Once the hardware components are manufactured, the ASICs are incorporated onto the PCB and verified during hardware integration. Once base hardware functionality is verified, the software is added to the system during system integration.

Typically the EDA industry has focused on tool sets to get through the flows faster. Although there is still considerable progress that can be made in these areas, one of the biggest delays that can be encountered during product design is that which results from a design iteration. Major design iterations result from problems detected as late as hardware or system integration time.

With today's design processes, it is not unrealistic to have fully functioning hardware components at the end of the first design cycle. Iterations caused by problems detected at integration time are usually due to design errors relating to the functionality of the system. The system works but does not implement the correct function.

It is significant to note that in the development of such a new system as an ATM switch, the design and development of software and hardware proceed in parallel. In addition, the inter-working of the embedded software and hardware is highly interdependent and impossible to verify separately.

4. Goals

Briefly stated, we have three main goals. The elimination of design iterations due to ambiguous specifications, the reduction of lab integration time and the elimination of major problems discovered during lab integration. For this discussion, we will limit ourselves to the verification of the SE hardware-software interface.

In the design of a large switching system, the verification process is quite extensive. Brute force techniques will result in a syndrome we refer to as death by simulation. The design team runs larger and larger simulations in order to verify the system. If they are uncertain as to the capability of the verification system it is extremely difficult for them to know when they have completed the task.

It is crucial to note that the only aspect of the hardware which is relevant is the limited functionality seen by the software. This eliminates virtually all the details of the hardware datapath.

We must also clearly state what we do not need to verify. The embedded microprocessors and other standard commercial components can be assumed to be correct. The detailed hardware functionality of the custom ASICs will be verified by the hardware design process. The high level software functions can be verified independent of the hardware because of a layered software architecture.
5. Strategy

It is clear from the brief description above that the system is clearly orders of magnitude too complex to use brute force techniques. The strategy must take advantage of the system architecture and use techniques that are well tuned to the immediate requirements. To achieve this, a number of options are available.

5.1. Options

Six techniques commonly used are mentioned below.

Detailed specification: This approach relies on the specifications being complete and sufficiently clear that the three design flows can proceed independently and have all issues resolved at system integration. It generally results in long system integration times since we must have fully functional hardware before any major software verification can begin. It also assumes that most problems can be resolved through software modification.

Full simulations: Since the hardware function can be represented using HDL models, the software can be run on these models using a standard event driven logic simulator or hardware accelerator. Although there have been some successes using this approach [3,4], the performance of today's simulators could not support a system as large as an ATM switch.

Microprocessor boards: Special boards containing the microprocessors can be used. They allow the software developers to download the embedded software on the real microprocessor and verify its operation. This approach is very useful when high level programming language compilers do not exist for the microprocessor; for verifying software performance; or for verifying use of special features contained only on the specific processor. The development environment on such systems is generally not as mature as is present on UNIX workstations. Systems containing custom ASICs must wait for the real components to be manufactured or alternate mechanisms must be developed for verifying the interface to them.

Hardware emulators: These are similar to specialized microprocessor boards but usually also include a development environment on a host computer, usually a PC or UNIX workstation. These systems are quite useful in verifying software performance. They generally lack the ability to interface a large number of custom ASICs even if the ASICs were available.

FPGA emulators: This strategy relies on using FPGA based technology to develop a rapid prototype of the system hardware which can then be used for software verification. The first pass system integration is accomplished using the prototype and final sign-off of the real hardware for fabrication is gated by the results.

FPGA based systems generally run at slower clock rates than those in an ATM switch. As a result, one must develop alternate techniques for generating the cell streams for the datapaths or for interfacing any commercial components at these reduced rates. In the case of ATM, this can become a design challenge in itself. At a cost of $1-$2 per design gate instance, for systems the size of an ATM switch, the cost can be prohibitive. Similar to real lab systems, such a prototype would generally contain no development environment for the software developers. It would be a single resource which must be shared by a number of software developers. This approach is also gated by the availability of an RTL description of all the ASICs. A complete software load is required, which in a concurrent development environment, may not be ready in time.

This approach is better suited to the design of systems which contain a relatively small number of custom ASICs, and must support a large existing base of software (e.g. successive generations of
microprocessors). They are better suited for verifying that hardware can execute the software as opposed to allowing software developers to debug their software.

Software emulators: A software emulator can be developed for the hardware being designed. The emulator can be run with the embedded software, and mimic the relevant hardware functions. A software emulator can be brought on line very early in the design and used throughout the life cycle of the product, even after the real hardware is available. With a UNIX host based development environment, software developers would not be competing for a scarce lab resource and would be using an existing and familiar development environment. Multiple instances of the system, a requirement for verifying functions such as protection switching, can easily be created.

Such systems take advantage of system architecture to ensure that the development of the emulator is manageable and not of the same order of magnitude as the actual hardware. A software emulator can only be used to verify the function of the embedded software. Performance verification and detailed hardware-software timing relations need to be verified using a microprocessor board or at integration time.

What is missing from all of these methods is a strategy for verifying that specifications are correct prior to the start of implementation. They all require a considerable amount of the design activity to be completed before the verification process can begin.

5.2. Techniques Used

The specific techniques used encompass a number of the options listed. We addressed the goals in two ways. First through the use of front end modeling to ensure that the product interface specification was correct. Second by using a software emulator, system integration can start before the hardware is available.

The approach is geared specifically to verifying the functionality of the ECS. It is assumed that the functionality of the hardware itself and the SCS are verified through alternate means.

The specific techniques used on this projects were

- Early development of hardware and software models specifically for software development.
- Development of behavioural and/or RTL models of all hardware components and ASICs in the design.
- Instantiation of ASIC synthesizable RTL descriptions into system simulations to verify ASIC system conformance.
- Development of a UNIX host, based software emulator of the hardware for the development and verification of the embedded software.

The major difference is in the use of the first and forth technique for ensuring the hardware-software interface was correct. The techniques used in the hardware flows are very similar to the practices advocated by top-down design strategies and not described further.

The SE control functions are closely linked to the hardware. Due to the complexity of the software, both the SCS and ECS were modeled using ObjecTime (OT) [5]. The goal is similar to the use of high level models in the design of hardware. The ObjecTime models are used to verify a model of the SCS before it is developed. The software development environment provided by OT can also be used to evolve the model into the real code.

Given that the software was being modeled prior to development, an OT model of the hardware was also developed. Such a model allows software developers to verify their software model against a model of the hardware. The team developing the OT hardware model was different than the team developing the software OT model in order to gain from independent interpretations of the specification.
In general, the performance of a full function OT model of such a large hardware system would render it useless. In examining the system architecture, the only hardware functionality that is really seen by the software is the mechanisms for accessing the ASIC registers and the generation of interrupts. These aspects of the hardware can be modelled quite easily in OT.

An OT model of the system cannot only be used to verify the hardware-software operational model, but also for training. During the course of a project additional software developers will join the team. The model provides a fast and accurate mechanism for new members to learn about the design. It can also be used to resolve any uncertainties in the specification.

In addition to OT models to verify the front end of design, we also decided to develop a software emulator for the hardware to be used to verify the real software. The emulator is similar to the OT model, except that it is implemented in C/C++ and has significantly better performance than a comparable ObjectTime model. The emulator runs on a host UNIX workstation and mimics only the hardware functionality that is seen by software.

Figure 3. SE layered architecture.

To obtain a better understanding of the applicability of a software emulator, we need to take a more detailed look at the system architecture (Figure 3). In the real system, the SE software tasks communicate with the real hardware components through a series of tasks referred to as the Hardware Adaptation Layer (HAL). HAL tasks are used to drive the microprocessor bus and translate requests to specific devices into the physical addresses used to represent them.

In an emulated system (Figure 4), the lower sections of the HAL tasks are stubbed out and replaced by a socket connection to a process which is emulating the software relevant aspects of the hardware. Through the use of operator overloading in C++, the SE source does not need to be modified to accomplish this.

Using a UNIX based simulator, RTOSim, for the RTOS, the SE software can be run on a UNIX platform, communicating with the emulated hardware through UNIX sockets. This is a highly flexible structure. If a microprocessor development board is used, the emulator can be connected to the board through standard external ports (e.g. RS-232) (Figure 5). The structure also gives every software developer access to a UNIX based development environment for carrying out the functional verification of the embedded software.
6. Results

In the execution of the work to date, a number of results have been obtained.

The front end modeling work was instrumental in resolving a number of software system issues. We were also surprised by the significant windfall benefits obtained by having independent modeling teams. A significant number of issues related to specification uncertainty and misinterpretation were caught due to this. Having a software emulator for the hardware also removed the software team’s critical dependence on real hardware. Hence the software delivery schedule was relatively immune to changes in the hardware schedule. They are able to verify all except for the most detailed interactions prior to entering the lab.

Due to the independent structure of the emulator, the actual SCS can be run against the real ECS with the intervening hardware functions properly modelled.

A number of issues are not addressed. The correctness of the emulator must be verified through other means. The emulator can only be used to verify functionality. Alternate mechanisms are still required for verifying any detailed timing or cycle protocols between the software and hardware.

7. Conclusions

The paper has described the complexity of verifying the hardware-software interface of a large ATM switch. It is clear from the discussion that brute force techniques cannot be used. Rather the strategies used must take advantage of the system architecture to abstract out hardware details. It is also apparent that the techniques used for verifying a software interface to hardware are different than those used to verify that hardware will correctly execute existing software.

The major conclusion from the work is that any strategy used must be developed in the light of the hardware and software development schedules and who will be the primary customer of the work. The development environments for software and hardware are totally different. You cannot verify an interface between the two simply by combining the two development strategies into one.

9. References


