FORMAL VERIFICATION OF AN SOC PLATFORM PROTOCOL CONVERTER

Jounaïdi Ben Hassen and Sofiène Tahar

Concordia University
Electrical and Computer Engineering
Montreal, Quebec, Canada

ABSTRACT

In this paper we investigate the formal verification of the Memory Manager block of a System-on-a-Chip platform Protocol Converter using the FormalCheck tool of Cadence. The Memory Manager represents the main block of the protocol converter and is responsible for the reception of packets and their treatment for conversion. For the verification, we first extracted some constraints to define the environment for the Memory Manager. Then, we specified a number of relevant liveness and safety properties expressible in FormalCheck. Through extensive verification under the defined set of constraints, we have been able to find a few bugs in the design that were omitted by simulation. This experience demonstrates the usefulness of formal verification as complement to traditional verification by simulation.

1. INTRODUCTION

The increasing complexities of hardware designs have made verification and error detection on the critical path of the design process. Moreover, some of these errors may cause catastrophic loss of money, time, or even human life. This is in particular a serious problem for System-on-a-chip (SoC) designs which may contain processor cores, custom logic, memory and IP (Intellectual Property) blocks on a single chip. Traditionally, simulation is used to verify the "correctness" of a design. However, simulation is no longer able to keep pace with the increasing complexity of hardware designs. To overcome this difficulty, formal hardware verification methods [1] are now being deployed, and became useful tools for detection of functional design errors. By using formal verification alongside the design efforts, the overall design cycle can be reduced. This ensures a maximum design coverage while maintaining a high degree of confidence in the verification result. The objective of formal verification is to verify that the design model conforms to an abstract specification, consisting of a set of properties. Together, these properties describe (partially) the intended functionality of the design. Formal verification techniques have proven their efficiency by verifying industrial size systems.

This paper aims to explore the verification by model checking of the Memory Manager component of an SoC platform Protocol Converter System and to show that formal methods are strong enough for the verification of complex designs. The Protocol Converter system was designed by the "Groupe de Recherche en Micro-électronique" at the École Polytechnique de Montréal [2]. The architecture of the Memory Manager is described at the Register-Transfer Level coded in VHDL and represents the main block of the protocol converter system. The Memory Manager is composed of five modules: a Memory Manager Controller, an Address Counter Register, a Data Counter Register, a Packet Counter Register and a Packet Assembler. The model checking of the Memory Manager has been conducted using the FormalCheck tool of Cadence [3]. First, we extracted some constraints to define the environment for the Memory Manager. Then we specified a number of relevant liveness and safety properties expressible in FormalCheck. We successfully accomplished the properties verification under the defined set of constraints. This verification enabled us to find a number of bugs in the design that were not caught during the simulation process. This experience demonstrates how formal verification techniques can powerfuly complement traditional verification by simulation.

2. THE MEMORY MANAGER BLOCK

The Protocol Converter System at hand is based on an SoC platform developed at the École Polytechnique de Montréal [2]. The main advantage of such platform is its flexibility. In fact, its modularity allows the addition, the change or the drop of some modules without affecting the global architecture of the system. The Protocol Converter accepts incoming packets from a physical bus, converts their protocols and then sends them back through the physical interface. The Protocol Converter System is subdivided into three blocks as shown in Figure 1. The first block, composed of a Memory Manager and a Controller, is specialized in the reception of packets and preliminary treatments for the conversion. The second is responsible for the transmission of converted packets, while the third performs the

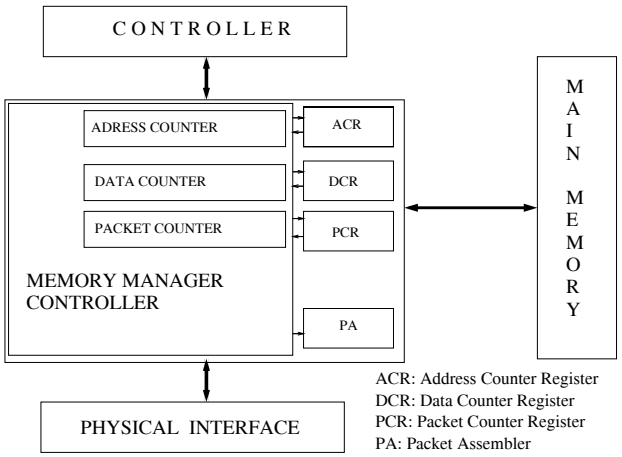

Fig. 1. The Memory Manager Architecture

conversion of packets coming from the first block and sends them back to the second block.

Packets come through a physical interface which communicates with the Memory Manager by transmitting data and some control signals. Upon reception, the Memory Manager stores incoming data in the Main Memory and transmits to the Controller the characteristics of the packet in transmission. These characteristics include the address of the packet in the Main Memory, the protocol of the packet and its size. The communication between the Memory Manager and the Controller is insured via the Memory Manager Controller. Once the protocol conversion is done, the Controller asks the Memory Manager, to remove the converted packet from the Main Memory. So, the address of this packet and some other control signals are transmitted to the Memory Manager. The functionality of the Memory Manager is insured by its different components. Besides the Memory Manager Controller, the Memory Manager has three registers and a Packet Assembler. Intuitively, the Memory Manager Controller is responsible for the communication between the Memory Manager and the Controller. The registers are used as counters of addresses and packets. The Packet Assembler is a module that concatenates packet's address and data.

3. ENVIRONMENT CONSTRAINTS

In FormalCheck, primary signals are assumed to be nondeterministic, meaning they could acquire any value within their range on any edge of the clock. However, in most cases correct design operation is allowed on a single edge of the clock. For this reason, properties should be observed using the appropriate clock edge. When we deal with formal verification by model checking, we should pay great attention to the set of defined constraints. Since constraints are generally used to simulate the environment on which the system operates, they are the most critical task. This set should be as complete as possible to describe the exact behavior of the environment. However, it should not contain more constraints than necessary to avoid obtaining an over-constrained system. In this work, we have established twelve constraints. A full list of all constraints is given in [4]. They include, for instance, a constraint to define the system clock, a constraint to define the reset of the system and some others to describe the duration and the synchronization of input signals (with respect to the clock system), as well as to describe the interaction between the external environment and the Memory Manager block. For example, the signal `Phyn_Sop` is activated during one cycle of the system's clock rising edge (`Sys_Clk=rising`) to indicate the start of packet submission. This can be expressed by the following *safety constraint*:

```
Constraint OneClockSop
  Type: Never
  Assume Never:(Phyn_Sop=1) and (WasSop=1)
```

This constraint means that the signal `Phyn_Sop` can never be activated during two clock cycles, where `WasSop` is a *state variable* that indicates if the signal `Phyn_Sop` is already activated:

```
WasSop: Range 0 to 1
  Initial: 0
  if (Phyn_Sop=1) and (Sys_Clk=0)
    then WasSop := 1;
    else WasSop := 0;
  end if;
```

The signal `Phyn_Sop` changes value only on the rising edge of the system's clock and can be expressed by the following safety constraint:

```
Constraint SyncSop
  Type: Always
  Assume Always: (Sys_Clk=rising) or
  (Phyn_Sop=stable)
```

meaning that the signal `Phyn_Sop` changes value only on the rising edge of the system's clock. In other words, if `Phyn_Sop` changes its value, forcibly the system's clock `Sys_Clk` is rising. Otherwise, `Phyn_Sop` remains stable.

4. PROPERTIES SPECIFICATION

After establishing the proper environment, and defining all needed constraints, we have specified sixteen *queries* (set of constraints and properties) of the Memory Manager, including liveness and safety properties. These properties have been defined based on the design textual documentation and the test benches given in [2]. In the following, we describe three samples for illustration purposes. A full list of all defined properties is given in [4]. Our main goals were to verify (1) the global reset of the Memory Manager, (2) the duration and the synchronization of output signals (with

respect to the system's clock) and (3) the response of the Memory Manager according to received signals. In other terms, we wanted to verify that each time the Memory Manager receives a packet from the Physical Interface, it stores it in the Main Memory and informs the Controller. We wanted also to verify that the Controller is informed about all stored packets and vice-versa. For example, according to the Memory Manager documentation [2], when the Memory Manager begins the reception of a packet, it informs the Controller by activating the signal `Mgr_Sop` during one cycle. This activation should start at the falling edge of the system's clock defined by the signal `Sys_Clk`. In this query, we verify the duration of the signal `Mgr_Sop` (first property) and its synchronization with the clock (second property). In FormalCheck, this query is expressed as follows:

```
Query: Verif_Mgr_Sop
 Property: OneClkMgr_Sop
   Never: (Mgr_Sop=1) and (WasMgr_Sop=1)
 Property: SyncMgr_Sop
   Always: (Sys_Clk=falling) or
   (Mgr_Sop=stable)
 WasMgr_Sop: Range 0 to 1
  Initial: 0
  if (Mgr_Sop=1) and (Sys_Clk=1)
    then WasMgr_Sop := 1;
    else WasMgr_Sop := 0;
  end if;
```

where `WasMgr_Sop` is a state variable that indicates if the signal `Mgr_Sop` is already activated or not.

5. EXPERIMENTAL RESULTS

All verifications in this project were executed on an *Ultra 5* Sun workstation with 256 MB RAM and UNIX operating system. For all properties, we used *Symbolic (BDD)* as algorithm and *1-Step* as reduction technique [3]. We found these modes effective enough to conduct the verification process. The experimental results are shown in Table 1, including the status of the queries verification, the number of reached states (RS), the average state coverage (SC), the CPU time (real time) in seconds and the memory usage in MB. From Table 1, we can see that the system reset and signal duration properties are verified, but some others failed. For instance, when the Physical Interface starts a packet transmission, there is no guarantee that this packet will be detected by the Memory Manager and thereby will be stored in the Main Memory. In this case, the Controller will not be informed about this transmission and the packet will be lost. Intuitively, it seems logic that if the Memory Manager will not detect some packet transmissions, then it will not detect the end of these transmissions or the occurrence of some errors. In such cases, the Controller will not be informed by these activities performed by the Physical Interface. In fact, there is no direct dialogue between the Controller and the Physical Interface. Thus, all information concerning the packet transmission is transmitted to the Controller via the Memory Manager.

Table 1. Summary of Experimental Results

Query	Status	RS	SC (%)	CPU	Mem
Q1	Verified	2.68^{+7}	87.50	70	14.37
Q2	Verified	3.21^{+3}	96.77	10	2.32
Q3	**Failed**	5.09^{+3}	88.98	12	10.62
Q4	Verified	3.14^{+3}	96.77	9	2.32
Q5	**Failed**	5.25^{+3}	10	9	9.54
Q6	Verified	7.85^{+6}	97.06	9	2.32
Q7	**Failed**	8.89^{+6}	98.08	40	10.85
Q8	Verified	3.08^{+3}	96.77	8	2.32
Q9	Verified	4.37^{+5}	100	18	9.56
Q10	Verified	5.98^{+5}	96.97	15	9.66
Q11	Verified	4.38^{+5}	98.53	15	9.68
Q12	Verified	2.15^{+5}	100	18	9.66
Q13	Verified	2.15^{+5}	100	17	9.67
Q14	Verified	2.82^{+9}	98.41	12	2.34
Q15	Verified	2.82^{+9}	98.39	13	2.34
Q16	**Failed**	2.82^{+9}	98.44	192	16.63

In [2], there is an illustrated case in which packets are lost. When the Main Memory is full and no more space is available to store incoming packets, the Memory Manager will simply reject incoming packets without informing the Physical Interface. In such case, the Physical Interface will continue sending packets to the Memory Manager. However, according to [2], this situation was defined as the only case in which packets are lost. In our work, FormalCheck gives a counterexample and shows that in spite of memory space availability, incoming packets can still get lost. The failure of Q16 means that some packets are stored into the Main Memory but the Memory Manager does not inform the Controller about them once they are received. This means that some packets will remain in the Main Memory without conversion. Furthermore, the Controller will never ask for the suppression of these packets.

6. RELATED WORK

Related case studies, where model checking techniques were used to verify commercial products, are too numerous to be listed here. In the following, we elaborate on a few of them. For instance, in [5], FormalCheck was used to verify the implementation of a SCI-PHY (Saturn Compatible Interface for ATM-PHY devices) Level 2 protocol engine, commercialized by PMC-Sierra, Inc. From the set of eleven established properties, only one error has been detected. In comparison, the design considered in our project

is more complex and presents a more generic behavior. In [6], the model checking tool VIS (Verification Interacting with Synthesis) [7], was adopted for the verification of an Asynchronous Transfer Mode (ATM) switch used for real applications in the Cambridge Fairisle network. This earlier work shows how VIS can partially verify large size circuit design by using specific reduction, abstraction and property division techniques. In [8], the same ATM switch fabric design is verified in FormalCheck and a comparison between the two hardware verification tools is given. From the experimental results in [8], it was shown that FormalCheck is faster than VIS and that the memory usage in FormalCheck is less than that in VIS for all verified properties. Moreover, no manual reduction or property composition were required in FormalCheck. This justifies again our choice of FormalCheck to verify our system. In [9], VIS was used for the verification of an ATM ring (ATMR) media access control (MAC) protocol. VIS was also used in [10] to formally verify a commercial product of PMC-Sierra, Inc. In this latter work, a design error which could lead the system to a deadlock state was detected. However, properties that involve the introduction of time delays were not verified because unlike FormalCheck, as used in our project, VIS does not support timed Verilog models. In [11], the MDG (Multiway Decision Graph) [12] tool was used in the formal verification of a Telecom System Block commercialized by PMC-Sierra, Inc., which processes a portion of the SONET (Synchronous Optical Network) line overhead of a received data stream. While the MDG tool possesses efficient features for data abstraction, it does not support neither Verilog nor VHDL to be considered in a complex project like ours.

7. CONCLUSION

In this study, we explored the formal verification by model checking of the Memory Manager Component of the Protocol Converter system. Our experimental results demonstrated the presence of many residual bugs in the design. These errors can lead to serious problems since they cause the non-conversion of many received packets and the non-liberation of the Main Memory. Despite the important effects of such situations on the functionalities of the Protocol Converter system, these errors were not detected by extensive post-design simulation efforts.

The main contribution of our work is the emphasis on the importance of formal methods for design verification. In our case, we were able to detect errors that were not detected by simulation. However, we should mention that formal techniques are not by themselves an alternative to verify if a design is correct with respect to a specification. They should be used with simulation as a complementary process to insure a maximum error detection.

In summary, our work joins other successful industrial-sized case studies in using model checking for hardware verification. We believe to have contributed fostering the evidence that model checking is now powerful enough to be widely used in industry to help in the verification of developed complex hardware designs.

8. REFERENCES

[1] C. Kern and M. Greenstreet, "Formal Verification in Hardware Design: A Survey," in *ACM Transactions on Design Automation of Electronic Systems*, Apr. 1999, vol. 4, pp. 123–193.

[2] S. Carniguian et al., "Intégration et Vérification d'un convertisseur de protocoles," Tech. report, Département de génie életrique, École Polytechnique de Montréal, Canada, 2002.

[3] Cadence, *Formal Verification Using Affirma FormalCheck, Version 2.4*, Aug. 1999.

[4] J. Ben Hassen and S. Tahar, "Formal Verification of a Protocol Converter Memory Manager using FormalCheck," Tech. report, Concordia University, Montreal, Canada, Apr. 2003.

[5] L. Barakatain and S. Tahar, "Functional Verification of a SCI-PHY Level 2 Protocol Engine," in *Proc. IEEE Int. Conf. on Information, Communications and Signal Processing*, Singapore, Oct. 2001.

[6] J. Lu et al., "Model Checking of a Real ATM Switch," in *Proc. IEEE Int. Conf. on Computer Design*, Austin, Texas, USA, Oct. 1998, pp. 195–198.

[7] R.K. Brayton et al., "VIS: A System for Verification and Synthesis," in *Proc. Conf. on Computer Aided Verification*, Rutgers, New York, USA, Jul. 1996, pp. 428–432.

[8] L. Barakatain and S. Tahar, "Model Checking of the Fairisle ATM Switch Fabric using FormalCheck," in *Proc. IEEE Canadian Conf. on Electrical and Computer Engineering*, Toronto, Canada, May 2001, pp. 907–912.

[9] H. Peng and S. Tahar, "Hardware Modeling and Verifiation of an ATM Ring MAC Protocol," in *Proc. IEEE Int. Conf. on Microelectronics*, Teheran, Iran, Nov. 2000, pp. 21–24.

[10] P. Murugesh and S. Tahar, "Formal Verification of the RCMP Egress Routing Logic," in *Proc. IEEE Int. Conf. on Microelectronics*, Kuwait City, Kuwait, Nov. 1999, pp. 89–92.

[11] M.H. Zobair and S. Tahar, "Formal Verification of a SONET Telecom System Block," in *Proc. Int. Conf. on Formal Engineering Methods*, Shanghai, China, Oct. 2002, pp. 447–458.

[12] F. Corella et al., "Multiway Decision Graphs for Automated Hardware Verification," in *Formal Methods in System Design*, Feb. 1997, vol. 10, pp. 7–46.