An Executable Specification of the PCI-X Bus Standard in AsmL

Haja Moinudeen, Ali Habibi and Sofiène Tahar
Electrical and Computer Engineering Department
Concordia University
1455 De Maisonneve, West
Montreal, Quèbec, H3G 1M8, Canada
E-mail: {haja_m,habibi,tahar}@ece.concordia.ca

Abstract

In this paper, we describe an executable formal specification of the PCI-X bus standard using the Abstract State Machines Language, AsmL. PCI-X, is the latest extension of the PCI local bus that is designed to meet increased I/O demands of recent technologies. The actual specification of PCI-X, provided by the PCI Special Interest Group (PCI-SIG), is informal (in natural English). Hence, it is prone to inherent problems of incompleteness, inconsistency and ambiguity. In our approach, we first model the bus in UML and then map it to AsmL. This AsmL model can be executed using the Asmlt tool that can generate the Finite State Machine (FSM) of the model. Such FSM can be of great use for verification purposes.

Keywords— *Formal Specification, PCI-X, Abstract State Machine Language (AsmL), UML.*

1 Introduction

PCI-X [11] is the latest implementation of PCI [11]. It was adopted as industrial standard ratified by the PCI-SIG. Using the same 64-bit architecture as the current standard, PCI-X has tremendously increased the clock speed to 533 MHz, allowing transfer speeds up to 4 GB/sec and it is backward compatible with standard PCI cards. Furthermore, the PCI-X bus plays a vital role in today's System-On-Chip (SoC) designs involving various components connected using high-speed standard buses. Such bus protocols, however, are not easy to specify correctly due to their complex nature. The original specification of PCI-X from PCI-SIG is given in the English natural language. Conventional informal specifications, however have inherent problems such as misunderstanding. In contrast, formal specifications are good solutions for the aforementioned problems, but they are not being widely used in the industry because they require an advanced mathematical and theoretical knowledge [5].

In this paper, we present a three-step specification methodology of the PCI-X standard bus. We first model the bus in UML [13] using class and sequence diagrams in order to have a clear view of the design modules and their interactions. The UML representation is then mapped to the Abstract State Machines Language, AsmL [3]. Our preference to this relatively new language is due to its object-oriented nature, simplicity, precision, understandability, executability, inter-operability and scalability [3]. Moreover, AsmL is integrated in the .NET framework and Microsoft development tools allowing both the execution of AsmL specifications as stand-alone applications. AsmL specifications can also be used to generate an FSM (Finite State Machine) by the AsmL tester (Asmlt) [8]. The FSM generation is of interest for the following three reasons: (1) it can be used to check the correctness of an implementation of the PCI-X bus with respect to its specification using conformity testing; (2) it can be used as input to a model checking tool to verify certain properties of the bus; and (3) it can present a good coverage metric for simulation based verification (nodes coverage, paths coverage, etc.).

There exists a few related work to ours in the open literature. For instance, Shimizu *et al.*[12] presented a specification of PCI bus as a Verilog monitor. This approach, however, makes any modification or refinement of the model complex since the level of specification of the PCI is very low. Oumalou *et al.* [10] implemented the PCI bus in SystemC [9]. First, they specified the bus in UML; then, mapped the UML representation to AsmL and finally translated the AsmL code into SystemC. In [4], Habibi *et al*, specified and implemented the Look-Aside Interface [7] using the same approach as in [10]. Our work is similar to the work of [10] and [4], but distinguishes itself by correctly specifying the latest high-speed bus standard (PCI-X) including its very complex transaction rules. Furthermore, we provide an executable specification that can be used to explore the design space before implementation to check unintended behavior of the system. Once a system has been implemented, we can also run our specification in parallel with the implementation to find out whether the implementation produces the expected results. Besides, the Asmlt tool can automatically generate behavioral test cases from the specification.

The rest of the paper is organized as follows: In Section 2, we present the general architecture and an overview of PCI-X transactions. Section 3 describes our specification methodology. Section 4 concludes the paper with hints to some future work.

2 The PCI-X Bus

Improvements in processors and peripheral devices have caused conventional PCI technology to become a bottleneck to performance scalability. The introduction of the PCI-X technology has provided the necessary bandwidth and bus performance needed to avoid the I/O bottleneck,

0-7803-8886-0/05/$20.00 ©2005 IEEE

CCECE/CCGEI, Saskatoon, May 2005

thus achieving optimal system performance. For instance, version 2.0 of PCI-X specifies a 64-bit connection running at speeds of 66, 133, 266, or 533 MHz, resulting in a peak bandwidth of 533, 1066, 2133 or 4266 MB/s, respectively.

PCI-X provides backward compatibility by allowing devices to operate at conventional PCI frequencies and modes. Moreover, PCI-X peripheral cards can operate in a conventional PCI slot, although only at PCI rates and may require a 3.3 V conventional PCI slot. Similarly, a PCI peripheral card with a 3.3 V or universal card edge connector can operate in a PCI-X slot, however the bus clock will remain at a frequency acceptable to the PCI card. Figure 1 shows the general architecture of PCI-X with one Initiator (Master) and Target (Slave). There is an arbiter that performs the bus arbitration among multiple Initiators and Targets. Unlike the conventional PCI bus, the arbiter in PCI-X systems monitors the bus in order to ensure good functioning of the bus.

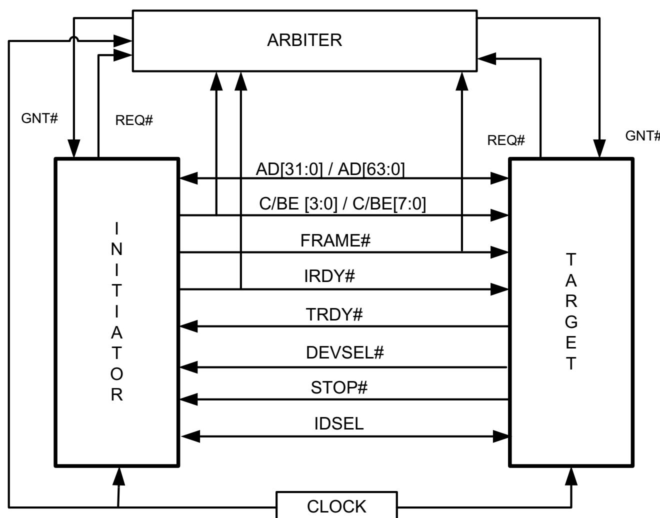

Figure 1: General Architecture of PCI-X.

PCI-X supports two modes of transactions: Mode 1 and Mode 2. Mode 1 operates at either 66 or 133 MHz and uses a parity error checking scheme. The higher transfer rates (266 or 533 MHz) in PCI-X 2.0 are defined in Mode 2, which uses ECC as its error correcting scheme. Split Transactions in PCI-X replace Delayed Transactions in conventional PCI [11]. If a Target cannot complete the transaction within the Target initial latency limit, the Target must complete that transaction as Split Transaction. If the Target meets its initial latency limits, it can optionally complete the transaction immediately.

Both PCI-X Initiator and Target has a pair of arbitration lines that are connected to the Arbiter. When an Initiator requires the bus, it asserts REQ#. If the arbiter decides to grant the bus to that Initiator, it asserts GNT#. FRAME# and IRDY# are used by the Arbiter to decide the granting of an Initiator request for the bus. Unlike PCI, the targets can only insert wait states by delaying the assertion of TRDY#. TRDY# and IRDY# have to be asserted for a valid data transfer. An Initiator can abort the transaction either before or after the completion of the data transfer by de-asserting the FRAME# signal. In contrast, a Target can terminate a bus transaction by asserting STOP#. If STOP # is asserted without any data transfer, the Target has issued a retry and if STOP# is asserted after one or more data phases, the Target has issued a disconnect. Unlike PCI, the Target has also REQ# and GNT# that are connected to the Arbiter. This facilitates the Split Transaction of PCI-X which does not exist in conventional PCI.

3 Specification Methodology

Figure 2 depicts our specification methodology of the PCI-X bus standard. We start with the informal specification given in [11], where all protocol rules have been described. The core components of the protocol, the different types of transactions and respective transaction rules are identified. Thereafter, we specify the bus semi-formally using UML's class and sequence diagrams. From the sequence diagrams, we write the formal (executable) specification of the bus using AsmL and then compile it using the AsmL tool (Asmlt). The Asmlt has its own FSM generation algorithm from the AsmL code and can generate test cases in C# [1]. Such C# test cases can be used to verify the model by simulation.

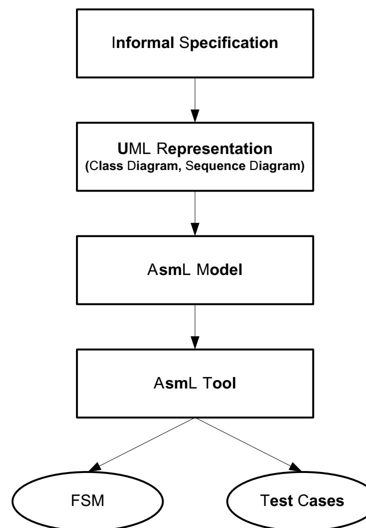

Figure 2: PCI-X Specification Methodology.

3.1 UML Representation

The core components of the bus viz, Initiators, Targets, Arbiters, PCI-X bus are represented as classes, where specific instances of the components are called as objects. In

addition to these four components, we also added another component, the Simulation Manager (SimManager), in order to have a notion of the Clock. The detailed description of these components (classes) is discussed in Section 3.2. Figure 3 shows the class diagram of the PCI-X bus. As it can be seen from the figure, we have five classes and each class has its own data members and methods.

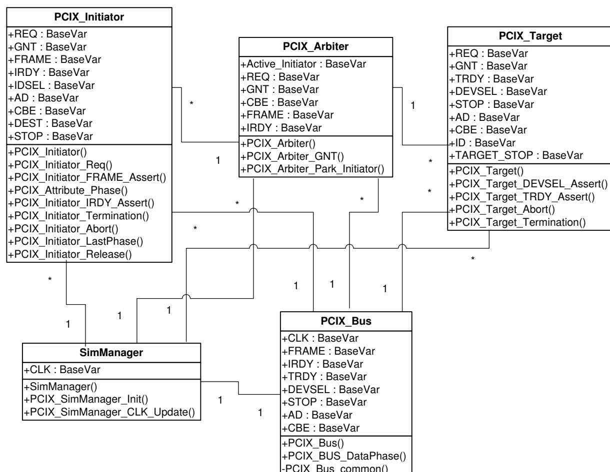

Figure 3: Class Diagram of PCI-X.

In Figure 4, we show the protocol sequence of the Mode 1 transaction of PCI-X using a sequence diagram. Figure 4 is a best case scenario of Mode 1 transactions, with one Initiator and one Target and without any wait states. In the first clock cycle, the Initiator asserts the REQ# signal to get the control of the bus. The Arbiter asserts the GNT# signal to that Initiator. In the third clock cycle, the address phase takes place and also the FRAME# signal is asserted by the Initiator to signal the start of the transaction. In the next clock cycle, the attribute phase takes place, where additional information included with each transaction is added. In clock cycle N+4, the DEVSEL# signal is asserted by the Target and in the next clock cycle, the data phase is started with the assertion of the IRDY# and TRDY# signals by the Initiator and the Target, respectively. Before the last data phase, the FRAME# signal is de-asserted to signal the completion of the data transfer and in the termination phase all the other signals are de-asserted. In order to represent the clock constraints of the PCI-X transaction, we added an additional operator @. This operator is used to specify at which clock cycle a particular action should occur.

3.2 AsmL Model

In this section, we discuss our approach of mapping the UML representation of PCI-X into an AsmL model by showing some of the important methods of our AsmL code. We used AsmL's class features to model all four components discussed in the previous section. Each of these has its own data members (signals) and methods (behavior) in

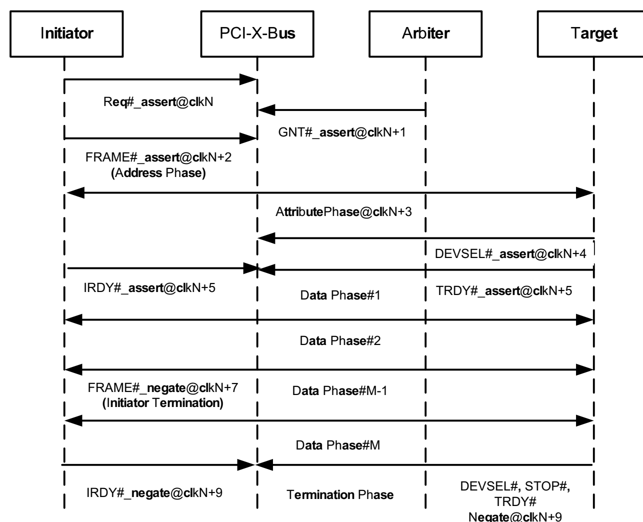

Figure 4: Sequence Diagram of Mode 1 Transaction.

addition to the Constructors. Constructors are used to initialize the objects and they are invoked whenever an object is created. We also used enumeration features (*enum*) of AsmL to model different modes of PCI-X, different types of transaction phases, the state of the system and the clock. (see Figure 5).

```
enum SystemStatus            enum Clock
  INIT                         CLK_UP
  STARTED                      CLK_DOWN

enum Transaction_Phase       enum Transaction_Mode
  IDLE_PHASE                   MODE_1
  ADDR_PHASE                   MODE_2
  ATTR_PHASE
  TGT_RES_PHASE
  DATA_PHASE
  INR_TER_PHASE
  TURN_AROUND_PHASE
```

Figure 5: Enumeration in the AsmL Model.

In addition, we exhaustively used the *require* and ”*:=*” statements of AsmL in our design approach. *require* is the pre-condition statement in AsmL used to check if a certain condition is satisfied or not, and ”*:=*” is the update statement used to change the system state. Figure 6 shows the Arbiter grant method (*PCIX_Arbiter_GNT()*). In order to grant the bus to the requested Initiator, this method has the following pre-conditions (*require*) to be met: (1) there must be at least one Initiator requesting the bus and that Initiator has not been granted the bus at the time of the request; (2) the clock is on the rising edge; and (3) the mode can be either Mode 1 or Mode 2. If these pre-conditions are met, the Arbiter updates the GNT# signal.

Figure 7 shows the Initiator termination method (*PCIX_Initiator_Termination*). This method basically signals the end of a transaction if $BYTECOUNT$ is less than 2. This signaling is done by de-asserting the FRAME#

```
public PCIX_Arbiter_GNT()
   require
(exists x in Initiators where x.REQ =  true and x.GNT = false) and  me.GNT =
false and Smanager.CLK = CLK_UP and (Mode = MODE_1 or Mode = MODE_2)
   me.Active_Initiator := min y | y in Initiators_Range where  (Initiators(y).REQ =
true)
   me.REQ := true

 require me.REQ = true and me.GNT = false
   me.GNT := true
  Initiators(Active_Initiator).GNT := true
```

Figure 6: Arbiter GNT Method.

signal and updating the transaction phase to Initiator Termination Phase (INI_TER_PHASE). This method has some pre-conditions that need to be true so that it can terminate the transactions. The pre-conditions (*require*) are the following: Initiator's REQ#, GNT#, FRAME#, IRDY#, DEVSEL#, TRDY# are asserted, $BYTECOUNT$ is less than 2, and the clock is on the rising edge. If all the above pre-conditions are true, this method updates the FRAME# signal to false and the phase to *INI_TER_PHASE* using the ":=" operator. After this FRAME# signal de-assertion, the Initiator's last phase method (*PCIX_Initiator_LastPhase()*) is invoked.

```
public PCIX_Initiator_Termination()
require me.GNT = true and me.REQ = true and me.FRAME = true and
        me.IRDY    = true and Bus.TRDY = true and Bus.DEVSEL = true   and
BYTECOUNT < 2 and me.STOP = false and Smanager.CLK = CLK_UP
   me.FRAME := false
   Bus.FRAME := false
   Phase := INR_TER_PHASE
```

Figure 7: Initiator Termination Method.

Figure 8 shows the clock update method of the AsmL code. It checks if the system and simulation are started in order to update the clock.

```
public PCIX_SimManager_CLK_Update()
   require SystemFlag = STARTED and SimStatus = RUNNING
   if CLK = CLK_UP then
      CLK := CLK_DOWN
   else
      CLK := CLK_UP
```

Figure 8: Clock Update Method.

Figure 9 shows the data phase method of the AsmL code. The method decreases the $BYTECOUNT$ by 1 based on the following pre-conditions: IRDY#, DEVSEL#, TRDY# are asserted, the phase has the value *DATA_PHASE*, the clock is on the rising edge, and the system started running.

```
public PCIX_BUS_DataPhase()
   require SystemFlag = STARTED and SimStatus = ON_INIT and
           Bus.IRDY =   true and Bus.TRDY = true and Bus.DEVSEL = true
           and   Smanager.CLK = CLK_UP and Phase = DATA_PHASE
BYTECOUNT := BYTECOUNT - 1
```

Figure 9: Data Phase Method.

Once the modeling of the bus in AsmL was completed, we used the Asmlt tool to execute the AsmL code and generated the model's FSM. Details of our AsmL implementation, UML representations and the generated FSM of the PCI-X standard bus can be found in the project website[1].

4 Conclusions

In this paper, we presented an executable specification of PCI-X bus standard using UML representation and AsmL. Starting from the informal description of the bus standard, we provided a formal specification using the UML class and sequence diagrams then mapped the sequence diagrams into an AsmL model. The AsmL model is executed using Microsoft's Asmlt tool and an FSM of the model is generated. In a future work, the generated FSM can be used as input to verify formally using model checking tools such as SMV [6] or FormalCheck [2]. The developed model can also be used to check the correctness of an implementation of the PCI-X bus with respect to its specifications using conformity testing in Asmlt.

References

[1] C# Language Specification, Microsoft Incorporation. http://msdn.microsoft.com/vcsharp, 2004.

[2] Cadence. Affirma Formalcheck version 2.6, 2000.

[3] Y. Gurevich, B. Rossman, and W. Schulte. Semantic Essence of AsmL. Technical report, Microsoft Research, MSR-TR-2004-27, March 2004.

[4] A. Habibi, A.I. Ahmed, O. Ait-Mohamed, and S. Tahar. On the design and verification of the look-aside interface. In *Proc. Design Automation and Test in Europe*, Munich, Germany, March 2005 (to appear).

[5] A. V. Lamsweerde. Formal Specification: A Roadmap. In *Proc. of Conference on The Future of Software Engineering*, pages 147–159, Limerick, Ireland, June, 2000. ACM Press.

[6] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[7] Network Processing Forum. *Look-Aside (LA-1) Interface, Implementation Agreement, Revision 1.1.* Kluwer Academic Publishers, April 15, 2004.

[8] Microsoft Research Foundations of Software Engineering. Asml for microsoft .net. http://www.research.microsoft.com/foundations/asml.

[9] Open SystemC Initiative. www.systemc.org, 2004.

[10] K. Oumalou, A. Habibi, and S. Tahar. Design for verification of a PCI bus in SystemC. In *Proc. Symposium on System-on-Chip*, pages 201–204, Tampere, Finland, November 2004.

[11] PCI Special Interest Group. www.pcisig.com, 2004.

[12] K. Shimizu, D. L. Dill, and A. J. Hu. Monitor-based formal specification of PCI. In *Formal Methods in Computer-Aided Design*, pages 335–353. LNCS 1954, Springer-Verlag, 2000.

[13] Unified Modeling Language. http://www.uml.org, 2003.

[1] http://hvg.ece.concordia.ca/Research/Soc/PCIX