UML to SystemVerilog Synthesis for Embedded System Models with Support for Assertion Generation

Lun Li*, Frank P. Coyle, and Mitchell A. Thornton
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas, U.S.A.
{lli, coyle,mitch}@engr.smu.edu

Abstract

SystemVerilog encapsulates both design description and verification properties in one language and provides a unified environment for engineers who have the formidable challenge of designing and verifying present day complex systems. However, the gap between the definition of system specifications and design validation assertions that formally describe embedded system properties poses a challenge for HDL design engineers. We describe a prototype tool that utilizes UML as a system specification language and generates SystemVerilog assertions for hardware design correctness validation with a corresponding Verilog module describing the hardware. An example is provided for a controller specified in UML with the resultant Verilog-RTL description and SystemVerilog assertion file produced automatically.

1. Introduction

As Moore's Law continues to push the boundaries of hardware capabilities, embedded systems designers are confronted with the challenge of how best to draw the line between hardware and software. Functionality that was once relegated to software is now considered for the possibility of implementation in hardware while hardware components must integrate with higher-level software APIs.

To help address these problems, the area of hardware/software co-design has evolved as an architectural approach to system design where decisions concerning hardware-software allocation are deferred as late as possible within the design cycle. A Model Driven Architecture (MDA) approach is proposed in [1] where system functionality and semantics at a high level of abstraction is used and then tools are described that generate different platform specific implementations.

While MDA is technically neutral about the syntax and structure of high-level models, the *Unified Modeling Language* (UML) [2] has emerged as a common foundation for MDA modeling. Initially proposed as a unifying notation for object-oriented design, UML has added a semantic underpinning that makes it possible to build platform independent descriptions that can be used by designers and architects to make informed decisions about hardware/software tradeoffs. Since UML was initially introduced in the software domain, most commercial tools based on UML descriptions have the ability of generating software code such as Java and C++. However, no such tools are commercially available that can synthesize UML models into a *Hardware Description Language* (HDL) model directly, thus imposing a limitation for the usage of UML in embedded system design.

Additionally, it is also observed that assuring correct functional behavior is the dominating factor of a successful hardware design. Today, in many circuit design projects, up to 80% of the overall design costs are due to verification tasks.

Assertion-based verification plays an important role in today's large, complex designs. The basic function of an assertion is to specify some behavior that is expected to hold true for a given design or component. However, system specification of a design is usually given in a natural language which is hard to translate into an assertion directly. It is necessary to describe the system specification in a more rigorous and well-formed language in order to bridge the gap between a system specification and design validation assertions that formally describes such specifications.

In this paper, our main contribution is the development of synthesis and assertion generation tools that utilize UML version 1.3 as a system specification language and automatically generate assertions that can be used by a verification engineer to validate the hardware as well as a synthesizable HDL-RTL module description. We believe this is an important contribution since the description of validation for a particular design developed during the time of system specification is as important as property specification at later stages of synthesis.

The remainder of the paper is organized as follows. In Section 2, we provide background information about UML and SystemVerilog. In Section 3, our tool that comprises an automatic assertion generator and HDL synthesis based on a UML description is introduced. An example is given to demonstrate the capability of the tools. Future work and conclusions are provided in Section 4.

2. Preliminaries and Related Work

In this section, we introduce the use of UML and HDL for describing hardware and assertions that are used for circuit validation.

2.1 Unified Modeling Language (UML)

UML is a modeling language that is very well accepted in the software domain for system modeling. UML

defines twelve types of diagrams which can be divided into three categories: (i) *Structural Diagrams* which include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram, (ii) *Behavior Diagrams* which include the Use Case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and State Machine Diagram, and (iii) *Model Management Diagrams* which include Packages, Subsystems, and Models. UML has been used across a wide variety of domains, from computational to physical, making it suitable for specifying systems independently of whether the implementation is accomplished via software or hardware.

The recent addition of action semantics to UML has led to the development of executable UML (xUML) which supports the direct execution of UML models [2]. UML is supported by a wide range of tools such as IBM's Rational Toolset [3] and I-Logix's Rhapsody [4]. The exchange of models between tools is supported by the XMI standard. This standard is an XML-based description language which captures the details of UML model diagrams in a portable, machine readable format. Most UML tools can automatically generate XMI which is used as the basis for our prototype tool as described here.

2.2 SystemVerilog and SystemVerilog Assertions (SVA)

The SystemVerilog language evolved from the Verilog hardware description language as an industrial standard language to describe hardware design as well as to write assertions supporting the enormous task of verifying the correctness of a design. The basic function of an assertion is to specify a set of behaviors that is expected to hold true for a given design or component. The capability to encapsulate a design and a verification strategy in one language provides a unified environment for engineers who have the formidable challenge of designing and verifying present day complex systems. Assertions can dramatically decrease the amount of effort required to define intelligent, self-checking testbenches, and at the same time, increase the effectiveness of a testbench [6].

SystemVerilog provides two types of assertions: immediate and concurrent. Both assertion types are intended to convey the intent of the design engineer and to identify the source of a problem as quickly and directly as possible. Immediate assertions are procedural statements that can occur anywhere within **`always`** or **`initial`** blocks, and include a conditional expression to be tested and a set of statements to be executed depending on the result of the expression evaluation.

Concurrent assertions provide the ability to specify sequential behavior concisely and to evaluate that behavior at discrete points in time, usually at clock ticks (e.g. "**`posedge clk`**").

2.3 Related Work

In [10], the authors present a framework for generating VHDL specifications from a subset of UML, and a set of rules to map UML class and state diagrams to VHDL. The authors concentrate on UML behavioral models described using state diagrams, and use a mapping for class diagrams that differs from our approach. Bjhrklund and Lilius [11] generate VHDL code from UML behavioral models (state and activity diagrams) using SMDL as an intermediary language. Authors in [12] present an approach for modeling embedded systems using extended UML as well as generating SystemC code from UML class and object diagrams. In [13], Damasevicius and Stuikys examined system level design processes that are aimed at designing a hardware system by integrating soft IPs at a high level of abstraction. They combine this concept with object-oriented hardware design using UML and metaprogramming paradigm for describing generation of domain code. However, none of the above approaches use UML to automatically generate assertions for the purpose of verifying the functional correctness of the synthesized code.

2.4 Case Study

In this section of the paper we introduce an example that is used to demonstrate the capability of our tools. This example is a simple traffic light controller. Although we present a simple example, the validation for this system is very important and encompasses important properties such as safety, liveness, and fairness. Figure 1 shows the traffic light environment. The highway traffic light is normally green and the farm road traffic light is red until a sensor from farm road traffic is detected indicating a vehicle is waiting for pass-through.

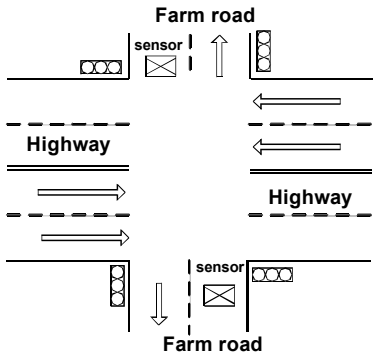

Figure 1. Traffic light Environment [7]

The system for the traffic light is shown in Figure 2. The whole system consists of 5 parts: *TrafficLightController*, *Timer*, *Roadsensor*, *HighwayTrafficLight*, and *FarmTrafficLight*. *TrafficLightController* is the central part of the system which accepts a signal from the *Timer* and the *Roadsensor* and controls the *Timer*, *HighwayTrafficLight* and *FarmTrafficLight*. The *Timer* is used to control the length of the traffic light in every state. The *Roadsensor* detects if a vehicle is coming from the farm road and sends the information to the *TrafficLightController*.

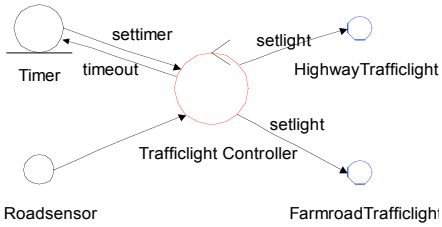

Figure 2. Traffic Light System

The detailed requirements for the traffic light system are listed as follows [7]:

1. Normally the highway light shall be green and the farm road shall be red.
2. When a sensor signals that a vehicle is on the farm road, the highway light shall change to yellow.
3. After a *Short-Time Interval* (STI, normally 10 seconds) the highway light shall change to red and the farm road light shall change to green
4. The farm road light shall stay green until the tractor clears the sensor or after a *Long-Time Interval* (LTI, normally 50 seconds), whichever comes first. Then the farm road light shall become yellow.
5. After an STI, the farm road light shall become red and the highway light shall become green.
6. The system shall stay in that state for at least the length of an LTI.
7. After an LTI the highway shall be switched when the sensor detects a tractor.

3.Prototype Design Tools

Figure 3 shows an overall design flow that includes our tools. The system description will be first described in UML where functional requirements are represented as UML models and non-functional requirements are represented using the UML real-time profile based on a UML quality of service profile or some other specified comments. UML diagrams are exported to XMI, a standard XML-based intermediate form. XMI uses predefined XML elements and attributes to specify the states, events, and actions that make up the state diagram. The initial impetus for XMI is to enable UML diagrams to be imported and exported across different UML toolsets. Our approach uses XMI as an intermediate input to the next stage of our processing. Two separate tools have been developed: ***u2SVA*** and ***u2HDL.***

In this section of the paper, we will describe more details of the synthesis tool ***u2HDL*****,** and the automatic assertion generation tool ***u2SVA***. We first introduce a high-level synthesis tool, ***u2HDL*****,** that transforms UML models into HDL (we only support the Verilog HDL currently). ***u2SVA*** is an automatic SVA generation tool that transforms a system specification represented by a UML diagram into SVAs. After the HDL code and SVAs are generated, both are sent to the SystemVerilog simulator/Assertion Checker to validate if the properties hold for the design. ***u2SVA*** and ***u2HDL*** are totally independent tools in that they are used with third party designs or SVAs separately.

For the traffic light controller example described previously, the system specification can be described in UML as shown in Figure 4.

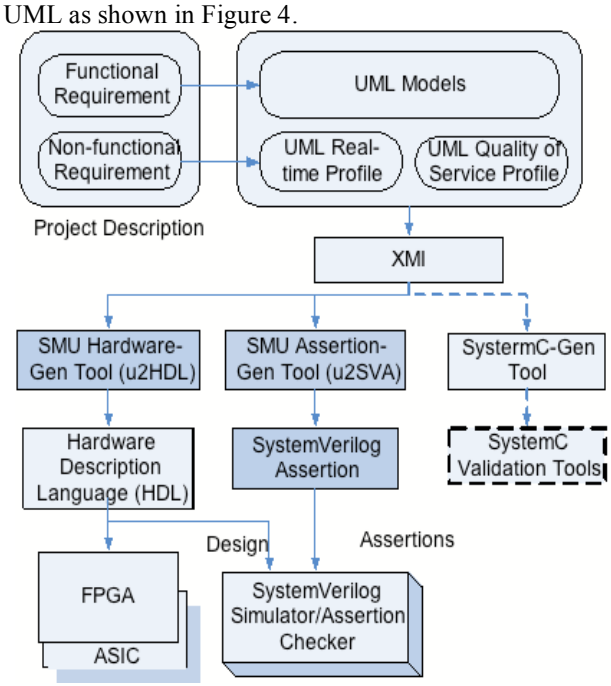

Figure 3. Overall System Design Tool Flow

3.1 *u2HDL*: A Synthesis Tool from UML to HDL

u2HDL is a high-level synthesis tool that translates UML models into HDL descriptions. Currently, we only support the Verilog HDL.

The translation procedure is accomplished in two steps as shown in Figure 5. During the translation, a tool referred to as XLST is used because we use XMI as an output format for UML models. XSLT [8] is used to transform a XML document into another XML document, or another type of document that is recognized by a browser, such as HTML or XHTML or some other text editor. The first XSLT translates XMI to State Machine eXtensible Markup Language (SCXML) [9]. SCXML was developed by the Voice Browser Working Group as part of the W3C Voice Browser Activity. SCXML provides a generic state- machine based execution environment based on CCXML and Harel State Tables. Harel State Tables are a type of state machine notation that is part of UML.

Different commercial UML tools generate different forms of XMI. Using SCXML as a middle layer can avoid writing different tools for each commercial UML tool. We focus on the second part of XSLT which translates SCXML to SVA. In this translation task, the extraction of various property assertions is a challenge.

The second part of XSLT shown in Figure 5 translates SCXML into HDL (Verilog). For a State Machine diagram, the "case" template can be applied for state transitions.

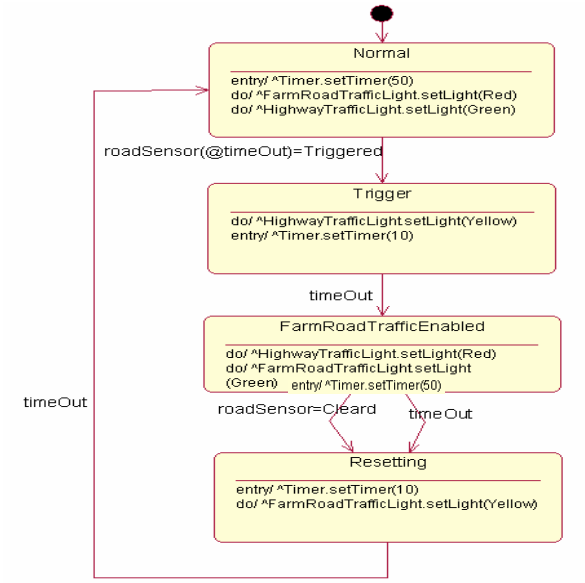

Figure 4. UML State Machine Diagram for Traffic Light Controller

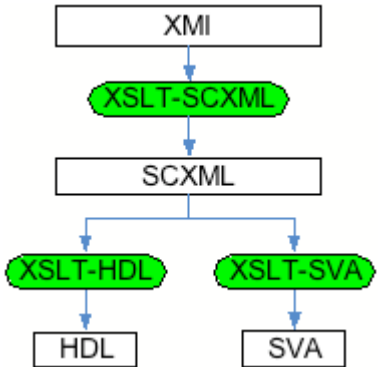

Figure 5. Procedures of u2SVA and u2HDL

There are two major parts in the case statement, one part specifies the state transition information in given state while the other part specifies the action in the given state. The transition information can be obtained from the edges in the UML state diagram. An example of a transition definition in UML is shown in Figure 6. We can easily extract the information of the state transition required for HDL, shown as the part 2 in Figure 7.

The second portion is the part specifies the action in the given state. UML allows one to define different types of action. Currently supported actions are "on entry", "do", "on exit" and "on event". For each action, you can also specify "Send arguments" and "Send Target" as shown in Figure 8.

Based on the example in Figure 8, we can extract information shown as Part 1 in Figure 7.

```
…
<UML:Transition xmi.id = 'G.15' name =
'' visibility = 'public' isSpecification
= 'false' >
   <UML:Transition.trigger>
     <UML:Event xmi.idref =
'S.220.0226.38.9'/>
   </UML:Transition.trigger>
   <UML:Transition.source>
     <UML:StateVertex xmi.idref='G.14'/>
   </UML:Transition.source>
   <UML:Transition.target>
     <UML:StateVertex xmi.idref='G.1'/>
   </UML:Transition.target>
</UML:Transition>
```

Figure 6. Transition Definition in UML

```
…
case (pstate)
 `Normal:begin
    Timer_setTimer=50;                    Part 1
    FarmRoadTrafficLight_setLight=`Red;
    HighwayTrafficLight_setLight=`Green;

    if(roadSensor==`Triggered&&(timeOut))
       nstate=`Trigger;
    else                                  Part 2
       nstate=`Normal;
    end
```

Figure 7. Resulting HDL code

To generate more efficient HDL, several more things that are addressed in the translation procedure are, the state encoding method for state variables; the BUS width for input, output, and registers, and, input ports and output ports. We explain each of these considerations in the following text.

The default state encoding style is minimum encoding where the number of encoding bits is calculated by the integer ceiling of $\log_2$(# *of* states available) . A user can specify an alternative encoding style by using UML comments with the special keyword "**CODINGSTYLE**". Currently, our prototype tool supports state encoding methods of one-hot and gray (minimum-distance).

The BUS width for inputs, outputs, and registers is specified by the system designer in order to generate an efficient hardware implementation. Non-specified ports are set to a 16-bit width as a default condition. The system designer can set an alternative bus width by specifying the maximum value (keyword "**MAXVALUE**") for a port or by emulating all possible values (keyword "**LIST**") of a port.

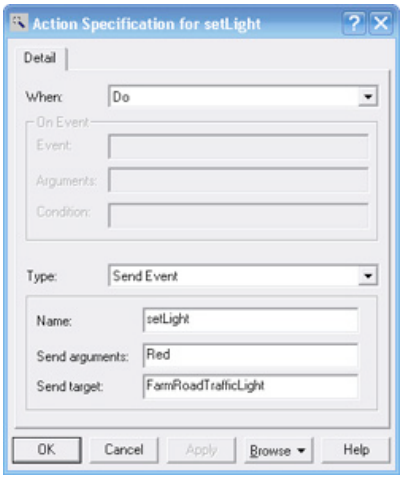

Figure 8. Supported action in UML

Regarding input ports and output ports, currently, we list all control lines in a state machine as input ports and all assignment lines in a state machine as an output port.

3.2 *u2SVA*: A Validation Tool from UML to SVA

u2SVA is a validation tool that translates a system specification represented as a UML model into SystemVerilog Assertions.

3.2.1 Automatic Assertion Extraction from State Machine diagram

We classify properties into two categories: state properties and transition properties. *A State property* defines the property that should hold within states while a *Transition property* defines the property that should hold when a state transition occurs. We use an example to show the functionality of ***u2SVA***. Figure 9 shows a portion of the state machine in Figure 4.

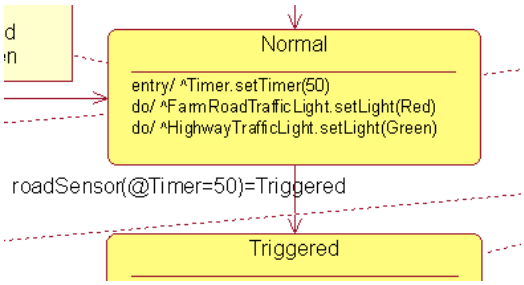

Figure 9. Example for state and transition properties

As described previously, state properties are those that should hold within each state. If the system is currently in the state *Normal*, two safety properties that should always hold are *HighwayTrafficLight* = *Green* and *FarmRoadTrafficLight* = *Red* which is categorized as a *do* action in the state *Normal*. The *entry* action is ignored for this state property since it can be validated with the *Timer*. The corresponding property is

```
…
property state_Normal;
 @(posedge clk)
   pstate==`Normal |->
FarmRoadTrafficLight==`Red &&
HighwayTrafficLight==`Green ;
…
```

Transition properties are properties that should hold for state transitions. If the system is currently in state *Normal*, then if *roadSensor* is triggered at *Timer*=50, the state goes to state *Triggered*, otherwise, it stays in the *Normal* state.

```
…
property transition_Normal;
 @(posedge clk)
   pstate ==`Normal && roadSensor==
Triggered && Timer==50 |-> pstate
==`Triggered;
```

3.2.2 Additional Assertion Specification

For some properties that cannot be extracted from a state machine diagram, we specify comments that start with the keyword **`ASSERTION`** to address this instance. There are usually three kinds of properties that are important to check, *liveness*, *safety,* and *fairness* [5]. For the traffic light example, the liveness property could be: if *Roadsensor* is triggered when *timeout*, the *FarmRoadTrafficlight* will eventually be green. A safety property could be that the *FarmRoadTrafficlight* and *HighwayTrafficLight* cannot be green at the same time. A fairness property could be that eventually the *FarmRoadTraffic* light will become green. Here we define some keywords that we use to define various properties in a UML description as shown in Table 1.

Table 1. Keywords for SVA Assertions

Keywords	Explanation
Always	The property should always hold
Exist	The property should hold at least one time
After …	The property should hold after some time units
Next	The property should hold in next time unit
Not	The inverse of property should hold
If **A** Then **B**	If condition A hold, Property B should hold
And	Two properties should both hold
Or	At least one property should hold
==	Equal

We use properties of the traffic light controller example to show how the **`ASSERTION`** comment works.

Liveness property: if the road sensor is triggered during timeout, the farm road traffic light will eventually be green. (in this example, eventually means 10 seconds).

The Assertion Comment is:

```
ASSERTION: Always (If(roadsensor == triggerd and timeout==1) Then (After(10), FarmRoadTrafficLight == green))
```

Given above description, the generated SVA are show in the following:

```
...
property liveness;
 @(posedge clk)
   (roadsensor == triggerd &&
timeout==1) |=> ##[0:$]
FarmRoadTrafficLight == green;
```

Safety property: farm road traffic light and highway traffic light cannot be green at the same time.

The Assertion Comment is

```
ASSERTION: Always ( Not ( FarmRoadTrafficLight == green and HighwayTrafficLight == green))
```

Given above description, the generated SVA is shown in the following:

```
...
property safety;
 @(posedge clk)
   not((FarmRoadTrafficLight ==
green) && (HighwayTrafficLight ==
green)
```

The specified properties may be used in a later synthesis stage by tools that perform formal model checking [14] or by automatic testbench generation tools tools to generate test vector files.

4. Conclusion

In this paper, we have described a high-level synthesis and validation tool that translates UML descriptions of embedded systems into Hardware Description Language models with associated SVAs. SVAs are used to check if the desired properties of an embedded system hold in the synthesized design. These tools are independent in that they can be used for third party design and SVAs separately. We believe this is a very convenient approach for embedded system designers as it allows for system property assertions (for the purpose of design validation) to be written at the time of specification.

In the work presented here, we have focused on the category of the "behavior" diagrams among the 12 specified diagrams in the UML standard with specific emphasis on the state machine diagram. In future work we plan to include the "structure" diagram category. We have most recently extended our tool to support datapath portions of hardware subsystems [16] and we plan to elaborate on this aspect in a future publication. Furthermore, we have utilized textual annotations in the comment field of the UML diagram to specify information used to form the SVAs. We believe the emergence of the SysML specification allows for formalisms within the specified diagrams that may be directly used to generate the SVAs and we intend to investigate utilizing SysML [17] instead of UML as a specification medium for our tool in the future.

Acknowledgement

We wish to thank the Synopsys Corporation for their generous donation of design tools and training.

References

[1] F.P. Coyle and M.A. Thornton, From UML to HDL: a Model Driven Architectural Approach to Hardware-Software Co-Design, in Proceedings of *Information Systems: New Generations Conference* (ISNG), April 4-6, 2005, pp. 88-93.

[2] G. Booch, I. Jacobson, and J. Rumbaugh, **Unified Modeling Language User Guide**, Addison-Wesley, 1998.

[3] IBM Rational Software, http://www-306.ibm.com/software/rational.

[4] I-Logix's Rhapsody Software, http://www.ilogix.com.

[5] J. L. Peterson, **Petri Net Theory and the Modeling of Systems**, Prentice Hall, 1981

[6] Tom Fitzpatrick**, Assertions in SystemVerilog: A Unified Language for More Efficient Verification,** Synopsys Inc., http://www.synopsys.com/products/simulation/assert_sverilog_wp.html

[7] Jorge Buenfil, "*Project Estimation based on Requirements Analysis with UML/SysML*," Presentation given at INCOSE Heartland Chapter, Jun. 2005, http://www.incose.org/heartld/.

[8] Khum Yee Fung, **XSLT: Working with XML and HTML**, Addison Wesley, 2001.

[9] RJ Auburn, Jim Barnett, Michael Bodell, and T.V. Raman, "*State Chart XML (SCXML): State Machine Notation for Control Abstraction 1.0*," W3C Working Draft 5, July 2005 http://www.w3.org/TR/2005/WD-scxml-20050705/.

[10] W.E. McUmber and B.H.C. Cheng, UML-based analysis of embedded systems using a mapping to VHDL, in Proceedings of *lEEE Int. Symposium on High Assurance Software Engineering* (HASE'99). November 1999, Washington, DC, USA, pp. 56-63.

[11] D. Bjarklund and J. Lilius. From UML behavioral descriptions to efficient synthesirable VHDL, in Proceedings of *20th IEEE NORCHIP Conference*, 11-12 November 2002, Copenhagen, Denmark.

[12] V. Sinha. D. Doucet, C. Siska, and R. Gupla, YAML: a tool for hardware design visualization and capture, in Proceedings of 13th *Int. Symposium on*

System Synthesis (ISSS), 20-22 September 2000, Spain, pp. 9-16

[13] R. Damasevicius and V. Stuikys, Application of UML for hardware design based on design process model, in Proceedings of the *Asian South Pacific Design Automation Conference* (ASP-DAC), 2004.

[14] Clarke, E.M., Grumberg, O., and Peled, D.A., **Model Checking**, The MIT Press, 1999, ISBN 0-262-03270-8.

[15] Object Management Group, OMG Unified Modeling Language Specification, Version 1.3, March 2000, http://www.omg.org/cgi-bin/doc?formal/00-03-01.

[16] K. Hawkins, *An Automated Tool for HDL and Configuration File Generation from UML Descriptions*, M.S.C.p.E. thesis, Department of Computer Science and Engineering, Southern Methodist University, June 2007.

[17] Object Management Group, OMG SysML Specification, Version 1.0, April 2007, http://www.sysml.org/docs/specs/OMGSysML-FAS-06-05-04.pdf.