From UML to HDL: a Model Driven Architectural Approach to Hardware-Software Co-Design

Frank P. Coyle and Mitchell A. Thornton
Computer Science and Engineering Dept
Southern Methodist University
Dallas TX 75275
{coyle, mitch}@engr.smu.edu

Abstract

The SMU Co-Design Project is an effort to target the problem of hardware/software co-design via an open source laboratory for studying hardware-software integration. The project focuses on the use of Model Driven Architectures (MDA) to define high-level model-based system descriptions that can be implemented in either hardware or software. Utilizing component and state diagrams based on the Unified Modeling Language (UML), we demonstrate MODCO, a transformation tool which takes a UML state diagram as input and generates HDL output suitable for use in Field Programmable Gate Array (FPGA) circuit design. With this tool as a first step, we plan to continue to bridge the gap between hardware and software design taking advantage of trends in both areas to work with higher level description languages and use software transformation tools to manage lower-level hardware or software implementation details. This project also serves as the basis for a new generation of software and computer engineers who understand each other's problems and issues and are able to leverage the capabilities of model-based system description languages.

Keywords: hardware/software co-design, Unified Modeling Language, UML, Model-Driven Architecture, MDA

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 once relegated to software now has the possibility of implementation in hardware while hardware components must integrate with higher-level software APIs. [1]

For embedded systems developers, the primary architectural/design issue has been the partitioning of system functionality across both hardware and software. Common practice involves creating separate specifications for hardware and software. These specifications, often written in non-formal languages, are delivered with functionality allocated *a priori*. Because changes to the partition may necessitate extensive redesign and because software rework is viewed as easier than hardware redesign, design solutions often have a heavy software component with hardware allocation specified to meet anticipated timing constraints.

However, such an approach has several drawbacks which include (a) lack of a unified hardware-software representation, leading to difficulties in verifying the entire system and to incompatibilities across the HW/SW boundary; (b) a *priori* definition of partitions, which leads to sub-optimal designs; (c) lack of a well-defined design flow, which makes specification revision difficult, impacting time-to-market.

1.1 Hardware/Software Co-Design

To help address these problems, the area of hardware/software co-design has developed as an architectural approach to system design where decisions concerning hardware-software allocation are deferred as late as possible within the design cycle. Approaches to hardware/software co-design include the development of co-synthesis algorithms [2], simulation [3], and dataflow techniques [4].

In this paper we present the use of Model Driven Architecture (MDA) [5] to address the challenges of hardware/software co-design. Specifically we outline details of MODCO, a tool for transforming UML state diagrams directly into HDL, the first step in the automated synthesis of an FPGA circuits.

The paper is organized as follows: section 2 provides an overview of the SMU Co-Design project and related technologies; section 3 describes the MODCO tool for generating HDL from high-level UML diagrams using XML; section 4 outlines the implications of this work and future directions, and section 5 provides a summary.

2 The SMU Co-Design Project

The SMU Co-Design Project is a collaborative effort between software engineering and computer engineering faculty to bring model-driven design to bear on the problem of hardware/software co-design. The Computer Science and Engineering Department at SMU includes the subspecialties of Software Engineering and Computer Engineering presenting a unique opportunity for collaboration. The objective is to bring together hardware and software practitioners to support the creation of fully integrated and optimized systems.

The project focuses on the construction of high-level, model-based system descriptions. Working from system requirements, high-level UML models are created which serve as the basis for either hardware or software implementations. Collectively, these models comprise a model driven architecture (MDA), an approach to system construction supported by the Object Management Group (OMG)[6] . Using models and notation drawn from the *defacto* UML standard, our model-driven approach captures system functionality and semantics at a high level of abstraction and then uses tools to generate different platform specific implementations, in this case either hardware or software.

2.1 Model Driven Architecture

Model Driven Architecture (MDA) is an approach to software development that focuses on the production of high-level models that are used as the basis for automating system implementation. As in all engineering disciplines there is a gap between system models and the actual system under construction. The goal of MDA is to reduce that gap by automating coding and implementation through the creation of knowledge-based tools.

The MDA approach centers on the definition of a Platform Independent Models (PIMs) using a high-level specification language. The goal is to develop models that are precise enough to support code generation so that a PIM may be transformed into one or more Platform Specific Models (PSMs) for the actual implementation. The advantage of the MDA approach is that models and code are more easily kept up to date and incremental, iterative development is facilitated by the direct transformation from model to code.[7]

2.2 Unified Modeling Language (UML)

While MDA is technically neutral about the syntax or structure of the high-level models, UML [8] 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 their hardware/software tradeoffs.

UML defines twelve types of diagrams, 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 Statechart 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 software or hardware. The recent addition of action semantics to UML has led to development of executable UML (xUML) which supports the direct execution of UML models. [9]

UML is supported by a wide range of tools such as IBM’s Rational Toolset [10] and I-Logix’s Rhapsody [11]. The exchange of models between tools is supported by the XMI standard [12], an XML-based description language which captures the details of UML model diagrams in a portable, machine readable format. Most UML tools and serves can automatically generate XMI which is used as the basis for our MODCO transformation from UML to HDL.

Also part of UML is an underlying action semantics model based on timed Petri Nets [13]. This provides the basis for model execution centered on concurrently executing action objects that take a set of inputs and transform them into a set of outputs. This concurrency model is a natural fit to the distributed execution environment of modern enterprise and embedded systems applications.

2.3 Hardware Description Language (HDL)

HDL is one of a class of computer languages used to provide formal description of electronic circuitry [14]. An HDL standard text-based expression is capable of describing the temporal behavior and/or (spatial) circuit structure of an electronic system. HDL is widely used in hardware design to specify details of chip design for either specialized chips or FPGAs. For custom or standard-cell based integrated circuit, such as a processor or other kind of specialized digital logic chip, HDL specifies a model for the expected behavior of a circuit before that circuit is designed and built. Special logic synthesis tools are then invoked that ultimately provide the geometric information used to produce photo-lithographic masks necessary for the fabrication of the device.

For programmable logic devices such as FPGAs, HDL code is first delivered to a logic compiler (FPGA synthesis tool), and the output is uploaded into the device. The unique property of this process, and of programmable logic in general, is that it is possible to alter the HDL code many times, compile it, and upload into the same device for testing.

From an MDA perspective, the decision to go from HDL to either an integrated circuit or FPGA may be viewed as a decision to go from a PIM description (HDL) to a PSM (ASIC or FPGA). Within the industry, the decision to go FPGA or specialized IC is a complex one and is based on factors that are traceable back to higher level architectural and market considerations.

2.4 SMU Co-Design Project Overview

The SMU Co-Design project is organized around a series of transformations that go from system requirements to hardware/software implementation. The transformational flow is illustrated in Figure 1. From a process perspective, there are two major phases in going from requirements to implementation.

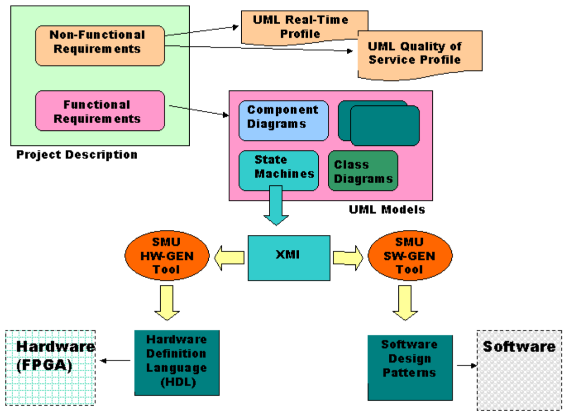

Fig 1. UML models are the first stage in the transformation to either hardware or software implementation.

Phase I consists of building a sufficiently detailed system model to support the hardware vs. software tradeoff decision. To accomplish this, UML system models are generated from both functional and non-functional system requirements. Functional requirements, which describe basic system functionality, are mapped to UML component, class, use case, and state diagrams. Non-functional requirements, which describe system quality attributes such as performance and timing requirements, are mapped to UML annotations that describe performance constraints using property-value pairs defined by UML profiles – auxiliary domain-specific semantic information that can be added to UML models.

Phase II uses the UML models from phase I as the basis for the automated generation of hardware or software components. Our initial effort, as described in this paper, is the generation of HDL directly from the UML using our MODCO tool (described in detail in section 4). The generated HDL provides a standard text-based representation of the temporal behavior and (spatial) circuit structure of an electronic system suitable for FPGA construction. HDL provides explicit notation for expressing time and concurrency based on data coming from the UML

3. From UML to HDL

The transformation from high-level UML state diagram to HDL is based on a multi-step process. Figure 2 illustrates the process which consists of the following steps.

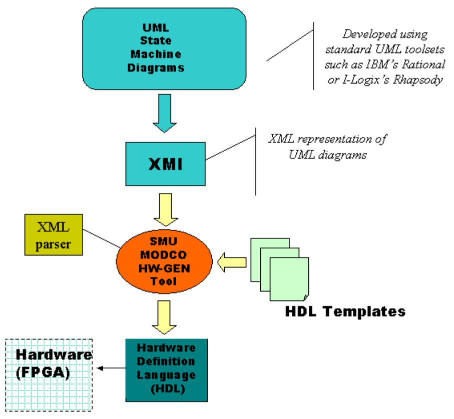
UML State Machine Diagrams
Developed using standard UML toolsets such as IBM's Rational or I-Logix's Rhapsody
XMI
XML representation of UML diagrams
XML parser
SMU MODCO HW-GEN Tool
HDL Templates
Hardware (FPGA)
Hardware Definition Language (HDL)

Fig. 2. The transformation from UML to HDL.

Step 1. State diagrams are created using UML state diagram notation. These diagrams describe system behavior using states, events and actions and correspond closely with the high-level design approach taken by circuit designers.

Step 2. 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 was to enable UML diagrams to be imported and exported across different UML tools. Our approach uses the XMI as input to the next stage of processing.

Step 3. The XML representation of state machines is parsed by a Java-based XML parsing utility developed as part of the MODCO tool.

Step 4. Data extracted from the XMI by the Java parser is mapped to HDL templates, resulting in HDL suitable for use in FPGA construction.

3.1 Transformation Example

The following example illustrates the conversion of a simple UML state machine. We begin with a simple state machine as illustrated in Figure 3.

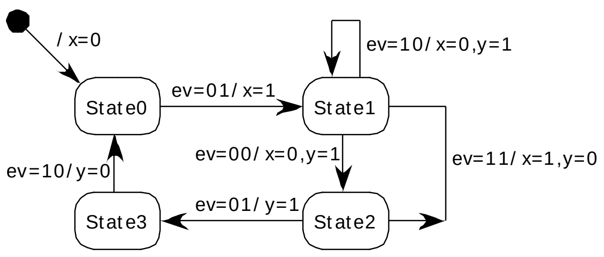

Figure 3. A simple UML state machine

In the above state machine, events ev1 thru ev5 drive the transitions from state to state. Transitions are of the form "event / action" so that ev=01/x=1 is interpreted as on the occurrence of event 01 the action, x=1 is taken. Listing 1 shows the HDL output from the state diagram transformation.

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm is
        port (        signal EV:  in std_logic_vector(1 downto 0);
                      signal INPT, CLK:  in std_logic;
                      signal X, Y: out std_logic );
end fsm;

architecture BEHAVIORAL of FSM is
        type STATES is (STATE0, STATE1, STATE2, STATE3);
        signal PRES_STATE, NEXT_STATE : STATES;
        signal X, Y, CLK, RESET : std_logic;
        signal EV: std_logic_vector(1 downto 0);
        begin        --Synchronous Portion with Asynchronous reset
        CLOCKED_PROC : process (CLK, RESET)
           begin
             if (RESET = '1') then
               PRES_STATE <= STATE0;
             elsif (CLK'event and CLK = '1') then
               PRES_STATE <= NEXT_STATE;
             end if;
          end process CLOCKED_PROC;

        COMB_PROC : process (INPUT, PRES_STATE)
          begin
case PRES_STATE is
 when STATE0 =>
  X <= '0';
  if (EV = "01") then
  begin
    X <= '1';
    NEXT_STATE <= STATE1;
   end
  else
   NEXT_STATE <= STATE0;
   when STATE1 =>
    if (EV = "00") then
     begin
      X <= '0';  Y <= '1';
       EXT_STATE <= STATE2;
      end
    elsif(EV = "10")
     begin
      X <= '0';  Y <= '1';
      NEXT_STATE = STATE;
     end
    elsif(EV = "11)
      begin
       X <= '1';  Y <= '0';
       NEXT_STATE <= STATE2;
      end
    else
      NEXT_STATE <= STATE1;
   end if;

  when STATE2 =>
    if (EV = "01") then
    begin
     Y <= '1';
      NEXT_STATE <= STATE3;
     end
    else
      NEXT_STATE <= STATE2;
     when STATE3 =>
```

```
          if (EV = “10”) then
          begin
            Y <= ‘0’;
             NEXT_STATE <= STATE0;
           end
          else
               NEXT_STATE <= STATE3;
end case;
end process COMB_PROC;
end BEHAVIORAL;
```

Listing 1. UML to HDL transform output

4 Implications and Future Directions

The ability to move from UML diagrams to HDL hardware descriptors is the first step in an effort to use model-based architecture to further optimize and automate the embedded systems development. For example, having made the decision to implement a function using FPGAs, there are numerous decisions concerning the positioning of components that will impact product viability. One aspect of all circuit-level designs that impacts the final deliverable is power consumption. While the capability to temporarily shut down chip subsystems for power savings is becoming critical for many handheld devices, most hardware design reflects previous generation thinking where energy management is a mere afterthought. However, as software moves more and more into the embedded application space, power management plays an increasingly important role in chip design and deployment.

Currently, we are exploring ways to use MDA to help configure subsystems for power savings by associating power regions with system functionality so that subsystem activation is correlated with scenarios of usage as described in UML Use Cases. By analyzing the whole system in terms of supporting software and hardware, it is possible exploit new opportunities for power savings that may not be apparent to a hardware design team focused on a single hardware block.

5 Summary

In this paper we describe a model driven architectural (MDA) approach to hardware-software co-design through the generation of HDL from high-level UML diagrams. In Phase I, UML system models are generated from both functional and non-functional system requirements leading to hardware-software partitioning. In phase II, HDL descriptions are generated from UML diagrams using XMI, an XML-based description language initially designed to allow UML designs to migrate between different vendor tools. In our approach, XMI model descriptions are parsed by XML parsers, which fill in details for predefined HDL templates. We believe that this approach bridges the gap between hardware and software design with broad applicability for embedded systems development.

Beyond the technical aspects of this approach, we believe that the collaborative work between software engineering and computer engineering helps create a community and infrastructure that will permit students and faculty from two different subspecialties to work more closely together in pursuit of common computing objectives and agendas.

References

[1] A. A. Jerraya and W. Wolf, "Hardware/Software Interface Codesign for Embedded Systems," *Computer*, vol. 38, 2005.

[2] Y. Li and W. W. Wolf, "Hardware/Software Co-Synthesis with Memory Hierarchies," in *Readings in Hardware/Software Co-Design*, G. D. Micheli, R. Ernst, and W. Wolf, Eds. San Francisco: Morgan Kaufmann, 2002, pp. 265-277.

[3] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, " Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation, special issue on "Simulation Software Development,"* vol. 4, pp. 155-182.

[4] S. S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee, "Generating Compact Code From Dataflow Specifications Of Multirate Signal Processing Algorithms," *IEEE Transactions on Circuits and Systems*, vol. 42, pp.? 1995.

[5] D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*: OMG Press, 2003.

[6] "http://www.omg.org/mda."

[7] J. Warmer and A. Kleppe, *The Object Constraint Language Second Edition: Getting Your Models Ready for MDA.*: Addison-Wesley, 2003.

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

[9] S. J. Mellor and M. J. Balcer, *Executable UML. A Foundation for Model-Driven Architecture*. Indianapolis: Addison-Wesley, 2002.

[10] "http://www-306.ibm.com/software/rational/."
[11] "http://www.ilogix.com/rhapsody/rhapsody.cfm."
[12] "http://www.omg.org/technology/xml/."
[13] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs: Prentice Hall, 1981.
[14] D. J. Smith, *HDL Chip Design. A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog*. Madison, AL.: Doone Publications, 1996.