Hardware Modeling in VHDL for Mixed Hardware/Software System Level Co-simulation of Embedded Systems

B. Tabbara*

EECS Department
U.C. Berkeley
Berkeley, CA 94720
U.S.A

E. Filippi

CSELT †
V. Reiss Romoli 274,
I-10148 Torino
Italy

L. Lavagno

Cadence Berkeley Labs
2001 Addison St., 3rd Floor
Berkeley, CA 94704
U.S.A

A. Sangiovanni-Vincentelli

EECS Department
U.C. Berkeley
Berkeley, CA 94720
U.S.A

Abstract

We present a method for modeling and then simulating a mixed hardware/software embedded system implementation in VHDL starting from a high level design representation. In our approach, a complete system description including the communication interfaces is generated automatically in VHDL: Software is modeled by using behavioral VHDL constructs, annotated with timing information derived from basic block-level timing estimates, while hardware can be either pre-existing Intellectual Property (IP) or synthesized from a functional specification. Our approach has been incorporated into a comprehensive co-design environment, and while we describe elsewhere in detail the software synthesis and modeling in VHDL[9, 8], we focus here on the Finite State Machine with Datapath (FSMD) Register Transfer Level (RTL) modeling of hardware, and the subsequent system validation in VHDL.

1 Introduction

Embedded systems consist of a mix of hardware and software components. Hardware is usually needed for performance while software is used for flexibility. It is often quite desirable to be able to specify the design functionality and constraints at a high level, and then synthesize the hardware, software, and the necessary interfaces. Verifying the sometimes complex interaction between this mix of components is then the major task that follows synthesis.

Typical hardware/software co-simulation methods have involved either running the software on a hardware model of the processor [5] which is very slow because of the RTL models involved, or piecing together hardware and software simulators and filtering the data being sent back and forth between the two partitions [3][10].

The latter method requires extensive manual intervention to "abstract" the interface, by hiding events such as instruction fetches or some memory accesses from the hardware simulation. The co-simulation methodology that we developed is aimed exactly at filling this "validation gap" between fast models without enough information (e.g., instructions without timing or bus cycles without instructions) and slow models with full detail. It assumes that *execution time estimates* are available for each basic block of software [4]. In particular, such estimates are easily available if one uses a software synthesis-based approach to co-design, as described in [1], or can be obtained from approaches such as [4].

In this paper we present a methodology for synthesizing and modeling in mixed-level VHDL hardware and software, derived by partitioning a single implementation-independent specification. Our approach has been incorporated into a co-design framework that consists of a comprehensive tool set including partitioning, scheduling, estimation, and constraint handling. The scope of this paper, however, will be limited to synthesis and simulation in VHDL of the hardware tasks starting from the high-level design description. We refer the reader to [9] for an in-depth description of software task and scheduler modeling in behavioral VHDL.

VHDL is a standard language and many well supported tools for efficient synthesis and simulation are commercially available. This makes integration of automatically synthesized modules with external blocks (perhaps designed with different methodologies) quite straightforward [9]. We leverage on this feature in our method and as a result get a lot of added value in terms of:

- design re-use,

- incorporation of Intellectual Property (IP) Libraries, and

- integration of our new co-design techniques into an established industrial design flow.

The paper is organized as follows. In Section 2 we provide some brief background and introduction. In Section 3 we outline the mixed hardware/software co-simulation technique, and then focus primarily on the hardware modeling approach for simulation and synthesis. In Section 4 we show simulation results for an industrial-size application modeled using this approach. Finally, in Section 5 we discuss the status of this work and outline directions for future research.

2 Overview of the Co-design Environment

We have incorporated our approach into the POLIS co-design environment for reactive embedded systems ([1]), since it is open, available to the public, and provides synthesis and estimation capabilities for both hardware and software. In this Section, we briefly describe the hardware and software synthesis strategies in POLIS as they relate to the modeling for co-simulation.

2.1 Design Representation: CFSMs

POLIS is centered around a design representation known as a network of Co-design Finite State Machines (CFSMs). Each element of a network of CFSMs describes a component of the system to be modeled, and defines the partitioning and scheduling granularity. The CFSM model is based on Extended Finite State Machines (EFSMs) that operate according to a *Globally Asynchronous Locally Synchronous (GALS)* communication model by using *events*.

Figure 1 shows a simple CFSM, implementing a seat belt controller that turns on the alarm if the driver does not fasten the seat belt 5 seconds after turning on the ignition key, and turns off the alarm after 10 seconds, or when the seat belt is fastened.

2.2 Hardware Synthesis

Hardware CFSM components of a design in POLIS are implemented as Finite State Machines (FSMs) that consist of a combinational part for the next state logic of the transition relation and the data path, and latches (all driven by the same clock) for the outputs and states. The result is a logic netlist (generated using classical logic synthesis techniques) that can be mapped to a specific technology.

2.3 Software Synthesis and Estimation

Software synthesis and performance estimation in POLIS is based on a simplified Control-Data Flow Graph (CDFG) called S-GRAPH. An S-GRAPH is a Directed
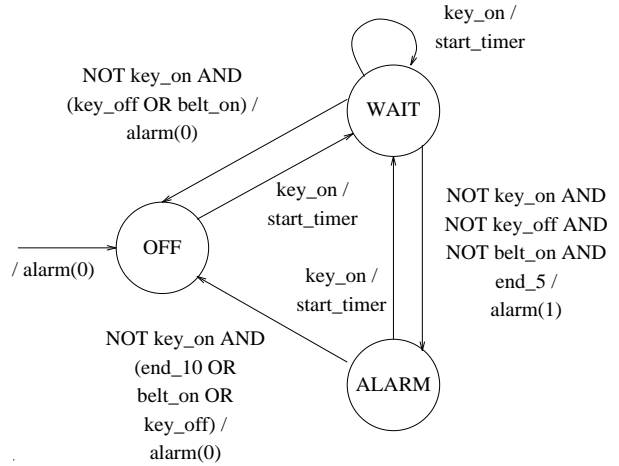

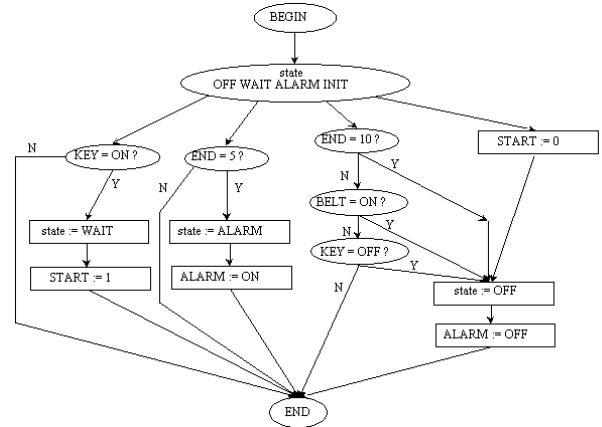Figure 1: The CFSM of the Seat Belt Alarm Controller

Figure 2: The S-GRAPH of a Seat Belt Alarm Controller

Acyclic Graph where each path implements a transition of the CFSM. Figure 2 shows the S-GRAPH for the seat belt controller CFSM of Figure 1.

The S-GRAPH CDFG has a 1-to-1 mapping to sequential code. C or VHDL code can be generated at the same granularity level [8]. Estimation (execution time, and code/data size) is also performed for each node type (**BEGIN**, **TEST**, **ASSIGN**, **END**) depending on the complexity of the node function (tested or assigned to an output variable).

3 High-level Co-simulation Using VHDL

Our approach to co-simulation is based on the decomposition of the system into three classes of components:

1. Software CFSMs, synthesized by POLIS and executed on a single processor under the control of a

Real-Time Operating System (RTOS). The RTOS, also synthesized by POLIS, handles communication within the processor and with the rest of the system,

2. hardware CFSMs, also synthesized by POLIS and communicating via a standardized protocol with the rest of the system,

3. existing pieces of hardware IP, modeled in VHDL (behavioral or RTL).

We synthesize a VHDL model for each CFSM, for the RTOS scheduler, and for the interfaces (see [9]).

3.1 Modeling Hardware Components

The technique used for modeling hardware components of the embedded system depends on whether they have been designed by using classical techniques (e.g. directly in synthesizable VHDL), or have been synthesized from CFSMs:

- In the former case, the designer must make sure that the incorporated modules use the CFSM communication protocol (see [8]). Techniques such as those described in [6] can be used to automate this adaptation task.

- In the latter case, we can generate a *synthesizable* Finite State Machine with Datapath (FSMD) Register Transfer Level (RTL) VHDL model that can be simulated by any VHDL simulator.

We now outline two different strategies for generating the synthesizable FSMD, one that requires simpler interface models (since the buffering of events is directly implemented by the VHDL simulation engine), and one that is fully synchronous, and hence can be synthesized and simulated in a cycle-based fashion.

3.1.1 Asynchronous Mealy Implementation

The basic idea of our hardware modeling approach is as follows: the S-GRAPH is interpreted as an asynchronous FSM with datapath, with one "state" for each S-GRAPH node. The execution of the S-GRAPH from **BEGIN** to **END** then becomes an ordered traversal of a sequence of labels in the FSM combinational part. In some sense, this FSM is a *sequential implementation* of the transition function of the CFSM, with several "micro states" (one for each S-GRAPH node), reached during a BEGIN-END traversal, that implement an abstract "macro state" transition. The structure of the VHDL code for an *Asynchronous Mealy* implementation for the belt controller hardware task described in Section 2 is shown in Figure 3.

The operation of the CFSM modeled as a *VHDL process* and shown in Figure 3 for the seat belt controller example is as follows. A "macro"-transition of the CFSM occurs only when the VHDL process detects a rising edge on the **clk** input and executes its *sequential part* (labeled in the Figure) thus updating its internal state, and its outputs. On the other hand, the process executes the *combinational part* that consists of the "micro"-transition loop representing the proper S-GRAPH path (labeled in the Figure) and therefore evaluates its next state logic *whenever it senses a change in any of its input events*. These inputs may change more than once within a clock cycle, in which case the CFSM re-evaluates its next state logic, that's why this is an *Asynchronous Mealy* implementation. There are no races or hazards in this asynchronous FSM, because of the single path execution of the S-GRAPH.

The generated code has several features that should be pointed out:

- Each CFSM ("macro") transition takes *one* clock cycle,

- Each path in the S-GRAPH model (executing a single macro transition) is modeled in VHDL as a **loop**, that updates the next state logic, sensing inputs "immediately" from other components. This **loop** terminates since S-GRAPH is a DAG,

- An explicit **clk** signal (added to the *process sensitivity list*) represents the global hardware clock which synchronizes the CFSM move from one macro state to another and also the update of outputs at the rising edge of this signal.

- Interfaces consist simply of wires from each signal that connect to all the tasks sensitive to this event,

- Inputs are sampled *whenever* they change instead of once per clock cycle, and the outputs are reset at the beginning of each clock cycle.

It should be noted that the asynchronous modeling simplifies the modeling of the interfaces, at the expense of more events in the VHDL simulator. The model uses the *VHDL scheduler* (and the process *sensitivity list*) to effectively model infinite-sized queues so that there is no need to explicitly model these buffers.

3.1.2 Synchronous Mealy Implementation

As an alternative to the previous model, that conceivably should be used in later phases of the design when inter-CFSM queues have been sized and a synthesizable
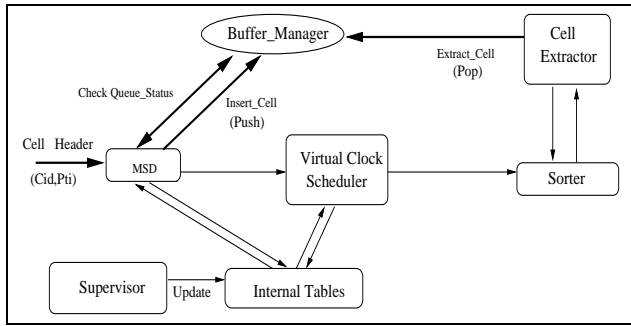
Figure 4: A High Level Description of the ATM Server Control Unit

RTL (% of Design)	Clock Cycles per CPU Sec.
100%	7,000
50%	15,000

Table 1: VHDL RTL Co-simulation Results for the Synthesized ATM Server Control Unit

model is required, our tool can also generate a *Synchronous Mealy* model of each hardware CFSM.

The most significant differences are that now only the `clk` signal triggers the processes, and that explicit interface modules are required. In the synchronous model the CFSM samples its input and evaluates its next state logic *once per clock cycle* as opposed to the *once per trigger event* in the previous model. Internal state and outputs are updated at the beginning of the transition.

4 A Practical Case Study: An ATM Server

The co-simulation technique described in this paper has been used to validate the design of an industrial case study from the communication networks domain: an ATM server suitable for implementing Virtual Private Networks (VPN) in ATM nodes as described in [7].

The system is composed of two parts: a fast hardware data path, and a control unit. The fast data path includes two standard UTOPIA interfaces, an ATM cell address lookup unit, a buffer logic queue manager, and a large buffer memory. It is implemented with a set of VHDL synthesizable models [2] and some commercial memories. The control unit has been designed using POLIS, and implements the server core custom functionalities. Figure 4 shows a high-level description of the control unit functional blocks.

VHDL co-simulation has been used to validate the whole system (including both the data path and the control unit). The ATM server design is composed of about 14000 VHDL code lines, of which about 7000 lines are from RT-level IP modules, about 6700 lines have been synthesized from CFSMs, and the rest are handwritten code. In other words, one half of the design comes from reusable RT-level Intellectual Property.

The co-simulation results for the RTL modeling are shown below (data collected from a commercial VHDL simulator on a Sun Ultra 2 workstation with 256MB of memory and 2 CPUs). These results are relative only to the part of the design that has been *fully* synthesized using POLIS. The first column of Table 1 displays what percentage of the design is implemented in hardware in RTL (the remainder is implemented in software).

5 Conclusions and Future Work

In this paper we have presented a mechanism for co-simulating synthesized hardware and software together with existing hardware Intellectual Property in a single VHDL-based environment. This technique uses software timing estimation to efficiently synchronize the VHDL processes modeling software tasks with those modeling hardware components and the test-bench. We have focused in this paper on presenting the hardware modeling technique for co-simulation with the software.

We chose the S-GRAPH software synthesis strategy to model synthesized hardware for several reasons which include:

1. Abstraction: of hardware, which allows for faster simulation than otherwise possible from a structural VHDL description obtained after logic synthesis.

2. Compactness: of the DAG based "unrolling" of the CFSM functionality as opposed to traditional truth table based approaches.

3. Re-use: of the modeling algorithm for both hardware and software.

In the future, we plan to explore VHDL synthesis starting from the CFSM abstraction directly instead of the S-GRAPH level. We will evaluate both simulation (VHDL code size and simulation speed) and implementation (silicon area and clock speed) for all the different alternatives mentioned above (the synchronous and asynchronous approach, as well as the current hardware synthesis strategy in POLIS).

References

[1] Felice Balarin, Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Ellen Sentovich, Kei Suzuki, and Bassam Tabbara. "Hardware-Software

Co-Design of Embedded Systems: The POLIS Approach", *Kluwer Academic Publishers*, May 1997. `http://www-cad.eecs.berkeley.edu/~polis`

[2] E. Filippi, L. Licciardi, A. Montanaro, M. Paolini, M. Turolla, M. Taliercio "The Virtual Chip Set: A Parametric IP Library for System on a Chip Design" *CICC*, Santa Clara, May 1998.

[3] R. Klein and S. Leef "New Technology Links Hardware and Software Simulators" *In Electronic Engineering Times*, June 1996.

[4] Y. Li and S. Malik "Performance Analysis of Embedded Software Using Implicit Path Enumeration" *In Proceedings of the Design Automation Conference*, June 1995.

[5] J. Rowson "Hardware/Software Co-simulation" *In Proceedings of the Design Automation Conference*, pp. 439-440, June 1994.

[6] A. Seawright, U. Holtmann, W. Meyer, B. Pangrle, et. al. "A System for Compiling and Debugging Structured Data Processing Controllers" *EURO-DAC* p. 86-91, Sept. 1996.

[7] E. Filippi, L. Lavagno, L. Licciardi, A. Montanaro, M. Paolini, R. Passerone, M. Sgroi, A. Sangiovanni-Vincentelli "Intellectual Property Re-use in Embedded System Co-design: an Industrial Case Study" *ISSS*, Dec. '98.

[8] —, "Validation of Reactive Real-Time Systems Using VHDL" *Masters Thesis*, University of California at Berkeley, May 1998.

[9] —, "Fast Hardware-Software Co-simulation Using VHDL Models" *Design, Automation and Test in Europe*, March 1999.

[10] V. Zivojnovic and H. Meyr "Compiled HW/SW Co-simulation" *In Proceedings of the Design Automation Conference*, June 1996.

```vhdl
Architecture CFSM_rtl of belt is
    signal e_timer_e_end_5_to_belt_control_0 : bit; −−internal (receiver)
    signal e_key_on_to_z_belt_control : bit; −−input
    signal e_alarm : bit; −−output
    ...
belt_control:
process(clk, e_key_on_to_belt_control, ...)
    type L_Type is (LB,L1,L2,...,LE);
    −−state and local variable declarations
    variable v_alarm_tmp : integer := 0;
begin
−−sequential part
if ( clk'event and (clk = '1') ) then
    −− update state variables
    v_alarm_tmp := v_alarm;
    −− reset output events
    if (e_alarm /= '0') then
        e_alarm <= '0';
    end if;
end if;


−−combinational part
loop −− one CFSM transition

Lbl := Next_Lbl;

case Lbl is
when LB =>
    −− sample input events
    e_key_on_tmp := e_key_on_to_belt_control;
...
when L2 =>
    if (e_key_on_tmp = '1') then
        Next_Lbl := L17;
    else
        Next_Lbl := L3;
    end if;
...
```

```vhdl
    ...
    when LE =>
        Next_Lbl := LB;
        exit; −−end of CFSM transition
    end case;


    end loop;


    end process belt_control;
```

Figure 3: The FSMD VHDL Code (Asynchronous Mealy Implementation) for the Hardware Seat Belt Controller