System Modeling, Hardware-Software Codeign, and Mixed Modeling with Hardware Description Languages

Sidhartha Mohanty
AT&T Bell Laboratories
Allentown, PA

Philip A. Wilsey
Department of ECECS, PO Box 210030
Cincinnati, OH 45221-0030

Abstract

Hardware Description Languages (HDL) such as VHDL provide a powerful platform for modeling at different levels of abstraction. Hardware-software co-design provides the designer with a design environment in which the designer can make effective partitioning decisions, and concurrently consider developing hardware and software components of a system, selected for the best possible performance, within a set of performance and design constraints. This paper describes our work to use VHDL to build design libraries that support rapid prototyping and conceptual modeling as software components in a hardware-software co-design environment. These library components can be instantiated for rapid prototyping, simulation, performance analysis of the design in the software domain, and for hybrid modeling containing interpreted and uninterpreted components. This helps in creating an uniformly robust design environment using a single HDL that supports the entire design cycle.

1 Introduction

The increasing complexity of the design cycle has brought the design process to a different level of abstraction. Entire systems including both hardware and software components are being installed on a single piece of silicon. This is in the so called System on Silicon (SOS). In this approach, the hardware and software components are defined from top-down, and the design process involves interactive evolution and interaction of these components. The hardware components of SOS include the use of macrocells, ASIC cores, standard cell design, and full/structured custom designs [8]. The challenge is to define robust, reusable software components which transcend the hardware and software boundaries. Hardware Description Languages (HDLs) such as VHDL [17, 18, 9] provide a very powerful platform for the modeling of digital systems at different levels of abstraction (switch, gate, RTI, behavioral). They provide a powerful platform for system modeling and mixed modeling containing hybrid models as both interpreted and uninterpreted components, thus creating an uniformly robust design environment using a single HDL supporting the entire design cycle.

The Electronic Design Automation (EDA) industry is currently in a stage where there is considerable interest and research in the field of top-down design, hardware-software codeign and coynthesis. There has been considerable interest in the so called Electronic System Design Automation (ESDA) tools which support top-down design. In contrast, hardware-software codeign focuses on methodologies used for concurrently developing hardware and software components of a system to cost effectively satisfy the stated design constraints. The designer must consider trade-offs in the way the hardware and software components operate to exhibit a specified behavior, given a set of performance goals and implementation constraints [26]. This does not necessarily require an automation of the entire design process. Rather, the problem is best addressed with tools and/or methodologies to aid the human designer in his task. ESDA tools are concerned with providing facilities for top-down design of digital systems.

This paper describes our implementation of software (HDL) component libraries for hardware-software codeign, applicable to an SOS design environment. In this paper, we primarily address the design of high level HDL design libraries supporting the construction of performance models built as Queueing models and Petri nets. Queueing models and Petri nets have been chosen because of their close correspondence to hardware components and because these models achieve a favorable balance between efficiency, ease of modeling, and accuracy of the models. Furthermore, these libraries have also been augmented with components enabling the evolutionary replacement of uninterpreted regions in the design with interpreted implementations. The hardware description language VHDL [17, 18, 9] is used for this investigation.

Designing at such a high level of abstraction provides the designer with the flexibilities of making high-level partitioning decisions at an earlier stage of the design. Estimates for power, frequency, etc., can be made at the system level and early design decisions (like partitioning) taken accordingly. This provides a greater control of the design flow, allows the designer to make critical design decisions early in the design cycle and allows a smooth transition between the hardware and software components and vice versa.

The remainder of this paper is organized as follows. Section 2 provides some background and related work information. Section 3 describes the elements of a design libraries constructed to support evolutionary prototyping, hybrid modeling, hardware prototyping using FPGAs, and codeign. This section also presents a few case studies. Section 4 briefly addresses the integration of the model libraries with a graphical tool, RAPID-GrASS [14, 21] to facilitate hardware-software codeign using a top-down design methodology and performance visualization. Finally, section 5 contains some concluding remarks.

2 Background and Related Work

Rapid system prototyping using hardware and software components, performance analysis and evaluation form an important step in any design process. The building of accurate performance models is crucial for the successful realization of the most cost effective designs. However, the construction of accur
rate performance models is a difficult task as it often involves many speculative estimates of the parameters of the design being modeled. Formal techniques for design can greatly alleviate the difficulty of developing accurate system models.

Unfortunately, most CAD tools tend to concentrate on a particular type of system or a particular phase of the design process. This has led to the development of tools and environments satisfactory for only a particular design phase or design methodology. As the design evolves from the design phase to the implementation, the models representing the design are manually translated from one tool environment to the other — increasing the costs, probability of errors, design time, and also increasing the complexity of the design process. In such a situation, the design community requires a design environment which can represent the design throughout the design process — from design conception to final implementation.

Related Work

The approaches to automating the design of hardware/software systems are categorized as either fully-automated synthesis approaches [7, 23] or semi-automated design-driven approaches [21, 22, 23]. The first approach is attractive in that it automatically derives an implementation from a detailed functional specification and a set of performance constraints. However, this approach operates only in a limited design environment and often results in sub-optimal results. On the other hand, a semi-automated approach leaves the design decisions, the hardware/software partitioning decisions, in the hands of the designer and provides an environment in which the designer can implement and lead the design to meet functional and performance requirements. This paper deals with the latter approach.

Some popular modeling systems are TRANSCEND [23], N.2 [3], SIMS/GRT II.5 [3], SLAM II [20], SES/workbench24, and ADAS [6]. Most of these tools are modeling tools, which model systems at a high level of abstraction using Queues and/or Petri Nets, providing little or no support to co-design in an HDL environment. In ADAS, the hardware/software co-design methodology avoids a bottom-up approach to performance assessment by combining top-down design with iterative refinement of performance measures.

Ptolemy [11], from the University of California, Berkeley, is a hardware-software co-design system specifically targeted towards DSP applications. The Ptolemy framework is a very useful tool for simulation, prototyping, and software synthesis of systems containing both hardware and software components. The hardware/software partitioning is conducted manually in this system, providing the designer with advanced tools that enable a creative exploration of the design space. Our approach to co-design is based on the same principles. We however take the design process one step further in providing the designer with software components based on HDLs. These reusable components help reduce the complexity of the design cycle.

In the paper we consider the construction of generic VHDL design libraries to aid the designer in building rapid prototypes and conceptual system models. Hybrid modeling within a hardware-software codeign environment helps the designer refine the models to an implementation, ensuring that performance constraints are fully satisfied. A graphical visualization tool is also discussed. The tool helps the designer in visualizing the design and analyzing the design with performance results — all using colors and graphics.

3 Design Libraries

The VHDL Hardware Description Language (VHDL) [4, 18, 9] provides the designer with a powerful and flexible design environment for implementing interpreted modeling. By demonstrating strong VHDL support for uninterpreted modeling and hybrid modeling, we attempt to use VHDL to support the entire design process starting from the initial conceptual model to the final implementation. As the design progresses, the uninterpreted models of the system under design, are simulated to obtain performance statistics. These are used to tune up the design to meet the exact performance specifications. Finally, the interpreted model of the design can be implemented and simulated in the same environment. This helps in reducing the design time and the design costs. A single design environment also helps maintain an uniformity in the design decisions and the implementation, thus minimizing the speculative estimates of the parameters of the design.

Queueing models and Petri nets are popular modeling tools for the modeling and simulation of digital systems. These models are mainly classified as uninterpreted models and they operate by exchanging tokenized or event data between the system components. In general, a system model evolves from an uninterpreted form to an interpreted form by the piecewise replacement of uninterpreted components by interpreted components. The model should also support a mechanism to interface tokenized (or event) data with actual data.

In the following sections, we examine the construction of VHDL design libraries that allow the designer to easily construct system prototypes based on Queueing models and marked Petri nets.

3.1 Petri Nets and Queueing Libraries

Petri nets are widely used for building prototypes of digital systems. They have been shown to be equivalent to Turing machines in both their power to model computable systems and in decision power [19]. In a typical Petri net based performance modeling tool, the Petri nets incorporate the design of the digital hardware and software. Finally, the Petri net model is replaced by the hardware or software modules that correspond to the individual entities of the Petri net. The VHDL design entity interfaces are shown in Figure 1. The design entities are called place and transition corresponding to the notions of Petri net theory [1, 19]. Their implementation is completely hidden from the designer; he simply instantiates the design entities with the appropriate generic parameters and port bindings.
An important consideration in the construction of these library components is the easy instantiation of these models. These models are especially interesting as they can provide the designer with a highly efficient and accurate model of the system behavior and greatly aid in the analysis of the system.

The Queueing models examined in this paper have been implemented by instantiating VHDL [17, 18] components. More precisely, the following design entities for building Queueing models have been implemented: (a) event source model with token (event) generation characteristics, (b) a queue with generic parameters defining the buffer size and the queue discipline (FIFO, LIFO), (c) distribution module (or fork) which distributes input tokens (events) onto its outputs with different distribution characteristics, (d) join module which routes input events from various locations connected to its input arcs, to its output arc, (e) a set of server modules with different service distributions and (f) a sink module. The representations of these components are shown in Figure 2.

The VHDL library components for the Queueing model have been constructed with generic parameters as shown in Figure 3. The ports of the top-level design would specify the interconnections between the arcs of the various components used in the design. The various generic parameters have to be specified. For example, the queue buffer size is specified by defining the generic parameter number_entries in the queue entity. The generic argument out_control for the fork module can cause output to be generated on all the output arcs or on one output arc (chosen, for example, with a uniform distribution) [16, 27]. An important consideration in the construction of these library components is the easy instantiation of these models. These library components are built with generic parameters and the implementation details like handshaking protocols are hidden from the view of the designer. This is done to reduce the complexity of the design process and to help the designer make quick design changes. The designer has the flexibility to quickly manipulate the design parameters of the software models by changing the generic parameters of the components instantiated — reducing the complexity of exploring design alternatives.

### 3.2 Hybrid Modeling

A hardware-software codesign environment typically contains system modules at different levels of abstraction. In such a hybrid design environment, support for mixed or hybrid modeling provides the designer with a powerful technique to perform system modeling with components at different abstraction levels. In an uninterpreted model the information flow between components is denoted by tokens. The tokens represent the flow of data and the state of the system. No actual data value is transmitted anywhere within the model and no actual transformations are performed on data values. In contrast, an interpreted model is an abstract description of the implementation of the system containing functions and data values to be transformed according to these functions [3, 15, 23]. Typically, uninterpreted models are used at the conceptual modeling stage of the design and the interpreted models represent the later stages of the design cycle. To have a design environment supporting the complete design cycle, it is essential to have mechanisms supporting the integration of the two modeling techniques (a model that includes both uninterpreted and interpreted subassemblies — a hybrid model). In this work, we integrate these two types of models by providing mechanisms to translate data as it moves across the interpreted/uninterpreted boundaries.

The first approach is to translate data passed from the in-
The VHDL design entity declarations for interfacing the token/data boundary.

```vhdl
entity token_to_data is
  generic (number_in_arcs : integer := 1;
            number_out_arcs : integer := 1;
            debug_control    : debug := none);
  port (in_arc : in arc
         arc_array(1 to number_in_arcs);
         out_arc : in out
         data_array(1 to number_out_arcs) bus);
end;
```

Figure 5: The VHDL design entity declarations for interfacing the token/data boundary.

The VHDL process statement to interface the token/data boundary.

```vhdl
begin
  if input_token.handshake = sending then
    if (out_control = unif) then GenRnd(unif);
      rnd := unif.rnd;
      trand := rnd * 1.0 fs;
      elsif (out_control = norm) then GenNorm(norm);
      rnd := norm.rnd;
      trand := rnd * 1.0 fs;
      elsif (out_control = fixed) then
        trand := generation.mean * 1.0 fs;
      else assert false report "out_control error" severity error;
      end if;
    wait for trand;
    out_data <= token_to_data(input_token);
    end if;
```

Figure 6: VHDL process statement to interface the token/data boundary.

3.3 Evolutionary Prototyping

As the design of a system evolves, a designer should ideally be able to take early conceptual system designs and later be able to refine and develop the system to meet the stated design requirements. To be successful in this task, the design environment should facilitate prototyping and also support quick refinement and analysis of the design models in the same design environment, without requiring a large overhead on time and effort. The performance related data produced from the models should also be used to refine the system under design to meet the exact system requirements.

In this section, we present the modeling of a parallel computer system by replacing the components of the model by corresponding software design entities from our VHDL Queueing library. The model considered is a closed model of a parallel computer system [10, 12]. In the closed or limited population model, as opposed to an open or infinite population model, the number of customers are fixed and there are no arrivals or departures to the external world. Though closed models are generally more realistic than open models (since they capture physical limitations better), they are usually harder to solve. The Queueing model representing the equivalent of a closed model is shown in Figure 7. In this model, a customer modeled as a token (event), queues up at the CPU to have some computation done. After completing this computation, the job moves on to one of the Disks for some disk I/O. The CPU here is represented as a Queue and Server combination. The job going to the disks chooses disk1 with probability q and disk2 with probability (1-q). This is controlled by the Fork unit. In the model, the number of customers is fixed (as is the case in a finite population model) and is less than or equal to the maximum size of the queue buffer. When a customer returns to the Terminal station represented as a Server, that customer finishes and another statistically identical customer (token) takes its place in the network. The VHDL model of the design is simulated and the relevant performance data like utilization, average queue length, average service time, average waiting time for service and the total time spent by a token within the system are obtained. This data is very useful in characterizing the behavior.
of the system and determining the ideal partitioning between the hardware and software components of the design.

3.4 Hardware Components and Prototyping using FPGAs

At the high level system design state, the entire design is composed of multiple hardware and software components performing various tasks of the system under design. Typically, the designer attempts to bring the hardware and software components of the system close to one another such that in most cases the hardware/software boundaries are transparent. The hardware is emulated using complex Field Programmable Gate Arrays like the 26K or 40K gate ORCA FPGA family from AT&T [2]. FPGAs are very useful in these designs because of their ease of use, reprogrammability, and fast turn-around times. These chips provide the designer with the flexibility to map complex systems onto these devices as well as allowing him to change system specifications within no time by either reprogramming the FPGA device or by changing the generic parameters of the software libraries. In such a design environment the hardware and software boundaries are flexible and often transparent.

4 Integration with RAPID-GrASS

The design process gradually evolves from a conceptual model of the system to an implementation. The Queueing and Petri net libraries are used extensively during the conceptual design phase for the performance analysis and evaluation of the system. As the design evolves, these uninterpreted components are slowly replaced by the corresponding interpreted components. To aid in the analysis of the design process, this process has been integrated into a graphical hardware-software code-syn design environment which allows the user to graphically annotate his designs, build his designs, simulate them, and graphically view the simulation process and results — all within a common graphical framework. This tool, called RAPID, [14, 21] allows the user to complete the entire design process graphically. GrASS is a framework used with RAPID for the automatic analysis of design data and its graphical presentation into the graphical design (generally through color). Graphical design entry and analysis appeals to the intuition of the designer, thus enhancing his/her productivity and making the design process simpler and more attractive.

RAPID supports the use of textual as well as graphical representations of the design. That is, each component of the system can have up to three equivalent descriptions: (i) a (graphical) structural decomposition into its components, (ii) a behavioral VHDL description, or (iii) a software program representation. RAPID uses information from the design links to access the appropriate library and components and automatically generates structural VHDL descriptions. The simulation process in GrASS provides performance-related data for the design. The graphical design entry, graphical presentation of simulation steps and simulation statistics, and visualization of the performance metrics, helps the user have a better understanding of the behavior of the system. Changing the design is a trivial task as the designer just has to replace a graphical component with another graphical component and re-simulate the design, and carry on this process until an acceptable design has been achieved. Generic values passed to the entities can also be specified and modified.

5 Conclusion

In this paper, the construction of VHDL design libraries to support evolutionary prototyping, performance modeling and analysis, within a common hardware-software codesign environment has been demonstrated. The design entities in VHDL have a direct correlation to the corresponding component of the modeling domain, (e.g., queues, servers, sinks, places, transitions, etc.). These design entities have been designed with generic parameters which allow the user to configure each component as required in the design. The design entities have been built to make the design task as simple as possible, and to aid in the quick migration from one level of abstraction of the design to the other. The graphical design tool RAPID-GrASS also aids the designer in rapid prototyping of his designs, in performance evaluation and analysis of his designs. This is achieved in a user-friendly design environment which assumes no prior knowledge of the VHDL language.

Hybrid modeling, in a hardware-software codesign environment is also discussed. The interface between these two modeling environments translates uninterpreted tokenized data into interpreted actual data and vice versa. This translation between environments requires a thorough analysis and synthesis of the actual data values from statistical or trace data.

Acknowledgments

We would like to thank Venkata Ram Krishnamoorthy for helping us integrate the design libraries with the RAPID-GrASS graphical design tool. Many thanks to Peter Ashenden of the University of Adelaide for providing us with valuable suggestions and ideas for implementing the Queueing model libraries.
We would also like to thank Gnanasekaran Swaminathan of the University of Virginia for providing the random number generator package in VHDL that we used.

References


