Automated Generation of Accurate VLSI Behavioral Processor Models for Simulation and Synthesis

Yong-kyu Jung [1], *Vijay K. Madisetti* [1], *John W. Hines* [2]
[1] ECE - Georgia Tech, Atlanta, GA 30332-0250
[2] U.S. Air Force, WL/ELED Wright Patterson AFB, OH 45433-7319

Proceedings of *Second Annual ARPA RASSP Conference*, Crystal City, VA, July 25-27, 1995.

Abstract

A new process for automating the creation of Full-Behavioral (FBM) and Instruction Set Architecture (ISA) models in VHDL for complex processors and components is described, with results from the automation of a PowerPC 601 described in some detail. A number of advantages to this approach are described together with its impact on the hardware/software codesign and system prototyping processes.[1]

1 Introduction

The Rapid Prototyping of Application-Specific Signal Processors (RASSP) project of the US Department of Defense (ARPA and Tri-Services) targets a 4X improvement in the design, prototyping, manufacturing, and support processes (relative to current practice). As per E/F current practice (1993), the prototyping time from system requirements definition to production and deployment, of multiboard signal processors, is between 37 and 73 months [8]. Out of this time, $25 - 49$ months is devoted to detailed hardware/software (HW/SW) design and integration (with $10 - 24$ months devoted to the latter task of integration). With the utilization of a promising top-down hardware-less codesign methodology based on full-behavioral VHDL models of HW/SW components, it appears feasible that the HW/SW integration time could be reduced to a few weeks ($1 - 2$ months) [10]. Potential show-stoppers lie in the limited availability of high quality VHDL behavioral models of components (timing and function). In addition, the time to build a single model of a complex RISC processor (such as i860XP) is approximately a man-year. We describe a mechanism via which full-behavioral models of complex components can be automatically generated in VHDL from published information available in data manuals. This method could also be used in the iterative design synthesis of custom pipelined processors for domain/application-specific applications.

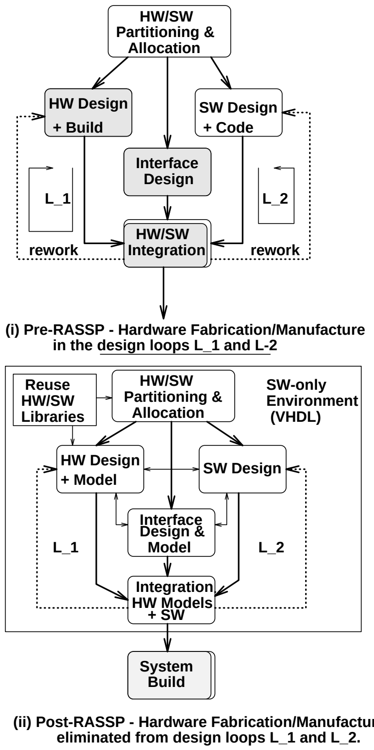

Fig 1. **HW/SW codesign - (i) Current practice (1993-1994) and (ii) True HW/SW codesign. Note elimination of hardware fabrication, assembly and board/system level manufacture from the design loops. Software can be tested on virtual hardware that is also concurrently being designed. Savings in time and cost, capability for customer input, and concurrent life-cycle support and upgrade planning is facilitated. Shaded areas imply hardware (board/MCM level).**

[1] This research was supported by Advanced Research Projects Agency (ARPA), Department of Defense, under the RASSP program, 1994-1997.

2 HW/SW codesign practice

The Educator/Facilitator current practice (1993) model for signal processor design is presented in detail in [8] in this proceedings. The various stages in a "waterfall"-type process flow are demarcated together with time ranges (min, max) for each stage. The time lines have also been validated via communications with the industrial entities involved in large system design and implementations. In this paper, we focus on the specific tasks of hardware, software, and interface design and their eventual integration.

2.1 Whither true codesign ?

True HW/SW codesign allows both hardware and software to be designed within a common framework, and simulated together before being fabricated. Current practice attempts to automate this process via HW/SW/Interface partitioning followed by three individual paths to HW, SW and Interface design and implementation, respectively (as shown in Figure 1). A drawback with this approach is that software can be designed and tested only if a hardware platform (at board and rack levels) is available. The latter is time- and cost-consuming (even if it utilized FPGA technology or HW modelers). It must be understood that the software is not just application-specific software, but also control, diagnostic and test software. Often, control, diagnostic, and test software requires an order of magnitude larger man-hour effort than does application software [8]. Conventional hardware software co-design methods assign a token interest in the issue of software required for control, diagnostic and test purposes, and attempt to catch all integration issues under the term "interface". The approach shown in Figure 1(ii) represents a "true" HW/SW codesign wherein software models (in a HDL such as VHDL) of HW are provided to the SW developers and the entire software is designed and tested and integrated with the HW models long before any hardware is fabricated or manufactured. Thus, the design loops L_1 and L_2 are quick, and require no hardware fabrication & engineering cost, and in addition provide capability for complete system design using a process known as *virtual prototyping* [1,5,10,11].

Virtual prototyping in top-down VHDL based design involves transition of the design through a number of abstraction levels, each of which represents an executable specification of the signal processor to be designed. At the network performance-level of virtual prototyping very little of the functionality (if any) of the target system is modeled, and the focus is on the utilization and efficiency of the SW implementation on a candidate HW architecture. Since this design takes place at a high level of abstraction, single events can represent functional blocks of a few tens to a few thousands of HOL instructions. The loss of functional detail is compensated by high simulation speeds (100,000-1,000,000 instructions/second) that can assist in the rapid architectural evaluation and selection. When limited functionality is added to performance models, additional estimates can be obtained about HW/SW functionality and timing (at the coarse level of wall-clock seconds) of prototype. Simulation speeds are reduced consequently (10,000-20,000 instructions/second). At an even higher accuracy of system modeling and simulation, with $bit-true$ functional and $clock-cycle$ level timing resolution, the use of ISAs and FBMs is recommended. This increase in accuracy is traded-off with slower simulation speeds (10-1000 instructions/second).

2.2 Showstoppers

The assumption, of course, is that libraries of full-behavioral HW models in SW are available, accurate, and interoperable, and that simulation times can be kept manageable. VHDL can be used with advantage in this true HW/SW codesign philosophy — one that embraces a hardware-less system design. Recent experience with hardware-less HW/SW codesign has shown that it is efficient, often reducing time for HW/SW integration to a matter of weeks, and also allows rapid upgrades, together with savings in cost [9]. Once virtual prototyping is completed, it is expected that that pathway through which a field prototype can be manufactured, supported and upgraded will be straighforward.

3 Modeling for HW/SW codesign

Several classes of models have been found suitable for HW/SW codesign. When emphasizing HW/SW integration, two classes have been found particulary useful - ISAs and FBMs. We will utilize the RASSP taxonomy [6] to define these two classes.

(1) *Instruction Set Architecture (ISA) Models* — An ISA model describes the function of the complete instruction set recognized by a given programamble processor, along with (and as operating on) externally known register set and memory/input-output space. An ISA model will execute any machine program for that processor and give exactly the same results as that processor (e.g., bit-true) as long as the initial states are the same for both simulation and the real system. Port registers, if modeled, are also bit-true. Instructions span multiple clock cycle, and ISA models need not contain any internal structural implementation information.

(2) *Full-Behavioral Models (FBMs)* — A FBM (also known as full-functional model [10]) is a processor model that exhibits all documented timing and functionality of the modeled component, without specifying internal structural implementation details. Thus, the full-behavioral model is more detailed than the ISA model in that it includes clock-edge timing information in addition to functionality. A number of full-behavioral processor models are available from Georgia Tech's RASSP Techbase effort [5]. The issue of creating ISA and FBMs will be examined next.

3.1 Populating VHDL model libraries

While complete or incomplete gate-level VHDL models are sometimes available from vendors, and are accurate for use as ISAs and FBMs, a number of limitations exist — (1) gate-level models are very slow in terms of simulation times, (2) reveal confidential component design (intellectual property) information, and (3) HW/SW codesign assumes that the hardware component is continuously being designed (e.g., changing instruction set, optimized behavior, etc), and thus the gate-level description does not exist. Thus, the focus is on creation of behavioral models of complex parts. Commercial Instruction-set simulators (ISS), which can provide debug information for processors, have limited applicability within a VHDL-based environment (without wrappers and loss in efficiency) where multiple models at varying levels of detail are co-simulated during the top-down design process. In addition, they do not allow redesign of the hardware component, a trend that is increasing finding favor in application-specific markets (e.g., use of core-based functional design of DSP ASICs [2]).

The current approach to model development is best described in [3,4,10]. All these approaches model the internal and external microarchitecture of the component behaviorally from manufactured-supplied data (or via abstraction to higher levels of functional and timing information from gate-level descriptions). This is a manual, time consuming (in man-years), and error-prone (i.e., verification) operation, and often equivalent to designing the component all over again. While we have used this approach, and continue to use this approach in developing ISAs and FBM models, an investigation into automated generation of these models was long overdue.

3.2 A new approach - autogeneration

An alternative approach to developing ISAs and FBMs that is automated is described in Figure 2. The processor being modeled (or designed) is described by parametrized generalized time-stationary [2] pipelines (single or multi-), associated memories/registers, and a generalized controller. The user-defined or vendor-supplied information on the instruction set, architectural constraints (hazards, timing), are captured in terms of processor-specific input data files. These parametric input data files then are automatically converted to lookup tables (LUTs). The LUTs are utilized by the AMG to generate the control (timing) and functional information from the input application instruction stream. We have used this approach to synthesize behavioral models of the PowerPC 601 RISC processor and an implementation will be described in the next section.

3.3 A new approach - iterative synthesis

The approach described in Figure 3 describes the process flow for automating the iterative synthesis of application-specific processors. Here the instruction-set of a programmable processor can itself be customized and iteratively designed during the HW/SW codesign process. The application drives the iterative instruction-set and architecture codesign (which are captured from input data files as LUTs) by the

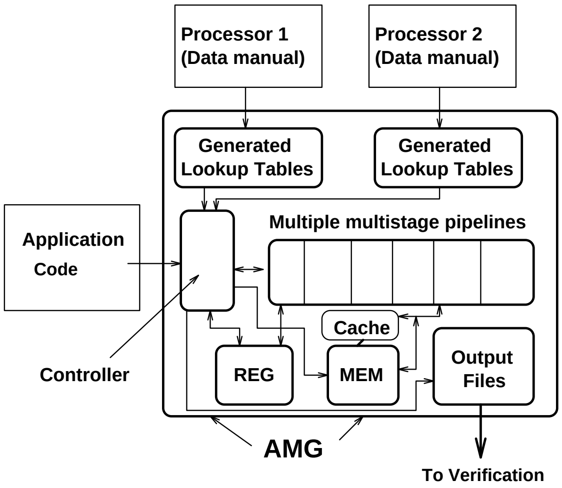

Fig 2. **Automated Processor Model Generation (AMG) for Simulation**

AMG. The controller, pipeline, and associated logic of the AMG are then simulated to measure performance on the target application. After optimization of the instruction set and timing, the AMG may be synthesized using commercial RTL-level or behavioral synthesis tools. Application-specific functional libraries can also be used with advantage when combined with VHDL and the emerging VITAL standards for sign-off quality timing simulation. Future papers will discuss and document the approach of Figure 3.

4 Automated Model Generator - AMG

The automated model generator (AMG) is an ISA or FBM model that accepts the application instruction stream and processor-specific data in the form of input tables, that are processed internally to provide all documented functional and timing characteristics as output files. The same AMG can be reused for creating models of multiple versions of the same chip, or independent families of processors.

4.1 Structure of the AMG

The automated ISA model generator consists of six major "blocks", as described below (See Figure 4).

1. **Pipeline:** A single pipeline for a RISC processor consists of the following six stages — (1) Instruction Fetch (IF), (2) Instruction Dispatch (IDP), (3) Instruction Decode (ID), (4) Instruction Execute (IE), (5) Cache Access (CA), and (6) Write back (WB). These stages were implemented as procedures within a VHDL process description of the pipeline.

2. **Memory Block (MB):** The MB consists of an *instruction queue (IQ), instruction and data memories (IM & DM)*, and a *cache (CACHE).*