RACE: A Rapid, ArChitectural Simulation and Synthesis Framework for Embedded Processors

Roshan Ragel[1], Angelo Ambrose[2],
Jorgen Peddersen[2], and Sri Parameswaran[2]

[1] Embedded Systems and Computer Architecture Lab (ESCAL)
Department of Computer Engineering, University of Peradeniya,
Peradeniya 20400 Sri Lanka

[2] Embedded Systems Lab (ESL)
School of Computer Science and Engineering,
University of New South Wales,
Sydney 2052 Australia

Abstract. Increasingly, embedded systems designers tend to use Application Specific Instruction Set Processors (ASIPs) during the design of application specific systems. However, one of the design metrics of embedded systems is the time to market of a product, which includes the design time of an embedded processor, is an important consideration in the deployment of ASIPs. While the design time of an ASIP is very short compared to an ASIC it is longer than when using a general purpose processor. There exist a number of tools which expedite this design process, and they could be divided into two: first, tools that automatically generate HDL descriptions of the processor for both simulation and synthesis; and second, tools that generate instruction set simulators for the simulation of the hardware models. While the first one is useful to measure the critical path of the design, die area, etc. they are extremely slow for simulating real world software applications. At the same time, the instruction set simulators are fast for simulating real world software applications, but they fail to provide information so readily available from the HDL models. The framework presented in this paper, RACE, addresses this issue by integrating an automatic HDL generator with a well-known instruction set simulator. Therefore, embedded systems designers who use our RACE framework will have the benefits of both a fast instruction set simulation and rapid hardware synthesis at the same time.

Keywords: Design Automation, Simulation, Synthesis

1 Introduction

Embedded systems are ubiquitous, and are present in low-end systems such as wireless handsets, networked sensors, and smart cards, to high-end systems such as network routers, gateways, firewalls, and servers. Embedded systems are seen

as application specific equipment and they differ from general purpose computing machinery since they execute a single application or a class of applications repeatedly.

The heart of an embedded system is usually implemented using either general purpose processors, ASICs or a combination of both. General Purpose Processors (GPPs) are programmable, but consume more power than ASICs. Reduced time to market and minimized risk are factors which favour the use of GPPs in embedded systems. ASICs, on the other hand, cost a great deal to design and are nonprogrammable, making upgradability impossible. However, ASICs have reduced power consumption and are smaller than GPPs.

Recently a new entrant called the Application Specific Instruction-set Processor (ASIP) has taken centre stage as an alternative contender for implementing functionalities in embedded systems. These are processors with specialized instructions, selected co-processors, and parameterized caches applicable only to a particular program or a class of programs. An ASIP will execute an application for which it was designed with great efficiency, though they are capable of executing any other program (usually with reduced efficiency). ASIPs are programmable, quick to design and consume less power than GPPs (though more than ASICs). ASIPs in particular are suited for utilization in embedded systems where customization allows increased performance, yet reduces power consumption by not having unnecessary functional units. Programmability allows the ability to upgrade, and reduces software design time. Tools and customizable processors such as ASIPmeister [1], Xtensa [2], LISATek [3], ARCtangent [4], Jazz [5], Nios [6], and SP5-flex [7] allow rapid creation of ASIPs. The advent of tools to create ASIPs has greatly enhanced the ability to reduce design turnaround time.

However, there exists a limitation. The tools listed above except the one presented in [4] will either generate the hardware description language (HDL) model of the embedded processor or a model where only Instruction Set Simulation (ISS) could be performed. The HDL models are good for precise synthesis and power measurement of the processor, but fail to provide fast simulation results such as the clock cycle count of an application that runs on such a model. The ISS models are good for faster simulation of applications, but fail to provide synthesis results which are essential in embedded system design. Even though tools such as the one from Tensilica [2] try to address this issue, they do not provide the flexibility (such as accurate power measurement using the HDL model, full control of the instruction set of the processor, etc.) expected in other ASIP design tools such as ASIPmeister.

In this paper, we present a framework, named RACE, which provides both an ISS model for fast simulation and an HDL description for fast synthesis of an embedded processor during its design. We make use of ASIPmeister [1], an automatic processor generation tool for preparing the HDL model and SimpleScalar tool-set [8] for preparing the ISS model. The detail of how these are integrated to form the RACE framework is discussed in this paper.

The rest of this paper is organised as follows. Section 2 summarizes the previous work related to embedded processor simulation and synthesis. Section 3 details our framework. Section 4 explains how our framework incorporates processor customization and Section 5 discusses a typical experimental setup of our framework. Finally, Section 6 concludes the paper.

2 Related Work

With the demand for shorter design turnaround times, many commercial and research organizations have provided base processor cores, so that fewer modifications have to be made on the design to achieve particular performance requirements. This has led to the emergence of reconfigurable and extensible processors. Xtensa [2], Jazz [5] and PEAS-III (used by ASIPmeister) [1] are examples of processor template based approaches which build ASIPs around base processors.

Xtensa [2] is a configurable and scalable RISC core. It provides both 24-bit and 16-bit instructions to freely mix at a fine granularity. The base processor supports 80 base instructions of the Xtensa Instruction Set Architecture (ISA) with a 5-stage pipe-line. New functional units and extensible instructions can be added using the Tensilica Instruction Extension (TIE) language. Synthesizable code can be obtained together with the software tools for various architectures implemented with Xtensa. However, it fails to provide the flexibility for altering the base processor.

The Jazz Processor [5] permits the modelling and simulation of a system consisting of multiple processors, memories and peripherals. Data width, number of registers, depth of hardware task queue, and addition of custom functionality are its input parameters. It has a base ISA which supports addition of extensible instructions to further optimize the core for specific applications. The Jazz processor has a 2-stage instruction pipeline, single cycle execution units and supports interrupts with different priority levels. Users are able to select between 16-bit or 32-bit data paths. It also has a broad selection of optional 16-bit or 32-bit DSP execution units which are fully tested and ready to be included in the design. However, Jazz is suitable only for VLIW and DSP architectures.

ASIPmeister [1] is able to capture a target processors specification using a GUI. A micro-operation level simulation model and RTL description for logic synthesis can be generated along with software tool chain. It provides support for any RISC architecture type and a library of configurable components. The core produced follows the Harvard style memory architecture. Even though it provides both the simulation and the synthesisable models, the simulation model could only be used with an HDL simulator such as ModelSim and therefore, real world applications will take hours (if not days) for simulation. Researchers have proposed extensions to ASIPmeister, such as the one presented in [9], so that it could be used as a fully fledged simulation system with system call support, file handling, etc. However, they failed to solve the problem of the extended simulation time taken to simulate real world applications as explained earlier.

The RACE framework we propose here uses similar techniques to that of [9] to generate the synthesis model of the processor. However, we propose to use an independent instruction set simulator which is derived from the SimpleScalar tool-set [8] for faster simulation of the same processor. We show how the instruction sets could be altered (reduced/amended/added) in both the simulation and the synthesis models of a target processor by taking PISA, the ISA used in the SimpleScalar tool-set as an example.

Therefore, in summary, the contributions are:

- a framework that performs both fast simulation and synthesis of an embedded processor model; a fully flexible and rapid ASIP design flow based on our framework; and,
- a scheme on how an instruction set could be altered to explore the design space of both the simulation and synthesisable models.

However, there exist the following limitations:

- Designing the initial models of the processors might take a longer time (a day or two to a familiar designer). However, this is a one-time process and the same model could be used later for rapid design development.
- It is assumed that the compiler tool-set is available as open source for the instruction set used in the design.

3 The RACE Framework

RACE is a hardware-software co-design framework, where both the software binary of a target application as well as the hardware model to run such a binary are designed and implemented. In this section, we explain the process of software binary generation a target ISA, and then we describe the generation of the hardware models, for instruction set simulation and for synthesis.

3.1 Software Generation

SimpleScalar cross compiler (such as *sslittle-na-sstrix-gcc*) is used to generate the instruction and the data memory dump (we call it the binary) from the application program. In the HDL models, both memories will communicate with the CPU model to function as a complete processor, executing the program. Further details on the memory generation can be found in an earlier publication [9].

Figure 1 depicts the typical software generation process. A C/C++ application is compiled to the target binary by using the SimpleScaler compiler tool-set using a cross compiler.

As depicted in Figure 1, if necessary, support for new instructions (to the ISA) is added to the assembler of the SimpleScalar cross compiler. Given that the cross compiler is a derivative of the well understood open source GNU/GCC compiler tool-chain; this task can be performed with relative ease. When the support

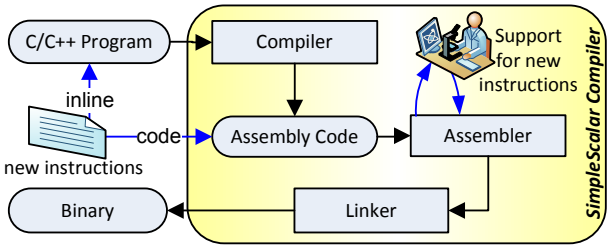

Fig. 1. The software generation process: Support for new instructions can be added to the assembler and programs can be written with either in-lined or added new assembly instructions.

for new instructions is available in the assembler, application programs can be written either in a higher level language like C with inline assembly (for new instructions) or in the target assembly language by using the new instructions. Here, the new instructions will both be designed and inserted (to the application) manually by the designer. Even though, this could be considered a limitation of the RACE framework (as pointed out earlier under limitations), we argue that it gives better control of the design flow to the designer. If absolutely necessary, support for such automation can be established by extending the compiler tool-set.

3.2 Hardware Generation for Simulation

Figure 2 depicts the generation process of hardware models in RACE framework. As depicted, RACE framework generates three hardware models of an ASIP from two input set of specifications.

All three models depicted as derived in Figure 2 can be used for simulation of an application program. However, as these three models vary in the level of detail used for implementing the hardware, the times taken to perform the simulation vary significantly. For example, while a typical embedded system application would take days (if not weeks) to be simulated using gate level simulation, it can be done in seconds or minutes using an instruction level simulation.

Therefore, the RACE framework uses the ISS to run complete application program simulations. Given that these simulations are cycle accurate, they will be used to count the number of clock cycles taken to simulate applications. The number of clock cycles along with the clock period (that is calculated from the synthesis discussed in the next subsection) is used to compute the execution time of an application, one of the main design metrics of any ASIP design.

SimpleSim, the ISS of the SimpleScalar tool-set is used to derive the ISS for RACE. The modular design of SimpleSim allows us to add/remove/amend instructions of the target ISA. As depicted on the right hand side of Figure 2, the machine.def file of SimpleSim is altered to change the target ISA.

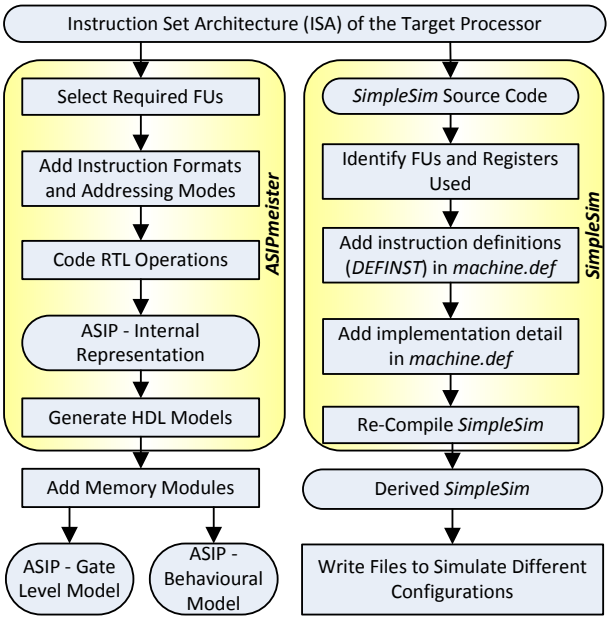

Fig. 2. The generation process of hardware models: On the left side is the HDL generation (both gate [ASIP - Gate Level Model] and behavioural [ASIP - Behaviour Model] models) with the help of ASIPmeister and on the right is the ISS generation for cycle accurate simulation.

3.3 Hardware Generation for Synthesis

An ASIP design tool, ASIPmeister, generates a model in HDL (both gate level and behaviour models) for a given ISA. As shown on the left side of Figure 2, to generate a processor using ASIPmeister, the first step is to create a suitable description of the processor, including the hardware resources (such as register file, ALU, divider, etc.) and pipeline stages. The instructions, their formats and addressing modes and the tasks to be performed by each instruction at run-time are defined as micro-operations (RTL operations), where each pipeline stage of the instruction is coded.

RACE uses the Portable Instruction Set Architecture (PISA: as implemented in SimpleScalar tool-set [8]) as its base processor. However, the base ISA could be of any other RISC processor. When the processor models (both gate level and behavioural) are generated, they are integrated with HDL models of memory modules to complete the ASIP models. Additional hardware can now be added to the design such as cache and memory mapped I/O.

4 Customized ASIPs

When the base models are designed in RACE, they can be customized in a number of ways either to explore the design space with different configurations or to add a totally different domain of tasks (such as instruction changes to perform security checks [11]) to the models. We discuss such customizations in this section.

Most of the applications hardly utilize the whole instruction set of a processor, thus the need for ASIPs. If an application does not need a specific instruction, it would be quite useful to turn off that instruction from the processor. This will reduce the area usage and power consumption, benefiting an embedded system [9].

ASIPs are famous for 'special instructions', instructions that are not available in the base ISA. RACE allows its users to have their own instructions. Special instructions can be utilized to add new customized hardware modules to perform repeated tasks and therefore make the processing faster [11].

5 Simulation and Synthesis Setup

Figure 3 depicts the simulation and synthesis setup used by RACE framework. The behavioural model is typically used during the design stage for debugging and testing of the ASIP (by performing simulation in ModelSim). The debugging and testing is performed by running test applications to cover all the instructions in our target ASIP. The completed gate level model is used with Synopsys Design Compiler to create the synthesized version, which is ready to be fabricated. The software binary is an input to the synthesis model, by which the memory size of the ASIP can be computed. Synthesis reports include power consumption, clock period and, area in gates and cells.

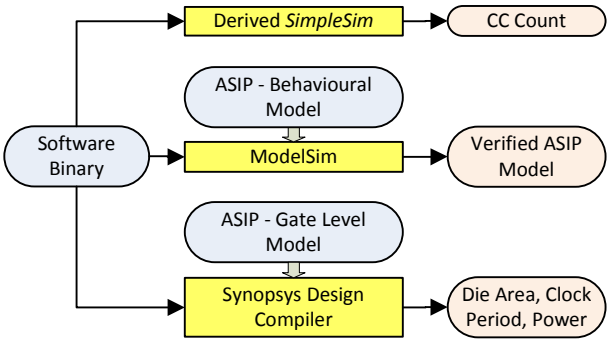

Fig. 3. Simulation and Synthesis: Simulations are performed by using the SimpleSim ISS and ModelSim. Synthesis is performed by using Synopsys Design Compiler.

Simulation is performed with SimpleSim to count the number of clock cycles (CC) particular software binaries would take. CC is multiplied by clock period (a metric computed from the synthesis using Synopsys Design Compiler) to compute the total execution time of the application.

Comparing the design time of large design problems with and without the RACE framework is currently being performed. We propose this as a future work for this paper.

6 Conclusion

In this paper, we reported RACE, a simulation and synthesis framework for rapid hardware-software co-design of ASIPs. RACE framework integrates an automatic HDL generator with a well-known instruction set simulator to support rapid processor development.

References

1. Itoh, M., Higaki, S., Sato, J., Shiomi, A., Takeuchi, Y., Kitajima, A., Imai, M.: In: Proceedings of 2000 International Conference on Computer Design, pp. 430-436. (2000)
2. Tensilica Inc., Xtensa Processor, Available at `http://www.tensilica.com`
3. CoWare Inc., LISATek, Available at `http://www.coware.com/products/`
4. ARC International, ARCtangent, Available at `http://www.arc.com`
5. Improv Inc., Jazz DSP, Available at `http://www.improvsys.com`
6. Altera Corp., NIOS Processor, Available at `http://www.altera.com`
7. 3DSP Corp., ARCtangent, Available at `http://www.arc.com`
8. Burger, Doug ,Austin, Todd M.: The SimpleScalar tool set, version 2.0. SIGARCH Computer Architecture News. 25, 3, 13–25 (1997)
9. Peddersen, Jorgen, Shee, Seng Lin, Janapsatya, Andhi, Parameswaran, Sri: Rapid Embedded Hardware/Software System Generation. In: Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05), pp. 111–116. (2005)

10. Ragel, R.G., Parameswaran, S.: IMPRES: Integrated Monitoring for Processor REliability and Security. In: Proceedings of the Design and Automation Conference 2006 (DAC'06), pp. 502–505. ACM Press (2006)
11. Hoon Choi, Jong-Sun Kim, Chi-Won Yoon, In-Cheol Park, Sung Ho Hwang, Chong-Min Kyung: Synthesis of application specific instructions for embedded DSP software. IEEE Transactions on Computers 8, 6, 603–614 (1999)