A cycle-accurate synthesizable MIPS simulator in Simulink

Thomas Sideropoulos
Dept of Electrical and Computer Engineering
Aristotle University
Thessaloniki, Greece
thomasgs@auth.gr

Nikos P. Pitsianis
Department of Computer Science
Duke University
Durham NC, USA
nikos.pitsianis@eng.auth.gr

ABSTRACT

We introduce a novel methodology for creating a synthesizable, cycle-accurate simulator of the MIPS32 processor with concise, high-level programming expressions using Simulink and other MATLAB tools. The simulator, named SimuMIPS, is capable of running binaries generated by the GNU gcc compiler and associated binutils. It can be easily configured, modified and extended not only for academic instruction but also to be included in commercial SOC products. Synthesizable instantiations of SimuMIPS in Verilog and VHDL may be generated by Simulink HDL Coder for FPGA programming and system-on-chip prototyping. In addition, the SimuMIPS simulator can run on embedded processors, rapid prototyping boards, and off-the-shelf microprocessors via the Embedded Coder generated C and C++ implementations.

Categories and Subject Descriptors

D.2 [Software system structures]: Software architectures, Abstraction, modeling and modularity—Simulator / interpreter; I.6 [Modeling and simulation]: Model development and analysis—Modeling methodologies; B. [Hardware]: Design reuse and communication-based design—Hard and soft IP

General Terms

Design

1. INTRODUCTION

We introduce in this paper a novel methodology, by adopting and utilizing Simulink [4], for rapid design and development of SimuMIPS, a MIPS32 processor simulator that allows customization and adaptation to a particular embedded system.

For every processor ever built, several software simulators are usually built for different roles: as an intermediate stage of the processor design, an integral part of a software development kit for that processor, a testbed for performance evaluation in certain aspects, or an education kit [10].

While the advantages of building and using software simulators are well recognized and leveraged, there is a great need to advance simulation techniques. Presently, building a simulator for a complex system, such as a MIPS processor simulator, entails meticulous effort to exact detail at a specific level. In addition, if the simulator is required for a co-design of an embedded system, there is the lack of flexibility for adaptation. Commercially available soft processors are monolithic in their availability, which limits the scope of customization and hence optimization for the embedded system design. Rather than simply rendering another processor simulator, we present an elegant and effective methodology for expediting the design process for a cycle-accurate processor simulator as well as enabling customization.

The only related work we were aware of is that by Ou and Prasanna [8] for simulating the execution of software programs running on soft processors with the use of Simulink. In their paper they proposed a high-level cycle-accurate hardware/software co-simulation environment, based on MATLAB and Simulink, for application development using soft processors. They demonstrated their proposed system by implementing a Simulink block for the MicroBlaze soft processor. However, the MicroBlaze block is not a model of the processor, but rather an intermediary facilitator that communicates with the Xilinx-provided cycle-accurate simulator via the GNU debugger mb-gdb using the TCP/IP protocol. We built a fully functional cycle-accurate model of the processor to be simulated.

Our methodology can be described as a cascade of stages, in concept, abstraction and development. The design is simple with minimal functionality at the initial stage, and refined in detail and expanded in functionality in the subsequent stages. The methodology is facilitated by Simulink and code generators developed by MATHWORKS.

A traditional programming language for hardware, like SystemC [3], might be a more appropriate choice, if implementing a processor was the only task. However, if the processor implementation is part of a wider application, for instance, a co-design environment for a signal processing embedded system; the capability to implement the whole application in a rich environment like Simulink, becomes important and the availability of a programmable processor as part of the simulation indispensable.

We describe in Section 2 the basic features of the tools we have used. We describe our new methodology in Section 3 by illustrating with the process of generating a cycle-accurate synthesizable MIPS simulator. MIPS is important...
in its own right. MIPS implementations are in many embedded systems, such as portable devices, routers, and video game consoles. In the past they were also used in SGI, DEC computers and many others. The MIPS architecture is also commonly used at universities in teaching and learning computer architecture and assembly programming.

2. TOOLS

We describe the critical and rich features of Simulink and two related code generator tools we have used to support the new methodology for simulator design, development and generation.

Simulink, by MathWorks, is a user-friendly and productive programming environment for system model design, simulation and analysis [4]. Among others, Simulink has been used in system evaluation, verification, and validation, to system production. Its applications have grown into many other research and development domains beyond signal and image processing system design.

We utilize the following features of Simulink. The programming is graphical with diagrams of component blocks and connections. Via a graphical user interface (GUI), designers specify components by simply dragging and dropping individual components from library and user-provided blocks. Connections describe the flow of control, data and signals. Signal propagation is scheduled and synchronized by Simulink. The diagrams, with their blocks and connections, can be drawn, specified and displayed at different levels of detail. This feature supports our design in cascaded stages.

Simulink is closely coupled with MATLAB. The functionality of the blocks can be specified by corresponding functions in MATLAB.

Simulink models can be transformed by two code generators from MathWorks into system prototypes and products in hardware or software. We use Embedded Coder [1] to generate C and C++ implementations as stand-alone programs that can be executed on embedded processors, rapid prototyping boards, and off-the-shelf microprocessors. We also use Simulink HDL Coder [2] to generate synthesizable instantiations of SimuMIPS for FPGA programming in Verilog and VHDL. This has the additional benefit in achieving higher simulation speed on Xilinx and Altera FPGA boards.

3. METHODOLOGY

We describe in this section our methodology by illustrating and detailing the design and development of SimuMIPS, a MIPS simulator. MIPS is a reduced instruction set computer architecture developed by MIPS Technologies. Multiple revisions of the MIPS instruction set exist, including the revisions MIPS32 and MIPS64 for 32-bit and 64-bit implementations, respectively [6, 7]. In SimuMIPS, we follow the same design patterns as described in the Patterson and Hennessy textbook [9] for the implementation of the MIPS32 processor. The design at the initial stage was simple. It consisted of the program counter, the data memory, the register file, which are connected and controlled by a deliberately oversimplified control unit. Each of the basic components can be easily specified by a MATLAB function. Once the basic components were developed and operated in a single cycle, we proceeded to the next stage. We introduced the pipeline stages, added the intermediate buffers and dealt with the potential pipeline hazards.

![Figure 1: The Register File block in Simulink.](image)

The MIPS architecture is functionally completed with the arithmetic-logic unit (ALU), the control unit and program memory. At least one instruction is implemented for each different instruction type: arithmetic, arithmetic immediate, load, store and branch on equal. These instructions were used as prototypes for the design of the basic single-cycle datapath. Along the design process, we also built necessary testing scaffolds to provide the input signals from the yet-to-be implemented components, to validate and visualize the output.

We give further detail. Figure 1 shows the graphical representation of the MIPS Register File in Simulink and the input and output signals. No clock is required for block synchronization, because the clock cycle is matched with the simulation step.

We list below the complete MATLAB function that describes the functionality of MIPS Register File. The function supports the corresponding Simulink block.

```matlab
function [readData1, readData2, dumpReg] = registers(regWrite, readAddr1, readAddr2, ... writeAddr, writeData)
    %#codegen
    % Register File
    persistent regFile
    N = 31;
    if (isempty(regFile))
        regFile = zeros(N,1);
        regFile(28) = hex2dec('10008000'); % gp
        regFile(29) = hex2dec('7FFFEFFC'); % sp
    end
    if (regWrite == 1 & writeAddr ~= 0)
        regFile(writeAddr) = writeData;
    end
    readData1 = 0; % R0 = 0 hardwired
    if (0 < readAddr1 & readAddr1 <= N)
        readData1 = regFile(readAddr1);
    end
    readData2 = 0; % R0 = 0 hardwired
```
if (0 < readAddr2 & readAddr2 <= N)
    readData2 = regFile(readAddr2);
end

dumpReg = regFile;

The input to the MATLAB function, named registers, consists of the data to be stored in a register, the read and write address of the registers to access, and the write-control signal. The output consists of the two read registers. The entire register array is also exported to the base workspace as part of the machine state. The register file consists of the persistent vector regFile with length 31, accommodating registers $1$ to $31$. Register $0$ is hardwired to zero. Registers $28$ and $29$ are used for global pointer $gp$ and stack pointer $sp$. MIPS memory map places the $gp$ at $0x10008000$ and $sp$ at $0x7FFFEFFC$. In each cycle, first, the specified register is written when the enabling write signal is on, followed by the two registers read.

We list next a code portion of the MATLAB function, named ALU, in order to illustrate the basic functionality of certain instructions implemented in the ALU.

function aluOut_t = ALU(aluA, aluB, alu_ctrl, sa)

    %#codegen
    in1 = uint32(aluA);
    in2 = uint32(aluB);
    persistent hi lo                 % 64 bit accumulator
    if (isempty(hi))
        hi = uint32(0);
    end
    if (isempty(lo))
        lo = uint32(0);
    end
    switch alu_ctrl
        case 0  % and
            aluOut = bitand(in1, in2);
        case 1  % or
            aluOut = bitor(in1, in2);
        case 2  % addu
            imax = intmax('uint32');
            if (in2 > (imax - in1))
                aluOut = in2 - (imax - in1) - 1;
                ovf = 1;
            else
                aluOut = in1 + in2;
                ovf = 0;
            end
        case 3  % lui
            aluOut = uint32(bitshift(in2,16));
    end

    if (ovf == 1)
        aluOut = uint32(bitshift(in2,16));
    end

Inputs to the MATLAB function ALU are the two operands aluA and aluB, the control signal alu_ctrl from the ALU control unit, which handles the operation codes originating from the control unit block and the shift amount sa, originating from the R instruction field. Note the implementation of 64 bit accumulator, formed by two 32 bit persistent variables, hi and lo.

SimuMIPS also implements the MIPS parallel pipeline multi-cycle datapath to complete one instruction per execution step. Figure 2 displays the graphical representation of the MIPS processor in Simulink showing the pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Write Back (WB). Addressing pipeline hazards was the most complex part of the design. It involved gathering control and data signals from the execution steps and determining whether a pipeline hazard will occur. The hazards were resolved using forwarding techniques and stalls [9].

SimuMIPS in its present version implements most of the MIPS32 instruction set. Next versions will include signed operations, traps, interrupts, load/store byte and halfword instructions, floating point operations, certain branch and link instructions, system calls and privileged instructions (cache, wait, exception return). The design of the exception handling unit is to be completed and is in progress.

4. TESTING

We evaluate the resulting simulator design with a suite of test programs. The test programs are in assembly and C. They are compiled with 4.7.3 GNU mips-elf-gcc cross-compiler for MIPS, provided by Sourcery Codebench [5]. Stress tests are carried out, with simple assembly programs that test each implemented instruction, all possible instruction formats, the pipeline datapath, the forwarding mechanism and all possible conditions that may give rise to pipeline hazards.

More complicated instruction sequences were tested with programs in C. A collection of 35 small C programs which implement most C language constructs including pointers, arrays, structures, recursive function calls, static memory allocation and complex combinations of loops and conditions were used. We modified the C programs to make them self testing and return special codes to be recognized as a pass or a failure.

We automated the test process with makefile scripts that compile, execute and collect the return codes of all test codes from a given directory and produce a test report. We are in the process of constructing and collecting a wider set of test cases of various forms of complexity. Strategically, we have short tests carried out in order to give quick feedback to any change in the simulator design, followed by longer tests to test pipeline hazards, and exhaustive tests prior to a new release.

5. DISCUSSION

We have introduced a new methodology for rapid design and development of a complex processor system by utilizing Simulink. We have demonstrated the methodology and its effectiveness by the process of designing and implementing SimuMIPS, a cycle-accurate MIPS processor simulator, via multiple-level abstraction. The implementation of SimuMIPS can be re-targeted to run on general and embedded processor systems via the MathWorks Embedded Coder generator as well as into HDL via MathWorks HDL Coder to run on FPGAs.

This work aims at facilitating and accelerating the design
process. It is also intended to bring together research, development and education. The methodology with multi-level abstraction, development and visualization will help clarify the basic concepts and engineering mechanisms in teaching and learning of processor design and development. By adopting and utilizing Simulink, we also make connections among system designs in different discipline areas.

In its current version, SimuMIPS can execute most commonly used MIPS instructions and can run ELF binaries generated by the GNU C compiler and binary utilities. The Simulink model file, directions to get and install the GNU MIPS-ELF cross compiler, makefile script and sample C and assembly test codes can be found at the website in the footnote\(^1\). In subsequent versions, SimuMIPS will be completed with interrupts and traps, the implementation of common system calls to accommodate rudimentary screen and file I/O as well as a simple console to display the machine status.

6. ACKNOWLEDGMENTS

The authors thank the referees and Xiaobai Sun for their constructive comments. Nikos Pitsianis acknowledges the support of the EU via a Marie Curie International reintegration grant.

7. REFERENCES

[1] Embedded Coder, Generate C and C++ code optimized for embedded systems.


\(^1\) http://www.cs.duke.edu/~nikos/SimuMIPS