Verification of e-class processor based on RISC-V ISA

A Project Report

submitted by

B V S Kavya Sree
EE12B012

i*n Partial Fulfillment of the Requirements*
for the Degree of

Bachelor of Technology

DEPARTMENT OF ELECTRICAL ENGINEERING INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

June 2016

THESIS CERTIFICATE

This is to certify that the thesis entitled **Verification of E-class processor based on RISC-V ISA**, submitted by **B V S Kavya Sree, EE12B012** , to the Indian Institute of Technology Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. V. Kamakoti
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT Madras, 600 036

Place: Chennai

Date: **20th June, 2016**

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude towards several people who enabled me to reach this far with their timely guidance, support and motivation.

First and foremost, I offer my earnest gratitude to my guide, Dr. V. Kamakoti whose knowledge and dedication has inspired me to work efficiently on the project and I thank him for motivating me, and allowing me freedom and flexibility while working on the project.

My special thanks and deepest gratitude to Rahul B and Vinod G who has been very supportive with invaluable suggestions.

ABSTRACT

The project involves understanding and verification of E-class processor(SoC) which is based on 32-bit RISC-V Instruction set architecture. The E-class processor is implemented using 3-stage pipelining which is realized in Bluespec SystemVerilog(BSV). The functionality and correctness of the e-class processor is verified using the AAPG and Spike tools.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

BSV	Bluespec SystemVerilog
TLM	Transaction Level Modelling
FIFO	First In First Out
RISC	Reduced Instruction Set Computer
HDL	Hardware Description Language
CPU	Central Processing Unit

CHAPTER 1

Introduction

1.1 Overview

The Processor design team of Reconfigurable and Intelligent Systems Engineering (RISE) Lab in the Computer Science Department of IIT Madras has been actively involved in research of The SHAKTI Processor project. The SHAKTI processor project aims to build variants of processors based on the RISC-V ISA from UC Berkeley. The project will develop a series of cores, SoC fabrics and a reference SoC for each core family. One such core variant is the E-Class processor. It is a 32 bit 3 stage in-order core aimed at 10 - 50 Mhz uC variants. The processor has optimal memory protection and very low power static design. The design of the E-class processor is done using a Hardware Description Language(HDL) named Bluespec SystemVerilog(BSV). This report aims at the description of the implementation of E-class processor and its verification using tools like AAPG- Automatic Assembly Program Generator and SPIKE, a RISC-V ISA Simulator.

1.2 Organisation of thesis

Chapter 2 gives some insight about the HDL- Bluespec SystemVerilog- its key features and Python Scripting

Chapter 3 discusses about RISC-V Instruction Set Architecture and different base encodings.

Chapter 4 gives us an understanding of the implementation of 3 stage processor with intermediate FIFOs

Chapter 5 contains an explanation of the verification environment.

Chapter 6 contains a conclusion of the work.

CHAPTER 2

Background

2.1 Bluespec SystemVerilog

The design of the blocks and their testing is written in Bluespec SystemVerilog (BSV). BSV is a high level Hardware Description Language. It expresses synthesizable behavior with rules, a rule can be viewed as a declarative assertion expressing a potential atomic state transition. The BSV compiler produces efficient RTL code that manages all the potential interactions between rules by inserting appropriate arbitration and scheduling logic, logic that would otherwise have to be designed and coded manually. BSV connects the modules by interfaces and methods. It also provides predefined library elements like FIFOs, BRAMs etc. which are modeled using BSV methods.

It has powerful static type checking which removes potential human errors which can't be detected at the stage of compilation normally but can be detected now during the compilation. BSV also has more general type parameterization (polymorphism) due to which modules and functions can be parameterized by other modules and functions, this enables the designer to reuse designs and glue them together in much more flexible ways. BSV's static elaboration helps to arrive at the design much faster than the other HDLs. The BSV compiler also can generate the synthesizable Verilog code of the written bluespec code which can be used later for synthesis purposes.

BSV has an inbuilt package called TLM (Transaction Level Modeling) which was

used in implementing the 3-stage pipeline core.

2.2 Python Scripting

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. Python is a great flexible programming language. Indeed, the AAPG is based on Python language. Python programming was necessary to understand the existing AAPG code and modify it accordingly to verify the e-class processor code.

Understanding the make.py file and suitably modifying it, needed the knowledge of Bash Commands also. Bash commands were needed in order to compile and link to generate the assembly program. These bash commands using the riscv64-unknown-elf compiler and linker were incorporated in the make.py file using the subprocess module in Python.

CHAPTER 3

RISC-V ARCHITECTURE

RISC-V (pronounced "risk-five") is an open source instruction set architecture (ISA) based on established reduced instruction set computing (RISC) principles. It was originally designed to support computer architecture research and education and is now set to become a standard open architecture for industry implementations. Main goals of RISC-V include:

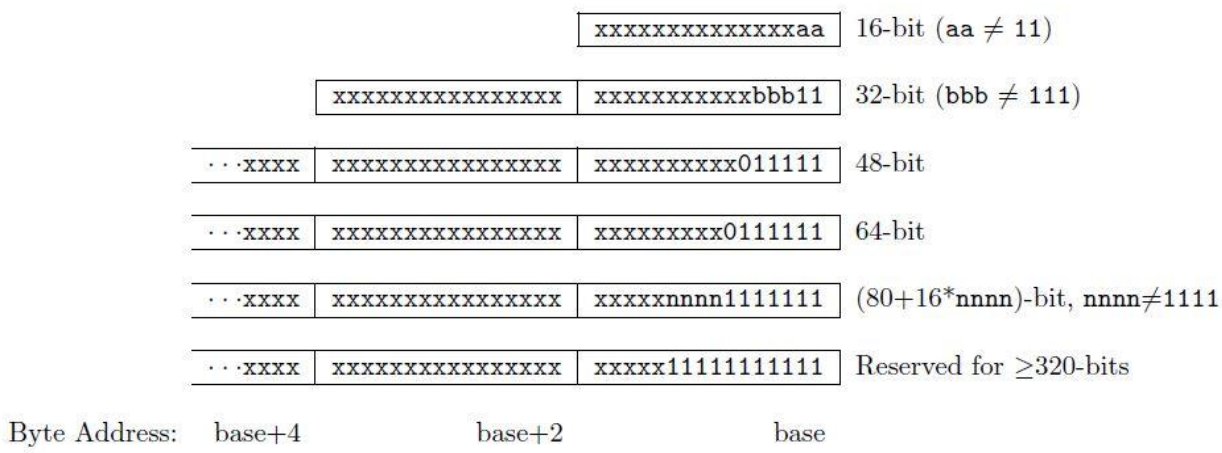
- A completely open ISA that is freely available to academia and industry.
- A real ISA suitable for direct native hardware implementation, not just simulation or binary translation.
- An ISA separated into a small base integer ISA, usable by itself as a base for customized accelerators or for educational purposes, and optional standard extensions, to support general purpose software development.
- Support for the revised 2008 IEEE-754 floating-point standard.
- Both 32-bit and 64-bit address space variants for applications, operating system kernels, and hardware implementations.
- An ISA with support for highly-parallel multicore or manycore implementations, including heterogeneous multiprocessors.
- Optional variable-length instructions to both expand available instruction encoding space and to support an optional dense instruction encoding for improved performance, static code size, and energy efficiency.

- A fully virtualizable ISA to ease hypervisor development.

3.1 RISC-V ISA Overview

The RISC-V ISA is defined as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA. Each base integer instruction set is characterized by the width of the integer registers and the corresponding size of the user address space. The base RISC-V ISA has fixed-length 32-bit instructions that must be naturally aligned on 32-bit boundaries.

Figure 3.1: RISC-V instruction length encoding



3.2 Base Instruction Format

In the base ISA, there are four core instruction formats (R/I/S/U), as shown in Figure 3.2.1. All are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory. An instruction address misaligned exception is generated if the pc is not four-byte aligned on an instruction fetch. The RISC-V ISA keeps the source (rs1 and rs2) and destination (rd) registers at the same position in all formats to simplify decoding. Immediate are packed towards the leftmost available bits in the instruction and have been allocated to reduce hardware complexity. In particular, the sign bit for all immediate is always in bit 31 of the instruction to speed sign-extension circuitry.

Figure 3.2.1: RISC-V base instruction formats.

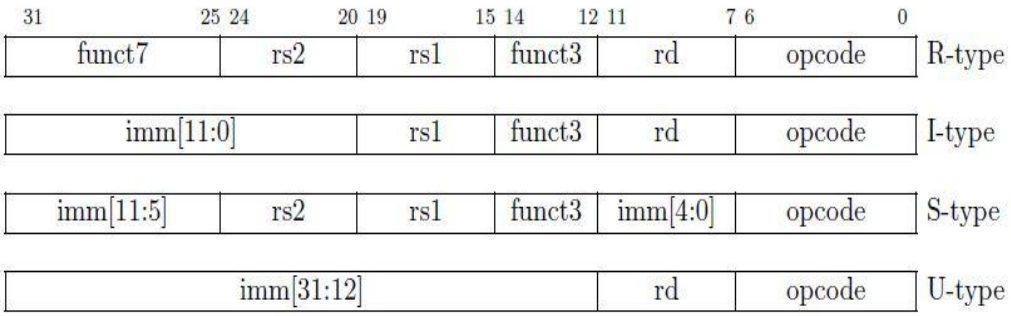
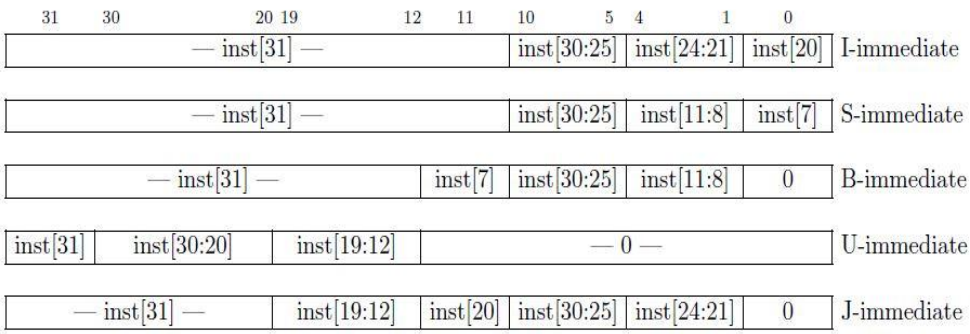


Figure 3.2.2: Types of immediate produced by RISC-V instructions. The fields are labeled with the instruction bits used to construct their value. Sign extension always uses inst[31].



CHAPTER 4

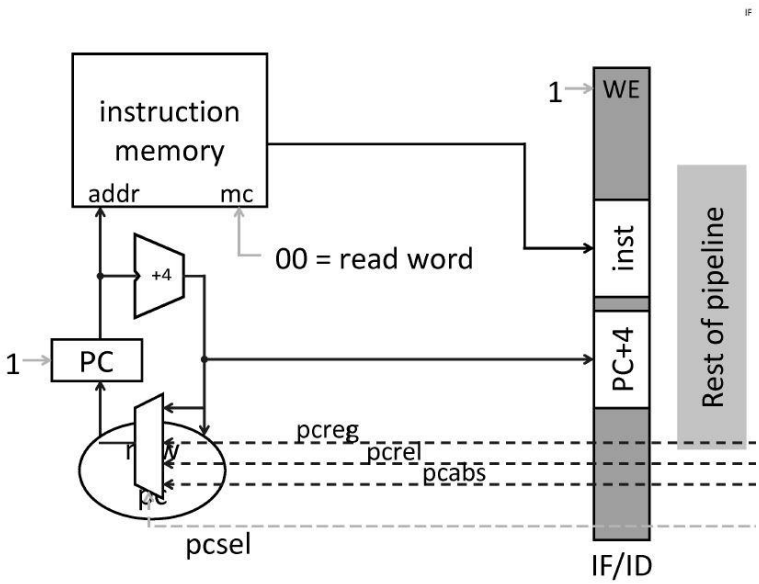
3 Stage pipeline

Pipelining is breaking down execution into multiple steps, and executing each step in parallel. It is an implementation technique in which multiple instructions are overlapped in execution. Multiple tasks operating simultaneously using different resources. Pipelining doesn't help latency of single task, it helps throughput of entire workload. Pipeline rate is limited by the slowest pipeline stage. There are three stages in E-class core pipeline. Basic 3 stage pipeline includes Fetch, Decode and Execute processes. Here we use 2 FIFOs to control the flow of data from each stage to next till the present task is executed in that cycle. The pipeline organization gives one instruction per cycle throughput.

4.1 Stages of pipeline

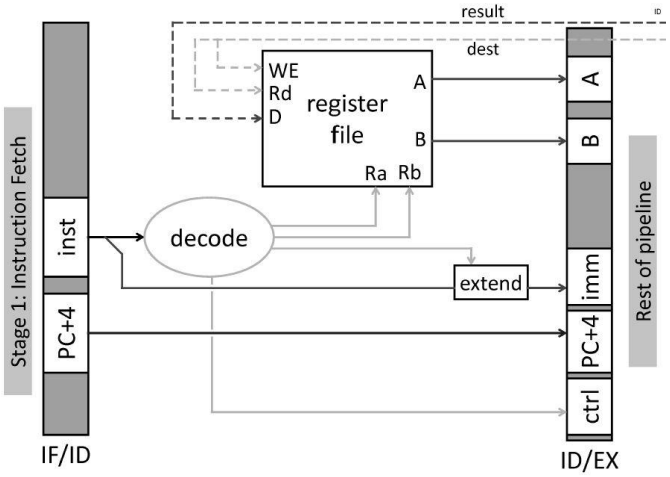
1. **Instruction Fetch (IF):**

   Index the Current Program counter (PC) to the instruction memory. Increment the PC at the end of cycle. Fetch the data from input and write the values of interest to Pipeline FIFO (IF/ID) between IF and Instruction decode stage.



2. **Instruction Decode (ID)**:

   Read from IF/ID FIFO to get instruction bits. Decode instruction, generate control signals and then read from register file. Write values of interest to next pipeline FIFO (ID/EX). Control information, Rd index, immediates, offsets, contents of Ra, Rb are sent to next stage.



3. **Execute (EX):**

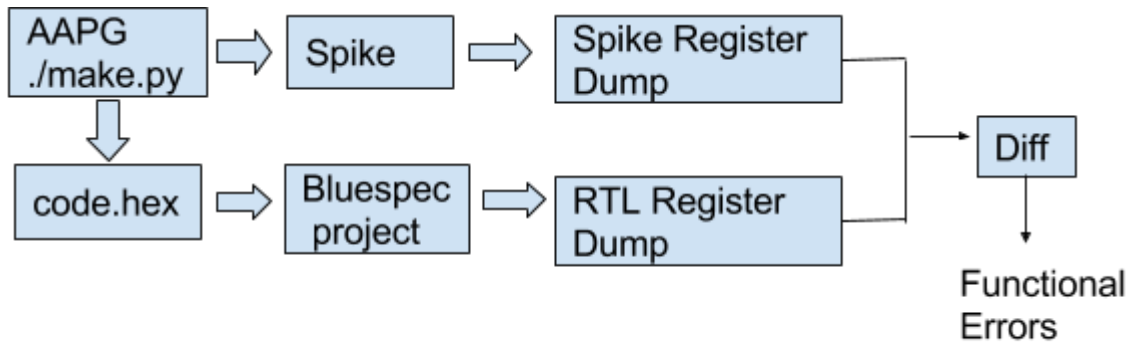
   Read ID/EX pipeline FIFO to get value and control bits and then perform ALU operations. Compute targets (PC+4+offset) n case this a branch statement. Then write back the result.

CHAPTER 5

Verification Environment

This chapter discusses about the available verification environment and tools like AAPG and Spike. AAPG stands for Automatic Assembly Program Generator generates a random and valid assembly program. The program is compiled with RISC-V toolchain and is run through spike. Spike is an RISCV ISA Simulator. Spike will dump register values after execution of every instruction. The AAPG also generates instruction and initial memory in hex format which can be run on the processor RTL. The RTL Register Dump and the spike Register Dump are compared using diff operations to check the functionality and correctness of the core.

The diff operations can be done either by modifying the AAPG make.py code or by modifying the Makefile of the bluespec project.



5.1 AAPG make.py file

The make.py file is an Python Shell Script. It requires Python 3 environment. It contains functions to compile the assembly program, link it to the object file, generate the object dump, run spike to generate Spike register dump, and to run the RTL simulation. This makefile contains the following maketypes: clean, all, generate assembly and run spike, generate assembly program only, compile assembly program only, link assembly program to object file only. One of the most used packages for writing shell scripts in Python is the subprocess package. The simplest use of this package is to use the call function to call a shell command. Hence, all the above mentioned functions have been implemented using this subprocess.call() function in Python.

The difference operation between the RTL register dump and Spike register dump is implemented in the maketype:all. The output in the terminal saying if the results match or not is done by checking the stdout, stderr of the diff operation. This is implemented by using subprocess.getoutput() function. This subprocess.getoutput(cmd) function returns the output that is stdout or stderr of executing *cmd* in a shell. So, we check if there is any output from subprocess.getoutput(). If there isn't any that implies the results match. Else if there is output from subprocess.getoutput() that implies that the results do not match and further the E-class processor code is revisited to check for the above mis-match and correct the processor code accordingly.

5.2 System Requirements

The system requirements for the make.py to successfully execute are: Python 3, the

modified riscv tools and the modified spike.

5.3 Bluespec project Makefile

The Bluespec project Makefile can be exported from the Bluespec workstation once the e-class processor project is opened in the workstation. This makefile contains compile, link, simulate, clean and full clean modules. One more module can be added naming difference. This difference module shall compare the Spike register dump and the RTL register dump and write the difference log to a log file. This log file contains the Spike and RTL register dump values next to each other. However, the register values that have not matched are separated by the character vertical bar '|'. Hence, searching for this vertical bar character can point us to the mis-match and subsequently the processor code can be revisited to correct the mis-match.

CHAPTER 6

Conclusion

This project has given me an opportunity to study and understand the functionality of the E-class 3-stage pipelined processor based on RISC V 32-bit Instruction set architecture that was realized in BSV. Integrated verification environment for the E-class processor has been set up by modifying AAPG to compare the RTL and Spike register dumps.

Bibliography

1. https://riscv.org/2016/04/risc-v-offers-simple-modular-isa/
2. Bluespec,Inc.Bluespec System Verilog User Guide
3. Bluespec,Inc.Bluespec System Verilog Reference Guide