FPGA-based RISC Microprocessor and Compiler (Rev. 1.00)

PRCO304 - Final Stage Computing Project

Ben Lancaster 10424877

March 14, 2018

Revision History

Table 1: Document revisions.

Date	Version	Changes
11/03/2018	1	Initial section outline.

Abstract

ben

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern computing and electronics equipment, like function generators, oscilloscopes, and spectrum analysers, use FPGAs to implement their compute intensive logic. These FPGAs are often accompanied by a small, low-cost, microprocessor to supervise and provide interfaces to external peripherals.

The aim of this project is to implement this side-microprocessor into the FPGA to save on BOM costs, PCB space, and power costs, which contribute to higher development and product costs. While savings can be made by the lack of side microprocessor, the product may need a larger FPGA to accommodate the embedded microprocessor. The project will produce a small, soft-core, CPU design and compiler. Although there is no direct client in this project, I believe this project will produce an attractive product for FPGA-based product designers wishing to employ an embedded processor solution.

1.1 Background

1.1.1 Current Implementations

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2 Project Overview

This project aims to provide an efficient and cost-saving alternative for board and hardware product designers utilising side-microprocessors by designing, implementing, and demonstrating, a small, portable, FPGA processor core design to be used in-place of the side-microprocessor.

The processor core will implement it's own processor and instruction set architecture and so a compiler and assembler will also be provided so that software code can easily be executed on the processor.

1.2.1 Core Deliverables

These core (C) deliverables are the base requirement for the project to be released in a functional and worthwhile state.

C1. To improve my knowledge and experience of FPGA development, processor architecture, compilers, and embedded systems engineering.

C2. To build a working and operational soft-core processor core capable of performing simple tasks.

C3. Implementation of the soft-core processor design on real hardware.

C4. To provide product designers with an affordable alternative to a side-microprocessor in their FPGA-based products.

C5. To provide a technical processor reference guide and specification for the embedded core.

1.2.2 Extended Deliverables

These extended (E) deliverables may not be achievable in the time frame specific in section 2.1 as they may require extra time to design and implement, require more experience or skill, or require resources currently unattainable.

E1. To provide embedded products a convenient solution to in-field updating.

E2. To provide easy interfacing between the FPGA design and the embedded core.

E3. GCC/LLVM/8CC compiler backend for C programming.

E4. Wishbone interface for easier modularity and inter-module communication.

E5. Multi-core design with Wishbone (2).

E6. Configurable build options (register/bus widths, optimisations/pipelining, user/privileged mode to support modern operating systems).

E7. Memory management modules to provide protected and virtual memory lookup tables.

1.3 Legal and Ethical Considerations

1.3.1 Privacy

The PRCO304 processor will be able to read and write to all data passing through it and control all connected peripherals (such as UARTs, SDRAMs, and SD Cards). The processor does not track or store usage behaviour, instructions and their frequency, memory contents, or timing statistics, or any other usage metric.

1.3.2 Fit for Purpose

The PRCO304 processor is not designed to run general purpose operating systems, such as Linux or embedded RTOS systems. All memory devices attached to the FPGA are fully accessible to the processor core and instructions/programs running through it, meaning that operating systems or secure applications storing private and sensitive information is not protected by modern processor features such as privilege modes and virtual memory sections. The processor lacks common components required to run modern operating systems, such as a memory management unit (MMU) and privilege modes, and so should not be run on the processor.

The PRCO304 processor is not designed to run in high-reliability or safety-critical environments that require established safety standards, such as the UK Defence Standard 00-56 (**?**) and IEC 61508 (**?**).

The PRCO304 processor, by design, should be used as a replacement for a simple micro-controller accompanying a main processing module.

1.3.3 Third-party Libraries

This project uses only 1 external library for the processor core's universal asynchronous receiver-transmitter (UART) module that does not depend on any other libraries. This allows me to guarantee that: the project rights are secure; and application behaviour is well-defined and predictable (no exploits introduced/injected from external libraries). The UART module does feature a large first-in-first-out (FIFO) buffer for temporary storage of in- and out- going messages. This FIFO is internal to the FPGA design and so is protected from external viewing/modification by probing the board in which the core is running on.

The compiler sub-project does not use any external library dependencies, does not record telemetry or usage statistics, and does not require an internet connection to run.

1.3.4 Generated Code

The code generated by the compiler is **not guaranteed** to:

- **Produce constant time executable code for expressions**. For example, the compiler output for an *if* statement may implicitly vary depending on it's condition expression, which may have been optimised out, constant-folded, or without-optimisation. This also applies for user code aiming to create reliable and accurate time delay loops; although the processor does not perform optimisations such as instruction caching or branch prediction, access to memory and ALU operations may vary in time, resulting in unreliable instruction times.
- **Produce code for secure-environments**. The compiler will not randomise, obfuscate, or split-up and spread, output code. Output machine code will be in a predictable format (global variables in low-memory, instruction memory in middle-memory, and stack memory in high-memory) making the binary easily subject to reverse-engineering and modification.

Chapter 2

Project Management

2.1 Time Management

2.2 Version Control

Version control will be utilised to improve work-flow, reference and review code changes, and protect the project from data loss and corruption. GitHub, a git hosting provider, will be utilised to host all project files, including documentation and design files.

The repository can be found here: https://github.com/bendl/prco304.

2.3 Method of Approach

Development of the **core** and **compiler** will be done in separate stages of the project (see section 2.1). The two deliverables will be split into 2 sub-projects. Both sub-projects will employ the **Agile development process**, using Agile's sprints to split up tasks into sub-tasks and Agile's scrums to discuss progress, features, and changes. This technique allows revisiting of tasks to tweak and iterate over their implementation which will be key when for incrementally adding features to both sub-projects, for example, adding to the core's ALU module to add conditional branching, or adding new instructions to the core's decoder module.

2.4 Requirements

2.5 Resources and Dependencies

For the first half of the development cycle, the core can be developed and verified using the Verilog simulator and test suite, **Verilator**, and VHDL and Verilog simulator, **iSim**.

The second half of development will require deploying and debugging on real hardware. This will require an FPGA development kit. To better emulate customer products, the development kit should feature common components such as LEDs, GPIO, USB interface, flash-based storage and memory, and optionally an analogue audio output port. The low-middle range of FPGA devices the project is targeting is the popular and affordable yet feature rich Spartan-6 and Artix-7 FPGAs. From my placement, I have gained experience in Xilinx FPGAs and so will be targeting them for this project to reduce risk and development time.

The following FPGA development kits are suitable for this project:

1. MiniSpartan6+ - Scarab Hardware - *$*79 (already owned) (**?**). The MiniSpartan6+ features a Spartan-6 XC6SLX9 FPGA, 8 LEDs, 2 digital and analogue headers, FT2232 FTDI USB to JTAG, 64Mb SPI flash memory, 32MB SDRAM, an audio output jack, and a MicroSD socket.

2. Arty Artix-7 FPGA Development Board - Digilent - *$*100 (**?**). The Arty development board features a larger Artix-35T FPGA with over 20x the number of logic cells and block memory compared to the LX9 in the MiniSpartan6+. The board components include 256MB DDR3 RAM, 16MBx4 SPI flash memory, USB-JTAG, 8 LEDs (4 of which are RGB), 4 switches, 4 buttons, and multiple Pmod connectors.

   The greater number of IO options and larger FPGA make the Arty board better suited to emulating real customer products.

The project will require a computer or laptop to develop the core and compiler on and continuous integration systems to perform testing on the incremental builds. For the project demo, an oscilloscope (already owned) or digital logic analyser may be required to demonstrate some of the core's features.

Chapter 3

PRCO304 Processor Design

3.1 High Level Design

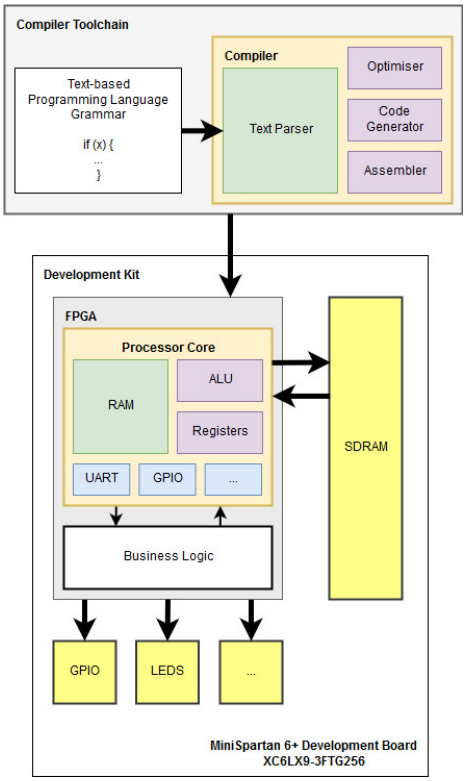

Figure 3.1: test

3.2 Registers

3.3 Pipeline Architecture

3.4 Testing and Verification

Chapter 4

PRCO304 Compiler

Chapter 5

Conclusion

5.1 Project Post-mortem

Chapter 6

Appendices

6.1 Appendix A. PRCO304 Core Reference Guide

6.2 Appendix B. PRCO304 Compiler Reference Guide

6.3 Appendix C. Project Initiation Document