Multi-core RISC Processor Design and Implementation (Rev. 1.00)

ELEC5881M - Interim Report

Ben David Lancaster
Student ID: 201280376

Submitted in accordance with the requirements for the degree of
Master of Science (MSc)
in Embedded Systems Engineering

Supervisor: Dr. David Cowell
Assessor: Mr David Moore

University of Leeds
School of Electrical and Electronic Engineering

April 11, 2019

Revision History

Date	Version	Changes
20/05/2018	3.14	Add background research to appendix.
19/05/2018	3.13	Update abstract to align with guidelines.
19/05/2018	3.12	Fix ISA pseudo-codes.
11/03/2018	1.00	Initial section outline.

Table 1: Document revisions.

Abstract

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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Acknowledgements

I would like to thank my project supervisors Nigel Barlow and Serge Thill for their support and guidance throughout this project.

I would also like to thank James Spalding (Spirent Communications) and firmware team for their encouragement, ideas, and industrial sponsorship supporting this final project.

Declaration of Academic Integrity

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated in the report. The candidate confirms that appropriate credit has been given within the report where reference has been made to the work of others.

This copy has been supplied on the understanding that no quotation from the report may be published without proper acknowledgement. The candidate, however, confirms his/her consent to the University of Leeds copying and distributing all or part of this work in any forms and using third parties, who might be outside the University, to monitor breaches of regulations, to verify whether this work contains plagiarised material, and for quality assurance purposes.

The candidate confirms that the details of any mitigating circumstances have been submitted to the Student Support Office at the School of Electronic and Electrical Engineering, at the University of Leeds.

Name: Ben David Lancaster
Date: April 11, 2019

Table of Contents

Chapter 1

Abstract (not in toc)

Chapter 2

Declaration of Academic Integrity (not in toc)

Chapter 3

Acknowledgements (not in toc)

Chapter 4

Introduction

4.1 Why Multi-core?

4.2 Why RISC?

4.3 Why FPGA?

Chapter 5

Background

5.1 Single core vs. Multi-core vs. Many-core

5.2 Network-on-chip

5.2.1 OpenPiton

5.3 Summary

Chapter 6

Project Overview

6.1 Project Deliverables

The project's deliverables are split into two sections: core deliverables (CD) – each deliverable must be satisfied for the project to be a minimum viable product (MVP), and extended deliverables (ED) – deliverables that are not required for an MVP, features that only improve upon an existing feature.

6.1.1 Core Deliverables (CD)

The project's core deliverables are described below.

CD1. Design a compact 16-bit RISC instruction set architecture.

CD2. Design a synthesisable Verilog RISC core that implements the ISA in **CD1.**.

CD3. Design and implement an on-chip interconnect for multi-core processing (2 to 32 cores) using the RISC core from **CD2.**.

CD4. Analyse performance of serial and parallel software algorithms, such as parallel DFT [**?**], on the processor.

6.1.2 Extended Deliverables (ED)

The project's extended deliverables are described below.

ED1. Design a RISC core with an instructions-per-clock (IPC) rating of at least 1.0 (a single-cycle CPU).

ED2. Design a RISC core with a pipe-lined data path to increase the design's clock speed.

ED3. Design a scalable multi-core interconnect supporting arbitrary (more than 32) RISC core instances (manycore) using Network-on-Chip (NoC) architecture.

ED4. Design a compiler-backend for the PRCO304 [**?**] compiler to support the ISA from1 **CD1.**. This will make it easier to build complex multi-core software for the processor.

ED5. The RISC core can communicate to peripherals via a memory-mapped addresses using the Wishbone [**?**] bus.

ED6. Implement various memory-mapped peripherals such as UART, GPIO, LCD, to aid visual representation of the processor during the demonstration viva.

ED7. Store instruction memory in SPI flash.

ED8. Reprogram instruction memory at runtime from host computer.

ED9. Processor external debugger using host-processor link.

6.2 Project Timeline

6.2.1 Project Stages

The project is split up into many stages to aid planning and management of the project. There are 8 unique stage areas: 1. Inital project conception; 2 Basic RISC core development; 3. Extended RISC core development; 4. Multi-core development; 5. Processor quality-of-life (QoL) improvements; 6. Compiler development; 7. Demo preparation, and 8. Final report.

The project stages are shown in Table 6.1.

6.2.2 Timeline

The project stages from Table 6.1 are displayed below in a Gantt chart.

Stage	Title	Start Date	Days	Core	Applicable Deliverables
1.0	Research	Feb 04	7	x	
1.1	Requirement gathering/review	Feb 11	14	x	
1.1	Processor specification, architecture, ISA	Feb 18	100	x	**CD1.**
1.2	Stage/Time Allocation Planning	Feb 25	7	x	
2.1	Decoder, Register Set, impl & integration	Feb 25	14	x	**CD2.**
2.2	Register set impl & integration	Mar 04	14	x	**CD2.**
2.3	Local memory impl & integration	Mar 11	14	x	**CD2.**
3.1	Memory mapped register layout & impl	Apr 01	21		**ED5.**
3.2	Wishbone peripheral bus connected to MMU	Apr 08	21		**ED5.**
3.3	Pipelined implementation and verification	Apr 15	21		**ED2.**
3.4	Cache memory design & impl	Apr 22	28		**ED2.**
4.1	Multi-core communication interface	TBD	TBD	x	**CD3.**
4.2	Shared-memory controller	TBD	TBD	x	**CD3.**
4.3	Scalable multi-core interface (10s of cores)	TBD	TBD	x	**CD3.**
4.4	Multi-core example program (reduction)	TBD	TBD	x	**CD4.**
5.1	SPI-FPGA interface for OTG programming	TBD	TBD		**ED7.**
5.2	FPGA-PC interfacing	TBD	TBD		**ED9.**
5.3	FPGA-PC debugging (instruction breakpoints)	TBD	TBD		**ED9.**
6.1	Compiler backend for vmicro16	TBD	TBD		**ED4.**
6.2	Compiler support for multi-core codegen	TBD	TBD		**ED4.**
7.1	Wishbone peripherals for demo	TBD	TBD	x	**CD4.**
8.1	Final Report	TBD	TBD	x	

Table 6.1: Project stages throughout the life cycle of the project.

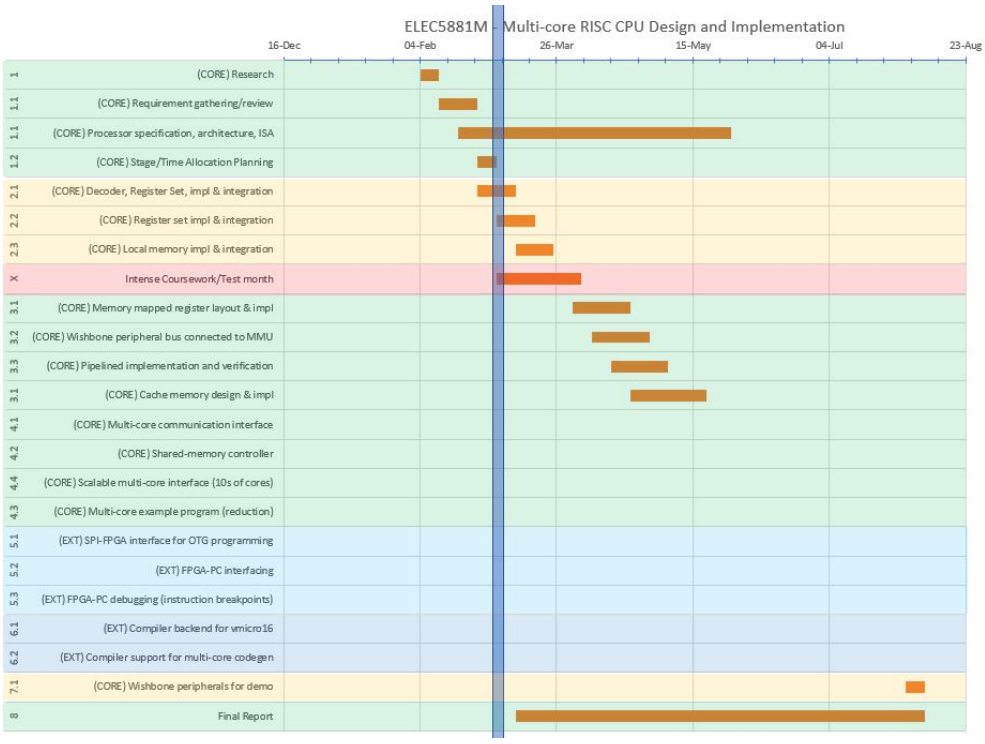


Figure 6.1: Project stages in a Gantt chart.

6.3 Resources

6.3.1 Terasic DE1-SoC Development Board

6.3.2 Minispartan 6+ FPGA Development Board

Chapter 7

Current Progress

7.1 RISC Core

7.1.1 Instruction Set Architecture

The goals of the ISA are to:

ISA1. Use a fixed width of 16-bits for all instructions.
This will significantly reduce RTL resources and encourage efficiency by not wasting spare bits. In addition, many SPI flash and RAMs support 16-bit wide data reads which will allow each instruction fetch to only require one clock cycle, thus increasing processor performance.

ISA2. Be able to select at least two registers for common instructions.
This will reduce the number of required instructions to manipulate register data. A disadvantage of using two instead of three reigster selects is that instructions are always destructive – they always *destroy* existing data in the destination register (e.g. R0 = ADD R0 R1) unlike constructive instructions that provide a dedicated register select for the destination (e.g. R2 = ADD R0 R1).

ISA3. Reduce bit-space for frequently used instructions (MOV, MOVI, ADD).
Due to the 16-bit limit, two register selects, and immediate values, the opcode bits are reduced resulting in fewer unique instructions. To overcome this constraint, spare bits in other instructions will be appended to the opcode bits to extend the opcode range. This however, will require a more complex decoder that must first switch the opcode, then switch any spare bits to determine the final opcode. This method will significantly increase the number of unique instructions provided by the instruction set.

ISA4. Provide frequently used actions as options for existing instructions.
In software, frequently used actions include incrementing/decrementing by 1 and

performing logical comparisons which usually take more than one instruction on some RISC architectures. As they are common actions, the instruction overhead and time may be significant and can affect performance. To provide a solution to this problem, in addition to using spare bits to extend the opcode range, spare bits will be used to signify a frequently used action action to be performed by the ALU.

As shown in Figure 7.1, frequently used commands such as incrementing/decrementing and logical comparions are provided by setting spare bits to special values. For example, the instructions `ARITH_UADDI` and `ARITH_SSUBI` extend the `ARITH_U` and `ARITH_S` opcodes by filling the spare bit, 4. If this bit is not set (0), the instruction allows for a 4-bit immediate value to be added in addition to the two register selects. The 4-bit immediate allows adding a small number to the ALU which is useful in the case of software for loops where an increment/decrement of more than 1 is required.

Another example is the `SETC` instruction. Inspired by Intel's x86 `SETCC`, the instructions sets the destination register to zero or one depending on the result of the `CMP` instruction's flags. Without this instruction, multiple branches would be required to convert the comparion's flags to logical zeros and ones.

ISA5. Provide instructions for performing bitwise manipulations.
RISC processors are commonly used for microprocessing and microcontroller actions which typically includes bit manipulation. The ISA provides bitwise OR, XOR, AND, NOT, and shifting instructions under a single opcode to fill this need.

ISA6. Provide instructions for explicitly performing signed and unsigned arithmetic.

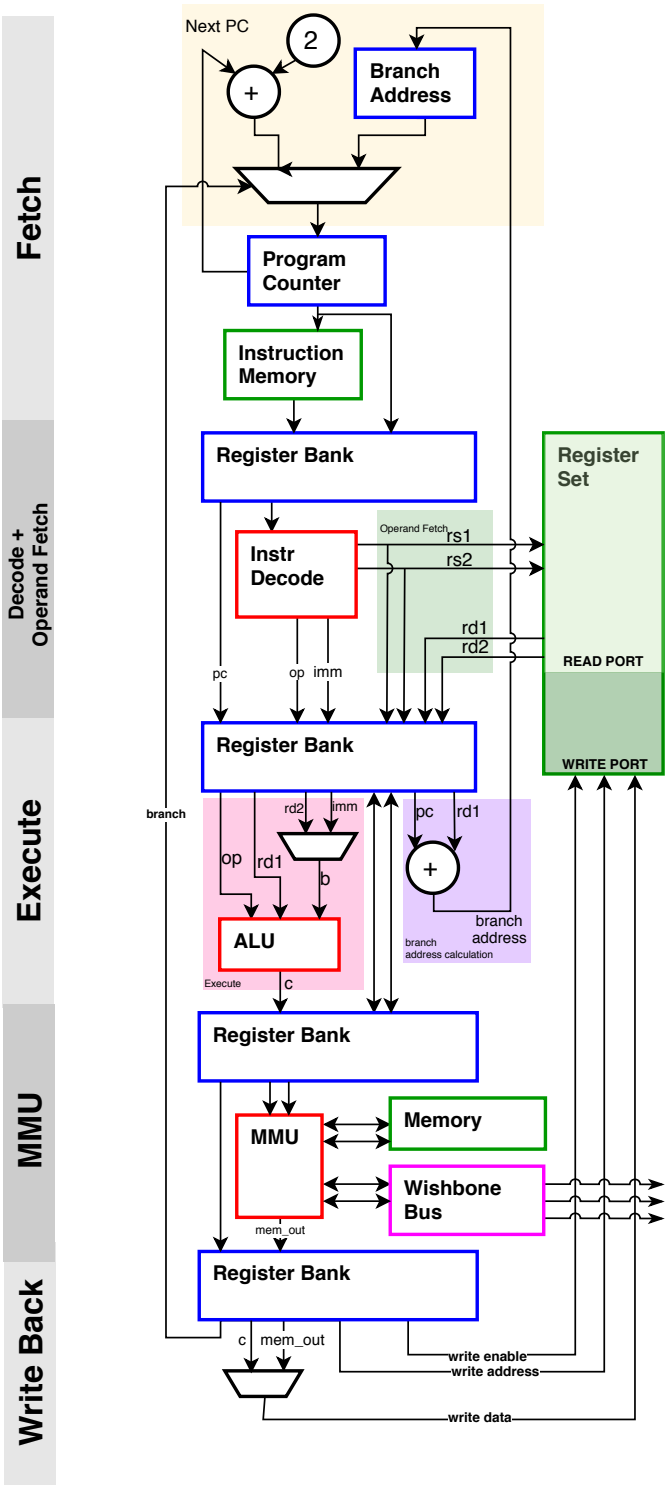
7.1.2 Design

7.1.3 Implementation

7.1.4 Verification

	15-11	10-8	7-5	4-0	rd ra simm5
	15-11	10-8	7-0		rd imm8
	15-11	10-0			nop
	15	14:12	11:0		extended immediate
NOP	00000	X			
LW	00001	Rd	Ra	s5	Rd <= RAM[Ra+s5]
SW	00010	Rd	Ra	s5	RAM[Ra+s5] <= Rd
BIT	**00011**	**Rd**	**Ra**	**s5**	**bitwise operations**
BIT_OR	00011	Rd	Ra	00000	Rd <= Rd | Ra
BIT_XOR	00011	Rd	Ra	00001	Rd <= Rd ^ Ra
BIT_AND	00011	Rd	Ra	00010	Rd <= Rd & Ra
BIT_NOT	00011	Rd	Ra	00011	Rd <= ~Ra
BIT_LSHFT	00011	Rd	Ra	00100	Rd <= Rd << Ra
BIT_RSHFT	00011	Rd	Ra	00101	Rd <= Rd >> Ra
MOV	00100	Rd	Ra	X	Rd <= Ra
MOVI	00101	Rd	i8		Rd <= i8
ARITH_U	**00110**	**Rd**	**Ra**	**s5**	**unsigned arithmetic**
ARITH_UADD	00110	Rd	Ra	11111	Rd <= uRd + uRa
ARITH_USUB	00110	Rd	Ra	10000	Rd <= uRd - uRa
ARITH_UADDI	00110	Rd	Ra	0AAAA	Rd <= uRd + Ra + AAAA
ARITH_S	**00111**	**Rd**	**Ra**	**s5**	**signed arithmetic**
ARITH_SADD	00111	Rd	Ra	11111	Rd <= sRd + sRa
ARITH_SSUB	00111	Rd	Ra	10000	Rd <= sRd - sRa
ARITH_SSUBI	00111	Rd	Ra	0AAAA	Rd <= sRd - sRa + AAAA
BR	**01000**	**Rd**	**i8**		**conditional branch**
BR_U	01000	Rd	0000 0000		Any
BR_E	01000	Rd	0000 0001		Z=1
BR_NE	01000	Rd	0000 0010		Z=0
BR_G	01000	Rd	0000 0011		Z=0 and S=O
BR_GE	01000	Rd	0000 0100		S=O
BR_L	01000	Rd	0000 0101		S != O
BR_LE	01000	Rd	0000 0110		Z=1 or (S != O)
BR_S	01000	Rd	0000 0111		S=1
BR_NS	01000	Rd	0000 1000		S=0
CMP	01001	Rd	Ra	X	SZO <= CMP(Rd, Ra)
SETC	01010	Rd	Ra	X	Rd <= Imm8 == SZO ? 1 : 0
MOVI_LARGE	1	Rd	i12		Rd <= i12

Figure 7.1: Vmicro16


Fetch
Decode + Operand Fetch
Execute
MMU
Write Back
Next PC
2
+
Branch Address
Program Counter
Instruction Memory
Register Bank
Register Set
Operand Fetch
Instr Decode
rs1
rs2
rd1
rd2
READ PORT
WRITE PORT
pc
op
imm
Register Bank
branch
rd2
imm
pc
rd1
op
rd1
b
+
branch address
branch address calculation
ALU
Execute
c
Register Bank
MMU
Memory
Wishbone Bus
mem_out
Register Bank
c
mem_out
write enable
write address
write data

Chapter 8

Future Progress

8.1 Multi-core Functionality

8.2 Interconnect Goals

8.3 Interconnect Layout

8.3.1 Bus Design

8.3.2 Core Pin Assignments

8.4 Core-to-core Communication

8.5 Shared-Resource Control

8.5.1 Resource Scheduling

8.6 Verification

8.7 Conclusion

Chapter 9

RISC Core Design

9.1 Design Goals

9.2 Instruction Set Architecture

9.2.1 Data Sizes

9.2.2 Registers

9.2.3 Endianness

9.3 High Level Design

9.4 Memory-mapped peripherals

9.4.1 Wishbone Master Bus

9.5 Optimisations

9.5.1 Pipelining

9.5.2 Stall Avoidance

9.6 Core Verification

9.7 Conclusion

Chapter 10

Multi-core Communication Design

10.1 Interconnect Goals

10.2 Interconnect Layout

10.2.1 Bus Design

10.2.2 Core Pin Assignments

10.3 Core-to-core Communication

10.4 Shared-Resource Control

10.4.1 Resource Scheduling

10.5 Verification

10.6 Conclusion

Chapter 11

Core Analysis

11.1 FPGA Implementation Analysis

11.1.1 Space/Resource Usage

11.1.2 Static Timing Analysis

11.2 Speed Analysis

11.2.1 Parallel Reduction Algorithms

11.2.2 Parallel DFT Algorithm

Chapter 12

Conclusion

12.1 RISC Core Review

12.2 Multi-core Review

12.3 Verification Review

12.4 Goal Review