# **Multi-core RISC Processor Design & Implementation**

**Demonstration Viva** 

Ben Lancaster

201280376 ELEC5881M - Main Project

July 9, 2019

## **Quick Links**

#### Main Project

B. Lancaster

Introduction

Top Level Design

Multi-core Functionalit

Dogulto

- GitHub repository: https://github.com/bendl/vmicro16
- Full Report: https://github.com/bendl/vmicro16/blob/master/docs/reports/build/ELEC5881M\_Ben\_Lancaster\_201280376\_Final.pdf

B. Lancaster

Introductio

Top Level Design

Multi-core Functionalit

Results

Conclusio

1 Introduction Why Multi-core? Why RISC?

- 2 Top Level Design
  Overview
  Memory Map
  Interconnect
  Interrupts
- Multi-core Functionality HW/SW Requirements Context Identification Atomics

- Pesults
  Results 1
- 5 Conclusion
  Accomplishments
  Future Improvements
  Q&A

#### B. Lancaster

# Introduction Why Multi-core?

Why RISC?

Design

Multi-core Functional

Results

nesuits

. . .

1 Introduction Why Multi-core? Why RISC?

2 Top Level Design

Multi-core Functionality

4 Results

# Why Multi-core?

Main Project

B. Lancaster

Why Multi-core?

Why RISC?

Multi-core

Regulte

riesuits

Block Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

# Why RISC?

#### Main Project

B. Lancaster

Introduction
Why Multi-core?
Why RISC?

Top Level Design

Multi-core Functionality

Results

B. Lancaster

1 Introduction

Introduction

# Top Level

#### Design Overview

Overview Memory Map

nterconnect

Multi-core

Functionality

Results

Conclusion

2 Top Level Design

Overview

Memory Map

Interconnect

Interrupts

3 Multi-core Functionality

4 Results



### Main Project

B. Lancaster

Introductio

Top Level Design Overview Memory Map Interconnect

Interrupts
Multi-core

Functiona

Conclusion

## What this project produces:

System-on-Chip with multi-processor functionality
 Tested on FPGA hardware with 1-96 CPU cores.

### Main Project

B. Lancaster

Introductio

Design Overview Memory Map Interconnect

Interrupts

Multi-core

Functionalit

Results

Canalysian

## What this project produces:

- System-on-Chip with multi-processor functionality
   Tested on FPGA hardware with 1-96 CPU cores.
- Custom 16-bit RISC CPU
   With interrupts and its own Instruction Set Architecture (ISA).

### Main Project

B. Lancaster

Introductio

Design Overview Memory Map Interconnect

Multi-core Functionalit

Results

Conclusion

## What this project produces:

- System-on-Chip with multi-processor functionality
   Tested on FPGA hardware with 1-96 CPU cores.
- Custom 16-bit RISC CPU
   With interrupts and its own Instruction Set Architecture (ISA).
- Software/Assembly compiler
   PRCO304 programming language/Intel assembly syntax.

### Main Project

B. Lancaster

Introductio

Design
Overview
Memory Map
Interconnect

Multi-core

Functional

nesuits

Conclusion

## What this project produces:

- System-on-Chip with multi-processor functionality
   Tested on FPGA hardware with 1-96 CPU cores.
- Custom 16-bit RISC CPU
   With interrupts and its own Instruction Set Architecture (ISA).
- Software/Assembly compiler
   PRCO304 programming language/Intel assembly syntax.
- Aimed at Design Engineers, not end users
   Project is provided as source code/design files for Design Engineers to customise and implement in hardware themselves.

# **Top Level Hierarchy**

#### Main Project

B. Lancaster

Introductio

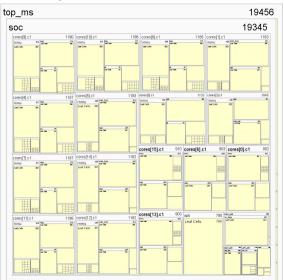
Top Level

## Overview

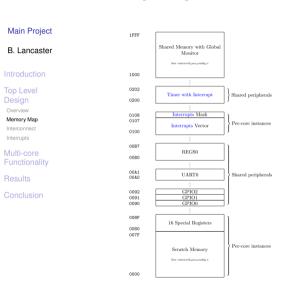
Memory Map Interconnect Interrupts

Multi-core

Results



# **Memory Map**



- Shared Memory with Global Monitor
- Timer with Interrupt
- Per-core Interrupt Vector and Mask
- Shared Register Set
- UART Transceiver
- Multiple GPIO ports
- Per-core scratch memory
- Per-core Special Registers
- Customisable by designers

# Interconnect

#### Main Project

B. Lancaster

Introduction

Top Level Design Overview

Memory Map

Interconnect Interrupts

Multi-core

Regulte

# **Interrupts**

#### Main Project

B. Lancaster

Introduction

Ton Lovel

Design Overview

Memory Map

Interconne

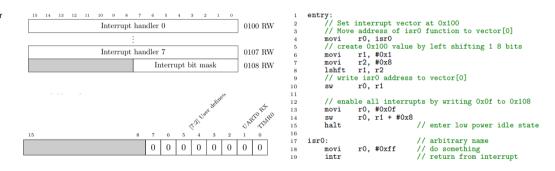
Interrupts

Multi-core

Functional

Results

Conclusion



Demo: 2 Core LED toggle (GPIO0) with TIMR0 1s interrupt (interrupts\_2.s)

# **Timer Interrupt Example**

				2,870.000 ns	3,870.000 ns	
Main Project	Name	Value	2,500 ns	3,000 ns 3,500 ns	4,000 ns 4,500 ns	5,000 ns   5,500 ns
B. Lancaster	> " r_pc[15:0] > " r_instr[15:0]	0010 4000	0010	\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0	\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0	\(\begin{array}{ccccc} \O & \O
ntroduction	> 16 regs[0:7][15:0] > 16 COALU	0010,0100,0008,0				0010,0100,0008,0000,0000,0000
Top Level	✓ <u>I</u> IINT					
Design	> W ints[7:0]	01	00	CO	co	<b>X</b> CO
Overview	> W ints_vector[127:0]	00000000000000				000000000000000000000000000000000000000
Memory Map	> W ints_mask[7:0]	Of				0f
Interconnect	¼ w_intr	0				
Interrupts	M has_int	1				
Aulti-core	18 int_pending	0				
unctionality	int_pending_ack	0				
Results	V regs_use_int > MMU	0				
Conclusion	> <u>™</u> C1					
Jonolasion	✓ 📉 TIMRO					
	M out	1				
	₩ S_PSELx	0				

Figure: TIMR0 1us interrupt with context switching

# **Timer Peripheral Registers**

### Main Project

B. Lancaster

Introduction

Top Leve Design Overview Memory Map

Memory Ma Interconnect

Interrupts

interrupts

Functionalit

Results

Conclusi

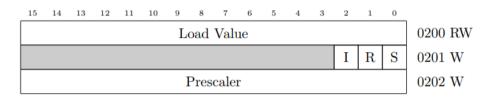


Figure: t = 20 ns \* load \* prescaler

Resolution (32-bit timer): 20ns to 85s.

## Examples:

- For 1us: Load = 0x32, Prescaler = 0 (20ns \* 0x32 = 1000ns)
- For 1s: Load = 0x1000, Prescaler = 0x3000 (demo)
   (20ns \* 0x1000 \* 0x3000 = approx. 1s)



B. Lancaster

1 Introduction

Introduction

Top Leve Design 2 Top Level Design

Multi-core Functionality

Requirements
Context Identification
Atomics

Results

Conclusion

3 Multi-core Functionality

HW/SW Requirements
Context Identification
Atomics

- 4 Results
- 5 Conclusion

Main Project

B. Lancaster

Introduction

Top Leve Design

Multi-core

HW/SW Requirements Context Identification Atomics

Results

Conclusion

Hardware:

Bus Arbitration

(scheduling: priority, rotating, etc.)

Main Project

B. Lancaster

Introductio

Top Leve Design

Functionalit

Requirements

Context Identification

Atomics

Results

Conclusion

## Hardware:

- Bus Arbitration (scheduling: priority, rotating, etc.)
- Atomic functions
   (atomic versions of load/store to prevent race conditions)

Main Project

B. Lancaster

Introduction

Top Leve Design

Functionalit

Requirements

Context Identification

Atomics

Results

Conclusion

### Hardware:

- Bus Arbitration (scheduling: priority, rotating, etc.)
- Atomic functions
   (atomic versions of load/store to prevent race conditions)
- Per-core instruction memory

#### Main Project

B. Lancaster

Introductio

Top Leve Design

Functionali

Requirements
Context Identification

Results

Conclusio

### Hardware:

- Bus Arbitration (scheduling: priority, rotating, etc.)
- Atomic functions
   (atomic versions of load/store to prevent race conditions)
- Per-core instruction memory
- Per-core context-switching for interrupt handling

### Main Project

B. Lancaster

Introductio

Top Leve Design

Functionalit

Requirements

Context Identification

Results

Conclusio

### Hardware:

- Bus Arbitration (scheduling: priority, rotating, etc.)
- Atomic functions
   (atomic versions of load/store to prevent race conditions)
- Per-core instruction memory
- Per-core context-switching for interrupt handling

### Software:

Semaphores/Mutexes
 (exclusive memory access)

#### Main Project

B. Lancaster

Introduction

Top Leve Design

Functionalit

Requirements
Context Identification

Results

Conclusio

### Hardware:

- Bus Arbitration (scheduling: priority, rotating, etc.)
- Atomic functions
   (atomic versions of load/store to prevent race conditions)
- Per-core instruction memory
- Per-core context-switching for interrupt handling

- Semaphores/Mutexes
   (exclusive memory access)
- Thread synchronisation (memory barriers)

#### Main Project

B. Lancaster

Introductio

Top Leve Design

Multi-core Functionality

Requirements
Context Identification

Results

Conclusio

### Hardware:

- Bus Arbitration (scheduling: priority, rotating, etc.)
- Atomic functions
   (atomic versions of load/store to prevent race conditions)
- Per-core instruction memory
- Per-core context-switching for interrupt handling

- Semaphores/Mutexes
   (exclusive memory access)
- Thread synchronisation (memory barriers)
- Context identification
   What core am I?
   How many cores?
   How much memory?

## **Context Identification**

#### Main Project

B. Lancaster

Introduction

Top Leve Design

Multi-core Functionality

HW/SW Requirements Context Identification

Atomics Results

Conclusio

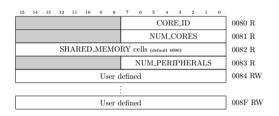


Figure: Special Registers 0x0080 to 0x008F

```
entry:
    // get core idx 0x80 in r7
            r7. #0x80
    movi
    lw
            r7, r7
       Branch away if not core 0
            r7, r0
    cmp
    movi
            r0. exit
            r0 . BR_NE
    br
       Core 0 only instructions
            r0 . r0
    qon
            r0, r0
    qon
            r0, r0
    gon
exit:
    halt
            r0 . r0
```

## **Atomic Instructions**

#### Main Project

B. Lancaster

Introduction

Top Leve Design

Functionality
HW/SW
Requirements
Context Identification

Atomics Results

Conclusion

- Enables semaphores, mutexes, memory barriers
- Prevent race conditions between threads/cores
- LW[EX] and SW[EX]

## Example:

```
try_inc:
    lwex
            r0 , r1
    // increment by 1
    addi
            r0. #0x01
       attempt store
            r0 . r1
    swex
       check success (== 0)
            r0 . r3
   cmp
       branch if failed
    movi
            r4, try_inc
            r4. BR NE
    hr
```

## **Exclusive Access**

Main Project

B. Lancaster

Introduction

Top Level Design

Multi-core

HW/SW Requirements

Context Identifica

. .

Conclusion

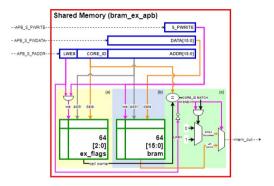


Figure: "Global Monitor" = track exclusive access to memory locations

Demo: 8 core number summation (sum.s)



#### B. Lancaster

Introduction

Top Level Design

Multi-core Functionality

Results 1

. . .

1 Intro

1 Introduction

2 Top Level Design

3 Multi-core Functionality

4 Results Results 1

# Multi-core vs Single-Core for Summation

#### Main Project

B. Lancaster

Results Regulte 1

Fach core has low work load.

- Sum subset of numbers in for
- loop
- Ideal scenario for parallelism
  - Highly parallelisable
  - Few inter-thread dependencies

Insert graph showing core count vs total time

# **Results 1**

Main Project

B. Lancaster

Introduction

Top Level Design

Multi-core Functionality

Results
Results 1

B. Lancaster

Introduction

Introduction

Top Leve Design 2 Top Level Design

Multi-core Functionality

unotional

Result

#### Conclusion

Accomplishments Future Improvements Q&A 3 Multi-core Functionality

4 Results

5 Conclusion
Accomplishments
Future Improvements
Q&A

# **Accomplishments**

Main Project

B. Lancaster

Introductio

Top Leve Design

Multi-core Functionalit

Result

Assemblehmen

Accomplishments
Future Improvement

 Near complete System-on-Chip design with various peripherals Timers, GPIO, UART, Registers, Memory

- Common multi-thread/core synchronisation primitives
   Semaphores, Mutexes, Memory Barriers, Atomic Instructions
- AMBA APB bus interface with Global Monitor Timers, GPIO, UART, Registers, Memory
- Working shared bus arbitration
   Schedules access to shared resources
- Working FPGA implementation for a 96 core design Nearly fills Cyclone V FPGA on the DE1-SoC
- Interrupts with hardware context-switching Low latency to react to interrupt
- Acknowledges design limitations and attempts to overcome LUT resources, block memories, power and temperature requirements

# **Future Improvements**

#### Main Project

B. Lancaster

Introductio

Top Leve Design

Multi-core Functionalit

Dogulto

Accomplishments
Future Improvements

Working Global Reset

Global resets are expensive (LUT resources) Resetting block memories is not trivial

- On-chip Programming
   Use the UART0 receiver to program each cores flash memory
- Per-core gating/enabling
   Improve power efficiency for ASIC implementation by disabling cores at run-time via software.
- Improve memory bottleneck
   Each core requires it's own memory reduce by multiplexing access to a single large memory.

B. Lancaster

Introduction

Top Level Design

Multi-core Functionality

Results

Conclusion

Accomplishments
Future Improvements

Q&A

Q&A