Multi-core RISC SoC Design & Implementation

Demonstration Viva

Ben Lancaster

201280376
ELEC5881M - Main Project

July 22, 2019

Quick Links

- 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`

- This presentation: `https://github.com/bendl/vmicro16/blob/master/docs/reports/build/ELEC5881M_Ben_Lancaster_201280376_viva.pdf`

- About me: `https://bendl.me/`

Main Project

B. Lancaster

Introduction
Top Level
Design
Multi-core
Functionality
Results
Conclusion

Main Project

B. Lancaster

Introduction
Why Multi-core?
Why RISC?

Top Level
Design

Multi-core
Functionality

Results

Conclusion

1 Introduction
Why Multi-core?
Why RISC?

2 Top Level Design

3 Multi-core Functionality

4 Results

5 Conclusion

Why a project on CPUs?

- **CPUs will be used for the rest of humanity**
  1000s of years

- **Understand constrains and considerations**
  1000s of years

- **Prepare myself for future employment/work**

Why Multi-core?

- **Rate of single-core speed improvements slowing**
  Pipelining, register renaming, branch predictions, clock speeds

- **Future of computing = parallel**

  - Identifying parallel opportunities

  - Massively parallel (NoC's)

  - Higher throughput

Why RISC?

- FPGA size limitations

- Easier design & impl

- Smaller = fit more cores on a chip

- Previous experience + future work

- RISC purist

Main Project

B. Lancaster

Introduction

Top Level
Design
Overview
Memory Map
Interconnect
Interrupts

Multi-core
Functionality

Results

Conclusion

1 Introduction

2 Top Level Design
Overview
Memory Map
Interconnect
Interrupts

3 Multi-core Functionality

4 Results

5 Conclusion

Overview

What this project produces:

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

Overview

Main Project

B. Lancaster

Introduction

Top Level
Design
Overview
Memory Map
Interconnect
Interrupts

Multi-core
Functionality

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).

Overview

Main Project

B. Lancaster

Introduction

Top Level
Design
Overview
Memory Map
Interconnect
Interrupts

Multi-core
Functionality

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.

Overview

Main Project

B. Lancaster

Introduction

Top Level
Design
Overview
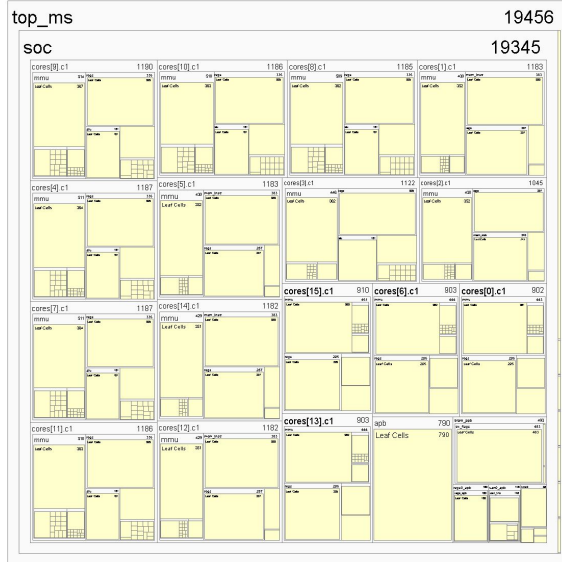Memory Map
Interconnect
Interrupts

Multi-core
Functionality

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.

- **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

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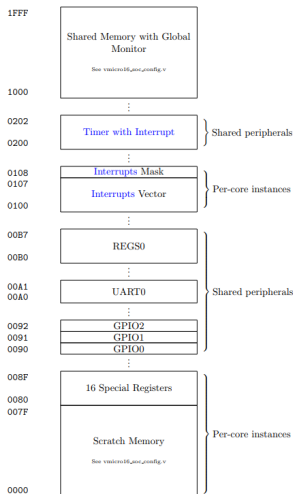
- **Shared Memory with Global Monitor**
- Timer with Interrupt
- Per-core Interrupt Vector and Mask
- Shared Register Set
- UART Transceiver
- Multiple GPIO ports
- Per-core scratch memory
- **Per-core Special Registers**
- Customisable by designers

Interconnect

Interrupts

Main Project

B. Lancaster

Introduction

Top Level
Design
Overview
Memory Map
Interconnect
Interrupts

Multi-core
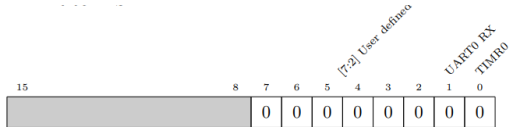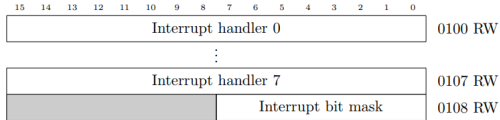Functionality

Results

Conclusion

```
 1  entry:
 2      // Set interrupt vector at 0x100
 3      // Move address of isr0 function to vector[0]
 4      movi    r0, isr0
 5      // create 0x100 value by left shifting 1 8 bits
 6      movi    r1, #0x1
 7      movi    r2, #0x8
 8      lshft   r1, r2
 9      // write isr0 address to vector[0]
10      sw      r0, r1
11
12      // enable all interrupts by writing 0x0f to 0x108
13      movi    r0, #0x0f
14      sw      r0, r1 + #0x8
15      halt                    // enter low power idle state
16
17  isr0:                       // arbitrary name
18      movi    r0, #0xff       // do something
19      intr                    // return from interrupt
```

Demo: 2 Core LED toggle (GPIO0) with TIMR0 1s interrupt (interrupts_2.s)

Timer Interrupt Example

Figure: TIMR0 1us interrupt with context switching

Timer Peripheral Registers

Main Project

B. Lancaster

Introduction

Top Level
Design
Overview
Memory Map
Interconnect
Interrupts

Multi-core
Functionality

Results

Conclusion

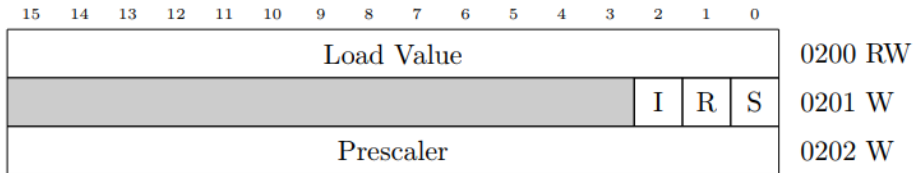| 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Load Value																0200 RW
													I	R	S	0201 W
Prescaler																0202 W

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

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)

1. Introduction

2. Top Level Design

3. Multi-core Functionality
   HW/SW Requirements
   Context Identification
   Atomics

4. Results

5. Conclusion

HW/SW Requirements

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
Requirements
Context Identification
Atomics

Results

Conclusion

Hardware:

- **Bus Arbitration**
  (scheduling: priority, rotating, etc.)

Software:

HW/SW Requirements

Hardware: Software:

- **Bus Arbitration**
  (scheduling: priority, rotating, etc.)

- **Atomic functions**
  (atomic versions of load/store to
  prevent race conditions)

HW/SW Requirements

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
Requirements
Context Identification
Atomics

Results

Conclusion

Hardware:

Software:

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

HW/SW Requirements

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
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
- Per-core context-switching for
  interrupt handling

Software:

HW/SW Requirements

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)

HW/SW Requirements

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
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
- Per-core context-switching for
  interrupt handling

Software:

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

HW/SW Requirements

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
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
- Per-core context-switching for
  interrupt handling

Software:

- **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
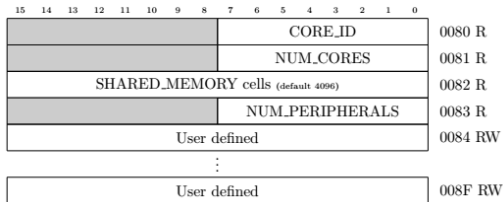
Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
Requirements
Context Identification
Atomics

Results

Conclusion

Figure: Special Registers 0x0080 to 0x008F

```
entry:
    // get core idx 0x80 in r7
    movi    r7, #0x80
    lw      r7, r7

    // Branch away if not core 0
    cmp     r7, r0
    movi    r0, exit
    br      r0, BR_NE

    // Core 0 only instructions
    nop
    nop
    nop

exit:
    halt
```

Atomic Instructions

- Enables semaphores, mutexes, memory barriers

- Prevent race conditions between threads/cores

- LW[EX] and SW[EX]

- Implementation in next slide

Example:

```
try_inc:
    // load and lock (if not already locked)
    lwex    r0, r1
    // do something (i.e. add 1 (semaphore))
    addi    r0, #0x01
    // attempt store
    swex    r0, r1

    // check success (== 0)
    cmp     r0, r3

    // if not equal (NE), retry
    movi    r4, try_inc
    br      r4, BR_NE

critical:
    // r0 is latest value
```

Exclusive Access Flow Chart

Main Project

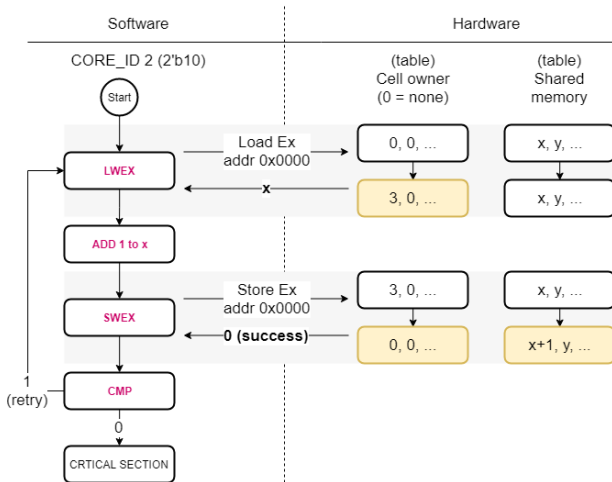B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality

HW/SW
Requirements

Context Identification

Atomics

Results

Conclusion

HW - How do I know which core this lwex/swex is from?

20	19	18 17 16	15	0	
LE	SE	CORE_ID	Address		PADDR[20:0]
			Write data		PWDATA[15:0]
			Read Data		PRDATA[15:0]
				WE	PWRITE[0:0]
				EN	PENABLE[0:0]

The Core Idx is sent with each MMU request to the shared bus.

83		62		41		20		0
Core N-1			$\cdots$		Core 1		Core 0	

PADDR*NUMCORES-1:0 interconnect input.

Exclusive Access

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality
HW/SW
Requirements
Context Identification
Atomics

Results

Conclusion

```
mutex_claim:
    // load and lock (if not already locked)
    lwex    r0, r1
    // do something (i.e. add 1 (semaphore))
    addi    r0, #0x01
    // attempt store
    swex    r0, r1

    // check success (== 0)
    cmp     r0, r3

    // if not equal (NE), retry
    movi    r4, mutex_claim
    br      r4, BR_NE

critical:
```
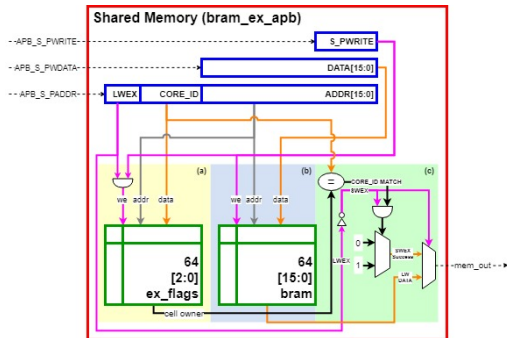
1.

Figure: HW impl

Demo: 8 core number summation (sum.s)

1 Introduction

2 Top Level Design

3 Multi-core Functionality

4 Results
  Results 1

5 Conclusion

Multi-core vs Single-Core for Summation

- Each 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

1 Introduction

2 Top Level Design

3 Multi-core Functionality

4 Results

5 Conclusion
   Accomplishments
   Future Improvements
   Q&A

Main Project

B. Lancaster

Introduction

Top Level
Design

Multi-core
Functionality

Results

Conclusion
Accomplishments
Future Improvements
Q&A

Accomplishments

- **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

- **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.

Q&A

Info

- 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`

- Presentation tools:
  - Latex Beamer

  - `\usecolortheme{orchid}`

  - `\useoutertheme[hideothersubsections]{sidebar}`