Multi-core RISC SoC Design & Implementation

Demonstration Viva

Ben Lancaster

201280376
ELEC5881M - Main Project

August 8, 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/

Why a project on CPUs?

- **CPUs will be used for the rest of humanity**
- **Understand design constraints and considerations**
- **Prepare for future employment/work**

Why Multi-core?

- **Rate of single-core speed improvements slowing**
- **Future of computing = parallel**

Why RISC?

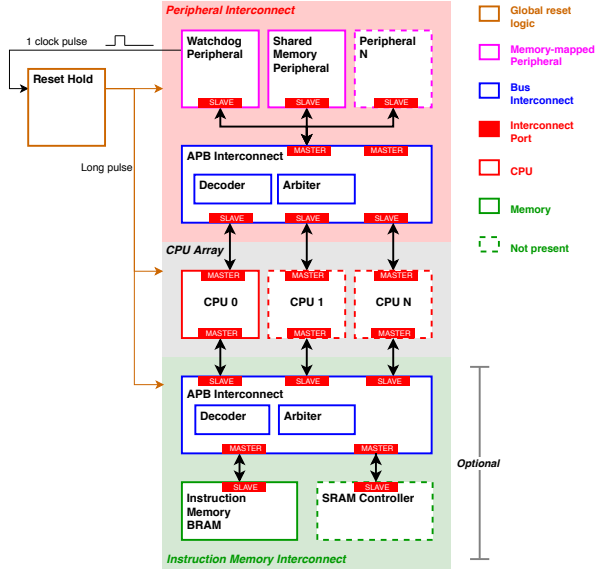
- Simpler design & impl
- Smaller = fit more cores on a chip
- Previous experience + future work
- I'm a RISC purist

Overview

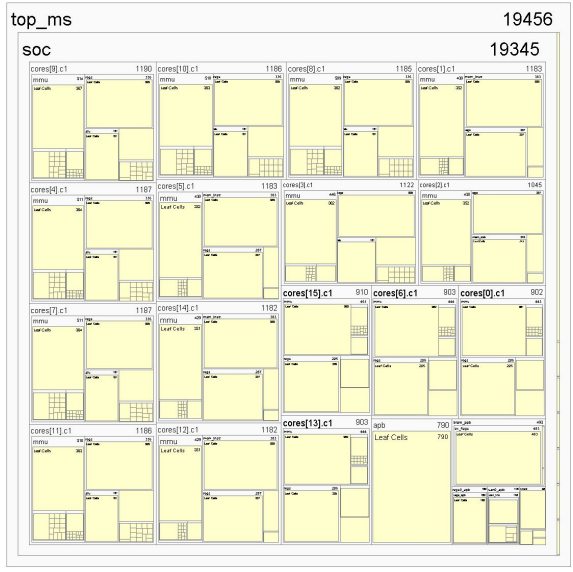
What this project produces:

- System-on-Chip with multi-processor functionality
- Custom 16-bit RISC CPU
- Software/Assembly compiler
- Aimed at Design Engineers, not end users

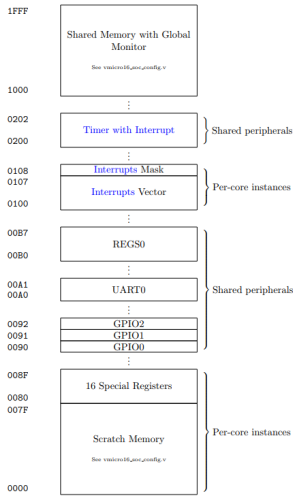
Top Level Block Diagram



Top Level Hierarchy



Memory Map



- **Shared Memory***
- Timer + interrupts
- UART send/receive
- GPIO
- Scratch memory
- Extra registers
- + more

Interconnect

- AMBA APB Bus
- Tristate & Non-tristate (mux) impl
- Originally Wishbone, now APB
- AHB too complex

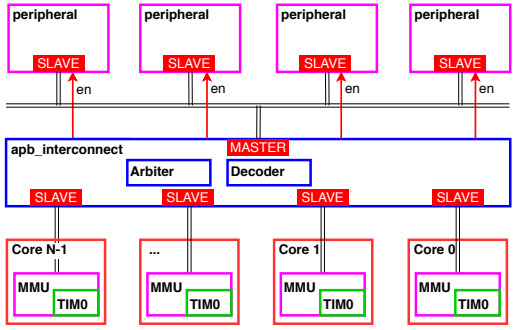


Figure: Vmicro16 interconnect

Interconnect Schematic

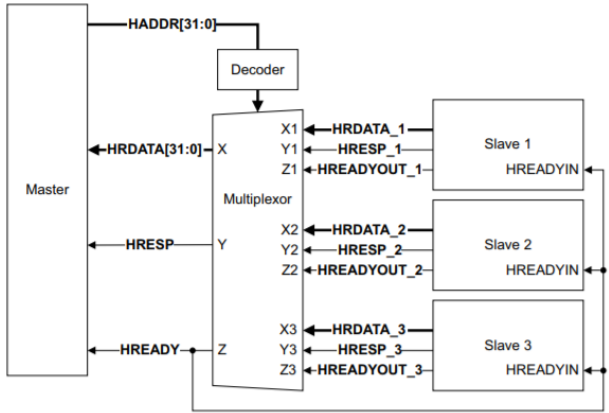


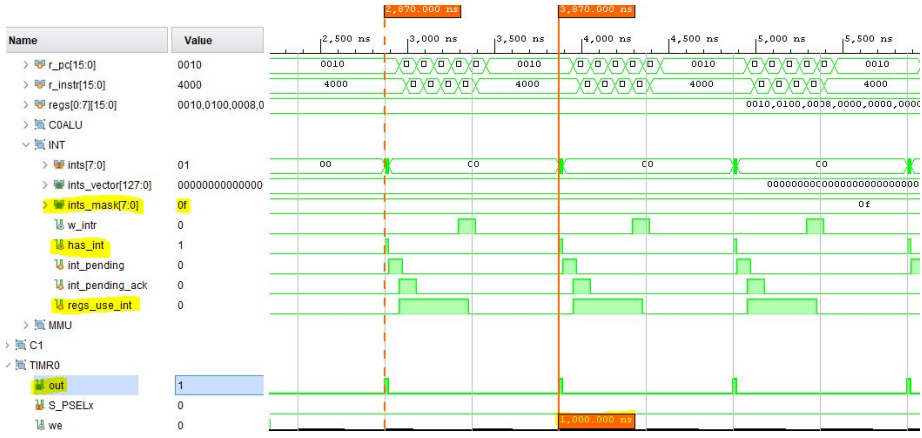
Figure: Source: ARM AHB-Lite Protocol Specification Figure 4-2.

Instruction Set Architecture

- 16-bits
- 35 instructions
- HALT (low power)
- INTR
- HW multiply

	15-11	10-8	7-0	rd imm8
	15-11	10-0		nop
	15	14:12	11:0	extended immediate
SPCL	00000	11 bits		NOP
SPCL	00000	11h'000		NOP
SPCL	00000	11h'001		HALT
SPCL	00000	11h'002		Return from interrupt
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	Imm8	Rd <= (Imm8 _f_ SZO) ? 1 : 0
MULT	01011	Rd	Ra X	Rd <= uRd * uRa
HALT	01100	X		
LWEX	01101	Rd	Ra s5	Rd <= RAM[Ra+s5]
SWEX	01110	Rd	Ra s5	RAM[Ra+s5] <= Rd Rd <= 0|1 if success

Timer Interrupt Example



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

Thread Synchronisation

- Semaphores, mutexes, memory barriers
- Prevent race conditions
- LW[EX] and SW[EX]

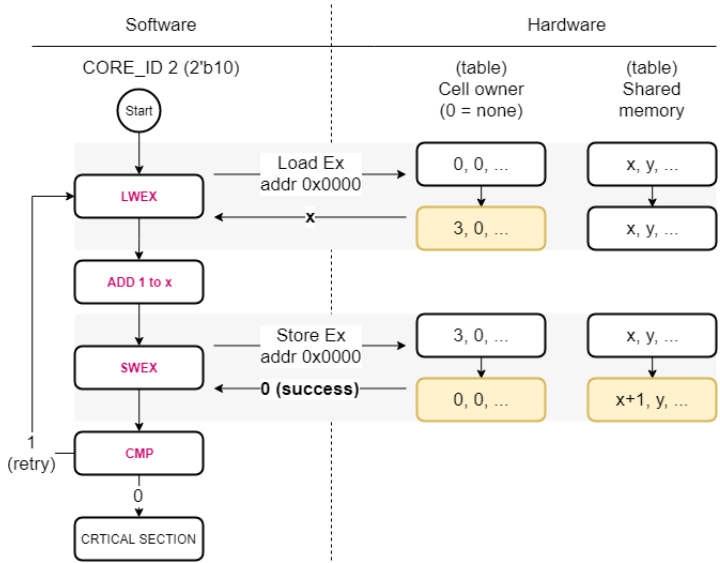
```
try_inc:
    // load and lock
    // (if not already locked)
    lwex    r0, r1
    // do something
    // (i.e. add 1 (semaphore))
    addi    r0, #0x01
    // attempt store
    swex    r0, r1

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

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

critical:
    // r0 is latest value
```

Thread Synchronisation - Flow Chart



Thread Synchronisation - HW

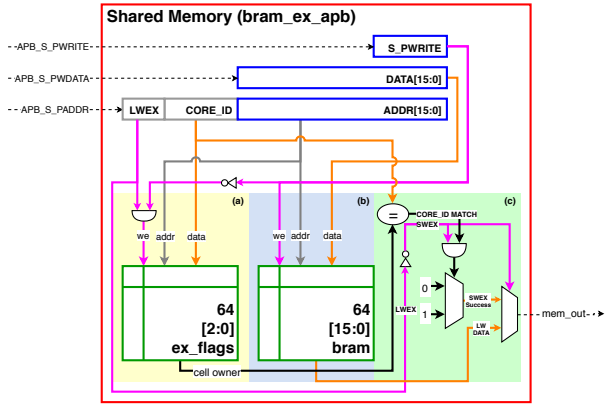


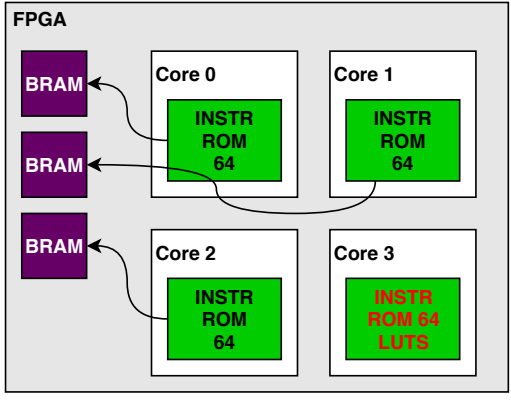
Figure: HW impl

Demo: 8 core number summation (sum.s)

Design Challenges

Memory Limitations

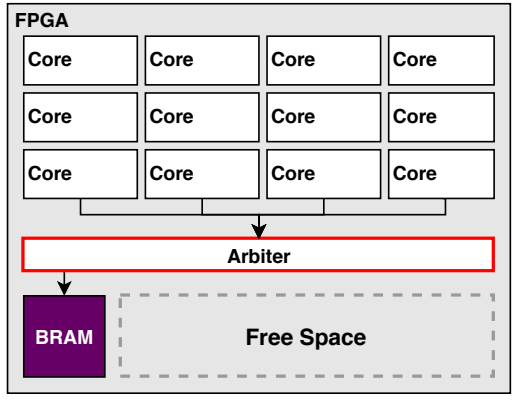
Each core has it's own instruction memory



Design Challenges

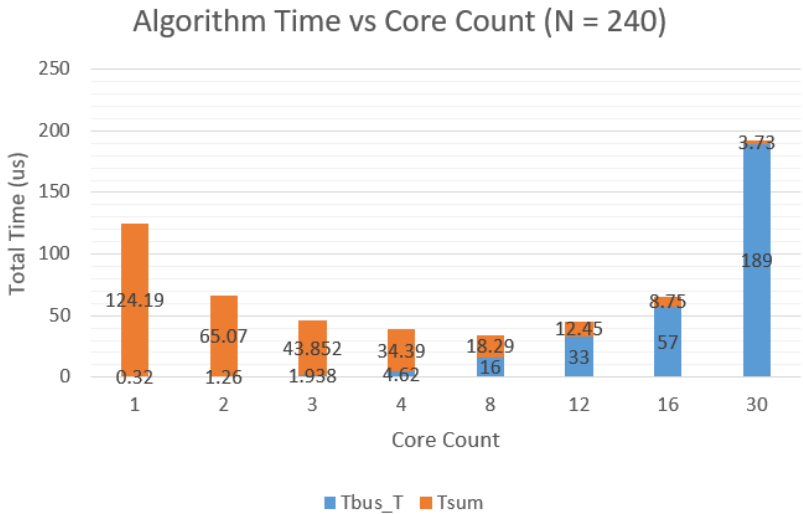
Memory Limitations - Solution

Global instruction ROM



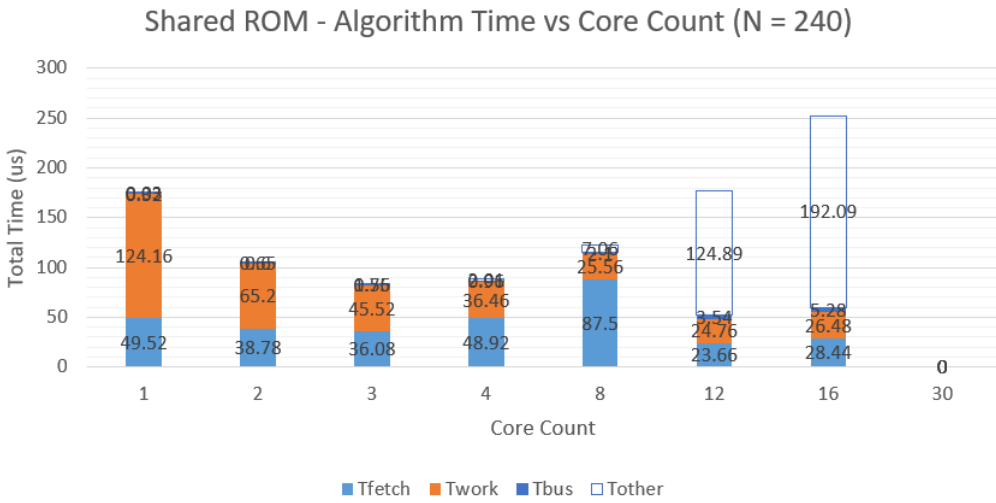
Summation - Multi-core vs Single-Core

240 samples (@30 cores = 8 samples per core)



Shared Instruction ROM

240 samples (@30 cores = 8 samples per core)



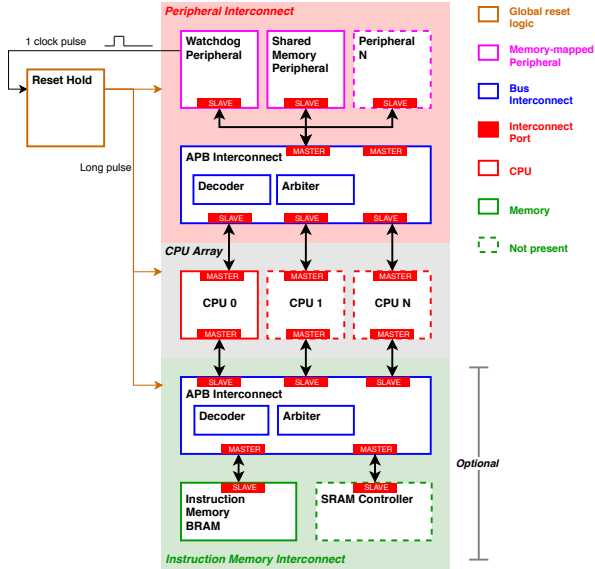
Accomplishments

- **System-on-Chip with peripherals**
  Timers, GPIO, UART, Registers, Memory
- **Common multi-thread/core synchronisation primitives**
- **AMBA APB bus interconnects**
- **Interrupts with hardware context-switching**
- **Understanding of limitations and solutions**

Future Improvements

- **Global Reset**
- **On-chip Programming**
- **Per-core gating/enabling**
- **Improve memory bottleneck**

Q&A



Q&A

- 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}`

Appendix

Context Identification

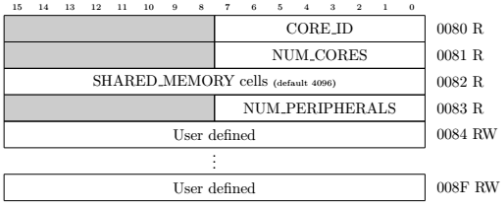


Figure: Special Registers 0x0080 to 0x008F

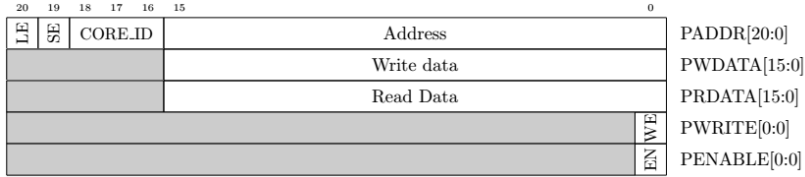
```
entry:
    // get core idx 0x80 in r7
    movi    r7, #0x80
    lw      r7, r7

    // 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
```

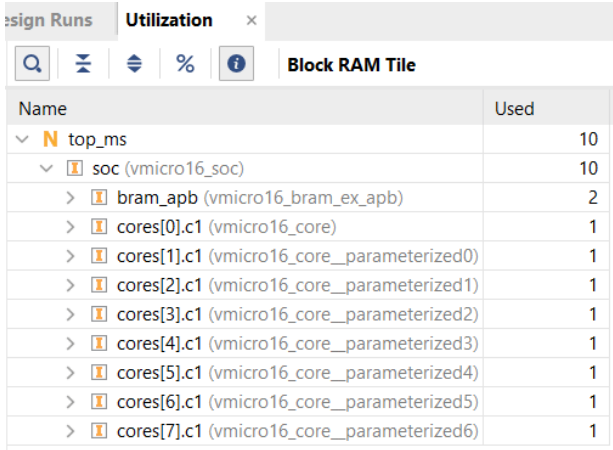
HW - How do I know which core this lwex/swex is from?



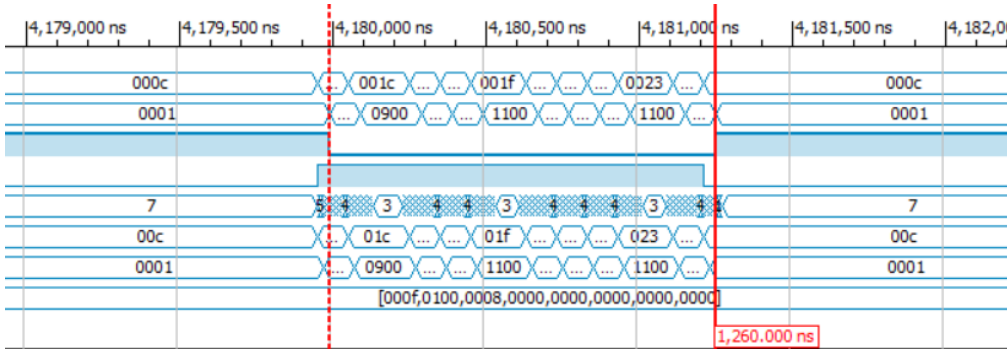
The Core Idx is sent with each MMU request to the shared bus.

PADDR*NUMCORES-1:0 interconnect input.

BRAM Utilisation per Entity



Halt State Low Power



Partial Address Decoding