An Open Source, RV32I based computer in FPGA

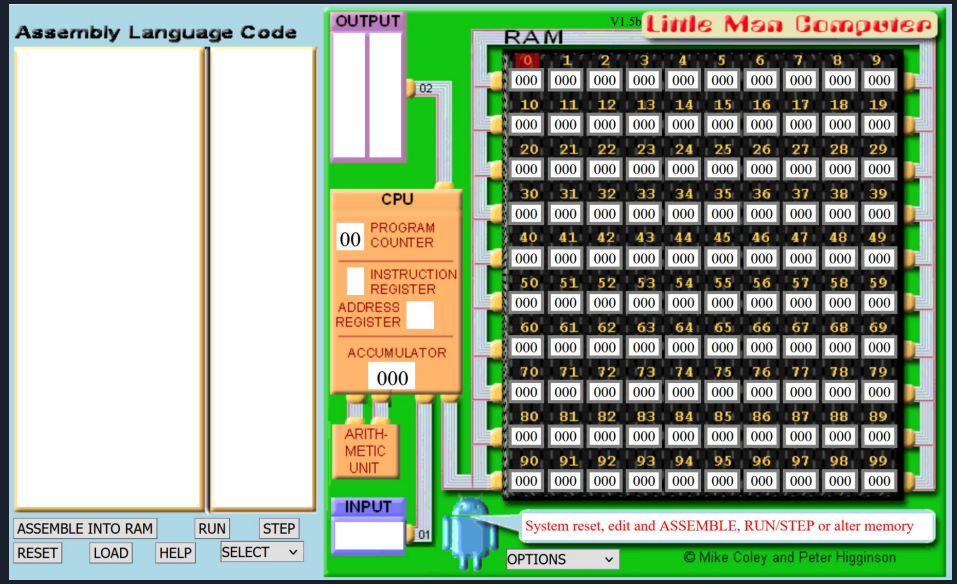
Fully open-source, from HDL to toolchain

Kristoffer Robin Stokke, PhD

https://github.com/memstick/fpga/tree/master/lib

How do computers work?

- Teaching assembler in IN1020; computer architecture IN2140
- This is the “Little Man Computer” ->
- Very simple
  - 8 instructions
  - Reads numbers
  - Writes ASCII
  - 100 memory cells
- But..
  - Decimal model — not binary hardware
  - Not a *real* machine
  - No toolchain - no C or Rust



(Difficult?) Theoretical Concepts

Program Counter

Stack

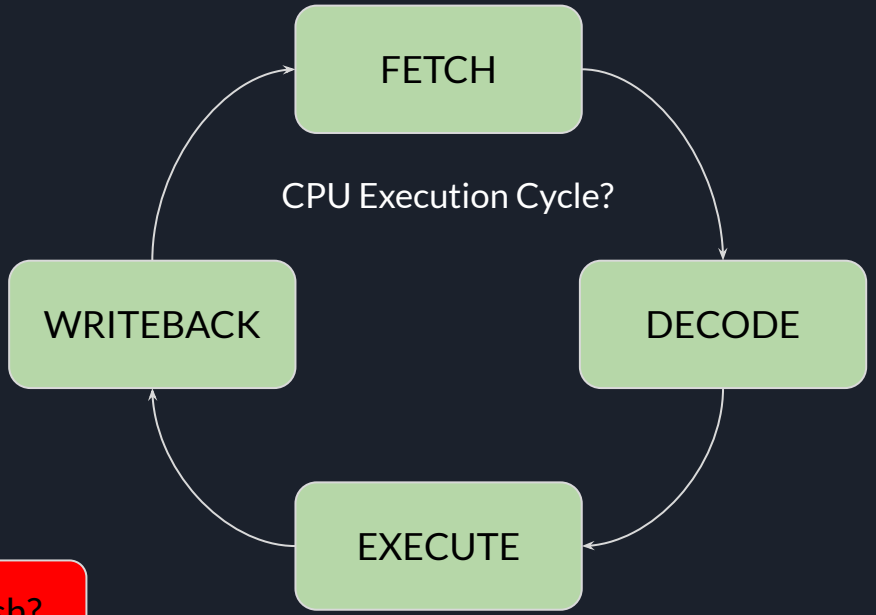
Zero register

Link register

MAR

MDR

Internal registers for bus access



Clock frequency?

Volatile memory?

Self-refresh?

Virtual Memory?

CPU Cache?

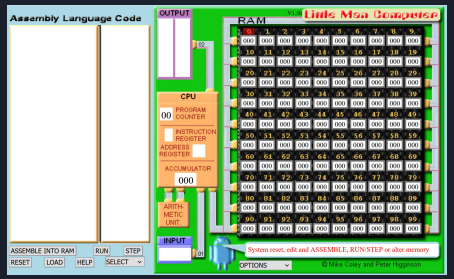
Compiler?

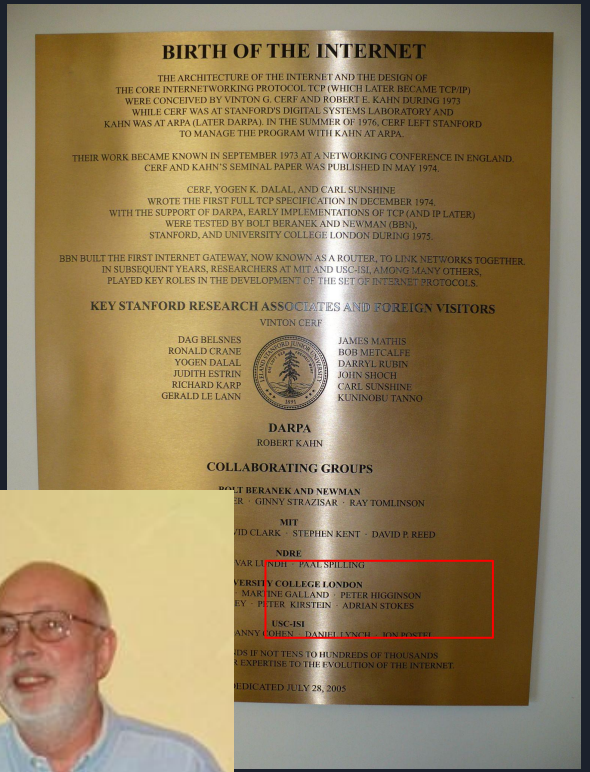
Assembler?

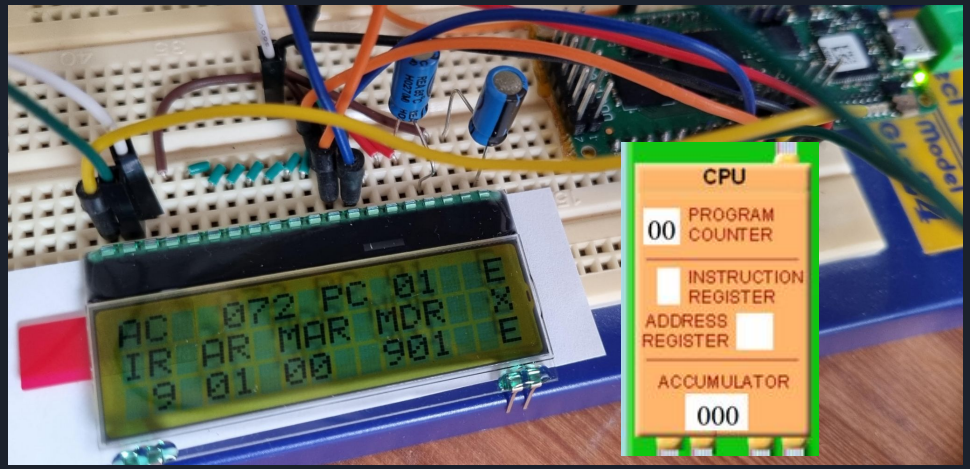
Linker?

Little Man Computer

- I met Peter in the UK!







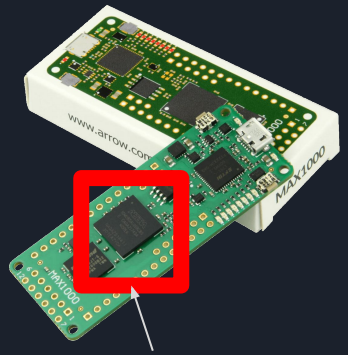
Peter Higginson

https://peterhigginson.co.uk/lmc/ https://peterhigginson.co.uk

Why Field Programmable Gate Array (FPGA)?

- FPGAs are the closest a “salaryman” can come to build their own integrated circuit with a CPU inside!
- No fabrication required (except connecting devices)

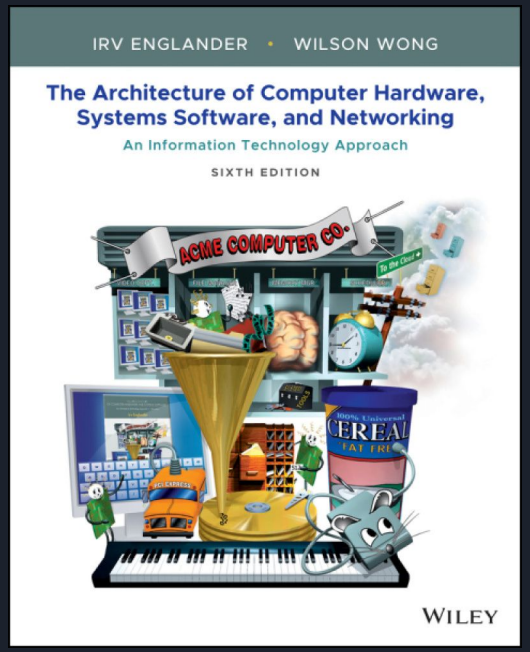
Pentium 4



Max10 FPGA

ALTERA

“Why not implement the RV32i”?

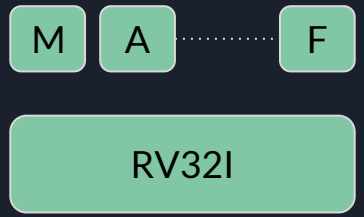


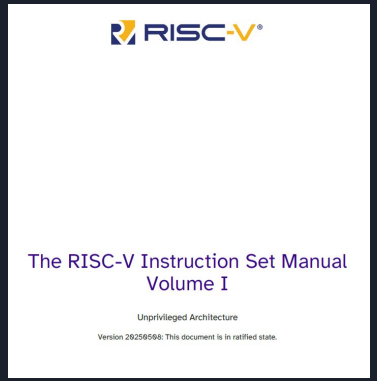
Håkon Kvale Stensland
https://www.simula.no/people/haakonks

Why RISC-V?

- Open ISA defines the instruction set — not how the processor is implemented
- Modular and minimal (RV32I = small base)
- GCC/LLVM/Rust support
- Royalty free

(M) Multiplication
(A) Atomics
(F) Floating point

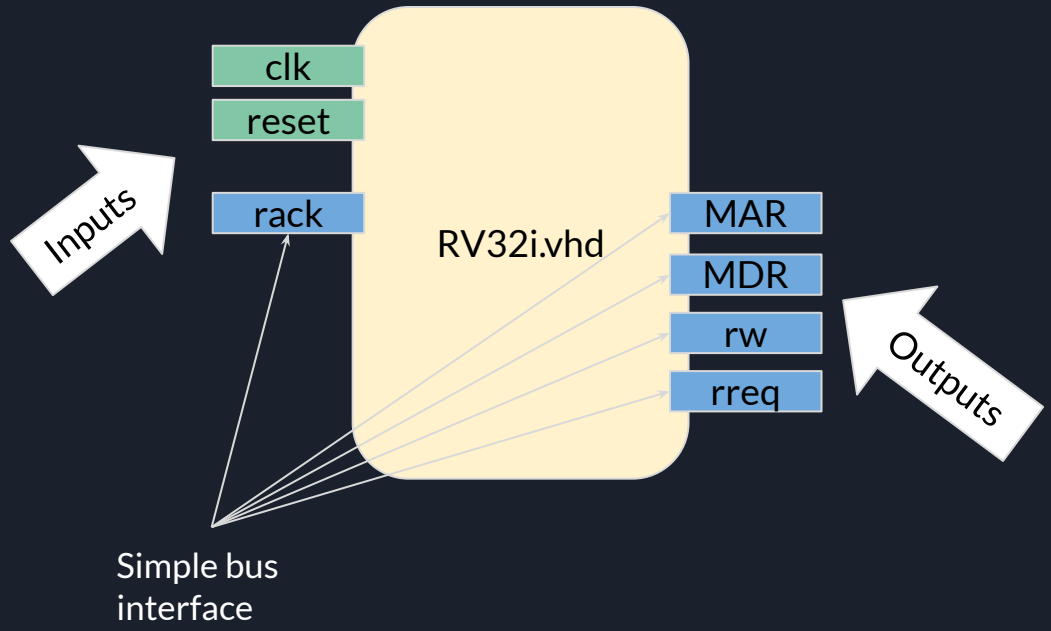




Jump | Load | Store
Branch | Integer | Logical

Rv32I Core Architecture

- 4-stage pipeline
- **In-order execution**
- **Stalls on memory ACK**
- **No branch speculation**
- 50 MHz system clock
- 32 bit
- This module is only 200 lines!



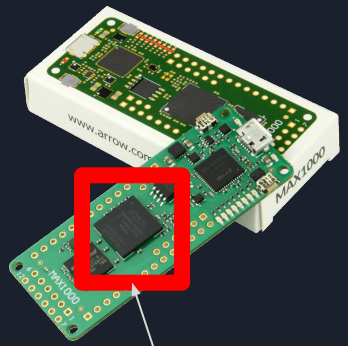
Just a small VHDL sample...

During EXECUTE stage, we pattern-match on opcode and call helper procedures.

```
when EXECUTE =>

    case OP is
        when rv32i_lui =>
            rv32i_execute_lui(IMM, GPR_RD, state);

        when rv32i_auipc =>
            rv32i_execute_auipc(IMM, PC, GPR_RD, state);

        when rv32i_jal =>
            rv32i_execute_jal(IMM, PC, nPC, GPR_RD, state);
```

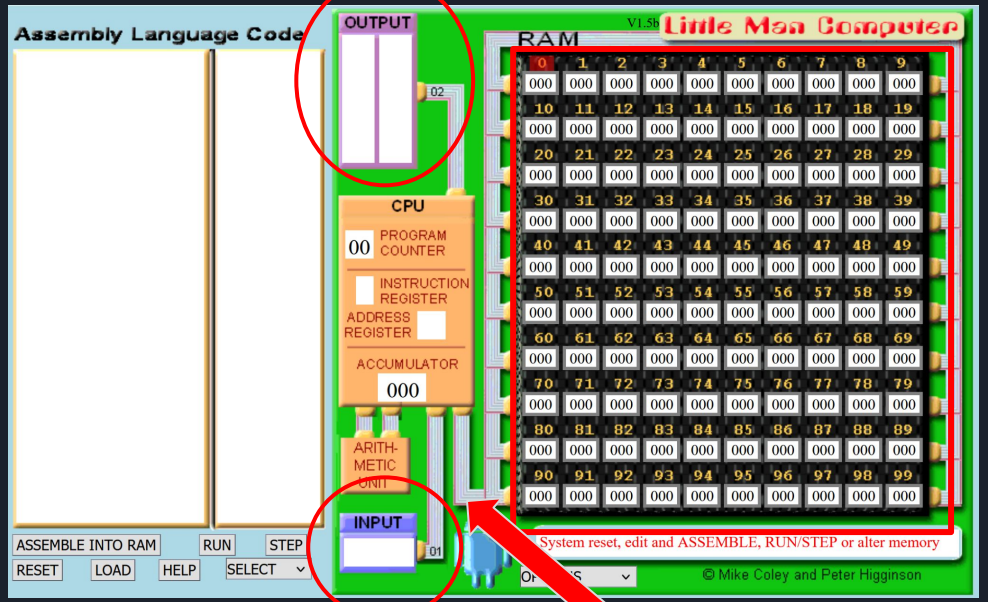
Is this enough to make a computer?

RV32i



Output?

Memory?



Input?

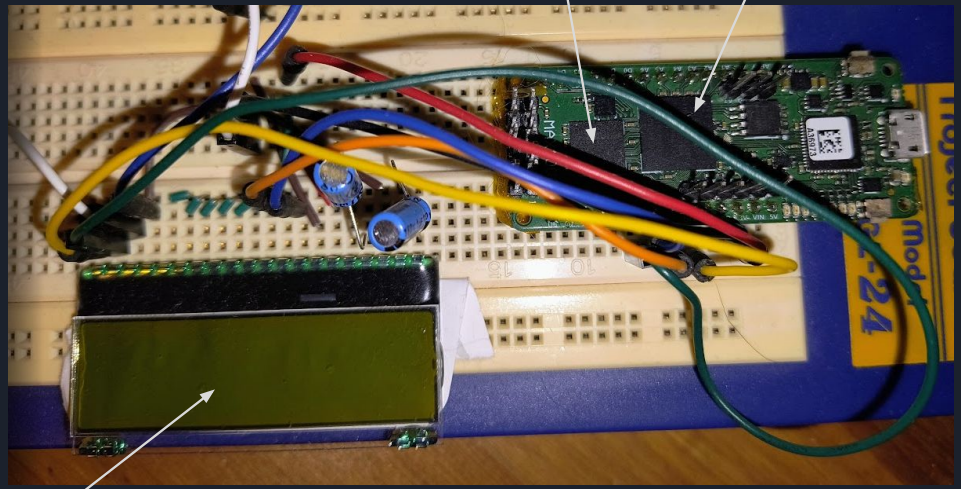
A **bus**?

What code should the core run when it starts (boot code)?

Loadout Needed:

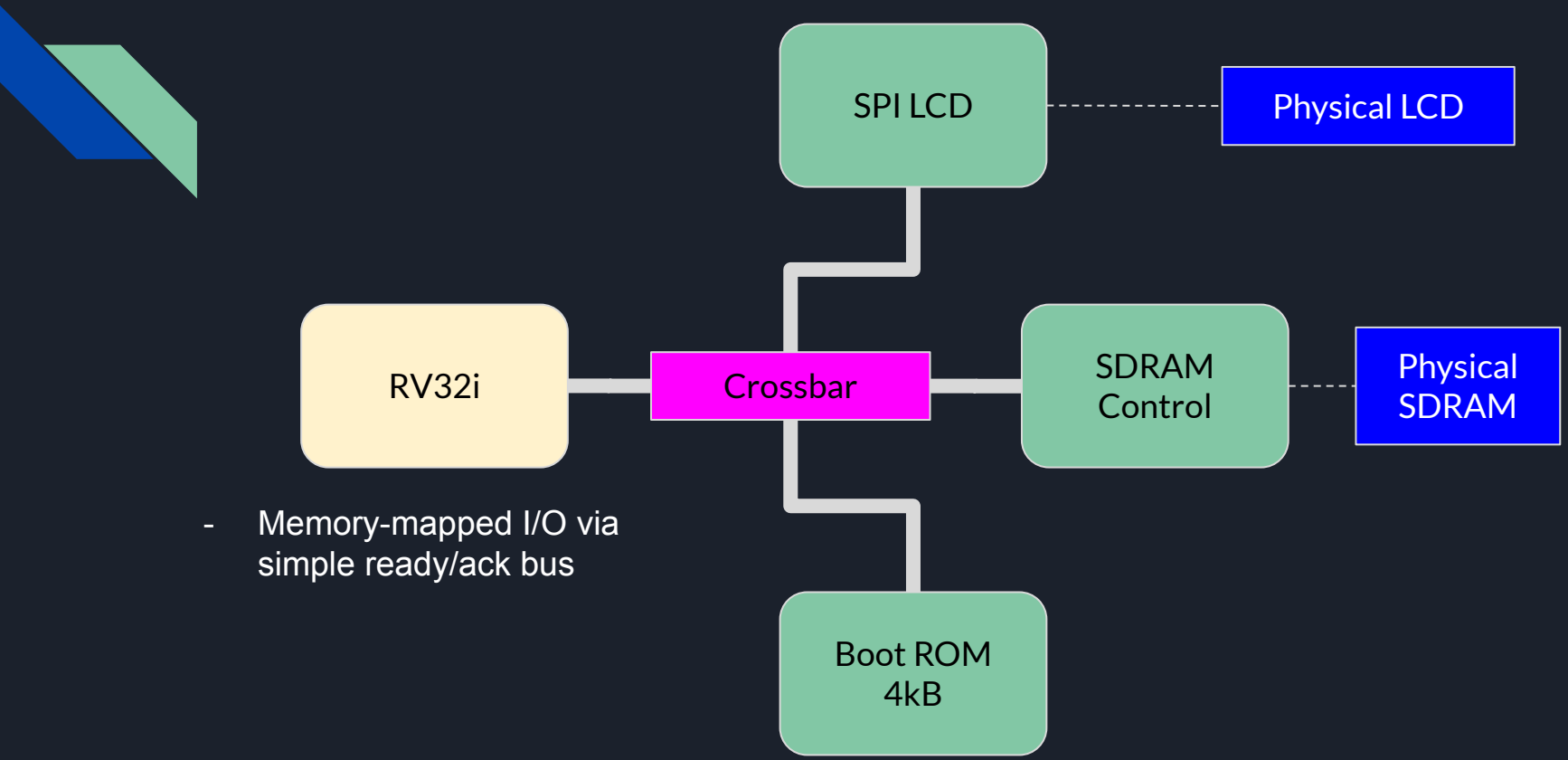
32MB SDRAM

Max10 FPGA
- RV32I core



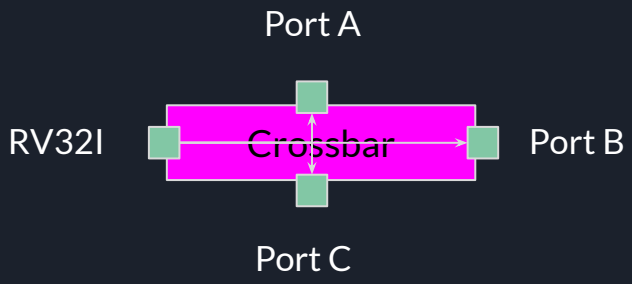
LCD Display ~150 NOK

https://no.rs-online.com/web/p/lcd-monochrome-displays/1711876



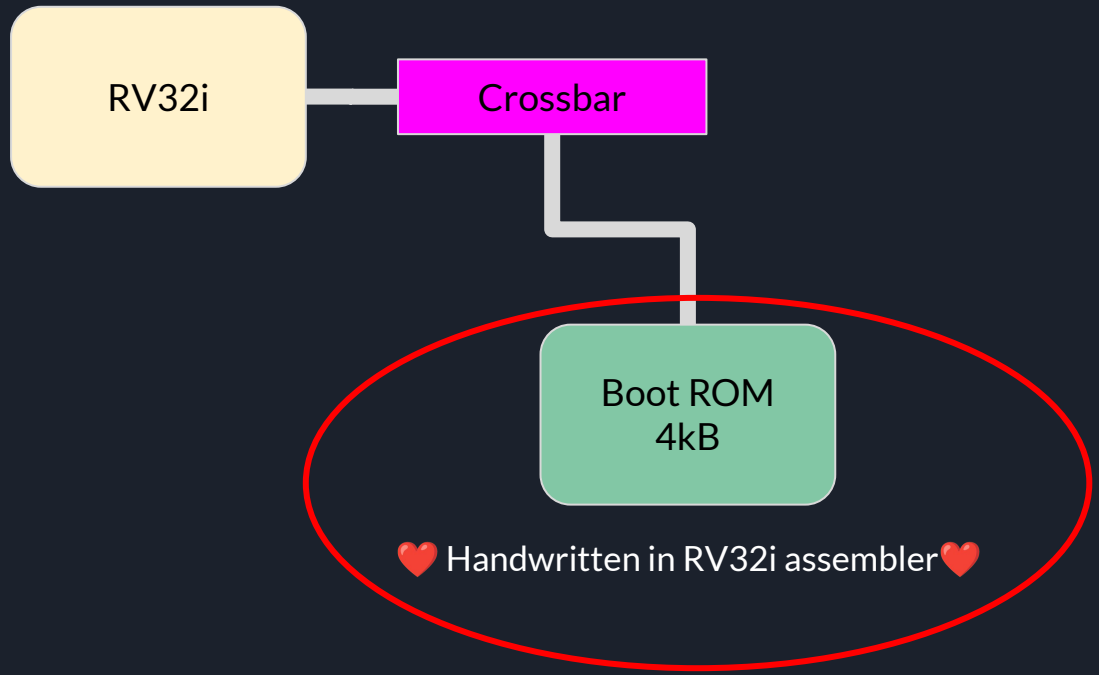
- Memory-mapped I/O via simple ready/ack bus

Crossbar: Memory Mapped IO

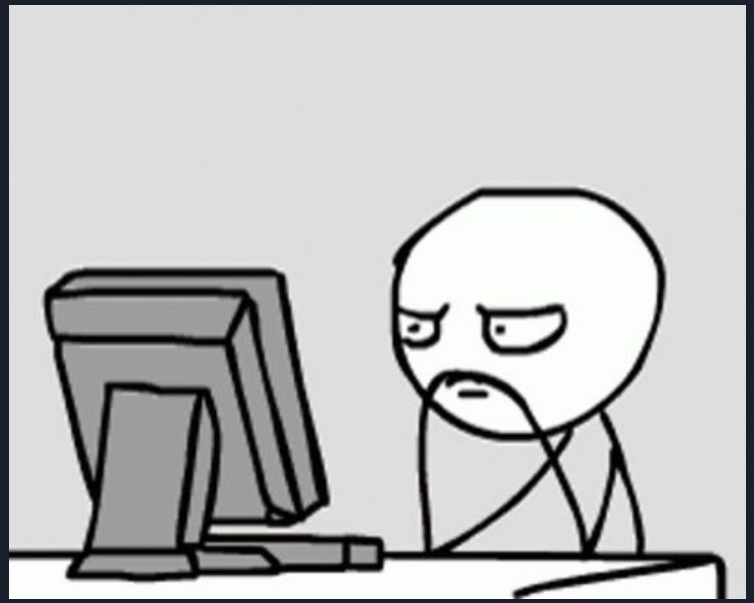


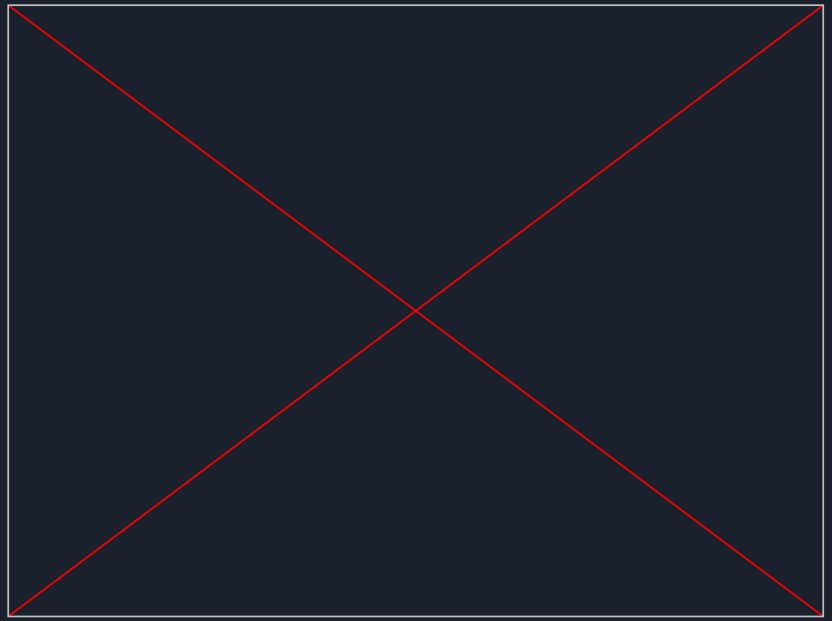
Address Prefix (first 4 bits)	Port	Module
0xxxxxxxxx	A	SDRAM
8xxxxxxxxx	B	LCD Framebuffer
4xxxxxxxxx	C	Boot ROM

What Should The Core Do When It Starts?



First RV32I demo: Written entirely in assembler





This demo is hand-written in assembler. Looping, LCD animation + SDRAM integer variable.

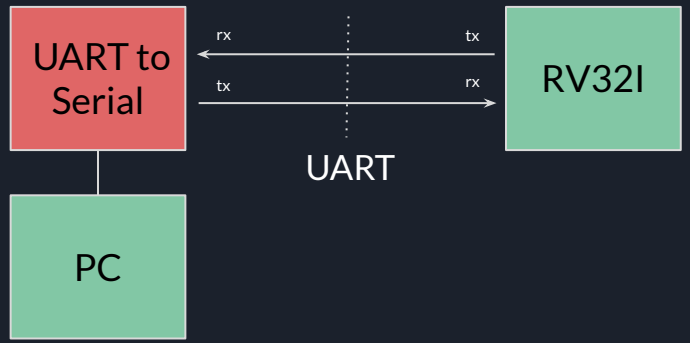
Gems from the World of Assembler

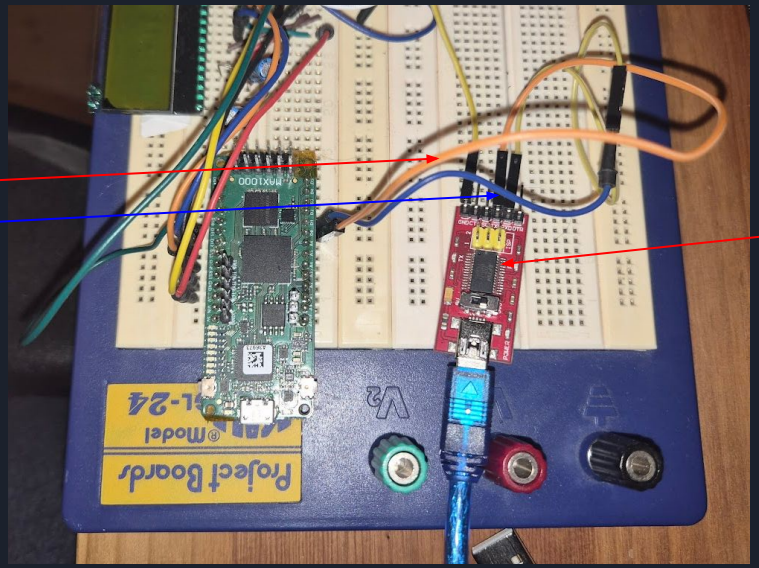
- I had not written testbenches during my last visit at Inventas.

The initial bringup was done with LEDs and “stepping” cycle by cycle with a button on the board.

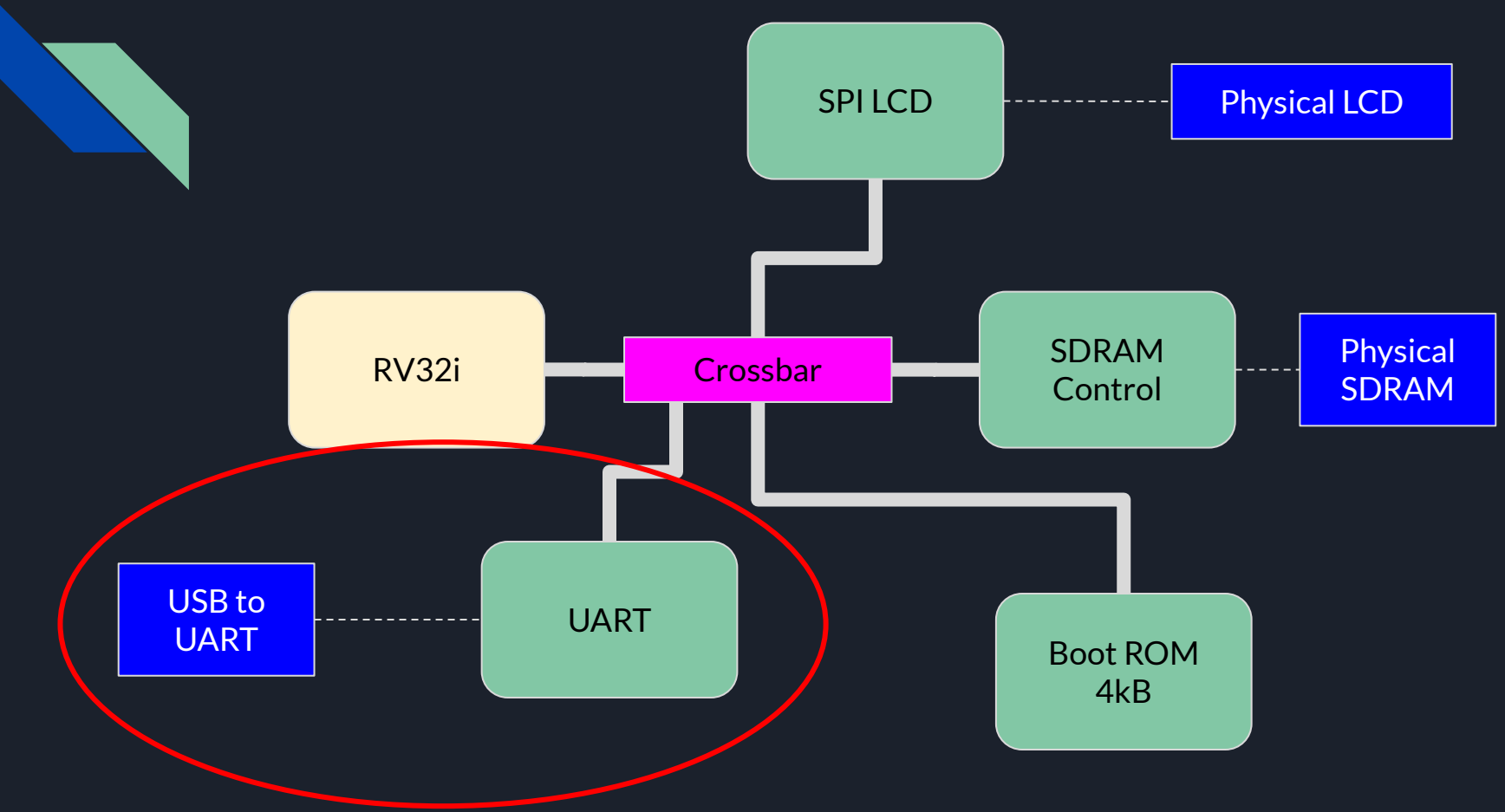
The RiscV core could only write chars to LCD - input was not possible

- UART
  - Two wires:
    - One for TX
    - One for RX
  - Old, low speed





UART to Serial
~150 NOK


SPI LCD
Physical LCD
RV32i
Crossbar
SDRAM Control
Physical SDRAM
USB to UART
UART
Boot ROM 4kB

What do you need to test this?

- Max1000 FPGA from Trenz electronics, USB-to-UART dongle
- Quartus Lite (free) with MAX10 device support
- GNU riscv (C) compiler
- Rust compiler
- All installation steps in project README.md :)

No standard libraries!

Wait, what.. RUST??

Memory mapped read/write!

```rust
#![no_std]
#![no_main]

use core::arch::asm;
use panic_halt as _;

const UART_BASE: usize = 0x2000_0000;
const UART_CTRL: *const u32 = (UART_BASE + 0x0) as *const u32;
const UART_DATA_WR: *mut u32 = (UART_BASE + 0x8) as *mut u32;

fn uart_putc(c: u8) {
    unsafe {
        while core::ptr::read_volatile(UART_CTRL) & 0x1 != 0 {}
        core::ptr::write_volatile(UART_DATA_WR, c as u32);
    }
}

fn uart_puts(s: &str) {
    for &b in s.as_bytes() {
        uart_putc(b);
    }
}

#[no_mangle]
pub extern "C" fn _start() -> ! {
    unsafe {
        asm!("la sp, _stack_top");
    }
    uart_puts("UART hello from Rust!\r\n");
    loop {}
}
```

Repository Layout

- software/helloworld_c
- software/helloworld_rust

Bare metal software build system

- lib/*/*.vhd

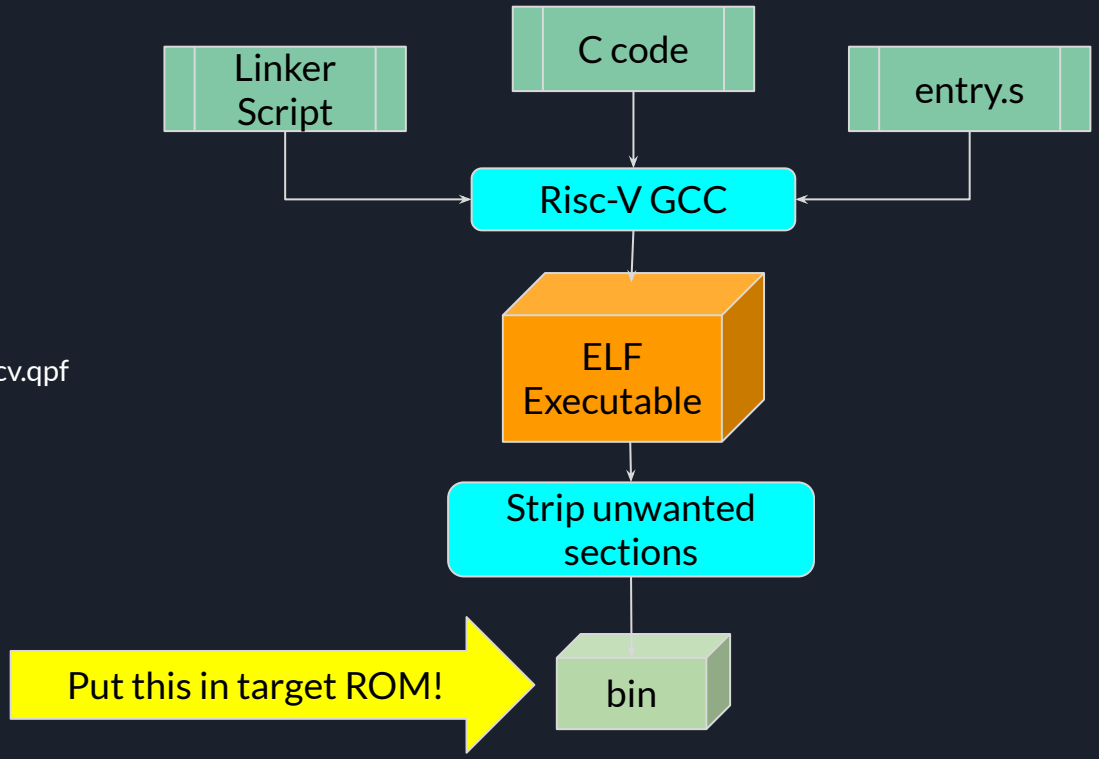
VHDL library files

- projects/

Top level projects for FPGA

The real test: Doing it in **C** / **Rust**

- Write some **bare metal** C/Rust code!
  - Use the provided demo examples!
- Run **make**!
- Open the FPGA project file in Quartus!
  - projects/max1000_riscv/max1000_riscv.qpf
- Build!
- Load the FPGA over µUSB cable!



Live Demo: C

- LCD animation + SDRAM read/write
- Write “Hello world” on UART
- Perform a full memory check
- Echo characters received from UART

Concluding Remarks

- This project demonstrates that you can run an entirely open, inspectable computer on a cheap FPGA.
- This is not high-performance and we do not compete with other RiscV implementations.
- The total cost to build a prototype board is around 800-1000 NOK.
- I hope the project can be inspiring for people who want to learn more about how computers work.
- Note that FPGA vendors typically do not provide open source VHDL toolchains.
- Licensed under Apache 2.0
- Hope that more people will take an interest and develop this further!

Questions?

🔍 Closer look at instruction encoding?

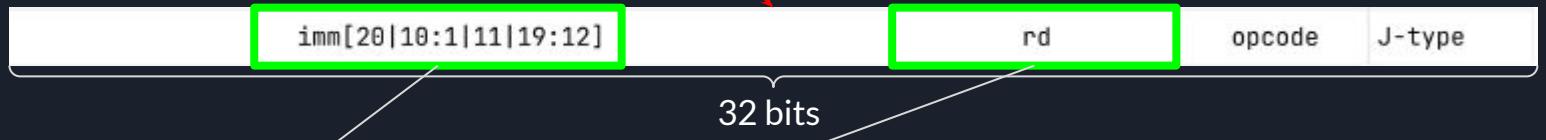
🧠 How the memory bus handshake works?

💾 SDRAM timing and burst configuration?

🛠️ Toolchain and linker details?

Other?

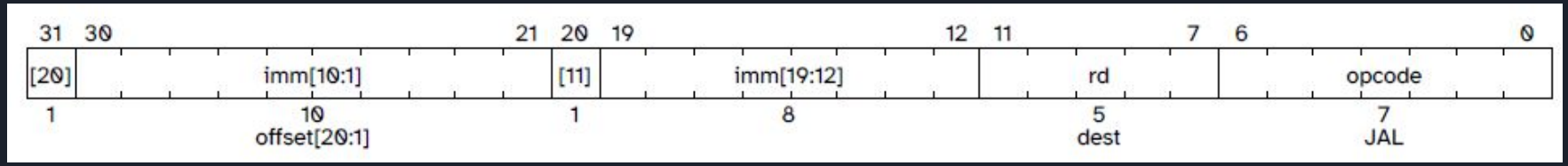
Instruction	Format	Opcode	Subencodings
Load	I-Type	0000011	LB, LH, LW, LBU, LHU
Store	S-Type	0100011	SB, SH, SW
Arithmetic	S-Type	0110011	ADD, SUB, SLL, SLT, ..
Jump	J-Type	1101111	JAL



We extract immediates and destination registers from the instruction depending on the format of the opcode.

Jump and Link (JAL)

- Jumps to PC + immediate (sign extended)
  - +/- 1MB range.
- Address following the jump (PC +4) -> destination register.



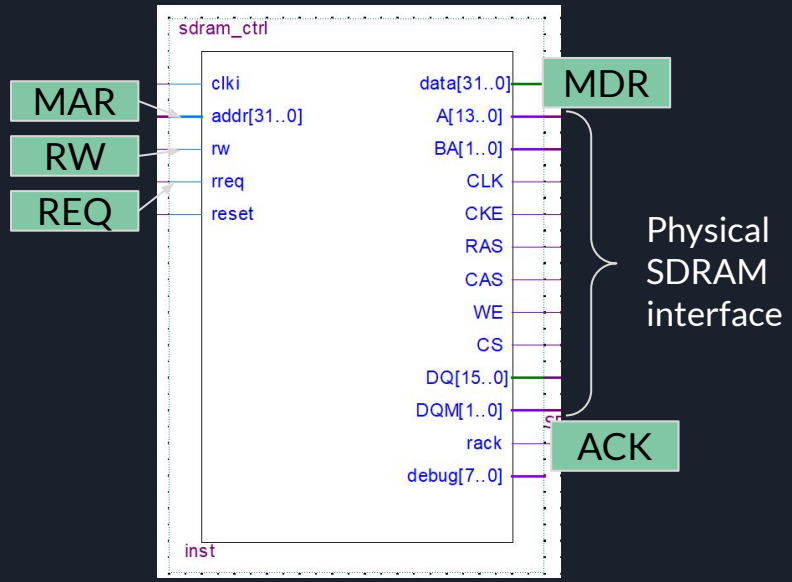
SDRAM

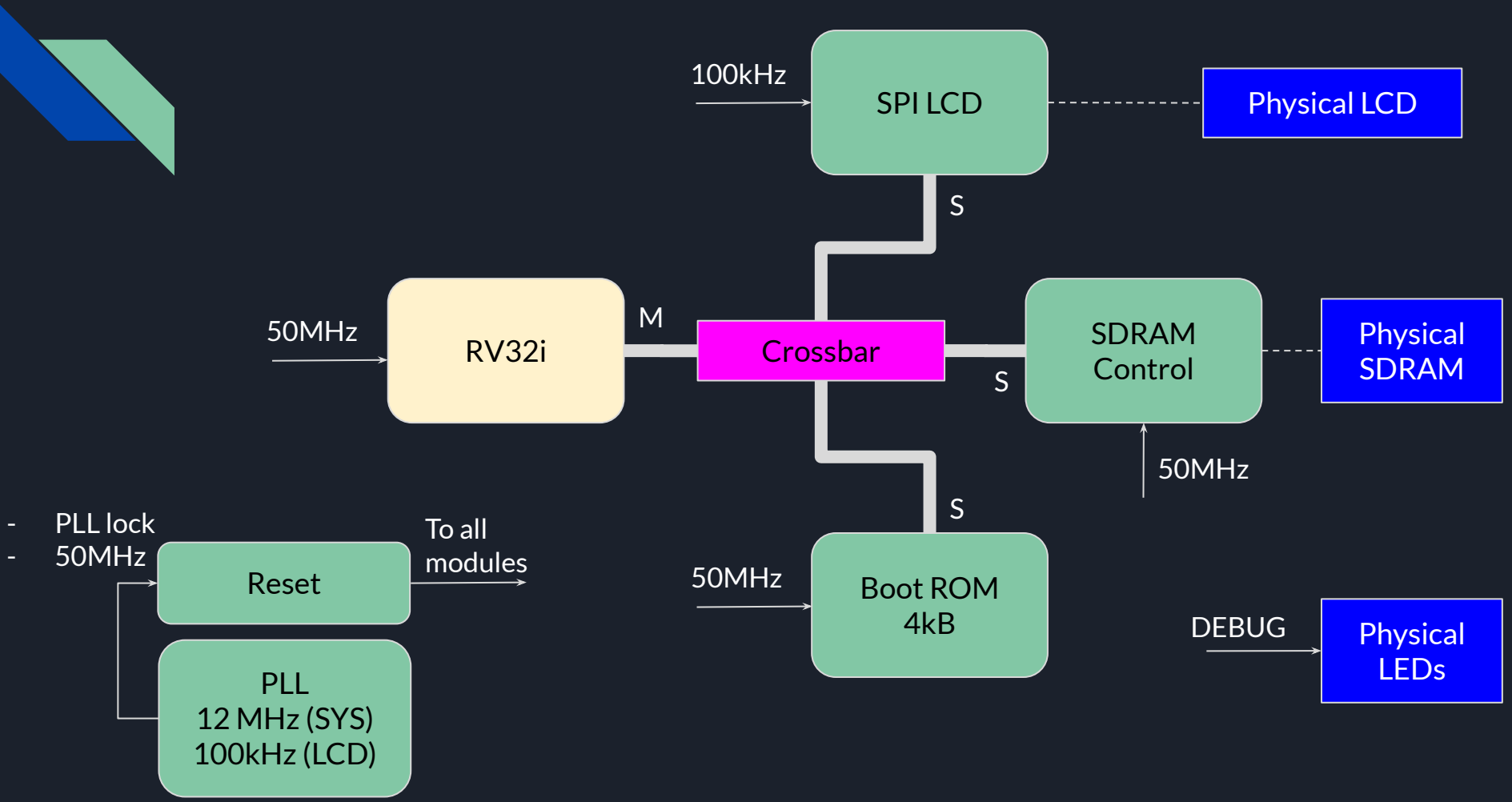
W9825G6JB

winbond

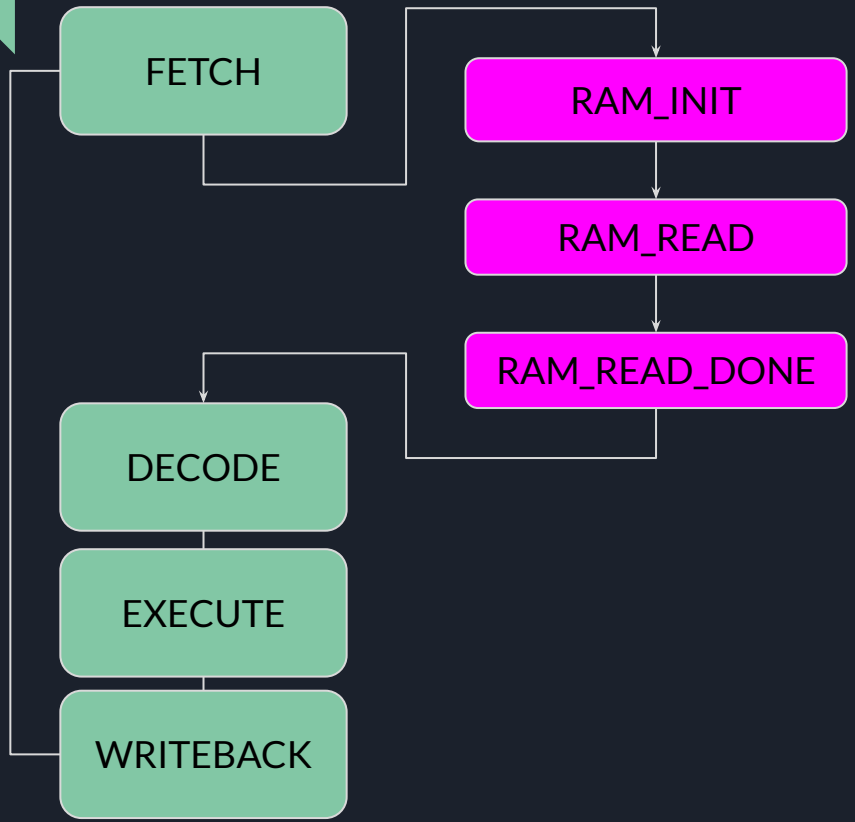
4 M × 4 BANKS × 16 BITS SDRAM

- External SDRAM on the SoM
- For the interested:
  - Burst length 2 (2x16bit = 32 bit accesses)
  - Interleaved addressing mode
  - CAS latency 3
  - Burst read + burst write




100kHz
SPI LCD
Physical LCD
S
50MHz
RV32i
M
Crossbar
S
SDRAM Control
Physical SDRAM
50MHz
S
- PLL lock
- 50MHz
Reset
To all modules
50MHz
Boot ROM 4kB
DEBUG
Physical LEDs
PLL
12 MHz (SYS)
100kHz (LCD)

Bus (memory) **accesses**



- During **fetch, we..**
  - get the next instruction from RAM/ROM

Bus (memory) accesses

RAM_INIT

E.g. to read:

*Set MAR, RW
*Set REQ <= ‘1’;

RAM_READ

*CPU: wait for ACK to go high
*RAM: fulfill read request, then set ACK <= ‘1’

RAM_READ_DONE

* CPU: Set REQ <= ‘0’;
* CPU: Wait for ACK = ‘0’