



2024-2025 MSc SoC Design Projects

**EEE
Department**

**John Goodenough
8/5/25**

S2 Wk 10 Meeting

FOCUS ON COMPLETING YOUR EXAMS FIRST!

Choose something interesting to you – an application domain maybe?

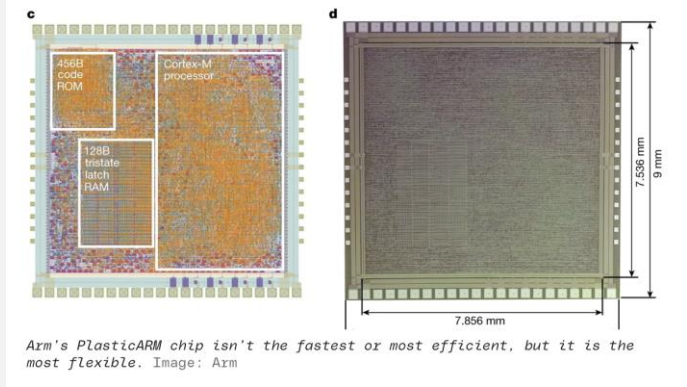
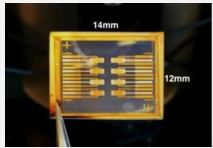
- Work on narrowing down your ***motivating system context*** and therefore the functionality/IP block you will design and integrate with the system.
- Think about your high level PLAN
 - Overall Timeline
 - Assessed Deliverables
 - Plan, Research, Learn/Upskill, execute
 - What else you have going on – modules exams, breaks
 - How much time will you spend – credits → hours???
- Think about your OPERATING MODEL
 - How are you going to organize yourself
 - How are you going to keep notes etc
 - How are you going to interact with your project supervisor

A big part of the project is about how you engage with the project – not the 'answer'

Thoroughly review the Blackboard pages for the project & understand what you need to produce

START YOUR LOGBOOK and make sure this can be shared with me

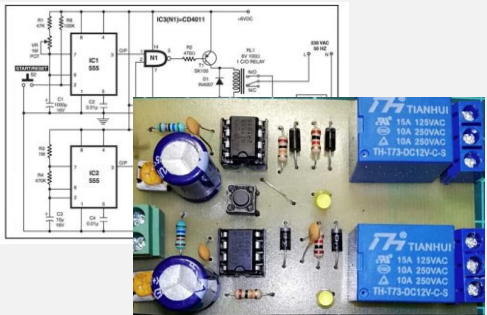
Si Smörgåsbord



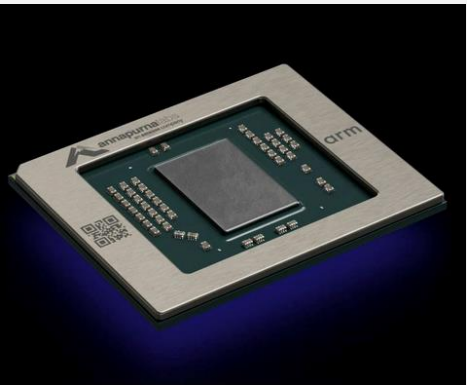
Product or Service



Electronic & Computing System



Chip or System on Chip



SoC and Systems

System on Chip
SoC

Why is the
Memory
Separate?

SoC is optimized for Application Performance and System SWaPE\$ target

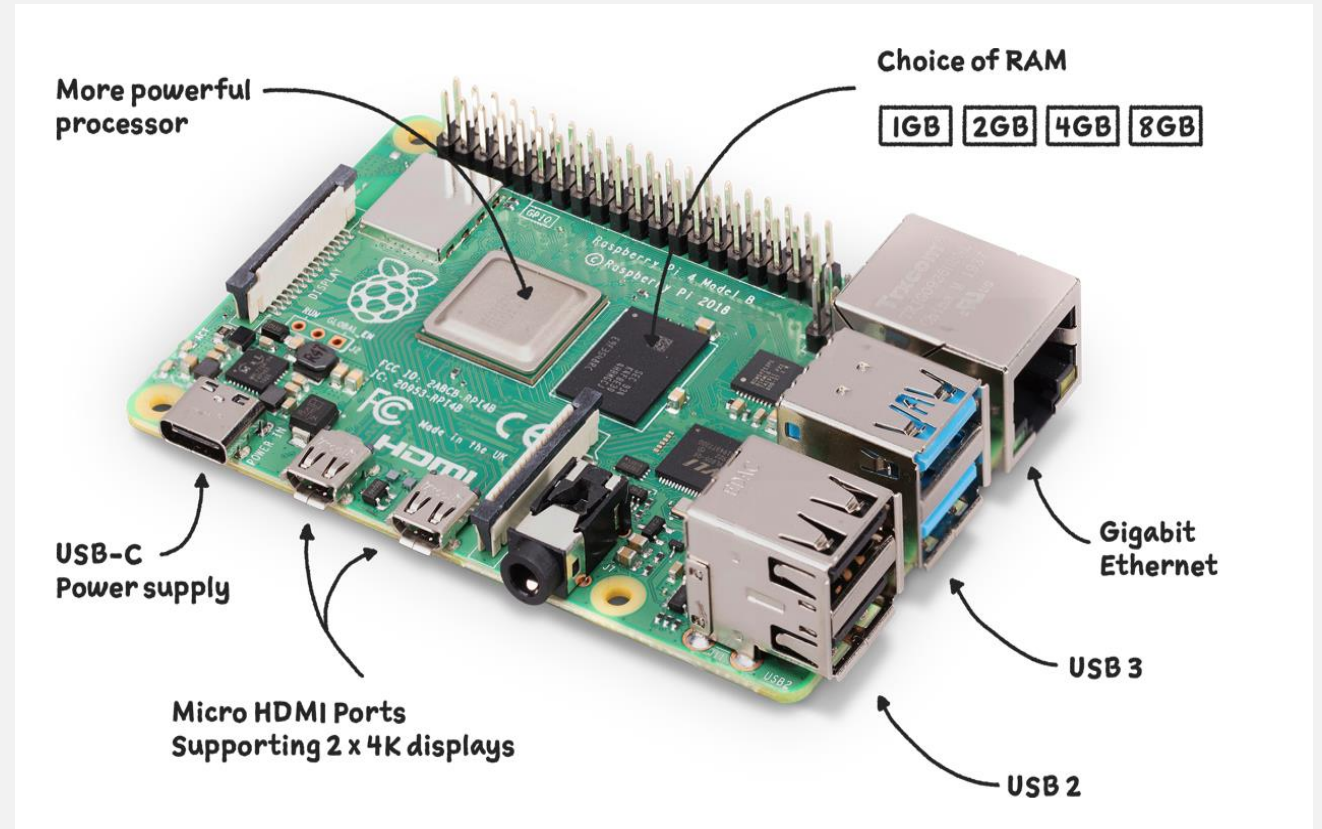
- Microcontroller or Standard part
- Do it yourself
- Application payload
- Some of the whole system is off chip
- Why?

System Memory (DRAM/HBM)

Power Supply

Passives

Connectors



Motivation and Context

You are going to *extend* a provided reference SoC to customize it for a system context

- **This motivating system and therefore what you will do to extend is your choice.**
- **I am not going to tell you what system context and therefore what detailed block you should do.**
- Your Project Requirement is to Design, Integrate and Test the new capability to extend the reference SoC platform that meets the requirements of the chosen system context

ONE OF THE FIRST THINGS YOU NEED TO DO IS CO-CREATE YOUR SYSTEM
CONTEXT/MOTIVATION

This leads to the specific functionality you will design and integrate

Do something/application domain you are interested in learning about (e.g automotive -> CAN bus)

Dont make it over complicated you don't have a huge amount of time (e.g a full AI offload accelerator for Gen-AI_

Project Learning Outcomes for all projects

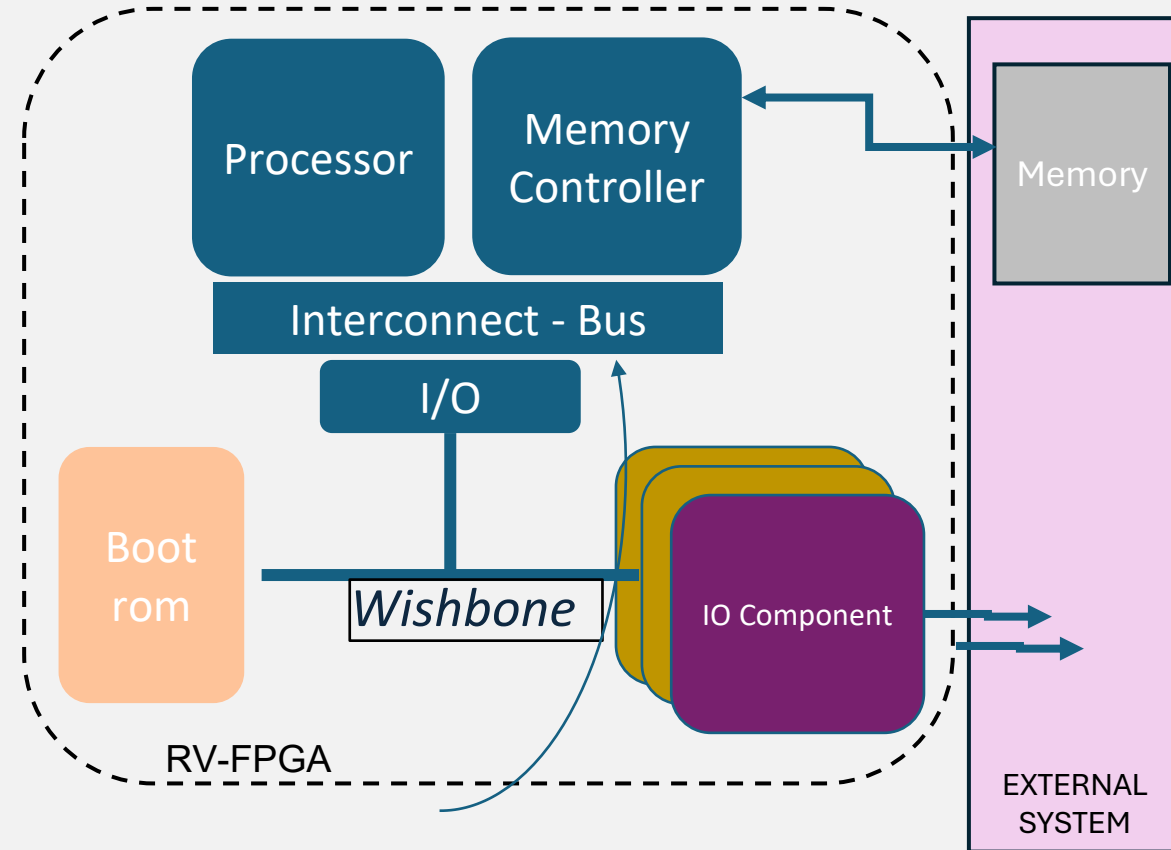
- You will be designing a specific (IP) Block that will extend the capabilities of the RV-FPGA platform to customize the platform to perform a specific system task
 - Each block is unique and has unique project objectives, but shares common **integration** challenges
 - You may work with other students in the cohort on these integration challenges and to understand the RV-FPGA SoC Architecture and the tooling environments
- By tackling this project you will gain the following reusable skills
 - Project Planning Management Research
 - Systems Architecture, SoC Architecture, Embedded Systems Architecture
 - Microarchitecture for digital processing blocks
 - HDL coding for SoC integration
 - Low-level C code programming to control hardware
 - Using a C compiler toolchain to generate assembly code
 - Using and FPGA synthesis tool to generate a **bitfile** that will *configure* a target FPGA

Possible Example Projects

- New i/o interface to allow 'connection' to another chip or system using standard or custom protocols
 - Specific protocol to connect to other systems chips eg I2C/I3C
- Graphics or Display Driver
- Camera(s) or Sensor(s) acquisition
 - Can also include Data processing on the way in and or out
- Accelerators for signal processing or DSP offload processing
 - Turns repeated code loops into hardware functions (go faster use less power take less energy)
 - Matrix Multiplication – Image Filtering etc etc
- Extend the capability of the platform itself
 - Security – protect from side channel attacks
 - Add low power low energy management
 - Silicon Lifecycle management or Telemetry

System on Chip (SoC) Projects - Implementation

Your project will be *extending* an SoC example – Rvfpga to perform a specific system function



Typically the Reference Platform is implemented in a single SoC chip or FPGA

Other components would be on a board see Slide 3

RV-FPGA is a reference **SoC Architecture**

It is a complete functional set of RTL (Verilog HDL) which describes the structure and behavior of an example SoC (Think of this like it's a *soft* R-Pi or Arduino)

It *instantiates* a SWERV RISC-V Processor Core

This is a micro-architectural implementation of the RISC-V Instruction Set Architecture (ISA) (You can write then compile a program to target this)

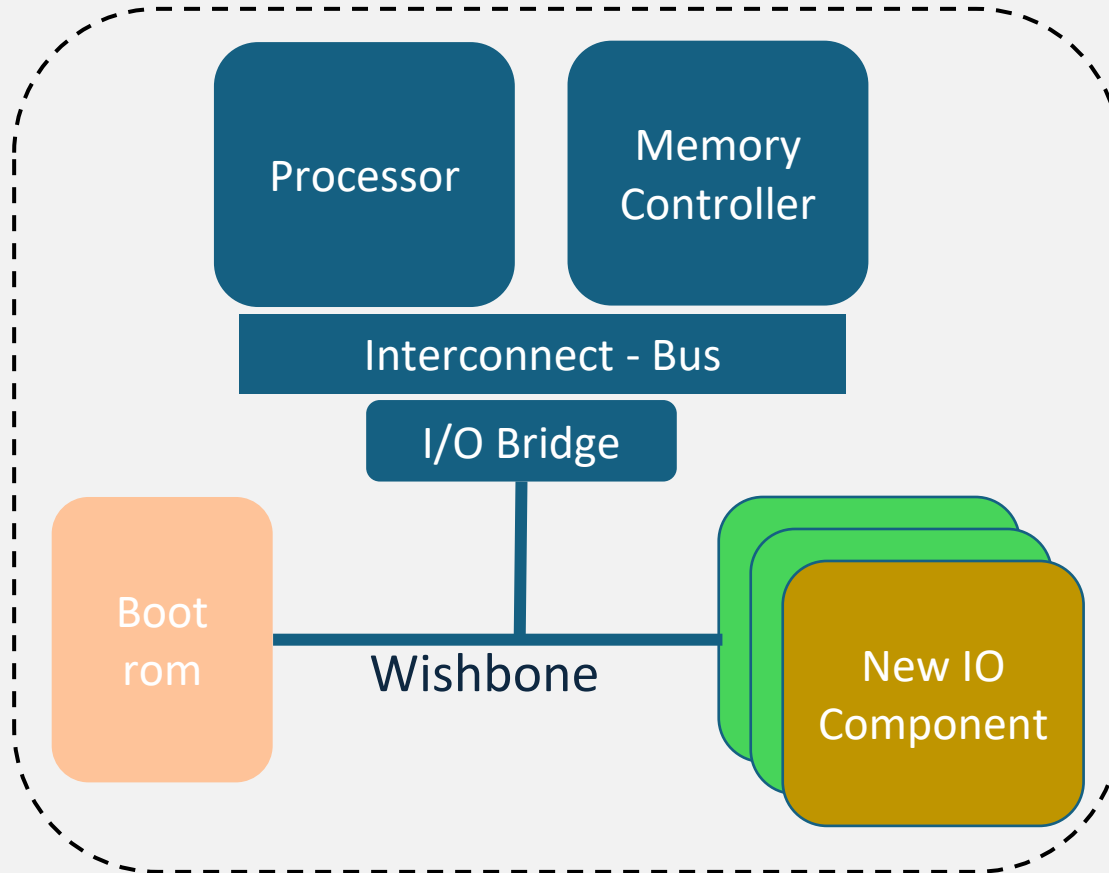
The example (which you will extend) will run code that is compiled and loaded into its memory

The design can be *simulated* as a Verilog HDL model

Or it can be **emulated** on an FPGA boards. You can test your design in a system context using the FPGA board

Example System on Chip Projects - New I/O Component

New I/O Component to perform a specific interface, control or sensing function



Existing i/o
component

New I/O Component (to implement a specific function unique to your project) can be added to SwervolfCore to extend the RVFpga functionality

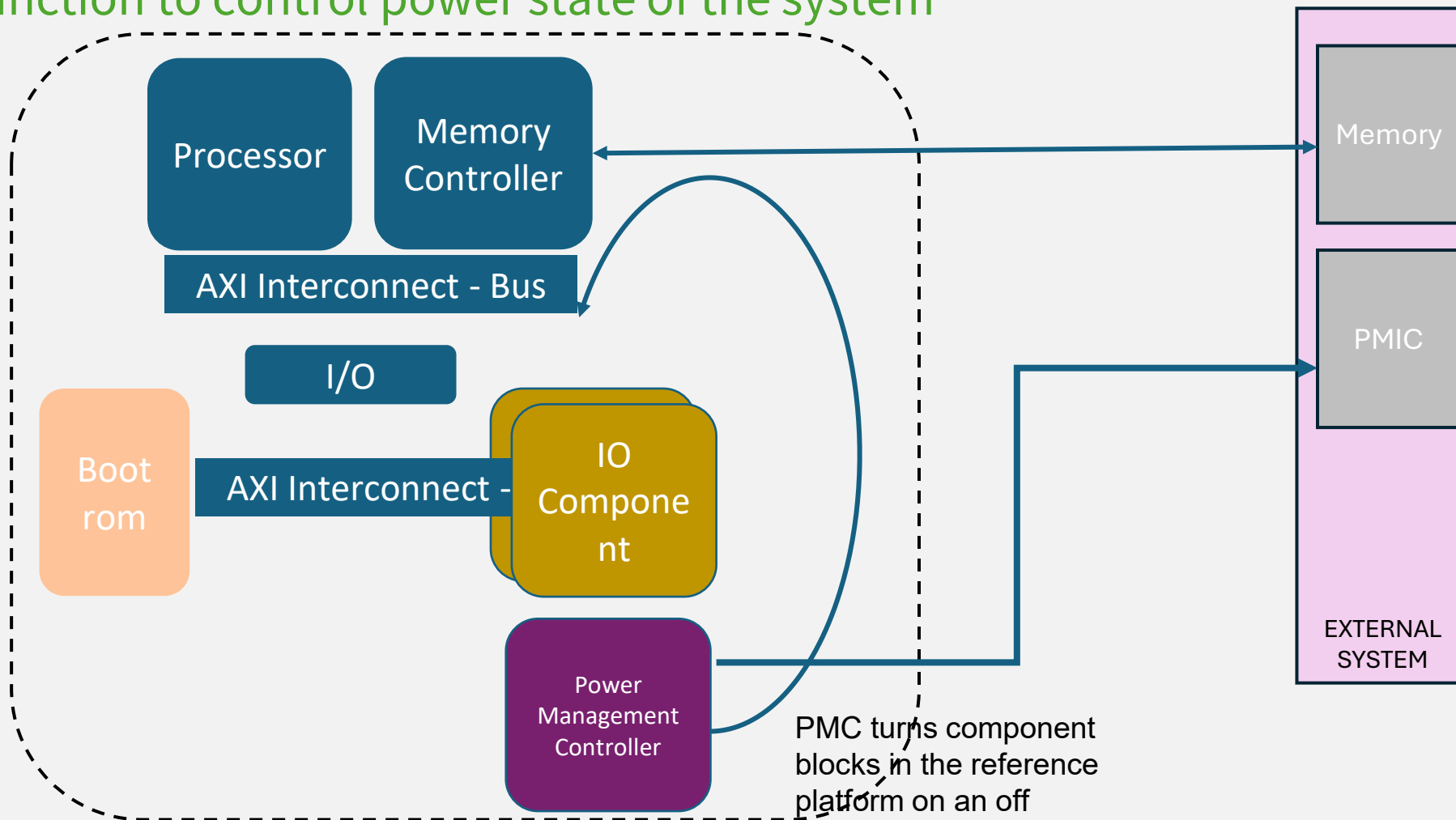
This will require:

- A core I/O Component to be designed
- Wrapping in a Wishbone Interface
- Integration into the memory space of the RVFpga (I/O mux needs to be extended)
- Code to be written to access, test and control the I/O device
- Implementation on to FPGA and tested

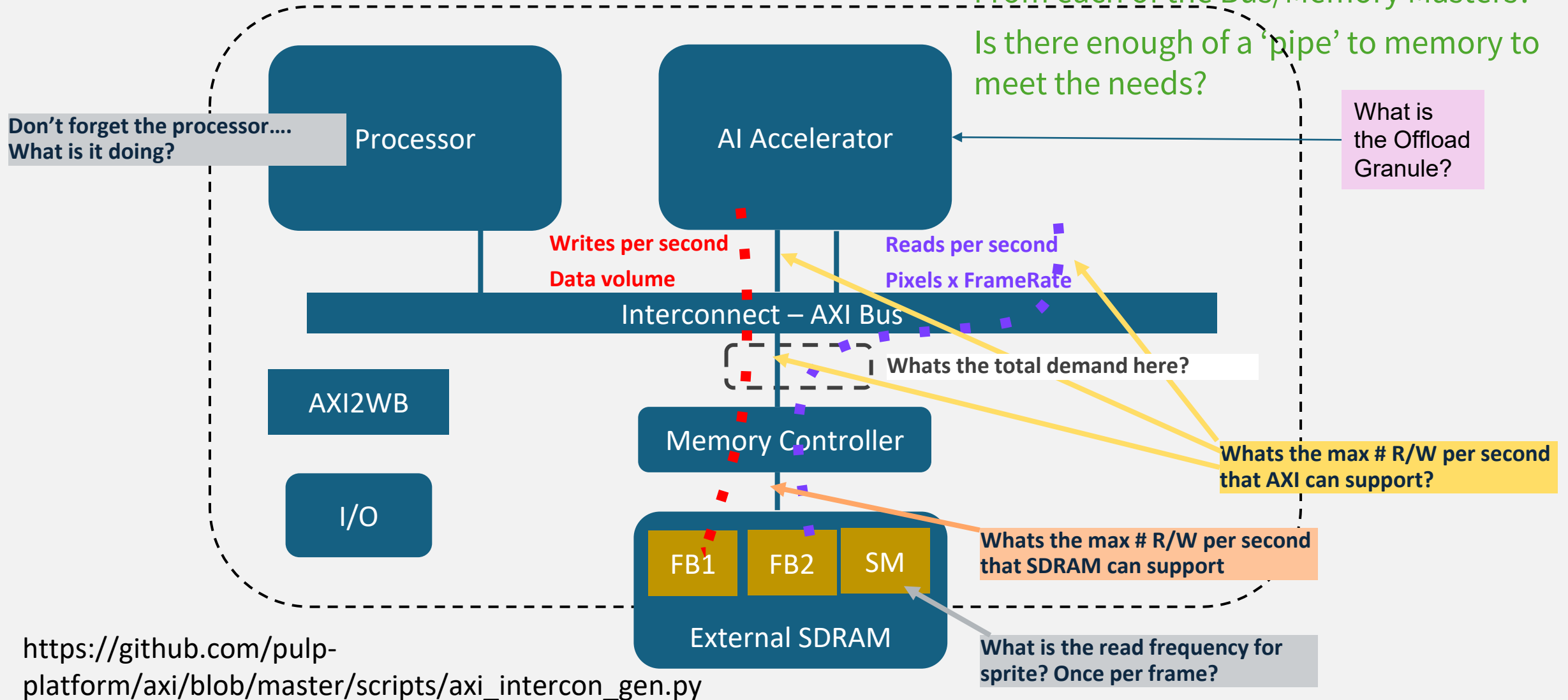
https://cdn.opencores.org/downloads/wbspec_b3.pdf

System on Chip (SoC) Projects - Power

Your project will be *extending* an SoC example – Rvfpga to have new specific system function to control power state of the system



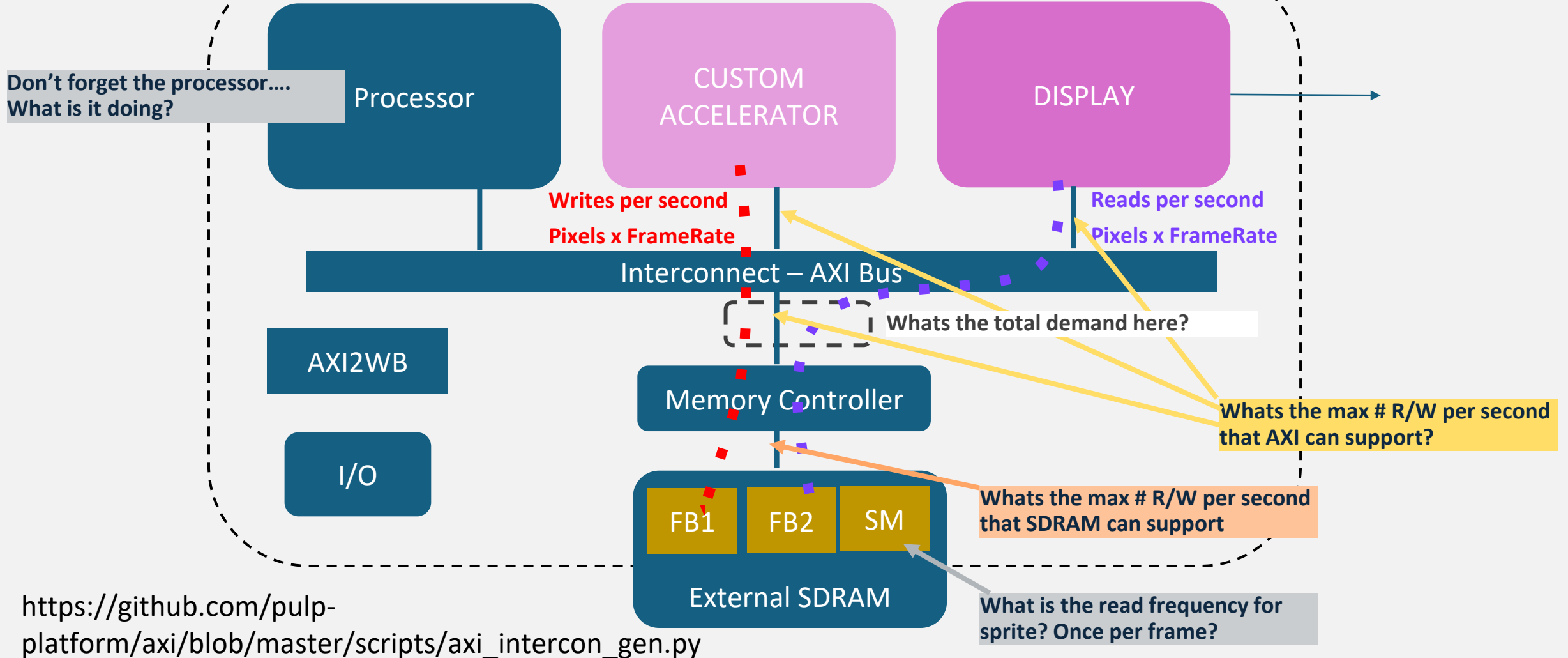
Example: SoC Project: Video/ Image (Also Sensor Fuse)



Example: System on Chip Project: Video/ Image

What is the Read and Write Bandwidth From each of the Bus/Memory Masters

Is there enough of a 'pipe' to memory to meet the needs?



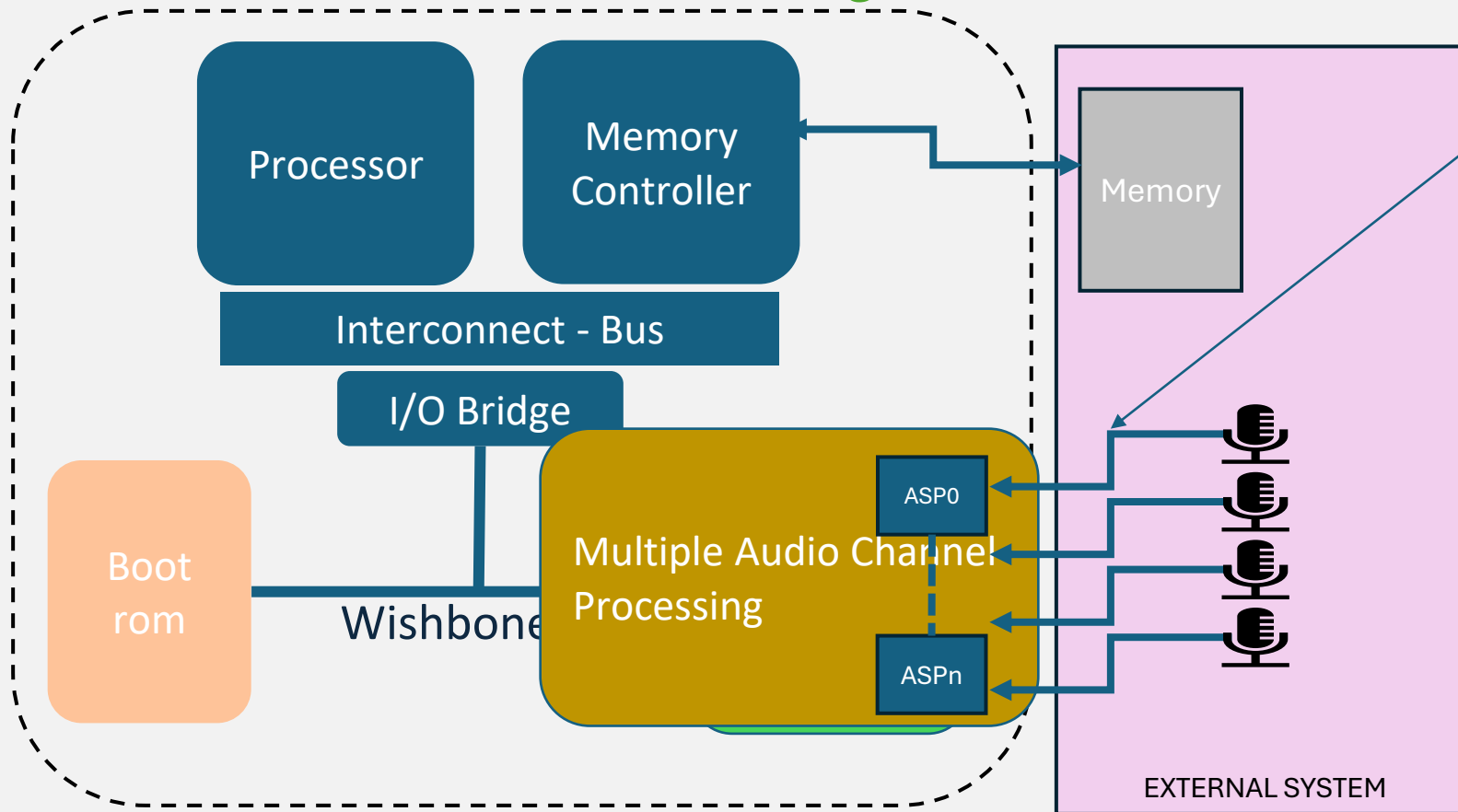
Add Leyuns Audio Example here

Example System on Chip Projects

Sensor Fusion

New I/O Component to perform a specific interface, control or sensing function

- Audio



Will need to figure out A/D for real world test

ASP Pipeline Configurable for up to n channels

To Do

Partition Processing between processor and the ASP (audio stream processor element)

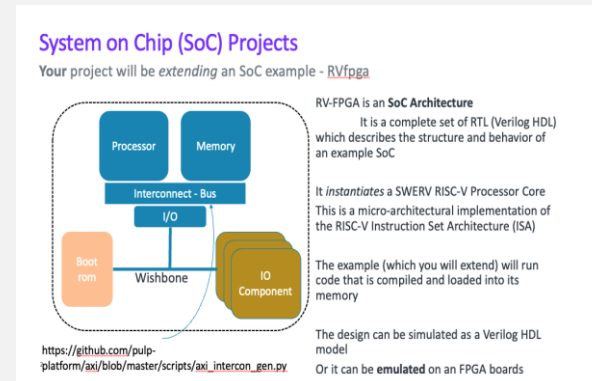
Existing i/o component

You will need to do 4 things to be successful

- **Plan** and maintain a plan for your project. – This is the most important one! (LOGBOOK)
 - Your plan will have both assessed and technical deliverables in it & risks (resk register).
- **Produce** the *Assessed Deliverables*
 - *PID, Presentation, Report*
- **Prepare** for the technical project
 1. Research and understand the details of the specific function of the block you are integrate. its systems context and system requirements
 2. Understand the SoC Design that you are going to integrate into
 3. Understand the toolchains you will use to create, integrate and test your design
- **Execute (Technical Deliverables)**
 - Write a Requirements Spec and **TestPlan**
 - Produce Microarchitecture and write specific code for your block
 - Architect how your block will integrate with the SoC
 - Implement and Test your block in isolation (Unit Leve Test)
 - Modify the provided SoC Design example to integrate
 - Test the integration in simulation

You can work on these three parallel with the lab excercise have labs

hint: if you don't do this, the integration may be hard to debug when it goes wrong!



Risc-V FPGA Training Collateral for Students

There is a complete set of example code and tutorials here

<https://drive.google.com/drive/folders/1Lq4r6CpMHuKSOkQQUY60YUluoHZ0jMWF>

<https://github.com/orgs/Microelectronic-Group-MISS-TUoS/repositories>

- You will also eventually need a Xilinx Nexsys A7 board but you can start without one
- Work through the tutorials 1 -10
 - **You do not need to deal with tutorials 11-20**
 - You do not need to understand tutorials 3 and 4 in detail but they will help a lot
 - Tutorials 6 – 10 are the important ones – these involve modifying hardware
- You **do not** need to know any internal details of the SWERV EH1 Core

- There are a set of slides in the directory
 - <https://docs.google.com/presentation/d/1KFzf6l5HYoGhUd-b-rGKUY8hMdkW40aG/edit?usp=sharing&oid=110995713148914427084&rtpof=true&sd=true>
- And a guided workbook
- NEW SUGGEST YOU USE THE GITHUB REPO!!!!

CHECK THESE
OUT BUT ASK
WHERE THE NEW
MATERIALS ARE!

RVfpga Labs Overview



Part 1: Labs 1-10

- Programming
- Vivado Project & I/O Systems

Part 2: Labs 11-20

- RISC-V Core
- RISC-V Memory Systems
- RISC-V Benchmarking & Performance Monitoring

All labs include **exercises** for using and/or modifying the RVfpga System to increase understanding through hands-on design. RVfpga includes C and assembly example programs and solutions.

  RVfpga v2.2 © 2022 <26> Imagination Technologies

IMPORTANT NOTES!

These labs were made as part of a 'complete module' which included assessed exercises

- **THE LABS ARE TO HELP YOU LEARN ABOUT THE ARCHITECTURE OF THE THE SoC and to familiarize yourselves with the *toolchains* you will use to complete your project.**
 - **YOU DO NOT NEED TO SLAVISHLY GET THE CODE 'right' or pass the assessed parts.**
 - **You DO need to understand whats going on though If you don't ASK**
 - **E.g lab 1. if you understand how to write new c code and compile it you don't need to do all the sub parts**
- You are free to collaborate with your peers on the labs
- Don't panic! there is quite a lot of new material but it can be worked through methodically
- It is vital that you ask questions (verbally or written) if you don't understand
- What the labs will do is help you build your engineering platform which you use to build (develop) your final project

Labs

- You may end up writing C or .asm code to test your integration
- You need to understand the tool chain to build an executable code image and load it on *both* the simulation model or FPGA board
- If you end up writing more complex integration code lab 3 & 4 will help with code structure

RVfpga Labs 1-4: Programming

- **Lab 1: C Programming:** Write a C program in PlatformIO, and run / debug it on RVfpgaNexys/RVfpgaSim/Whisper. Also introduce Western Digital's Board Support and Platform Support Packages (BSP and PSP) for supporting operations such as printing to the terminal.
- **Lab 2: RISC-V Assembly Language:** Write a RISC-V assembly program in PlatformIO and run /debug it on RVfpgaNexys/RVfpgaSim/Whisper.
- **Lab 3: Function Calls:** Introduction to function calls, C libraries, and the RISC-V calling convention.
- **Lab 4: Image Processing: C & Assembly:** Embed assembly code with C code.

RVfpga Labs 5-10: I/O & Peripherals

- **Lab 5: Creating a Vivado Project:** Build a Vivado project to target RVfpgaNexys to an FPGA board and simulate RVfpgaSim in Verilator.
- **Lab 6: Introduction to I/O:** Introduction to memory-mapped I/O and the RVfpga System's open-source GPIO module.
- **Lab 7: 7-Segment Displays:** Build a 7-segment display decoder and integrate it into the RVfpga System.
- **Lab 8: Timers:** Understand and use Timers and a Timer controller.
- **Lab 9: Interrupt-driven I/O:** Introduction to the RVfpga System's interrupt support and use of interrupt-driven I/O.
- **Lab 10: Serial Buses:** Introduction to serial interfaces (SPI, I2C, and UART). Show how to use the onboard accelerometer that uses an SPI interface.

- You will definitely be synthesizing your code to target an FPGA as part of the project
- You will have to create new HDL code and modify some of the existing examples
- Labs 6 through 10 will help you understand integration

Labs

- Lab 6&7 GPIO This is an example of a *simple* functional block.
- Make sure you understand the concept of the Memory Mapped (Control & Status) Registers for the I/O Peripheral. The Processor loads (reads) or stores (writes) to a specified memory address this causes data to be written to that specific memory address through a **bus transaction** on the AXI and Wishbone buses
 - Review the concept of overall memory map for the SoC
 - Review the concept of bus masters, bus transactions and bus protocols
 - Tool Usage – Pin Mapping going from a Named HDL Signal to a specific Pin on a Package
 - Pin to component connection on PCB
- Lab 8 is a more complex RTL block performing a specific function (timer)
 - Reinforce the control registers that determine how the independent hardware block functions
 - You will need to decide (architect) how you will split functionality between hardware and software
- Lab 10 looks at how to integrate your block using Interrupt Driven rather than a polled status
 - This is how you get the processor to respond to an asynchronous (external) event

The Labs WONT provide you with everything you need

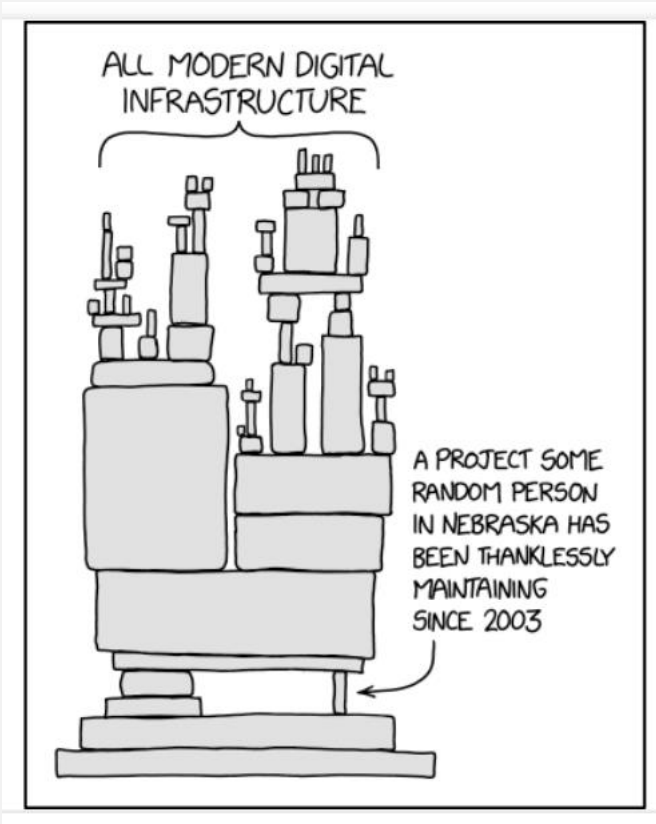
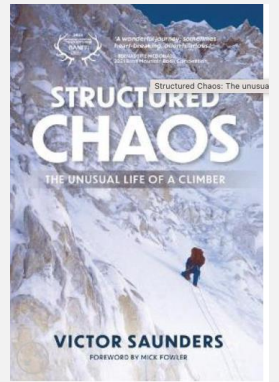
It is important that you ask 'what is different' about your project

- They do not cover AXI bus integration
 - Depending on your project you may need to do this
- They do not cover Writing good Unit level or System level testbenches
 - You will need to do both
- They do not cover how to start with a blank piece of paper and design good digital microarchitecture

A *very* important note on complexity.....

The Computer you run on the tools you use –
there are choices – none of them are perfect

Uniquification – your
Project is Unique



- IP Mix
 - SW Mix
 - Design Methodology
 - Tool Chains
 - Target Technology/Cost
 - Service Lifetime
 - Team (You)
 - Supply chain (The internet)
 - Budget (Time)
- Design Patterns
 - Problem Solving Approaches
 - Abstractions
 - The labs are one way to do things others exist

Platform.io helps
manage
configurations of
tools

Whats Going on: High Level

You will need to be familiar with Verilog to understand the design and code you block

- If you have are not familiar with Verilog you should review prior material. EEE232 for example
- You can also find lots of material on the web
- The Labs will equip you with two key things
 - An Understanding of the Architecture of a Processor Subsystem
 - An understanding in the use of the toolchains used to
 - Write Verilog
 - Simulate Verilog
 - Synthesise Verilog to FPGA
 - THEY WILL NOT HELP YOU WITH YOUR UNIQUE SYSTEMS CONTEXT OR HOW TO ARCHITECT YOUR UNIQUE FUNCTIONALITY

Important notes about the Labs

– They are not a test!

- Lab1: Emphasis on the exercises that interact with hardware on the board. The specifics of the algorithm code are not important – assumed that if your project needs you to write some ‘clever C’ you will be able to do so later
- Lab2: The exercise of writing .asm to control the LEDs is important this is about how to interact at a bit/binary level with a peripheral and how that links through to physical hardware.
- Lab 3 & 4 are about how to program in C and in mixed C and assembler. The detail of what you are programming is less important – if you understand the toolchains and how to embed .asm that is fine. Don’t get too hung up on the implementation of the image transformation..... {although if your project involves graphics/display you may find it useful}
- Lab 5 is super important – this covers the Vivado toolchain for synthesis and also looks at simulating your HDL

Add a slide on TARGETS

Progress & Time

