



ECE 2031 Final Design Project

Introduction

ECE 2031 Spring 2020

Design Project Motivation



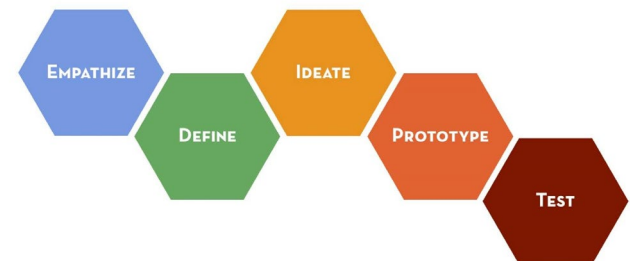
- ECE 2031 includes the sophomore-level team design experience
- You are developing a useful set of tools
 - eventually including an entire computer within the DE2 board
- Using tools creatively to solve problems is what engineers and computer scientists do

ECE 2031 Project Components

- Propose a solution to a problem:
 - What's the best approach for completing a given task within the constraints of the project?
 - More details next week on proposal
- Implement the proposed design so that it can be used on the “DE2Bot” in the future
- Demonstrate, present, and document your solution

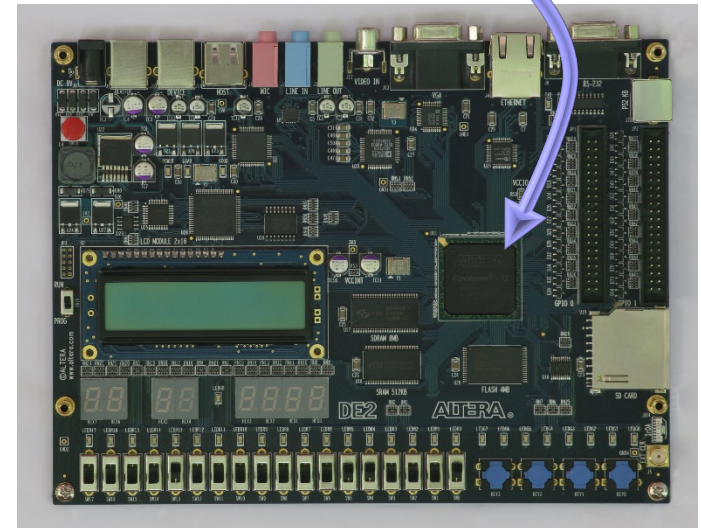
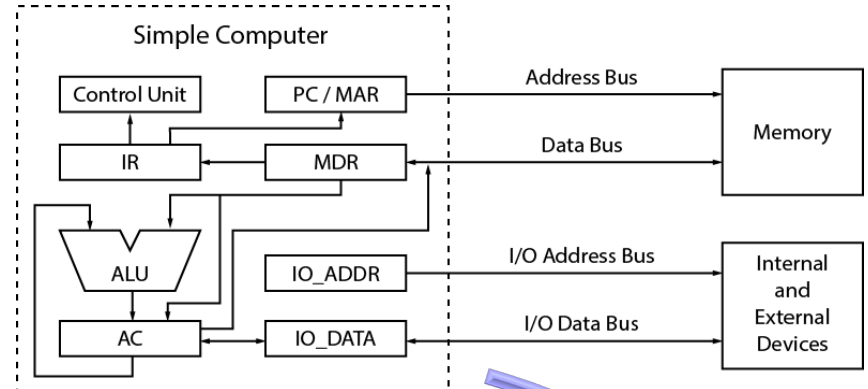
Open-ended Design

- Recall the very first lecture in 2031, about the design process
 - You will only need to do a little empathizing, but you will go through every other step
- You will also experience many other aspects of open-ended design
 - We will use some of today or next week to talk about limitations, debugging, and efficient design



The Simple Computer (SCOMP)

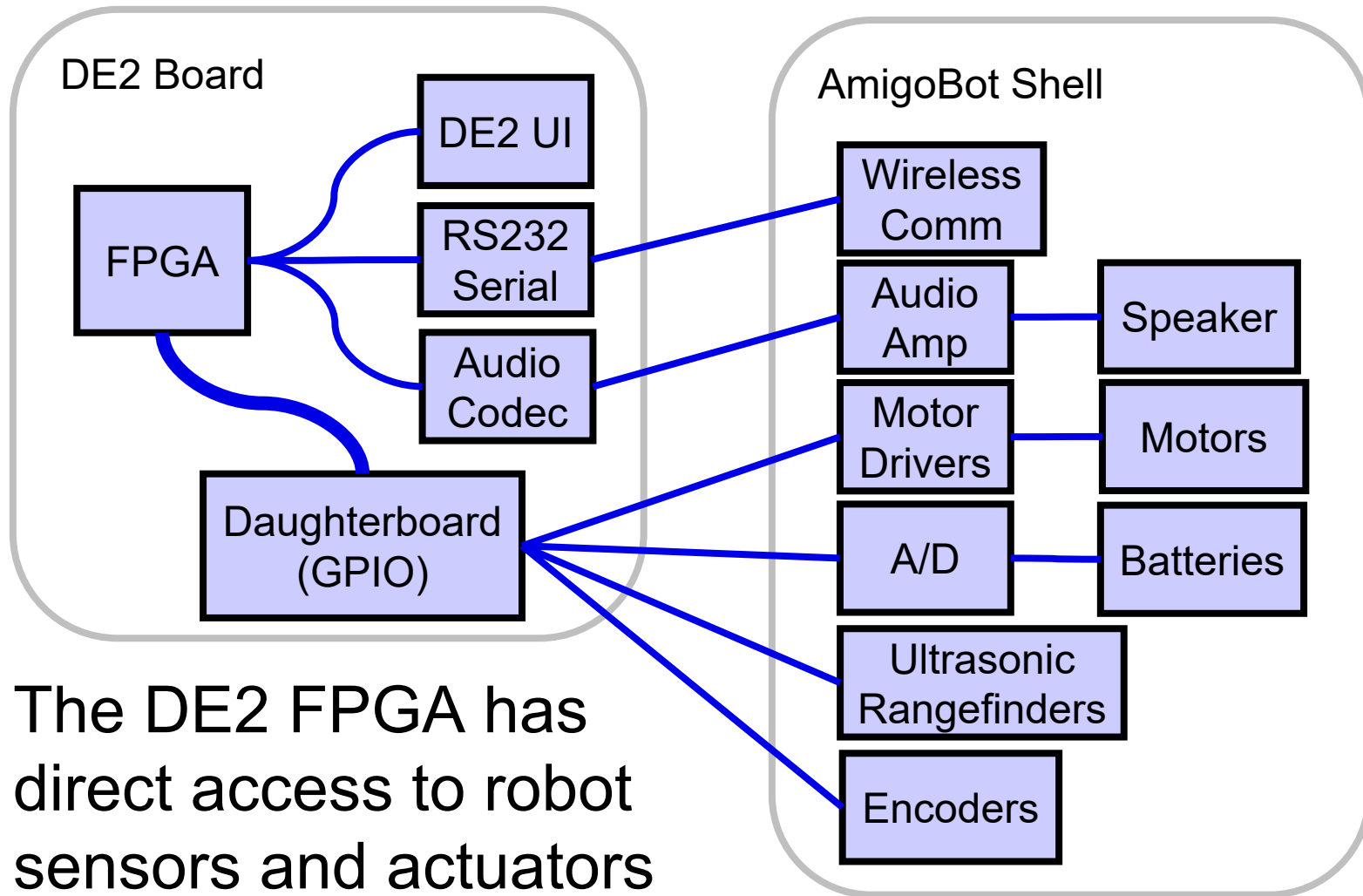
- In Labs 7 and 8, you implement a computer in the FPGA
 - Both hardware (a description of gates, flip-flops, etc. that form the computer)
 - And software (various programs that you load onto your computer, in its RAM)



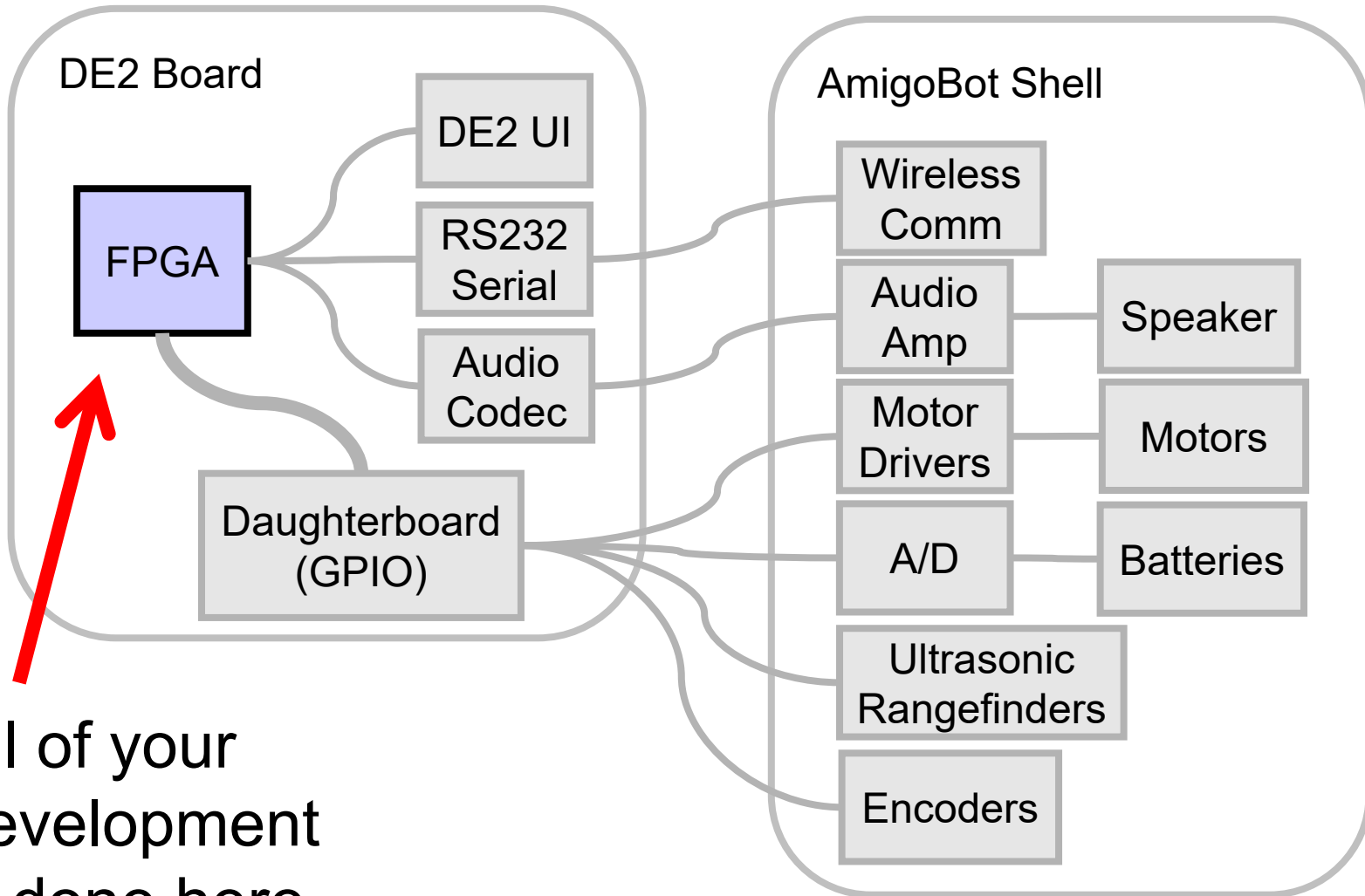
ECE2031 DE2Bot Past Projects

- Position/velocity feedback, and motor velocity control
- Processing of sonar transducers
- I²C interface for battery monitoring and audio codec control
- Odometry (position estimation from wheel rotation)
- Audio codec interface and digital sound generation
- Robot Self-test
- Infrared signal detection and “remote control” demonstration
- UART for serial communication
- Implementation of hardware interrupts for SCOMP
- Complex mathematical functions in software (ATAN)
- Analyzing sonar data to locate objects and make contact
- Many sensing, localization, and navigation demonstrations

DE2Bot Hardware Architecture

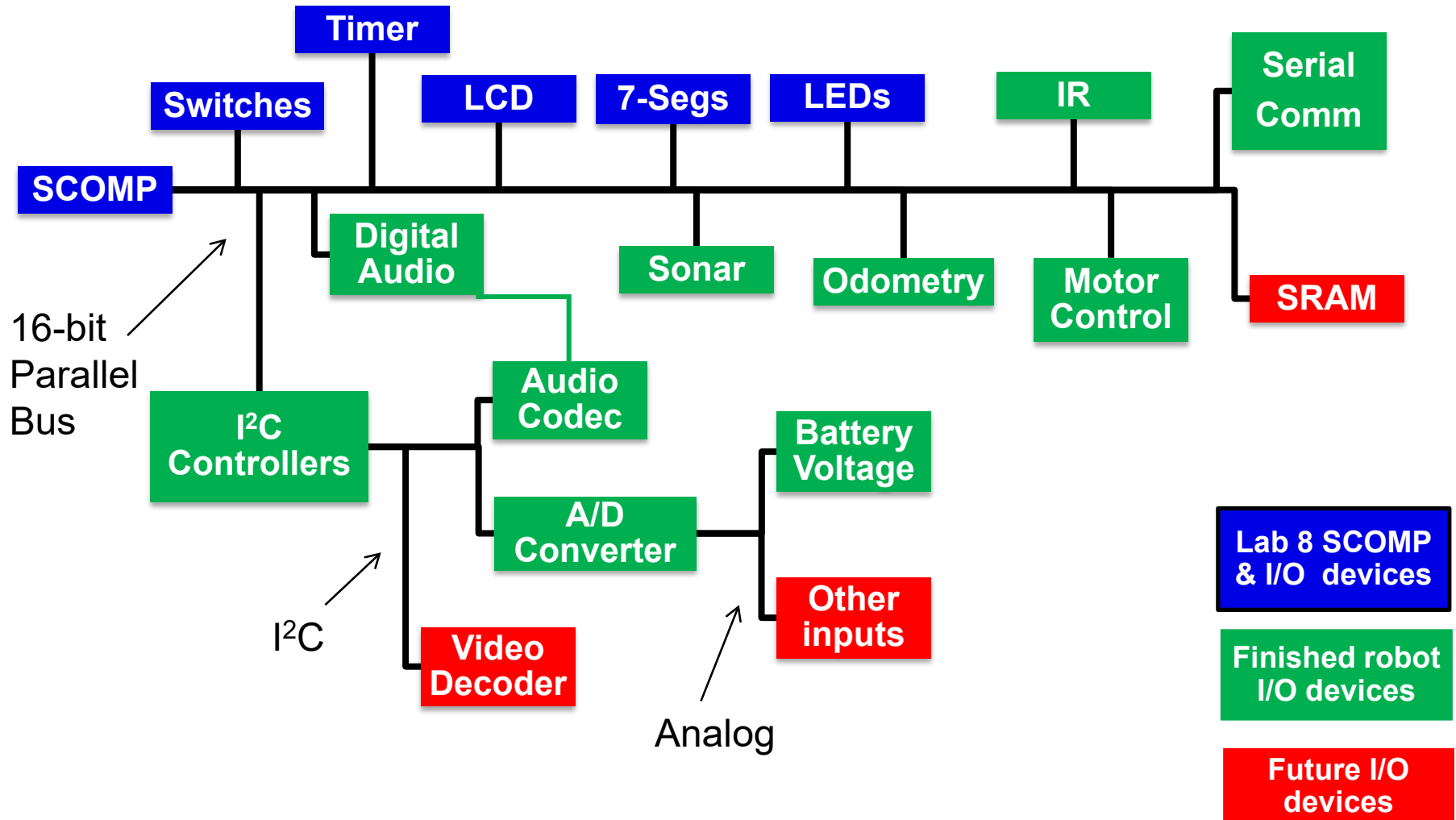


Project Development



All of your
development
is done here

DE2 and FPGA System Architecture

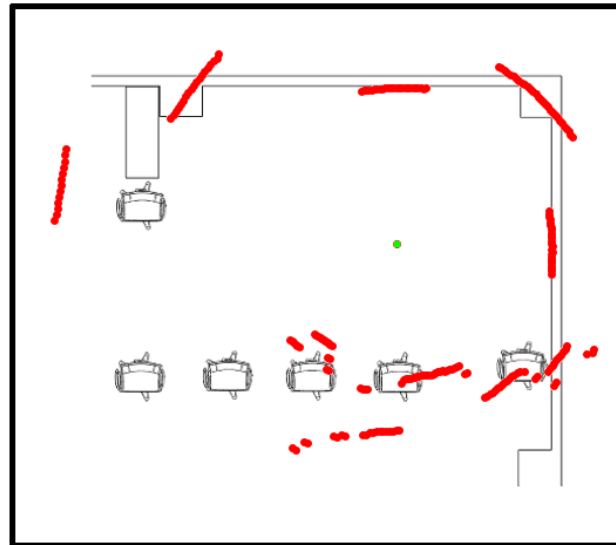


Your Design Task for Spring 2020

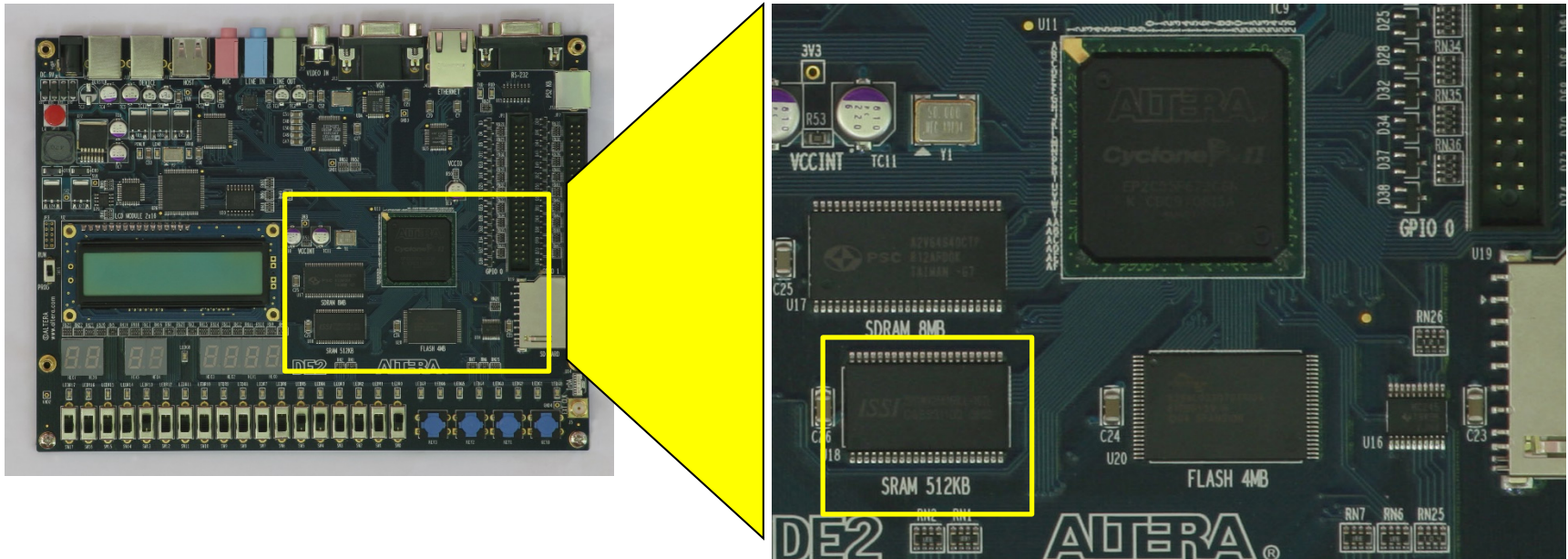
- Create an I/O peripheral for SCOMP that interfaces with the SRAM chip on the DE2
 - This will give future 2031 students 256k words of data memory – 128x what they currently have
- You will need to design the I/O interface
 - How does the user set the address?
 - How does the user control reads and writes?
 - You can use multiple SCOMP I/O addresses
 - Addresses 0x10 through 0x1F are reserved for you
- Speed and ease-of-use are desirable

Project Justification

- There have been projects where it has been useful to acquire lots of data during runtime.
- SCOMP's small memory (2k words during the project) have limited those projects.
- Example: Saving sonar scans to create a map of the area



SRAM on DE2 board



- This is NOT the on-chip FPGA memory you've been using

SRAM-FPGA Interface

- All 39 of the SRAM chip's signals connect directly to the FPGA
- The DE2 manual provides the mapping of SRAM pins to FPGA pins

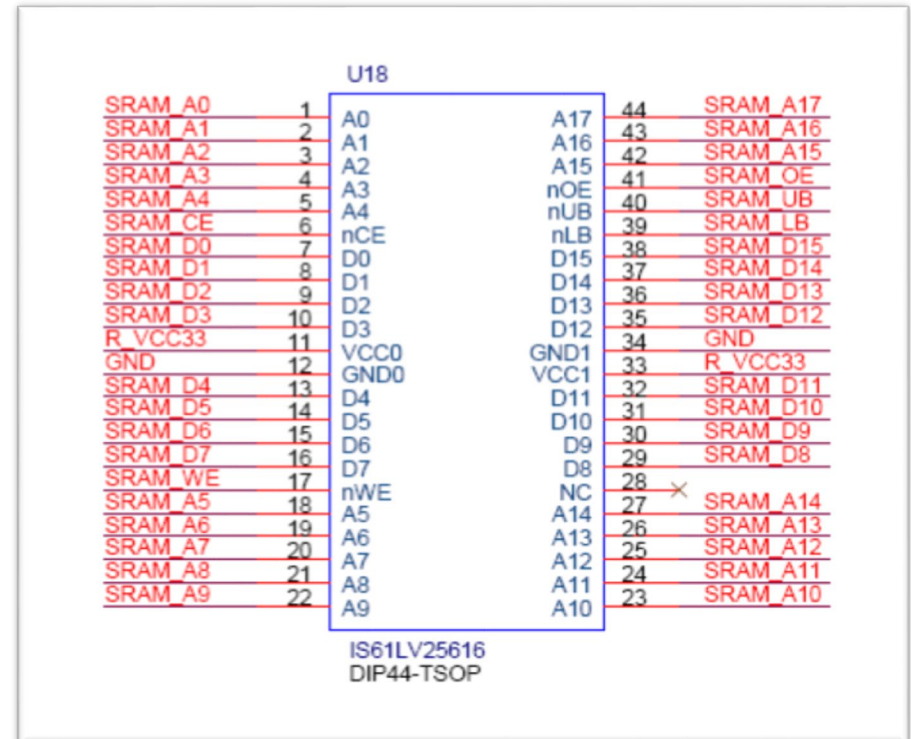
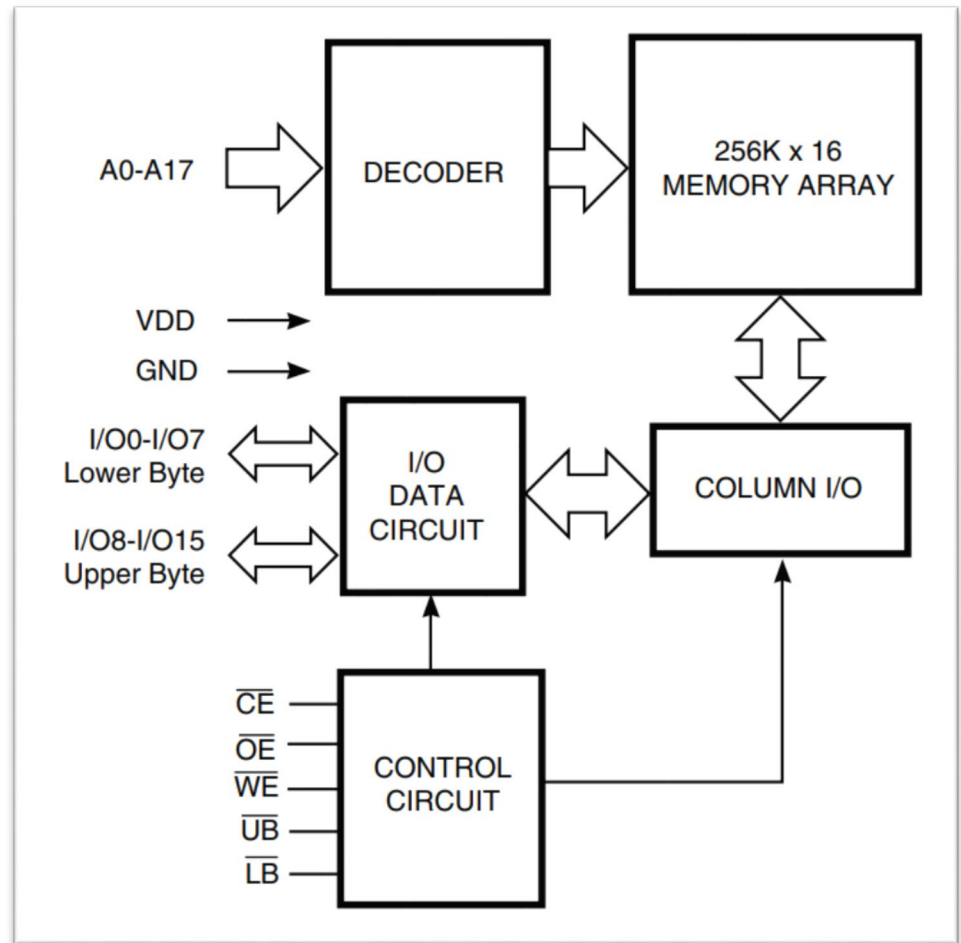


Image source:

Terasic DE2 Board User's Manual

SRAM Functional Block Diagram

- The SRAM chip has 18 bits of address to access 256k 16-bit words
- There is 8-bit word functionality, but you won't need it
- Of the five control signals, you will need /OE and /WE.



Control signal functionality

IS61WV25616ALL/ALS, IS61WV25616BLL/BLS,
IS64WV25616BLL/BLS

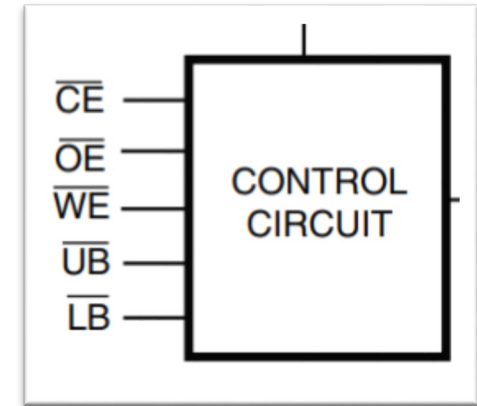


TRUTH TABLE

Mode	\overline{WE}	\overline{CE}	\overline{OE}	\overline{LB}	\overline{UB}	I/O PIN		V_{DD} Current
						I/O0-I/O7	I/O8-I/O15	
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	I_{CC}
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	DOUT	High-Z	I_{CC}
	H	L	L	H	L	High-Z	DOUT	
	H	L	L	L	L	DOUT	DOUT	
Write	L	L	X	L	H	DIN	High-Z	I_{CC}
	L	L	X	H	L	High-Z	DIN	
	L	L	X	L	L	DIN	DIN	

Control Signal Description

- Remember: “**active/inactive**” is not the same as “**high/low**”. All of the control signals are active-low.
- /CE is “chip enable”. You can set this to always active.
- /OE is “output enable”. It controls the chip driving its outputs, which it needs to do during reads.
- /WE is “write enable”. It controls whether the chip is in read or write mode (and overrides /OE).
- /UB and /LB control the upper and lower bytes, which you won't need to do (leave both active).



Notes on Signal Timing



- Violating timing requirements can at best corrupt your memory, and at worst damage the chips.
- All of the timing requirements for this chip are <10 ns, so a sequence of operations using the 50 MHz or 30 MHz clock will be safe.
- A sequence of operations based on SCOMP instructions will be *very* safe.
- Changing things together or in the wrong order are what you need to be careful of.



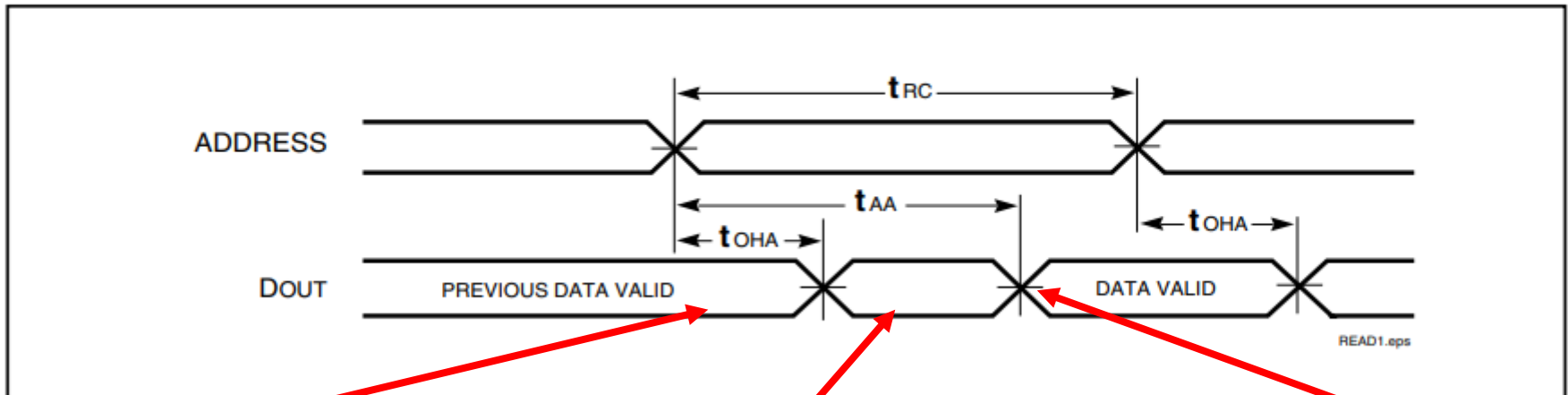
A Super Safe SRAM Read

1. Make sure the FPGA is not driving the SRAM data lines and wait a while.
2. Set the SRAM address, drive /OE low, and wait a while.
3. Latch the data that the SRAM is putting out to a register in the FPGA (e.g. into SCOMP's AC).
4. Deassert /OE so that the SRAM stops driving its data lines.

SRAM Read Timing with Address

This diagram explains what happens if you leave /CE and /OE active and change the address.

READ CYCLE NO. 1^(1,2) (Address Controlled) ($\overline{CE} = \overline{OE} = V_{IL}$, \overline{UB} or $\overline{LB} = V_{IL}$)

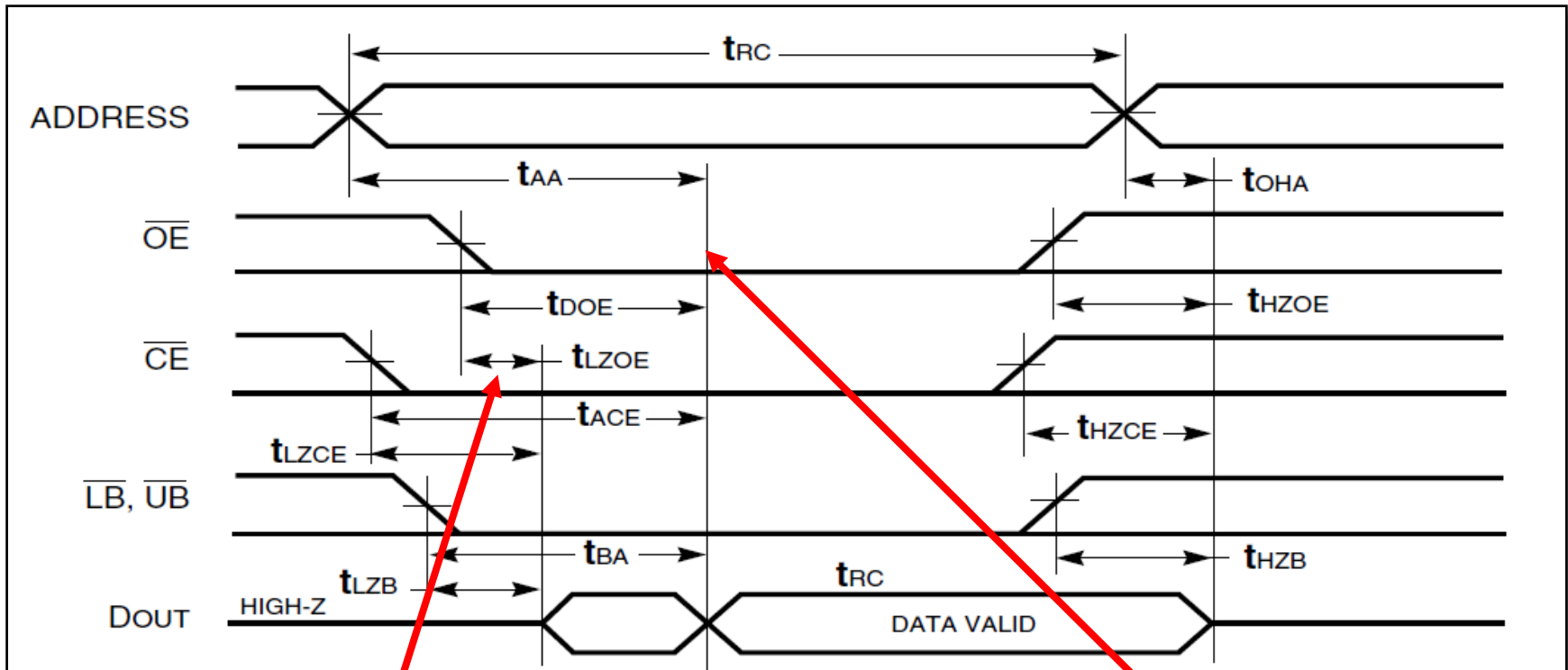


Old data remains on the outputs for t_{OHA}

There is a period of time with indeterminate outputs

Data becomes valid t_{AA} after the address changes

SRAM Read Timing with /OE



The outputs will be high-Z for t_{LZOE} after /OE becomes active (assuming /CE, /LB, and /UB are active)

You must wait for both t_{AA} and t_{DOE} to be met before data is valid.

Specific SRAM Read Timing

1. Make sure the FPGA is not driving the SRAM data lines at least t_{LZOE} before asserting /OE.
2. Set the SRAM address, and drive /OE low.
3. Wait at least t_{AA} after the address changed and at least T_{DOE} after /OE changed, whichever requires a longer wait.
4. Latch the SRAM data to a register in the FPGA.
5. Deassert /OE so that the SRAM stops driving its data lines (not actually necessary as long as the next operation is safe).

A Super Safe SRAM Write

1. Drive the address lines, (optionally) deassert /OE, and wait for a while.
2. Assert /WE (drive it low), and wait for a while.
3. Drive the data lines from the FPGA and wait for a while.
4. Deassert /WE and wait for a while.
5. Stop driving the data lines from the FPGA.

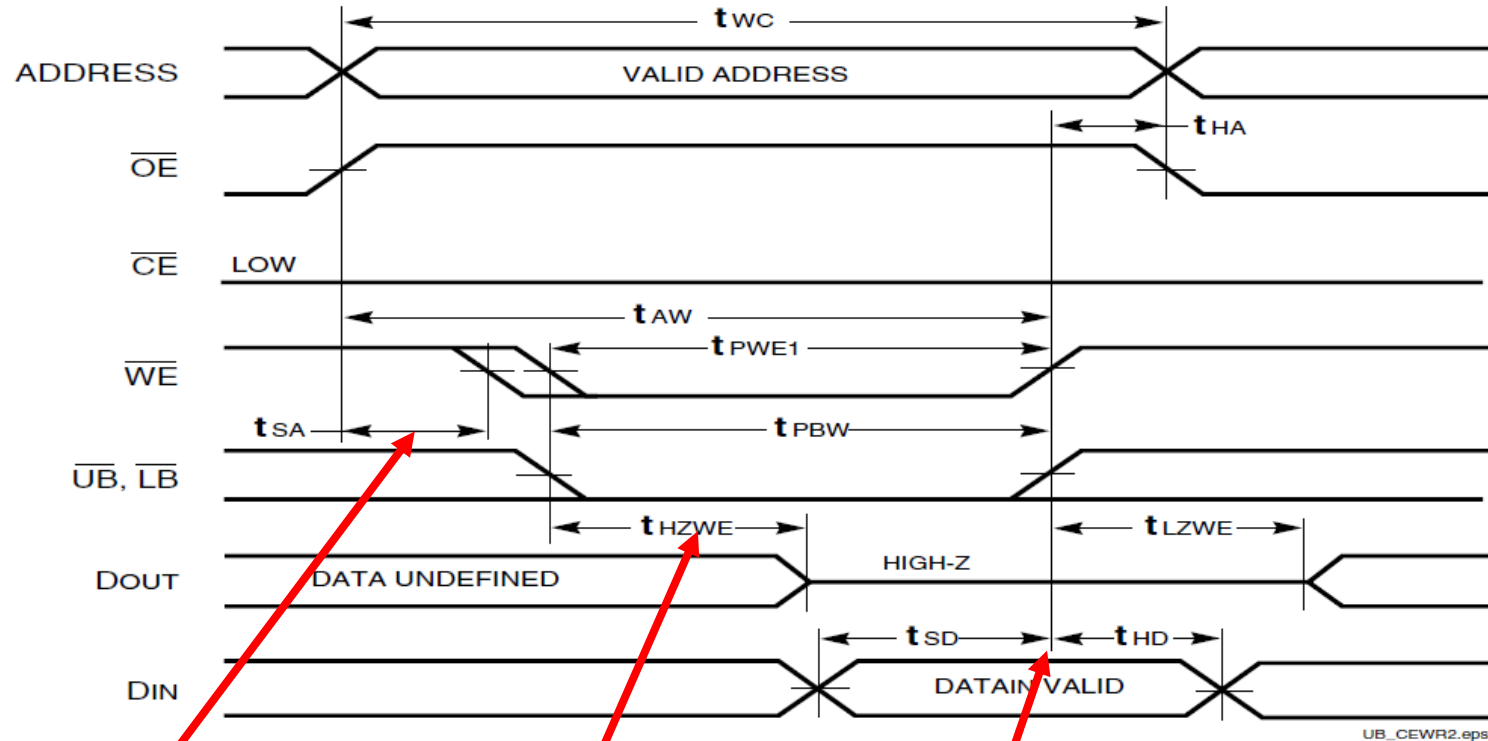
IS61WV25616ALL/ALS, IS61WV25616BLL/BLS,
IS64WV25616BLL/BLS



TRUTH TABLE

Mode	WE	CE	OE	LB	UB	I/O PIN		V _{DD} Current
						I/O0-I/O7	I/O8-I/O15	
Not Selected	X	H	X	X	X	High-Z	High-Z	I _{se1} , I _{se2}
	X	L	X	H	H	High-Z	High-Z	I _{cc}
Read	H	L	L	L	H	Dout	High-Z	I _{cc}
	H	L	L	H	L	High-Z	Dout	
	H	L	L	L	L	Dout	Dout	
Write	L	L	X	L	H	Din	High-Z	I _{cc}
	L	L	X	H	L	High-Z	Din	
	L	L	X	L	L	Din	Din	

Write Timing with /OE and /WE



Change address and wait for it to stabilize (address setup time).

Assert /WE and wait for data lines to be high-Z.

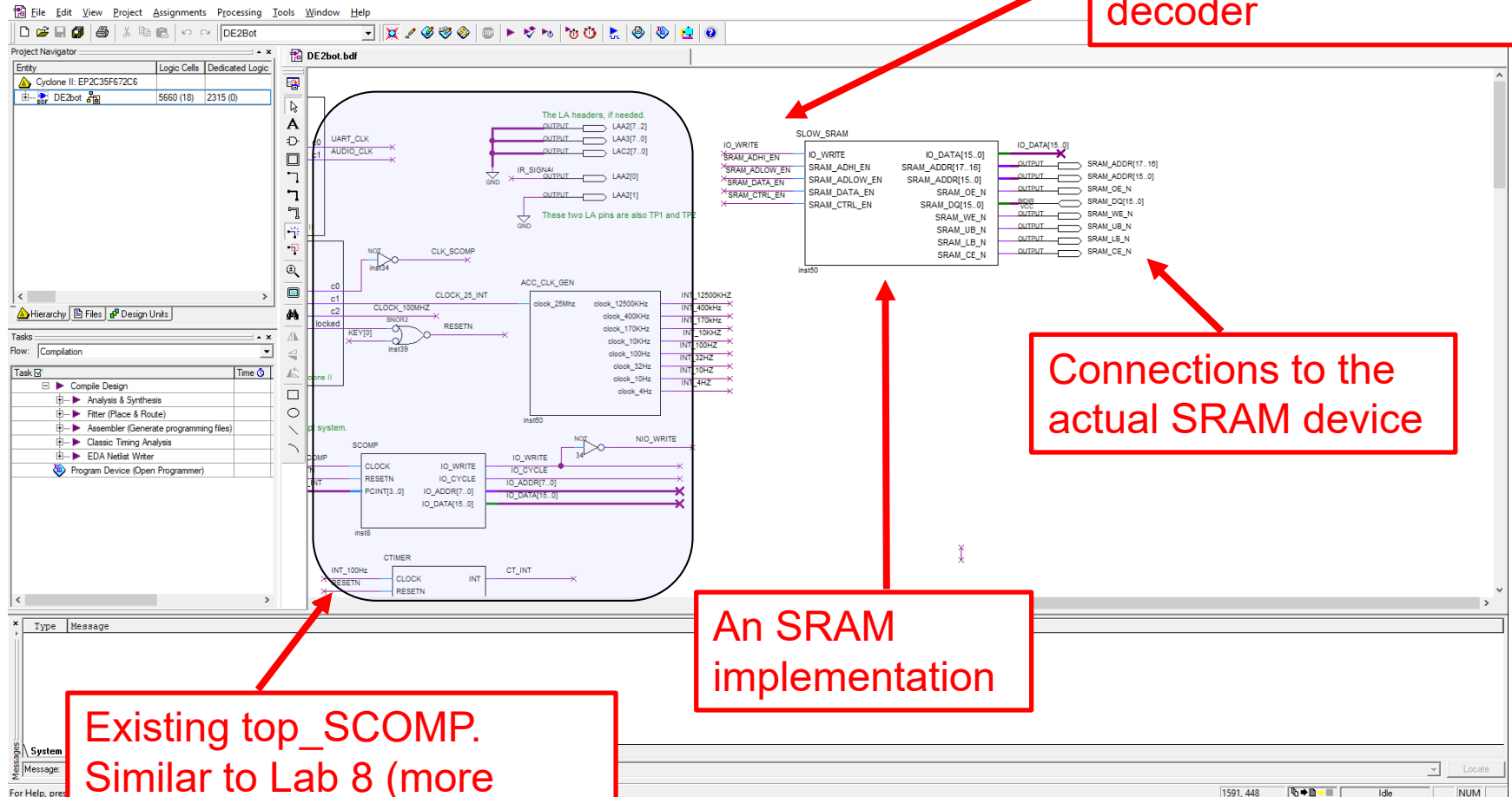
Drive the data and deassert /WE without violating setup or hold time.

Specific SRAM Write Timing

1. Drive the address lines and wait for the SRAM to “stabilize” on that address (t_{SA}).
2. Assert $/WE$, and wait for the SRAM to not be driving the data lines (t_{HZWE}).
3. Drive the data lines from the FPGA and wait for the SRAM’s data setup time to elapse (t_{SD}).
4. Deassert $/WE$ and wait for the SRAM’s hold time (address and data) to elapse (t_{HD} , t_{HA}).
5. Stop driving the data lines from the FPGA.

Interface with SCOMP

Signals from SCOMP and the I/O address decoder

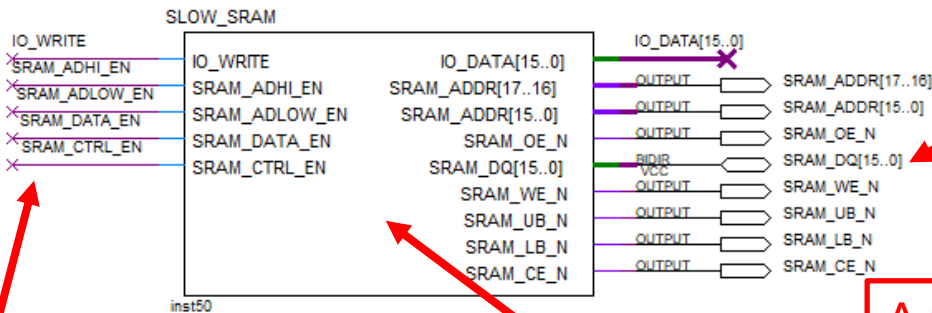


Connections to the actual SRAM device

An SRAM implementation

Existing top_SCOMP. Similar to Lab 8 (more not shown to left and down)

More detail of previous



Connections to the actual SRAM device

Signals from SCOMP and the I/O address decoder

A major part of your project is what you put in here.

- What you will have next week is a crude implementation, "SLOW_SRAM," done as a schematic.
- What you create will be better, and done as VHDL.

Inside “SLOW_SRAM”

IO_WRITE

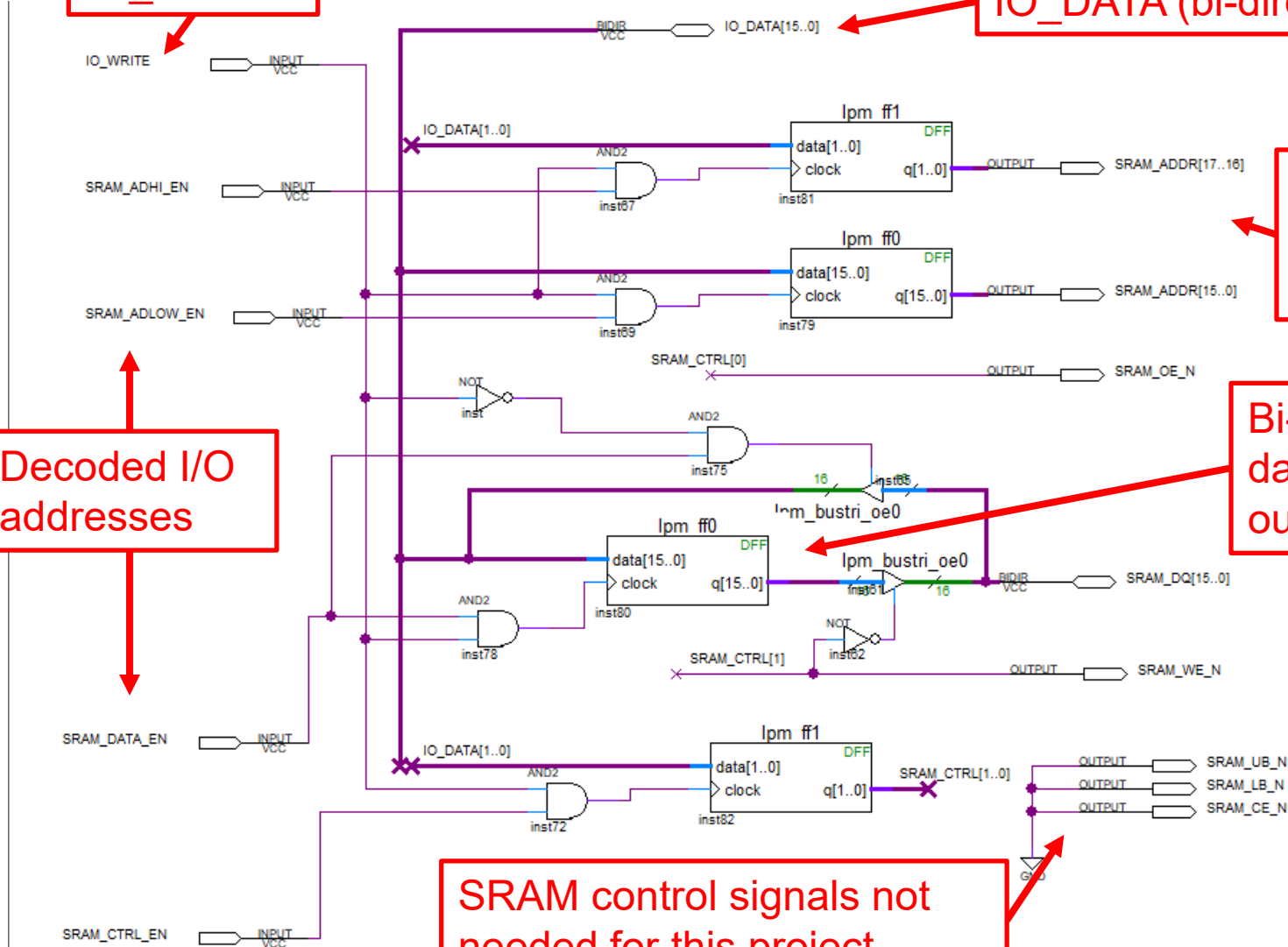
IO_DATA (bi-directional bus)

Latched address (2 registers)

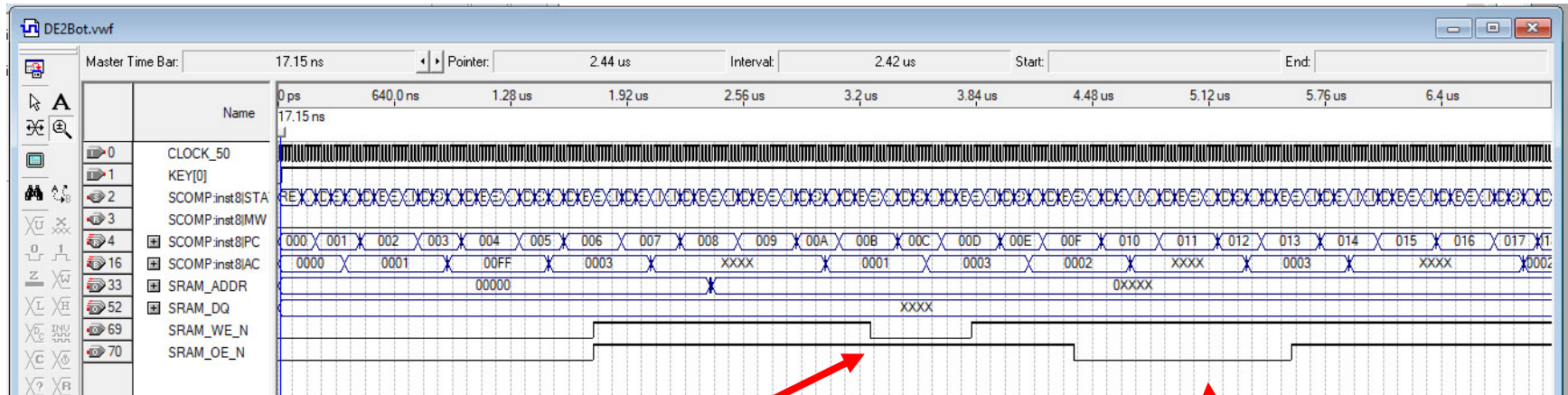
Bi-directional data path, with output register

Decoded I/O addresses

SRAM control signals not needed for this project



Operation of “SLOW_SRAM”

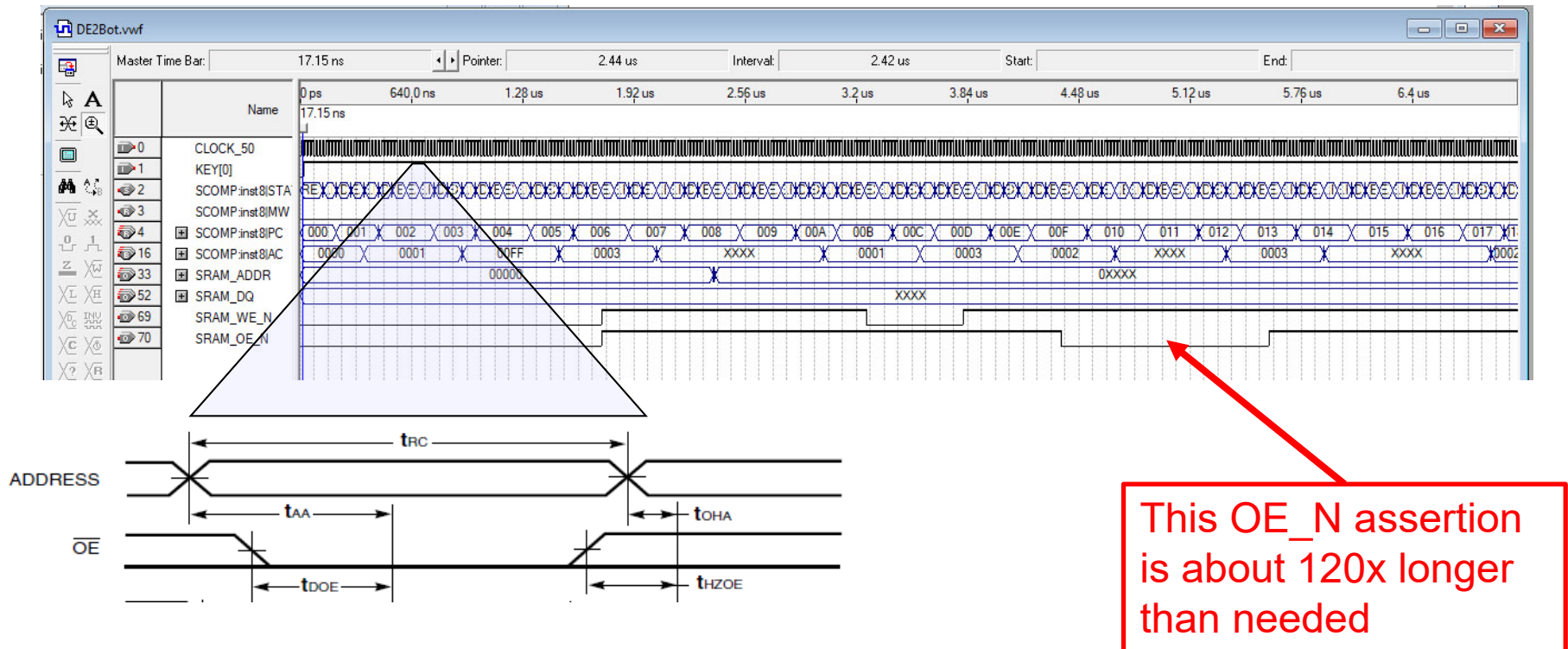


Write: SRAM_WE_N
active for about 0.56 μs (7
SCOMP_CLK periods)

Read: SRAM_OE_N active for about
1.2 μs (15 SCOMP_CLK periods)

- This implementation of the device requires multiple SCOMP instructions to latch address, latch data (for write), assert/deassert memory's WE (write enable) or OE (output enable)

Operation of “SLOW_SRAM”



Design Space (factors that drive design choices)

User Interface:

- How does the user read and write data using SCOMP assembly language instructions?
- How many SCOMP I/O addresses will you use?

Functional and Timing Considerations:

- How do you ensure that the memory is controlled correctly?
Changing address or control signals in the wrong order or at the wrong time can corrupt the memory.
- Can you reduce the number of instructions that SCOMP needs to execute to read and/or write data?

General:

- How will you organize your VHDL and maximize readability?
- Are there additional features you can add, such as sequential read/write optimizations?

Demo Details (SEE LAST SLIDE)

- ~~You will demonstrate your project to us to show that it works and explain the specifics of your new device~~
 - ~~This happens in your last lab section of the semester, labeled as “Project Demos” on the schedule~~
- ~~Exactly how you demonstrate your device is up to you (it’s part of the project), but we will give you some guidelines and tips later in these slides~~
 - ~~Along with how we will grade your project~~

What is reasonable?

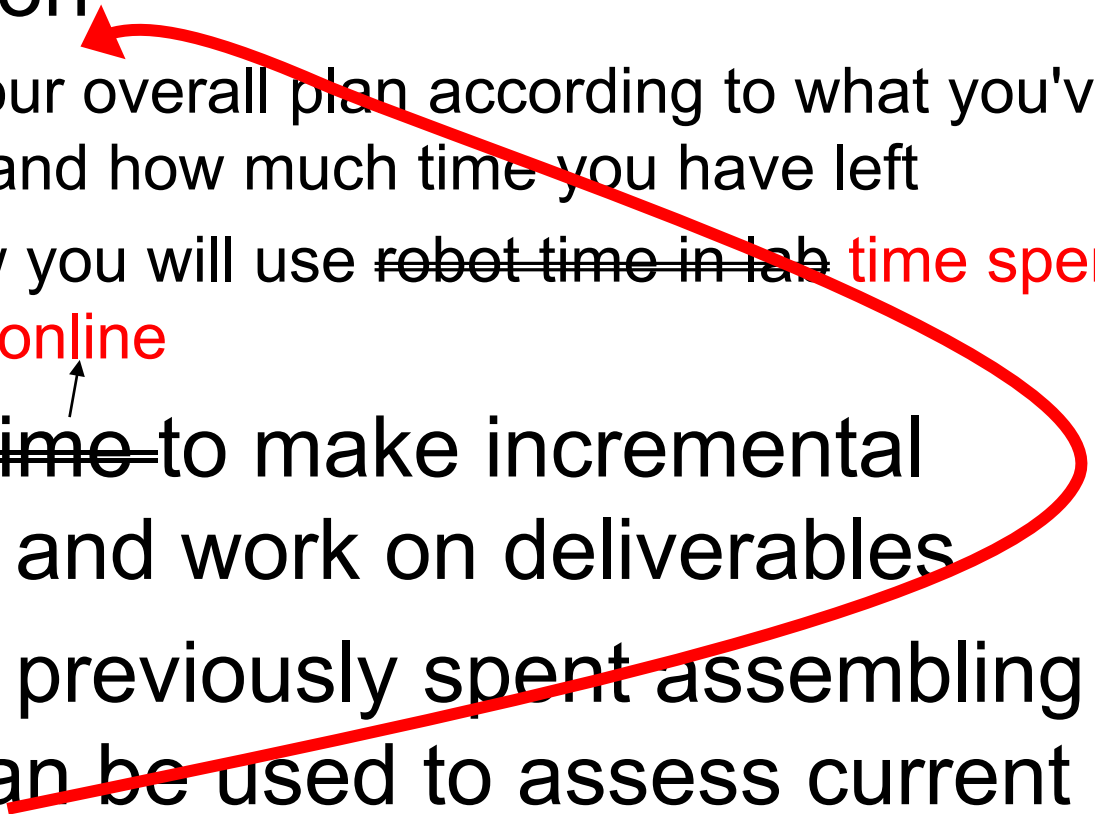


- This is deliberately open-ended
- There is no “perfect result,” and no “this device will earn this grade”
 - We don’t know what the results will be
- Do not overreach (and over-propose)
 - Proposing a “perfect solution” is doomed
- DO propose a progressive path with incremental performance improvements

Time management

- Focus on how you can build towards a certain result by completing smaller tasks.
- If you spend normal 2031 time and use that time wisely, you'll end up with an acceptable project
 - Typically, most 2031 projects are split A/B, with only a very few C grades
 - A conscientious effort is what we expect, and no more time in lab than you would normally spend (splitting effort among the team)

Project tasks vs. tasks in Labs 1-8

- Replace time spent on prelab work with preparation
 - Adjust your overall plan according to what you've finished and how much time you have left
 - Plan how you will use ~~robot time in lab~~ **time spent together online**
 - Use ~~lab time~~ to make incremental progress and work on deliverables
 - The time previously spent assembling lab results can be used to assess current progress
- 

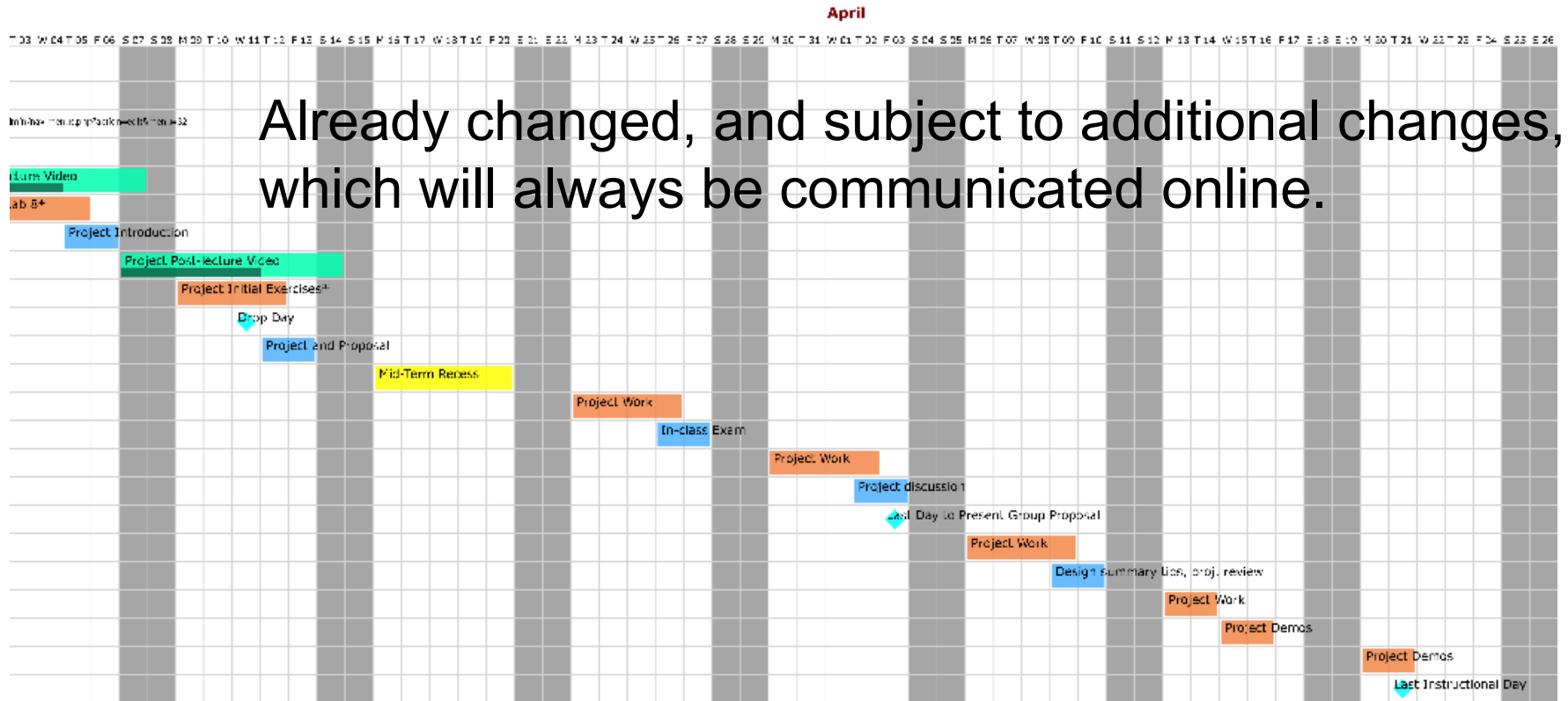
Effective use of lab time

- Do not all work on one piece of code
 - One or two team members can code
 - One or two can run ~~tests~~ **simulations and other activities verifying operation as best as possible**
 - Some can work on deliverables like the proposal
- ~~Open hours are still available~~
 - ~~Offered for convenience, not because we require you to use them~~
 - ~~Maintain a balance between this class, other classes, and personal time~~

Project Phases and Key Dates

- Prelab project preparation (before lab next week)
 - Some activities on Canvas.
 - Start your logbook, which you will maintain throughout the project.
- Introductory exercises (next week in lab)
 - Form project groups and discuss ideas
 - Complete some guided tasks
- Proposals presented by April 3rd
 - Incorporate brainstorming ideas into a polished presentation
- Complete your design by Tuesday, April 14th
 - You will not be able to work in the lab after this day
- Final demonstrations in lab April 15th to 21st
 - Demonstrate your solution in your section
- Final design summaries (an individual, not group, assignment) due April 21st (last day of classes)

Project Schedule



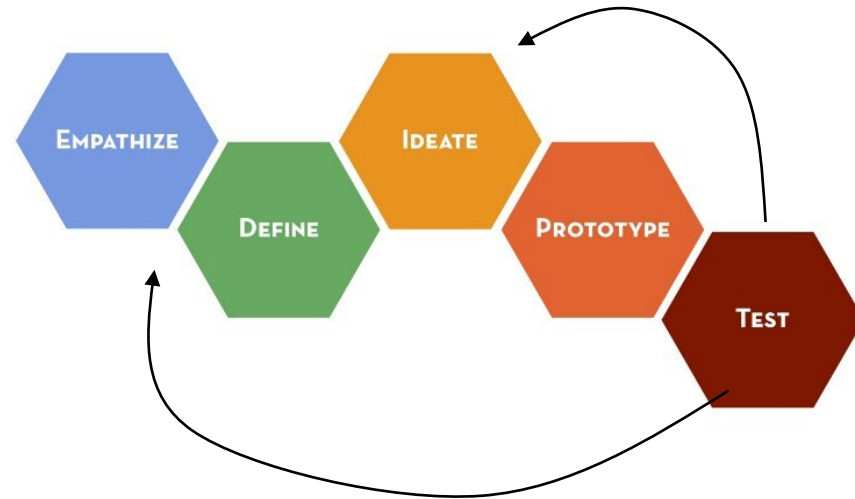
Clarifications



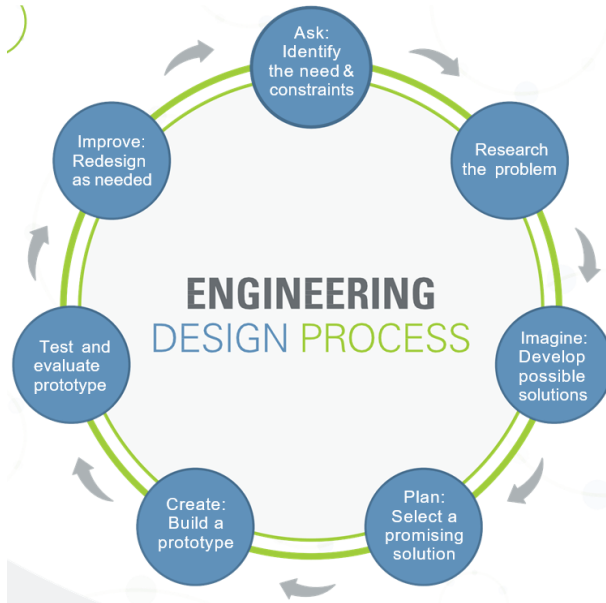
- Additional announcements and clarifications will be posted **on Canvas or Piazza**
 - You are responsible for information posted there
 - Could include changes to rules or assignments
 - Make sure you are monitoring it!
- Use Piazza to ask questions
 - If a general question is asked, everyone can benefit from the answer
 - If your question contains details specific to your design, you can limit the visibility to only instructors
 - Especially if you think your idea might be against the "spirit" of the project, ask us about it.

The Design Process revisited

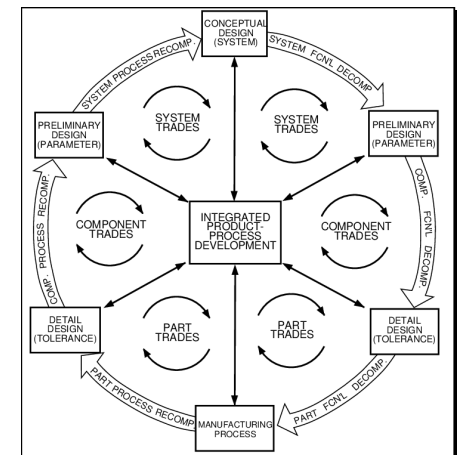
We discussed a simple abstract approach...



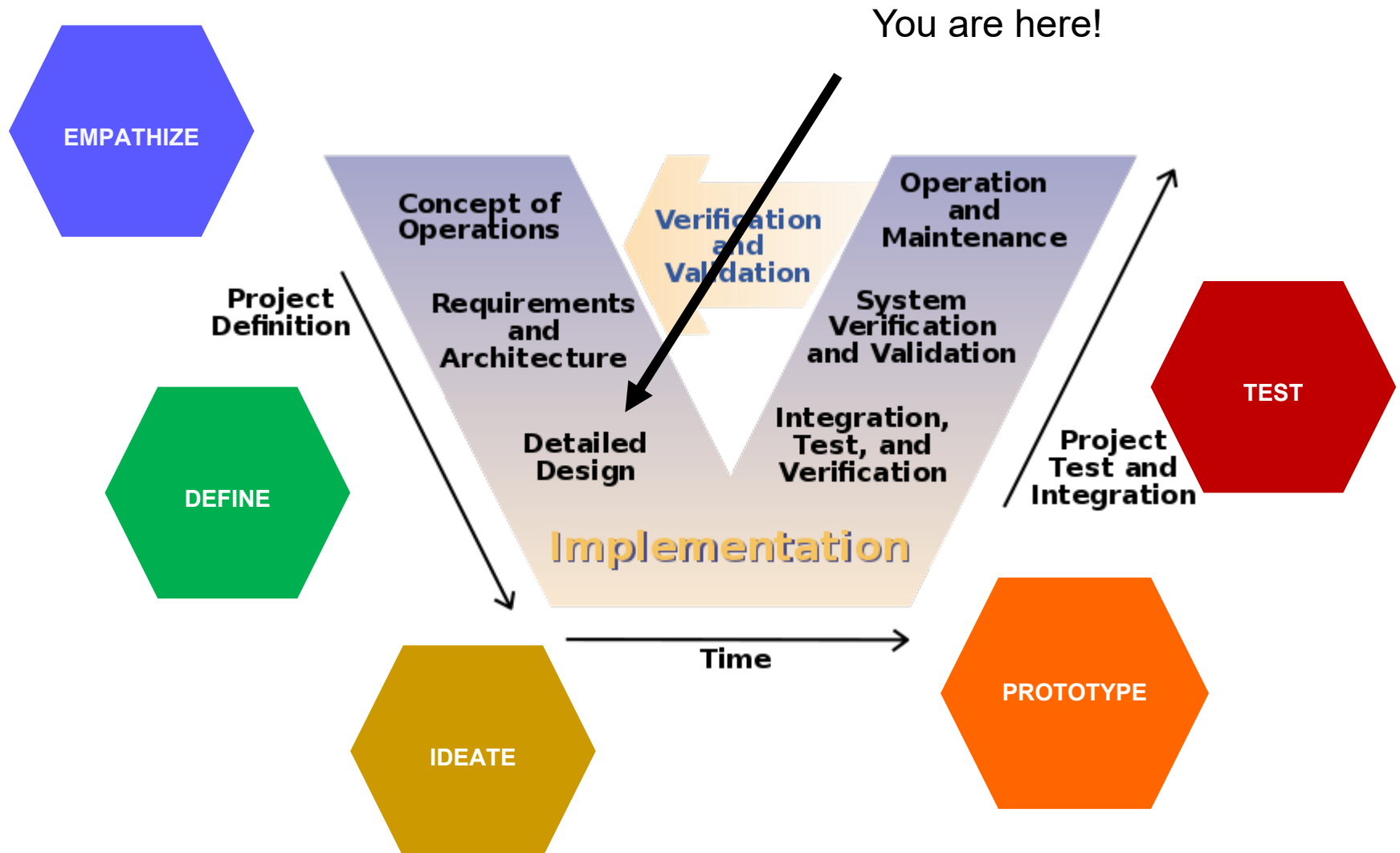
... but there are others more specific to engineering and software design



© 2018, Kaitlyn O'Toole,
University of Colorado
Boulder

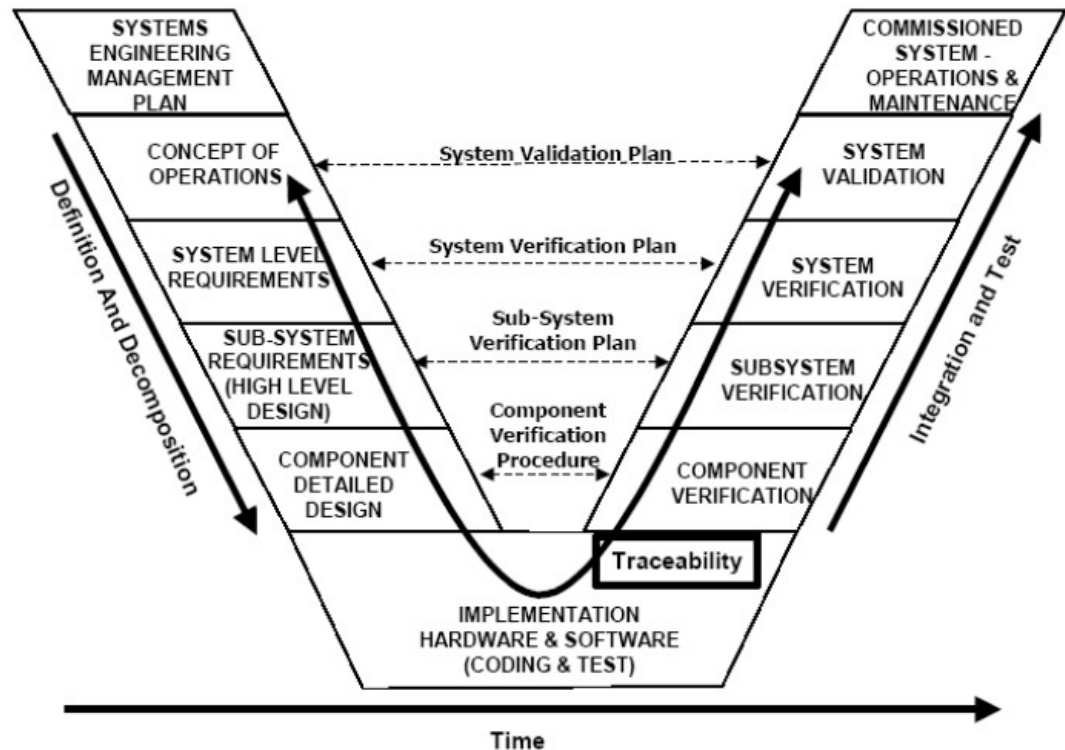


The V-model



Another take on the V-model

- Here, both sides of the V are subdivided
 - System
 - Sub-system
 - Component
- When ideating, start big, work toward details
- When implementing, get small components working, then put them together



How to approach detailed design?

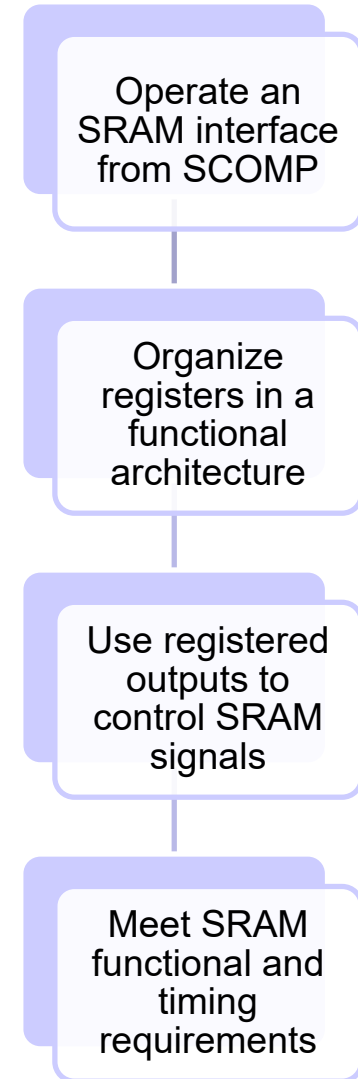
- Requirements often flow from higher levels to lower levels
 - Access more memory from SCOMP → Access the external SRAM on DE2 → Specifically, provide R/W access to all 512 kB of external SRAM → Create an I/O-based SRAM_interface → Use VHDL
- A systematic detailed design often flows from lower levels to higher levels
 - Meet SRAM timing requirements → Create registers and control signals to interface with SRAM → Define a functional architecture → Establish sequence of operations to access SRAM

Bottom-up Design

- More possible sources of problems means more difficult debugging
- It's in your best interest to build and test small pieces that you can then integrate
 - Each piece is easier to test and debug
 - Pieces can be used in multiple places, simplifying later stages of the design, and providing more options for “plan B”
 - Changes and improvements to individual pieces can benefit the whole design

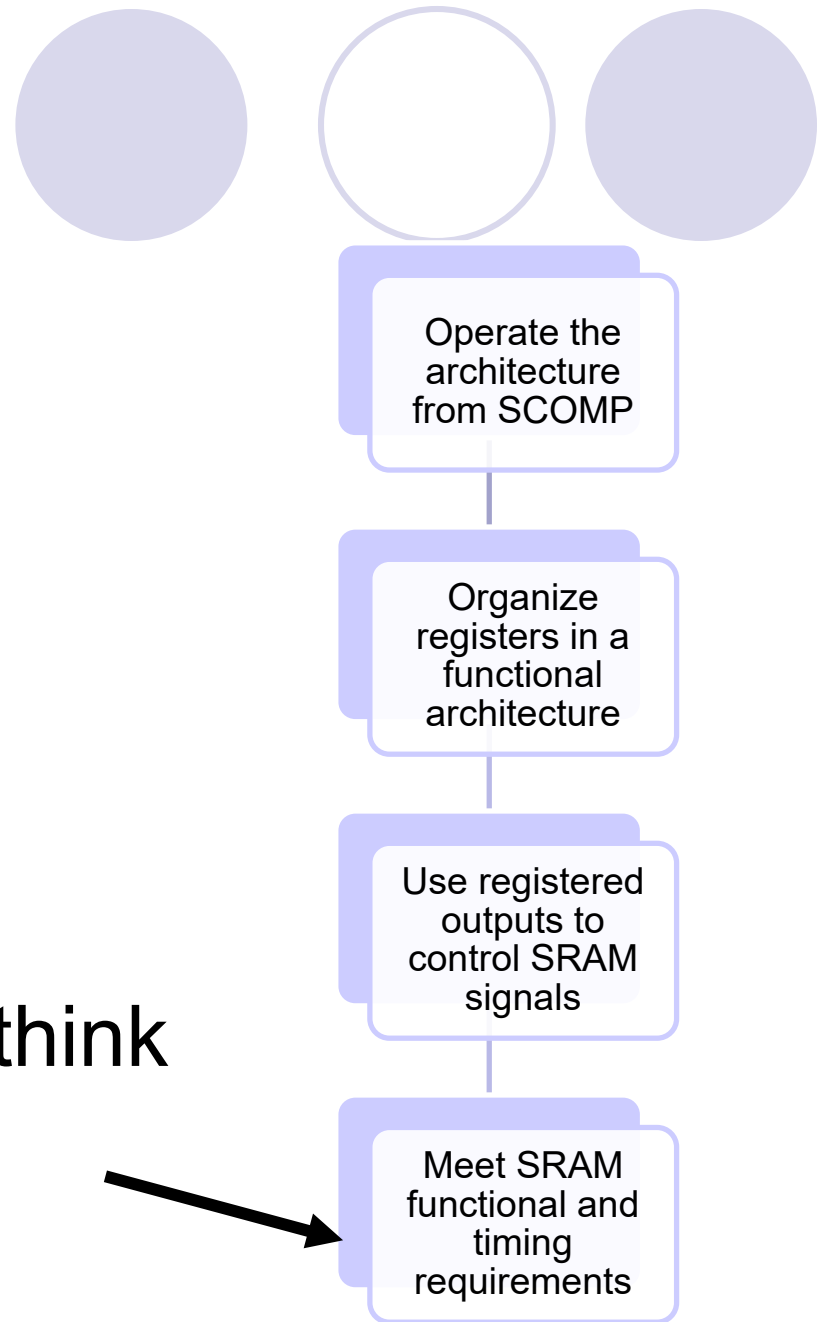
A possible project design hierarchy

- At the top – the project requirement
- Successively more primitive components underneath



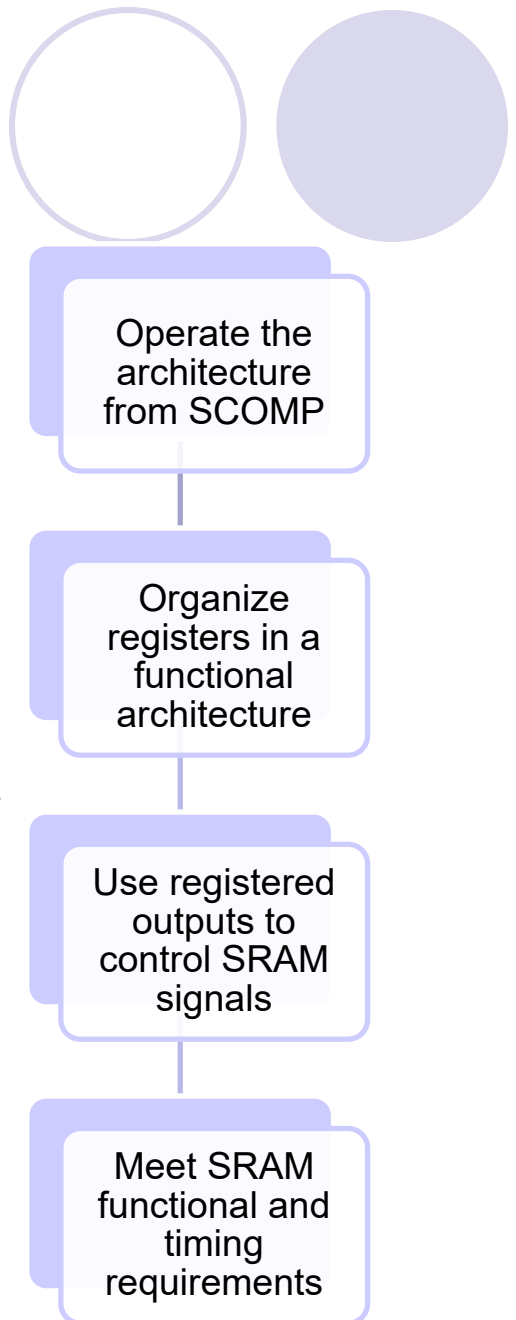
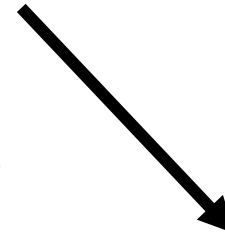
The bottom

- We tried to get you to think about this with initial exercises



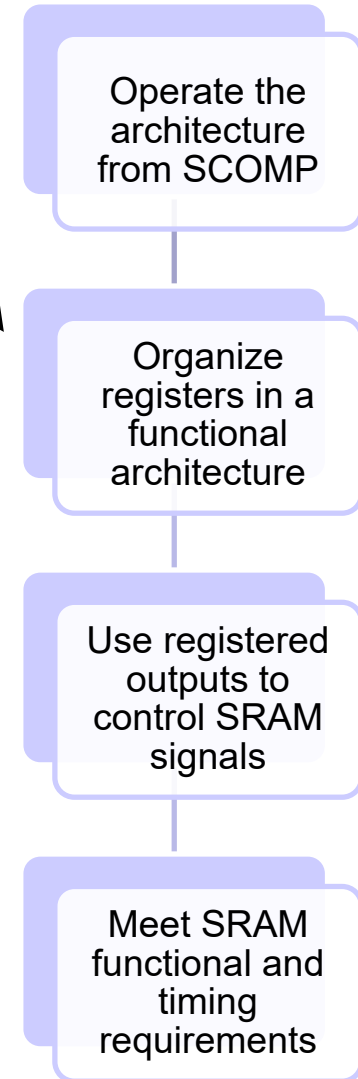
The next level up

- There are different ways to do this
 - The example gave a crude “bit-banging” approach to the /WE, /OE signals
 - It was all done in schematics, not meeting the VHDL requirement



Think about this level now

- Your design team probably already has some ideas
- Discuss the advantages of different strategies



Demonstration

- ~~If normal lab operations continue:~~
 - ~~Run a memory test (more specifics will be given later)~~
 - ~~Create an application demonstration using the DE2 interface~~
- If access to campus is restricted:
 - Make a group online presentation, highlighting
 - What the user does in assembly language for reads and writes (a flowchart, not code listings)
 - The architecture of the I/O device (block diagram)
 - Simulation output of reads and writes
 - Conclusions and recommendations for future work