ECE 385 – Digital Systems Laboratory

Lecture 18 – Final Project Kickoff

Zuofu Cheng

Fall 2018

Link to Course Website

ECE ILLINOIS

ILLINOIS

Final Project Proposal

- During week of Experiment 9.2 turn in final project proposal
  - Turn in via Compass just as with lab report
  - See template document on webpage under Final Project
- TAs will grade your proposal (10 points) and also give you feedback
  - Estimate difficulty points for assignment
  - Suggest additions or subtractions to features for demo
- Required lab sessions during final project are mid-checkpoint and final demo (other lab sessions are optional, TA will be present)
- Electronics service shop will check out additional hardware as needed (for example, camera boards, LCD screens)

Final Project Proposal

- Final project proposal point breakdown as follows:

Item	Points
Idea and Overview	2.0
Block Diagram	2.0
List of Features	2.0
Expected Difficulty and Justification	2.0
Proposed Timeline	2.0
Total	10.0

- Final project proposal should be about 1-2 pages long
- As long as you spend time researching and thinking about final project, proposal should get full credit
- In previous semesters, some students would not think about project early and write one-sentence proposals that were not thought out – leading to failed final projects!
- Having the proposal worth significant points forces you to **get started on thinking about your project early!**

Final Project Proposal – Overview

- Your overview should be in **abstract form** and consist of one or two paragraphs summarizing your project
- Describe what the final design should do and how it should go about doing it
- Should show some hours of technical research
- It should describe key components/features the game should have and what the general design approach is
- If you use a SoC, you should summarize which functions are software or hardware
- If you plan on using existing code, you should cite where you are getting the code from and make sure you can actually get it
- Describe how you are going to demo the design, especially if it is not obvious (e.g. if you are designing a CPU, how are you going to show that it works?)

Final Project Proposal – Overview Example

We propose to design and implement an Apple IIe computer on the FPGA as a System-on-chip. Our SoC will be based around a SystemVerilog 6502 CPU core which is found here (link). Additionally, we will implement using SystemVerilog essential components such as the System Bus, RAM, Video Display, Keyboard, and Emulated Disk Drives. Our design will also include a NIOS II CPU for the purposes of interfacing with the USB keyboard as in lab 8 to emulate the Apple keyboard. Our goal is to demonstrate our IIe SoC using the USB keyboard and VGA monitor, running a copy of The Oregon Trail.

Final Project Proposal – Block Diagram

- Your proposal should include a block diagram listing all the hardware modules you expect to need
- Individual signals do not need to be shown, but you should show the dataflow
- Dataflow diagram: which modules need to communicate with which other modules, and using what data?
  - If you have graphics, where are the graphic data (sprites) stored?
  - What module takes the sprite data, how does it take in commands?
  - What are the various busses that your CPU design will need?
  - How does the hardware communicate with the software?

Final Project Proposal – List of Features

- List of features, split into two categories:
  - Baseline set of features for the project to be considered working
  - Additional features that may be implemented for extra difficulty
- Elaborate on any features which are not self-explanatory – be as specific as possible
- Also elaborate on how you will demo a feature if it is not obvious
- For example, if you claim that you will have accurate ghost AI in Pac-Man, how will you demonstrate this to someone unfamiliar with the game?
- This is primarily how we will evaluate the functionality of your final demo

Final Project Proposal – Expected Difficulty

- Give an expected range of difficulty for your project (from 0 to 10)
- Give justification for this range
- This will use the same "scale" as the difficulty points awarded by your TA (also out of 10 points)
- Some rough examples:
  - Basic snake (0 – very easy)
  - Basic breakout – static blocks, no power-ups, no animation (0 – very easy)
  - Full featured breakout – different levels, animation, powerups (3 – medium easy)
  - Missile command – accurate gameplay, USB trackball/mouse (5 – medium)
  - Tetris – accurate gameplay, score keeping, etc (5 – medium)
  - Apple IIe computer running graphical games (7 – difficult)
  - Classic Macintosh computer running OS (10 – very difficult)
- Use this opportunity to highlight those aspects of your project which may increase the difficulty that may not be initially obvious

Final Project Proposal – Timeline

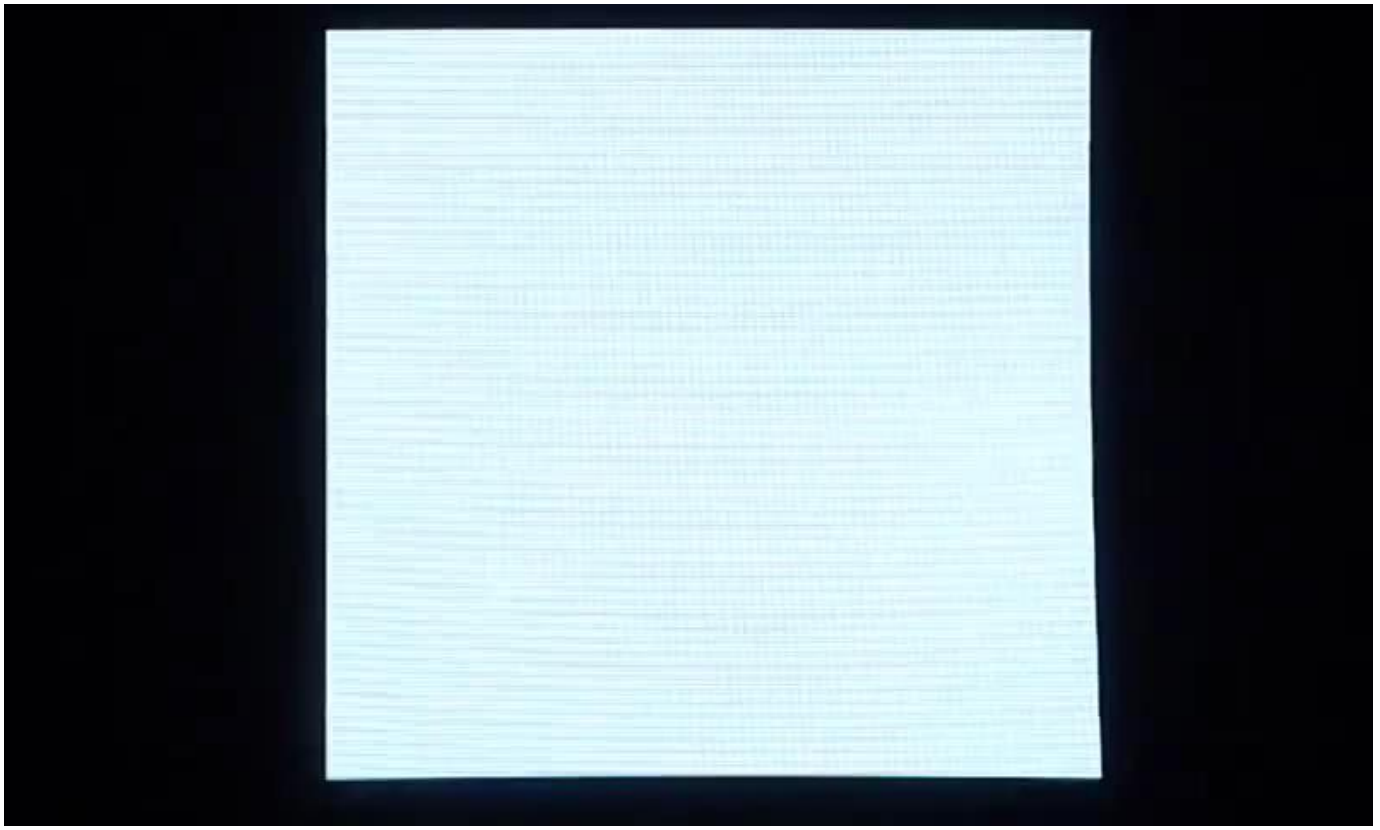
- Include an approximate timeline, should be at least week granularity
  - Project Week 0: 10/29
    - (Optional) Informally discuss with your TA about your final project and get feedback on expected difficulty. Note that this is also the week of the lab 9.1 demo.
  - Project Week 1: 11/5
    - Submit a detailed project proposal. This is worth 10 points - see the detailed description of the proposal (.PDF link). Your TA will give you feedback by the end of the week. Note that this is also the week of the lab 9.2 demo.
  - Project Week 2: 11/12
    - Work on your final project, lab is optional this week. Lab 9 report is due this week at the usual time.
  - Project Week 3: 11/26
    - (Required) Mid-checkpoint with your TA. You should show tangible progress to your TA, as described in your proposal.
  - Project Week 4: 12/3
    - Work on your final project, lab is optional this week.
  - Demo Week 5: 12/10
    - Demo project and submit written project report. Return lab kit. No Extensions!
- Also, be specific about what you plan to demo on the mid-project checkpoint

Hints for Final Project

- Propose a final project which can scale up and down
  - Be wary of projects which require critical hardware to work
  - Always have a backup plan, what can you demo if things don't go as planned
  - Games are a popular choice because features may be scaled up and down (e.g. can still get some demo points for Pac-Man even if there are no working ghosts)
  - Design pessimistically (assume things will be hard to get working), always unit test
- Use SoC to your advantage
  - Good hardware/software partitioning is essential to impressive project
  - Use software for low performance portions (e.g. game state, AI) and hardware for high performance portions (graphics, sound)

My (ECE 395) Final Project

- My 395 final project (using Xilinx XC2S50 – 1,728 LEs)
- Keep in mind, DE2-115 (EP4CE115 - **114,480 LEs**)
- Your final project should be 66 times better!/s



Final Project Recommendations

- Design **pessimistically** – always assume a module that you just wrote **won't work** right away
- **Test** each module, even if it seems trivial – even a simple flat testbench will suffice in most cases
- **Always** have something that works before adding a new module, always make sure you have some source control (e.g. Git/Dropbox), so that you can revert to a working copy
- **Better** to work together on same modules (pair/peer program) than split up tasks between partners
- **Do not** split up tasks such that project will completely fail if one partner fails to deliver

Final Project Recommendations

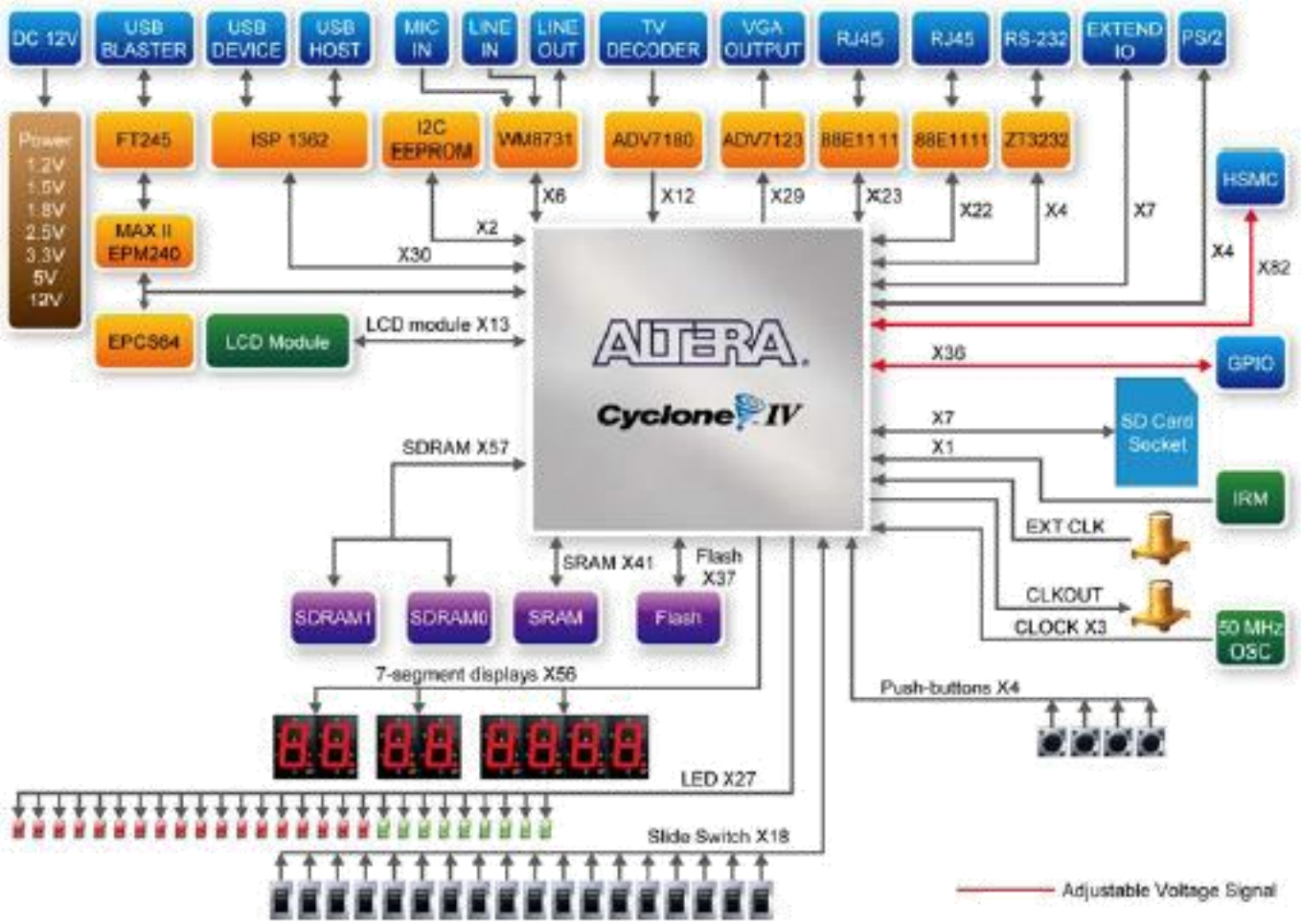
- **Do not** split up tasks such that project will completely fail if one partner fails to deliver (courtesy of the Facebook meme group)

"You do one half of the assignment I'll do the other half and then we'll join them together"

Available Memory on the DE2-115

- Many projects will be limited by memory and not FPGA logic
- Overview available memory on the DE2-115 and choose appropriate memory space for assets (code, graphics, sound, etc)
- Improper use of memory space (e.g. on chip memory instead of SDRAM) will make project not work or take forever to compile

Available Memories



Memory Technology Comparison

Technology	R/W	Size	Speed	Complexity	Initialization
On-Chip Memory (M9K)	R/W	9216 bits per block 432 blocks (3.888 Mbit)	Very Fast	Low	Directly in SV
SRAM	R/W	1M x 16 (16 Mbit)	Fast	Medium	Control Panel
SDRAM	R/W	32M x 16 x 2 (1 Gbit)	Variable	High	Control Panel
Flash	R (mostly)	8M x 8 (64 Mbit)	Fast (to read)	Medium	Control Panel

On-Chip Memory

- Tightly coupled to FPGA logic (on the same die)
- **Synchronous** SRAM technology, organized internally as 256x36
- Can be rearranged (e.g. 256x36, 512x18, 1024x9, etc…)
- Single port or dual port, single or dual clock
- Can be used as addressed memory or FIFO
- Maximum clock = 274 MHz (e.g. can run as fast as logic in almost all cases)

On-Chip Memory Continued

- OCM must be **synchronous**
- M9K blocks cannot be used for asynchronous memory
- Latches and/or combinational logic will be used for **asynchronous** memories (typically considered bad design, latches are poor use of FPGA area)
- Check "total memory bits" in Place & Route report to see if memory successfully inferred
- This is why InvSubBytes has a clock (even though it is just a ROM)

On-Chip Memory Continued

- Can simply instantiate using SystemVerilog
- FPGA Synthesis will infer (automatically use) M9K blocks if it can understand HDL
- Refer to *Recommended HDL coding Styles* to make sure inference works
- Can also instantiate in Qsys for use in SoC (e.g. as program memory)
- Alternatively, you can use the GUI tools (Megafunction) which gives you SystemVerilog module instantiation template

```
module ram_32x8(
 output logic [7:0] q,
 input [7:0] d,
 input [4:0] write_address, read_address,
 input we, clk
);

logic [7:0] mem [32];

Always_ff @ (posedge clk) begin
      if (we)
      mem[write_address] <= d;
      q <= mem[read_address];
end
endmodule
```

On-Chip Memory Continued

- Do **not** copy the lab 6 test_memory.sv, it is designed for simulation, not synthesis
- Although it will work for small memories (such as in lab 6), it will cause **very long** compilation times for larger memories…why?

```
...
always_ff @ (posedge Clk or posedge Reset)
begin
if(Reset)   // Insert initial memory contents here
begin
      mem_array[   0 ] <=    opCLR(R0)                  ;       // Clear the
      register so it can be used as a base
      mem_array[   1 ] <=    opLDR(R1, R0, inSW)        ;       // Load switches
      mem_array[   2 ] <=    opJMP(R1)                  ;       // Jump to the start
      of a program
                                                               // Basic I/O test 1
      mem_array[   3 ] <=    opLDR(R1, R0, inSW)        ;       // Load switches
      mem_array[   4 ] <=    opSTR(R1, R0, outHEX)      ;       // Output
      mem_array[   5 ] <=    opBR(nzp, -3)              ;       // Repeat
      ...
```

On-Chip Memory Continued

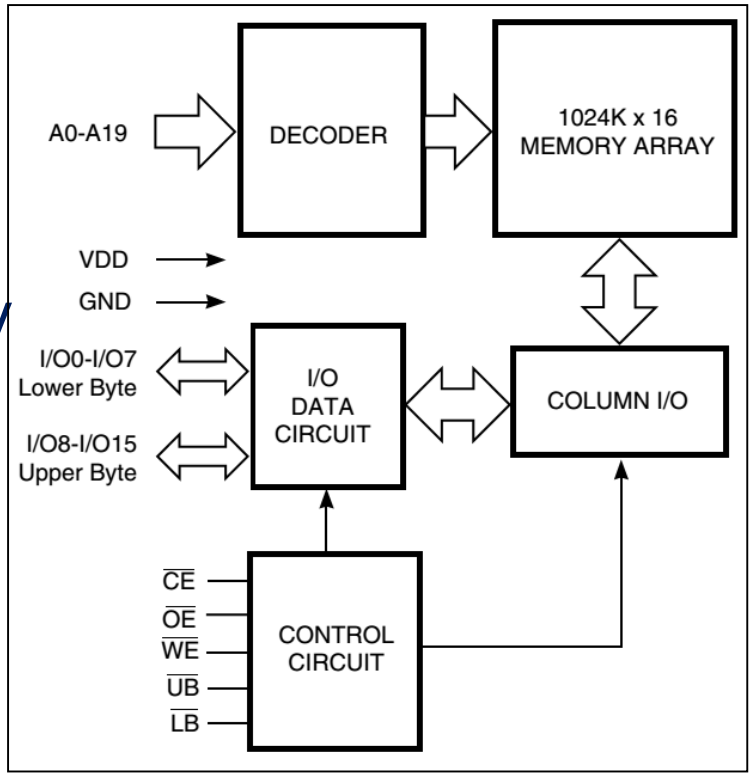
- If you want to initialize OCM, you should use the $readmem as specified in *Recommended HDL Coding Styles*
- This will create a MIF (memory initialization file) which initializes the OCM at **programming**
- Instead, use $readmem as specified in document, or initialize using GUI Megafunction
- $readmemb, $readmemh, etc… are special commands recognized by the synthesis tool for binary, hex data
- Place in initial procedure block
- Even though initial is typically unsynthesizable, it doesn't actually synthesize the $readmem command, the synthesis tool simply reads the file at compile time and initializes the contents

```
...
logic [7:0] ram[16];
initial
begin
     $readmemh("ram.txt", ram);
end...

ram.txt:
24
34
2e
2e
2e
2e
```

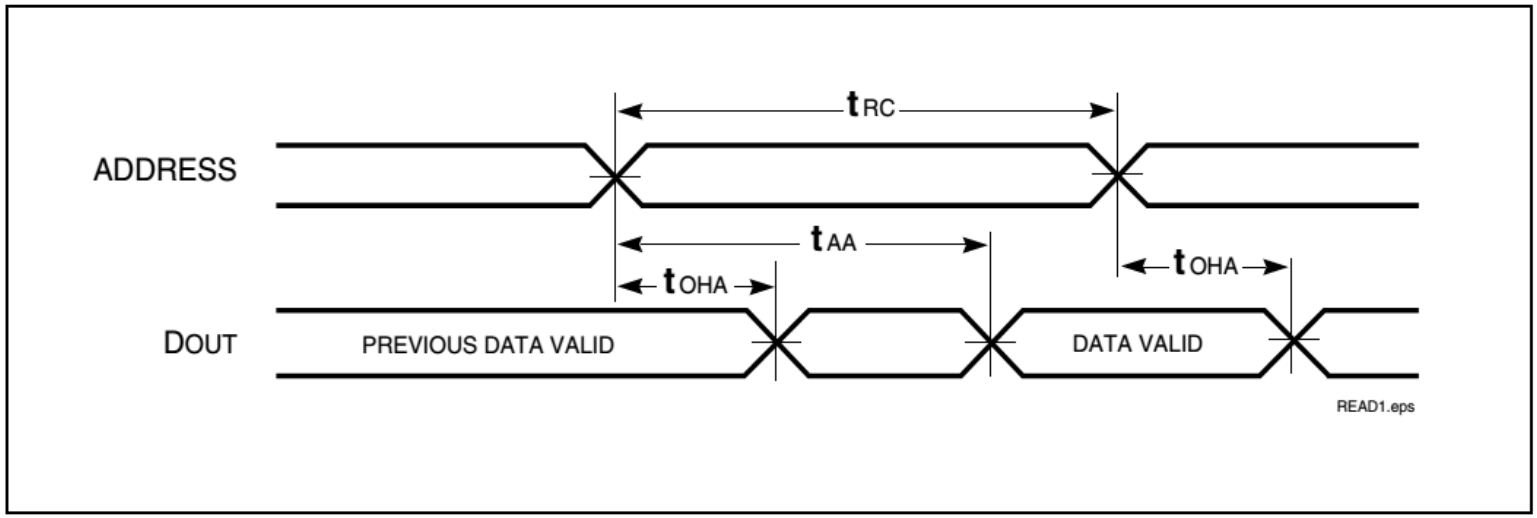
External SRAM

- 1M x 16 (2 Mbyte) organization
- Asynchronous (Access time = 10ns)
- 16 bit organization
- Byte access via UB/LB
- Datasheet here
- Used in Lab 6 for SLC3 memory



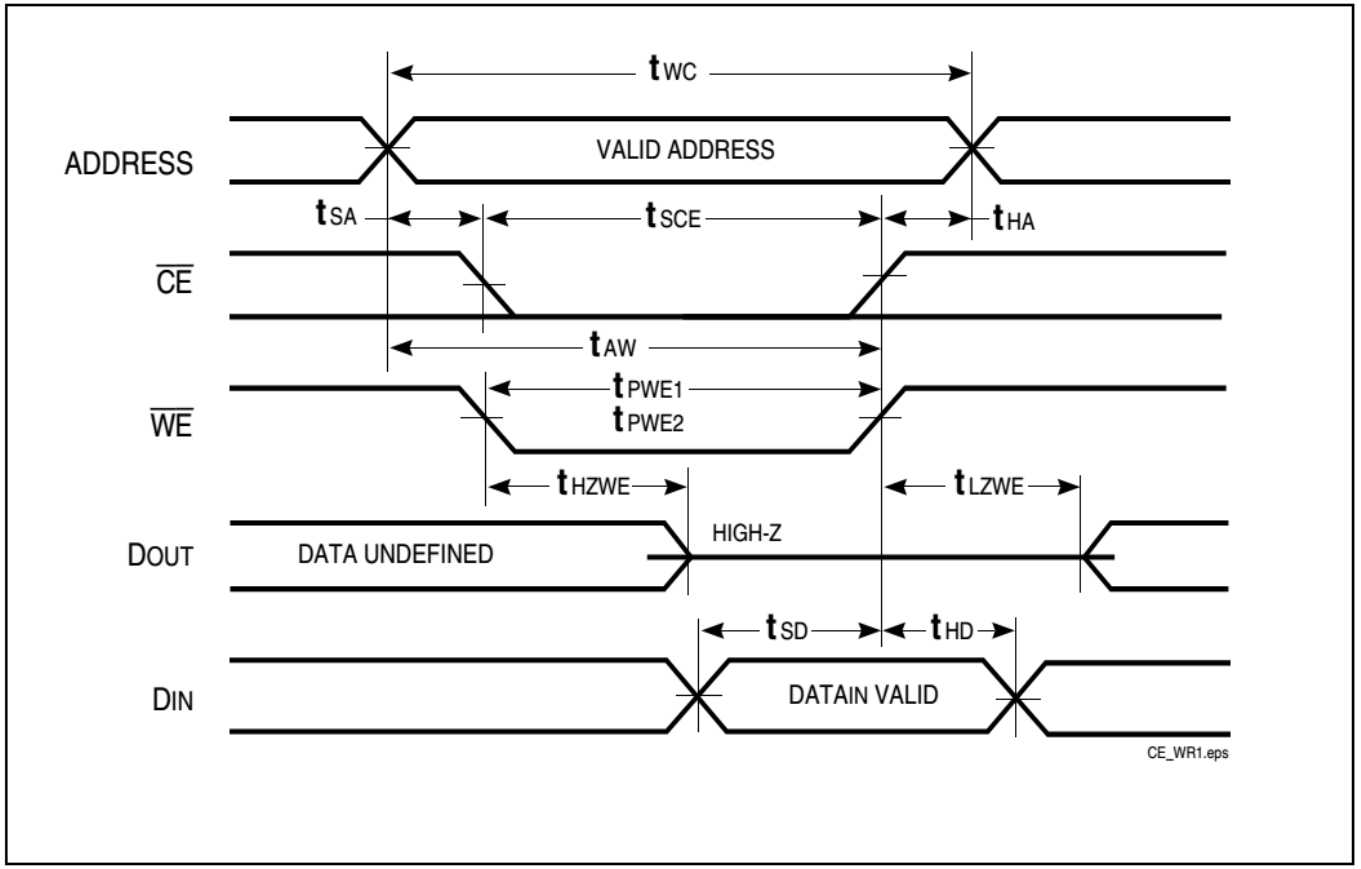
External SRAM Timing (Read)

- External SRAM is **asynchronous**
- !CE = !OE = 0 (in diagram below)
- Data is valid 10ns after address is valid
- If using state machine at 50 Mhz, data always valid by next cycle (from address)



External SRAM Timing (Write)

- Similar for write
- Cycle begins (is primed) on falling edge of WE, data is latched in on rising edge of WE



External SRAM Synchronization

- External SRAM is **asynchronous**
- Remember what we learned in Experiment 1 about asynchronous logic (it may have glitches)
- Cannot prevent glitches from appearing of FPGA combinational logic
- What is the solution?