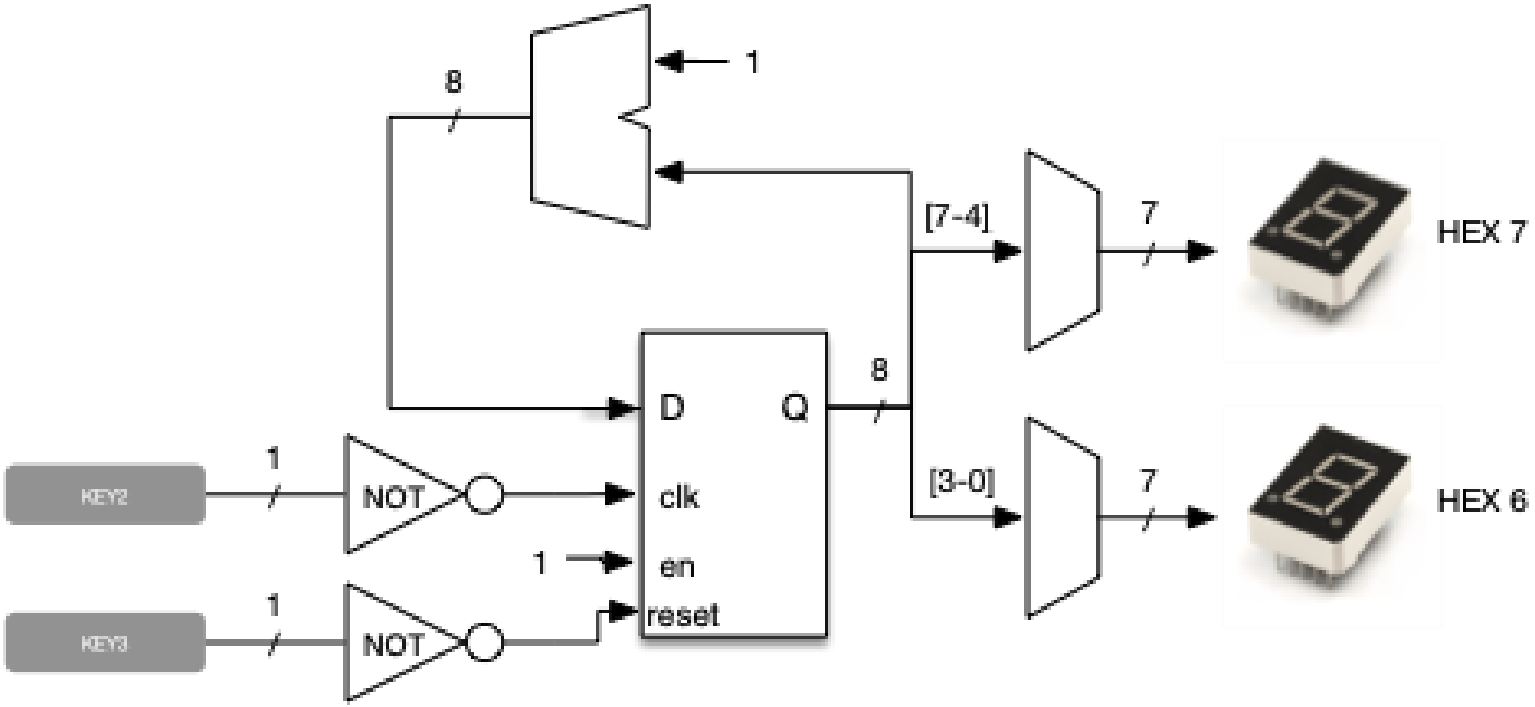
E93

VHDL Diagrams

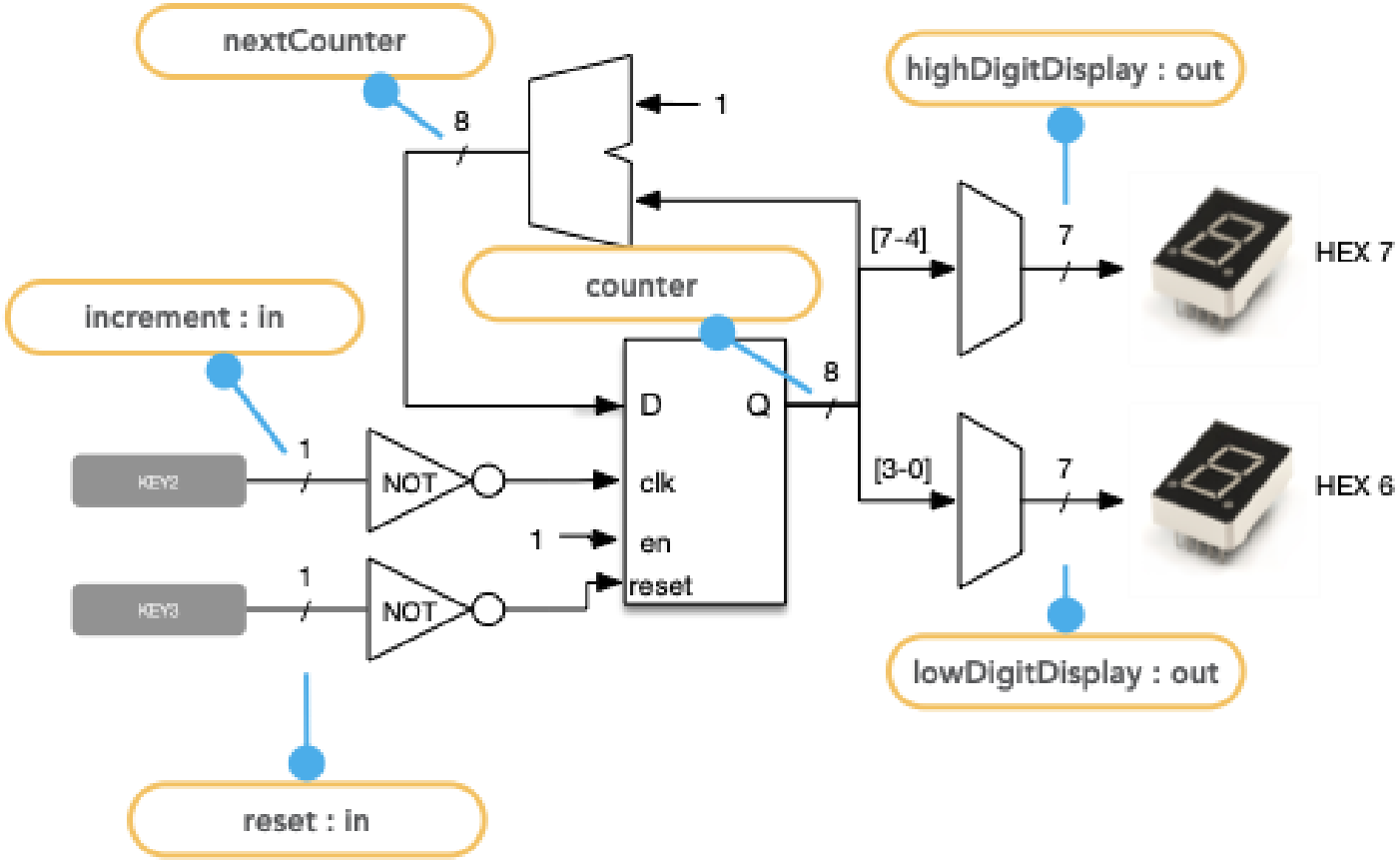
Getting Started

- Sketch the solution
- Identify top level entity
- Global inputs and outputs
- Local signals

PS3 Counter



PS3 Counter (with signal labels)

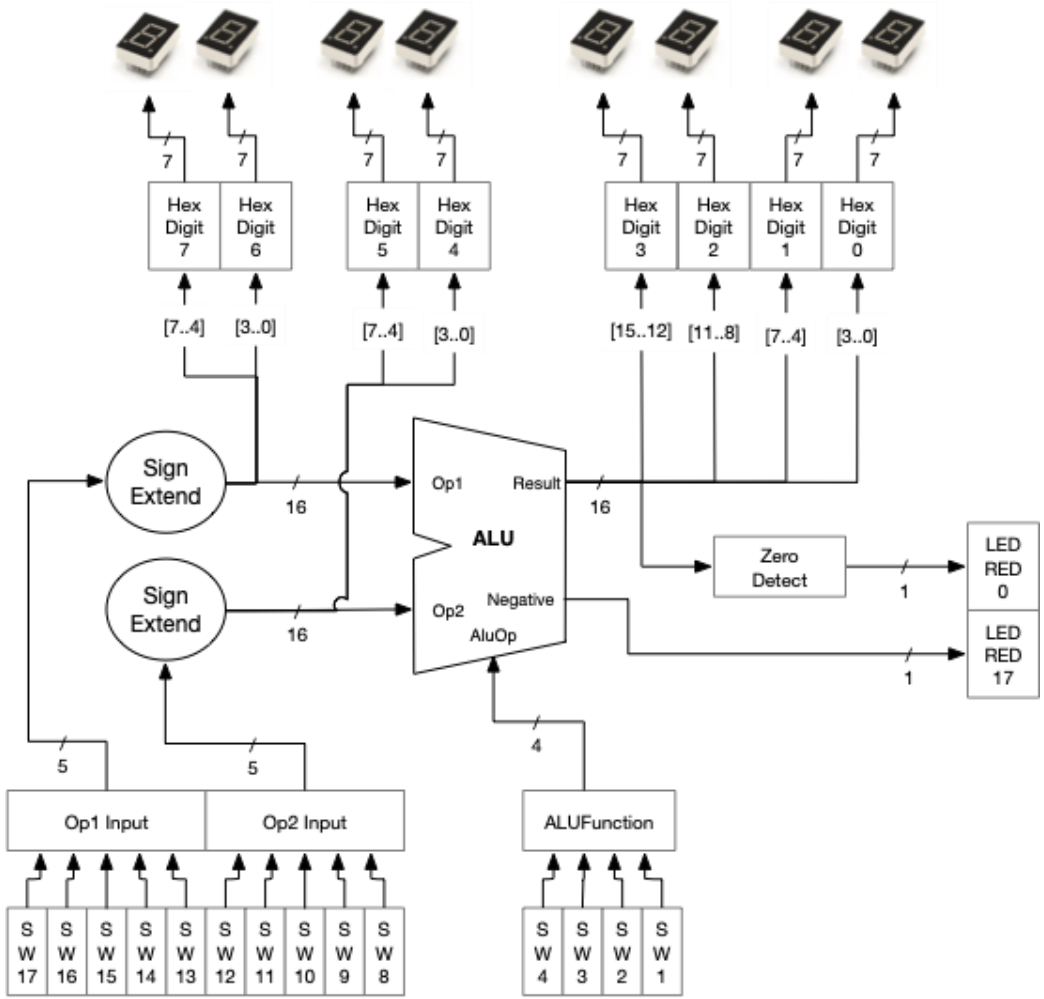


ALU Test Bench

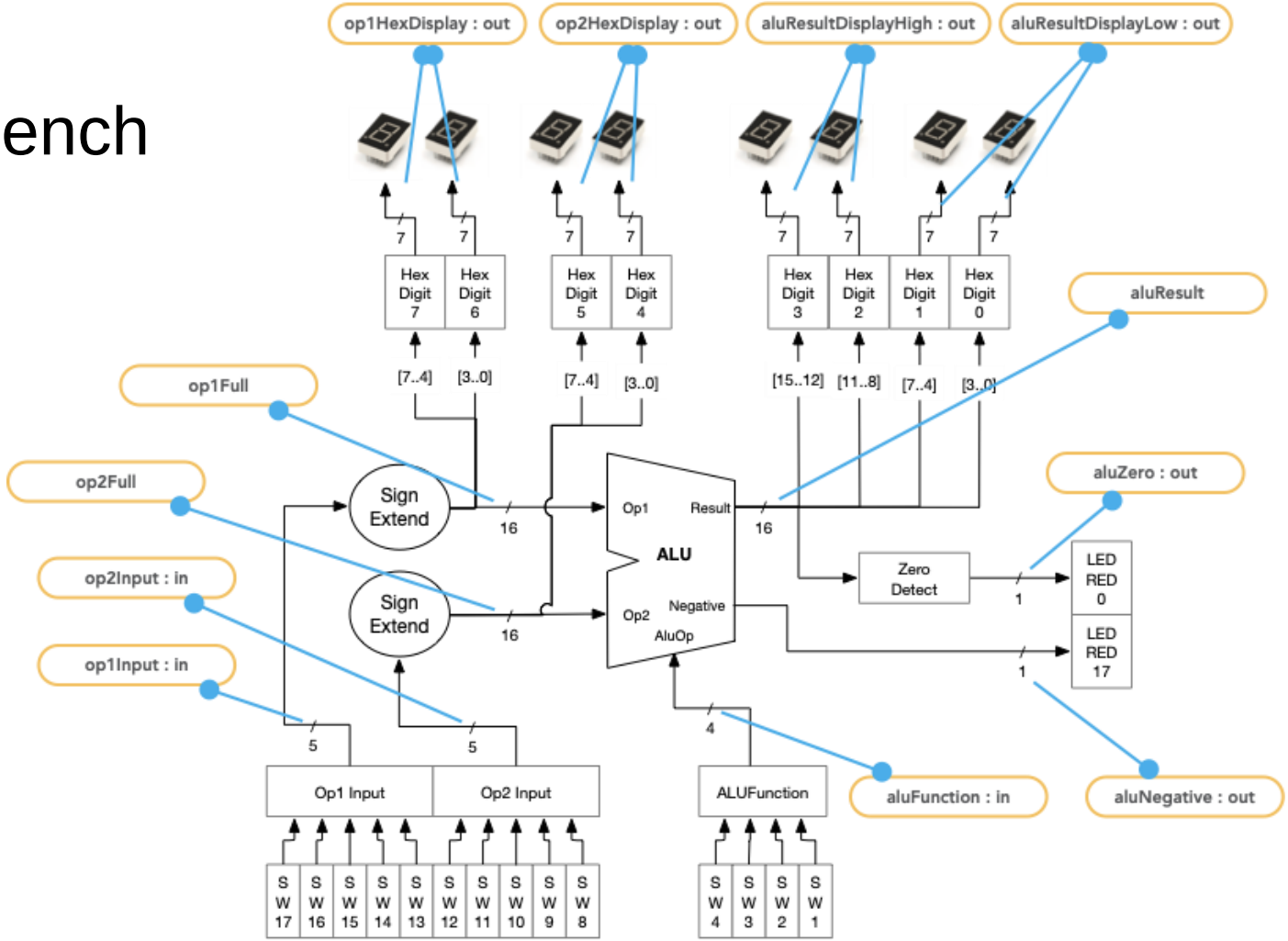
- Sketch the solution
- Identify top level entity
- Global inputs and outputs
- Local signals
- Get working with simple ALU first
  - Use + and - operators
  - Use shift_left() and shift_right() **(or wait until barrel shifter is reviewed)**
  - Use and, or, xor etc on std_ulogic_vectors
- Verify the test bench works with the simple ALU
- Replace the simple ALU with your custom ALU

ALU Test Bench

- Use switches for building values for the inputs
- We're using smaller inputs just for testing. Inputs are signed 4 bit numbers, displays support 8 bits, results supports full word.
- Good idea to support positive and negative inputs
- Use switches for the ALU function
- Use HEX displays for operands and result
- Use LED's for any PSW or other ALU outputs
    - zero detect is one example

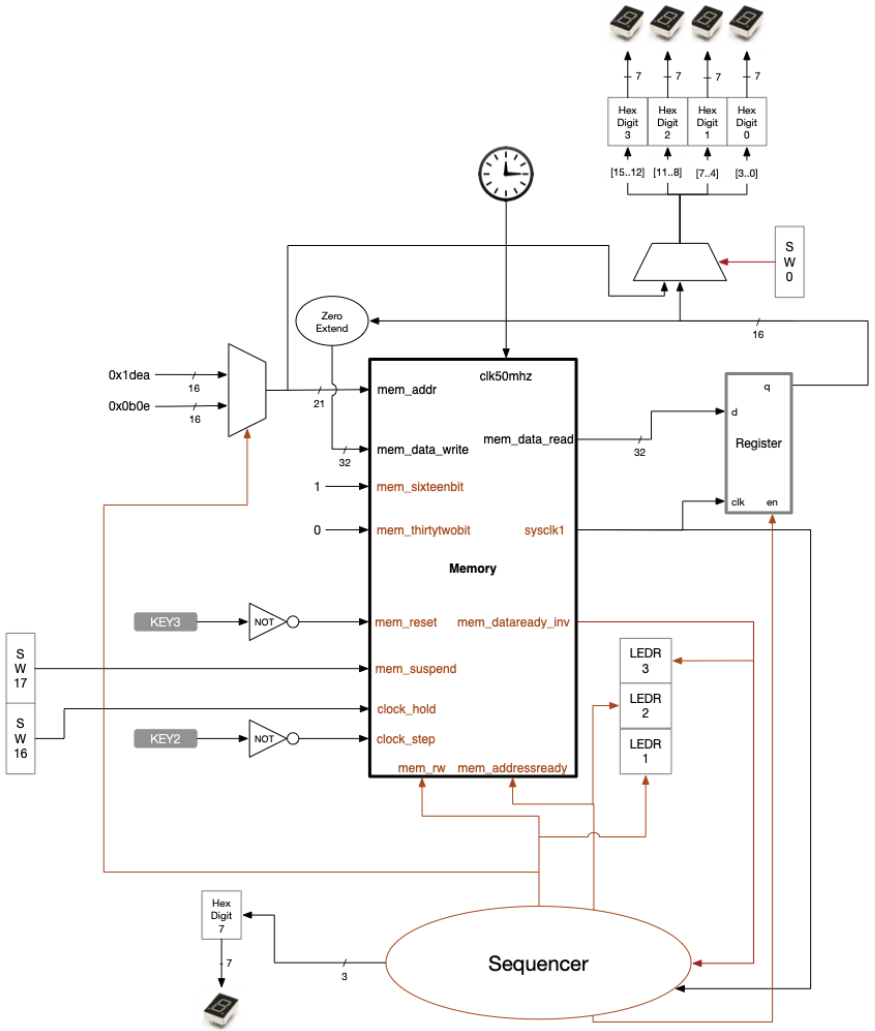


ALU Test Bench

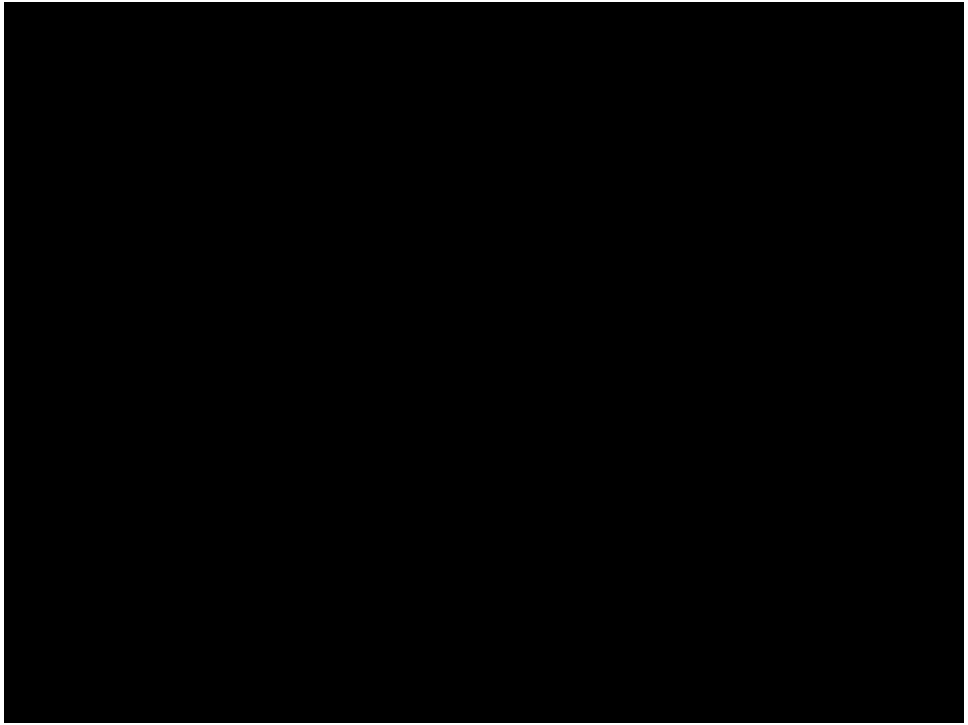


Memory test bench

- Only showing a subset of the memory inputs
- Continually copies from one mem address to another
- Stores the memory read in a register
- Not shown:
  - Connect slide switches to the clock divide (currently hard coded)



PS6 Demo



PS6 Tips 1

- The 10 steps in the memory handshake protocol are for each READ and WRITE.
- DE2-70 users must set an extra param in their project’s .qsf file or you will get compile time errors
- 0x1dea and 0x0b0e are memory addresses but the Quartus tools for viewing / changing memory are WORD indexes.
- Create a spreadsheet for the memory test bench combinational logic and FSM. This is a good exercise before implementing one for your full cpu. It makes writing the VHDL trivial!
- Create your top level entity and instantiate the components you need (memory controller, hex mappers, and register) and get it to compile BEFORE implementing any combinational logic or the FSM.

PS6 Tips 2

- The memory controller code is all std_logic_vector. You'll need to use the cast functions to connect them to your std_ulogic_vector signals. No cast is necessary for std_logic to std_ulogic, only needed for the vectors
- Casts can be used on either side of the port map

Modifying memory

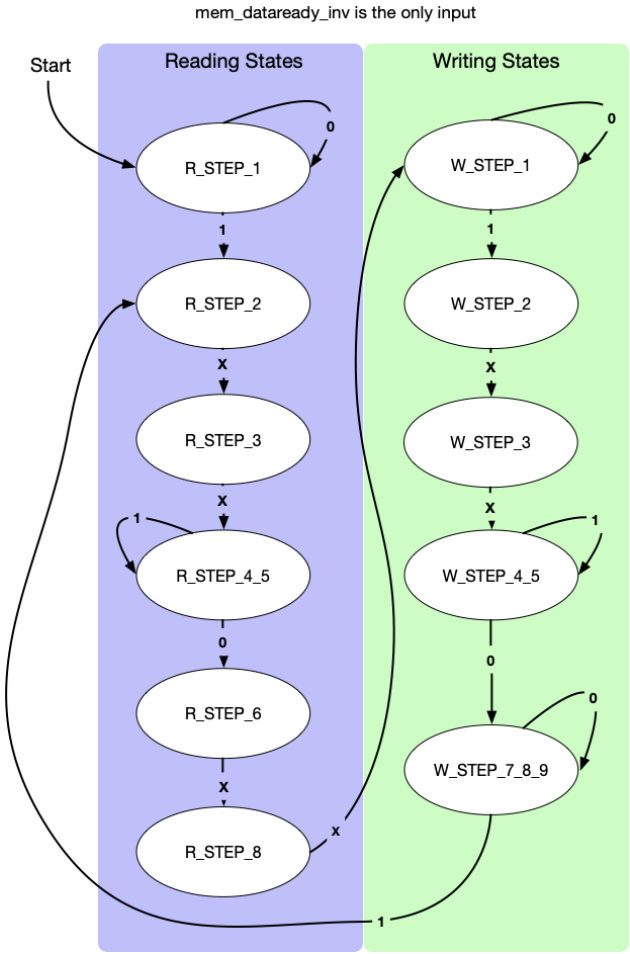
- See Application Note 4 on the course web site
- Creating a MIF file may save you some time

```
DEPTH = 16384;                  -- The size of memory in words
WIDTH = 16;                     -- The size of data in bits
ADDRESS_RADIX = HEX;            -- The radix for address values
DATA_RADIX = HEX;               -- The radix for data values
CONTENT                         -- start of (address : data pairs)
BEGIN

0ef5 : 1234; -- we are reading from address 0x1dea
0587 : 1111; -- we are writing to address 0x0b0e

END
```

The memory handshake protocol (as seen from the processor side) is:

1) Wait for mem_dataready_inv to go high.
2) Set the address lines (mem_addr), mem_rw line, mem_sixteenbit line, and mem_thirtytwobit line.
3) Set mem_addressready high.
4) Wait for mem_dataready_inv to go low.
5) mem_addr, mem_rw, mem_sixteenbit, and mem_thirtytwobit no longer need to be kept stable.
6) If the operation is a read (i.e., mem_rw is low), read the input data from the input data lines (mem_data_read).
7) If the operation is a write (i.e., mem_rw is high), then set the output data lines (mem_data_write) with the appropriate data to be written to memory. Of course, the mem_data_write lines can be set earlier (for example, in step 2 when the other lines are set).
8) Set mem_addressready low.
9) Wait for mem_dataready_inv to go high if you want to know that the write to memory has completed.
10) mem_data_write no longer needs to be kept stable.

The next cycle starts with: 1) Wait for mem_dataready_inv to go high.



Logic for PS6

	Combinational Logic			FSM Logic		
Current State	**mem_addr**	**mem_rw**	**mem_addressready**	**mem_dataready_inv**	**Next State**	**Comment**
R_STEP_1	X	X	0	0	R_STEP_1	
	X	X	0	1	R_STEP_2	
R_STEP_2	**0x1dea**	**0**	0	X	R_STEP_3	mem16 & mem32 must be stable
R_STEP_3	0x1dea	0	**1**	X	R_STEP_4_5	
R_STEP_4_5	0x1dea	0	1	1	R_STEP_4_5	
	0x1dea	0	1	0	R_STEP_6	
R_STEP_6	X	X	1	X	R_STEP_8	mem_data_read is now available, latch the register
R_STEP_8	X	X	**0**	X	W_STEP_1	
W_STEP_1	X	X	0	0	W_STEP_1	
	X	X	0	1	W_STEP_2	
W_STEP_2	**0x0b0e**	**1**	0	X	W_STEP_3	mem16 and mem32 must be stable
W_STEP_3	0x0b0e	1	**1**	X	W_STEP_4_5	
W_STEP_4_5	0x0b0e	1	1	1	W_STEP_4_5	
	0x0b0e	1	1	0	W_STEP_7_8_9	mem_data_write must be stable
W_STEP_7_8_9	X	X	**0**	0	W_STEP_7_8_9	
	X	X	0	1	R_STEP_2	