ECE 383 - Embedded Computer Systems II Lecture 12 - Datapath and Control

UNITED STATES AIR FORCE ACADEMY

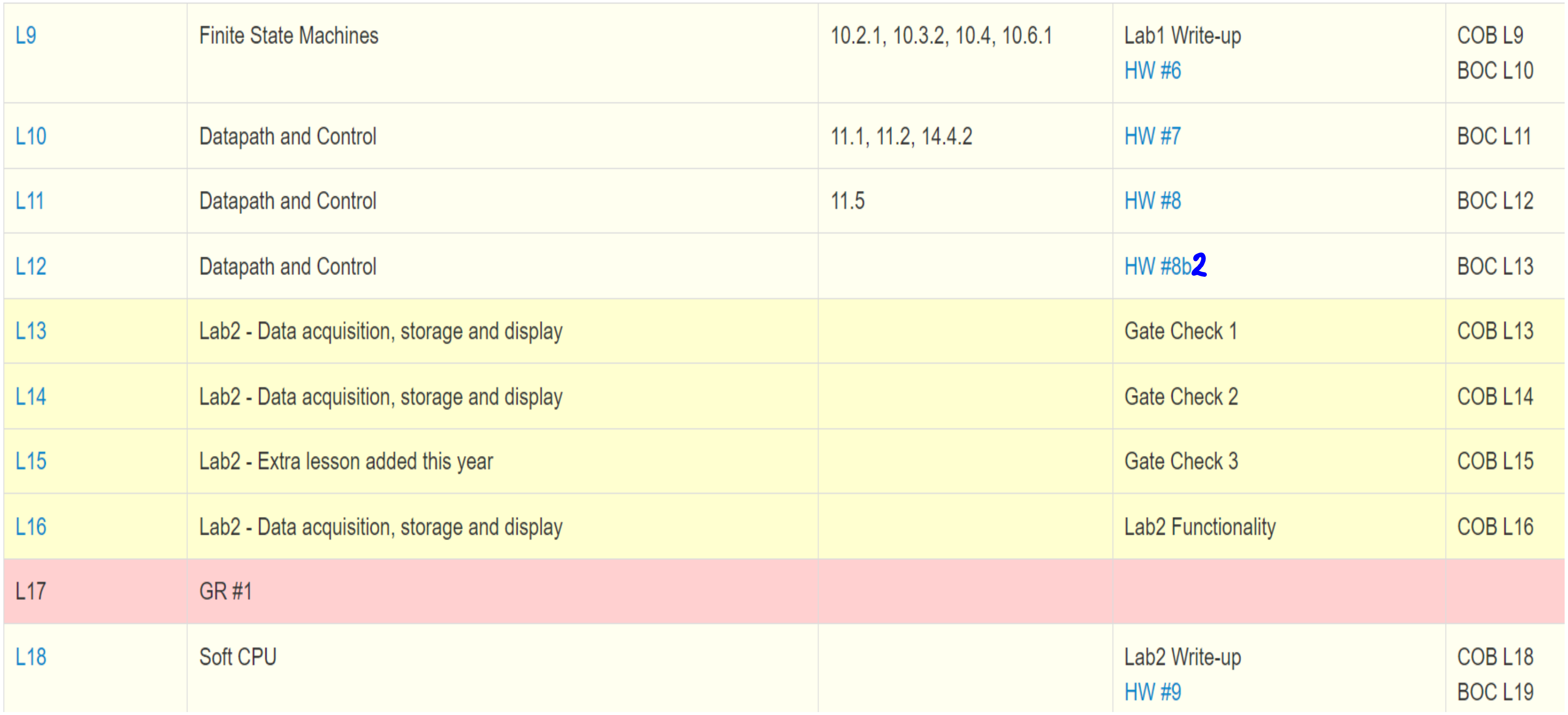
UNITED STATES
AIR FORCE
ACADEMY

Lesson Outline

- [illegible] CHECK GRADES IN BLACKBOARD
- HW# 8 Due
  - CpE's and device drivers or HW interface
  - Ready, Scan, CW, SW?
- GR Lesson ~~14~~ 17
  - Example code and Mini-C
- ~~2-Line Handshake~~ Skip this year
- Datapath and Control – BRAM
- Lab 2 Next Lesson!
  - Show schematic
- ~~Packages~~ [not required anymore]

HW 8b2 ?

Schedule

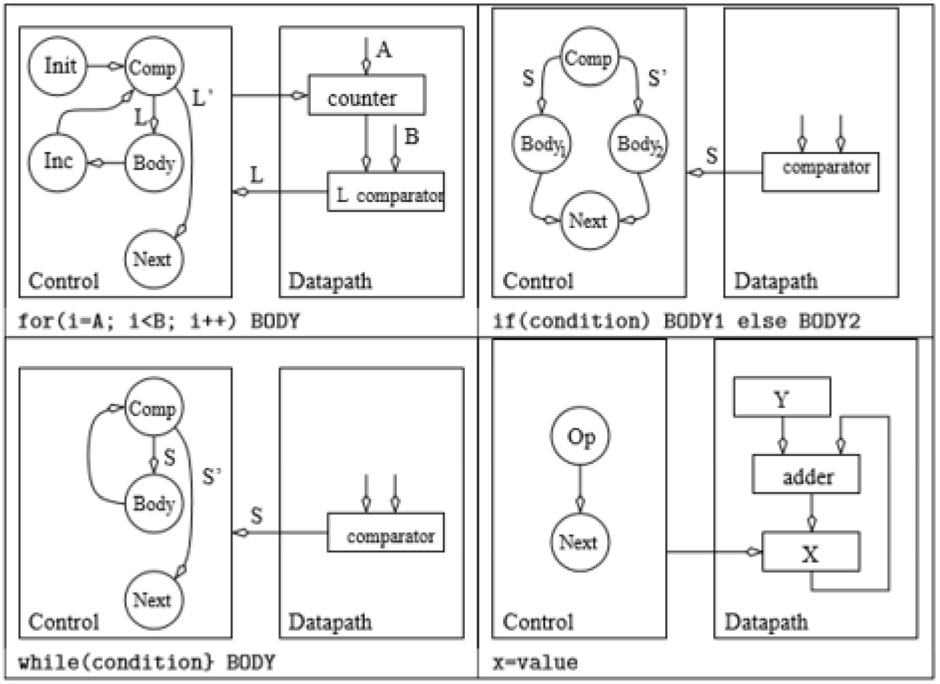


L9	Finite State Machines	10.2.1, 10.3.2, 10.4, 10.6.1	Lab1 Write-up HW #6	COB L9 BOC L10
L10	Datapath and Control	11.1, 11.2, 14.4.2	HW #7	BOC L11
L11	Datapath and Control	11.5	HW #8	BOC L12
L12	Datapath and Control		HW #8b2	BOC L13
L13	Lab2 - Data acquisition, storage and display		Gate Check 1	COB L13
L14	Lab2 - Data acquisition, storage and display		Gate Check 2	COB L14
L15	Lab2 - Extra lesson added this year		Gate Check 3	COB L15
L16	Lab2 - Data acquisition, storage and display		Lab2 Functionality	COB L16
L17	GR #1			
L18	Soft CPU		Lab2 Write-up HW #9	COB L18 BOC L19

UNITED STATES
AIR FORCE
ACADEMY

Given on GR

Mini-C to DataPath and Control

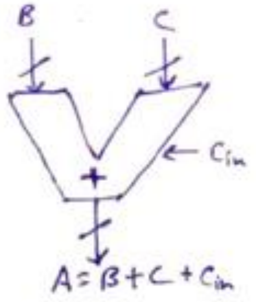


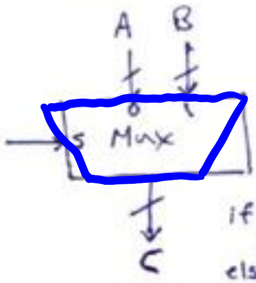
Given on GR

Example | BBBs

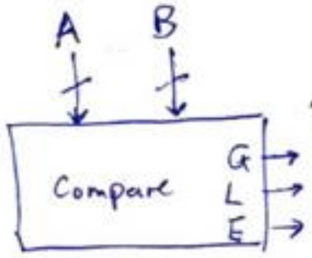
Combinational

Results are instantaneous (almost)





if s=0
c=A
else if s=1
c=B

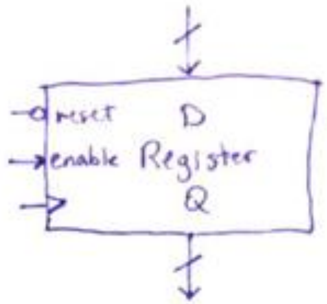


'1' if
A>B
A<B
A=B

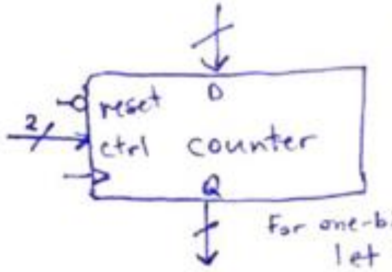
Sequential (Register-based)

For simplicity, can assume all resets hooked to Global Reset and all clocks hooked to Global Clock

Results appear on rising clock edge

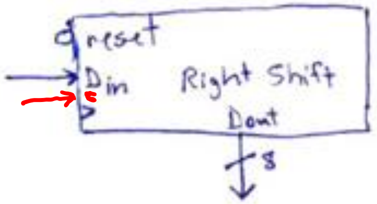


enable		
0	→ Q=Q	hold
1	→ Q=D	load



ctrl		
00	Q=Q	hold
01	Q=Q+1	count
10	Q=0	clear
11	Q=D	load

For one-bit control counter, let ctrl(1) = '0'



c=0 hold
c=1 shift

Dout = Din & Din(7 to 1)

HW8: Ready, Scan, CW, SW?

- Datapath and Control Design Methodology
  - <u>Datapath</u> - responsible for data manipulations
  - <u>Control</u> - responsible for sequencing the actions of the datapath

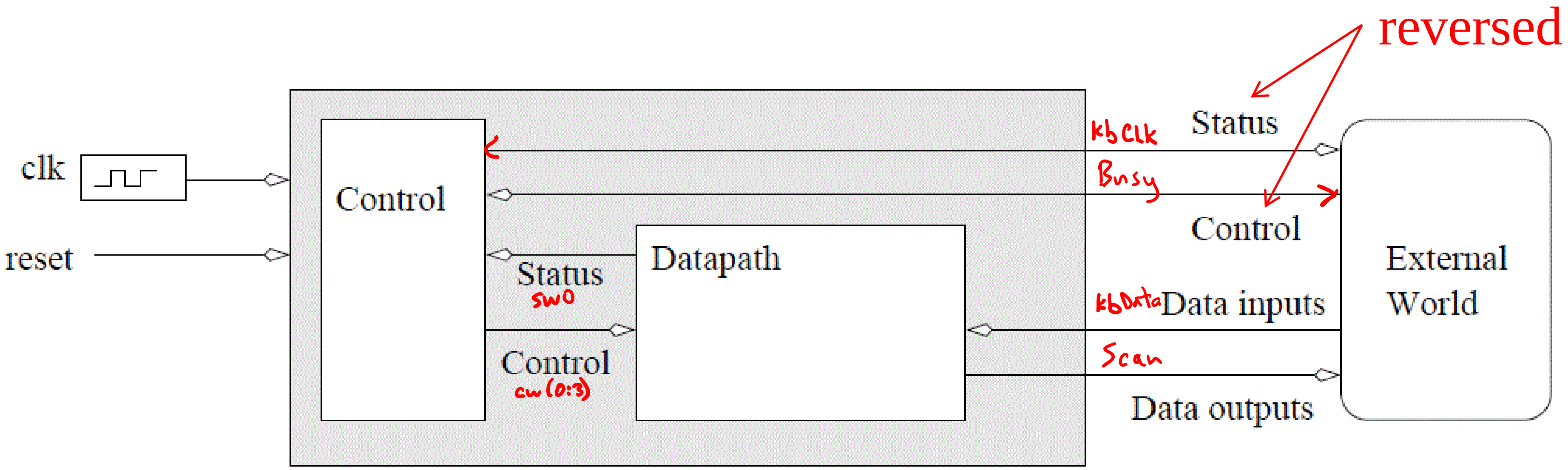
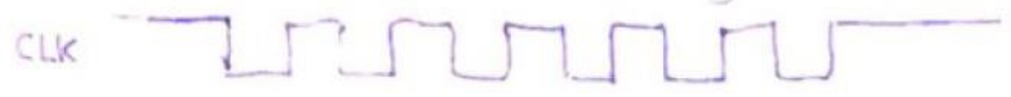


Fig 10.0 - An abstract digital system constructed from a datapath and a control unit.

GR Hints

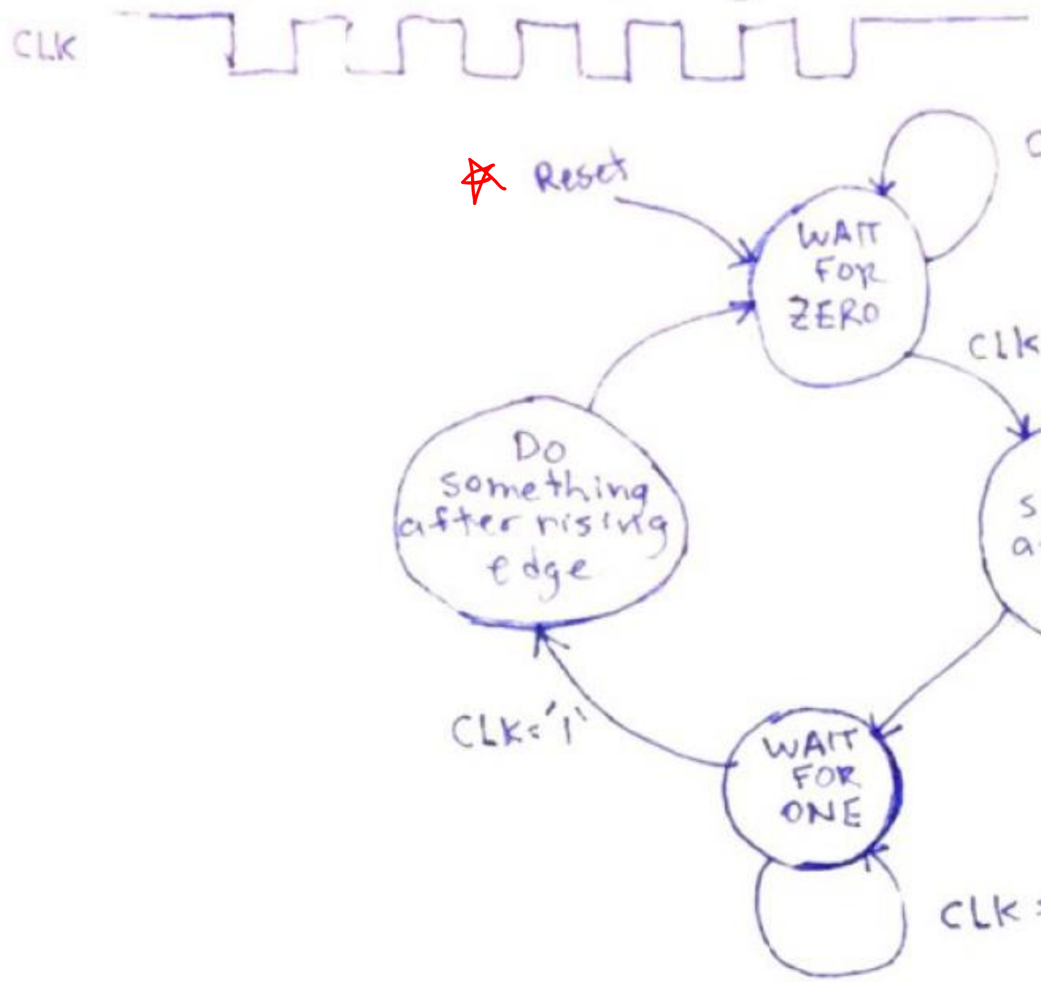
- Bring your calculator. No sympathy if you don’t.
- Clock and Reset on diagram?
- Two states or one state?
- Decoding clock, low and high
  - Can you make a 4-state FSM to track the rise and fall of a clock?

for syncing with and decoding a recieved clock signal



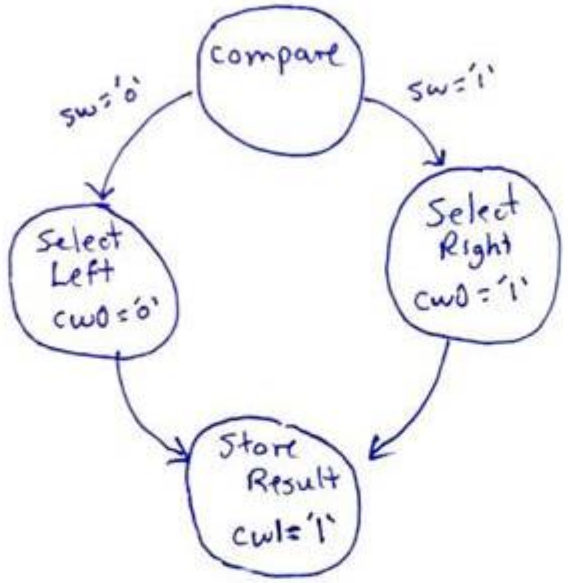
FSM

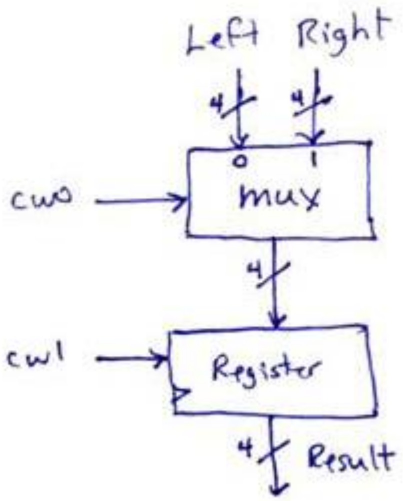
for syncing with and decoding a recieved clock signal

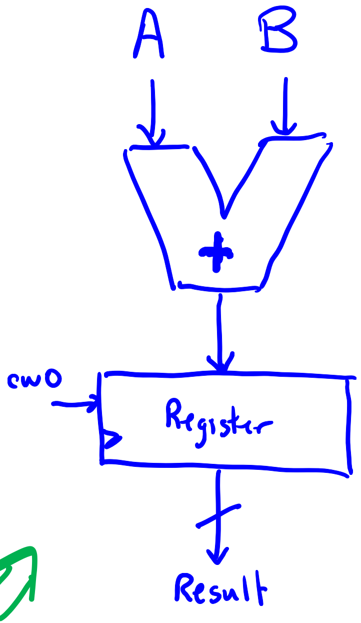


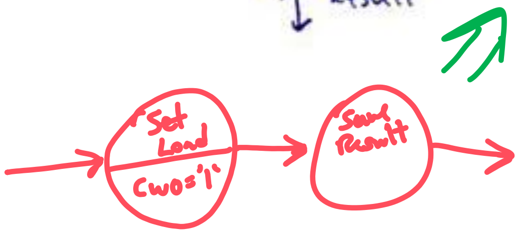
GR: two states or one state?

If you are designing a FSM controlling a datapath, and you need to do some action (from the controller) and then store the result, this takes two states (not one state).
For example, see the diagram below. The action is selecting "left" or "right", and then storing the value in the variable called "result". Takes two states! (I've seen students try to do this in one state).







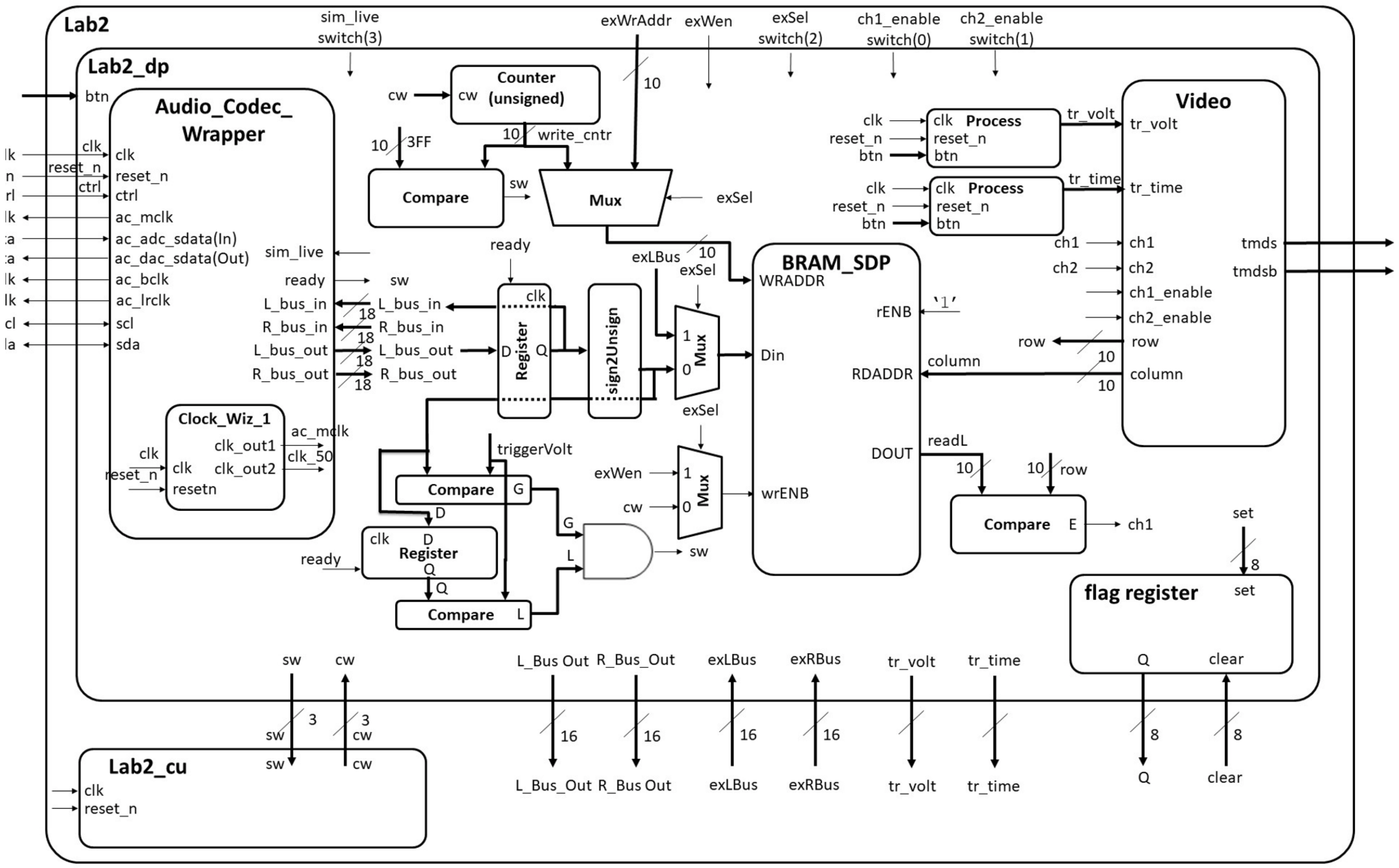


Sometimes an action and storing can take just one state, when the action does not require a signal from the controller (because combinational logic is just real-time computing the action). For example, suppose the action is an ALU computation (real-time combinational logic), and the result of the ALU computation just needs to be stored in a register. This required just one state → load register.

(Whoops! This assumes cw0 already is '1' [Load], from previous state!)

Datapath and Control - BRAM

LAB 2



9 February 2021

Datapath and Control - BRAM

- Artix 7 FPGA
  - First page of 7 Series Family Overview: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
  - Third page lists quantities how many of these resources our Nexys Video boards have.
    - For reference we are using the XC7A200T chip and the SBG484 package.

UNITED STATES AIR FORCE ACADEMY

Datapath and Control - BRAM

- In our upcoming Lab2, you will need a large RAM to store 18-bit audio samples streaming in from the ATLYS board.
- The Xilinx FPGA on our board, a Artix 7, contains built in block RAMs (BRAMs).
- You can select of the three main BRAMSconfiguration (BRAM_SDP_MACRO, BRAM_SINGLE_MACRO, BRAM_TDP_MACRO) available in the UNIMACRO library.
- We will be using a BRAM_SDP_MACRO in our design.
- According to Vivado Design Suite 7 Series FPGA Libraries Guide:

UNITED STATES
AIR FORCE
ACADEMY

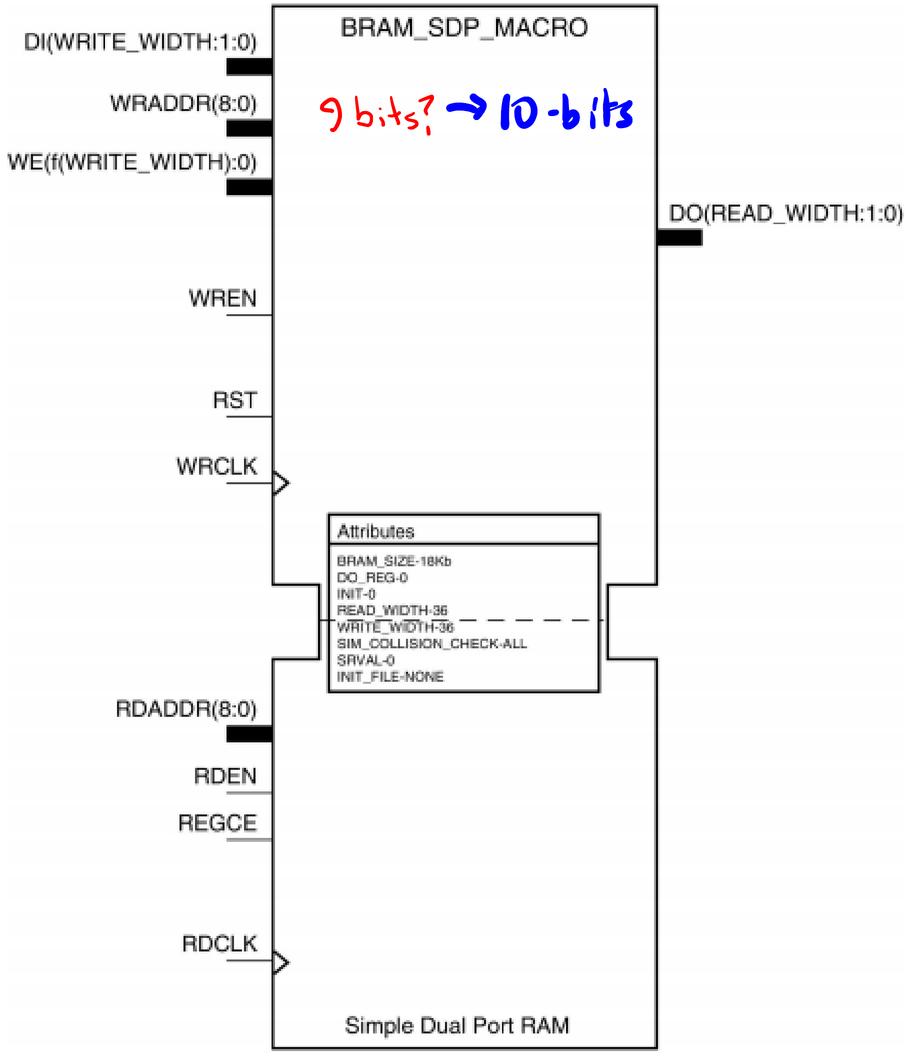
Datapath and Control - BRAM

- FPGA devices contain several block RAM memories that can be configured as general-purpose 18Kb or 36Kb RAM/ROM memories.
- These block RAM memories offer fast and flexible storage of large amounts of on-chip data.
- Both read and write operations are fully synchronous to the supplied clock(s) of the component.
- However, READ and WRITE ports can operate fully independently and asynchronously to each other, accessing the same memory array.
- Byte-enable write operations are possible, and an optional output register can be used to reduce the clock-to-out times of the RAM.

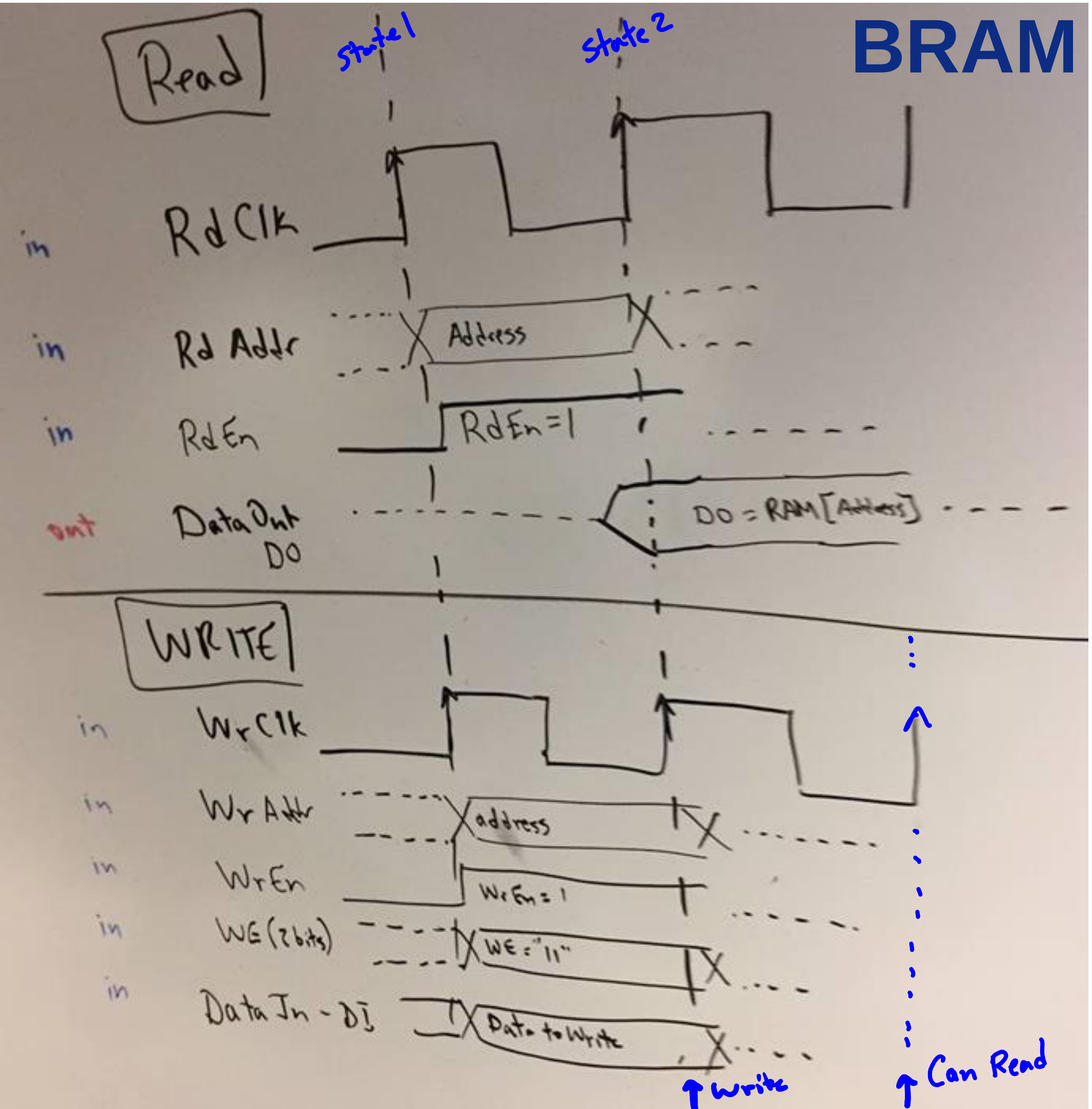
Datapath and Control - BRAM

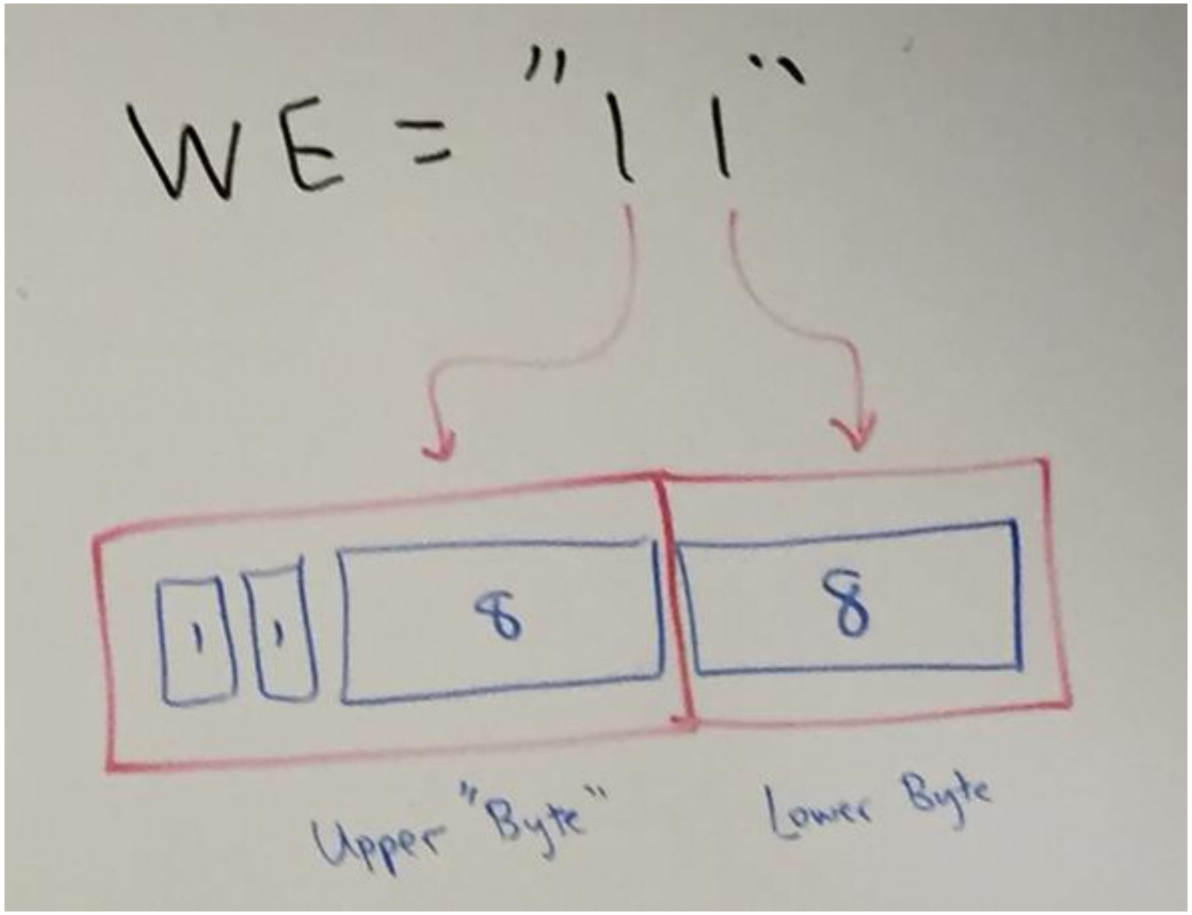
- This is a schematic symbol of BRAM memory.
- Notes:
  - Inputs are on left
  - Outputs are on the right.
  - Left top side - write functions
  - Left bottom - the read functions
- The three types of BRAMs are highly configurable, but may be overwhelming to the new designer.

Macro: Simple Dual Port RAM



BRAM read/write




WE = "11"
1
1
8
8
Upper "Byte"
Lower Byte

Datapath and Control – BRAM Example Instantiation

```
-----------------------------------------------------------------------------
-- Reference: Vivado Design Suite 7 Series FPGA Libraries Guide
--             UG953 (v 2012.4) July 25, 2012
--
-- Page:                                        10
-----------------------------------------------------------------------------
sampleMemory: BRAM_SDP_MACRO
        generic map (
                BRAM_SIZE => "18Kb",                    -- Target BRAM, "18Kb" or "36Kb"
                DEVICE => "7SERIES",                    -- Target device: "VIRTEX5", "VIRTEX6", "SPARTAN6, 7SERIES"
                DO_REG => 0,                            -- Optional output register disabled
                INIT => X"000000000000000000",          -- Initial values on output port
                INIT_FILE => "NONE",                    -- Not sure how to initialize the RAM from a file
                WRITE_WIDTH => 18,              -- Valid values are 1-72 (37-72 only valid when BRAM_SIZE="36Kb")
                READ_WIDTH => 18,               -- Valid values are 1-72 (37-72 only valid when BRAM_SIZE="36Kb")
                SIM_COLLISION_CHECK => "NONE",          -- Collision check enable "ALL", "WARNING_ONLY",
                                                                "GENERATE_X_ONLY" or "NONE"
                SRVAL => X"000000000000000000")         -- Set/Reset value for port output
```

Datapath and Control – BRAM Instantiation continued

```
port map (
        DO => readOutput,              -- Output read data port, width defined by READ_WIDTH parameter
        RDADDR => vecAddrRead,         -- Input address, width defined by port depth
        RDCLK => clk,                  -- 1-bit input clock
        RST => reset,                  -- active high reset
        RDEN => ~~cw(5)~~, '1'         -- read enable
        REGCE => '1',                  -- 1-bit input read output register enable - ignored
        DI => writeInput,              -- Input data port, width defined by WRITE_WIDTH parameter
        WE => ~~cw(3 downto 2)~~, "11" -- since RAM is byte read, this determines high or low byte
        WRADDR => vecAddrWrite,        -- Input write address, width defined by write port depth
        WRCLK => clk,                  -- 1-bit input write clock
        WREN => cw(4));                -- 1-bit input write port enable
```

```
process(clk)
begin
    if (rising_edge(clk)) then
        if (n_reset = '0') then
            addrWrite <= (others => '0');
        elsif (cw(1 downto 0) = "01") then
            addrWrite <= addrWrite + 1;
        elsif (cw(1 downto 0) = "11") then
            addrWrite <= (others => '0');
        end if;
    end if;
end process;

addrRead <= addrWrite - 1;                          -- Have the read follow the writes
writeInput <= "101010101010" & vecAddrWrite(3 downto 0);        -- Provide some changing data input

reset <= not n_reset;                               -- BRAM reset is active high
vecAddrWrite <= std_logic_vector(addrWrite);    -- type conversion (address are std_logic_vector)
vecAddrRead <= std_logic_vector(addrRead);
```

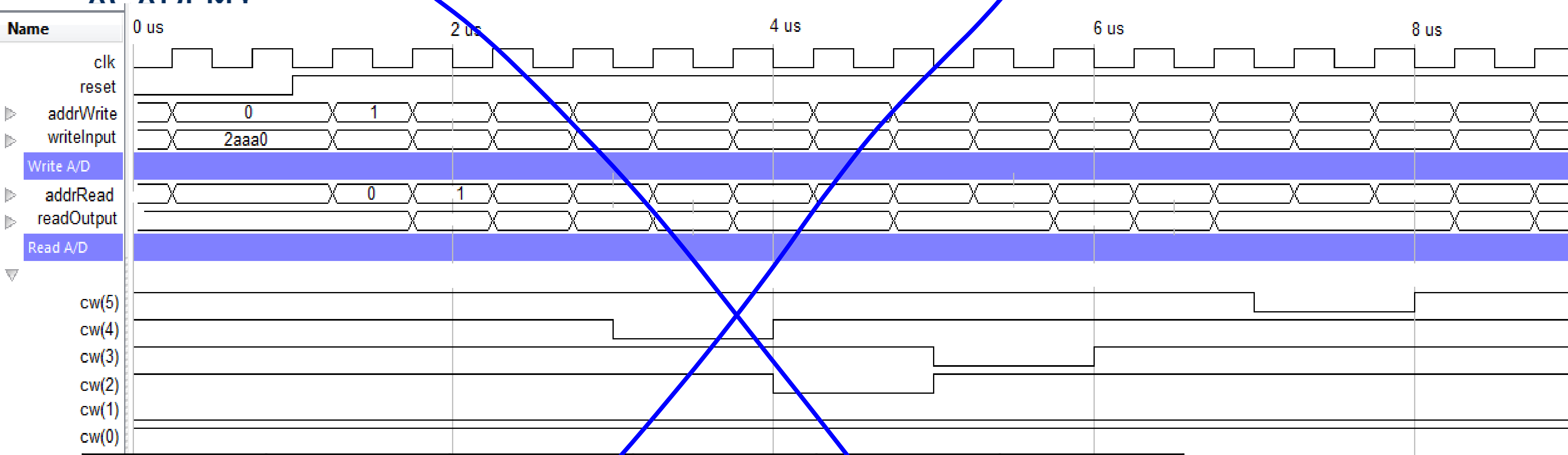
UNITED STATES
AIR FORCE
ACADEMY

Datapath and Control – Class Activity

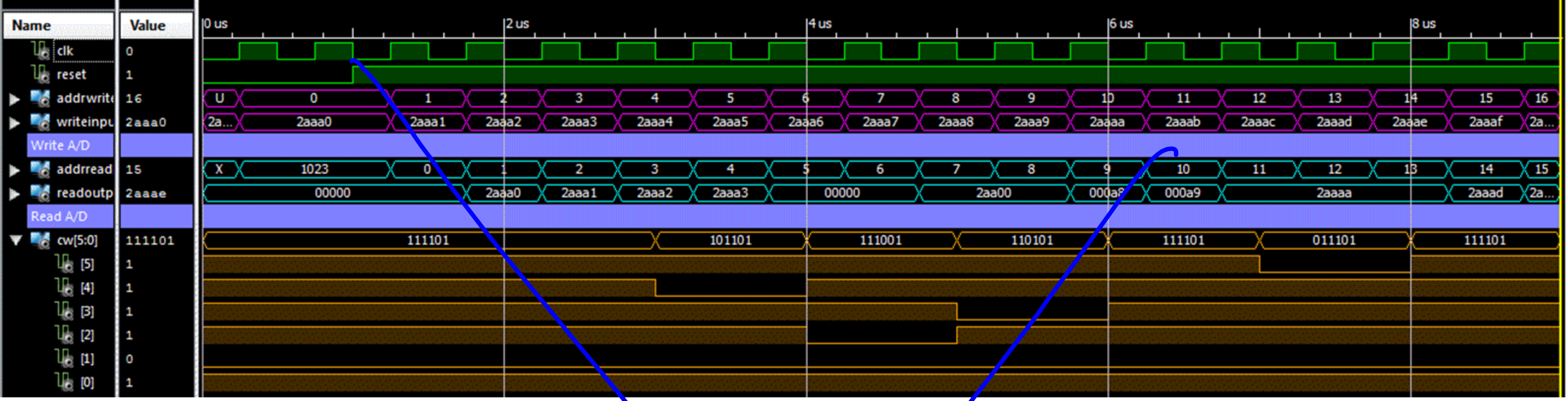
- Class Activity:
  - Determine what will happen inside the RAM defined above when subject to the following signals.

```
-- vecAddrRead = vecAddrWrite - 1
-- writeInput <= "101010101010" & vecAddrWrite(3 downto 0);
cw(5) <= '1', '0' after 7 us, '1' after 8 us;          -- READ ENABLE
cw(4) <= '1', '0' after 3 us, '1' after 4 us;          -- WRITE ENABLE
cw(3 downto 2) <= "11", "10" after 4 us, "01" after 5 us, "11" after 6us; -- BYTE WRITE ENABLE
cw(1 downto 0) <= "01";                    ?                              -- COUNTER CONTROL
```

Datapath and Control - BRAM



Addr	writeInput	WREN=cw(4)	WE=cw(3,2)	RDEN=cw(5)	readOutput
0x0	0x2AAA0				
0x1	0x2AAA1				
0x2					
0x3					
0x4					
0x5					
0x6					
0x7					
0x8	0x2AAA8				
0x9					
0xA					
0xB					
0xC					
0xD					
0xE	0x2AAAE				
0xF					



Name	Value
clk	0
reset	1
addrwrite	16
writeinpu	2aaa0
Write A/D	
addrread	15
readoutp	2aaae
Read A/D	
cw[5:0]	111101
[5]	1
[4]	1
[3]	1
[2]	1
[1]	0
[0]	1

```
addrwrite: U 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
writeinpu: 2a... 2aaa0 2aaa1 2aaa2 2aaa3 2aaa4 2aaa5 2aaa6 2aaa7 2aaa8 2aaa9 2aaaa 2aaab 2aaac 2aaad 2aaae 2aaaf 2a...
addrread:  X 1023 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
readoutp:  00000 2aaa0 2aaa1 2aaa2 2aaa3 00000 2aa00 000a8 000a9 2aaaa 2aaad 2a...
cw[5:0]:   111101 101101 111001 110101 111101 011101 111101
```

Addr	Write Input	WREN=cw(4)	WE=cw(3,2)	RDEN=cw(5)	Read Value
0x0	0x2AAA0	1	11	1	0x2AAA0
0x1	0x2AAA1	1	11	1	0x2AAA1
0x2	0x2AAA2	1	11	1	0x2AAA2
0x3	0x2AAA3	1	11	1	0x2AAA3
0x4	0x2AAA4	0	11	1	0x00000
0x5	0x2AAA5	0	11	1	0x00000
0x6	0x2AAA6	1	10	1	0x2AA00
0x7	0x2AAA7	1	10	1	0x2AA00
0x8	0x2AAA8	1	01	1	0x000A8
0x9	0x2AAA9	1	01	1	0x000A9
0xA	0x2AAAA	1	11	1	0x2AAAA
0xB	0x2AAAB	1	11	0	0x2AAAA
0xC	0x2AAAC	1	11	0	0x2AAAA
0xD	0x2AAAD	1	11	1	0x2AAAD
0xE	0x2AAAE	1	11	1	0x2AAAE
0xF	0x2AAAF	1	11	1	0x2AAAF

UNITED STATES AIR FORCE ACADEMY

Datapath and Control - Packages

- ~~**Packages are a nice way to hide lots of component declarations**~~
- ~~**Redundancy is one of the main contributors of complexity in software is redundancy.**~~
- ~~**Having an entities declaration in several different architectures is redundant.**~~
- ~~**Pulling all these declarations into one file eliminates this redundancy and make the code much easier to maintain and update.**~~
- ~~**So how do you create a Package?**~~

UNITED STATES
AIR FORCE
ACADEMY

Datapath and Control - Packages

- ~~**Packages – Package for Lab 2**~~
  - ~~http://ece.ninja/383/lecture/code/lab2_pack.vhdl~~
- ~~**Include this at the top of your file:**~~

~~**use work.lab2Parts.all; -- all my components are declared here**~~

UNITED STATES
AIR FORCE
ACADEMY

2-Line Handshake

- ~~In most cases, digital systems require data from the external world in order to perform their tasks.~~
- ~~In cases where the digital system and the outside word operate on independent clocks, the transfer of data is complicated by the lack of a common clock.~~
- ~~To understand how a reliable transfer of data can be performed in this circumstance, consider the following scenario of a producer trying to deliver a packet of candies to a consumer.~~

Figure 12.2: A timing diagram of a data transfer between a producer and a consumer.

2-Line Handshake

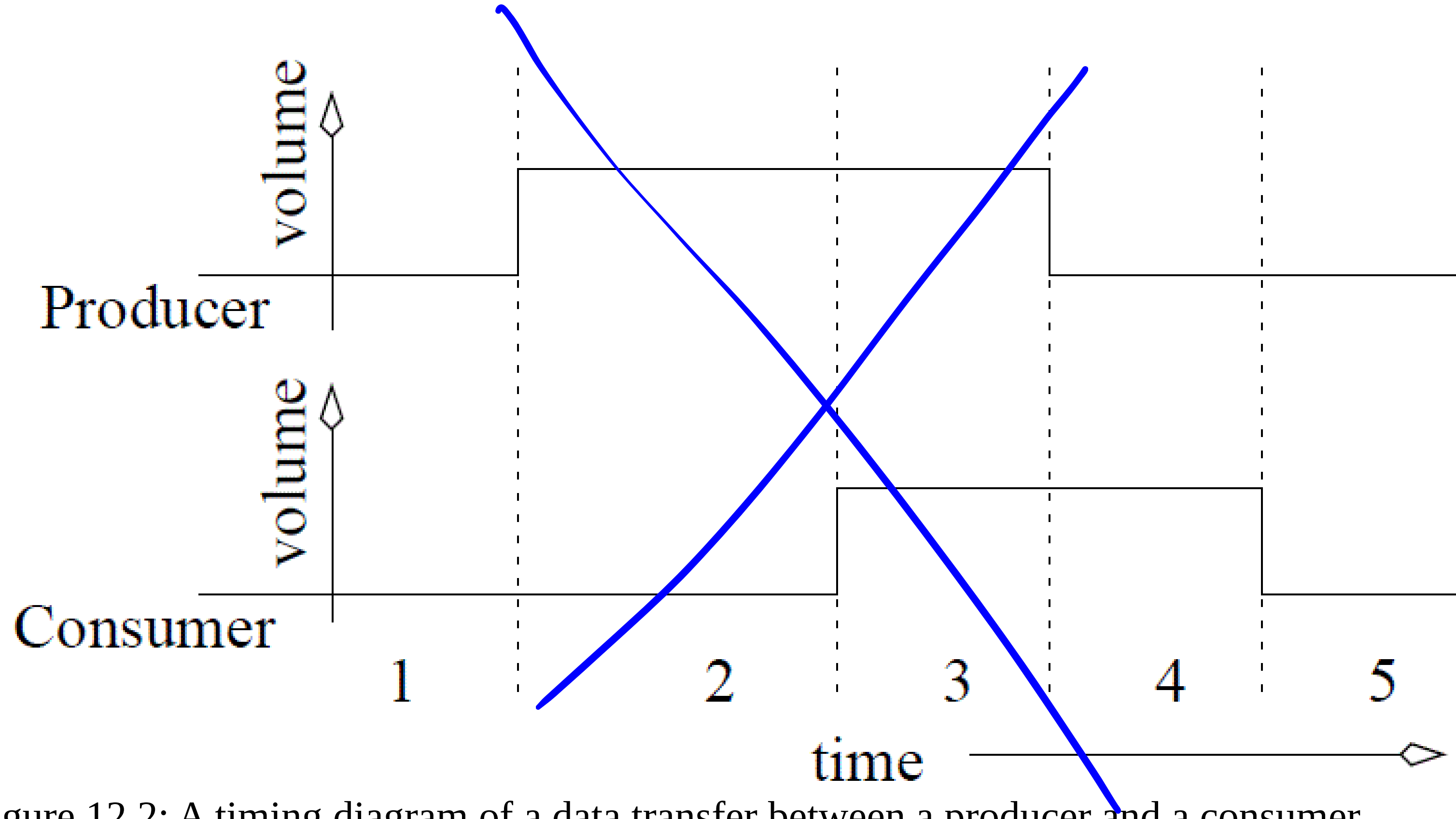


Figure 12.2: A timing diagram of a data transfer between a producer and a consumer.

UNITED STATES AIR FORCE ACADEMY

2-Line Handshake

- ~~This protocol, regardless of who is the producer or consumer, is called a two-line handshake because the communicating agents must have two, coordinating signals Request (REQ) and Acknowledge (ACK) and at least one data line.~~
- ~~REQ signal - used by the active agent to signal a readiness to perform a data transfer.~~
- ~~ACK signal - used by the passive agent to acknowledge the data has been transferred.~~

Figure 12.2: A timing diagram of a data transfer between a producer and a consumer.

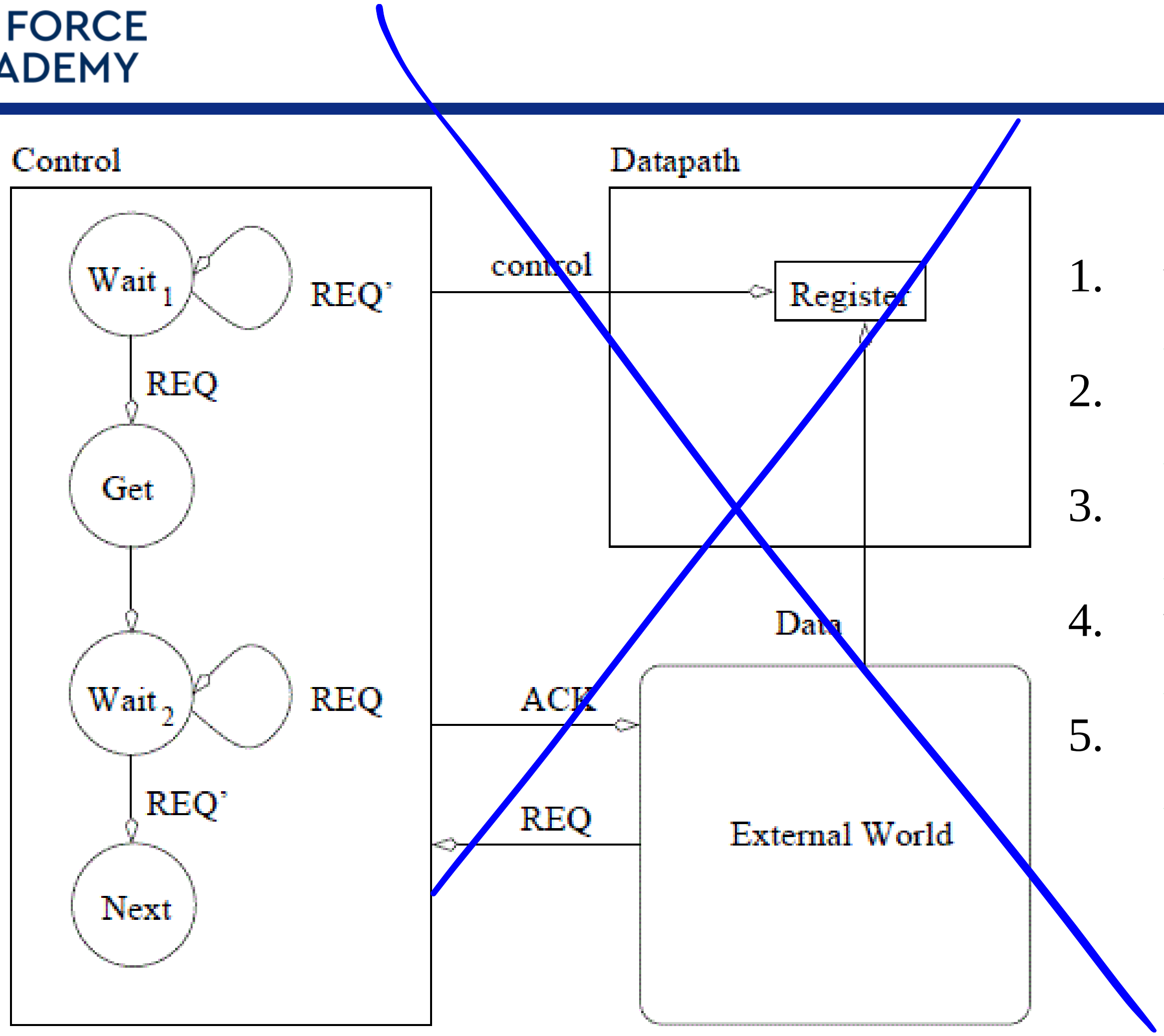
UNITED STATES
AIR FORCE
ACADEMY

2-Line Handshake

- ~~An algorithm description of the two-line handshake for a digital circuit which is the passive consumer is shown below.~~
  1. ~~while(REQ==0); // Do nothing but wait~~
  2. ~~register = DATA // Latch the data~~
  3. ~~ACK=1; // Acknowledge the producer~~
  4. ~~while(REQ==1); // Do nothing but wait~~
  5. ~~ACK=0; // Acknowledge the producer~~
- ~~In Line 1 and Line 4, the body of the while loops are empty; there is nothing to do but wait.~~
- ~~Furthermore, with respect to the external world, the ACK and REQ signals act as status and command bits, respectively.~~
- ~~The algorithm above is translated into datapath and control in Figure 12.3.~~

Figure 12.2: A timing diagram of a data transfer between a producer and a consumer.

2-Line Handshake



1. while(REQ==0);
   // Do nothing but wait
2. register = DATA
   // Latch the data
3. ACK=1;
   // Acknowledge the producer
4. while(REQ==1);
   // Do nothing but wait
5. ACK=0;
   // Acknowledge the producer

Figure 12.3: The datapath and control components required to implement a two-line handshake where the digital system is the passive consumer.

OLD

Datapath and Control – Exercise—HW#8b

- Build a circuit to read in an 8-bit KEY using a two-line handshake; the circuit is a passive consumer.
- The circuit should search an 18kx16 RAM, counting the number of words that match KEY.
- Assume the RAM is preloaded with data and it can respond to a read request with valid data within one clock

```
1. while(1) {
2.     while(REQ == 0);
3.     KEY = data;
4.     ACK = 1;
5.     while(REQ == 1);
6.     ACK = 0;
7.     match = 0;
8.     for(i=0; i<1024; i++) {
9.         MBR = RAM[i];
10.         if (MBR == KEY) {
11.             match=match+1;
12.         } // end if
13.     } // end for
14. } // end while
```

HW8b2

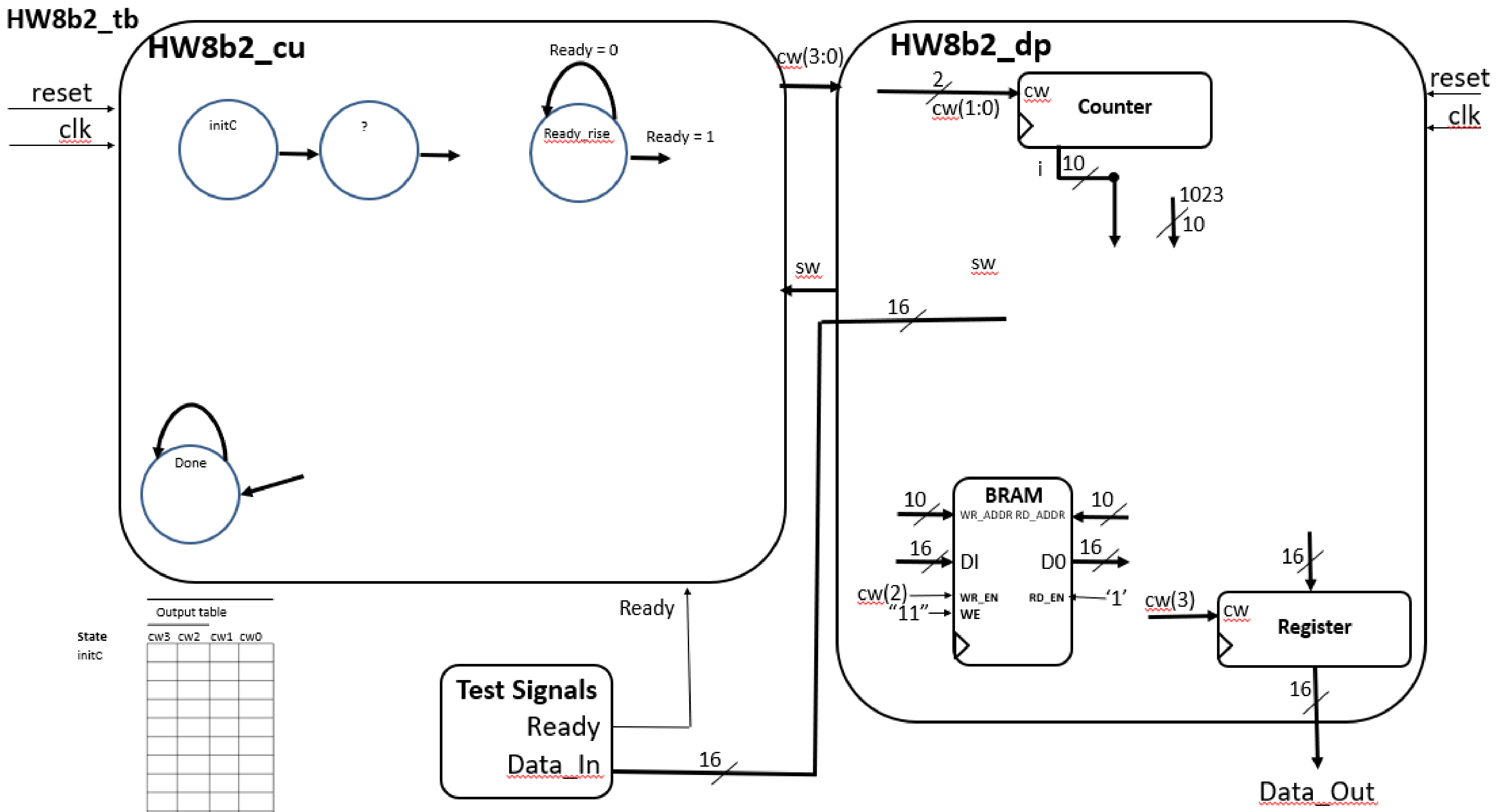
For 1023 memory locations in a BRAM, build a circuit to read in a 16-bit DATA_IN whenever READY goes HIGH, increment the data by 7, store the value in the current memory location, and then read the data stored at this memory location back out and store it in an output register called DATA_OUT. Then repeat for the next memory location.

You are given hw8b2_tb.vhdl,
You are given partial files for HW8b2_design_template.pptx, hw8b2_cu.vhdl, and hw8b2_dp.vhdl
Which you will need to complete.

Here is the mini-C to implement:

```
1.  for(i=0; i<1023; i++) {        // for every memory location
2.      while(READY==0);           // wait for ready (rising edge)
4.      RAM[i] = DATA_IN + 7; //  store data + 7 in memory location i
5.      DATA_OUT = RAM[i];  //  read data from memory location i
6.      while(READY==1);           // wait for ready (falling edge)
7.  } // end for
```

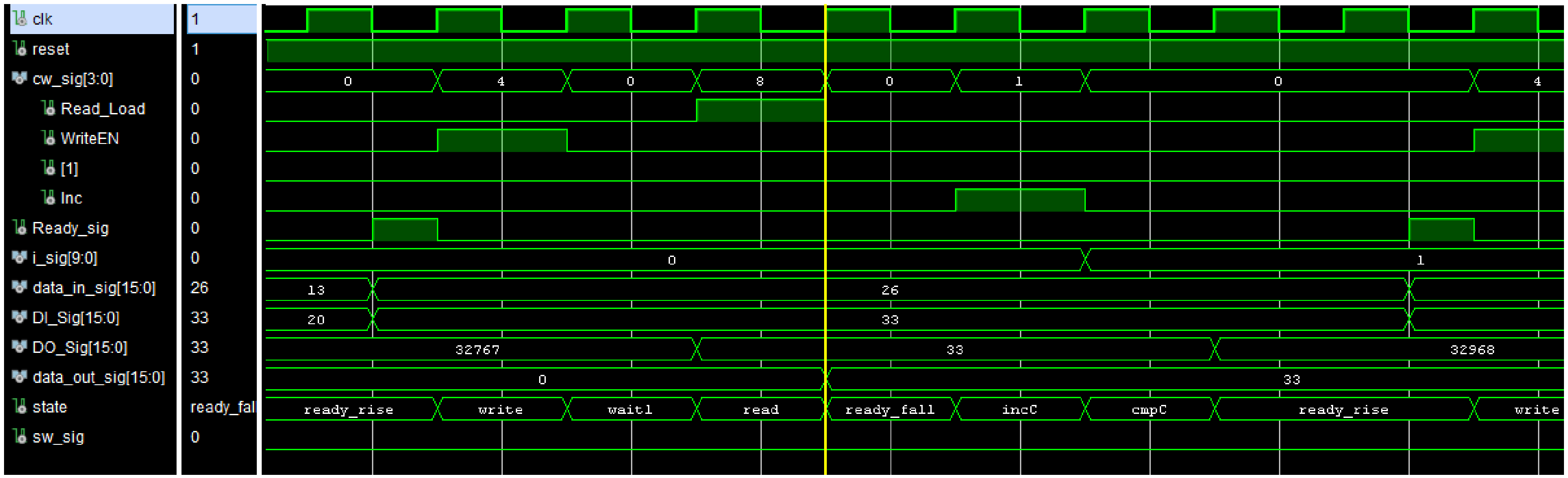
Design Template



Output table

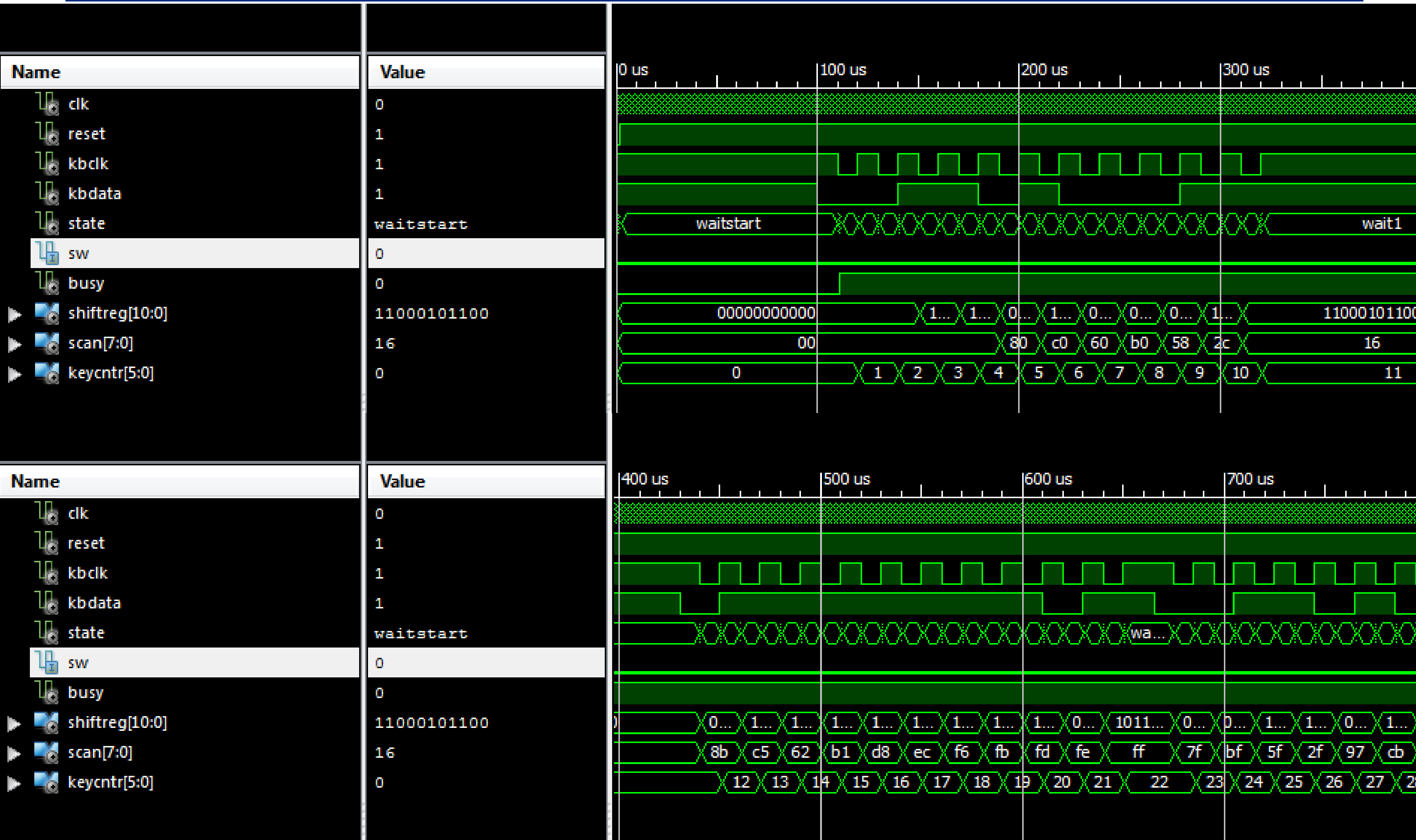
State	cw3	cw2	cw1	cw0
initC				

Example Simulation Plot



- For next class
  - Do HW8b
  - Prep for lab2
    - Read the assignment
    - Copy your lab1 files and given lab2 files, and build a lab2 project in vivado
    - GateCheck 1 due EOC next lesson
- Rest of Class today?
  - Work on HW8b?
  - Intro to Lab#2
  - GR Review?
  - HW8?

Homework #8



Name	Value
clk	0
reset	1
kbclk	1
kbdata	1
state	waitstart
sw	0
busy	0
shiftreg[10:0]	11000101100
scan[7:0]	16
keycntr[5:0]	0

Name	Value
clk	0
reset	1
kbclk	1
kbdata	1
state	waitstart
sw	0
busy	0
shiftreg[10:0]	11000101100
scan[7:0]	16
keycntr[5:0]	0

UNITED STATES
AIR FORCE
ACADEMY

HW8 solution?