CSC 343
Lab Exercise: Static Random-Access Memory (SRAM)
in VHDL and Model-SIM , Quartus, DE 2-70

March 20, 2015
Instructor: Professor Izidor Gertner

Objective:

You will create a static random-access memory chip using D-Latches. With an SRAM you are able to store multiple bits at various address locations. Your goal now is to gain knowledge on how to create a memory device that can hold 16-bit wide values and display in the 7-segment HEX displays.

NOTE: Before we start building our memory chip, we should be familiar with the operation and behavior of latches and flip-flops. The following pages have some introduction and background information on latches and flip-flops. Your actual task for this lab exercise starts on page 9.

Overview:

Latches – These are sequential devices that watch their input continuously and can change their output at ANY time (some cases require clock signal or enable signal to be asserted (set to High)).

Latches are said to be LEVEL triggered. In level triggering the circuit will become active when the gating or clock pulse is on a particular level, high or low.

Flip-flops – Sequential device that samples its input and changes its output ONLY when the clock signal is changing (at the rising or falling edge of the clock). In essence, flip-flops trigger their output based on an event (clock event).

Flip-flops are said to be EDGE triggered. In edge triggering the circuit becomes active at negative or positive edge of the clock signal. For example if the circuit is positive edge triggered, it will take input at exactly the time in which the clock signal goes from low to high. Similarly input is taken at exactly the time in which the clock signal goes from high to low in negative edge triggering.

What do we mean by "sequential" device?
For a sequential device, its output depends not only on its current input but on the past sequence of inputs, possibly arbitrarily back in time. Therefore, we can conclude that such devices have some sort of memory feature because they must remember their previous value generation in order to compute the next output.

We will now review: SR-Latch, SR-Latch with Control, D-Latch, D Flip-flop and SRAM cells.

SR-LATCH

Design:

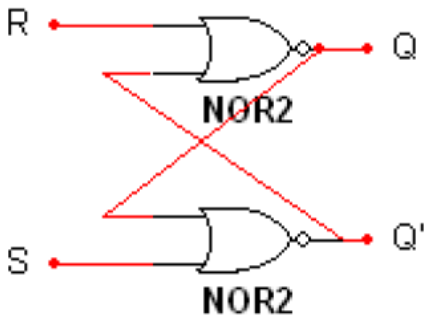

Figure 1. SR-Latch circuit using NOR gates

Functionality:

The Set/Reset Latch (SR-Latch) is a basic storage device used to store a binary bit. This latch has two functions, SET and RESET, which are controlled by the configuration of the input signals, S and R, respectively. These functions are demonstrated by the table in Figure 2. This table shows that when input S is 1 and input R is 0, the latch is in the "Set" state and its output, Q, will always be one. Likewise, when the input S is 0 and input R is 1, the latch will be in the "Reset" state and its output, Q, will always be zero. Furthermore, when both inputs are 0, the latch will hold the state of the previous output and will show no change in its output. However, when both inputs are 1, the latch will be in the undefined state; therefore, it will be impossible to predict the next state. In fact, one of the requirements of the SR latch is that the outputs Q and Q' always be compliment of each other.

The output equations for the SR-Latch are shown below in Figure 3. Notice that the output, Q, is simply R nor Q_p', which the previous Q' state.

S	R	Q	Q'	State
0	0	-	-	No Change
0	1	0	1	Reset
1	0	1	0	Set
1	1	?	?	Undefined

Figure 2. Truth table for SR-latch

$$Q = R \downarrow \hat{Q}_p$$
$$\hat{Q} = S \downarrow Q_p$$

Figure 3. Equations for the SR-latch

CONTROL SR-LATCH

Design:

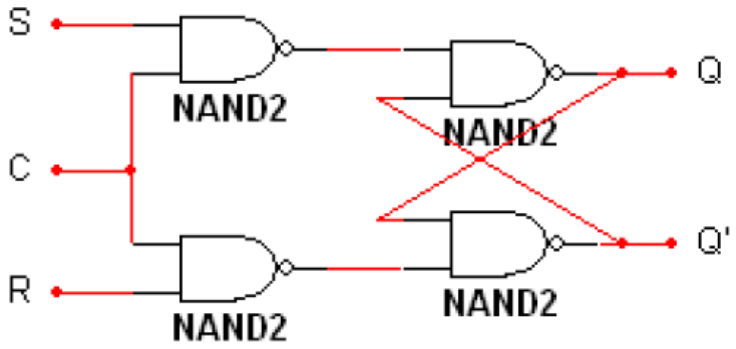

Figure 4. Control SR-Latch using NAND gates

Functionality:

This latch is similar to the SR-Latch in that it stores a binary bit, but uses an extra control but to determine state changes. For example, when the control signal, C, is 0, then the latch exhibits no changes and retains the previous state/bit. However, when the control signal is 1, the state of the latch is determined in the same manner as the regular SR-Latch using only inputs, S and R. This behavior is shown below in Figure 5, where Qn, denotes the next state of Q. The equations for the output are shown below in Figure 6, where the up arrow denotes the logical NAND operation and Q bar represents the output Q'.

C	S	R	Q_n	Q_n'	State
0	X	X	Q		
1	0	0	Q		
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	?	?	Undefined

Figure 5. Truth table for Control SR-Latch

$$Q_n = (S \uparrow C) \uparrow \hat{Q}$$
$$\hat{Q}_n = (R \uparrow C) \uparrow Q$$

Figure 6. Equations for Control SR-Latch

D-LATCH

Design:

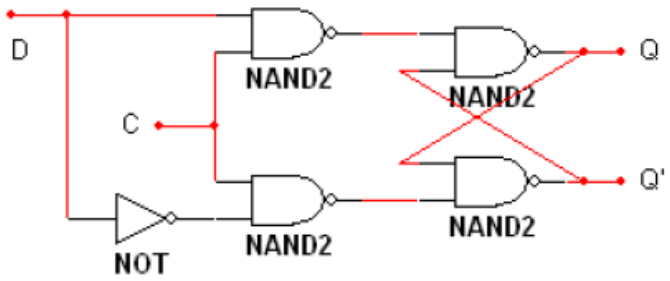

Figure 7. D-Latch using NAND gates

Functionality:

We obtain a D-Latch when we tie a NOT gate to the bottom NAND gate and only input D is used instead of independent Set and Reset. This will assure that S and R are always compliment or opposite of each other, thus eliminating falling into the undefined state when S=1, R=1.

This latch also stores a binary bit, but uses only one input signal and a clock signal. This latch is similar to the Control SR-Latch described above in that the clock signal is use in the same manner as the control signal. This relationship can be seen from the truth table of Figure 8 below. This table demonstrates that the D-Latch is a simplified version of an SR-Latch in that it has three states, Set, Reset, and No Change, but requires only a input signal and a clock rate. The equations for the output of the D-Latch can be seen in Figure 9 below, notice that the equations look very similar to that of the Control SR-Latch.

C	D	Q_n	Q_n'	State
0	X	Q	Q'	No Change
1		$Q_n = (D \uparrow C) \uparrow \hat{Q}$		t
1		$\hat{Q}_n = (\hat{D} \uparrow C) \uparrow Q$		

Figure 8. Truth table for D-Latch

Figure 9. Equations for D-Latch

MASTER-SLAVE D FLIP-FLOP

Design:

Two D-Latches are connected in series. Notice that the output, Q, of the first latch is tied to the input of the second latch, D, and both latches use the same clock signal, but the first latch inverts the clock signal.

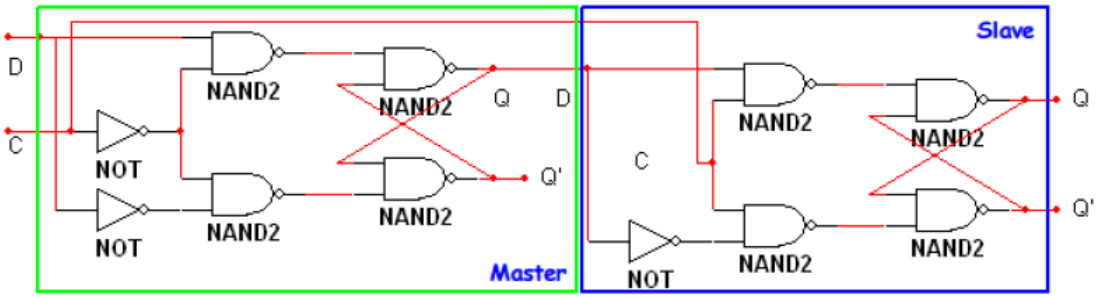

Figure10. Master-Slave D Flip-flop

It is important to note that this is a negative edge-triggered circuit, as opposed to the pulse-triggered design of the latches. This means that the output Q will equal the input D only on a clock edge (in this case, a falling edge). Therefore, Q can only change at the moment the clock goes from 1 to 0.

Functionality:

This flip-flop stores a binary bit, but uses two D-Latches in a master-slave configuration. It is different from the regular D-Latch because its state changes do not depend on the state of the clock signal, but the transition of the clock signal from zero to one. This can be seen from the truth table of Figure 11, where the left column represents the clock signal; DM is input and CM is clock for the master D-Latch. When the clock transitions from 0 to 1, the previous input is used to generate the global output for the system. The reason for this is that at state 0, the slave latch exhibits no change (retaining the global output, Q0), while the master latch generates a new input for the slave latch, q1. Then, when the clock state changes to 1, the master latch exhibits no change (retaining the previous output, q1), while the slave latch generates a new output based on the previous output (output of the master latch), Q1. Likewise, this pattern continues throughout the table.

Clock	D_M	C_M	D_S	C_S	Q_{out}
0	d1	1	q1	0	Q0
1	d2	0	q1	1	Q1
0	d3	1	q3	0	Q1
0	d4	1	q4	0	Q1
1	d5	0	q4	1	Q4
1	d6	0	q4	1	Q4

Figure 11. Truth table for Master-Slave D flip-flop

Static-RAM CELL

The SRAM is made up of static-RAM cells that are created from Master-slave D flip-flop.

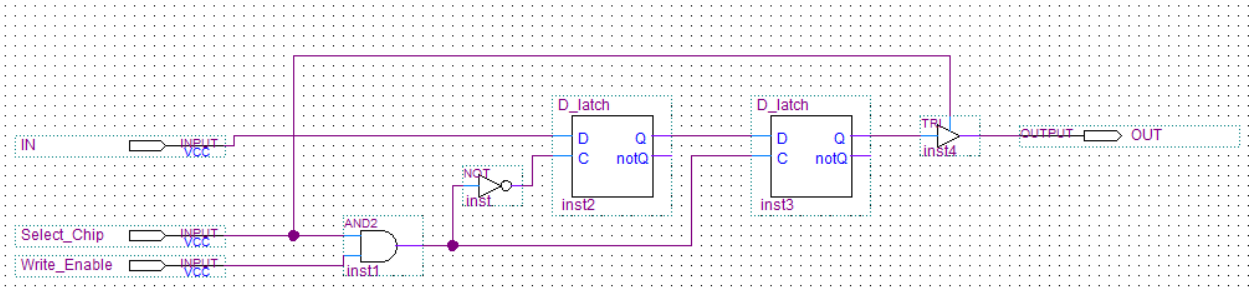

Figure 12. Static-RAM cell made of positive-edge triggered flip-flop

The static-RAM cell is shown in fig. 1. It has 3 inputs:

- IN: This is the latch data input.
- Select_Chip: This input activates the cell when it is high. If this input is low, then the cell will not store data, and there will be no output.
- Write_Enable: This input allows the latch to store the data from the IN data input.

Tristate Logic Buffer

You may have noticed a symbol in figure 12 called TRI, which you may or may not have seen before. This is a tristate logic buffer. This buffer outputs the input only when its B input is 1, otherwise the output is cut off completely and no signal is sent.

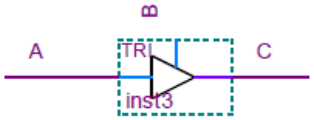

Figure 13. Tristate Logic Buffer

Input		Output
A	B	C
0	1	0
1		1
X	0	Z

Figure 14. Tristate Logic Buffer Truth Table

Figure 14 shows the truth table of the tristate logic buffer. As long as the B input is 1, the C output will equal the A input. When the B input is 0, the C output will not output anything, neither low nor high. You can find this buffer symbol in Quartus under libraries/primitives/buffer/tri (See figure 15).

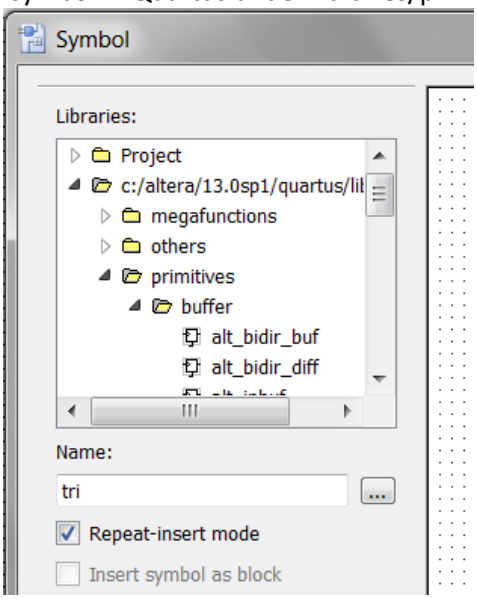

Figure 15. Libraries in Quartus showing the path to *tri* symbol

Internal Structure of a 4 x 4 Static RAM

When a static RAM is specified to be 8 x 4 that means that there are 8 storage spots that are 4 bits wide.

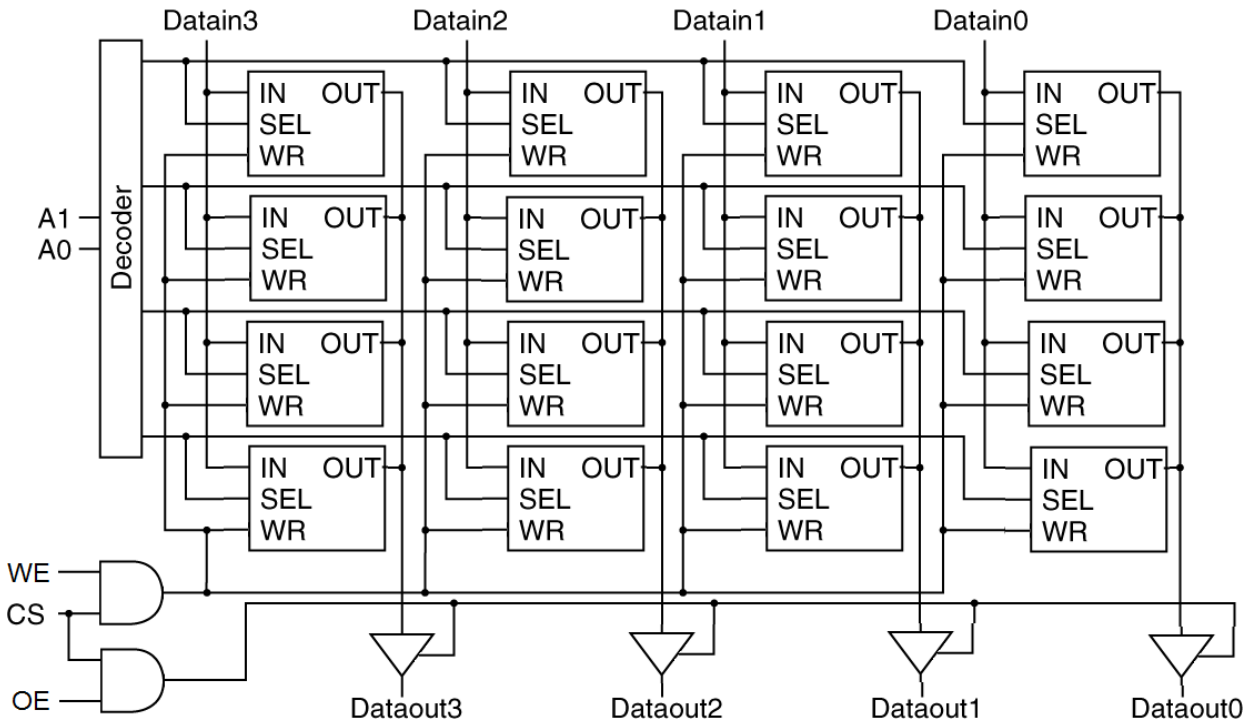

Figure 16. Internal structure of a 4x4 static RAM

Fig. 16 shows the structure of a 4 x 4 static RAM. Since there are 4 storage spots that are 4 bits wide, you will need 16 static-RAM cells to store all of the data. This schematic shows the benefits of using a tristate logic buffer. The output of each column can connect to the same output pin without the use of a multiplexer to separate the lines. As long as the tristate buffer cuts off the output of the other cells, the one cell that is on will be the only output that will be connected to the output pin.

The static RAM has several outputs:

- DataIn[3..0]: This is the input that will be written into the SRAM at a certain address.
- A[1..0]: This is the address input. This input determines which row of SRAM Cells will be turned on to write or read data. A standard 2-to-4 decoder is used to decode the address input and control the SRAM cells.
- WE: The Write Enable input. This input tells the SRAM to write to the cells when it is high. If it is low, the chip will not allow any data to be written to any cell.
- OE: The Output Enable input. When this input is high, it will allow the SRAM to output the data at the chosen address. Otherwise, the tristate buffer will cut the output of the SRAM off.
- CS: The Chip Select input. This input, when low, prevents any output from the SRAM and prevents any writing. It turns the SRAM off. Set this high to turn the SRAM on.

The data output is determined by the same address input used to write to the cells.

The A[1..0] decoder:

The function of the address decoder is to generate a one-hot code word from the address.
The output is use for row selection.

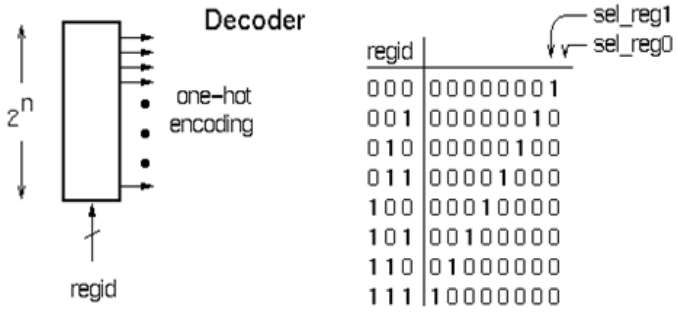

Figure 17. Decoder conceptualization

YOUR TASK

Design Problem:

- You will design an 16 x 4 SRAM by extending the SRAM shown in figure 16. The output of this SRAMwill be connected to a 4-to-7 decoder that outputs to the 7-segment-display.

Design Constraints:

- D-flip-flop – You must use D-flip-flops to design your SRAM cell
- 4-to-16 Decoder – We will provide you with the VHDL code for this decoder. No need to design a decoder yourself; however, feel free to make your own. The VHDL code for this decoder is provided below in Figure 18.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decode4to16 is
  port(
   oct : in  std_logic_vector(3 downto 0);
   dec : out std_logic_vector(15 downto 0));
end decode4to16;

architecture arch of decode4to16 is

begin

with oct select
   dec <=    "0000000000000001" when "0000",
             "0000000000000010" when "0001",
             "0000000000000100" when "0010",
             "0000000000001000" when "0011",
             "0000000000010000" when "0100",
             "0000000000100000" when "0101",
             "0000000001000000" when "0110",
             "0000000010000000" when "0111",
             "0000000100000000" when "1000",
             "0000001000000000" when "1001",
             "0000010000000000" when "1010",
             "0000100000000000" when "1011",
             "0001000000000000" when "1100",
             "0010000000000000" when "1101",
             "0100000000000000" when "1110",
             "1000000000000000" when "1111",
             "0000000000000000" when others;
end arch;
```

Figure 18. VHDL code for 4to16 decoder

Steps:

1. Open Quartus and create a new project
2. Create Block Diagram Files for SR-Latch and Control SR-Latch following the design on figures 1 and figure 4. Run waveform simulations and confirm that you obtain the same results as shown in Figures 2 and 5.
3. Create a new Block Diagram File and design a D-latch using the scheme given previously in figure 7. Convert your D-latch to a symbol.
4. Test your circuit by using a vector waveform tool and make sure that your waveform gives you the same behavior described by the truth table in Figure 8.

5. Take your D-latch symbol, duplicate it and arrange as shown in Figure 12. With this setup we are actually creating a SRAM with Master-slave D-flip-flop. Convert this SRAM circuit into a symbol.
6. Arrange 16x4 SRAM symbols using the logic of the Figure 16. Alternatively, if you find it overwhelming to stack 64 SRAM cell symbols in your screen, you can take advantage of symbol creation in Quartus. For example, in a new block diagram file, you can create a single "stack" of 16x1 SRAM cells, turn this into a symbol and then in another block diagram file join four of these 16x1 stacks together.
   Figure 19a and figure 19b exemplify this approach for a stack of 8 SRAM cells. You can follow this approach to extend it for 16:

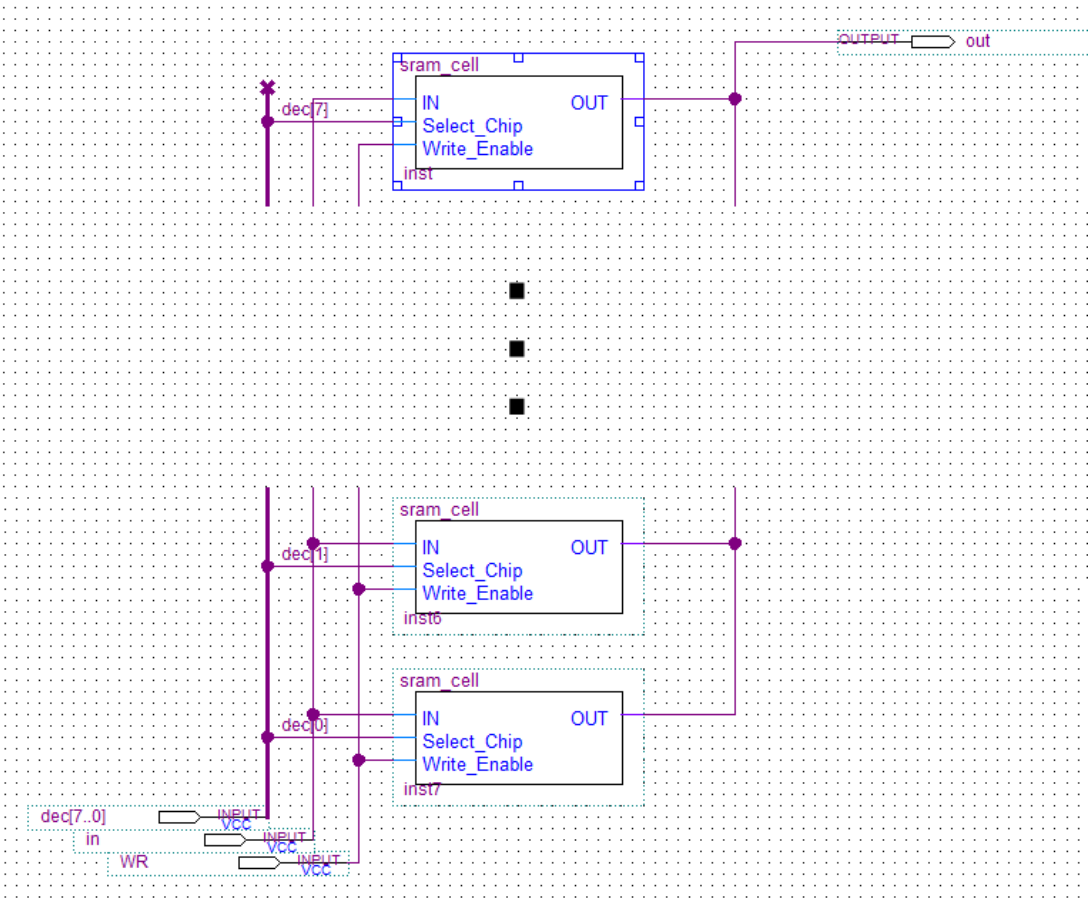

Figure 19a. Stack of 8 SRAM cells

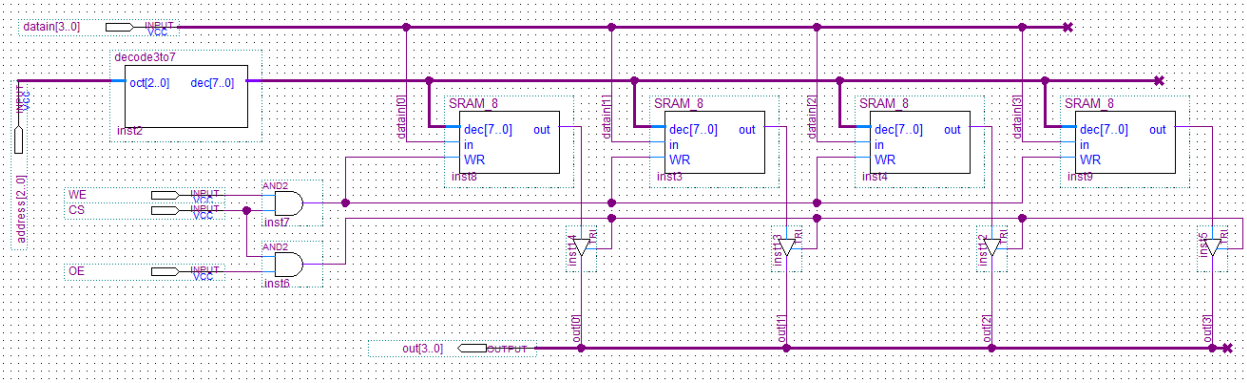

Figure 19b. 8x4 SRAM (Each SRAM symbol shown is a stack of 8 SRAM cells)

7. Run a vector waveform simulation to see if the SRAM functions as it should. Show that CS turns the chip off or on, when WE allows you to write, and when OE shows the output.
8. Convert your final SRAM into a symbol. You can now connect it to a 4-to-7 decoder which will allow you to output your results to the 7-segment display (See Appendix).
9. Pin Assignment: Create a new text file to add your pin assignments.

   DataIn[3..0]: Assign to switches 3, 2, 1, and 0.
   Address[2..0]: Assign to switches 6, 5, and 4.
   Chip_Select: Assign to switch 17.
   Output_Enable: Assign to switch 16.
   Write_Enable: Assign to switch 15.
   Lab 3 Seven Segment Decoder Output: Assign to the 7 segments of HEX display 0.

   Click here for pin assignment manual

You should now be able to answer the following questions:

- What is the difference between a latch and a flip-flop?
- Explain the behavior of a SR Latch
- Explain how the gated (control) SR-Latch is a modified SR-Latch
- Explain how the D-Latch is a modified gated SR-Latch
- Explain how to build a Master-Slave flip flop.
- Explain how to build SRAM cells from flip flops
- Why do we want to avoid the scenario when S=1 and R=1 in an SR-Latch? Could this scenario happen in a D-Latch? Why or why not?
- What is the difference between edge-triggered and level triggered devices? Is the flip-flop used for the SRAM cell in this lab level or edge triggered?

- Can you explain how an SRAM works?

APPENDIX

Output to the 7 Segment Display

For our labs, we need to output to the 7-segment display. In order to do that, we change our binary outputs to hex digits. Every group of four binary digits of our output will be mapped to a hexadecimal digit on the 7-segment display. In order to do that, we use a Case statement in VHDL. It works exactly like a switch statement in C or C++. The 7-segment display has 7 bits, each one corresponding to one of its LEDs. Depending on the input of the hex digit (4 bits) we map it to a set of corresponding 7 bits for the 7-segment display. Figure 20 shows the code that accomplishes that. Finally, we invert the 7 bits because the LED driver circuit is inverted.

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.all;
3   USE IEEE.STD_LOGIC_ARITH.all;
4   USE IEEE.STD_LOGIC_UNSIGNED.all;
5   -- Hexadecimal to 7 Segment Decoder for LED Display
6
7   ENTITY dec_to_hex IS
8   PORT( hex_digit : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9   segment_a, segment_b, segment_c, segment_d, segment_e, segment_f,
10  segment_g : OUT std_logic);
11  END dec_to_hex;
12  ARCHITECTURE a OF dec_to_hex IS
13  SIGNAL segment_data : STD_LOGIC_VECTOR(6 DOWNTO 0);
14  BEGIN
15  PROCESS (Hex_digit)
16  -- HEX to 7 Segment Decoder for LED Display
17  BEGIN -- Hex-digit is the four bit binary value to display
18
19  CASE Hex_digit IS
20  WHEN "0000" =>
21  segment_data <= "1111110";
22  WHEN "0001" =>
23  segment_data <= "0110000";
24  WHEN "0010" =>
25  segment_data <= "1101101";
26  WHEN "0011" =>
27  segment_data <= "1111001";
28  WHEN "0100" =>
29  segment_data <= "0110011";
30  WHEN "0101" =>
31  segment_data <= "1011011";
32  WHEN "0110" =>
33  segment_data <= "1011111";
34  WHEN "0111" =>
35  segment_data <= "1110000";
36  WHEN "1000" =>
37  segment_data <= "1111111";
38  WHEN "1001" =>
```

```
39    segment_data <= "1110011";
40    WHEN "1010" =>
41    segment_data <= "1110111";
42    WHEN "1011" =>
43    segment_data <= "0011111";
44    WHEN "1100" =>
45    segment_data <= "1001110";
46    WHEN "1101" =>
47    segment_data <= "0111101";
48    WHEN "1110" =>
49    segment_data <= "1001111";
50    WHEN "1111" =>
51    segment_data <= "1000111";
52    END CASE;
53    END PROCESS;
54    -- extract segment data bits and invert
55    -- LED driver circuit is inverted
56    segment_a <= NOT segment_data(6);
57    segment_b <= NOT segment_data(5);
58    segment_c <= NOT segment_data(4);
59    segment_d <= NOT segment_data(3);
60    segment_e <= NOT segment_data(2);
61    segment_f <= NOT segment_data(1);
62    segment_g <= NOT segment_data(0);
63    END a;
```

Figure 20. A 4-to-7 digit decoder to hexadecimal display