LAB 6 MEMORY LAB

CSC 343 FALL 2017

NOVEMBER, 13 2017

Jeter Gutierrez

TABLE OF CONTENTS

1.Objective

In this lab we will design static random-access memory chip using D-latches. We need to do this in order to store multiple bits of information in different locations like the functionality of Ram on our computers functions. We will also display the bits to a 7-segment display in order to show whether or not our sram is functioning correctly. It functions as a memory that we can later use to use larger bits of information and perform computations like addition or subtraction or any other computation considering that we have a limited amount of switches to input the data, which is why we need to use selectors, and sram to store our bits of information. We will have two parts that will be included in this lab the first section will have more components and we will design our own SRAM components and use the LPM component as well in order to store data, but we will then access the memory integrated to the board on the second part of this lab.

The Circuits we will be designing are for part 1 of this lab are:

1. Static-Ram CELL
2. 2-4 Decoder

3. 4x1 and 4x4 SRAM
4. 4-16 Decoder
5. 16x1 16x4 SRAM
6. 16x32 SRAM
7. 7 Segment Display
8. Ramport 1 16 4 bit words.

The purpose of the second part of the lab is to design a component that allows us to access the memory on the integrated fpga board. Each board comes with an internal sram in the case of this report we will be using the DE2-115 board for the second part of the lab and when we do that we will be using 20 bit addresses and 16 bits of data but we will only be specifying 8 bits for either case in our selection process and our input process. The same goes for storing data.

Part I SRAM AND RAMPORT 1

2.Static-Ram Cell

2.1 Functionality and specifications for Static Ram-Cell.

The static Ram cell is the component used for the SRAM in order to store data. It is made up of D-Latches. It has 3 inputs, IN which is the latch for the data input, the Select_Chip what is the input that activates only when it is 1 for the cell and if it is 0 it will not change the data and there will be no output. Write_Enable which is the input that is used to store the data from the IN pin from the first input.

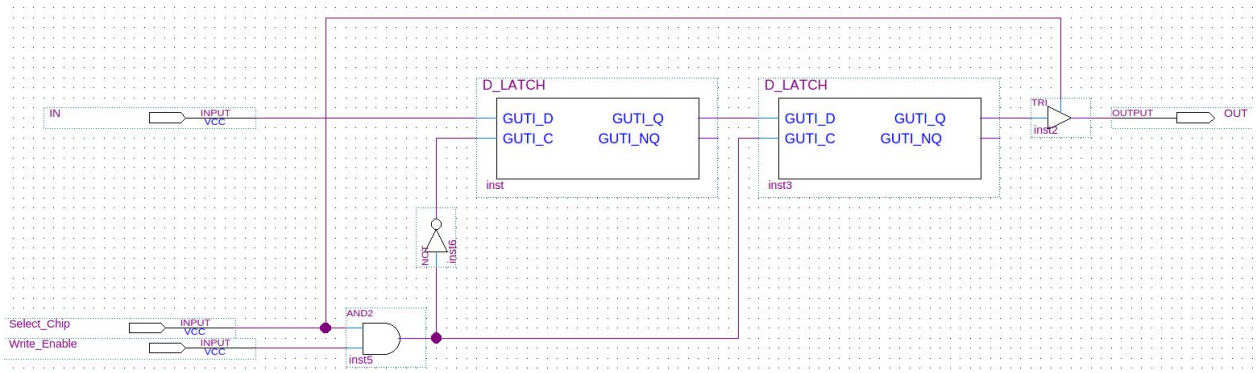


Figure 1: Block diagram for Static Ram-Cell.

2.2 Simulation for Static Ram-Cell

In this simulation IN, Select_Chip and Write_Enable input values, they will be giving different values alternating between 0 and 1 and will then be sent to OUT and we will observe the output in order to make sure that the logic of how the Static Ram cell should be working is correct.

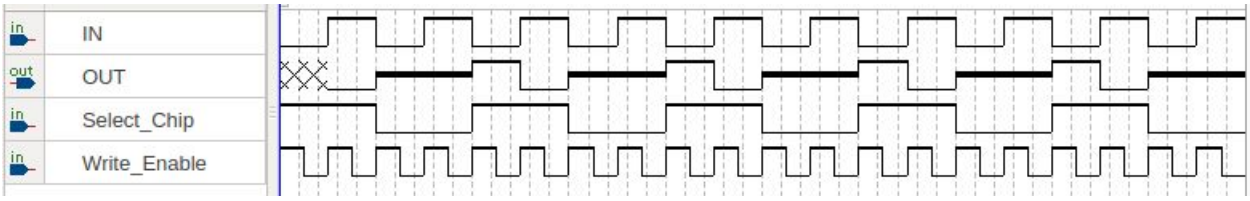


Figure 2: Vector Waveform simulation for Static Ram-Cell. As we can see the output for OUT is correct and is only shown when the chip is selected and the write is enabled which means we were able to properly design a static ram cell that remembers 1 bit of information.

3. 2-4 Decoder.

3.1 Functionality and Specifications of 2-4 Decoder.

The purpose of a decoder is to express compress data into its original form. In this case a decoder converts binary information from 2 bit to 4 bit. We are designing a 2:4 decoder where the inputs are GUTI_A and GUTI_B and the outputs are GUTI_Q0, GUTI_Q1, GUTI_Q2, and GUTI_Q3. In this case we are going to use this decoder as a component for the 4x4 SRAM the reason we

need a 2-4 decoder is so that we can have a selector in order to select which of the 4 bits of information we want to write to or read out of the 4 sections.

2:4 Decoder Truth Table

GUTI_A	GUTI_B	GUTI_Q0	GUTI_Q1	GUTI_Q2	GUTI_Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

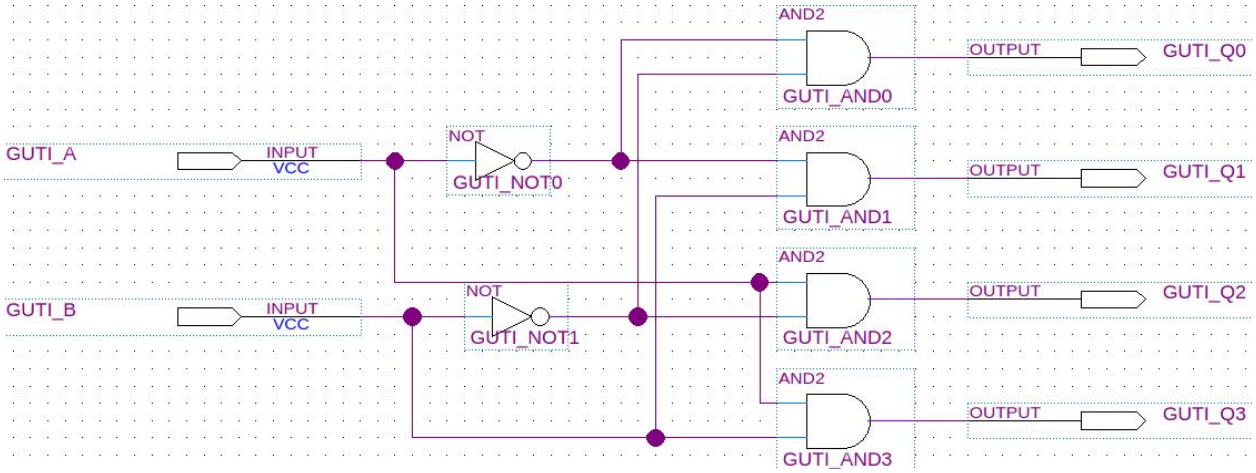


Figure 3: Block Diagram of 2-4 Decoder.

3.2 Simulation for 2-4 Decoder.

In this simulation, we will alternate values for GUTI_A and GUTI_B between 0 and 1, the output will then be projected and compared to the truth table.

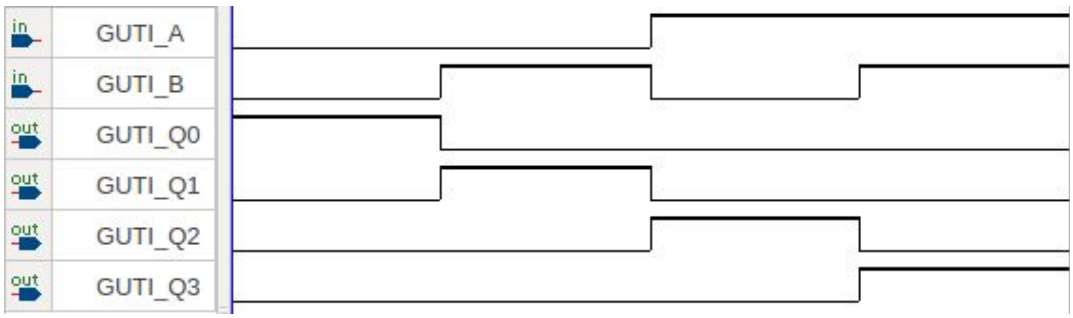


Figure 4: Vector Waveform Corresponding to 2:4 Decoder.

The simulation matches accordingly and correctly with the truth table and we were able to design a functioning 2:4 decoder.

3.3 Demonstration of 2:4 Decoder on DE1-SoC Board.

The input and outputs that are assigned to pins on the DE1-SoC Board are:

GUTI_A is assigned to PIN_AB12 SW[0]

GUTI_B is assigned to PIN_AC12 SW[1]

GUTI_Q0 is assigned to PIN_V16 LED[0]

GUTI_Q1 is assigned to PIN_W16 LED[1]

GUTI_Q2 is assigned to PIN_V17 LED[2]

GUTI_Q3 is assigned to PIN_V18 LED[3]

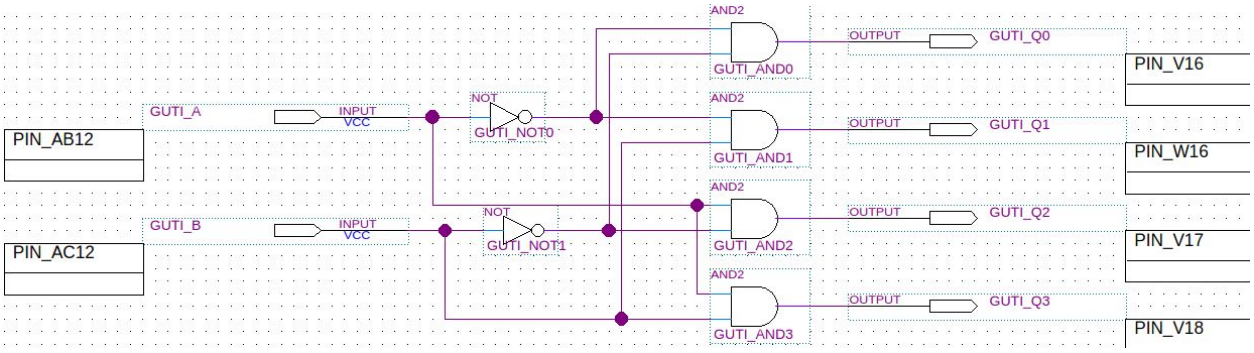


Figure 5: Pin Assignments of 2:4 Decoder on DE1-SoC Board.

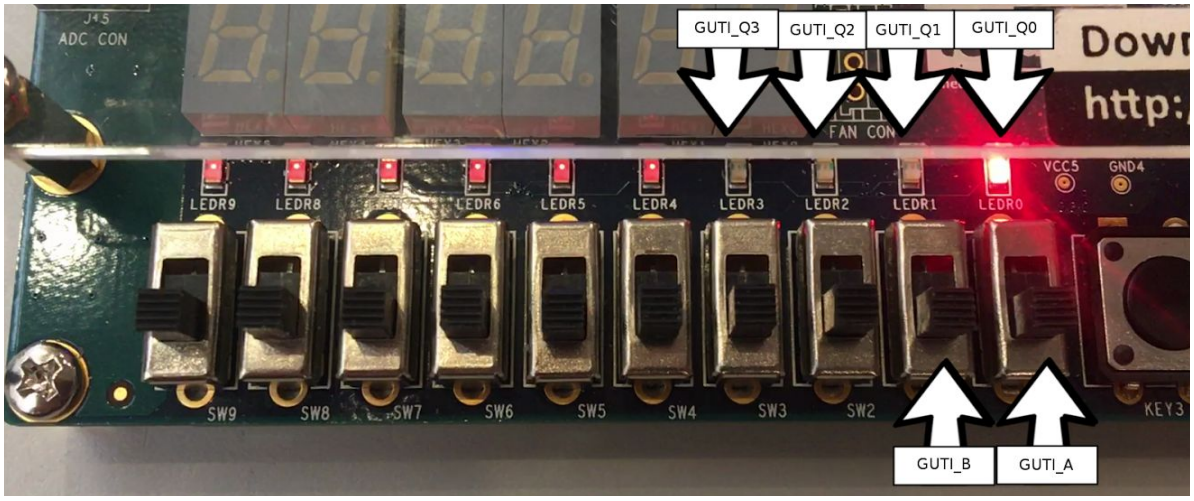


Figure 6: Digital Circuit of 2:4 Decoder on DE1-SoC Board where all inputs are 0, and GUTI_Q0 is 1.

4 4x1 and 4x4 SRAM

4.1 Functionality and specifications for 4x1 and 4x4 SRAM

The purpose of the 4xr static ram is to have 4 storage locations each being 4 bits wide. This will be used in order to test the way that the SRAM works for larger bits of information. In this case in order to make the design process simpler we will design a block for the sram that stores 4 bits of information. Then we will use the same block in a larger diagram that will then be used to design the 4x4 sram using the 2 to 4 decoder and the sram of 4 bits.

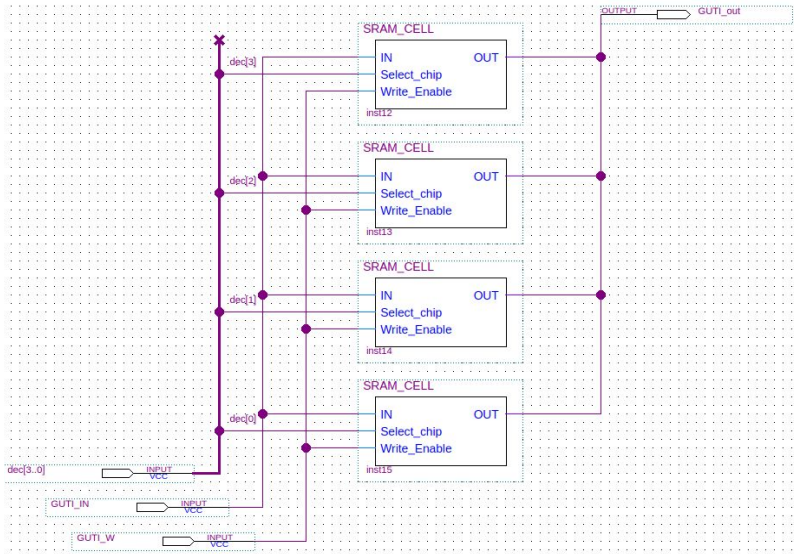


Figure 7: Block diagram for SRAM using 4 bits of information. It uses Static Ram cells in order to be organized to save 4 bits of information.

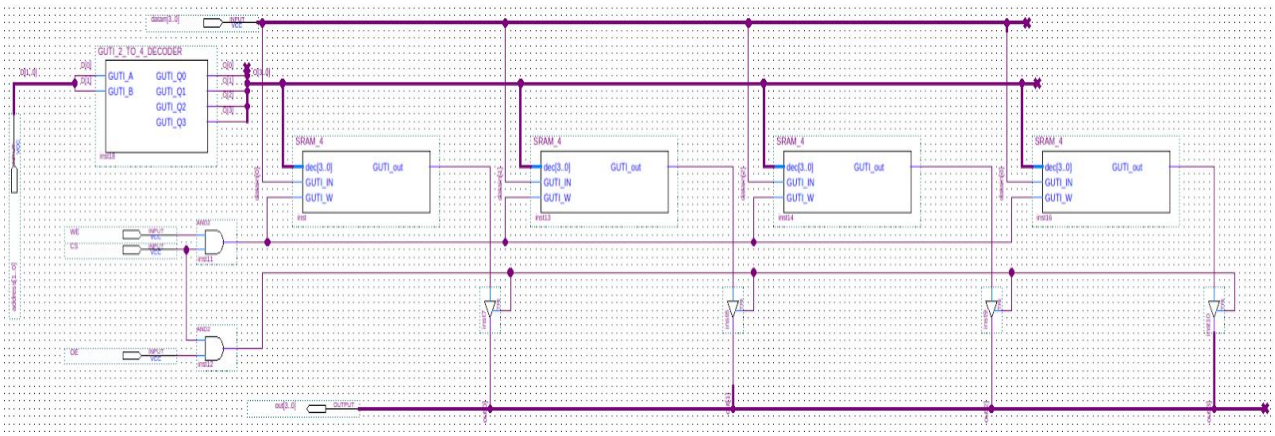


Figure 8: Block diagram for 4x4 SRAM using a 2-4 decoder to select from 4 of the srams and then choose which one to write into and which one to read from. The purpose of this is to access 4 addresses and then to also write or read to these locations, then we can use the same method in order to write using 16x4 or 16 locations and 4 bits of information each.

5 4-16 Decoder

5.1 Functionality and specifications for 4-16 decoder.

The purpose for the 4-16 decoder is to have 4 switches that will be used as a selector for the 16 different locations in our 16x4 SRAM and our 16x32 selectors, the same applies because it will alway have only one value present. This way we can write to 16 locations and read from 16 locations and are getting closer to the purpose of the Ram on our computes. It is necessary for us to make this decoder because we can't use the 2-4 decoder in this case, however we will make the decoder using VHDL in order to make the process without using as many gates as we would normally in order to create the decoder.

```
1   library IEEE;--JETER GUTIERREZ
2   use IEEE.STD_LOGIC_1164.all;--JETER GUTIERREZ
3   entity decode4to16 is --JETER GUTIERREZ
4   port(--JETER GUTIERREZ
5   oct : in std_logic_vector(3 downto 0);--JETER GUTIERREZ
6   dec : out std_logic_vector(15 downto 0));--JETER GUTIERREZ
7   end decode4to16;--JETER GUTIERREZ
8   architecture arch of decode4to16 is--JETER GUTIERREZ
9   begin--JETER GUTIERREZ
10  with oct select--JETER GUTIERREZ
11  dec <= "0000000000000001" when "0000",--JETER GUTIERREZ
12  "0000000000000010" when "0001",--JETER GUTIERREZ
13  "0000000000000100" when "0010",--JETER GUTIERREZ
14  "0000000000001000" when "0011",--JETER GUTIERREZ
15  "0000000000010000" when "0100",--JETER GUTIERREZ
16  "0000000000100000" when "0101",--JETER GUTIERREZ
17  "0000000001000000" when "0110",--JETER GUTIERREZ
18  "0000000010000000" when "0111",--JETER GUTIERREZ
19  "0000000100000000" when "1000",--JETER GUTIERREZ
20  "0000001000000000" when "1001",--JETER GUTIERREZ
21  "0000010000000000" when "1010",--JETER GUTIERREZ
22  "0000100000000000" when "1011",--JETER GUTIERREZ
23  "0001000000000000" when "1100",--JETER GUTIERREZ
24  "0010000000000000" when "1101",--JETER GUTIERREZ
25  "0100000000000000" when "1110",--JETER GUTIERREZ
26  "1000000000000000" when "1111",--JETER GUTIERREZ
27  "0000000000000000" when others;--JETER GUTIERREZ
28  end arch;--JETER GUTIERREZ
```

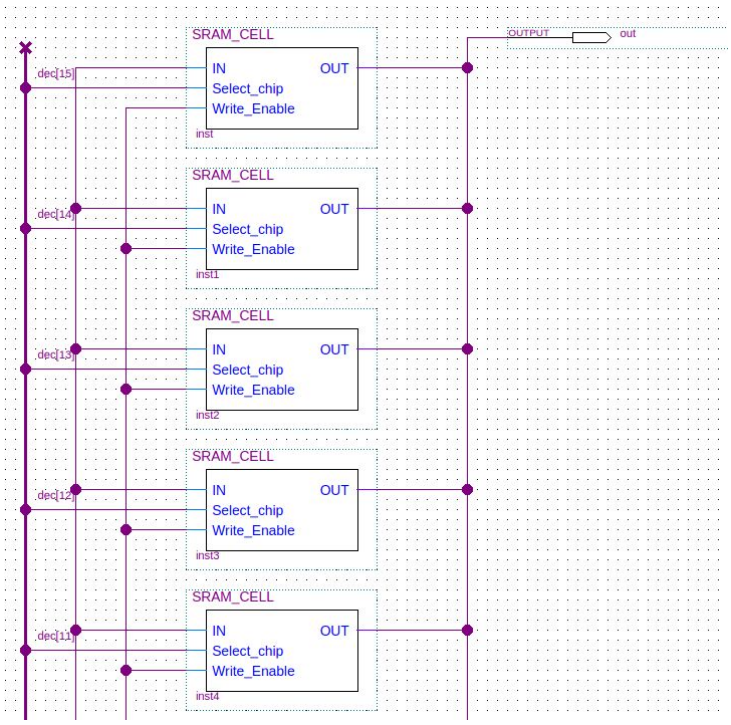
Figure 9: VHDL code for 4-16 decoder. The 4-16 decoder only allows one output in any instance, the input is any value between 0000 and 1111 in which each of the combinations determines the location of any of the other sections. Each value will determine only one of the

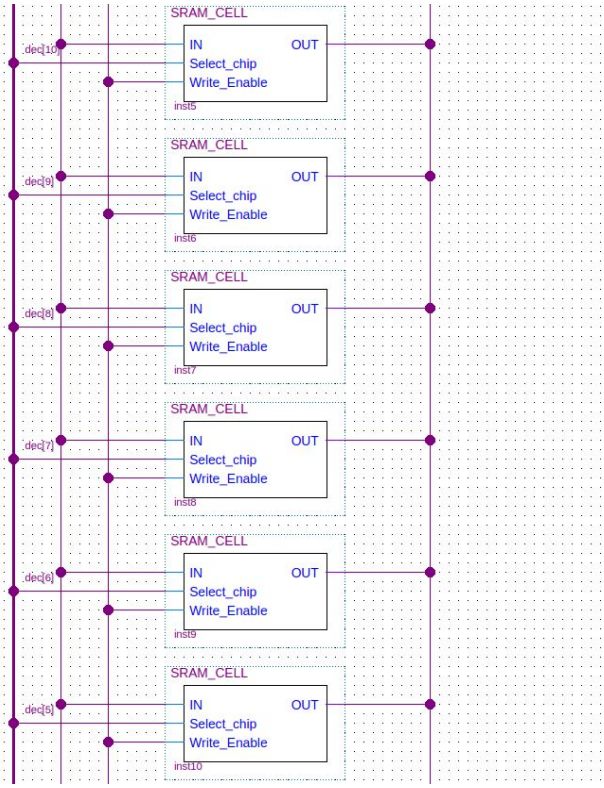
outputs to be 1 and the rest of them will be 0, this way only one section will be selected from our 16x4 or 16x32 SRAM components.

6 16x1 16x4 SRAM

6.1 Functionality and specifications for 16x1 and 16x4 Sram.

The purpose of the 16x1 component is to store 16 bits of information in order to use it for our 16x4 SRAM. In this case however we will make a 16x1 SRAM that will then be used to use any of the 4 sections in order to use less components when designing the SRAM and writing and reading to the desired location out of 16 locations. The final component in this section will be the 16x4 SRAM that will be used to write or read to 16 different locations using 4 bits of information to each location.




SRAM_CELL
IN
OUT
Select_chip
Write_Enable
dec[10]
inst5
SRAM_CELL
dec[9]
inst6
SRAM_CELL
dec[8]
inst7
SRAM_CELL
dec[7]
inst8
SRAM_CELL
dec[6]
inst9
SRAM_CELL
dec[5]
inst10

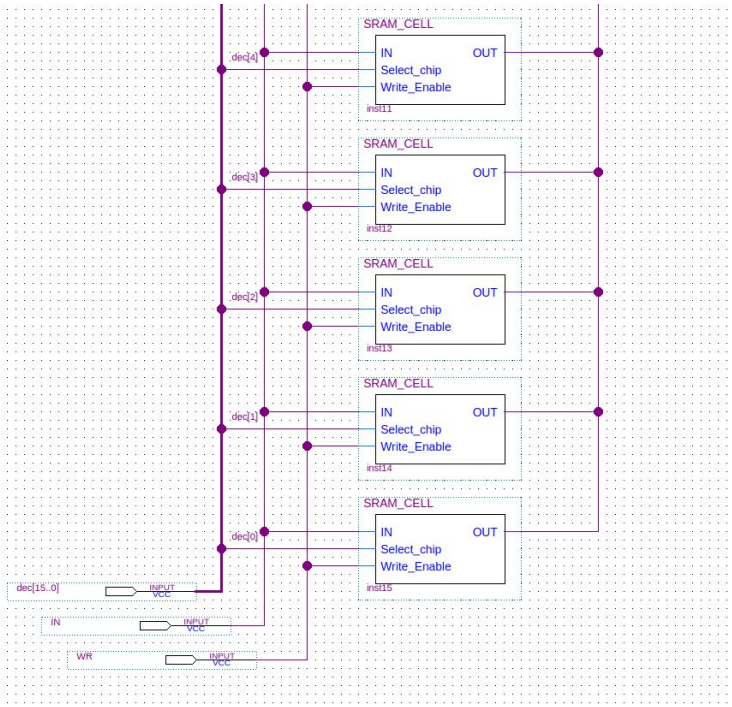

SRAM_CELL
IN
OUT
Select_chip
Write_Enable
dec[4]
inst11
SRAM_CELL
dec[3]
inst12
SRAM_CELL
dec[2]
inst13
SRAM_CELL
dec[1]
inst14
SRAM_CELL
dec[0]
inst15
dec[15..0]
INPUT
VCC
IN
WR

Figure 10: Block diagram for 16x1 SRAM where it has 16 inputs and only 16 outputs as well, the inputs come from a decoder and it also takes an input

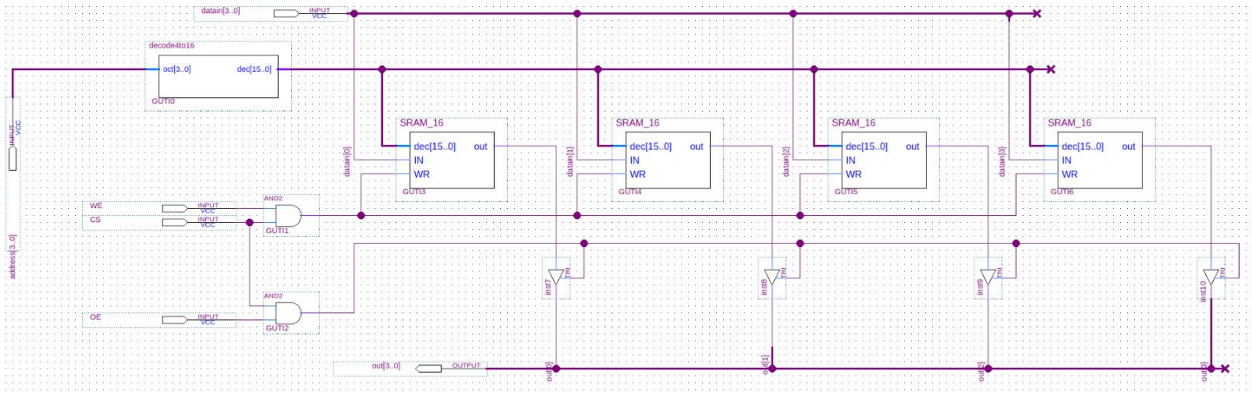


Figure 11: Block diagram of 4x16 decoder and 16x1 sram that will be used for the 16x4 sram. The purpose of this design is to have 16 locations that will be given by the 4x16 decoder and sends a signal to the SRAM components, for which only one value will be 1 out of 16, that means there will be 15 zeros and one 1. The section of the Sram that has a 1 is the location that will be accessed, then based on the CS and the WE and OE we will determine whether or not that location will be write enabled and if we can read the values that are stored that location. Then we can read and write 45 bits of information to 16 different locations.

7 16X32 SRAM

7.1 Functionality and specifications for 16x32 SRAM

The purpose of this component is to design an SRAM that will be used to access 16 different locations by using the addresses from [3..0] and then input the different values to store from datain[31..0] which represents the 32 bits of information that will be sent to the specified SRAM location out of 16 and then it can be stored and read if write enable is on and if the reading value is also one.

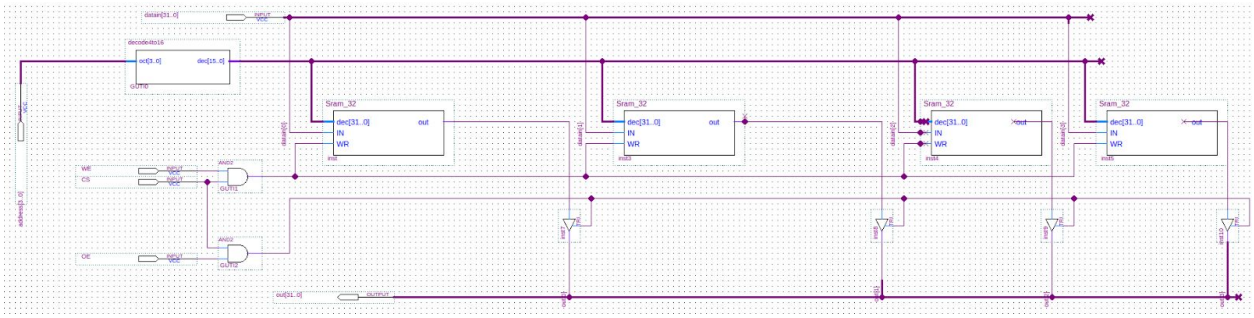


Figure 12: Block diagram for 16x32 SRAM the purpose of this block is to have the inputs that are from 31 down to 0 in order to represent 32 bits of information that can be stored and read, we were able to correctly design this component.

8 7 segment Display

The purpose of the 7 segment display is for us to display the 4 bits of information that will be stored in the specified section in our 16x4 SRAM. The code for the 7 segment display will be written in VHDL so that we can display hexadecimal values, we want to displays values from 0 to 15 which becomes from 0 to F. That means that we are displaying the values correctly for the sram in order to read the location and determine whether or not we are storing values correctly which we are. This is most necessary for a test in order to make sure our SRAM is keeping track of the values and functioning correctly as a memory buffer.

```
library IEEE;--JETER GUTIERREZ
use IEEE.STD_LOGIC_1164.all;--JETER GUTIERREZ
use IEEE.STD_LOGIC_ARITH.all;--JETER GUTIERREZ
use IEEE.STD_LOGIC_UNSIGNED.all;--JETER GUTIERREZ
--JETER GUTIERREZ--JETER GUTIERREZ
entity dec_to_hex is--JETER GUTIERREZ
port(hex_digit: in std_logic_vector(3 downto 0);--JETER GUTIERREZ
segment_a,segment_b,segment_c,segment_d,segment_e,segment_f,segment_g : out
std_logic);--JETER GUTIERREZ
end dec_to_hex;--JETER GUTIERREZ
--JETER GUTIERREZ--JETER GUTIERREZ
architecture a of dec_to_hex is--JETER GUTIERREZ
signal segment_data : std_logic_vector (6 downto 0);--JETER GUTIERREZ
```

```
begin--JETER GUTIERREZ
process (Hex_digit)--JETER GUTIERREZ
Begin--JETER GUTIERREZ
case Hex_digit is --JETER GUTIERREZ
When "0000" => --JETER GUTIERREZ
segment_data <= "1111110";--JETER GUTIERREZ
When "0001" =>
segment_data <= "0110000";--JETER GUTIERREZ
When "0010" =>
segment_data <= "1101101";--JETER GUTIERREZ
When "0011" =>
segment_data <= "1111001";--JETER GUTIERREZ
When "0100" =>
segment_data <= "0110011";--JETER GUTIERREZ
When "0101" =>
segment_data <= "1011011";--JETER GUTIERREZ
When "0110" =>
segment_data <= "1011111";--JETER GUTIERREZ
When "0111" =>
segment_data <= "1110000";--JETER GUTIERREZ
When "1000" =>
segment_data <= "1111111";--JETER GUTIERREZ
When "1001" =>
segment_data <= "1110011";--JETER GUTIERREZ
When "1010" =>
segment_data <= "1110111";--JETER GUTIERREZ
When "1011" =>
segment_data <= "0011111";--JETER GUTIERREZ
When "1100" =>
segment_data <= "1001110";--JETER GUTIERREZ
When "1101" =>
segment_data <= "0111101";--JETER GUTIERREZ
When "1110" =>
segment_data <= "1001111";--JETER GUTIERREZ
When "1111" =>
segment_data <= "1000111";--JETER GUTIERREZ
end case;--JETER GUTIERREZ
end process;
--JETER GUTIERREZ
segment_a <= NOT segment_data(6);--JETER GUTIERREZ
segment_b <= NOT segment_data(5);--JETER GUTIERREZ
segment_c <= NOT segment_data(4);--JETER GUTIERREZ
segment_d <= NOT segment_data(3);--JETER GUTIERREZ
segment_e <= NOT segment_data(2);--JETER GUTIERREZ
segment_f <= NOT segment_data(1);--JETER GUTIERREZ
```

```
segment_g <= NOT segment_data(0);--JETER GUTIERREZ

End a;
--JETER GUTIERREZ
```

Figure 13: VHDL code for 7 segment display in hexadecimal format.

9 Demonstration of 16x4 SRAM

In this section we will combine the 4x16 decoder and the 16x4 SRAM and the 7 segment display in order to program a FPGA programmable DE1-SoC board. The reason that we need to combine the components is to completely finish the project and show that we were able to successfully design a 16x4 SRAM that is going to be useful and can help us with further advancing in the course. In this design we will give all of the inputs into the board using switches and we will view the outputs in the 7-segment display which can be expressed in hexadecimal format which is going to be very convenient for us and easier to use thanks to the vhdl code we are able to design.

The inputs and outputs that are assigned to pins on the DE1-SoC board are:

GUTI_datain[0] is assigned to PIN_AB12

GUTI_datain[1] is assigned to PIN_AC12

GUTI_datain[2] is assigned to PIN_AF9

GUTI_datain[3] is assigned to PIN_AF10

GUTI_address[0] is assigned to PIN_AD11

GUTI_address[1] is assigned to PIN_AD12

GUTI_address[2] is assigned to PIN_AE11

GUTI_address[3] is assigned to PIN_AC9

GUTI_CS is assigned to PIN_AD10

GUTI_OE is assigned to PIN_AE12

GUTI_WE is assigned to PIN_Y16

GUTI_segment_a is assigned to PIN_AE26

GUTI_segment_b is assigned to PIN_AE27

GUTI_segment_c is assigned to PIN_AE28

GUTI_segment_d is assigned to PIN_AG27

GUTI_segment_e is assigned to PIN_AF28

GUTI_segment_f is assigned to PIN_AG28

GUTI_segment_g is assigned to PIN_AH28

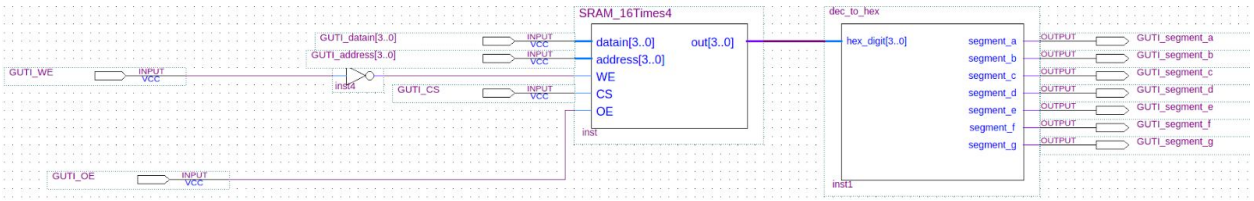


Figure 14: Block Diagram of the final product that will be used and assigned to the Programmable FPGA DE1-SoC Board.

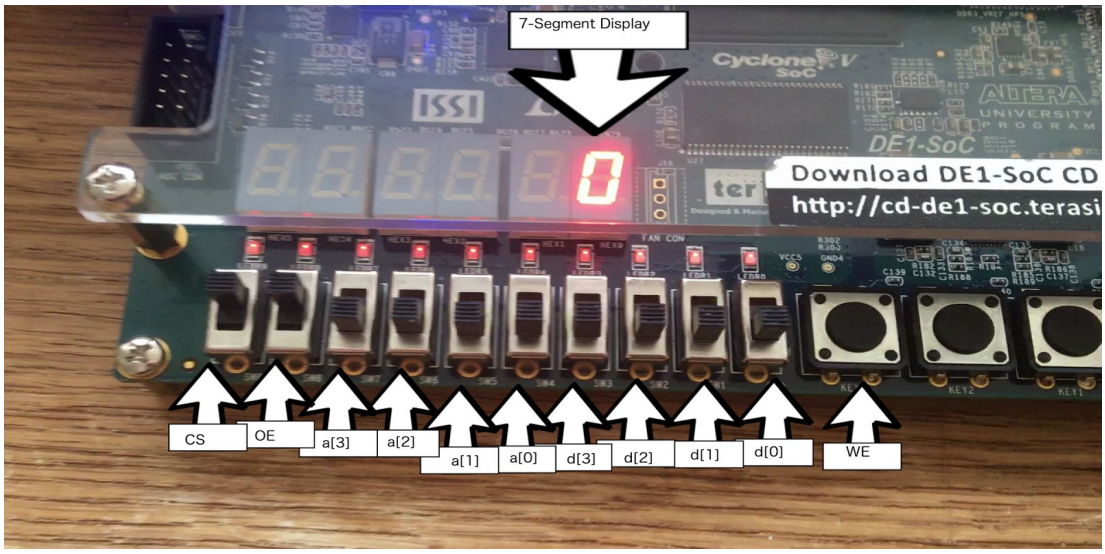


Figure 15: Digital Circuit of 16x4 SRAM where the inputs are all the 10 switches and one of the keys, the reason that the names of each the switches in the image are abbreviated is so that the fit in the image, each of the 16 locations are set to be 0 by default before they are written to instead of setting them to another default value in order to prevent complications.

10 RAMPORT 1 LPM COMPONENT

10.1 Functionality and specification for RAMPORT 1 LPM COMPONENT

The purpose of this component is to implement and test the LPM quartus designed SRAM example. We have just designed our own SRAM but in this case we will be using the one that was created using LPM in order to then compare and notice if our design was done correctly. We will be using a WE signal as well and be able to store to 16 different addresses, each word will be 4 bits wide.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY RAMPORT1 IS
  PORT
  (
    address    : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    clock    : IN STD_LOGIC  := '1';
    data     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    wren     : IN STD_LOGIC ;
    q    : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
  );
END RAMPORT1;


ARCHITECTURE SYN OF ramport1 IS

  SIGNAL sub_wire0  : STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN
  q    <= sub_wire0(3 DOWNTO 0);

  altsyncram_component : altsyncram
  GENERIC MAP (
    clock_enable_input_a => "BYPASS",
    clock_enable_output_a => "BYPASS",
    intended_device_family => "Cyclone IV E",
    lpm_hint => "ENABLE_RUNTIME_MOD=NO",
    lpm_type => "altsyncram",
    numwords_a => 16,
    operation_mode => "SINGLE_PORT",
    outdata_aclr_a => "NONE",
    outdata_reg_a => "CLOCK0",
    power_up_uninitialized => "FALSE",
    read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
    widthad_a => 4,
    width_a => 4,
    width_byteena_a => 1
  )
  PORT MAP (
    address_a => address,
    clock0 => clock,
    data_a => data,
    wren_a => wren,
    q_a => sub_wire0
  );


END SYN;
```

Figure 16: VHDL code for RAM 1 port that is 4 bits wide and can stored up to 16 addresses.

10.2 SIMULATION FOR RAMPORT 1 COMPONENT

In this simulation we will be giving different addresses for the RAMPORT 1 to write to. We will be alternating the clock signal, keeping write enable on and using random values to store the data into, then we will be reading values from the addresses that will be sent to q. We will then analyze the simulation to interpret its correctness.

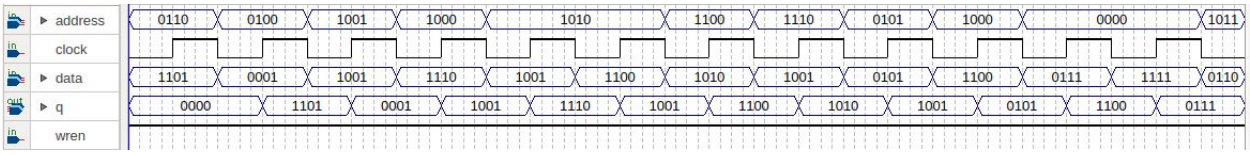


Figure 17: Simulation for RAMPORT 1 component. As we can see the values are being stored and displayed correctly when the address is being used, there is a clock delay as expected because it has to change its signal twice in order to store and ot read to and from the memory. This means that we were able to successful design the RAMPORT 1 component that can write 4 bit addresses to 16 different addresses.

PART II WRITING TO INTEGRATED STATIC RAM

11 WRITING TO MEMORY ON THE BOARD

11.1 Functionality and specifications for Writing to memory on the board.

The purpose of this design is to write to the memory that is integrated onto the board, we will not be running a simulation for this component because it will be of no use. We will instead provide a demonstration of when we write to the board directly. The board has a chip that is for SRAM and it is integrated directly. It has a 20 bit address on the DE2 115 and the data that is stored in each of the addresses is 16 bits. We will be specifying 8 bits for the address and 8 for the data as

well. That means that we will set the 12 most significant bits of the address to be equal to 0, and the 8 most significant bits for the data also to be 0.

```
library ieee;
use ieee.std_logic_1164.all;
entity GUTI_Sram is
    port(
                INPUT_ADDR: in std_logic_vector(7 downto 0);
                INPUT_DATA: in std_logiC_vector(7 downto 0);
                INPUT_WE: in std_logic;
        LEDR: out std_logic_vector(15 downto 0);---output
        SRAM_ADDR: out std_logic_vector(19 downto 0);
        SRAM_DQ: inout std_logic_vector(15 downto 0);
        SRAM_CE_N: out std_logic;
        SRAM_OE_N: out std_logic;
        SRAM_WE_N: out std_logic;
        SRAM_UB_N: out std_logic;
        SRAM_LB_N: out std_logic;
        HEX11: out STD_LOGIC_VECTOR(6 downto 0);
        HEX22: out std_LOGIC_VECTOR(6 downto 0);
                HEX33: out std_LOGIC_VECTOR(6 downto 0);
                HEX44: out std_LOGIC_VECTOR(6 downto 0));
end GUTI_Sram;
architecture arch of GUTI_Sram is
    signal address: std_logic_vector(7 downto 0);
    signal data: std_logic_vector(7 downto 0);
    signal output: std_logic_vector(15 downto 0);
    signal Wenable: std_logic;
    signal hex: std_logic_vector(15 downto 0);
component dec_to_hex is port (

               hex_digit: in std_logic_vector(3 downto 0);
               segment_a: out std_logic;
               segment_b: out std_logic;
               segment_c: out std_logic;
               segment_d: out std_logic;
               segment_e: out std_logic;
               segment_f: out std_logic;
               segment_g: out std_logic);
end component;
begin
         Wenable<=INPUT_WE;
         address <=INPUT_ADDR;
         data <= INPUT_DATA when Wenable = '1' else (others => 'Z');
    LEDR(15 downto 0)<= output;-- use 7 segment Display
    hex <= output;
    SRAM_WE_N <= not Wenable;
    SRAM_CE_N <= '0';
    SRAM_OE_N <= '0';
    SRAM_UB_N <= '0';
    SRAM_LB_N <= '0';
    SRAM_ADDR(19 downto 8) <= (others => '0');---set unused ports to 0
    SRAM_ADDR(7 downto 0) <= address;--assign address to first 8 ports
    SRAM_DQ(15 downto 8) <= (others => '0');
    SRAM_DQ(7 downto 0) <= data; --assign data to register

    output <= SRAM_DQ(15 downto 0);

    Hx0: dec_to_hex  port map(hex(3 downto 0), HEX11(0), HEX11(1), HEX11(2), HEX11(3),HEX11(4), HEX11(5), HEX11(6));
    Hx1: dec_to_hex port map (hex(7 downto 4), HEX22(0), HEX22(1), HEX22(2), HEX22(3),HEX22(4), HEX22(5), HEX22(6));
         Hx2: dec_to_hex port map (hex(11 downto 8), HEX33(0), HEX33(1), HEX33(2), HEX33(3),HEX33(4), HEX33(5), HEX33(6));


end arch;
```

Figure 18: VHDL code for Writing to the integrated memory on the board.

In this code we are using a 7-segment display to output the data that we are using and stored and changing the addresses for. We will be directly sending the data to chip on the board using the

switches and specifying 8 of the addresses. This works and we will give an example on the board in order to demonstrate the success of running the design.

11.2 Demonstration of Writing to the Board on the DE2-115 Board

The inputs and outputs that are assigned to the DE2-115 board are:

LEDR[0] is assigned to PIN_G19

LEDR[1] is assigned to PIN_F19

LEDR[2] is assigned to PIN_E19

LEDR[3] is assigned to PIN_F21

LEDR[4] is assigned to PIN_F18

LEDR[5] is assigned to PIN_E18

LEDR[6] is assigned to PIN_J19

LEDR[7] is assigned to PIN_H19

LEDR[8] is assigned to PIN_J17

LEDR[9] is assigned to PIN_G17

LEDR[10] is assigned to PIN_J15

LEDR[11] is assigned to PIN_H16

LEDR[12] is assigned to PIN_J16

LEDR[13] is assigned to PIN_H17

LEDR[14] is assigned to PIN_F15

LEDR[15] is assigned to PIN_G15

INPUT_DATA[0] is assigned to PIN_AB28

INPUT_DATA[1] is assigned to PIN_AC28

INPUT_DATA[2] is assigned to PIN_AC27

INPUT_DATA[3] is assigned to PIN_AD27

INPUT_DATA[4] is assigned to PIN_AB27

INPUT_DATA[5] is assigned to PIN_AC26

INPUT_DATA[6] is assigned to PIN_AD26

INPUT_DATA[7] is assigned to PIN_AB26

INPUT_ADDR[0] is assigned to PIN_AC25

INPUT_ADDR[1] is assigned to PIN_AB25

INPUT_ADDR[2] is assigned to PIN_AC24

INPUT_ADDR[3] is assigned to PIN_AB24

INPUT_ADDR[4] is assigned to PIN_AB23

INPUT_ADDR[5] is assigned to PIN_AA24

INPUT_ADDR[6] is assigned to PIN_AA23

INPUT_ADDR[7] is assigned to PIN_AA22

INPUT_WE is assigned to PIN_Y23

SRAM_ADDR[0] is assigned to PIN_AB7

SRAM_ADDR[1] is assigned to PIN_AD7

SRAM_ADDR[2] is assigned to PIN_AE7

SRAM_ADDR[3] is assigned to PIN_AC7

SRAM_ADDR[4] is assigned to PIN_AB6

SRAM_ADDR[5] is assigned to PIN_AE6

SRAM_ADDR[6] is assigned to PIN_AB5

SRAM_ADDR[7] is assigned to PIN_AC5

SRAM_ADDR[8] is assigned to PIN_AF5

SRAM_ADDR[9] is assigned to PIN_T7

SRAM_ADDR[10] is assigned to PIN_AF2

SRAM_ADDR[11] is assigned to PIN_AD3

SRAM_ADDR[12] is assigned to PIN_AB4

SRAM_ADDR[13] is assigned to PIN_AC3

SRAM_ADDR[14] is assigned to PIN_AA4

SRAM_ADDR[15] is assigned to PIN_AB11

SRAM_ADDR[16] is assigned to PIN_AC11

SRAM_ADDR[17] is assigned to PIN_AB9

SRAM_ADDR[18] is assigned to PIN_AB8

SRAM_ADDR[19] is assigned to PIN_T8

SRAM_CE_N is assigned to PIN_AF8

SRAM_OE_N is assigned to PIN_AD5

SRAM_WE_N is assigned to PIN_AE8

SRAM_UB_N is assigned to PIN_AC4

SRAM_LB_N is assigned to PIN_AD4

SRAM_DQ[0] is assigned to PIN_AH3

SRAM_DQ[1] is assigned to PIN_AF4

SRAM_DQ[2] is assigned to PIN_AG4

SRAM_DQ[3] is assigned to PIN_AH4

SRAM_DQ[4] is assigned to PIN_AF6

SRAM_DQ[5] is assigned to PIN_AG6

SRAM_DQ[6] is assigned to PIN_AH6

SRAM_DQ[7] is assigned to PIN_AF7

SRAM_DQ[8] is assigned to PIN_AD1

SRAM_DQ[9] is assigned to PIN_AD2

SRAM_DQ[10] is assigned to PIN_AE2

SRAM_DQ[11] is assigned to PIN_AE1

SRAM_DQ[12] is assigned to PIN_AE3

SRAM_DQ[13] is assigned to PIN_AE4

SRAM_DQ[14] is assigned to PIN_AF3

SRAM_DQ[15] is assigned to PIN_AG3

HEX11[0] is assigned to PIN_G18

HEX11[1] is assigned to PIN_F22

HEX11[2] is assigned to PIN_E17

HEX11[3] is assigned to PIN_L26

HEX11[4] is assigned to PIN_L25

HEX11[5] is assigned to PIN_J22

HEX11[6] is assigned to PIN_H22

HEX22[0] is assigned to PIN_M24

HEX22[1] is assigned to PIN_Y22

HEX22[2] is assigned to PIN_W21

HEX22[3] is assigned to PIN_W22

HEX22[4] is assigned to PIN_W25

HEX22[5] is assigned to PIN_U23

HEX22[6] is assigned to PIN_U24

HEX33[0] is assigned to PIN_AA25

HEX33[1] is assigned to PIN_AA26

HEX33[2] is assigned to PIN_Y25

HEX33[3] is assigned to PIN_W26

HEX33[4] is assigned to PIN_Y26

HEX33[5] is assigned to PIN_W27

HEX33[6] is assigned to PIN_W28

HEX44[0] is assigned to PIN_V21

HEX44[1] is assigned to PIN_U21

HEX44[2] is assigned to PIN_AB20

HEX44[3] is assigned to PIN_AA21

HEX44[4] is assigned to PIN_AD24

HEX44[5] is assigned to PIN_AF23

HEX44[6] is assigned to PIN_Y19

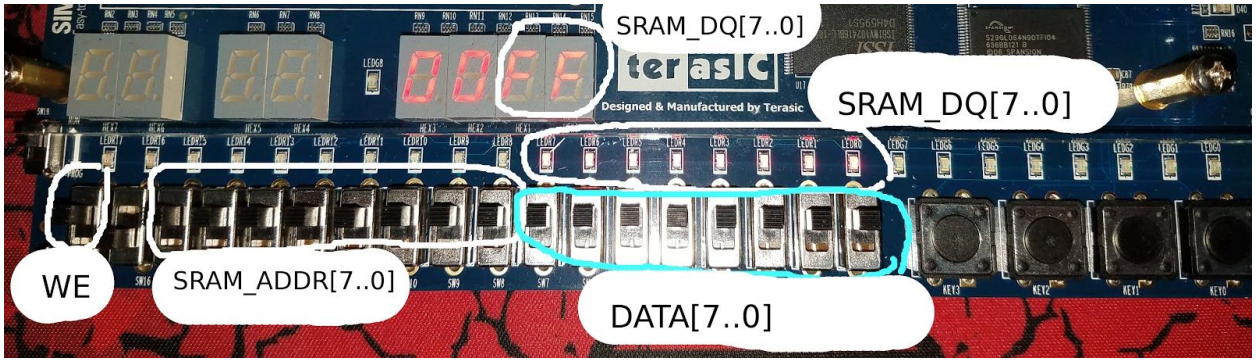


Figure 19: Digital circuit of Writing to the SRAM memory on the board. In this image we are saving the value FF to the Address FF and WE is on so we are able to read and store to the address and the data on the board. This means we were able to successfully design a circuit that would write and read from the address of the board on the DE2-115. We are displaying the value on the 7 segment display in hexadecimal format and we are also displaying them on the led lights.

12 Conclusion

As it turns out we were able to correctly design memory, something that can remember what we give as the inputs after we leave the access to that location. What made this lab difficult to understand is that we would need to use an extra key if we want to use more than 4 bits of information in each of the addresses that we are writing to, the reason for that is that we have too few switches and that means that based on the set up that we have currently we can not use more than 4 bits of information. If we have a key, the addresses that are used to determine which of the locations we are going to be writing to out of the 16 locations can be change to represent which of the 4 locations we are writing to in the 4 sections of 1 16 bit word once we select it. That is probably something that we could potentially do in the next lab as we are currently approaching designing CPU as is my understanding. It was also useful to write to the memory on the board

because it saves time in understanding its functionality. It was useful and beneficial and I was introduced to using inout signals too.

13 Appendix

```
To, Location
GUTI_datain[0], PIN_AB12
GUTI_datain[1], PIN_AC12
GUTI_datain[2], PIN_AF9
GUTI_datain[3], PIN_AF10

GUTI_address[0], PIN_AD11
GUTI_address[1], PIN_AD12
GUTI_address[2], PIN_AE11
GUTI_address[3], PIN_AC9

GUTI_CS, PIN_AD10
GUTI_OE, PIN_AE12
GUTI_WE, PIN_Y16

GUTI_segment_a, PIN_AE26
GUTI_segment_b, PIN_AE27
GUTI_segment_c, PIN_AE28
GUTI_segment_d, PIN_AG27
GUTI_segment_e, PIN_AF28
GUTI_segment_f, PIN_AG28
GUTI_segment_g, PIN_AH28
```

Figure 20: Pin Assignment for full project that is used after all of the components were made and then we connected to the FPGA board and were able to choose which switches were going to be used for inputs, and which hexadecimal display was going to be used to display the output.