CS343 LAB 2

Static Random-Access Memory (SRAM)

Prof. Izidor Gertner

Faheem Kamal

**Table of Contents**

**Objective 2**

**Functionality and Specifications 2 – 10**

**Simulation 10 - 16**

**Conclusion 16 - 20**

**Appendix 20 - 27**

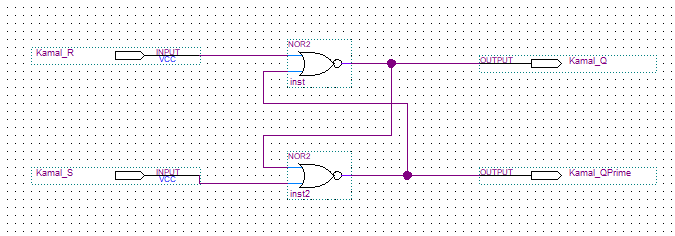
**Objective**

The goal of this lab is to understand and create static random-access memory chip using D-Latches. With this SRAM, you can store multiple bits in various address locations. Our goal is to gain knowledge and experience to create a memory device that can hold 16-bit values and to display these numbers in 7-segment HEX displays on our DE2 Board. The steps to design my lab are as follows:

1. Create SR-Latch
2. Create Control SR-Latch
3. Create D-Latch
4. Create SRAM
5. Create 16x1 SRAM
6. Add 4-to-16 Decoder to Project
7. Create 16x4 SRAM with 4-to-16 Decoder
8. Add Decimal to Hexadecimal Converter to Project
9. Add Decimal to Hexadecimal Converter to 16x4 SRAM
10. Create 16x8 SRAM
11. Create Buttons and Add to Project
12. Create 32x16 SRAM with Buttons

**Functionality and Specifications**

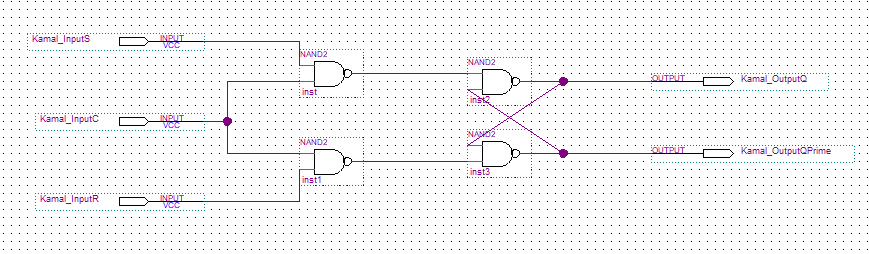
1. **SR Latch**



**Inputs:** Kamal\_R and Kamal\_S

**Outputs:** Kamal\_Q and Kamal\_QPrime

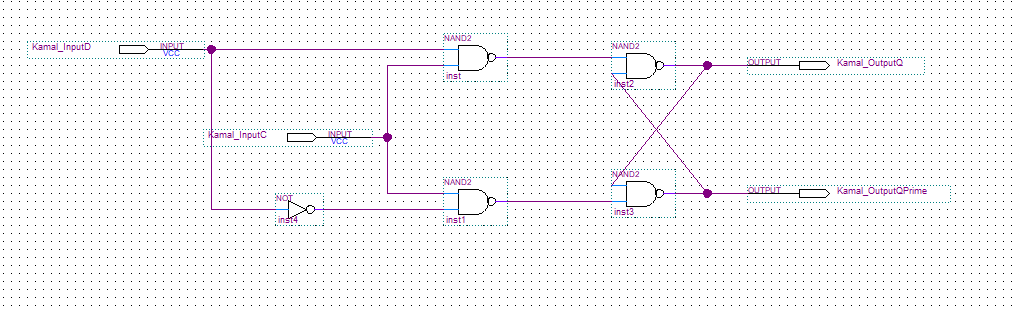
1. **Control SR Latch**



**Inputs:** Kamal\_InputS, Kamal\_InputC and Kamal\_InputR

**Outputs:** Kamal\_OutputQ and Kamal\_OutputQPrime

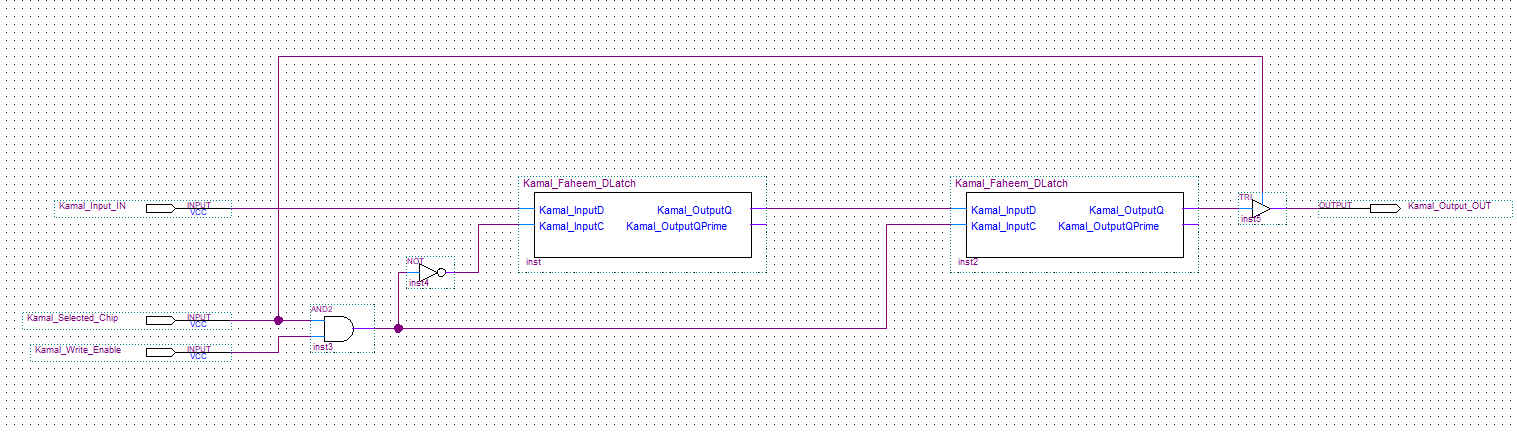
1. **D-Latch**



**Inputs:** Kamal\_InputD and Kamal\_InputC

**Outputs:** Kamal\_OutputQ and Kamal\_OutputQPrime

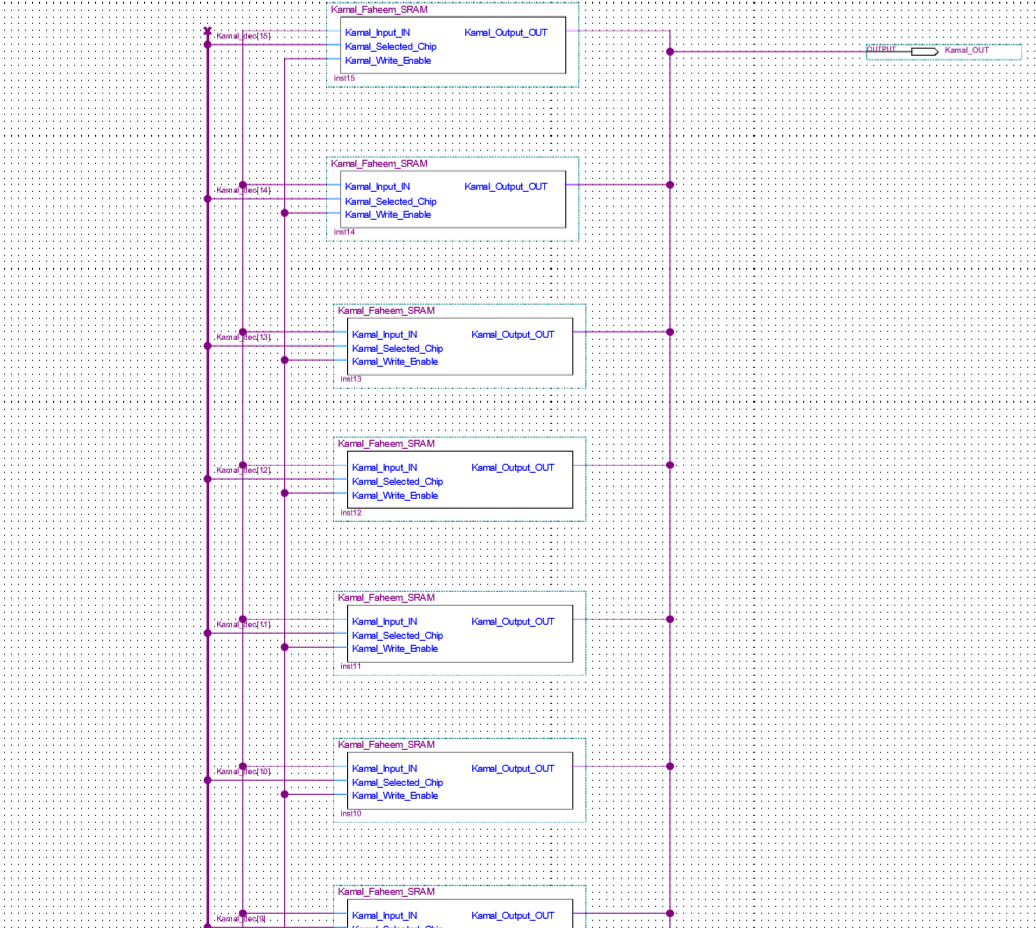
1. **SRAM**

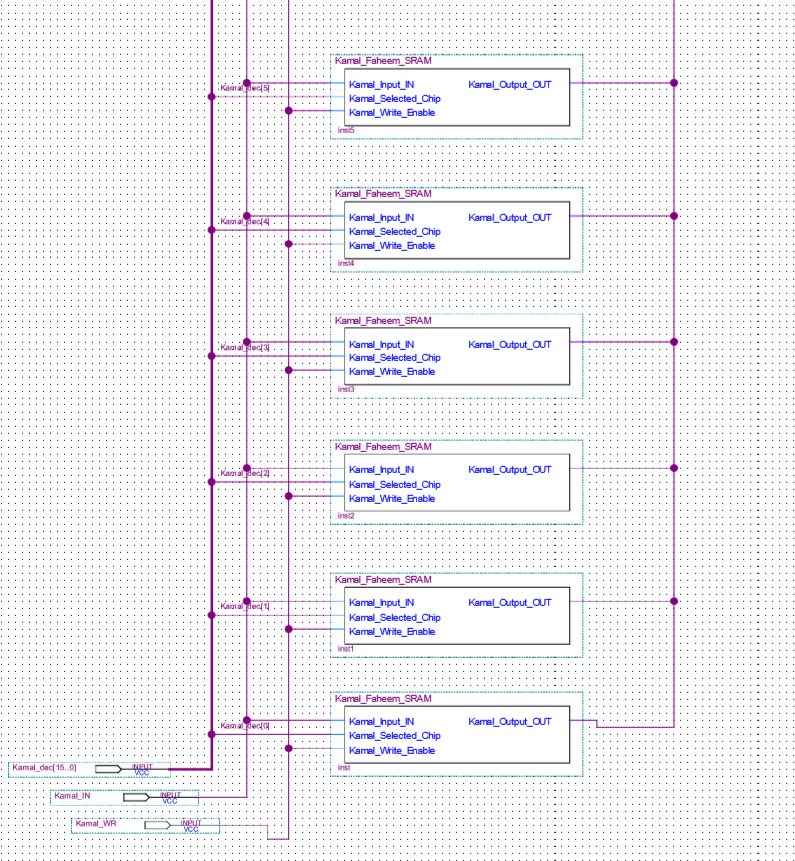


**Inputs:** Kamal\_Input\_IN, Kamal\_Selected\_Chip, Kamal\_Write\_Enable

**Outputs:** Kamal\_Output\_OUT

1. **16x1 SRAM**

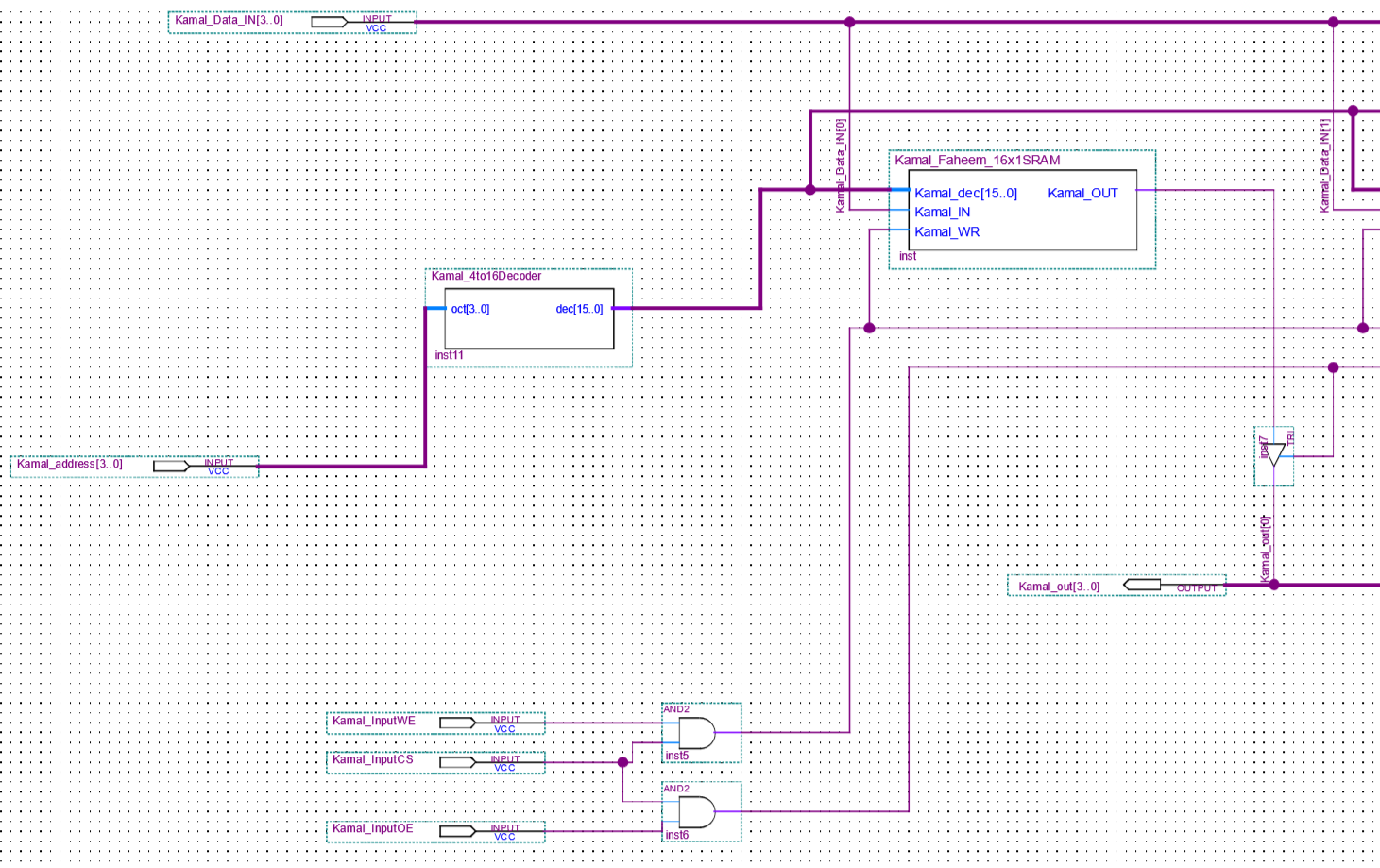


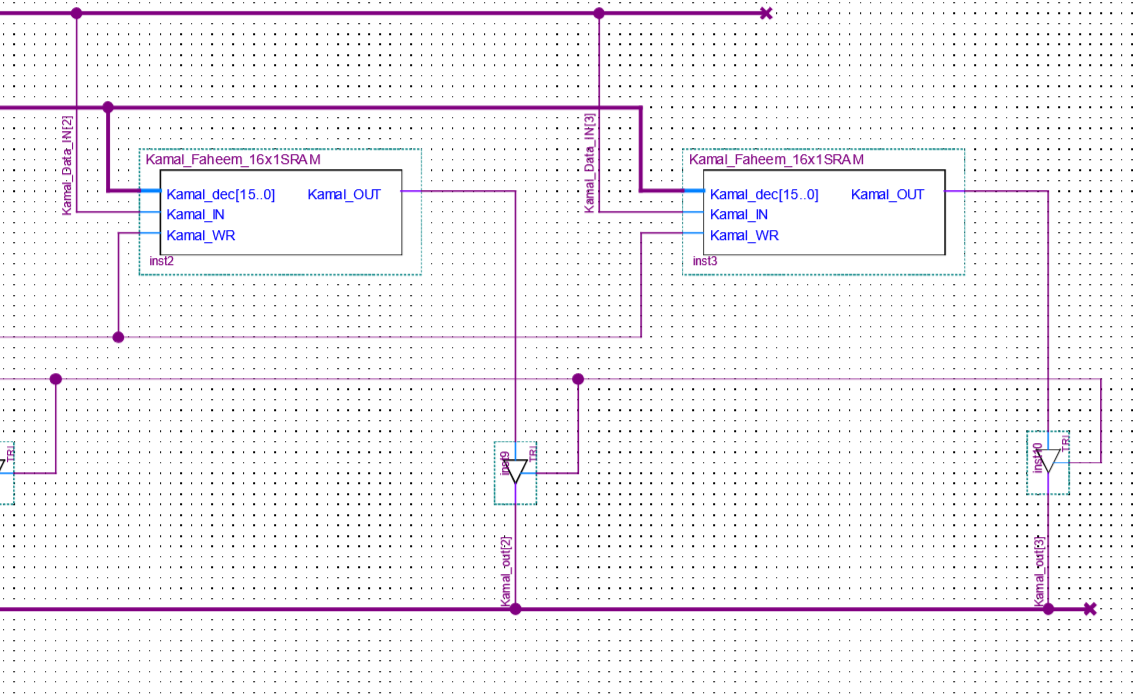


**Inputs:** Kamal\_dec[15..0], Kamal\_IN and Kamal\_WR

**Outputs:** Kamal\_OUT

1. **16x4 SRAM**

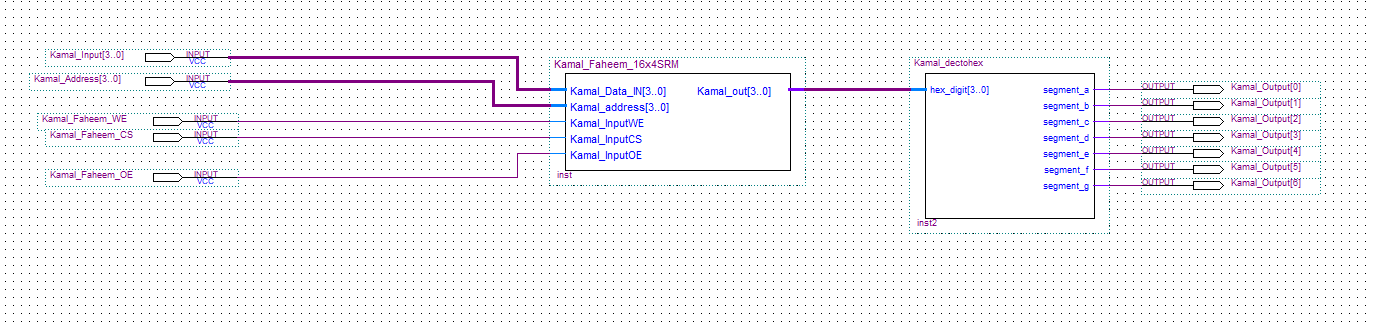




**Inputs:** Kamal\_InputWE, Kamal\_InputCS, Kamal\_InputOE, Kamal\_Data\_IN[3..0] and Kamal\_address[3..0]

**Outputs:** Kamal\_out[3..0]

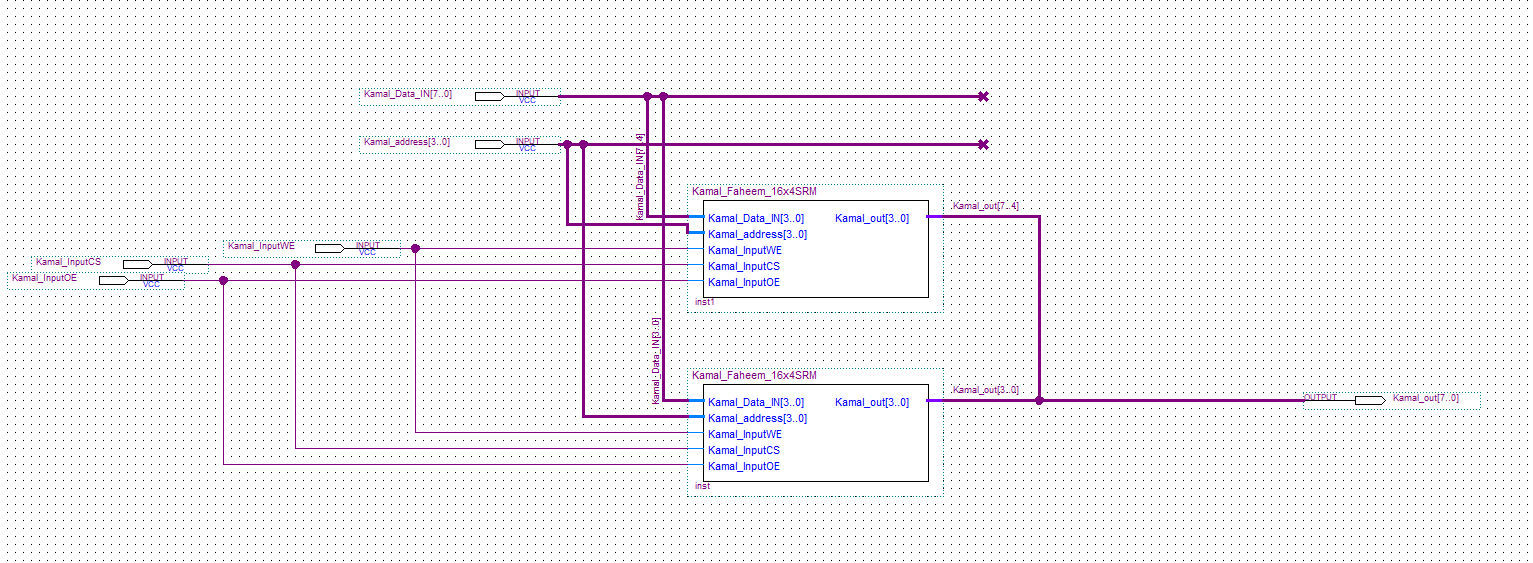
1. **16x4 SRAM with Dec-to-Hex**



**Inputs:** Kamal\_Data\_IN[3..0], Kamal\_InputWE, Kamal\_InputCS, Kamal\_InputOE and Kamal\_address[3..0].

**Outputs:** Kamal\_Output[0], Kamal\_Output[1], Kamal\_Output[2], Kamal\_Output[3], Kamal\_Output[4], Kamal\_Output[5] and Kamal\_Output[6].

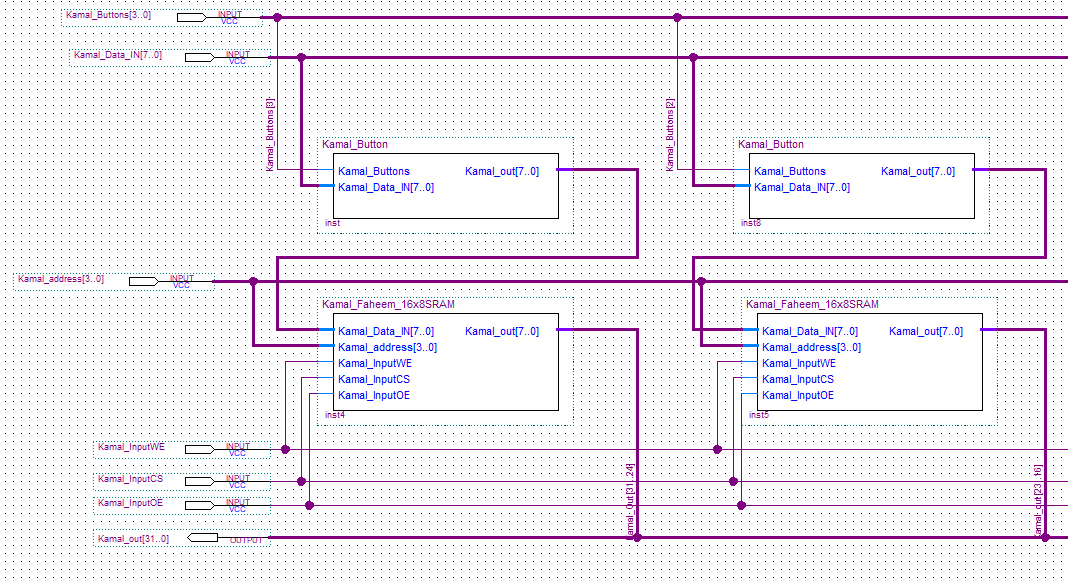
1. **16x8 SRAM**



**Inputs:** Kamal\_Data\_IN[7..0], Kamal\_address[3..0], Kamal\_InputWE, Kamal\_InputCS and Kamal\_InputOE.

**Outputs:** Kamal\_out[7..0]

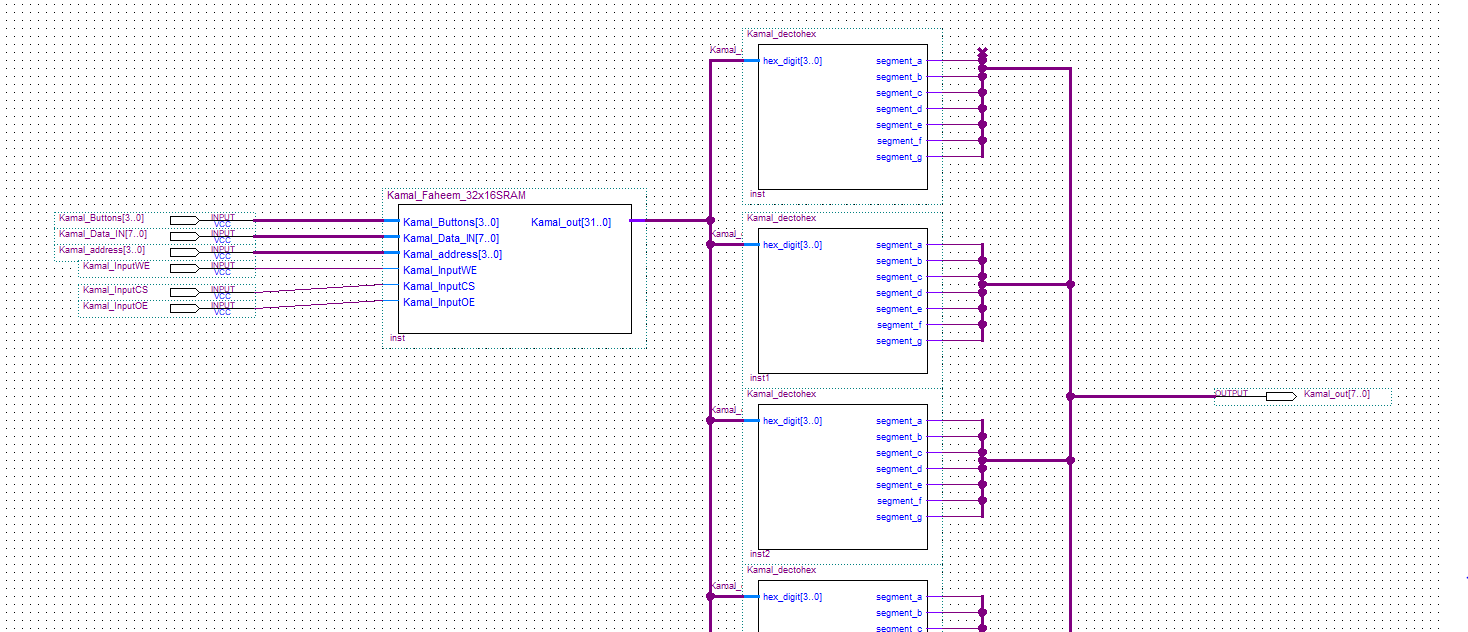
1. **16x32 SRAM**



**Inputs:** Kamal\_InputWE, Kamal\_InputCS, Kamal\_InputOE, Kamal\_address[3..0], Kamal\_Data\_IN[7..0] and Kamal\_Buttons[3..0].

**Outputs:** Kamal\_out[31..0]

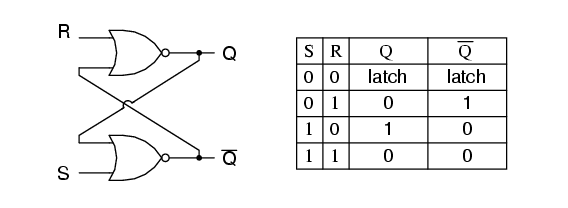
1. **16x32 SRAM with Dec-to-Hex Converter**

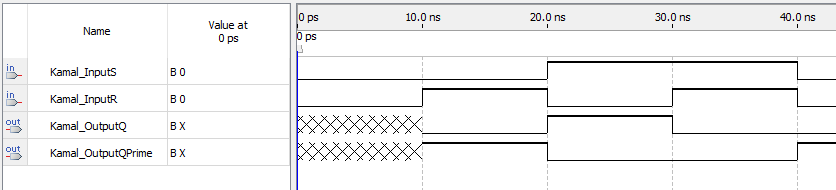
**Inputs:** Kamal\_Buttons[3..0], Kamal\_Data\_IN[7..0], Kamal\_address[3..0], Kamal\_InputWE, Kamal\_InputCS and Kamal\_InputOE.

**Outputs:** Kamal\_out[7..0]

**Simulation**

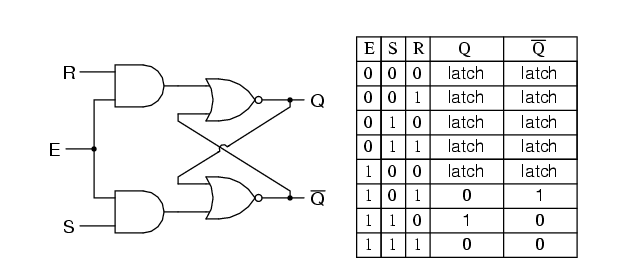
1. **SR-Latch**

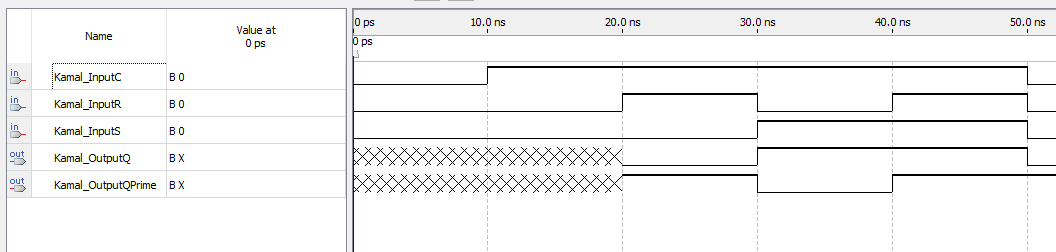




An SR-Latch is a bistable multivibrator that has two stable states, indicated by the prefix bi (meaning two) in its name. One state is referred to as set and the latter is referred as reset. The simplest bistable device is known as a set-reset latch. To create an SR-Latch, we can wire two NOR gates in a way where one output feeds back to the input of the other, and vice-versa. Kamal\_OutputQ and Kamal\_OutputQPrime are supposed to be in opposite states and Kamal\_InputS (set) and Kamal\_InputR (reset) are the inputs for the Quartus Design.

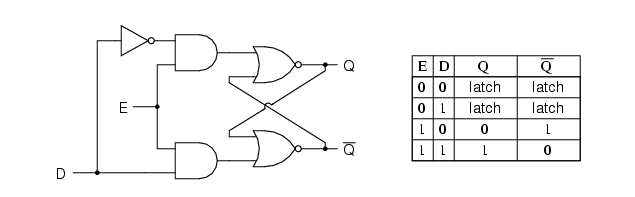
1. **Control SR-Latch**

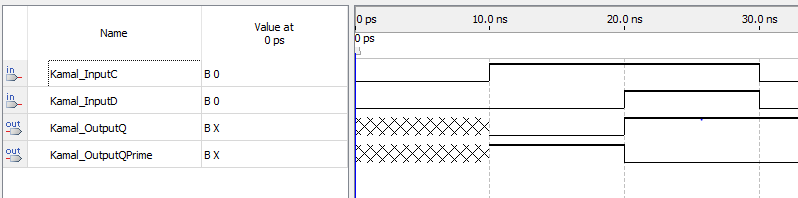




A Control SR-Latch is a multivibrator that changes only when certain conditions are met regardless of Kamal\_InputR and Kamal\_InputS states. The conditional input is called Kamal\_InputC, also known as the enable. When Kamal\_InputC is 0, the outputs of the two AND gates are forced to 0, regardless of the states of either Kamal\_InputS or Kamal\_InputR. Consequently, the circuit behaves as though Kamal\_InputS and Kamal\_InputR are both 0, latching the Kamal\_OutputQ and Kamal\_OutputQPrime in their last states. When the enable Kamal\_InputC is 1, the latch will respond to the Kamal\_InputR and Kamal\_InputS.

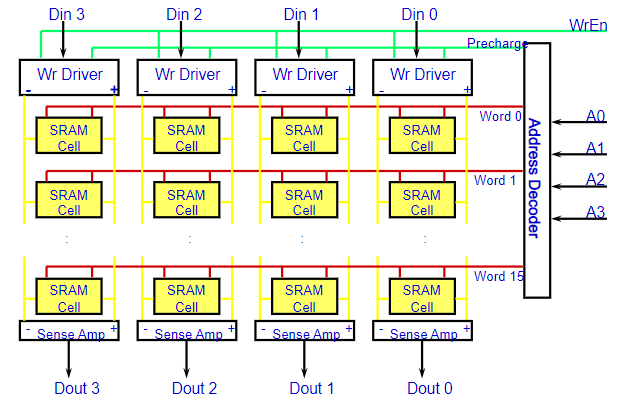
1. **D-Latch**

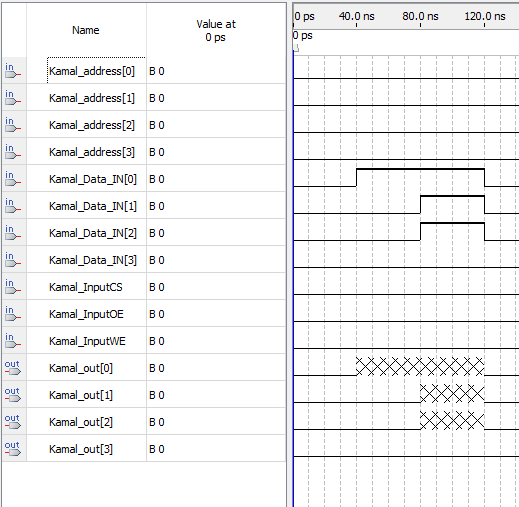




Latch is an electronic device used to store one-bit information. The D-Latch captures or ‘latches’ the logic level which is present on the Data Line when the Clock input is high. If the data on the D-Line changes state when the clock pulse is high, then the output Kamal\_OutputQ follows the input Kamal\_InputD. When the clock input is 0, the Kamal\_InputD is trapped and held in the latch.

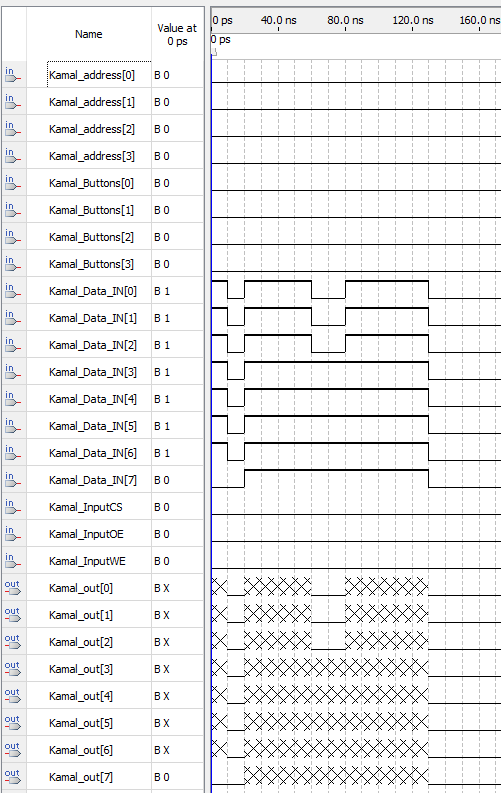
1. **16x4 SRAM**





SRAM or Static Random Access Memory is a type of volatile semiconductor memory to store binary logic '1' and '0' bits. SRAM uses bi-stable latching circuitry made of Transistors/MOSFETS to store each bit. Compared to Dynamic RAM (DRAM), SRAM doesn't have a capacitor to store the data, hence SRAM works without refreshing. In SRAM the data is lost when the memory is not electrically powered. It consists of a bi-stable flip-flop connected to the internal circuitry by two access transistors. When the cell is not addressed, the two access transistors are closed and the data is kept to a stable state, latched within the flip-flop. The flip-flop needs the power supply to keep the information. The data in an SRAM cell is volatile. In this case, in Quartus, I designed 16x1 SRAM and converted it into a symbol. I used that specific symbol to place 4 16x1 SRAMs along with Chip Select, Write Enable, Output along with the 4-to-16 decoder along with data input to make a 16x4 SRAM symbol. With this symbol and the dectohex converter, I programmed my FPGA board to test the logic for a 16x4 SRAM.

1. **16x32 SRAM**



For the 16x32 SRAM, I reused the symbol I made of the 16x4 SRAM and made a 16x8 SRAM. With this, I converted 16x8 SRAM into a symbol and added 4 blocks to create this 16x32 SRAM. I also added a button that I made to help me program my board to use keys. For the simulation, we needed to provide 32 bit output using 8 switches and 4 keys which I was able to achieve with this design.

**Conclusion**

**What is the difference between a latch and a flip-flop?**

Latches are building blocks of sequential circuits built from logic gates. Flip-flops are also building blocks of sequential circuits but can be built from the latches. Latches continuously checks inputs and changes its outputs correspondingly. Flip-flops continuously checks inputs, but it changes output correspondingly only at times determined by clocking signal. The latch is sensitive to the duration of the pulse and can send or receive data when the switch is on. Flip-flops are sensitive to a change of signal. They transfer data only at the single instant and data cannot be changed until the next signal change. Flip-flops are used as a register.

**Explain the behavior of a SR Latch.**

The SR Latch is a bistable multivibrator that has two stable states, set and reset. The SSR latch are wired two NOR gates where one output feeds back to the input of another, vice versa. The Q and QPrime are opposite states. I supposed to make both the S and R input equal to 1 results in both Q and QPrime not equal to 0. Having set and reset equal to 1 is called an invalid or illegal state for the SR latch. Otherwise, making S=1 and R=0 sets the multivibrator so that Q=1 an QPrime=0. Conversely, make R=1 and S=0 resets the multivibrator in the opposite state. When S and R are both equal to 0, the multivibrator outputs “latch” in prior states.

**Explain how the gated(control) SR-Latch is a modified SR-Latch.**

Gated SR-Latch only changes its output state when there is an enabling signal along with its required input. The inputs can only act upon a latch when its enabled otherwise no change in the output state when the inputs are applied. The latch is active when enable signal is high and it Is inactive when enable signal is low. These signals are applied in the form of clocked pulses. Gated SR-Latch is also called clocked SR Flip flop or synchronous SR latch.

**Explain how the D-Latch is a modified gated SR-Latch.**

The D- Latch is considered a one-input synchronous SR-Latch. The NAND stage converts two D input stages to two input combinations for the next SR Latch by inverting the signal of the data inputs. This configuration prevents the application of the restricted input combination. D-Latch also goes by the name transparent latch, data latch, or simply gated latch. It has a data input and an enable signal. The word transparent comes from the fact that, when the enable input is on, the signal propagates directly through the circuit, from the input D to the output Q. Gated D-latches are level-sensitive based on the level of the clock or enable signal.

**Explain how to build a Master-Slave flip flop.**

A Master-Slave flip flop is designed to utilize two separate flip flops. One serves as the Master flip-flop and the latter serves as the Slave. The circuit also contains an inverter. This inverter is connected to a clock pulse in a way that the inverted clock pulse is given to the slave flip-flop. If the clock pulse is 0 for the Master flip-flop, the inverter’s output is 1 and this specific value is assigned to the slave flip-flop. If the clock pulse is 1 for the Master flip-flop, the inverters output is 0 and this specific value is assigned to the slave flip-flop.

An example of a Master-Slave flip flop is a Master-Slave D flip flop is built by connected two gated D-latches in series and by inverting the enable input that leads to one of the gated D-latches. The name is derived from the fact that the second latch in the series only changes in response to the first latch. Hence, the reason why the first latch is considered the Master and the second latch is considered the Slave. For a positive-edge triggered flip flop,

**Explain how to build SRAM cells from flip flops.**

Within a static RAM, a form of flip-flop holds each bit of memory. This flip-flop for a memory cell takes either 4 to 6 transistors along with some wiring, but never has to be refreshed. This makes static RAM seem faster than dynamic RAM. However, static memory cells take up a lot of space on a chip compared to a dynamic memory cell. Hence, you get less memory per chip and that makes static RAM a lot more expensive.

**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?**

For an SR-Latch, we know that the output of NOR gate is 1 if and only if both inputs from S and R are 0; and 0 otherwise. When S=1, Q = 1 and therefore QPrime = 0 and when R=1, Q =0 and QPrime = 1. But when you set both of R and S to 1, we have that Q and QPrime at both 0 at the same time. This violates the conditions in an SR-Latch. Q and QPrime should never be 0 at the same time and both outputs should never be 1 at the same time. The logic is that QPrime will always be the opposite of Q. The other reason to avoid S=R=1 is that it leads to unpredictable results. Set and Reset should never be enabled at the same time, it is incorrect. For a D-latch, we don’t have a similar consequence since Q will be outputted as 1 and QPrime will be 0 since the D-latch only responds when the enable input is 1 and that’s when the Q output will follow the input of D. If the enable input is 0, then it will stayed latched in its last state.

**What is the difference between edge-triggered and level triggered devices? Is this flip-flop used for the SRAM cell in this lab level or edge triggered?**

An edge-triggered interrupt is an interrupt signaled by a level transition on the interrupt line, either as a falling edge or a rising edge. In edge-triggering, the circuit becomes active at negative or positive edge of the clock signal. So, when its positive edge-triggered, it takes an 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 for negative edge-triggered. A level triggered interrupt is when the circuit becomes active when the gate or clock pulse is on a level. When the circuit is active, the clock signal is low, and we have negative level triggering. Its positive when the clock signal is high, and the circuit is active. Level triggered is sensitive to glitches and Edge triggered is not sensitive to glitches.

**Can you explain how an SRAM works?**

SRAM is a static random-access memory that retains bits of data in memory as long as power is supplied. Unlike dynamic RAM(DRAM) stores bits in cells consisting of a capacitor and a transistor, SRAM is not refreshed. Static RAM provides faster access to data. SRAM takes 6 transistors to store a bit. It has 3 inputs and 2 outputs. SRAM has 3 options: hold, read and write. The hold operation accesses transistors and must be turn off due to the presence of latching element SRAM hold its state. The read operation both bit lines must be recharged to VDD and access transistors must be turn on. Based on data value, it stores any one of the bit line is discharged and the voltage difference between two-bit lines is sensed by an amplifier and are detected by what is present in memory. The write operation is we write to a given bit-line and access transistors are turned on. SRAM is used for a computer’s cache memory and is part of the random access memory digital-to-analog converter on a video card.

**Appendix**

1. **4-to-16 Decoder VHDL Code**

library ieee;

use ieee.std\_logic\_1164.all;

entity Kamal\_4to16Decoder is

port(

oct : in std\_logic\_vector(3 downto 0);

dec : out std\_logic\_vector(15 downto 0));

end Kamal\_4to16Decoder;

architecture arch of Kamal\_4to16Decoder 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;

1. **Dec-to-Hex 7 Segment Display VHDL**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity Kamal\_dectohex is

port( hex\_digit: IN std\_logic\_vector(3 downto 0);

segment\_a, segment\_b, segment\_c, segment\_d, segment\_e, segment\_f, segment\_g : out std\_logic);

end Kamal\_dectohex;

architecture a of Kamal\_dectohex is

signal segment\_data : std\_logic\_vector(6 downto 0);

begin

process (hex\_digit)

begin

case hex\_digit is

when "0000" =>

segment\_data <= "1111110";

when "0001" =>

segment\_data <= "0110000";

when "0010" =>

segment\_data <= "1101101";

when "0011" =>

segment\_data <= "1111001";

when "0100" =>

segment\_data <= "0110011";

when "0101" =>

segment\_data <= "1011011";

when "0110" =>

segment\_data <= "1011111";

when "0111" =>

segment\_data <= "1110000";

when "1000" =>

segment\_data <= "1111111";

when "1001" =>

segment\_data <= "1110011";

when "1010" =>

segment\_data <= "1110111";

when "1011" =>

segment\_data <= "0011111";

when "1100" =>

segment\_data <= "1001110";

when "1101" =>

segment\_data <= "0111101";

when "1110" =>

segment\_data <= "1001111";

when "1111" =>

segment\_data <= "1000111";

end case;

end process;

--extract segment data bits and invert

--led driver circuit is inverted

segment\_a <= not segment\_data(6);

segment\_b <= not segment\_data(5);

segment\_c <= not segment\_data(4);

segment\_d <= not segment\_data(3);

segment\_e <= not segment\_data(2);

segment\_f <= not segment\_data(1);

segment\_g <= not segment\_data(0);

end a;

1. **Pin Assignment for 16x4 SRAM with Decoder and Dec-to-Hex 7 Segment Display VHDL**

To, Location

Kamal\_Input[3], PIN\_AD27

Kamal\_Input[2], PIN\_AC27

Kamal\_Input[1], PIN\_AC28

Kamal\_Input[0], PIN\_AB28

Kamal\_Address[2], PIN\_AB26

Kamal\_Address[1], PIN\_AD26

Kamal\_Address[0], PIN\_AC26

Kamal\_Faheem\_WE, PIN\_AA22

Kamal\_Faheem\_CS, PIN\_Y23

Kamal\_Faheem\_OE, PIN\_Y22

Kamal\_Output[6], PIN\_H22

Kamal\_Output[5], PIN\_J22

Kamal\_Output[4], PIN\_L25

Kamal\_Output[3], PIN\_L26

Kamal\_Output[2], PIN\_E17

Kamal\_Output[1], PIN\_F22

Kamal\_Output[0], PIN\_G18

1. **Pin Assignment for 16x32 SRAM**

To, Location

Kamal\_Buttons[3], PIN\_R24

Kamal\_Buttons[2], PIN\_N21

Kamal\_Buttons[1], PIN\_M21

Kamal\_Buttons[0], PIN\_M23

Kamal\_Data\_IN[7], PIN\_AB26

Kamal\_Data\_IN[6], PIN\_AD26

Kamal\_Data\_IN[5], PIN\_AC26

Kamal\_Data\_IN[4], PIN\_AB27

Kamal\_Data\_IN[3], PIN\_AD27

Kamal\_Data\_IN[2], PIN\_AC27

Kamal\_Data\_IN[1], PIN\_AC28

Kamal\_Data\_IN[0], PIN\_AB28

Kamal\_address[3], PIN\_AB24

Kamal\_address[2], PIN\_AC24

Kamal\_address[1], PIN\_AB25

Kamal\_address[0], PIN\_AC25

Kamal\_InputCS, PIN\_Y23

Kamal\_InputOE, PIN\_Y24

Kamal\_InputWE, PIN\_AA22

Kamal\_Segment0[6], PIN\_G18

Kamal\_Segment0[5], PIN\_F22

Kamal\_Segment0[4], PIN\_E17

Kamal\_Segment0[3], PIN\_L26

Kamal\_Segment0[2], PIN\_L25

Kamal\_Segment0[1], PIN\_J22

Kamal\_Segment0[0], PIN\_H22

Kamal\_Segment1[6], PIN\_M24

Kamal\_Segment1[5], PIN\_Y22

Kamal\_Segment1[4], PIN\_W21

Kamal\_Segment1[3], PIN\_W22

Kamal\_Segment1[2], PIN\_W25

Kamal\_Segment1[1], PIN\_U23

Kamal\_Segment1[0], PIN\_U24

Kamal\_Segment2[6], PIN\_AA25

Kamal\_Segment2[5], PIN\_AA26

Kamal\_Segment2[4], PIN\_Y25

Kamal\_Segment2[3], PIN\_W26

Kamal\_Segment2[2], PIN\_Y26

Kamal\_Segment2[1], PIN\_W27

Kamal\_Segment2[0], PIN\_W28

Kamal\_Segment3[6], PIN\_V21

Kamal\_Segment3[5], PIN\_U21

Kamal\_Segment3[4], PIN\_AB20

Kamal\_Segment3[3], PIN\_AA21

Kamal\_Segment3[2], PIN\_AD24

Kamal\_Segment3[1], PIN\_AF23

Kamal\_Segment3[0], PIN\_Y19

Kamal\_Segment4[6], PIN\_AB19

Kamal\_Segment4[5], PIN\_AA19

Kamal\_Segment4[4], PIN\_AG21

Kamal\_Segment4[3], PIN\_AH21

Kamal\_Segment4[2], PIN\_AE19

Kamal\_Segment4[1], PIN\_AF19

Kamal\_Segment4[0], PIN\_AE18

Kamal\_Segment5[6], PIN\_AD18

Kamal\_Segment5[5], PIN\_AC18

Kamal\_Segment5[4], PIN\_AB18

Kamal\_Segment5[3], PIN\_AH19

Kamal\_Segment5[2], PIN\_AG19

Kamal\_Segment5[1], PIN\_AF18

Kamal\_Segment5[0], PIN\_AH18

Kamal\_Segment6[6], PIN\_AA17

Kamal\_Segment6[5], PIN\_AB16

Kamal\_Segment6[4], PIN\_AA16

Kamal\_Segment6[3], PIN\_AB17

Kamal\_Segment6[2], PIN\_AB15

Kamal\_Segment6[1], PIN\_AA15

Kamal\_Segment6[0], PIN\_AC17

Kamal\_Segment7[6], PIN\_AD17

Kamal\_Segment7[5], PIN\_AE17

Kamal\_Segment7[4], PIN\_AG17

Kamal\_Segment7[3], PIN\_AH17

Kamal\_Segment7[2], PIN\_AF17

Kamal\_Segment7[1], PIN\_AG18

Kamal\_Segment7[0], PIN\_AA14

1. **Signs Dec-to-Hex Converter for 7 Segment Display**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity Kamal\_dectohex2 is

port( hex\_digit: IN std\_logic\_vector(3 downto 0);

segment\_a, segment\_b, segment\_c, segment\_d, segment\_e, segment\_f, segment\_g : out std\_logic);

end Kamal\_dectohex2;

architecture a of Kamal\_dectohex2 is

signal segment\_data : std\_logic\_vector(6 downto 0);

begin

process (hex\_digit)

begin

case hex\_digit is

when "0000" =>

segment\_data <= "0110001";

when "0001" =>

segment\_data <= "0110001";

when "0010" =>

segment\_data <= "0110001";

when "0011" =>

segment\_data <= "0110001";

when "0100" =>

segment\_data <= "0110001";

when "0101" =>

segment\_data <= "0110001";

when "0110" =>

segment\_data <= "0110001";

when "0111" =>

segment\_data <= "0110001";

when "1000" =>

segment\_data <= "0000001";

when "1001" =>

segment\_data <= "0000001";

when "1010" =>

segment\_data <= "0000001";

when "1011" =>

segment\_data <= "0000001";

when "1100" =>

segment\_data <= "0000001";

when "1101" =>

segment\_data <= "0000001";

when "1110" =>

segment\_data <= "0000001";

when "1111" =>

segment\_data <= "0000001";

end case;

end process;

--extract segment data bits and invert

--led driver circuit is inverted

segment\_a <= not segment\_data(6);

segment\_b <= not segment\_data(5);

segment\_c <= not segment\_data(4);

segment\_d <= not segment\_data(3);

segment\_e <= not segment\_data(2);

segment\_f <= not segment\_data(1);

segment\_g <= not segment\_data(0);

end a;