Lab 7 SRAM

CS211 Summer 2017

July 20, 2017

Xu Mingzhi

Contents

[1. Objective 4](#_Toc488521752)

[2. SR Latch 5](#_Toc488521753)

[2.1 Functionality and Specification 5](#_Toc488521754)

[2.2 Simulation 7](#_Toc488521755)

[3. Control SR Latch 9](#_Toc488521756)

[3.1 Functionality and Specification 9](#_Toc488521757)

[3.2 Simulation 11](#_Toc488521758)

[4. D Latch 12](#_Toc488521759)

[4.1 Functionality and Specification 12](#_Toc488521760)

[4.2 Simulation 13](#_Toc488521761)

[5. Positive Edge Triggered Master Slave D Flip Flop 15](#_Toc488521762)

[5.1 Functionality and Specification 15](#_Toc488521763)

[5.2 Simulation 16](#_Toc488521764)

[6. Negative Edge Triggered Master Slave D Flip Flop 18](#_Toc488521765)

[6.1 Functionality and Specification 18](#_Toc488521766)

[6.2 Simulation 19](#_Toc488521767)

[7. JK Flip Flop 21](#_Toc488521768)

[7.1 Functionality and Specification 21](#_Toc488521769)

[7.2 Simulation 23](#_Toc488521770)

[8. T Flip Flop 24](#_Toc488521771)

[8.1 Functionality and Specification 24](#_Toc488521772)

[8.2 Simulation 26](#_Toc488521773)

[9. SRAM Cell 27](#_Toc488521774)

[9.1 Functionality and Specification 27](#_Toc488521775)

[9.2 Simulation 27](#_Toc488521776)

[10. 16x4 SRAM 28](#_Toc488521777)

[10.1 Functionality and Specification 28](#_Toc488521778)

[10.2 Simulation 32](#_Toc488521779)

[10.3 Demonstration 32](#_Toc488521780)

[11. 16x32 SRAM 36](#_Toc488521781)

[11.1 Functionality and Specification 36](#_Toc488521782)

[11.2 Simulation 37](#_Toc488521783)

[11.3 Demonstration 38](#_Toc488521784)

[12. Analysis 42](#_Toc488521785)

[12. Conclusion 44](#_Toc488521786)

# **Objective**

In this lab, we are introduced to latches and flip flops and to under the difference between the two and their functionalities. Latches and flip flops are the basic elements for storing information and one latch or flip flop can store one bit of information and they are sequential device that will remember the previous value inputs and outputs to compute the next output. To understand how latches and flip flop works, we will create several latches and flip flops using block diagrams and VHDL code then run waveform simulations and observe the behavior of these devices. Also, we will be using what we have learned from Latches and Flip-Flops to build SRAM.

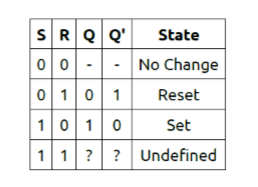
The circuit we will be designing in this lab are:

1. SR Latch
2. Control SR Latch
3. D Latch
4. Positive Edge Triggered Master Slave D Flip Flop
5. Negative Edge Triggered Master Slave D Flip Flop
6. JK Flip Flop
7. T Flip Flop
8. SRAM Cell
9. 16x4 SRAM
10. 16x32 SRAM

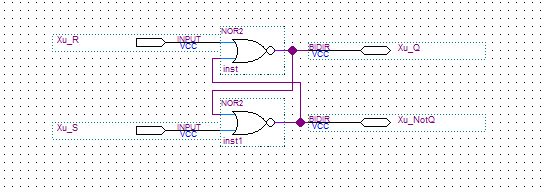
# **SR Latch**

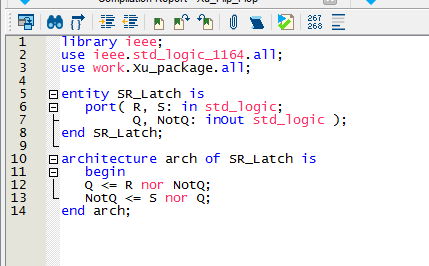
## ***Functionality and Specification***

The SR Latch is a basic storage device used to store a binary bit. This latch has two functions which are Set and Reset which are defined by S and R. Below is the characteristic table that describes the behavior of the SR Latch.



Based on the characteristic table, when input S and R are 0 there are no change for the output Q and Q’ which means whatever the previous outputs were then the current outputs will be the same. When S is 0 and R is 1 then it will conduct the function Reset which resets Q back to 0 and since Q’ stands for not Q it will always be the opposite of Q so Q’ will be 1. When S is 1 and R is 0 then it will conduct the function Set which sets Q to 1 and Q’ will be 0. When both S and R is 1, it is considered as the forbidden input that will put the output in a metastable state where it’s undefined.

Figure 1: Block diagram for SR Latch.

Figure 2: VHDL code for SR Latch.

This is the block diagram and VHDL code for SR Latch, the functionality of these two will be the same. It takes in R and S as inputs and bidirectional outputs which can also function as inputs Q and NotQ. The Boolean function for the outputs will be

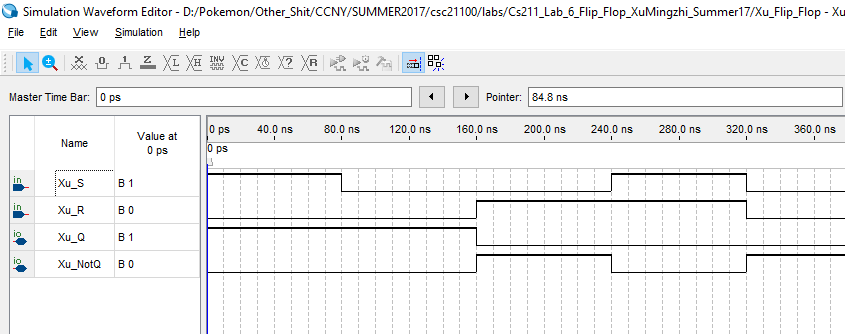
Q = R nor NotQ

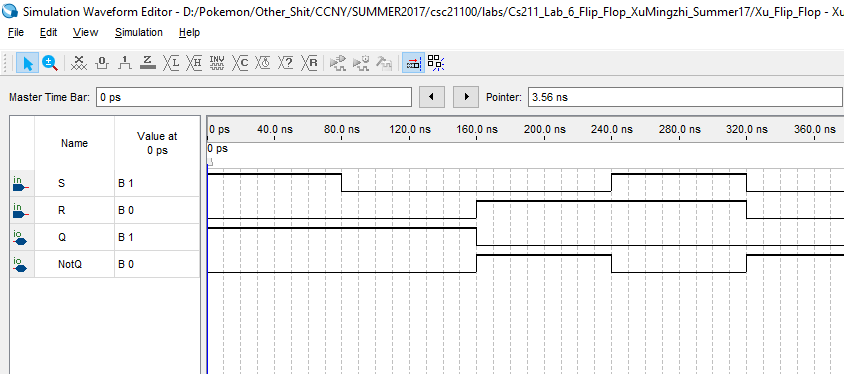
NotQ = S nor Q

Based on the last lab on VHDL language, we learn that unlike other coding languages which executes line by line, VHDL codes are about to execute parallelly which means all the lines are executed at the same time and alternate changes of the values.

## ***Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for SR Latch and compare the results that it’ll produce.

Figure 3: Vector waveform simulation for block diagram of SR Latch.

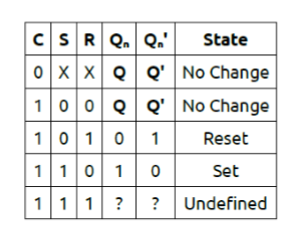
Figure 4: Vector waveform simulation for VHDL code of SR Latch.

We can observe that both simulations produce the same result and it corresponds to characteristic table of SR Latch.

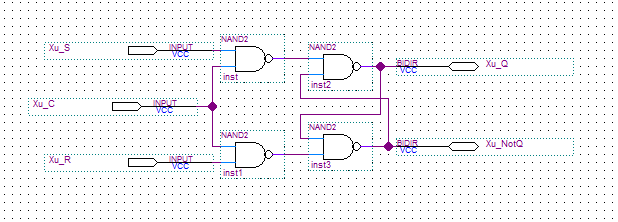
# **Control SR Latch**

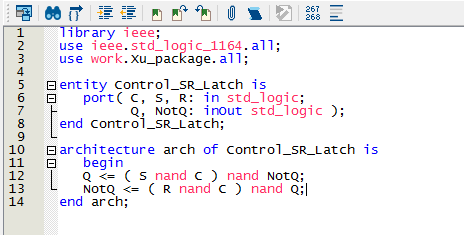
## ***Functionality and Specification***

The Control SR Latch is similar to the SR Latch that was designed in the last section, but this time it will have an extra input of C which stands for Control which means the user will have control when there will be a state change. Below is the characteristic table of the Control SR Latch.



Based on the characteristic table, when C is 0, no change will be made for whatever S or R are. Which means state changes will on occur when C is 1. When C is 1, it will function the same as SR Latch.

Figure 5: Block diagram of Control SR Latch.

Figure 6: VHDL code of Control SR Latch.

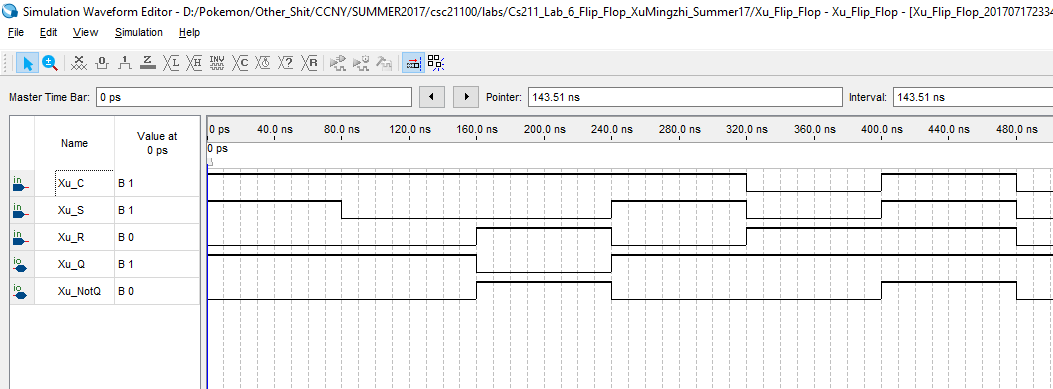
This is the block diagram and VHDL code for Control SR Latch, this time we will use four NAND gates to construct the block diagram and the Boolean function are

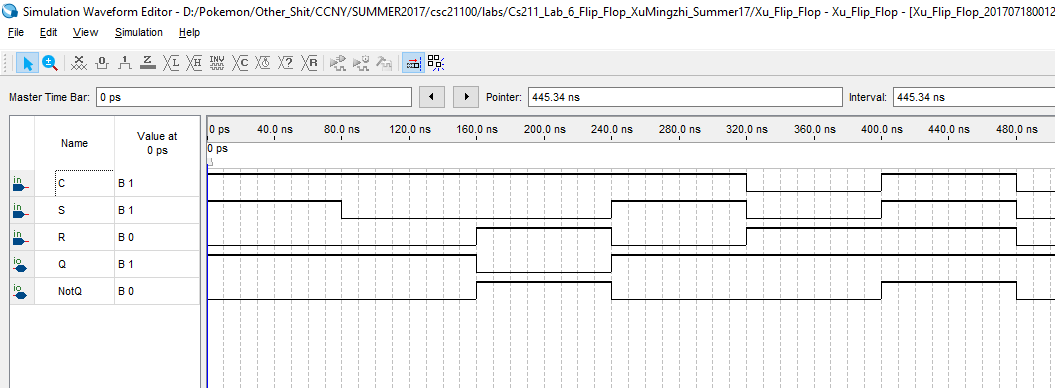
Q = ( S nand C ) nand NotQ

NotQ = ( R nand C ) nand Q

## ***Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for Control SR Latch and compare the results that it’ll produce.

Figure 7: Simulation for block diagram of Control SR Latch.

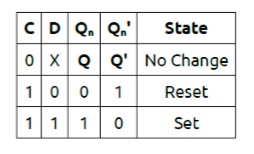
Figure 8: Simulation for VHDL code of Control SR Latch.

We can observe that both simulations produce the same result and it corresponds to characteristic table of Control SR Latch.

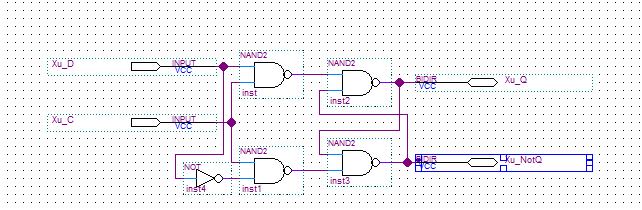
# **D Latch**

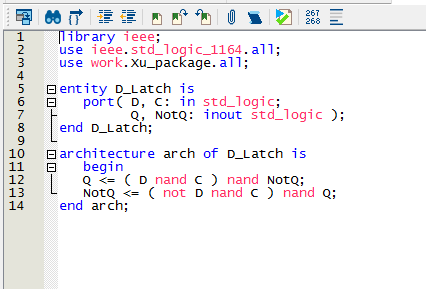
## ***Functionality and Specification***

The D Latch is a modified version of the Control SR Latch which only uses input D to replace Set and Reset. This will make sure that it will never go into metastable state since S and R input will always be opposite of each other due to the use of a NOT gate. Below is the characteristic table of the D Latch.



Based on the characteristic table we can see that the functionality of the D Latch is almost the same the Control SR Latch. When C is 0, there will be no change and will output the previous outputs. When C is 1 and D is 0 it will reset and Q will be 0 and Q’ will be 1. When C is 1 and D is 1 it will set Q to 1 and Q’ to 0.

Figure 9: Block diagram of D Latch.

Figure 10: VHDL code of D Latch.

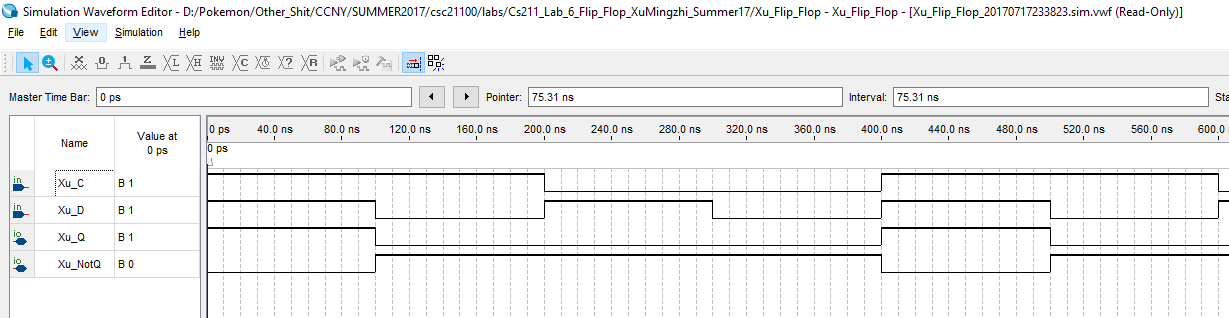
This is the block diagram and VHDL code for the D Latch. We will use a NOT gate and input D to replace the SR input in the Control SR Latch so there the metastable state will never occur. The Boolean function will be

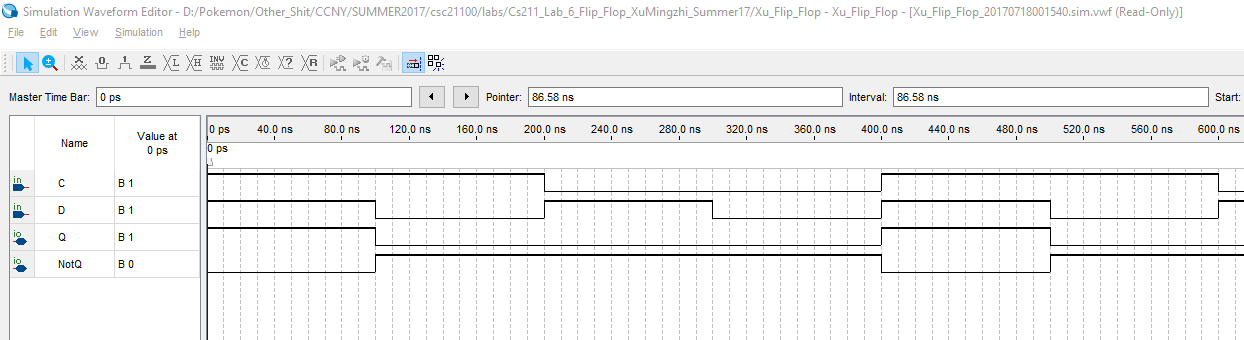
Q = ( D nand C ) nand NotQ

NotQ = ( Not D nand C ) nand Q

## ***Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for D Latch and compare the results that it’ll produce.

Figure 11: Simulation for block diagram of D Latch.

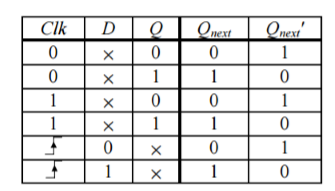
Figure 12: Simulation for VHDL code of D Latch.

We can observe that both simulations produce the same result and it corresponds to characteristic table of D Latch.

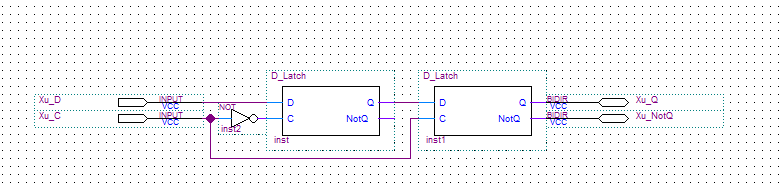
# **Positive Edge Triggered Master Slave D Flip Flop**

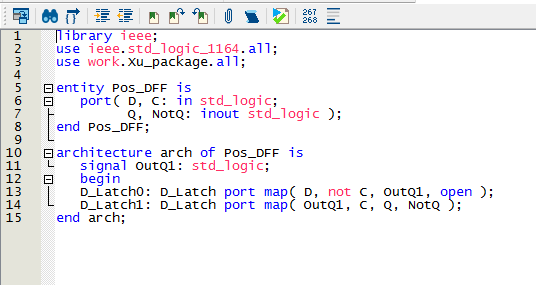
## ***Functionality and Specification***

This is a Positive Edge Triggered Master Slave D Flip Flop, the Master Slave D Flip Flop is made up of two D Latch component. It is different than the D Latch because it triggers the event to change state when the clock signal is going from 0 to 1 or 1 to 0. Since this is a positive edge triggered which means it will change state when the clock signal goes from 0 to 1. Below is the characteristic table for Positive Edge Triggered Master Slave D Flip Flop.



Based on the characteristic table we can see that whenever Clock signal is 0 or 1, the D input doesn’t affect the output Qnext and will output the previous output Q. When the clock signal goes from 0 to 1, the Qnext will change based on D at that moment.

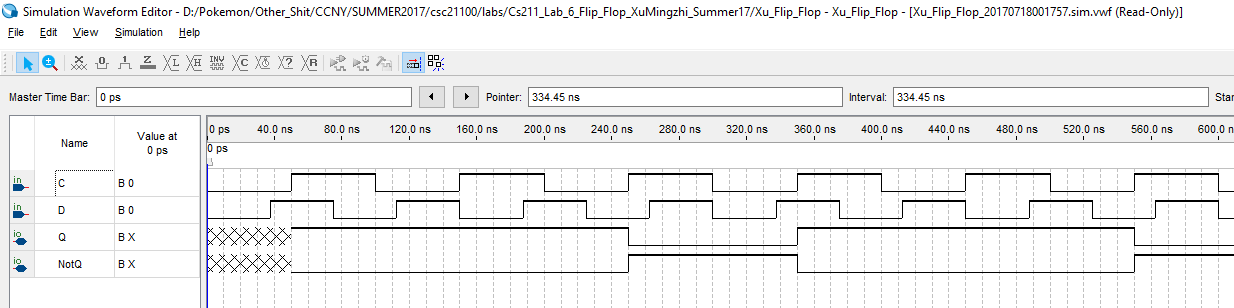
Figure 13: Block diagram for Positive D Flip Flop.

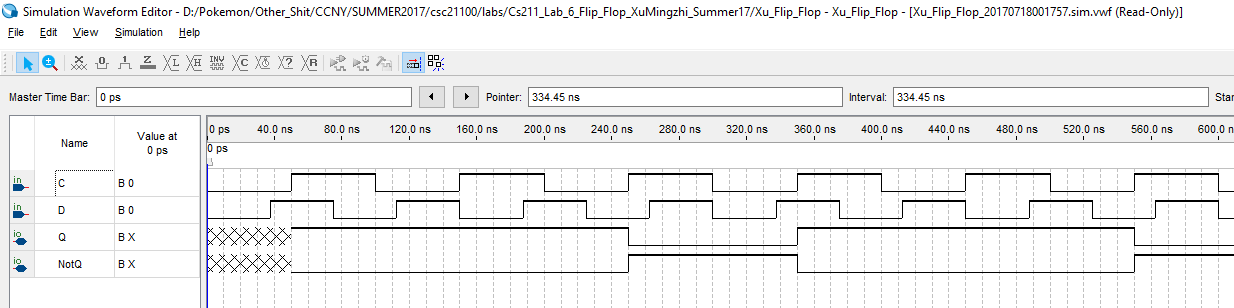
Figure 13: VHDL code for Positive D Flip Flop.

This is the block diagram and VHDL code for Positive Edge Triggered Master Slave D Flip Flop. It uses two D Latch as components and for it to have positive edge trigger, the first clock signal will have a NOT gate.

## ***Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for Positive Edge Triggered Master Slave D Flip Flop and compare the results that it’ll produce.

Figure 14: Simulation for block of Positive D Flip Flop.

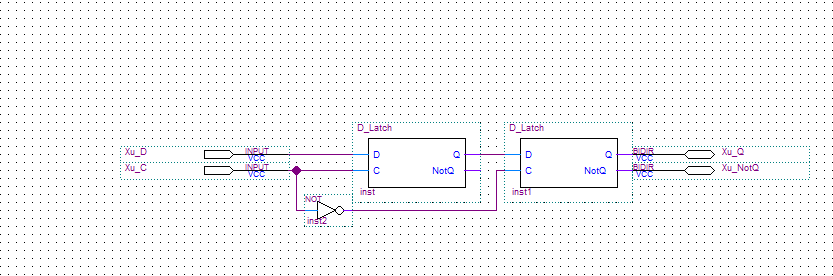
Figure 15: Simulation for VHDL of Positive D Flip Flop.

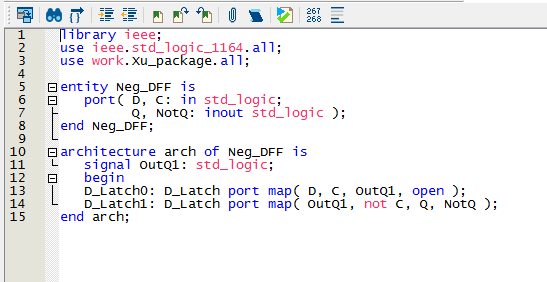
We can observe that both simulations produce the same result and it corresponds to characteristic table of Positive D Flip Flop.

# **Negative Edge Triggered Master Slave D Flip Flop**

## ***Functionality and Specification***

The Negative Edge Triggered Master Slave D Flip Flop functions similarly to the Positive Edge Triggered Master Slave D Flip Flop. The only differences is that it will change state and the clock signal goes from 1 to 0.

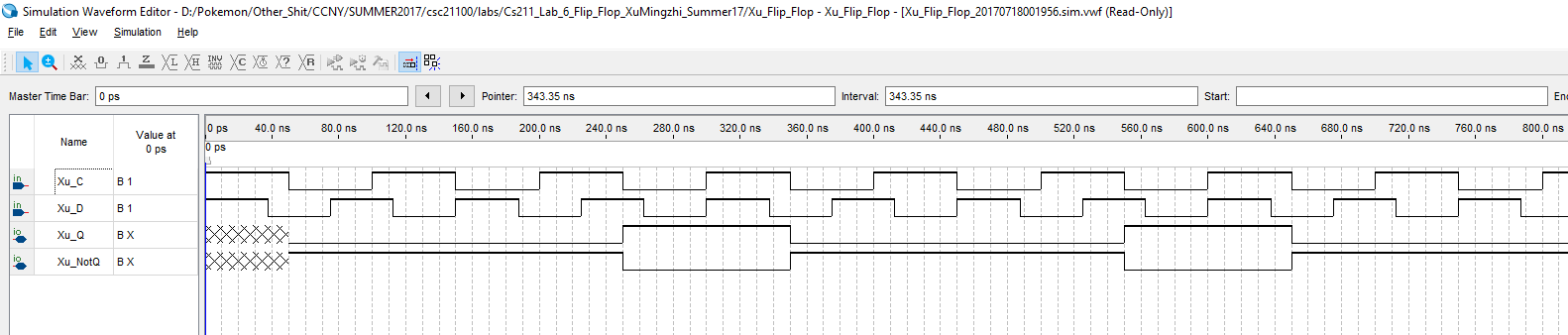
Figure 16: Block diagram for Negative D Flip Flop.

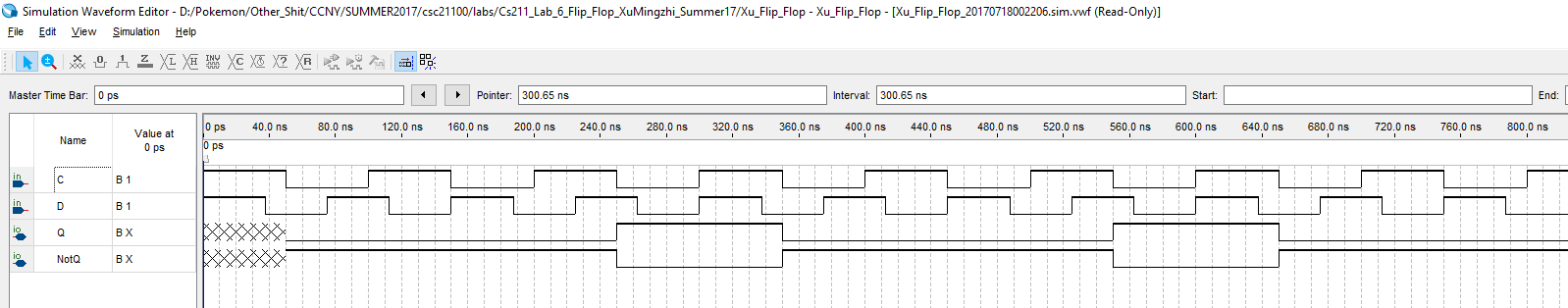
Figure 17: VHDL code for Negative D Flip Flop.

This is the block diagram and VHDL code for the Negative Edge Triggered Master Slave D Flip Flop. We can see that it is very similar to the Positive D Flip Flop, but in order for the clock signal to triggered the event to change state from 1 to 0 we will have the NOT gate connect to the clock input in the slave D Latch unlike the Positive D Flip Flop where the NOT gate goes to the Master D Latch clock input.

## ***Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for Negative Edge Triggered Master Slave D Flip Flop and compare the results that it’ll produce

Figure 18: Simulation for block of Negative D Flip Flop.

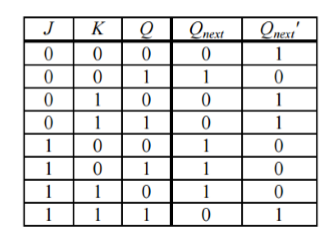
Figure 19: Simulation for VHDL of Negative D Flip Flop.

We can observe that both simulations produce the same result and it corresponds to characteristic table of Negative D Flip Flop.

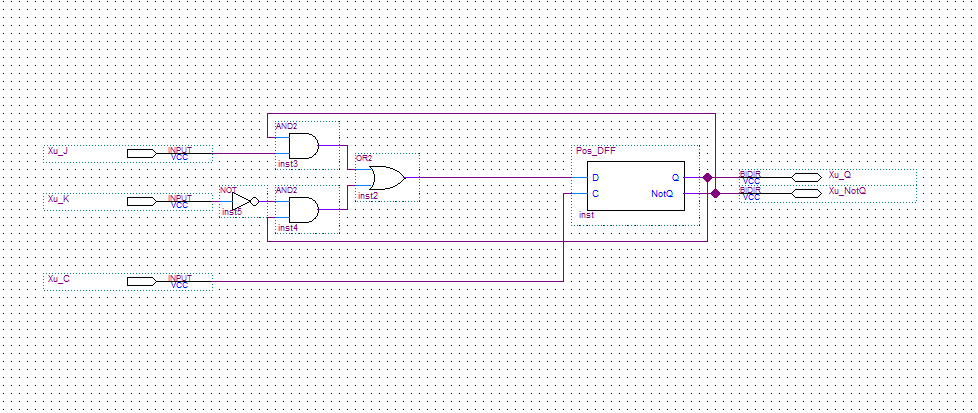
# **JK Flip Flop**

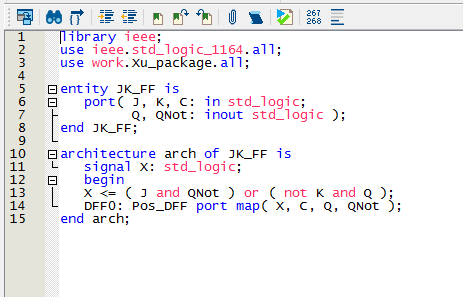
## ***Functionality and Specification***

The JK Flip Flop is very similar to the Control SR Latch but this is a flip flop which means it is a edge triggered device and the JK flip flop uses the Positive D Flip Flop as component. Below is the characteristic table for the JK Flip Flop.



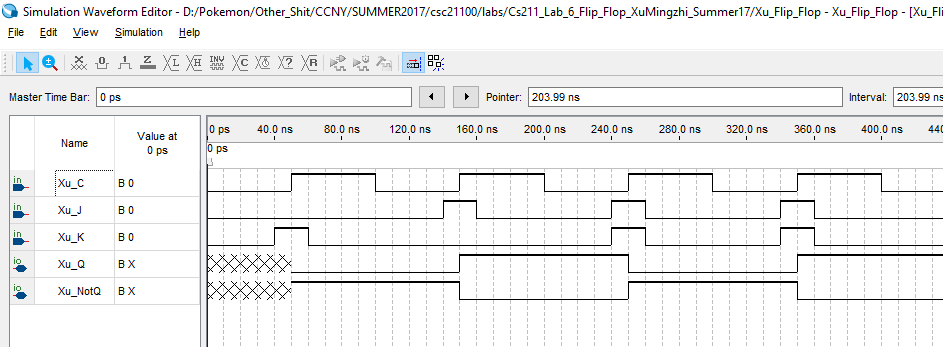
Based on the characteristic table we can observe that when both J and K is 0 there will be no change and Qnext will output the previous output Q. When J is 0 and K is 1 Qnext will Reset to 0. When J is 1 and K is 0 Qnext will Set to 1. When both J and K are 0, instead of going into metastable state like the Control SR Latch, it will instead output the NotQ of the previous Q.

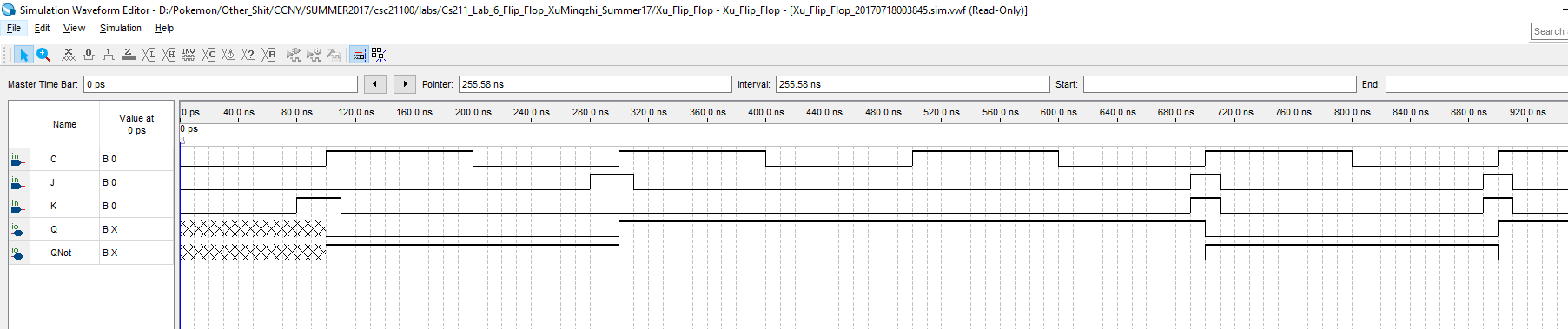
Figure 20: Block diagram of JK Flip Flop.

Figure 21: VHDL code of JK Flip Flop.

## ***Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for JK Flip Flop and compare the results that it’ll produce.

Figure 22: Simulation for block of JK Flip Flop.

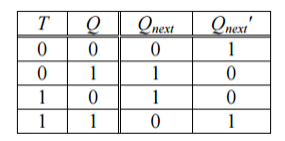
Figure 23: Simulation for VHDL of JK Flip Flop.

We can observe that both simulations produce the same result and it corresponds to characteristic table of JK Flip Flop.

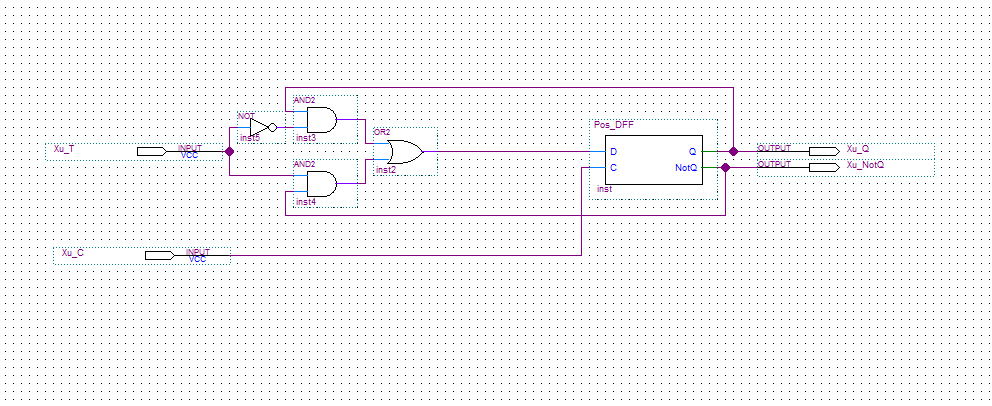
# **T Flip Flop**

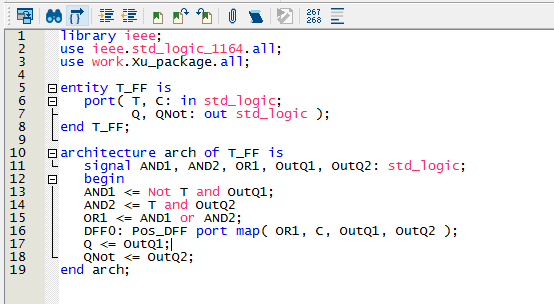
## ***Functionality and Specification***

The T Flip Flop has the same function as the JK Flip Flop. Below is the characteristic table for the T Flip Flop.



Based on the characteristic table, we can observe that when T is 0 Qnext will output the previous Q. When T is 1 Qnext will output the QNot of the previous Q.

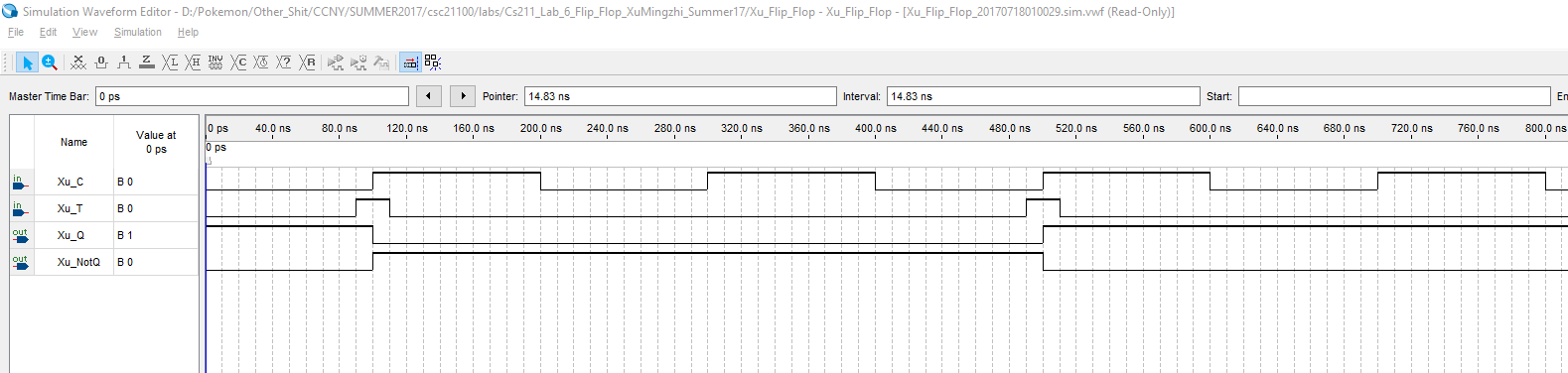
Figure 24: Block diagram of T Flip Flop.

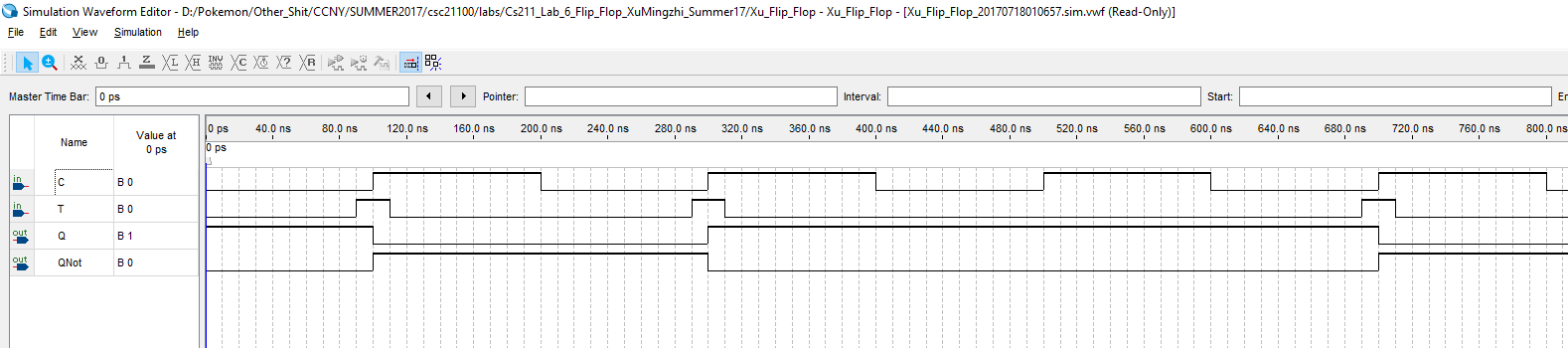
Figure 25: VHDL code of T Flip Flop.

This is the block diagram and VHDL code for the T Flip Flop. The T Flip Flop functions the same as the JK Flip Flop when both J and K is 1 which will output the QNot of the previous Q.

## ***8.2 Simulation***

Now we will run the waveform simulation for both block diagram and VHDL code for T Flip Flop and compare the results that it’ll produce.

Figure 26: Simulation for block of T Flip Flop.

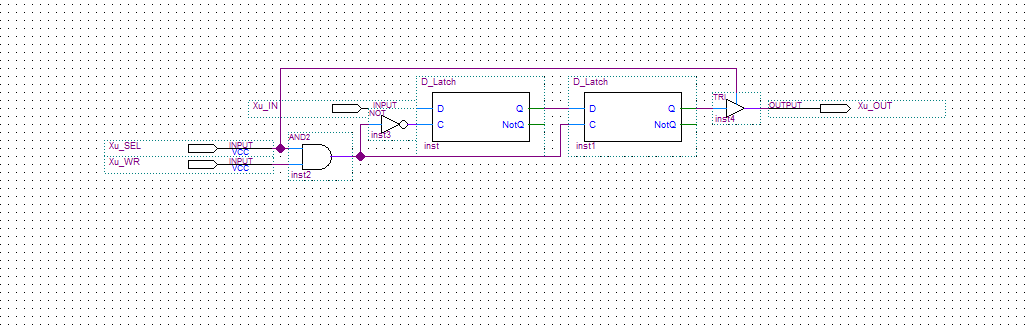
Figure 27: Simulation for VHDL of T Flip Flop.

We can observe that both simulations produce the same result and it corresponds to characteristic table of T Flip Flop.

# **SRAM Cell**

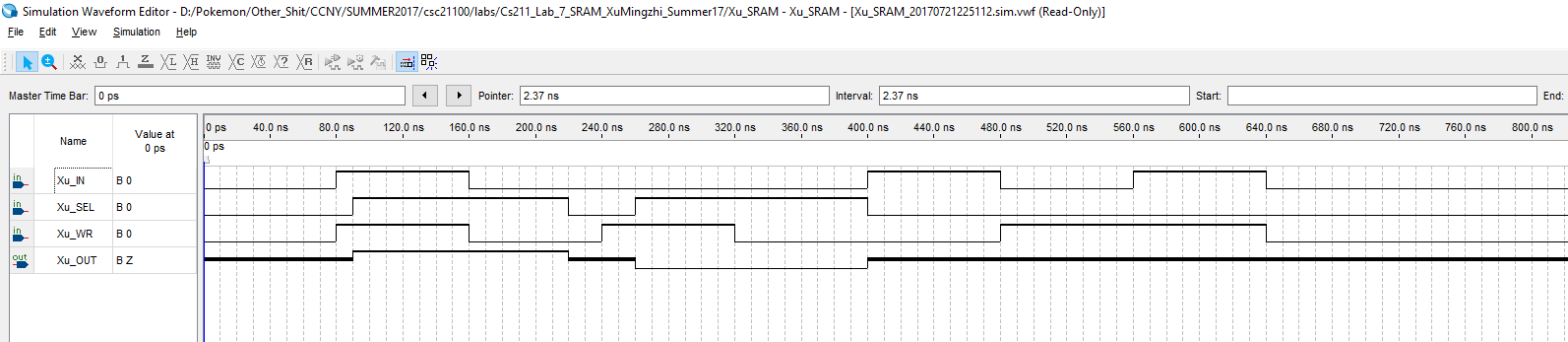
## ***Functionality and Specification***

The single SRAM Cell is made from using a Positive Edge Triggered Master Slave D Flip Flop, and the functionality is same as a Positive D Flip Flop, which is to store memory of 1-bit at the rising edge of the clock signal. The difference is that in the SRAM Cell, the clock signal is determined by the Select Chip and the Write Enable, we will use a Tristate Logic Buffer which connects to the Select Chip input meaning that Select Chip must be on for the Output to receive a result and the Write Enable is the event that triggers at the rising edge of the clock signal to enable this SRAM Cell to store either 0 or 1.

Figure 28: Block diagram of a SRAM Cell.

## ***Simulation***

In the simulation, we will test the Inputs and Outputs of a SRAM Cell.

Figure 29: Simulation for SRAM Cell.

We can observe from the simulation that when the inputs IN, SEL, and WR are 0, the output will be Z because of the Tristate Logic Buffer Gate since SEL is not on, the output will not receive and results. We can also see that when the inputs IN, SEL, and WR is 1 the output will be IN. The inputs and outputs corresponds to the D Flip Flop characteristic table but the clock signal is determined by both SEL and WR.

# **16x4 SRAM**

## ***Functionality and Specification***

The functionality of a 16x4 SRAM is the same as the single SRAM Cell but this time there will be 16 address locations and each address is able to store a 4-bit binary input. To build a 16x4 SRAM we can put together 64 SRAM Cells or by stacking. The idea of stacking is first we make a 16x1 by stacking 16 single SRAM Cells together which looks like this.

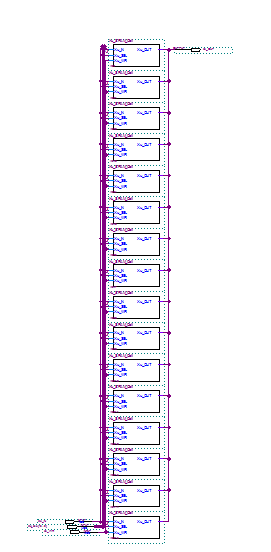
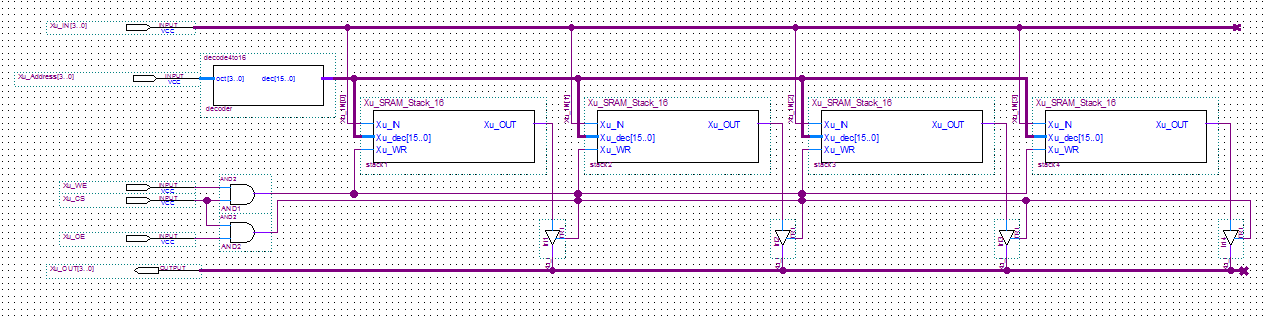
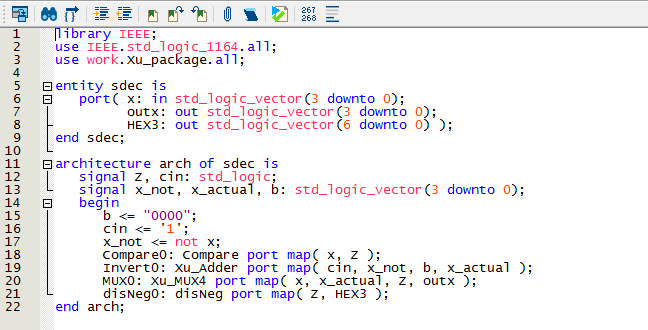


Figure 30: Block diagram of a 16x1 SRAM

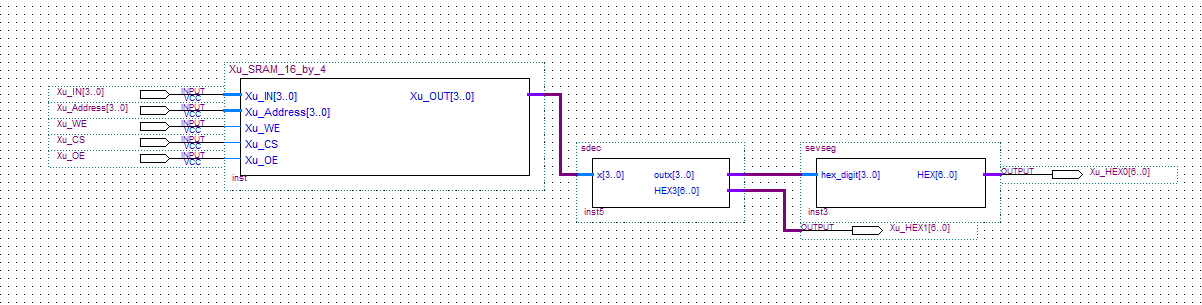
The 16x1 SRAM will be able to store 1-bit of binary in 16 different address by using a 4 to 16 decoder to determine which SRAM Cell will be written and displayed. The next step into creating a 16x4 SRAM, we will make the 16x1 SRAM into a symbol and use 4 of it as components which makes up a 16x4 SRAM.

Figure 31: Block diagram of 16x4 SRAM.

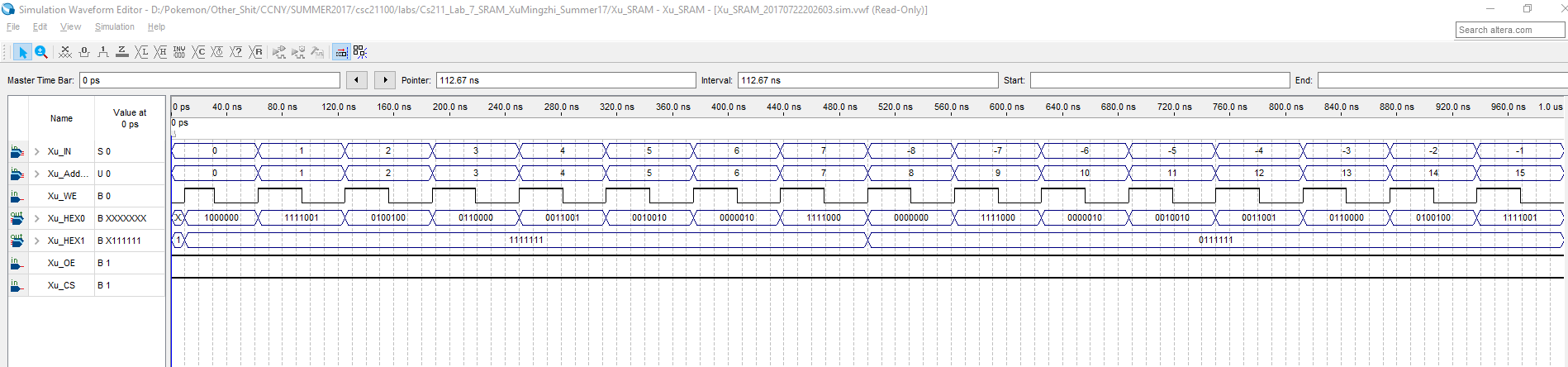
As you can see in the 16x4 SRAM block diagram, there will be 4 inputs which are the 4-bit inputs that will be stored in any of the 16 addresses determined by another 4 address inputs that goes through a 4 to 16 decoder. Then everything else will be the same as the SRAM Cell where there will be a Selected Chip input CS and Write Enable input WE which determines the clock signal and there will be another input OE to display the output that connects to the Tristate Logic Buffer Gates. In the SRAM Cell, the TRI gate is connected to the CS input but this time it will be a new input OE. Next, we will want to display the output result onto the hexadecimal display in signed decimal form. To do that we will use what we have learned in the VHDL Adder Subtractor Lab where we used an inverter, negative detection, and seven-segment display to display signed decimals.

Figure 32: VHDL code to display signed decimal.

In the VHDL code, I’ve used component Compare which is used for negative detection and the Xu\_Adder component which inverts the binary input and it will go through a 2 to 1 multiplexer to determine which input to use. If it is greater than 7 it will use the inverted inputs else it will use the normal inputs. The disNeg component is used to display the negative sign on the board determined by the Compare component.

Figure 33: Block diagram for 16x4 SRAM Display in signed decimal.

## ***Simulation***

Figure 34: Simulation for 16x4 SRAM display.

In the simulation, we will assign signed decimal inputs from -8 to 7 each to one of the 16 addresses, and we can see that HEX1 which is the negative sign display is on when it is a negative input and the simulation corresponds to what we have done in the previous lab of VHDL Adder Subtractor when we needed to display signed decimal on the seven segment displays.

## ***Demonstration***

The PIN assignment of the inputs and outputs on the DE1-SoC Board.

Xu\_IN[0] is assigned to SW[0] which is PIN\_AB12

Xu\_IN[1] is assigned to SW[1] which is PIN\_AC12

Xu\_IN[2] is assigned to SW[2] which is PIN\_AF9

Xu\_IN[3] is assigned to SW[3] which is PIN\_AF10

Xu\_Address[0] is assigned to SW[4] which is PIN\_AD11

Xu\_Address[1] is assigned to SW[5] which is PIN\_AD12

Xu\_Address[2] is assigned to SW[6] which is PIN\_AE11

Xu\_Address[3] is assigned to SW[7] which is PIN\_AC9

Xu\_WE is assigned to SW[8] which is PIN\_AD10

Xu\_CS is assigned to SW[9] which is PIN\_AE12

Xu\_OE is assigned to KEY[0] which is PIN\_AA14

Xu\_HEX0[0] is assigned to PIN\_AE26

Xu\_HEX0[1] is assigned to PIN\_AE27

Xu\_HEX0[2] is assigned to PIN\_AE28

Xu\_HEX0[3] is assigned to PIN\_AG27

Xu\_HEX0[4] is assigned to PIN\_AF28

Xu\_HEX0[5] is assigned to PIN\_AG28

Xu\_HEX0[6] is assigned to PIN\_AH28

Xu\_HEX1[0] is assigned to PIN\_AJ29

Xu\_HEX1[1] is assigned to PIN\_AH29

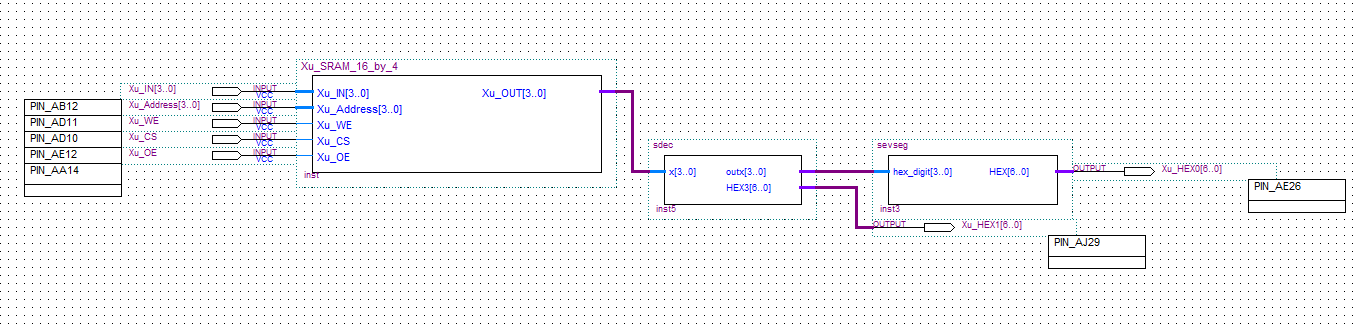
Xu\_HEX1[2] is assigned to PIN\_AH30

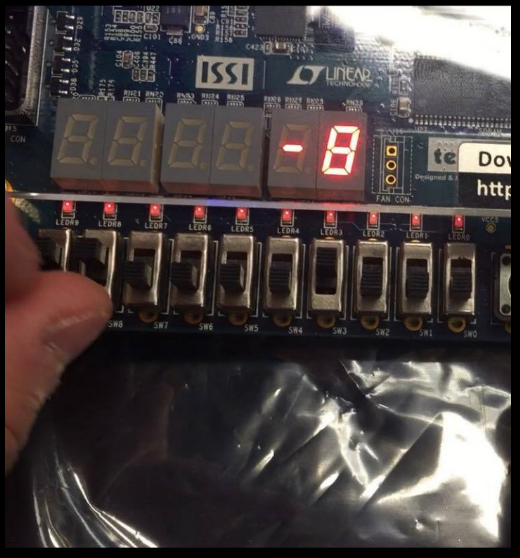
Xu\_HEX1[3] is assigned to PIN\_AG30

Xu\_HEX1[4] is assigned to PIN\_AF29

Xu\_HEX1[5] is assigned to PIN\_AF30

Xu\_HEX1[6] is assigned to PIN\_AD27

Figure 35: Block diagram of 16x4 SRAM with PIN assignments.

Figure 36: Storing 1000 in address 0000.

HEX1

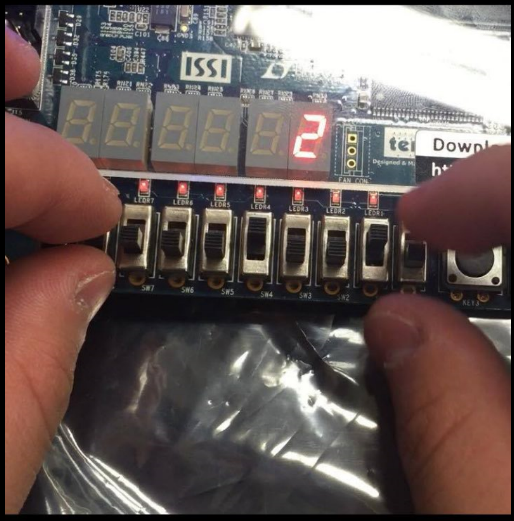
HEX0

WE

CS

Address[3..0]

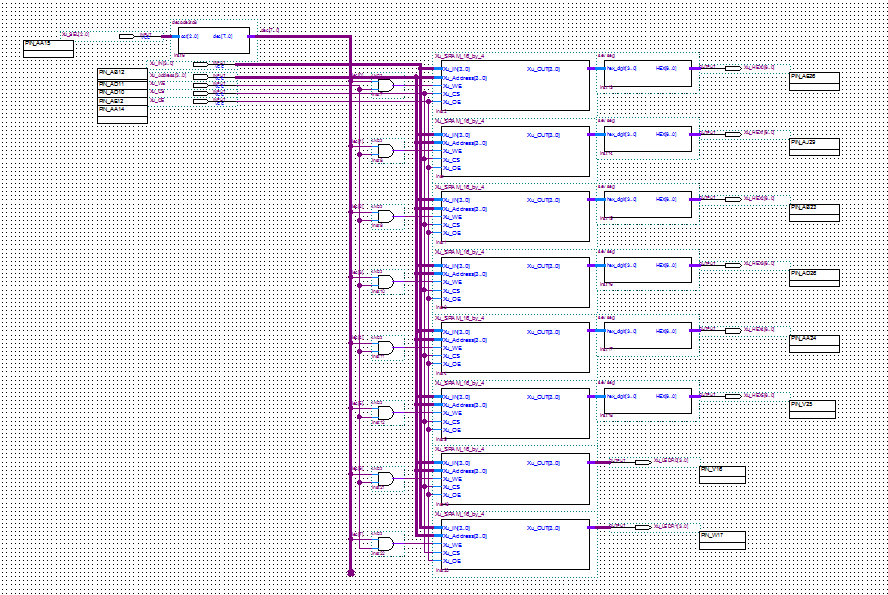
Input[3..0]

Figure 37: Storing 0010 in address 0001.

# **16x32 SRAM**

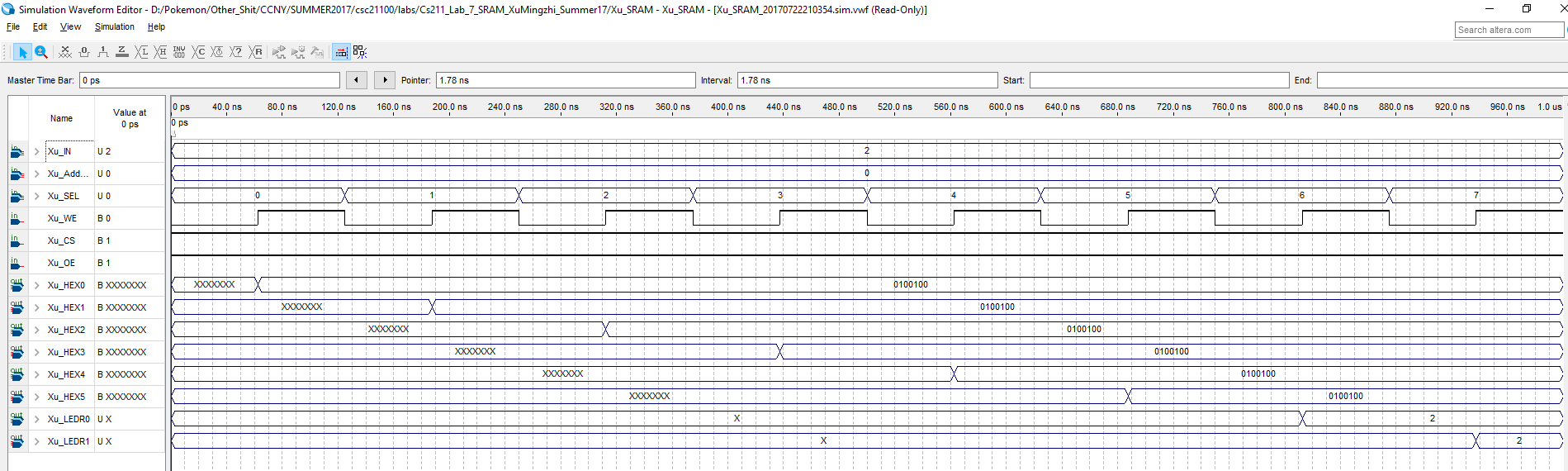
## ***Functionality and Specification***

The functionality of a 16x32 SRAM has the same ideas as the SRAM Cell and the 16x4 SRAM, but this time it has 16x32 storage which can store 512-bits. There will be 16 address locations and each address is able to store 32-bits. To do this, we will use the 16x4 SRAM we just created and make it into a symbol and use 8 of the symbols as components which makes up to be the 16x32 SRAM.

Figure 38: Block diagram of 16x32 SRAM.

The idea is same as 16x4 SRAM but there aren’t enough switches to have a 32 inputs, so therefore I’ve made a selector using the last three KEY components on the board as SELECTOR to determine which one of the eight position of a 32-bit input it’ll be writing using only 4 inputs. Therefore, I will need a 3 to 8 decoder which functions the same as how the addresses uses a 4 to 16 decoder to determine which address location it is being written to but the 3 to 8 decoder determines which 4 of the 32 position it will be written to. Since there are only 6 seven segment displays and we are displaying the result in hexadecimal form which means the seven segment displays can only display up to 24-bits. Therefore, I’ve used 8 LEDR to store the last 8-bits of the 32-bit inputs.

## ***Simulation***

Figure 39: Simulation for 16x32 SRAM.

In the simulation, I will be storing the binary input of 0010 which is 2 in hexadecimal to all 8 positions of display in one address. We can observe from the simulation that when we finish writing to all 8-different position of the display, it’ll display 2 on all 6 seven segment display and 0010 on the LEDR for the 7th and 8th display positions.

## ***11.3 Demonstration***

The PIN assignment of the inputs and outputs on the DE1-SoC Board.

Xu\_IN[0] is assigned to SW[0] which is PIN\_AB12

Xu\_IN[1] is assigned to SW[1] which is PIN\_AC12

Xu\_IN[2] is assigned to SW[2] which is PIN\_AF9

Xu\_IN[3] is assigned to SW[3] which is PIN\_AF10

Xu\_Address[0] is assigned to SW[4] which is PIN\_AD11

Xu\_Address[1] is assigned to SW[5] which is PIN\_AD12

Xu\_Address[2] is assigned to SW[6] which is PIN\_AE11

Xu\_Address[3] is assigned to SW[7] which is PIN\_AC9

Xu\_WE is assigned to SW[8] which is PIN\_AD10

Xu\_CS is assigned to SW[9] which is PIN\_AE12

Xu\_OE is assigned to KEY[0] which is PIN\_AA14

Xu\_SEL[0] is assigned to KEY[1] which is PIN\_AA15

Xu\_SEL[1] is assigned to KEY[2] which is PIN\_W15

Xu\_SEL[2] is assigned to KEY[3] which is PIN\_Y16

Xu\_HEX0[0] is assigned to PIN\_AE26

Xu\_HEX0[1] is assigned to PIN\_AE27

Xu\_HEX0[2] is assigned to PIN\_AE28

Xu\_HEX0[3] is assigned to PIN\_AG27

Xu\_HEX0[4] is assigned to PIN\_AF28

Xu\_HEX0[5] is assigned to PIN\_AG28

Xu\_HEX0[6] is assigned to PIN\_AH28

Xu\_HEX1[0] is assigned to PIN\_AJ29

Xu\_HEX1[1] is assigned to PIN\_AH29

Xu\_HEX1[2] is assigned to PIN\_AH30

Xu\_HEX1[3] is assigned to PIN\_AG30

Xu\_HEX1[4] is assigned to PIN\_AF29

Xu\_HEX1[5] is assigned to PIN\_AF30

Xu\_HEX1[6] is assigned to PIN\_AD27

Xu\_HEX2[0] is assigned to PIN\_AB23

Xu\_HEX2[1] is assigned to PIN\_AE29

Xu\_HEX2[2] is assigned to PIN\_AD29

Xu\_HEX2[3] is assigned to PIN\_AC28

Xu\_HEX2[4] is assigned to PIN\_AD30

Xu\_HEX2[5] is assigned to PIN\_AC29

Xu\_HEX2[6] is assigned to PIN\_AC30

Xu\_HEX3[0] is assigned to PIN\_AD26

Xu\_HEX3[1] is assigned to PIN\_AC27

Xu\_HEX3[2] is assigned to PIN\_AD25

Xu\_HEX3[3] is assigned to PIN\_AC25

Xu\_HEX3[4] is assigned to PIN\_AB28

Xu\_HEX3[5] is assigned to PIN\_AB25

Xu\_HEX3[6] is assigned to PIN\_AB22

Xu\_HEX4[0] is assigned to PIN\_AA24

Xu\_HEX4[1] is assigned to PIN\_Y23

Xu\_HEX4[2] is assigned to PIN\_Y24

Xu\_HEX4[3] is assigned to PIN\_W22

Xu\_HEX4[4] is assigned to PIN\_W24

Xu\_HEX4[5] is assigned to PIN\_V23

Xu\_HEX4[6] is assigned to PIN\_W25

Xu\_HEX5[0] is assigned to PIN\_V25

Xu\_HEX5[1] is assigned to PIN\_AA28

Xu\_HEX5[2] is assigned to PIN\_Y27

Xu\_HEX5[3] is assigned to PIN\_AB27

Xu\_HEX5[4] is assigned to PIN\_AB26

Xu\_HEX5[5] is assigned to PIN\_AA26

Xu\_HEX5[6] is assigned to PIN\_AA25

Xu\_LEDR0[0] is assigned to PIN\_V16

Xu\_LEDR0[1] is assigned to PIN\_W16

Xu\_LEDR0[2] is assigned to PIN\_V17

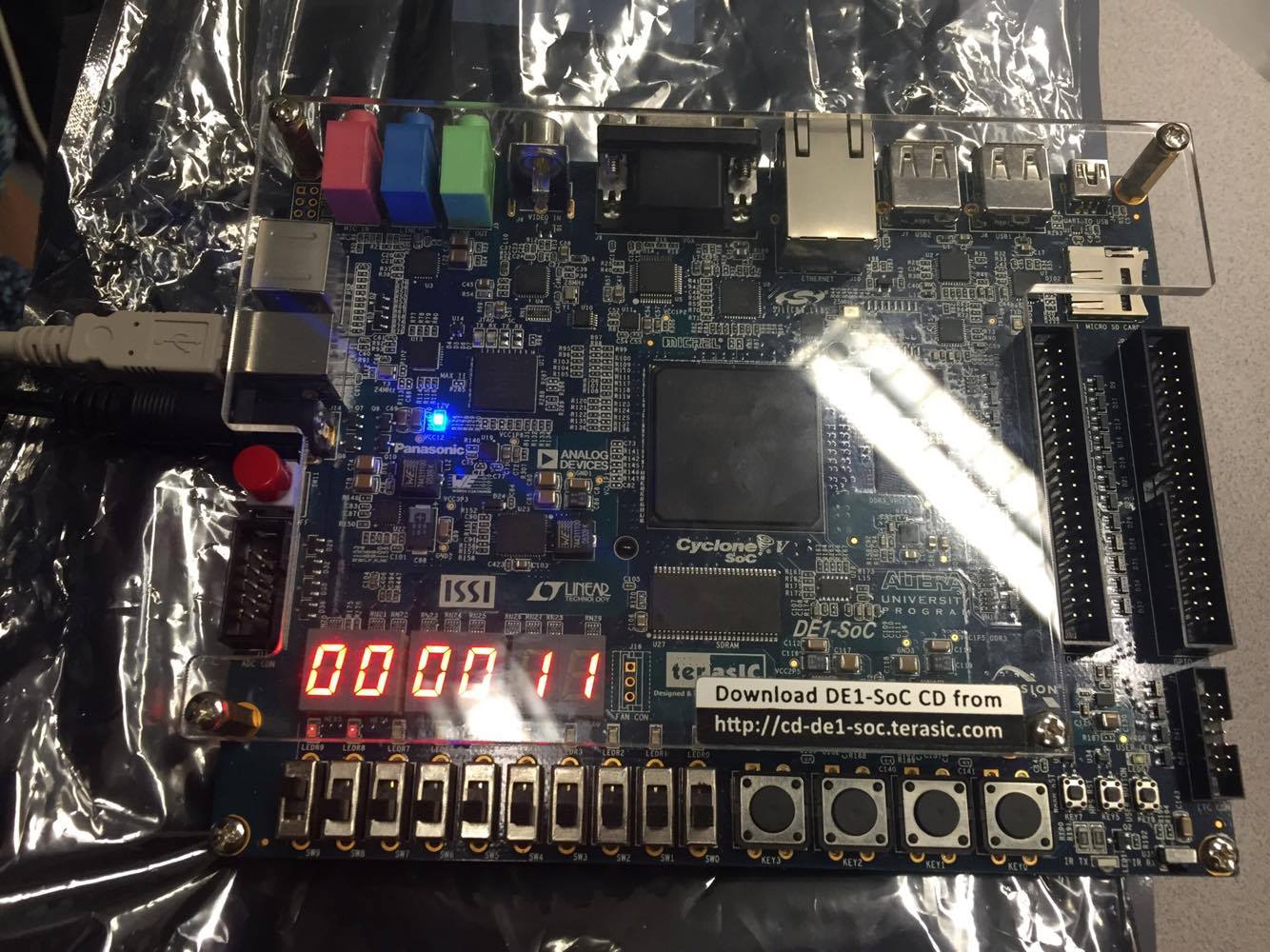
Xu\_LEDR0[3] is assigned to PIN\_V18

Xu\_LEDR1[0] is assigned to PIN\_W17

Xu\_LEDR1[1] is assigned to PIN\_W19

Xu\_LEDR1[2] is assigned to PIN\_Y19

Xu\_LEDR1[3] is assigned to PIN\_W20

Figure 40: Storing 0000 0000 0000 0000 0000 0000 0001 00001 to address 0000.

LEDR 8-0

Input[3..0]

WE

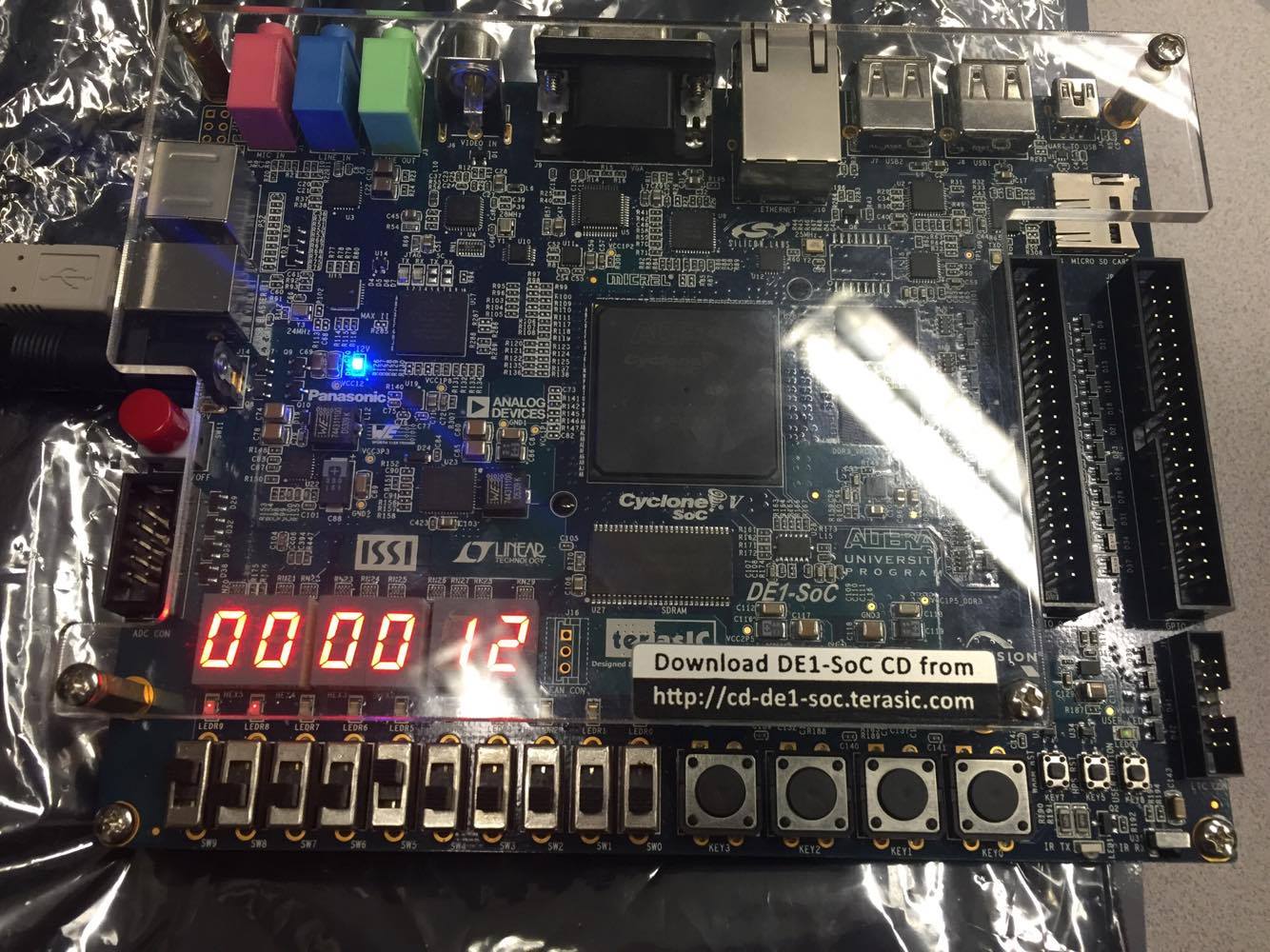
CS

HEX[5..0]

Address3-0

OE

SEL[3..0]

Figure 41: Storing 0000 0000 0000 0000 0000 0000 0001 0010 in address 0010.

# **Analysis**

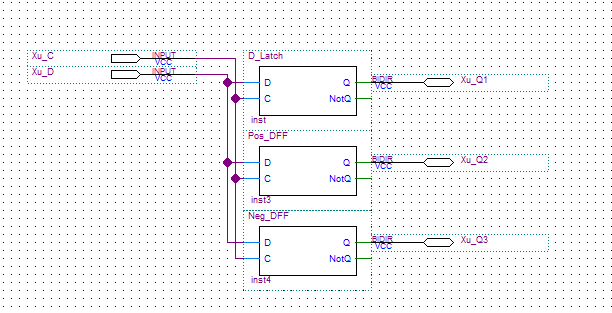
The difference between latches and flip-flops is that they have different event triggers for the clock signal. For latches state change will occur either when clock is 0 or 1 depending if it is a positive or negative D Latch. For flip-flops state change will occur when clock signal changes from 0 to 1 or from 1 to 0. It matters because Flip-Flops can be more flexible than Latches.

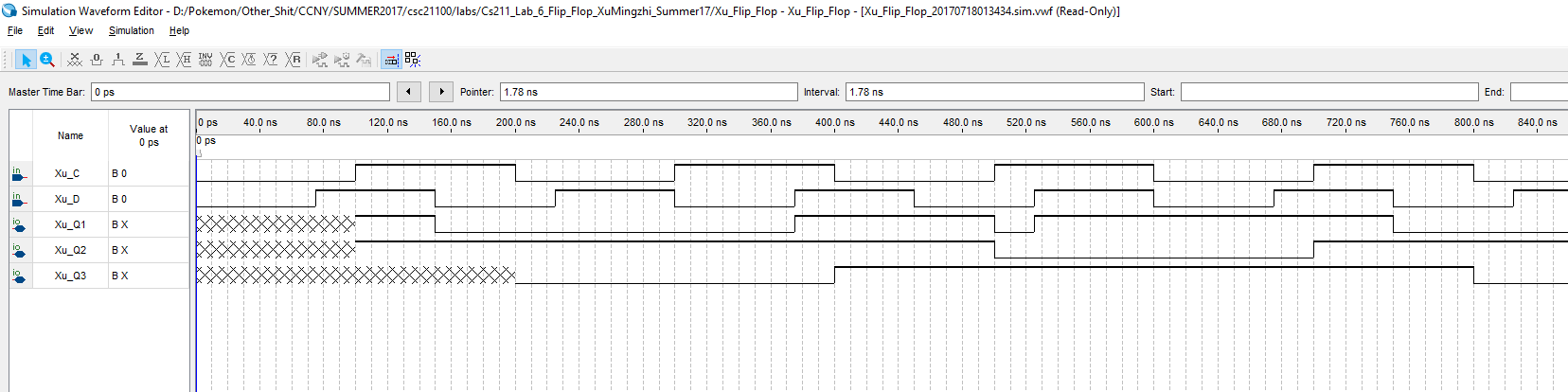
We want to avoid the scenario when S and R is 1 in an SR Latch because it will lead to metastable state which means there will be no indeterminate state, it could be avoided using a D Latch since the input D will replace S and R along with a NOT gate which means S and R will always be opposite of each other so metastable state will never occur.

The differences between edge-triggered and level triggered device is the same as flip-flops and latches. Latches are level triggered device which state change will occur when the clock is either 0 or 1. Flip-Flops are edge-triggered device which state change will occur when the clock goes from 0 to 1 or from 1 to 0. The T Flip Flop is an edge-triggered device in this lab.

Flip-Flops are always required to be clocked or else it will ignore the other inputs and there won’t be any outputs.

Now we will compare the results of a D Latch with Positive D Flip Flop and Negative D Flip Flop.

Figure : Block diagram to compare D Latch and D Flip Flops.

Figure : Simulation for compare D Latch and D Flip Flops.

We can observe from the simulation that for D Latch outputs 1 for Q1 when C is 1 and D is 1 and outputs 0 for Q1 when C is 1 and D is 0 and Q1 will output the previous Q1 when C is 0 and D can be either 0 or 1. The Positive and Negative Flip Flop functions the same as the D Latch but the state change occurs at the edge from 0 to 1 for Positive and from 1 to 0 for Negative.

# **Conclusion**

In this lab, we learned how latches and flip-flops works as sequential devices which sends feedbacks from output back to input. We also learned the difference between latches and flip-flops which have different event trigger, for latches it is a level triggered device which will have state change occur to output when the clock signal is either 0 or 1. For flip-flops it is an edge triggered device which will have a state change occur to output when the clock signal changes from 0 to 1 or from 1 to 0. These devices will also remember previous states which means it is storing 1-bit binary information. We used the Positive D Flip Flop to construct a single cell of SRAM which allows the storage of 1-bit memory and using that one cell SRAM we can using stacking and created a 16x4 RAM which has 16 address locations and each address allows the storage of 4-bit input. We also created a 16x32 SRAM which has 16 address locations and each address allows the storage of 32-bit input.