EXPERIMENT #5

An 8-Bit Multiplier in SystemVerilog

Name: Chuangy Zhang

NetIDL: czhan30

Name:Homero Vazquez

NetID: hvazqu6

Introduction:

For this experiment we created an 8-bit multiplier circuit. The circuit contains 2 8-bit registers A and B. Those registers will be combined serially to form a 16-bit output that will hold the end value of the multiplication of 2 numbers. The numbers are represented in 2's Complement form. There are 4 inputs to the Control Unit: Run, ClearA_LoadB, Reset and Clk. And there are 8 switches that will load the multiplier to Register B when signal ClearA_LoadB is high, and it will load the multiplicand to a 9-bit adder when the Run signal is high. With the help of all of the signals the circuit will shift and add values depending on the least significant bit of register B and at the end it will give the end result of the 2 numbers multiplied.

Pre-Lab Questions:

a. Rework the multiplication example on page 5.2 of the lab manual, as in compute 00000111 * 11000101 in a table similar to the example;

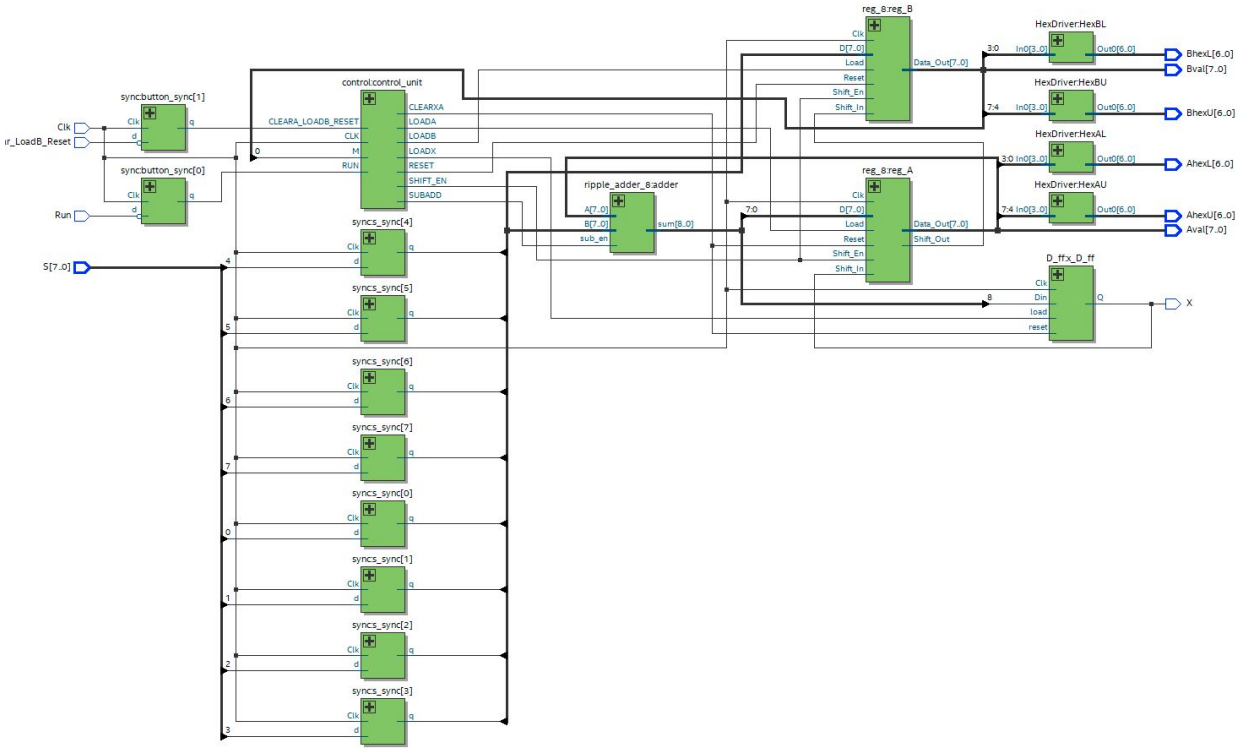
Function	X	A	B	M
Clear Load	0	0000 0000	1100 0101	1
ADD	0	0000 0111	1100 0101	1
SHIFT 0	0	0000 0011	1110 0010	0
SHIFT 1	0	0000 0001	1111 0001	1
ADD	0	0000 1000	1111 0001	1
SHIFT 2	0	0000 0100	0111 1000	0
SHIFT 3	0	0000 0010	0011 1100	0
SHIFT 4	0	0000 0001	0001 1110	0
SHIFT 5	0	0000 0000	1000 1111	1
ADD	0	0000 0111	1000 1111	1
SHIFT 6	0	0000 0011	1100 0111	1
SUB	1	1111 1100	1100 0111	1
SHIFT 7	1	1111 1110	0110 0011	1

Written Description and Diagrams of Multiplier Circuit:

a. Summary of Operation:

For this circuit we start with the ClearA_LoadB signal which clearls Register A and flip-flop X and then loads the value of the 8 Switches to Register B. Then when we hit the signal Run to high and the least significant bit of B is 1 we will add the value of the switches to the value of A then we will arithmetically right shift the bits of X A and B. We will repeat this process until we have shifted 7 times then we will check if the least significant beat of B is 1 then we will subtract the switch values to the values of register A then shift, but if LSB of B is 0 then we will just shift and output our result.

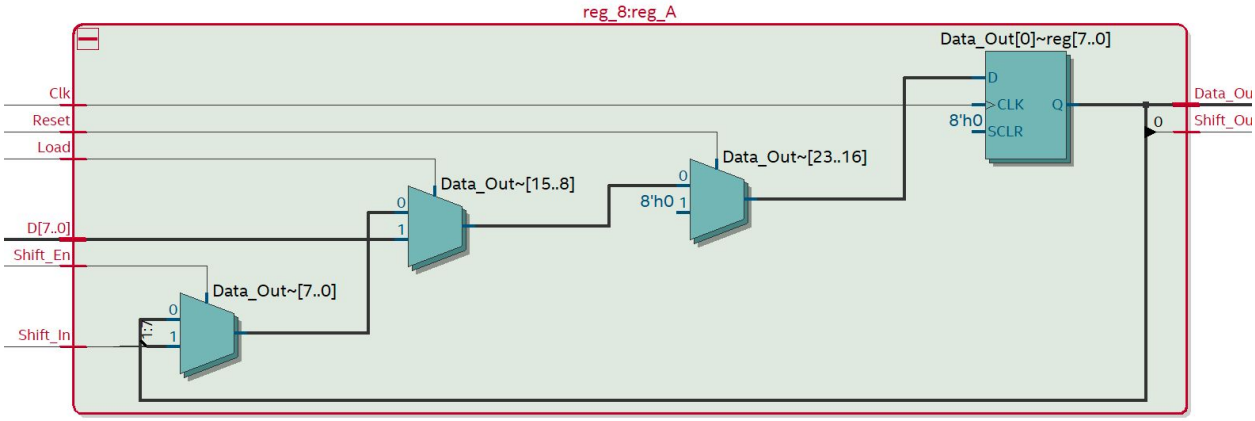
b. Top Level Block Diagram.



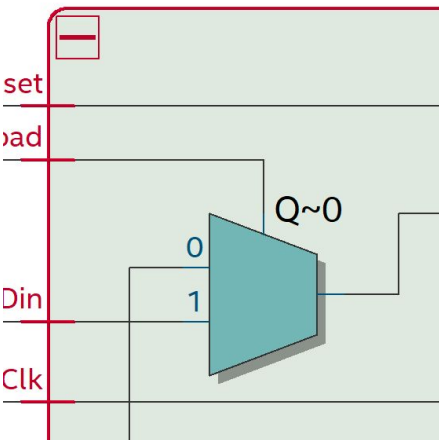
c. Written Description of .sv Modules

- **reg8:**
  Module: reg8.sv
  Inputs: [7:8] D, Clk, Reset, Shift_In, Load, Shift_En
  Outputs: [7:0] Data_Out, Shift_Out
  Description: This is a positive edge triggered 8-bit shift register with synchronous reset, load and shift.

Purpose: This module is used to create registers A and B that store the operands but they will also turn into the 16-bit output by putting them in series.

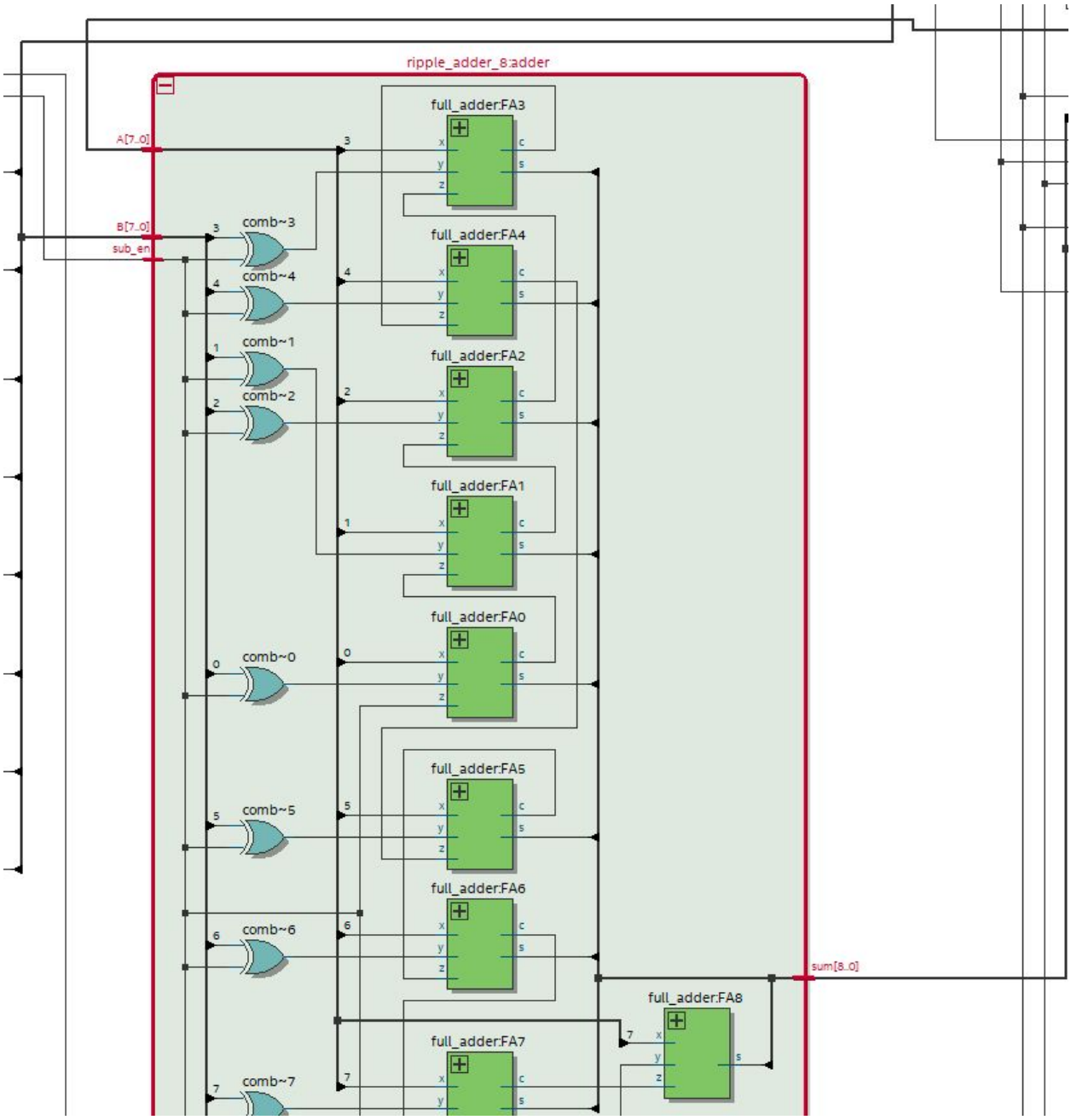


- **mux2to1:**
  Module: mux2to1.sv
  Inputs: [1:0] din, sel
  Outputs: out
  Description: This is a 2:1 Multiplexer. It takes 2 inputs and the sel signal selects which one will be the output.
  Purpose: The purpose of this module is to select between different inputs that go into the registers from the control unit.



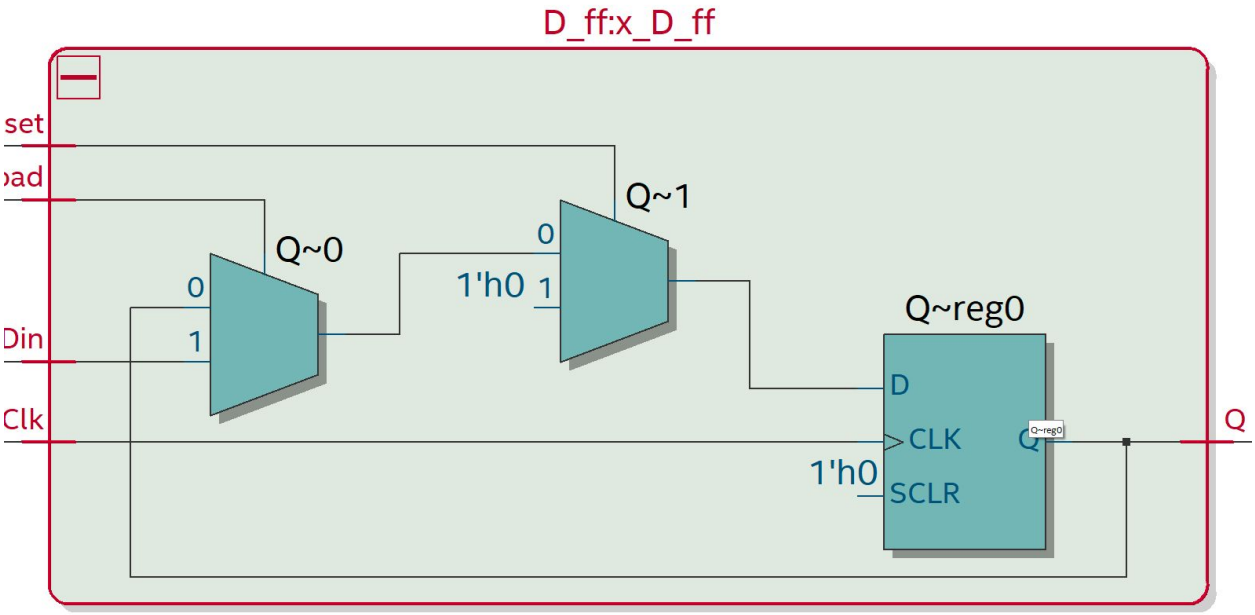
- **ripple_adder_8:**
  Module: ripple_adder_8.sv
  Inputs: [7:0] A, [7:0] B, sub_en
  Outputs:[8:9] sum

Description: This is a ripple adder that takes two 8-bit inputs and outputs a 9-bit value which is the sum of those 8-bit values with the ninth bit being the sign extension. It also has a sub_en signal which will be high when we want to subtract instead of add.
Purpose: The purpose of this module is to add the values of A and the Switches every time the LSB of B is 1 and store the output in X and A.



- **D_ff:**
  Module: D_ff.sv

Inputs: Din, Clk, reset, load
Outputs: Q
Description: This is a one bit D Flip-flop with synchronous reset and load. When load is high the Q will be equal to Din.
Purpose: The purpose of this module is to create the X register that is in charged of the sign extension of register A

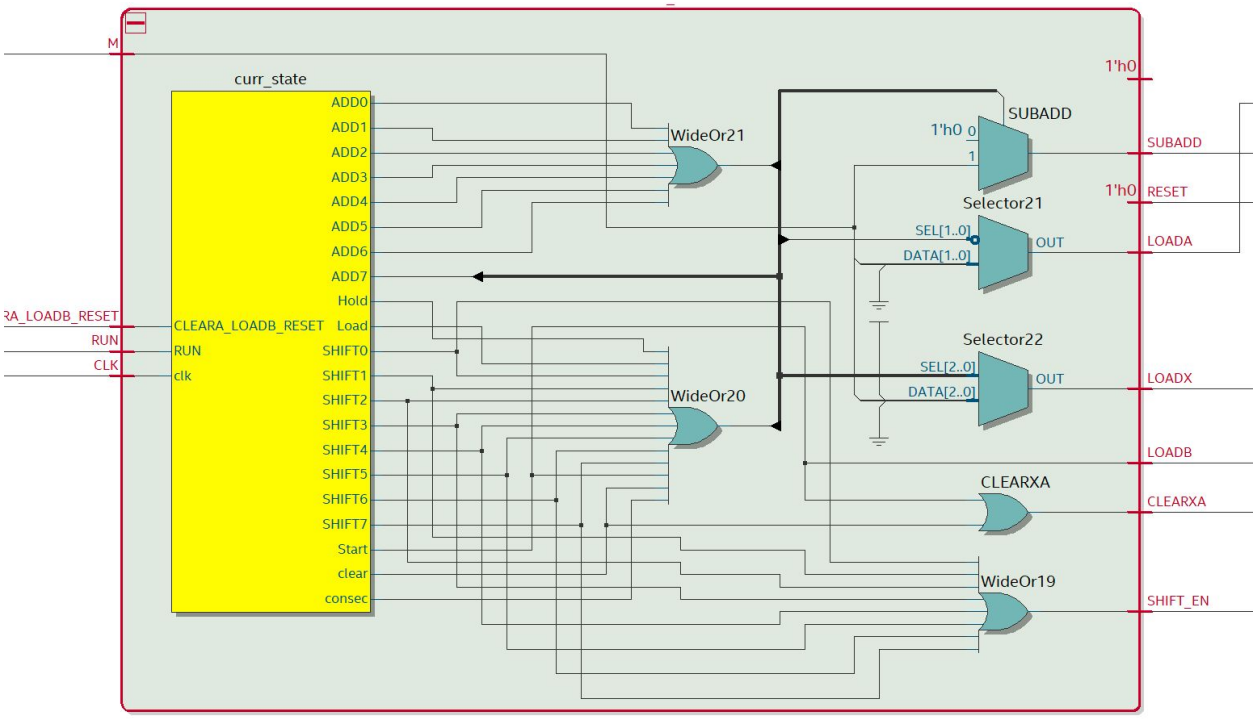


- **multiplier:**
  Module: multiplier.sv
  Inputs: [7:0] S, Clk, Run, Clear_LoadB_Reset
  Outputs: [6:0] AhexL, [6:0] AhexU, [6:0] BhexL, [6:0] BhexU, X, [7:0] Aval, [7:0] Bval
  Description: This is the multiplier it takes all the modules from before and combines them so that we actually perform the multiplication. The outputs will output the values of A and B to the HEX displays and to the registers A, B and X
  Purpose: The purpose of this module is to perform the multiplication with the inputs given from the control unit

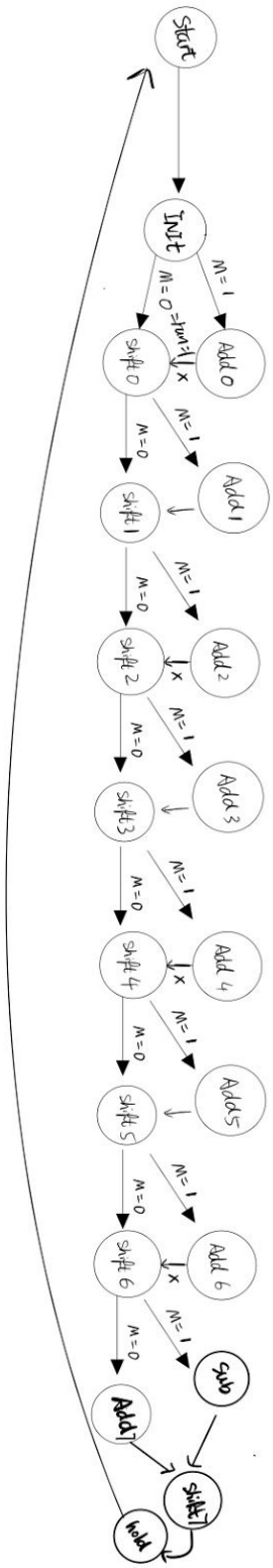
- **control:**
  Module: control.sv
  Inputs: RUN, CLEARA_LOADB_RESET, CLK, M
  Outputs: SHIFT_EN, SUBADD, CLEARXA, CLEARB, LOADB, LOADA, LOADX, RESET
  Description: This is the control unit and it takes in 4 inputs RUN that makes the circuit advance to the next state, CLEARA_LOADB_RESET which clears

registers A and X and it sends the signal to load register B from the switches, and the M will be taken in to produce a signal to add subtract or sut shift.

Purpose: The purpose of this module is to create a finite state machine that will control the states of our circuit.



d. State Diagram for Control Unit

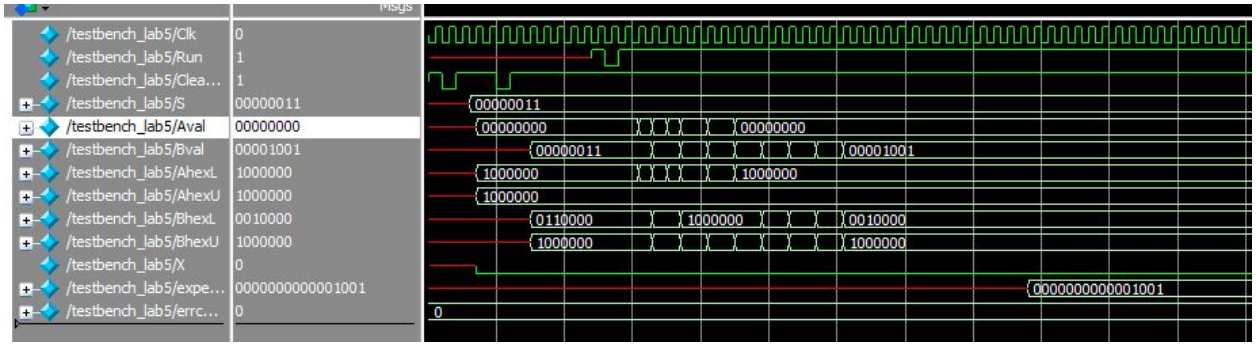


Annotated pre-lab Simulation Waveforms

a. Show 4 operations where operands have signs (+*+),(+*-),(-*+) and (-*-)f

(+*+) waveform screenshot

```
initial begin : TEST_VECTORS

//positive * posotvie


     Clear_LoadB_Reset = 1;
#2   Clear_LoadB_Reset =0;
#2   Clear_LoadB_Reset = 1;
#2   S = 8'h03;
#2   Clear_LoadB_Reset = 1;
#2   Clear_LoadB_Reset = 0;
#2   Clear_LoadB_Reset = 1;
#10   S = 8'h03;
#2   Run = 1;
#2   Run = 0;
#2   Run = 1;
#60  expectedValue = (8'h03*8'h03 );
        if ({Aval, Bval} != expectedValue)
              errcount++;



   end
endmodule
```

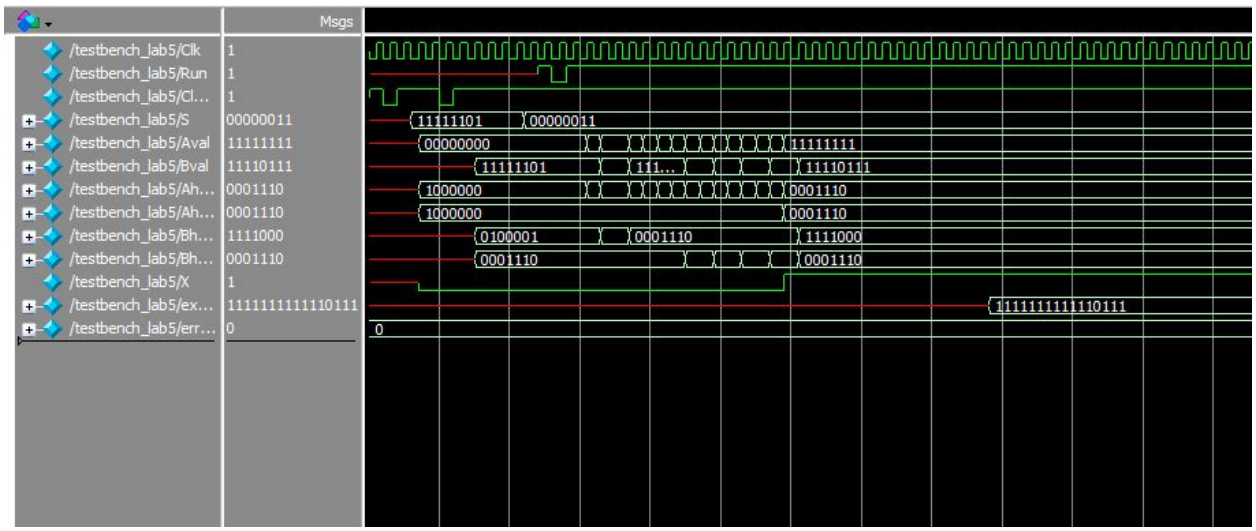


B is loaded with 8’h03, and A is loaded with 8’h03, and the result is equal to 16’h0009. The error count also is equal to 0, therefore, no errors detected.

Negative number X positive number

b. Waveform must have notes that clearly show the operands as well as the result

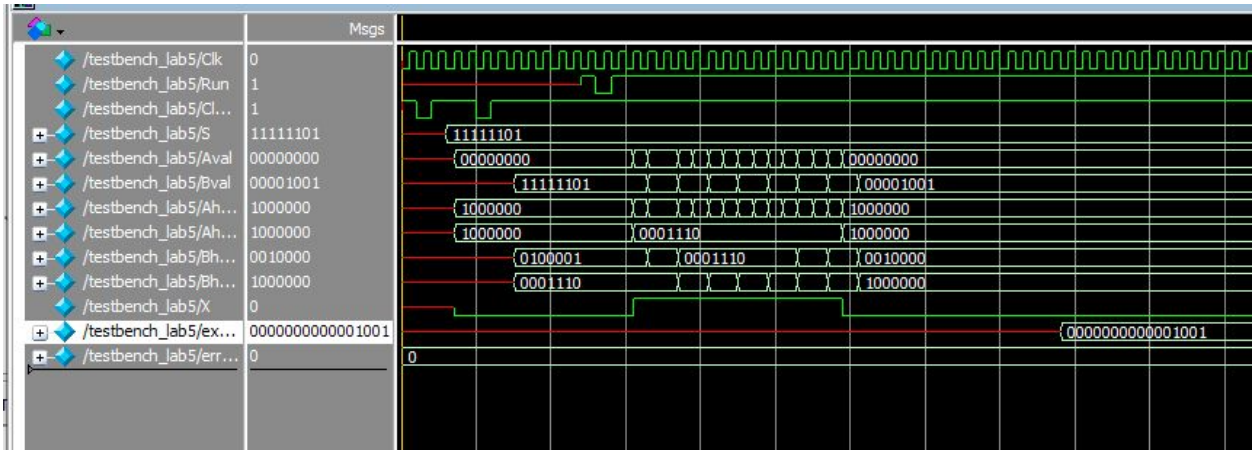
```
      Clear_LoadB_Reset = 1;
#2    Clear_LoadB_Reset =0;
#2    Clear_LoadB_Reset = 1;
#2    S = 8'b11111101;
#2    Clear_LoadB_Reset = 1;
#2    Clear_LoadB_Reset = 0;
#2    Clear_LoadB_Reset = 1;
#10    S = 8'b00000011;
#2    Run = 1;
#2    Run = 0;
#2    Run = 1;
#60   expectedValue = -3*3 ;
       if({Aval, Bval} != expectedValue))
             errcount++;
```



B is loaded with -3, and A is loaded with 3, the result is -9, and there is not error detected.

Negative x Negative number

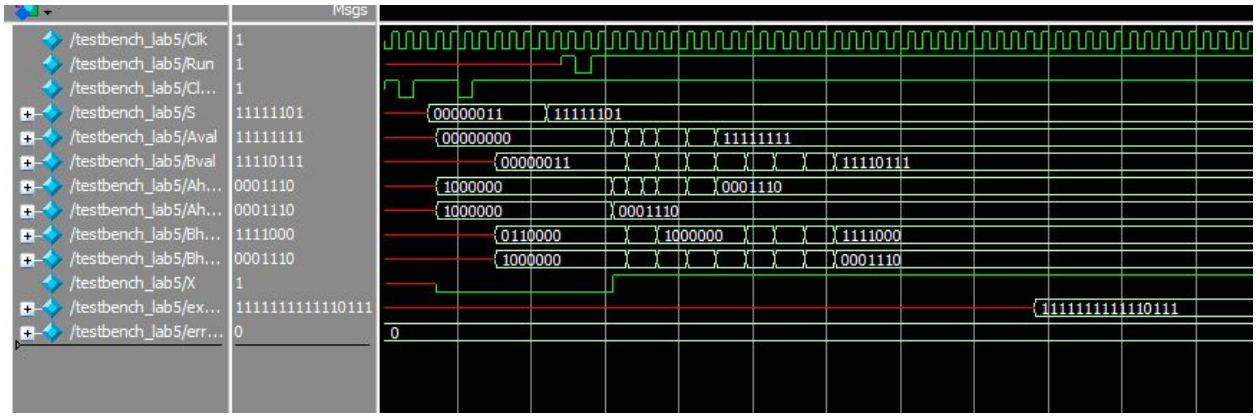
```
        Clear_LoadB_Reset = 1;

#2      Clear_LoadB_Reset =0;

#2      Clear_LoadB_Reset = 1;

#2      S = 8'b11111101;

#2      Clear_LoadB_Reset = 1;

#2      Clear_LoadB_Reset = 0;
#2      Clear_LoadB_Reset = 1;

#10      S = 8'b11111101;

#2      Run = 1;

#2      Run = 0;

#2      Run = 1;

#60     expectedValue = -3*-3 ;
         if({Aval, Bval} != expectedValue)
               errcount++;
```



B is loaded with -3, and A is loaded with -3, the result is 9, and there is not error detected.

Positive X Negative

```
    Clear_LoadB_Reset = 1;
#2  Clear_LoadB_Reset =0;
#2  Clear_LoadB_Reset = 1;
#2  S = 8'b00000011;
#2  Clear_LoadB_Reset = 1;
#2  Clear_LoadB_Reset = 0;
#2  Clear_LoadB_Reset = 1;
#10  S = 8'b11111101;
#2  Run = 1;
#2  Run = 0;
#2  Run = 1;
#60 expectedValue = -3*3 ;
     if({Aval, Bval} != expectedValue)
          errcount++;
```



B is loaded with 3, and A is loaded with -3, the result is -9, and there is not error detected.

Post-Lab Questions

a. Fill out the table with design statistics

LUT	95
DSP	0
Memory(BRAM)	0
Flip-Flop	48

Frequency	222.47MHz
Static Power	89.94 mW
Dynamic Power	0mW
Total Power	98.66mW

b. Additional Questions;
   i. What is the purpose of the Xregister. When does the X register get set/cleared?
   The purpose of the X register was to sign extend the value of register A. X gets cleared to 0 every time the most significant bit of A is 0 and it gets set to 1 every time the most significant bit of A is 1.

   ii. What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?
   The limitations of continuous multiplications will be that the numbers will keep getting bigger and bigger so there will be a point where the 16 bits will not be enough to hold the value and that will cause problems. If we multiply numbers that their multiplication will result in a number that cannot be represented in 16 bits.

   iii. What are the advantages and disadvantages of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?
   The advantages are that we do not need extra registers because if we would do the paper and pencil multiplication we would need a lot more 16-bit registers and for this one we only need 2 8-bit registers which reduces the hardware by a lot. The disadvantage is that with paper and pencil multiplication we can have a lot of registers and multiply everything parallelly and add it then it will be way faster than taking multiple states as in the one for this lab.

Conclusion:

a. Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it.

   Our design work as expected was able to multiply with different and similar signs and it worked fine overall. We did have a problem with the simulation and it was that we were not getting the correct output for (-*-) the way to fix it was to go state by state to see what our problem was.

b. Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester ?

   Overall the lab was good but our team struggled with knowing what the X register actually did so maybe in future labs there could be more description about this one value. If we had more description of it we would’ve not wasted much time trying to learn how it worked.