ECE 385

Spring 2020

Experiment 5

An 8-bit Multiplier in SystemVerilog

Michael Faitz and Zohair Ahmed

Section ABJ: Friday 2:00-4:50

Yuming Wu and Lian Yu

Introduction

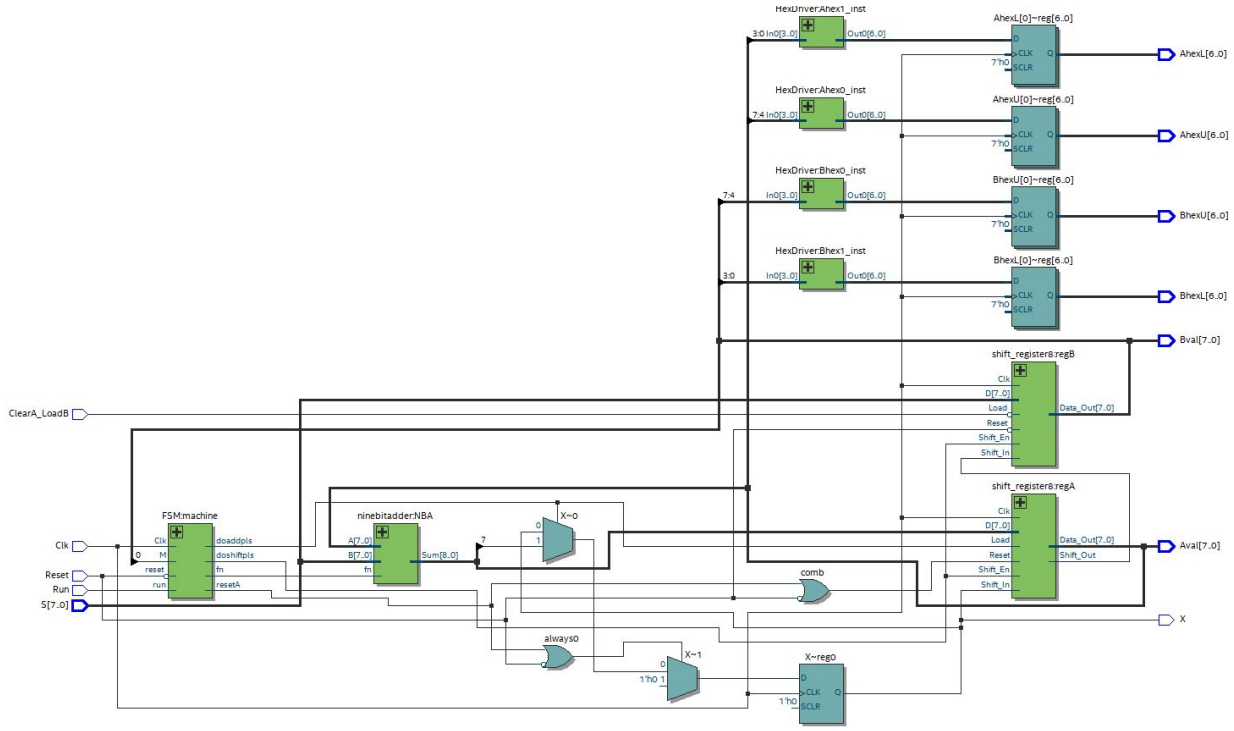
a. The basic function of the multiplier is to take 2 8-bit numbers in two's complement, and multiply them together. The output is displayed through two 8-bit registers and a flip-flop to display the product's sign.

Pre-lab Question

Function	X	A	B	M	Comments for the next step
Clear A, LoadB	0	0000 0000	0000 0111	1	Since M = 1, multiplicand will be added to A.
Add	1	1100 0101	0000 0111	1	Shift XAB by one bit after add complete
Shift	1	1110 0010	1000 0011	1	Since M = 1, multiplicand will be added to A.
Add	1	1010 0111	1000 0011	1	Shift XAB by one bit after add complete
Shift	1	1101 0011	1100 0001	1	Since M = 1, multiplicand will be added to A.
Add	1	1001 1000	1100 0001	1	Shift XAB by one bit after add complete
Shift	1	1100 1100	0110 0000	0	Since M = 0, do not add S to A. Shift XAB
Shift	1	1110 0110	0011 0000	0	Since M = 0, do not add S to A. Shift XAB
Shift	1	1111 0011	0001 1000	0	Since M = 0, do not add S to A. Shift XAB
Shift	1	1111 1001	1000 1100	0	Since M = 0, do not add S to A. Shift XAB
Shift	1	1111 1100	1100 0110	0	Since M = 0, do not add S to A. Shift XAB
Shift	1	1111 1110	0110 0011	1	8th shift done. Stop. 16-bit product in AB.

Written description and diagrams of multiplier circuit

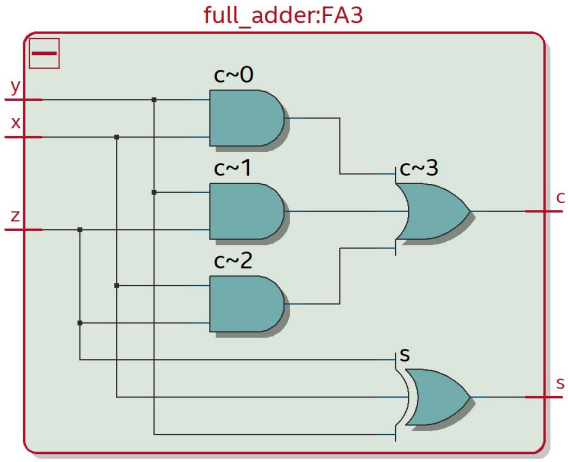
a. Summary of operation
   i. The multiplier circuit stores the multiplicand in register B, and the multiplier in switches. The registers are shifted to account for powers of 2, as done with powers of 10 in decimal multiplication. When the LSB of the shifted multiplicand is 0, we add nothing. When it is 1, we add the switches to A. We then shift XAB right to prepare for the next bit. When the register prepares to add for the 8th time, it checks the LSB (originally the MSB of the multiplicand) and if it is 1, we subtract the value in the switches from A. This accounts for multiplication of negative numbers. After the 8th shift, we have our product in XAB.
b. Top Level Block Diagram



c. Written Description of .sv modules

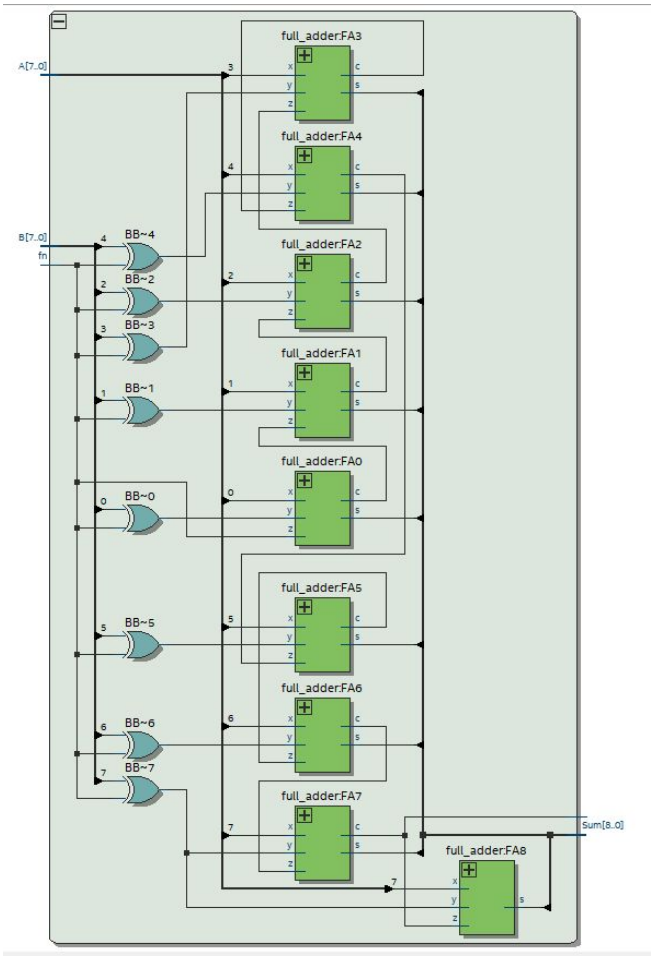
Module: full_adder
Inputs: x,y,z
Outputs: s,c
Description: Adds input binary signals x, y, and z and provides an output of their sum in binary and the carryout bit.
Purpose: Serves as the basic basic building block for larger adders which will be needed within the 9-bit adder.

RTL of full_adder



Module: ninebitadder
Inputs: A[7:0] , B[7:0], fn
Outputs: CO, Sum[8:0]
Description: Takes in two eight bit binary signals and depending on the fn bit decides whether to add or subtract the second “B” signal from signal “A.” If fn = 1 then the adder will subtract B from A. The result of this is then output as a bit extended sum. All of these operations are performed using 2’s complement.
Purpose: This module acts as the workhorse of the FSM which will use this module when it is told to add or subtract from the signal in the register. Serves as both the adder and the subtracting unit in order to account for the last (8th) adding operation needing to be a subtraction as discussed in the lab manual and lecture.

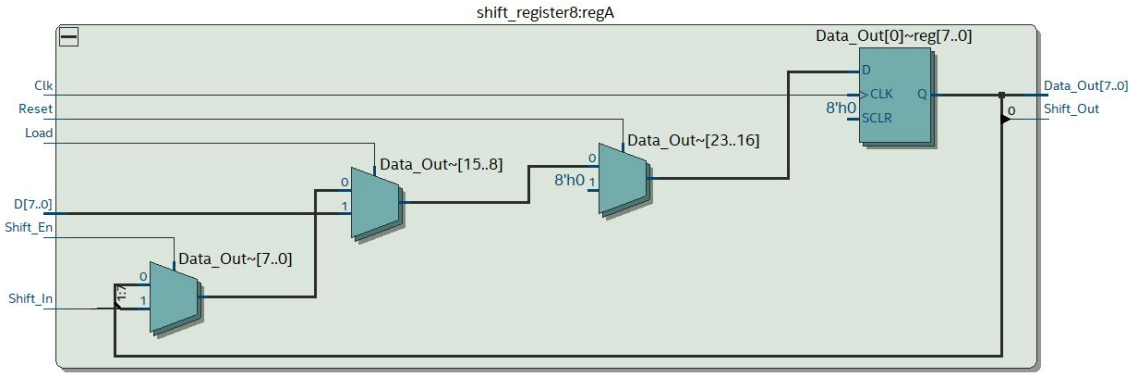
RTL of ninetbitadder



Module: shift_register8
Inputs: Clk, Reset, Shift_In, Shift_In, Load, Shift_En, D[7:0]
Outputs: Shift_Out, Data_Out[7:0]
Description: Serves as a basic 8 bit data storage unit that works on positive edge cycles of the input clock which can reset data stored inside to 00000000 when reset is high, can

store input data D[7:0] when Load is high, and right shifts the data inside to the right when the Shift_En is high. When the rightmost bit is shifted out of the register it is output as Shift_Out and the leftmost bit after the shift is replaced by the Shift_In input bit. Data_Out serves as an ability to see the data inside of the register and make various computations based on the data.
Purpose: Serves as storage registers for both A and B which will hold the final multiplication result and will be used to multiply the switches with the data that can be stored into register B. When the FSM calls for an addition operation the data in A will be updated as described in the lab notebook and on the next clock cycle the data in both registers A and B will be right shifted. Register A will also take in the shift bit found from the flip-flop X.

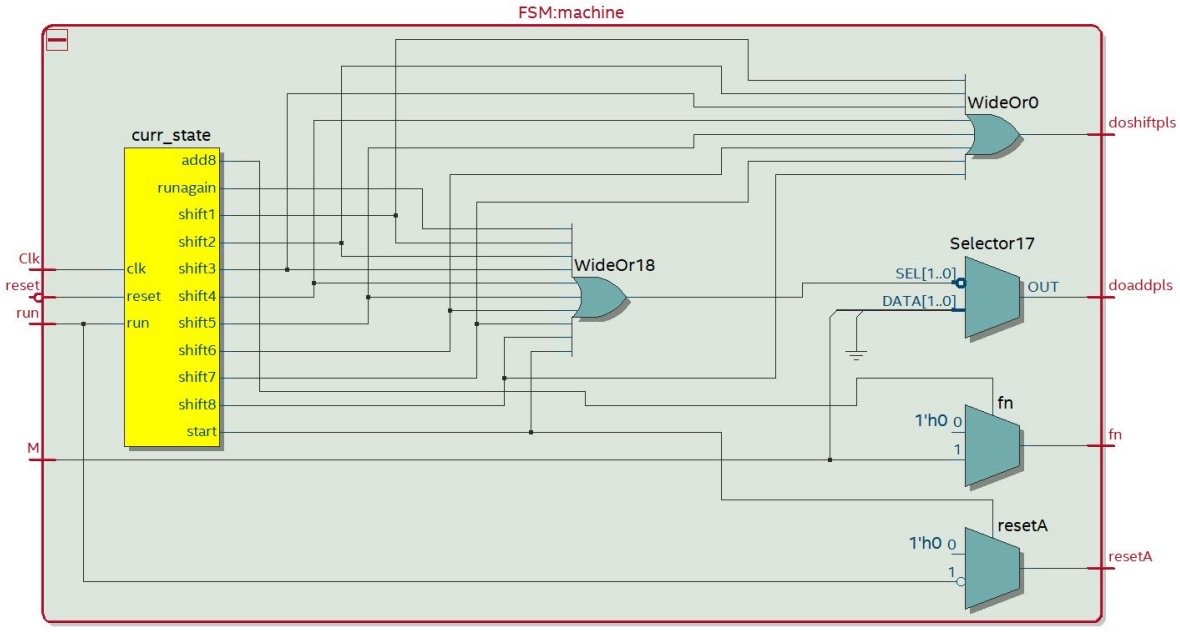
RTL of shift_register8



Module: FSM
Inputs: Clk, run, M, reset
Outputs: doshiftpls, fn, resetA, doaddpls
Description: Creates various states and available transitions for the states which are updated constantly in the top level module lab5. There is a synchronizer in the beginning of the module that allows the updating or beginning of the current state to be on the positive edge of the clock cycle. The general state diagram layout first starts in the start state on the first clock cycle. This state serves to initialize fn, resetA, doshiftpls, and doaddpls to zero effectively resetting the machine from previous cycles when run is hit. The following several states are a combination of an add state to a shift state. In each add state the shift output is brought to don't shift and the FSM checks the rightmost bit in register B. If the bit is high, then the FSM tells the other modules to add and if the bit is zero, then it does nothing other than transition to the shift state. In the shift state the add bit is updated again to not add and the shift bit is enabled thus telling the registers to shift. It then transitions to the next add state. When the FSM reaches the 8th cycle of the add state to shift state it also checks the input fn bit and if the bit is high then the FSM tells the adder to subtract the 2's complement of the data in the switches.

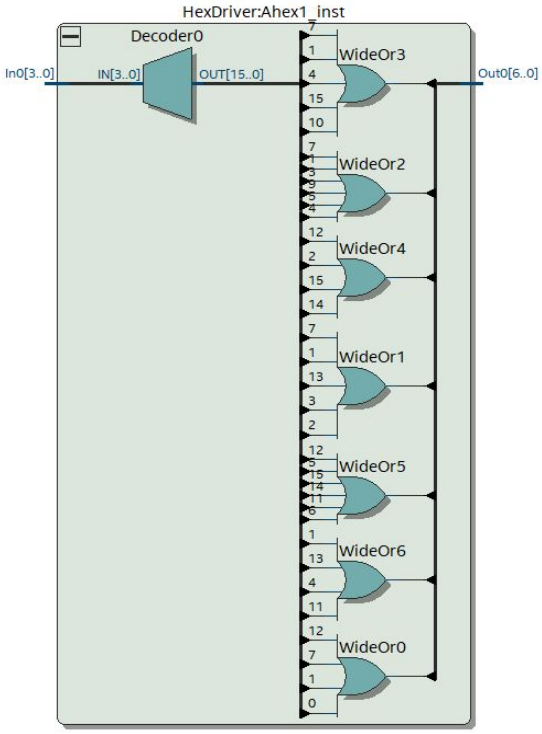
After this it shifts like before and encounters the runagain state. Within this state when run is not being pressed it transitions back to the start state to assure the device does not multiply very quickly (nanoseconds).
Purpose: The purpose of the FSM is to hold the states for the other modules to use within the system. It also provides the logic to control the flow and path of the operations performed. Within this hex multiplier, the FSM serves to have 8 add and shift states as well as a hold and start state. The FSM will then feed its outputs to the top level module which updates the use of the other modules depending on which state that is currently accessed.

RTL of FSM

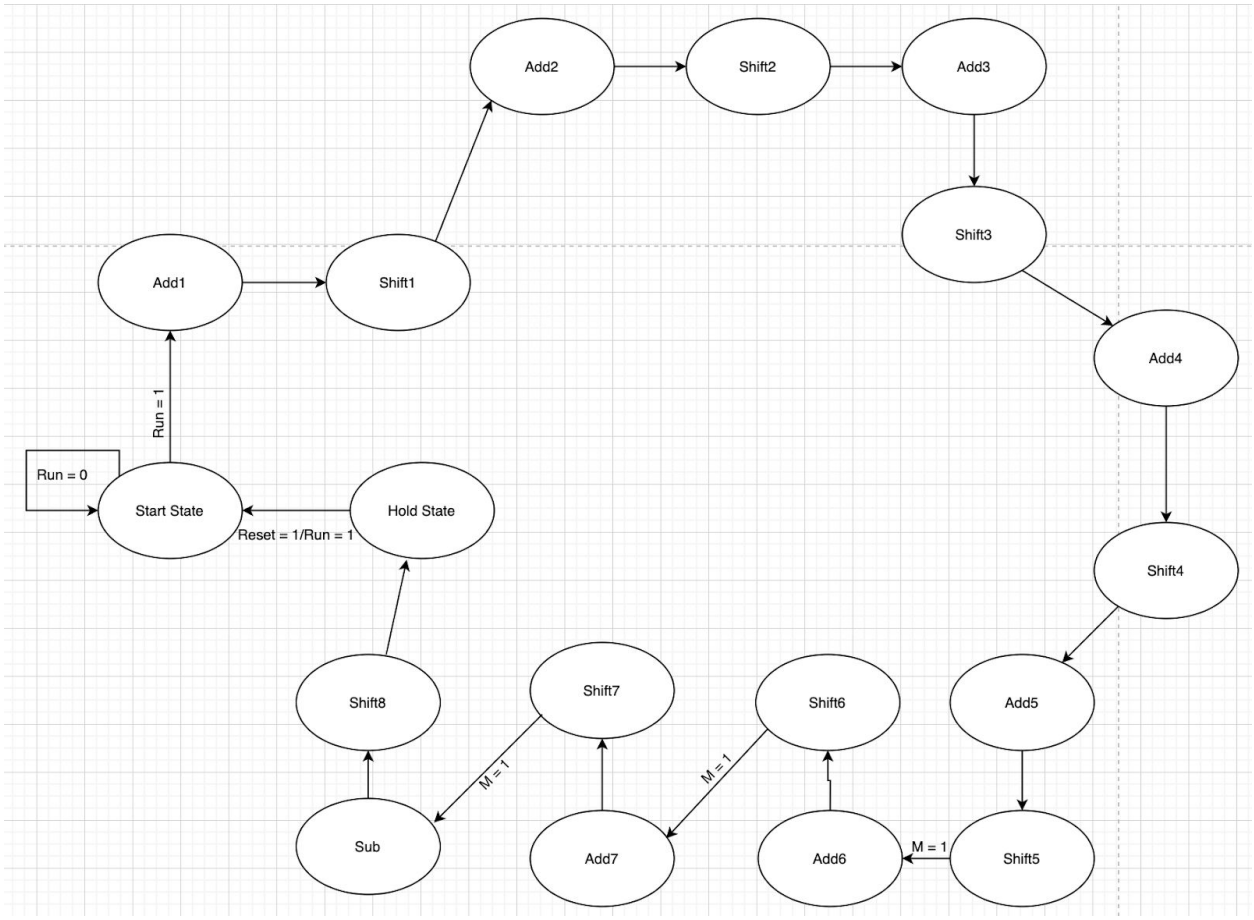


Module: HexDriver
Inputs: In0[3:0]
Outputs: Out0[7:0]
Description: transfers a 4 bit signal to represent a hex display on the FPGA. The 4 bits in are needed in order to have designations for hex values 0 to F.
Purpose: Serves as a display conversion used strictly for the board operation and display of the result of the multiplication stored in registers A and B.

RTL of HexDriver



Module: lab5
Inputs: S[7:0],Clk, Reset, ClearA_loadB, Run
Outputs: X, Aval[7:0], Bval[7:0], AhexU[6:0], AhexL[6:0], BhexU[6:0], BhexL[6:0]
Description: Serves as the top level module of these set of modules. This module declares the internal registers, contains the X flip-flop triggered on the positive edge cycle of the clock, checks outputs from the FSM and uses these variables within the calls of the shift_register8 modules and the ninebitadder. After calling all of these functions and compiling their data it then also instantiates the HexDriver module for the display of the data to take place on the FPGA.
Purpose: To act as the executioner of all the other modules mentioned above. Here this is used in order to transmute variables from the FSM to the other modules in order to give meaning and operation to the various states present within the FSM. Establishing the X flip-flop here is also critical as it is a vital part of the multiplication operation.

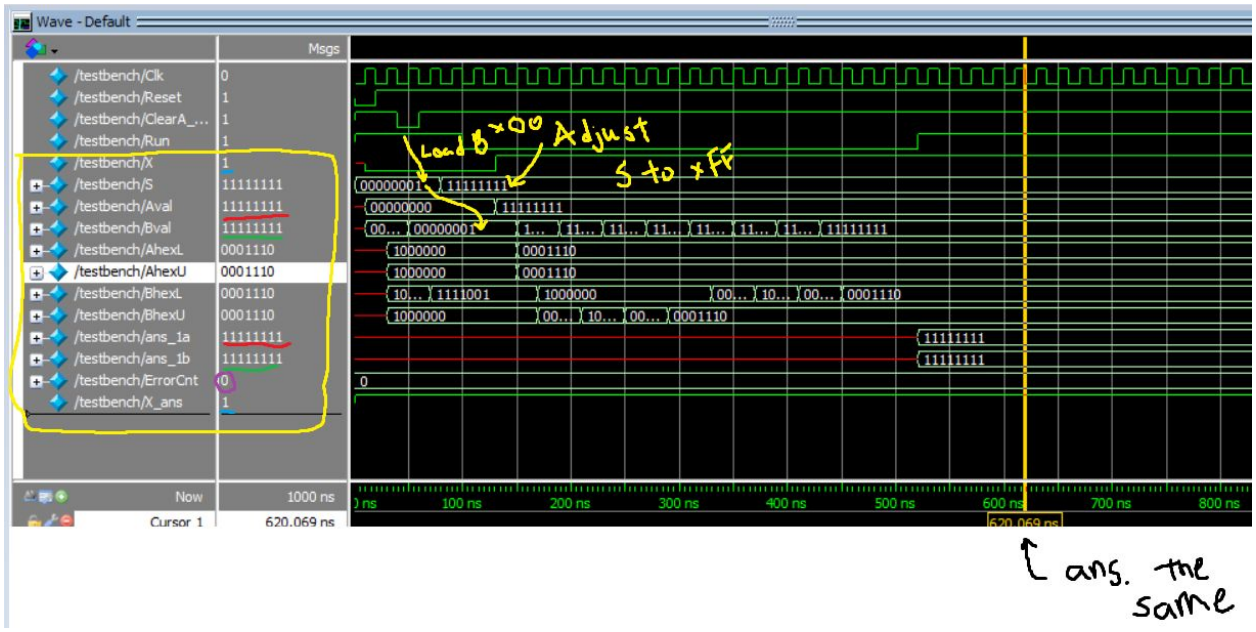


Annotated pre-lab simulation waveforms

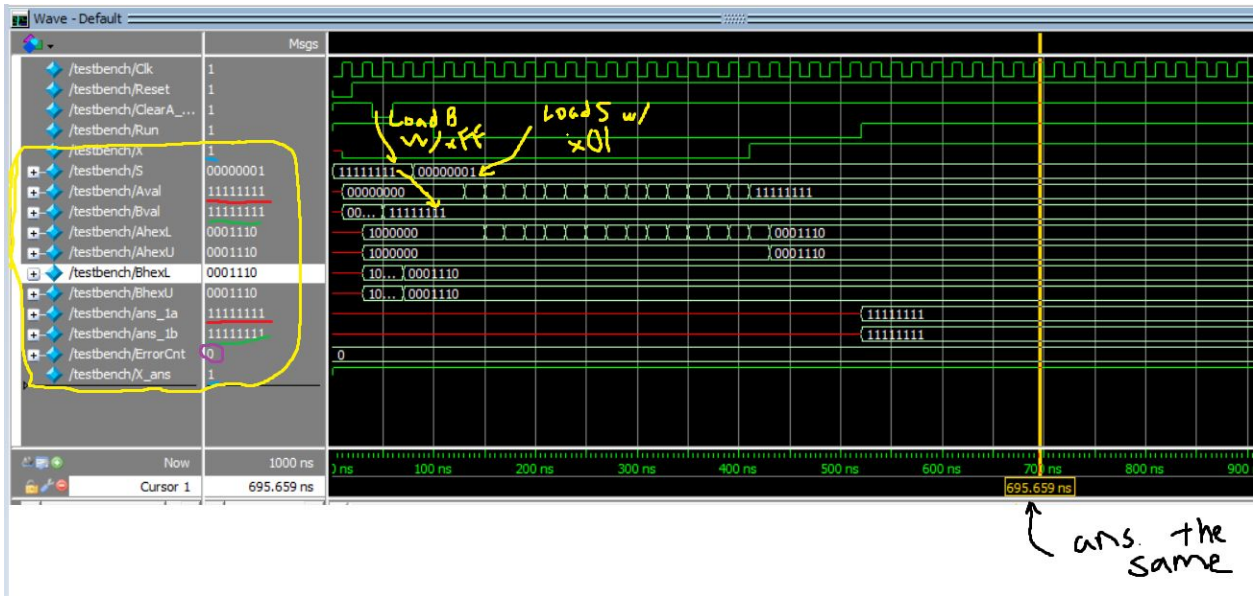
a. Must show 4 operations where operands have signed (+*+),(+*-),(-*+), (-*-)

1*1 Waveform

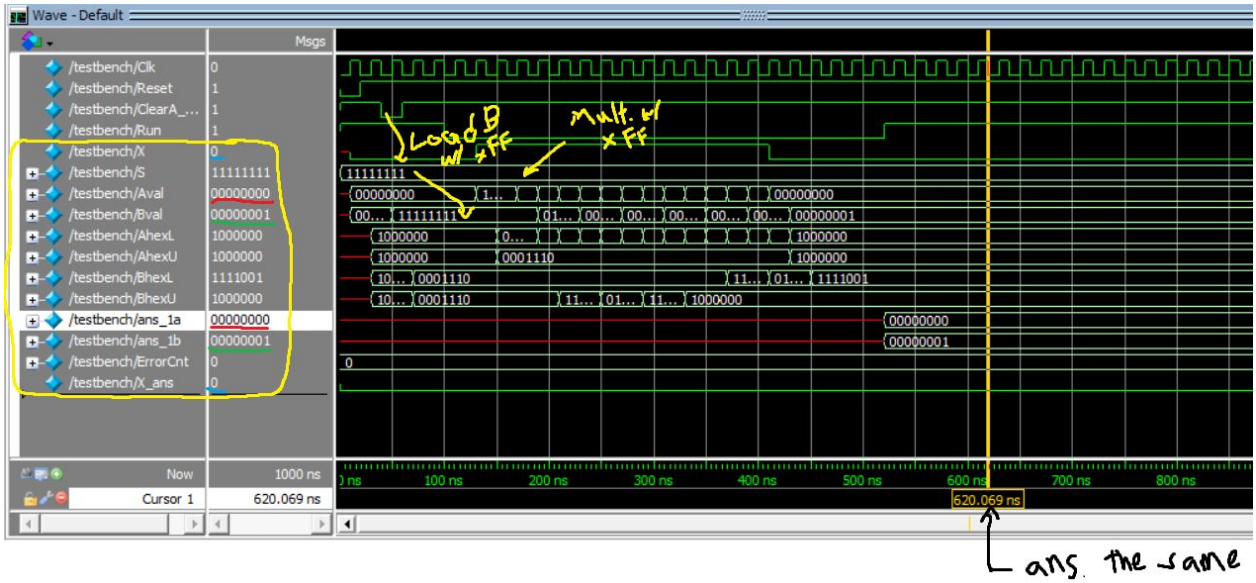
1*-1 Waveform



-1*1 Waveform



-1*-1 Waveform



Answers to post-lab questions

a. Answer all the post-lab questions. As usual, they may be in their own section or dispersed into the appropriate sections in the rest of the report.

i. Design resources and statistics table

Hex Multiplier	
LUT	88
DSP	0
Memory(BRAM)	0
Flip-Flop	63
Frequency	67.78 MHz
Static Power	98.52 mW
Dynamic Power	2.71 mW
Total Power	148.79mW

To change these design statistics and lower the number of gates used we could use NAND gates instead of XOR gates within the 8 bit adder.

ii. Post-lab questions

1. The purpose of the X register is to maintain the sign of the product, such that it does not get lost in overflow during an intermediate step.
2. The limitations of continuous multiplications is the possibility of overflow. These circumstances occur when the product exceeds the capability of 16 bits.
3. The advantages of the implemented multiplication method over pencil-and-paper is the ability to fit every step within 17 bits, requiring less memory. Additionally, the idea of shifting to account for powers of 2 after the first shift allows the multiplicand to stay in B until shifted out. Lastly,

using the LSB in B to determine whether to add the switches or not allows for easy debugging.

Conclusion

a. Our design worked in most cases. The test case of 10001100 * 10001100 did not work, and while trying to fix it, we started to run into issues with (-*-). A way to fix it would be to use the MSB of the output of the adder and the switches instead of the MSB of A.
b. The test case of 10001100 * 10001100 was revealed to us before our actual demo, I just wish it was made clear why this was a test case during our demo or beforehand. We had no idea what we did wrong before or after we missed it in our demo, even if it was simply knowing edge cases or (-*-) would have helped us learn more from this. Everything else about this assignment was alright.