ECE 385

Spring 2020

Experiment # 5

An 8-Bit Multiplier in SystemVerilog

Megh Shah, Ishaan Patel
Lab Section: ABC, Tuesday 11:30 AM
TA: Gene Shiue, David Zhang

Introduction:

The FPGA board has many use cases and is a very versatile piece of hardware. It is essentially programmable to do almost anything that the user defines it to. In this lab, we programmed the FPGA board to emulate an 8-bit multiplier. The board takes in two input data, the multiplier and multiplicand, and using the keys, we can simulate an actual multiplication process. The product of the two data is then displayed using four of the available HEX displays. Overall, by attempting and completing this lab, we believe that we have gained a better understanding and knowledge of how to use not only the FPGA board, but also how to code in SystemVerilog.

Answers to Pre-Lab Questions:

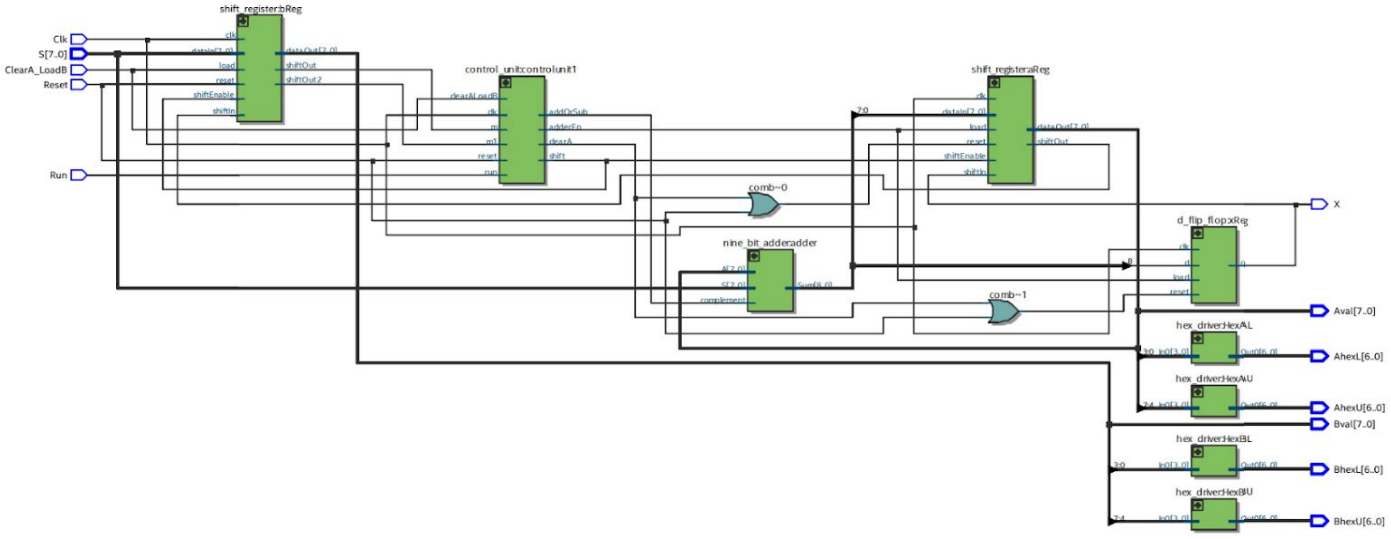
a. Reworked 8-bit multiplication process with S = -59 and B = 7

Function	X	A	B	M	Comments
ClearA_LoadB	0	00000000	00000111	1	Multiplicand →A
ADD	1	11000101	00000111	1	Shift XAB by one bit
SHIFT	1	11100010	10000011	1	Multiplicand →A
ADD	1	10100111	10000011	1	Shift XAB by one bit
SHIFT	1	11010011	11000001	1	Multiplicand →A
ADD	1	10011000	11000001	1	Shift XAB by one bit
SHIFT	1	11001100	01100000	0	Shift XAB by one bit
SHIFT	1	11100110	00110000	0	Shift XAB by one bit
SHIFT	1	11110011	00011000	0	Shift XAB by one bit
SHIFT	1	11111001	10001100	0	Shift XAB by one bit
SHIFT	1	11111100	11000110	0	Shift XAB by one bit
SHIFT	1	11111110	01100011	1	8th Shift Done. Stop. 16-bit product in XAB

Written Description and Diagrams of Multiplier Circuit:

a. Summary of Operation - This circuit has a very simple way of operating. First, the user inputs the desired data for register B which is the multiplicand by setting the switches to represent the 8-bit data. Afterwards, the user presses the ClearA_LoadB switch which loads the data into register B and clears register A which is the multiplier. Next, the user sets the switches to represent the 8-bit data that they want to multiply with the data they set into register B, and presses the execute button. After the execution is completed, the product is represented by both register A and B as it is 16-bits long. The product is displayed in HEX by the HEX display.

b. Top Level Diagram -



c. Written Description of .sv Modules -

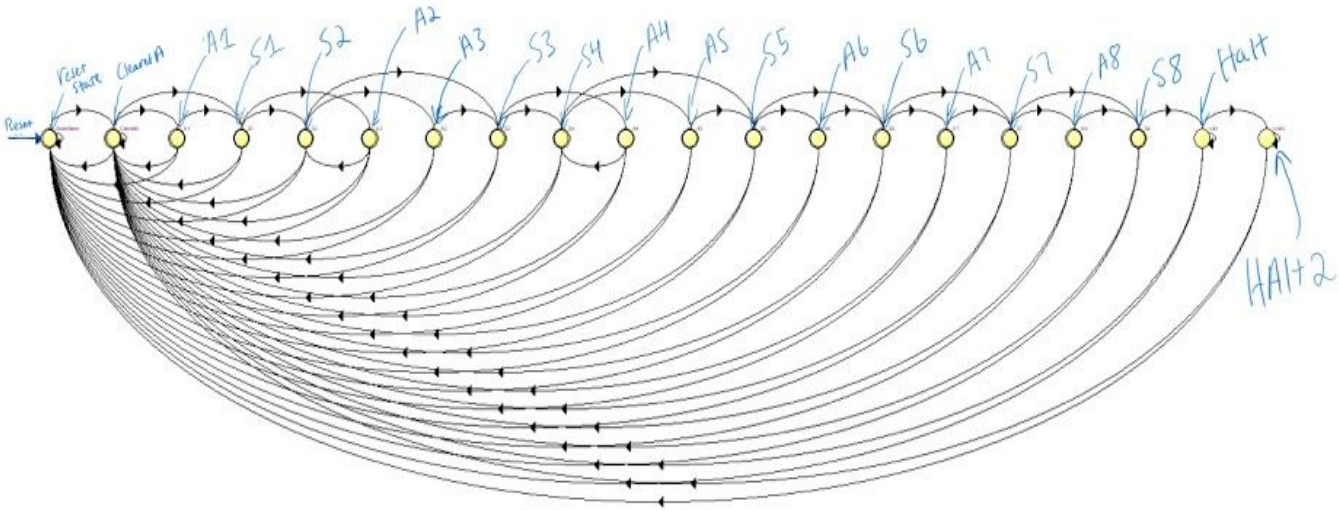
a. **full_adder.sv** - The full adder is a simple adder that takes in two one-bit data and a carry-in bit. It then then adds up the bits and along with the carry-in bit and produces a sum and a carry-out bit.

b. **d_flip_flop.sv** - The D Flip-Flop takes in one bit, and maintains the state or changes it every clock cycle. The Flip-Flop is cleared if a reset signal is passed to it.

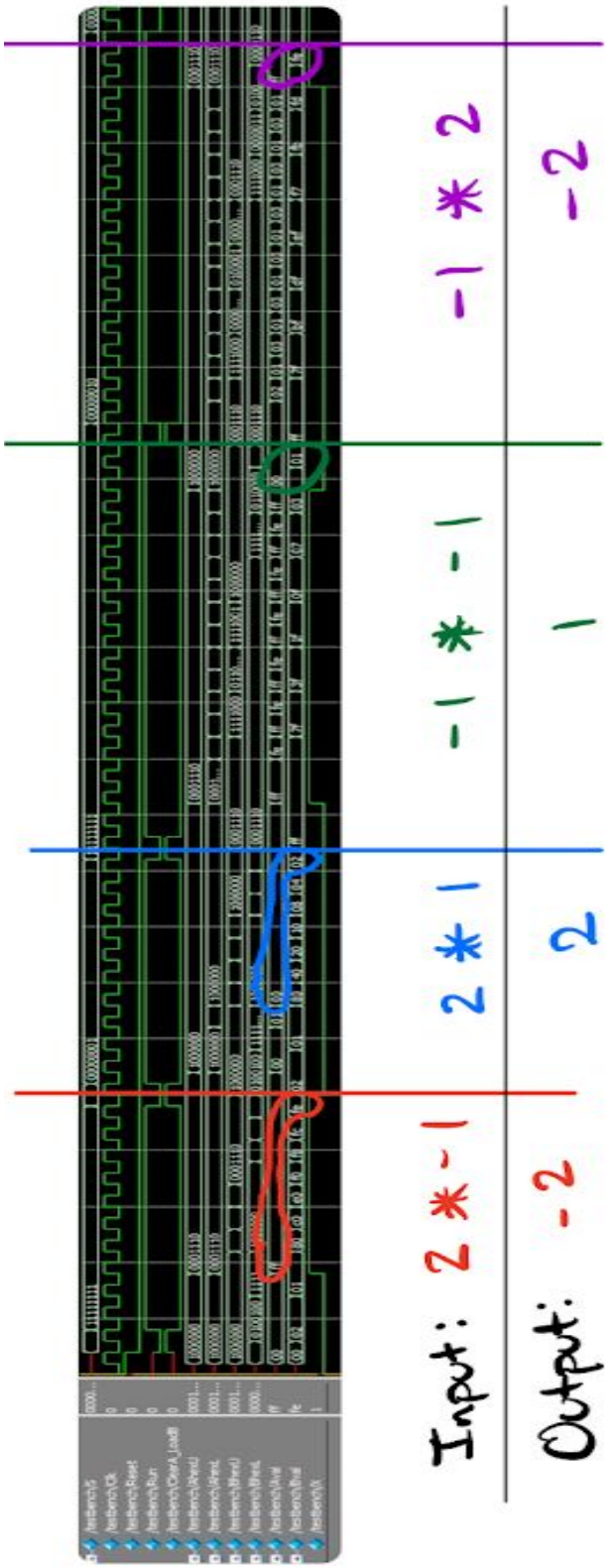
c. **shift_register.sv** - The shift register is an 8-bit register that has the ability to parallel load in data, or take in one shift-bit and produce a shift-out bit. When a reset signal is sent to the register, the contents of it are cleared. The register is used to hold the data for registers A and B.

d. **hex_driver.sv** - The hex drivers are initialized within this file. It sets the default cases for the 7 HEX displays and has the modules to allow use to manipulate the LEDs to display contents of certain contents, in our case the product of the multiplication process.
e. **nine_bit_adder.sv** - The 9-bit adder is a carry-ripple adder comprised of 9 full adders. The inputs for each are the individual bits of registers A and B and the carry-in bit from the previous full adder, if applicable. The last adder takes in the most significant bits of the datas A and B to compute the sign of the product and whether it will be positive or negative.
f. **control_unit.sv** - The control unit's purpose was to simulate the finite state machine by using combinational and sequential logic. It represents the process that the circuit takes everytime we run the multiplication process on it.

d. State Diagram for Control Unit -



Annotated Simulation Waveform:
As you can see above, the waveform simulation performs four types of operations: (+*-), (+*+), (-*-), (-*+). In each of the the four operations, register B is first loaded with the multiplicand which is an 8-bit signed binary data. Then, the switches, S, are set to the multiplier data. After the RUN button is pressed and a certain number of clock cycles have passed, we can see that the entire 16-bit product is stored in XAB.
1st: FFFE 2nd:0002 3rd:0001 4th: FFFE


Input: 2 * -1
2 * 1
-1 * -1
-1 * 2
Output: -2
2
1
-2

Answer to Post-Lab Questions:

1.

LUT	93
DSP	0
Memory (BRAM)	0
Flip-Flop	37
Frequency(Unconstrained)	234.91 Mhz
Static Power	98.50 mW
Dynamic Power	0.00 mW
Total Power	143.00 mW

One way we could increase frequency is by checking M and then immediately going to the add state. Originally, we shifted and then checked M and then went to the ADD state. This is redundant and can be simplified to just checking M right away before shifting and then adding.

2. The purpose of the X register is to maintain the sign of the answer. Our multiplier is a 2's complement multiplier and this requires us to maintain the sign of our bits. When you clearA_LoadB you clear X and when you press continue after a previous run, you also clear X. After the seventh shift is when X has a possibility to be set to a 1, depending on the the the value of M.
   The problem with continuous multiplication is that you clear both A and X. The issue with this is that if you have a value that is greater than 8 bits, part of it would be deleted when you do another multiplication again.
   The advantages of this algorithm is that we have a running sum of the multiplications, thus we just need to store the values in A and B. The problem with the pencil and paper method is that we need to do the multiplications separately and then add them in the end. This requires more storage and isn't as efficient for a computer, due to the excess number of logic elements.

Conclusion:

Due to our initial preparedness, we did not run into too many errors or bugs when designing and coding our circuit other than some basic syntax errors. However, one main issue we ran into was that we did not clear the X bit when we cleared the contents in register A. This led us to having either an off-by-one error or changing the sign of our product. We were able to test our code and find the issue by doing the actual multiplication process by hand. Another issue we ran into was that register A was being cleared before we were able to read the product. We fixed this by adding another state after the HALT state where the registers remained untouched until the RUN button was pressed. For this lab, we believe that most of the instructions and specifications given were clear and helpful. There is not anything that we would recommend be changed for next semester as we were able to follow all instructions clearly and get our circuit to work. Overall, we believe that this lab allowed us to further deepen our understanding regarding the use of Quartus and SystemVerilog.