ECE 385

Fall 2020

Experiment 5

8-Bit Multiplier in System Verilog

Zhicong Fan/Xin Jin
AB2/Online
Yucheng Liang

Introduction:

In this lab, we designed an 8 bits multiplier for two 2's compliment numbers. The system used the add-shift algorithm to do the multiplication, and the system support consecutive operation on the multiplicand and the results.

Pre-Lab Question:

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

A:

S = 1100 0101

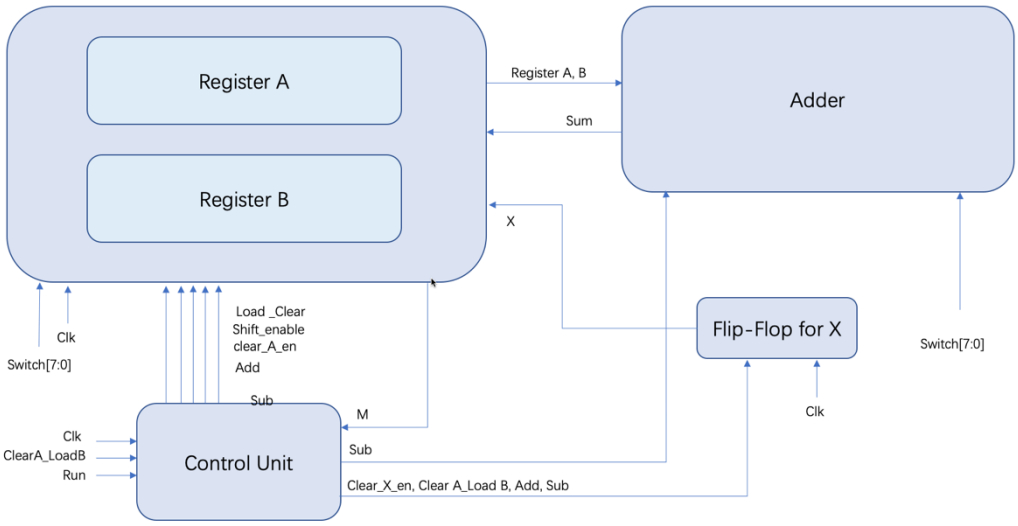
Function	X	A	B	M	Comments
Clear_Load	0	0000 0000	0000 0111	1	Clear register A, load B from switch, then change switch
ADD	1	1100 0101	0000 0111	1	Add S to A since M = 1
SHIFT	1	1110 0010	1000 0011	1	Shift XAB by one bit after ADD complete
ADD	1	1010 0111	1000 0011	1	Add S to A since M = 1
SHIFT	1	1101 0011	1100 0001	1	Shift XAB by one bit after ADD complete
ADD	1	1001 1000	1100 0001	1	Add S to A since M = 1
SHIFT	1	1100 1100	0110 0000	0	Shift XAB by one bit after ADD complete
SHIFT	1	1110 0110	0011 0000	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 0011	0001 1000	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1001	1000 1100	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1100	1100 0110	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1110	0110 0011	1	8th shift done. Stop. 16-bit Product in AB

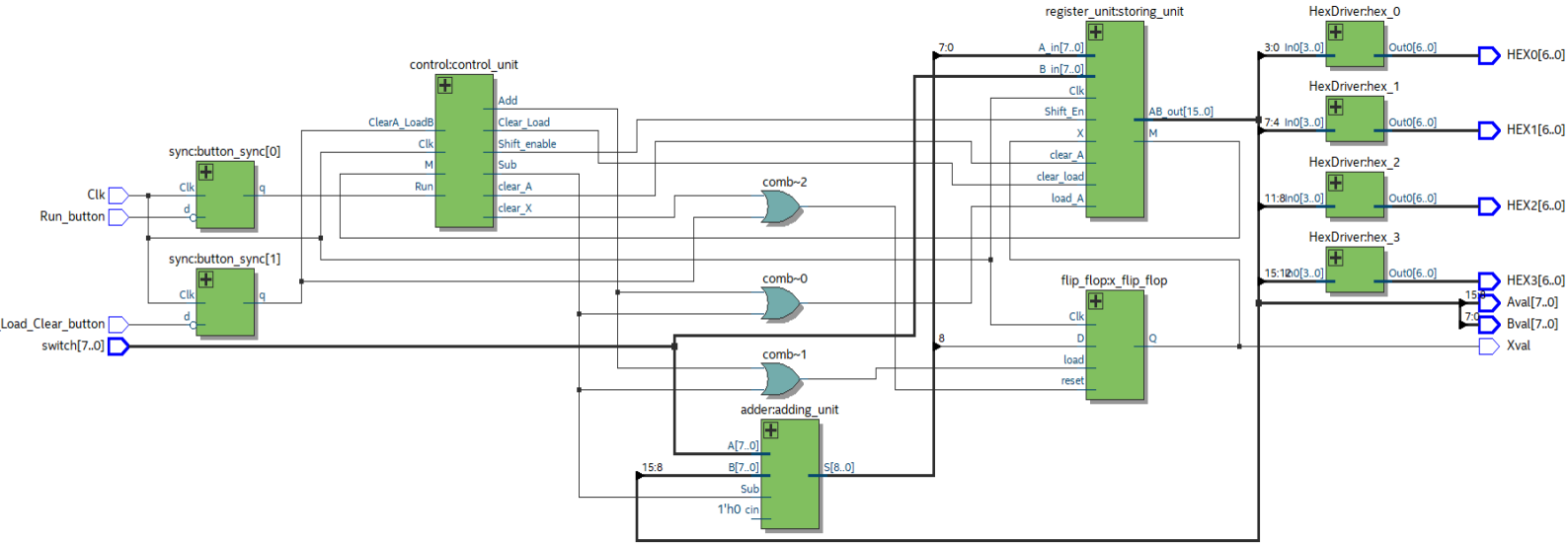
Multiplier Circuit:

Summary of Operation:

To do the multiplication, we would first flip the 8 switches for the multiplicand. Then, we would need to press the Reset_Clear_Load button to clear the multiplier and store the multiplicand. After storing the multiplicand, we would need to flip the switches again for the multiplier. Finally, we could press the Run button to conduct the multiplication operation. The circuit also supported consecutive operation. After each multiplication operation, we could press Run button for another multiplication, which would multiply the multiplier to the result and then stored it back to the result.

Block Diagram:





.sv Modules:

Module: multiplier.sv

Inputs: Clk, Reset_Load_Clear_button, Run_button, [7:0] switch

Outputs: [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [7:0] Aval, [7:0] Bval, Xval

Description: The top level of the System Verilog code. Handles inputs from the FPGA board and distributes then to other modules. When user pressed Reset_Clear_Load button, the module would clear register A and X, and load the content in switched to register B. When user press Run button, the module would conduct the multiplication on switch contents and the contents in register B, then display the result in hex decimal on the board.

Purpose: Computes the multiplication of two input numbers and display the result to the board.

Module: control_unit.sv
Inputs: Clk, ClearA_LoadB, Run, M
Outputs: Shift_enable, Add, Sub, Clear_Load, clear_X, clear_A
Description: This is a control unit is a state machine. It will change the state based on the current state and the user input. Shift_enable would be set if the shift operation is ready. Add would be set if the add operation is ready. Sub would be set if the sub operation is ready. Clear_Load indicates whether the register A should be cleared and register B should be loaded. clear_X indicates whether the flip-flop of X should be cleared. clear_A indicates whether register A should be cleared. The changes are positive edge triggered.
Purpose: Control the input signals and output flags for the other modules from the state machines.

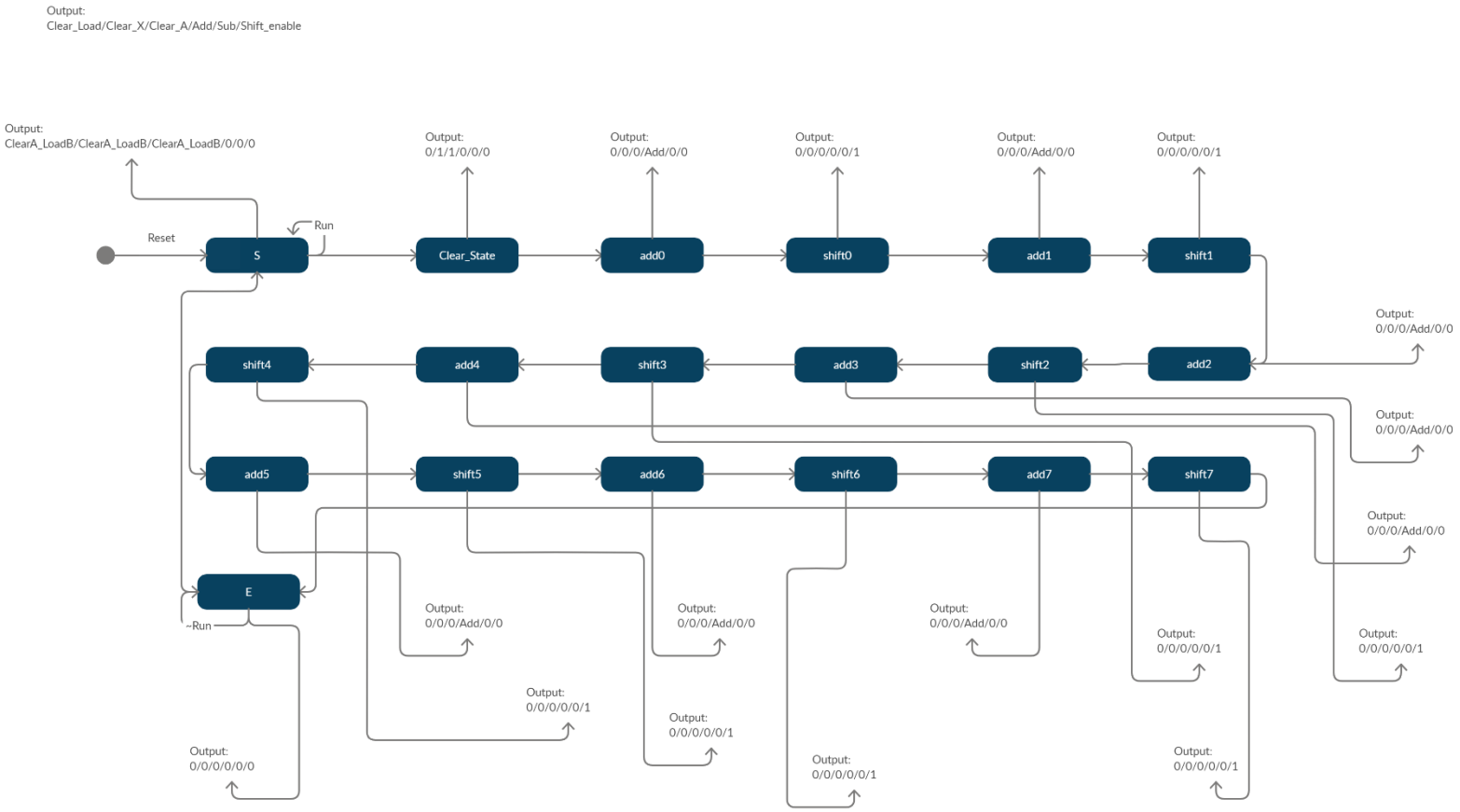
Module: shift_register.sv
Inputs: Clk, clear_load, Shift_En, clear_A, load_A, [7:0] A_in, [7:0] B_in, X
Outputs: M, [15:0] AB_out
Description: This is the 8 bits storing unit of the circuit. It stores the contents of A and B in shift registers. It will either shift(Shift_En), clear (clear_load/clear_A), parallel load(load_A) the registers based on the input signals. The registers are positive edge triggered.
Purpose: This module is used to store the value in register A and register B

Module: flip_flop.sv
Inputs: Clk, D, load, reset
Outputs: Q
Description: Keep track of the X value in the circuit by using a flip flop. It will either update the value based on D when load is high. Or, it will clear the it when reset is high.
Purpose: Store the X in the circuit, which keeps track of the sign of the result.

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: Convert 4 bits input data to a 7 bits board readable data.
Purpose: Convert the result to board readable hex decimal number.

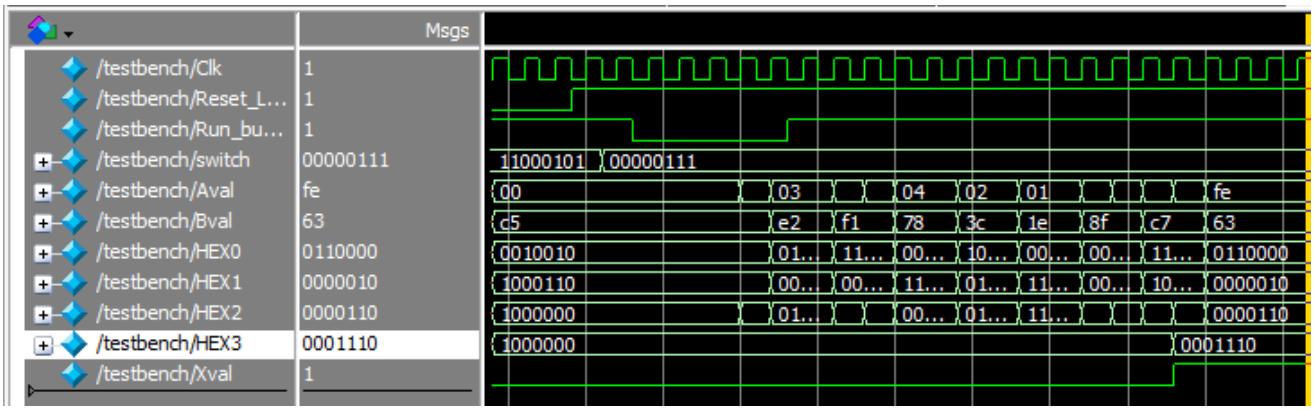
Module: Synchronizer.sv
Inputs: Clk, d
Outputs: q
Description: A positive triggered synchronizer that synchronizes the input d and output as q.
Purpose: synchronize the signals in the circuit.

State Diagram for Control Unit:

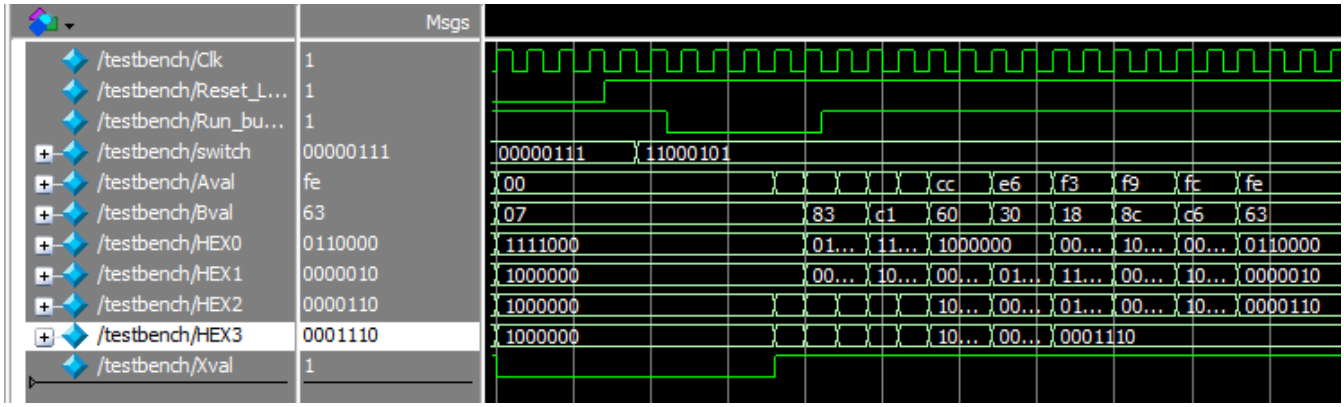


Pre-Lab Simulation Waveforms:

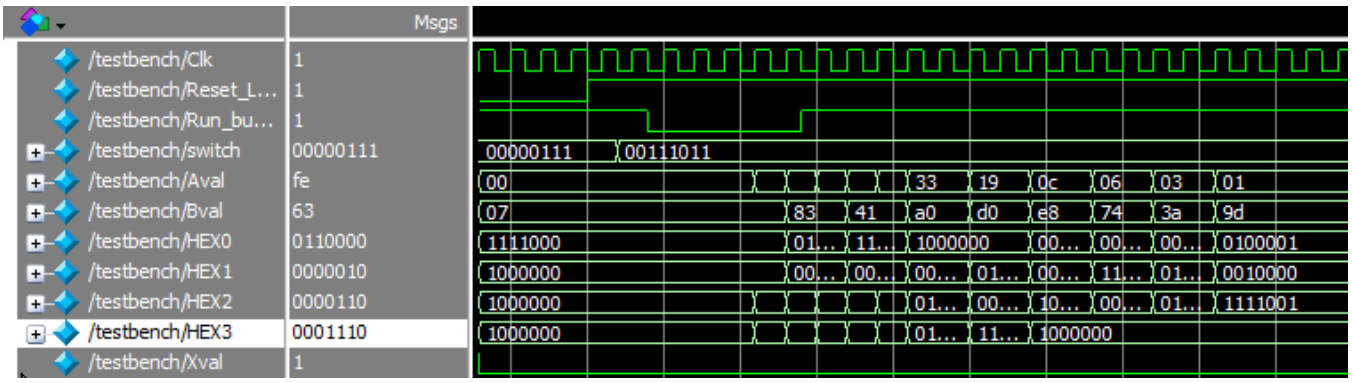
1: -59 * 7 (-*+) and the result should be FE63 shown in both Aval and Bval:



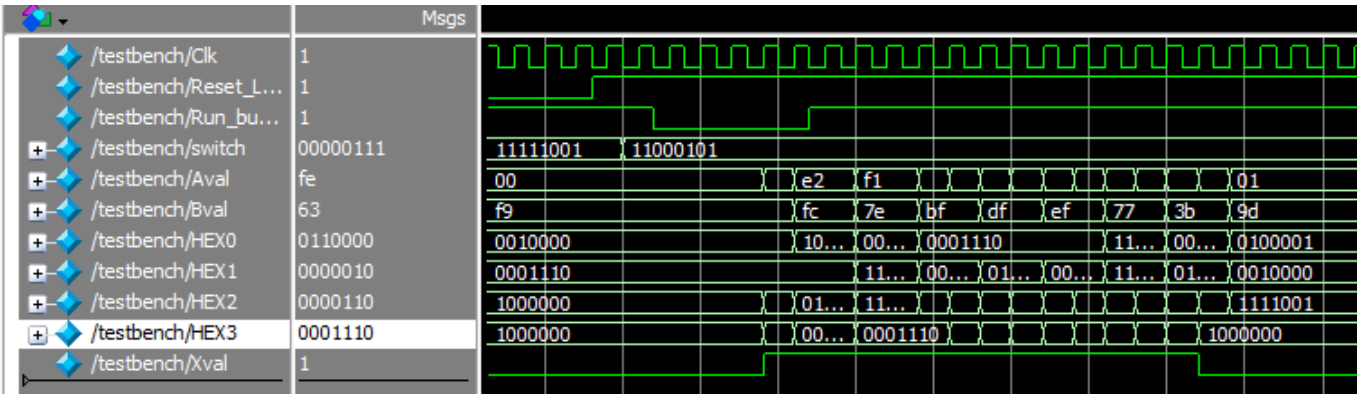
2: 7 * -59 (+*-) and the result should be FE63 shown in both Aval and Bval:



3: 7 * 59 (+*+) and the result should be 019D shown in both Aval and Bval:



4: -7 * -59 (-*-) and the result should be 019D shown in both Aval and Bval:



Post-Lab Question:

Q: Come up with a few ideas on how you might optimize your design to decrease the total gate count and/or to increase maximum frequency by changing your code for the design.

LUT	94
DSP	0
Memory (BRAM)	0
Flip-Flop	38
Frequency	207.43MHz
Static Power	89.94mW
Dynamic Power	0mW
Total Power	98.66mW

A: In this lab we used ripple adder instead of other two adders we learnt in the last lab. So in order to improve the Frequency, we might change the adder we used into either Lookahead Adder or carry select adder. To use less gates, we have to optimize the state diagrams and the logic we used when we implemented the multiplier. We used many redundant terms as signals to pass in.

Q: What is the purpose of the X register. When does the X register get set/cleared?

A: X was used to store the sign of the final result. However, we would also need to update the value of X for each shift/add operation. X would need to be cleared at the start of each Run operation.

Q: What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?

A: The signed 16 bits 2's compliment only supports for numbers in the range of −32,768 to 32,767. If the result of the continuous multiplication got too large, the result would be overflowed.

Q: What are the advantages (and disadvantages?) of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?

A: The advantage of the multiplication algorithm is that no matter how large the numbers are, the algorithm will be able to compute the result in a fixed number of cycles as long as the numbers do not exceed the range. However, if the numbers are relatively small, the pencil and paper would be able to do the computation much faster since the algorithm would need to finish all of the cycles even if the only the first few cycles are useful.

Conclusion:

Bug Encountered:

1. When we were designing test cases in the testbench, we set pressing the Resset_Clear_Load button for 2 clock cycle. We found that the register A and B were not updated properly. However, the functionality was totally normal on the FPGA. After testing, we found that 2 clock cycles were not long enough for the registers to be updated. We solved that problem by setting the button pressing cycle to be 5+ cycles.
2. When we were designing the

Lab Manuel Issues:

Pro:

1. The demo points and the layout and requirements for the lab report are informative.

Con:

1. We found the block diagram in the lab manual was a bit confusing. It seems that the block diagram was not for this version of the lab. For example, there is no reset button in this lab.
2. The add-shift algorithm and the pencil and paper algorithm do not make too much sense. The indentation is messed up for the add-shift algorithm and the pencil and paper method does not really expand the whole process.

Summary:

In this lab, we used System Verilog designed a multiplier which used the add-shift algorithm on FPGA board. The circuit was able to perform a 8 bits multiply 8 bits operation for 2's complement binary numbers. We used the idea of state machine to accomplish the design.