ECE385

Fall 2021

Experiment #4

An 8-Bit Multiplier in SystemVerilog

Haozhe Si

haozhes3

Section ABD

1. Introduction

In this lab, we designed and implemented an 8-bit 2's complement multiplier using SyetemVerilog. This multiplier can take in two 8-bit signed number and perform the signed multiplication, resulting in a 16-bit binary number with an additional sign bit. The multiplier consists of two 8-bit shift registers, one 1-bit shift register, a 9-bit adder, a control unit, eight switches and two buttons. To start a calculation, one should enter one multiplier on the switches, press the Reset_Clear_Load button to load it on one register, then enter the multiplicand and press Run button. The result will be saved in the two resisters, each taking 8 bits of the result.

2. Pre-Lab

We are going to perform an example of calculating multiplication using add-shift algorithm. In this example, we are going to calculate 11000101 x 00000111. As explained before, we will have two registers A and B, we will load the multiplier to B through switches S and put the multiplicand on S after the loading. The result will be in AB registers. We also introduce two variables: X will be the sign extend for A; M will be the least significant bit of B, signifying whether to perform the addition or not.

Specifically in this example, we are going to initialize the variables in the following way: for the registers A = 8'b0000 0000, B = 8'b0000 0111 (decimal 7), and the switches will have S = 11000101 (decimal -59). Consequently, X will be 0, M will be 1 initially.

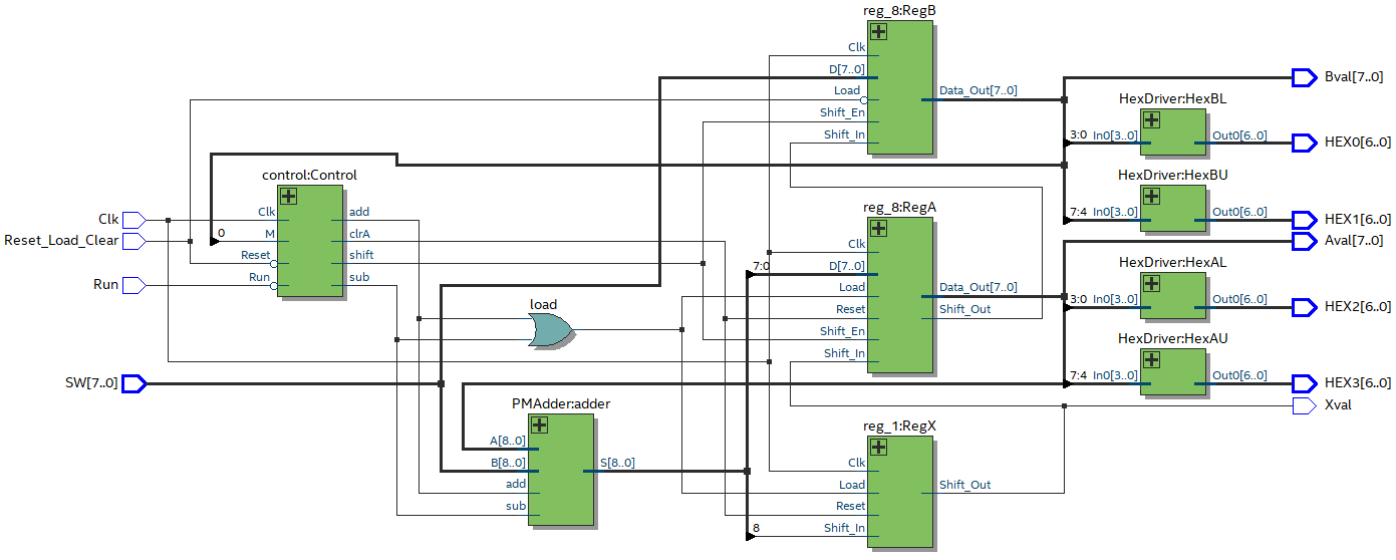
Function	X	A	B	M	Next Step
Clear A Load B Reset	0	0000 0000	*00000111*	1	M=1, add S to A
ADD	1	1100 0101	*00000111*	1	Shift XAB by one bit
SHIFT	1	1110 0010	1 *0000011*	1	M=1, add S to A
ADD	1	1010 0111	1 *0000011*	1	Shift XAB by one bit
SHIFT	1	1101 0011	11 *000001*	1	M=1, add S to A
ADD	1	1001 1000	11 *000001*	1	Shift XAB by one bit
SHIFT	1	1100 1100	011 *00000*	0	M=0, Shift XAB by one bit
SHIFT	1	1110 0110	0011 *0000*	0	M=0, Shift XAB by one bit
SHIFT	1	1111 0011	00011 *000*	0	M=0, Shift XAB by one bit
SHIFT	1	1111 1001	100011 *00*	0	M=0, Shift XAB by one bit
SHIFT	1	1111 1100	1100011 *0*	0	M=0, Shift XAB by one bit
SHIFT	1	1111 0011	01100011	1	8^{th} bit done, stop, result is in AB

3. Summary of Operation

Load: To load the operands, we need to first enter the binary multiplier on the switches. Then, press the Reset_Clear_Load button, which will clear the data in register A and X, and load the data on switch to register B. The button will also reset the state for the control system. The multiplicand will not be loaded into registers directly, but stays on the switch. Each time M equals one, we are going to add the value on switches S to the register A and shift.

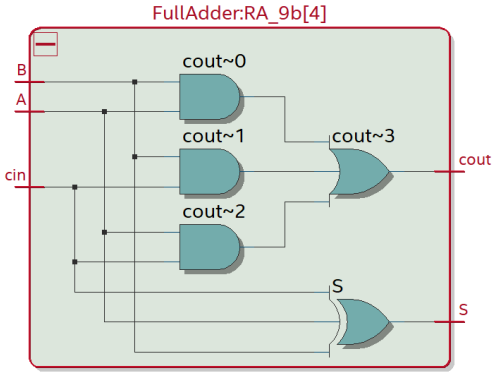
Multiply and Store: Once the multiplicand is entered on the switches, we will press Run to start the multiplication. To perform the shift-add algorithm, each time we will check the LSB, of the data in register B, which we call it M. When M = 1, we will add the value on S to A. Notice that although both A and S are 8-bit binary numbers, we are going to perform a 9-bit addition by sign extending A and S. The 9^{th} bit will be saved into register X and the reset will be saved into A. Then we will shift XAB by one bit to the right: the value in X will be the MSB of A and the LSB of A will be shift into B. The value of X will not change after the shift operation since it controls the sign of the result. When M = 0, we simply shift the XAB registers as mentioned above. When the 8^{th} bit of the multiplier reaches M and M = 1, we need to minus S from A since it is the sign bit. Therefore, to perform the 2's complement subtraction, we negate S, then plus one to S' and add it to A. Once the 8^{th} bit of the multiplier reaches M, and depending on the value of M we perform sub-shift or shift, the multiplication is done. The 16-bit result is saved in register AB, which we can read it out from right to left. The value saved in register X is the sign bit, which is the sign extend of the final result.

4. Top Level Block Diagram

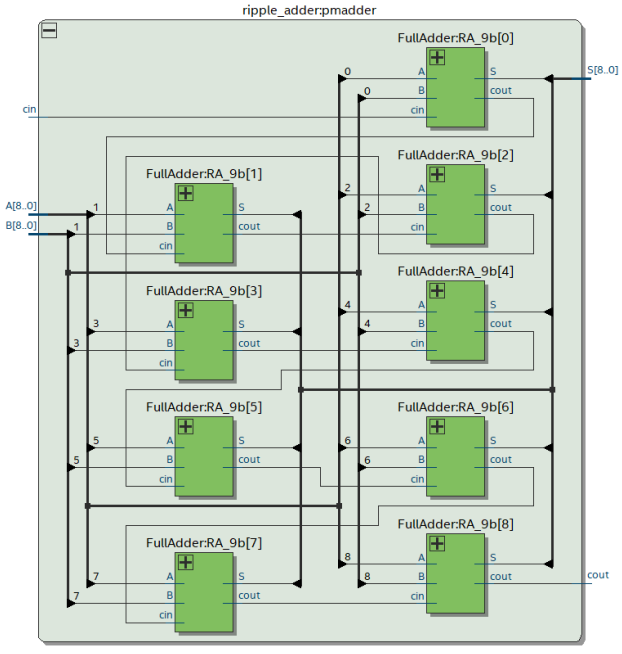


5. SyetemVerilog Module Description

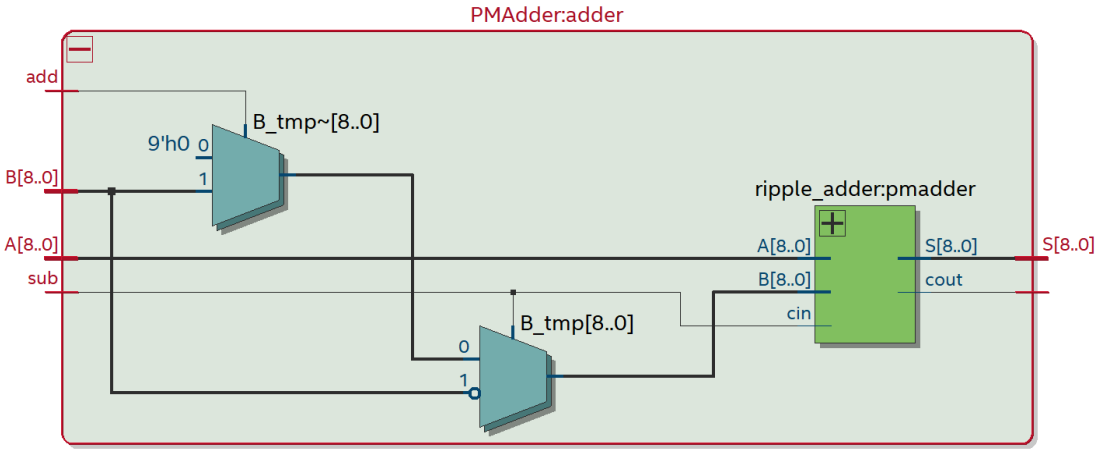
Module: FullAdder
Inputs: A, B, cin
Outputs: S, cout
Description: This module will perform combinational logic. It calculates *S* = *A* XOR *B* XOR *cin* and *cout* = the Majority of *A, B,* and *cin*.
Purpose: This module is used to performs one-bit full add and output the summation and carryout.



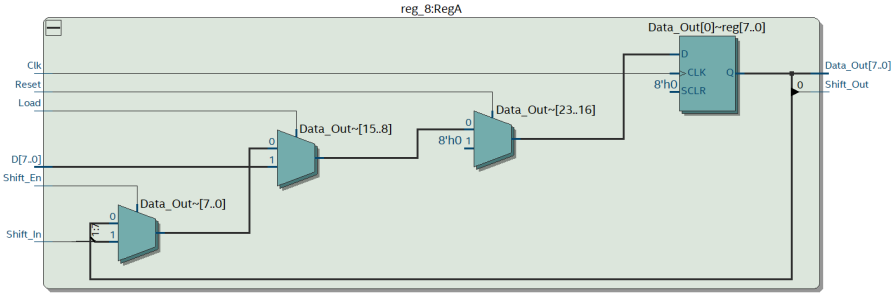
Module: ripple_adder
Inputs: [8:0] A, [8:0] B, cin
Outputs: [8:0] S, cout
Description: This module will perform combinational logic. It calculates the 9-bit summation between *A* and *B* given the carry *cin* by using 9 full adders to implement a 9-bit ripple carry adder. It will calculate the 9-bit *S* and a carry out *cout*.
Purpose: This is the 9-bit ripple carry adder.



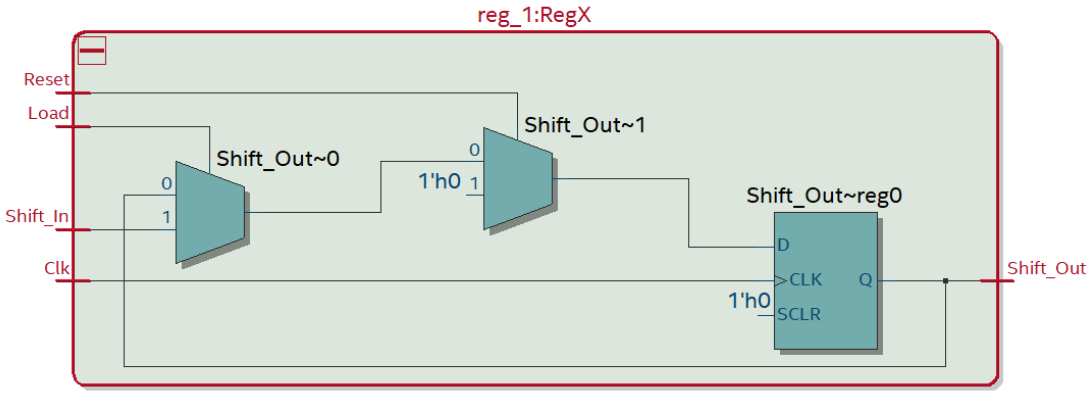
Module: PMAdder
Inputs: [8:0] A, [8:0] B, add, sub
Outputs: [8:0] S, cout
Description: This module will perform combinational logic. It will instantiate a 9-bit ripple carry adder and feed in the operands and carry in depend on *add* and *sub*. If *add* is high, we are going to set the operands to be *A*, *B* and carry in to be 1'b0. If *sub* is high, we are going to negate *B* first, and set the operands to be *A, B'* and carry in to be 1'b1. If neither is high, the operands will be *A*, 9'h00 and carry in to be 1'b0 so that the result will be A. *S* and *cout* will be the output *S* and *cout* from the 9-bit ripple carry adder.
Purpose: This module will perform the addition/subtraction of the register A and the multiplicand S. The *add*/*sub* signal depends on M and the state.



Module: reg_8
Inputs: [7:0] D, Clk, Reset, Load, Shift_In, Shift_En
Outputs: [7:0] Data_out, Shift_Out
Description: This is a positive-edge triggered 8-bit shift register module with asynchronous *Reset* and synchronous *Load* and *Shift_En*. When *Load* is high, data is loaded from *D* into the register on the positive edge of *Clk*. When *Shift_En* is high, the register will perform a logical right shift, and *Shift_In* will be the new MSB of *Data_Out* while *Shift_Out* will become the new LSB of *Data_Out*.
Purpose: This module is used to create the registers that store the multiplier and the result for the multiplier circuit. It will also perform the shift for the add-shift algorithm.



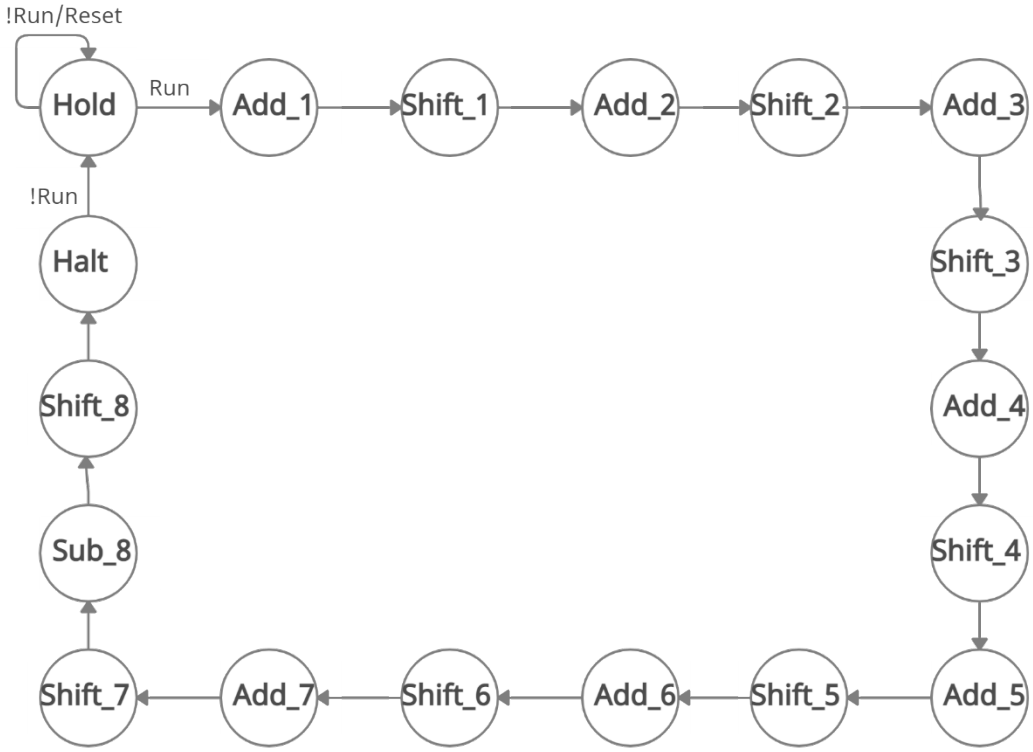
Module: reg_1
Inputs: D, Clk, Reset, Load
Outputs: Data_out
Description: This is a positive-edge triggered 1-bit register module with asynchronous *Reset* and synchronous *Load*. When *Load* is high, data is loaded from *D* into the register on the positive edge of *Clk*.
Purpose: This module is used to create the register for the sign extend variable X.



Module: control
Inputs: Clk, Reset, Run, M
Outputs: shift, add, sub, clrA
Description: This is a positive-edge triggered control module that has asynchronous *Reset* and synchronous *Run*. When *Run* is high at the first state, the control unit will set clrA to high for one clock cycle. The control unit will decide whether to enter the shift/add/subtraction state depending on the current state and *M*, and set either one of *shift*, *add* or *sub* to high.
Purpose: The control unit will control the shift/load operation of register units and the add/subtraction operation of the PMAdder unit depending on the current state and inputs.

Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: The module will output the code for seven-segment display depending on the input 4-bit data.
Purpose: This module is used to translate the binary data into the code for hex display on the board.

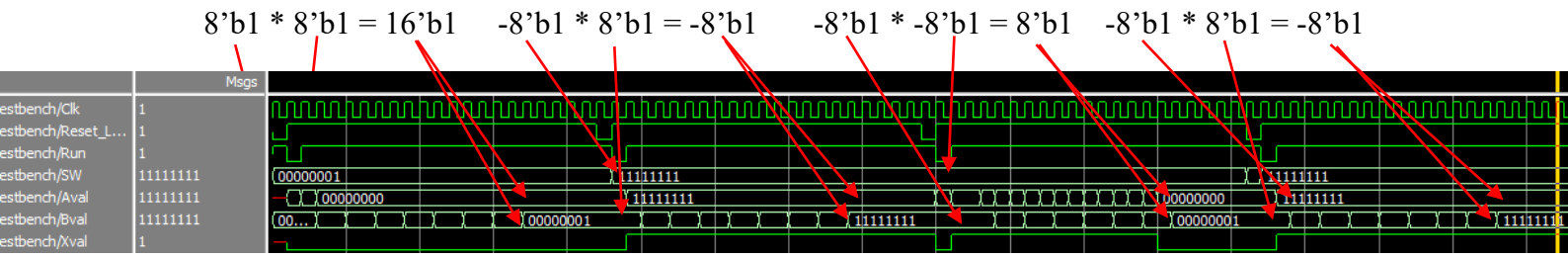
6. State Diagram for Control Unit



Notes:

1. At each Add_x state, the output will depend on M. To be specific, if M = 1, output will be shift = 0, add = 1, sub = 0; if M = 0, output will be shift = 0, add = 0, sub = 0.
2. At Sub_8 state, the output will depend on M. To be specific, if M = 1, output will be shift = 0, add = 0, sub = 1; if M = 0, output will be shift = 0, add = 0, sub = 0.
3. At each Shift_x state, the output will be shift = 1, add = 0, sub = 0.

7. Simulation Trace



8. Post-Lab Questions

1) Design Analysis

LUT	605
DSP	0
Memory (BRAM)	0
Flip-Flop	328
Frequency	77.02MHz
Static Power	89.99mW
Dynamic Power	5.90mW
Total Power	108.87mW

2) Purpose of Register X

Register X is the sign extend bit for register A. In add-shift algorithm, we need to perform the arithmetic shift. However, the shift registers will perform logical shift. Therefore, we would have a fix register X that can shift the sign bit to register A, so that the sign of the result can be consistent. The register X will be loaded after each addition/subtraction, together with loading register A. Noticing that we used a 9-bit adder for calculating the sum of A and S, the 9^{th} bit, as the sign bit, will be loaded to register X. Similarly, the register X will be cleared as register A being cleared, that is when we press Reset in the first place, or pressing Reset or Run once a multiplication calculation is done.

3) Limitation of 8-bit Adder

If we use 8-bit adder to calculate the addition/subtraction while use the carry out as the sign bit, this may lead to the change of sign if the input is negative but no carry out generates. For example, if one operand is 8'hFF and another one is 8'00, where the summation result will be 8'hFF, which is a negative number. However, in this calculation, no carry out generates, which will lead X to be 0. This is obviously problematic since if we perform shifting, the sign will be changed. Therefore, we need to perform 9-bit addition to keep the sign bit explicitly.

4) Limitation of Continuous Calculation

The limitation of continuous calculation is that as we stated before, when we

press Run after one calculation is done, we will clear the register A and X, which makes the 16-bit result of the first calculation will be truncated into 8 bits. Therefore, if the result of multiplication of the first calculation exceeds 8 bits, and we press the Run to continue the calculation, the multiplier will be truncated and thus the result will be wrong and the algorithm will fail.

5) Comparing with Pencil-and-Paper Method

Pencil-and-Paper method is intuitive for human to understand and calculate, and we can write down the expression and calculate time easily. However, this method is less feasible when implementing on FPGA since it is not worthy to save eight 16-bit binary data with half of them being zero and add them later. Add-shift is a more appropriate method to implement since it on one hand has the same basic idea as pencil-and-paper method when performing bit multiply and addition, on the other hand, it made use of the shift register so that we can move the multiplier instead of the multiplicand. This a more space saving strategy while no register is wasted during the calculation.

9. Conclusion

a) Functionality

The multiplier we designed can perform the multiplication of two 2's complement data with any sign combination. It can also perform continuous multiplication as long as the former result is less or equal to 8 bits. Initially our continuous multiplication function did not work, this is because we did not use a register to save X, but connect the 9^{th} bit of output directly to the Shift_In of register A. This led to the erroneous loading of the sign bit for register A. We fixed it by adding a register for X.

b) Comment on Lab Manual

Everything is good.