ECE 385 Lab Report

Spring 2023

Lab5: An 8-Bit Multiplier in SystemVerilog

Name: Zhu Hanggang, Hong Jiadong

Student ID: 3200110457, 3200110970

Prof. Chushan Li, Prof. Zuofu Cheng
ZJU-UIUC Institute
April 6, 2023

1 Introduction

The multiplier circuit we designed in this lab supports multiplication of 8-bit 2's compliment number. It consists of shift registers to store value of multiplicand and multiplier. The control logic is implemented with a Moore machine with 17 states and there's a 9-bit adder/subtractor to do computation. The algorithm used is the straightforward add-shift method.

2 Pre-lab question

8-bit multiplication example. Use Multiplier B = 00000111(7) and Multiplicand S = (11000101)(-59).

Function	X	A	B	M	Comments for the next step
Clear A, LoadB	0	0000 0000	*0000 0111*	1	M = 1, S added to A
ADD	1	1100 0101	*0000 0111*	1	Shift XAB
SHIFT	1	1110 0010	1*000 0011*	1	M = 1, S added to A
ADD	1	1010 0111	1*000 0011*	1	Shift XAB
SHIFT	1	1101 0011	11*00 0001*	1	M = 1, S added to A
ADD	1	1001 1000	11*00 0001*	1	Shift XAB
SHIFT	1	1100 1100	011*0 0000*	0	M = 0, 0 added to A, Shift XAB
SHIFT	1	1110 0110	0011 *0000*	0	M = 0, 0 added to A, Shift XAB
SHIFT	1	1111 0011	0001 1*000*	0	M = 0, 0 added to A, Shift XAB
SHIFT	1	1111 1001	1000 11*00*	0	M = 0, 0 added to A, Shift XAB
SHIFT	1	1111 1100	1100 011*0*	0	M = 0, Subtract 0 from A,Shift XAB
SHIFT	1	1111 1110	0110 0011	1	All shifts done, stop

Table 1: Pre-Lab Question

3 Written description and diagrams of multiplier circuit

3.1 Summary of operations

To operate multiplication, the basic operation includes addition/subtraction and shifting. The multiplier is stored in register B and the multiplicand is from input switches S. For each addition, result of S + A is stored in flip-flop X and register A, with X being the sign bit. For each subtraction, A - S is computed and result is stored in flip-flop X and register A. After each operation, the whole register XAB is arithmetically shifted right. Whether the addition/subtraction operation is executed depends on the least significant bit of B. If B[0] is 0, then no operation is required. After 8 times of shifting, the whole result is stored in XAB.

The control of each operation is implemented in a Moore Machine. There are 19 states in total. 16 states (represented as $S_0 ... S_{15}$ are used to indicate the operations of addition/subtraction or shifting. In addition/subtraction state, the state machine will output signal *Add* or *Sub* based on current least significant bit of B. Register A will load new computed data if either *Add* or *Sub* is set to high. In shifting state, the state machine will set output $Shift$ to enable shifting. As expected, all addition/subtraction state and shifting state take turns alternatively and the only subtraction state is $S14$, the last state of addition/subtraction state.

There is one initial start state S_{start}, and the machine only switches to S_0 if *Run* is set. In start state, If $ClearA\_LoadB$ is set by user, it will output corresponding output to registers. After all operations are done, the Moore machine will turn into the waiting state S_{wait}. The machine will stay in wait state until the *Run* signal is back to low. The machine will go back to start state when *Run* signal is not set to high anymore. To enable consecutive multiplication, we add one clear state S_{clear} between S_{start} and S_0, to

clear the data stored in XA before each multiplication. In all cases, if *Reset* is set, the machine will go to the start state. More details of the state machine can be seen in Figure 6

3.2 Top Level Block Diagram

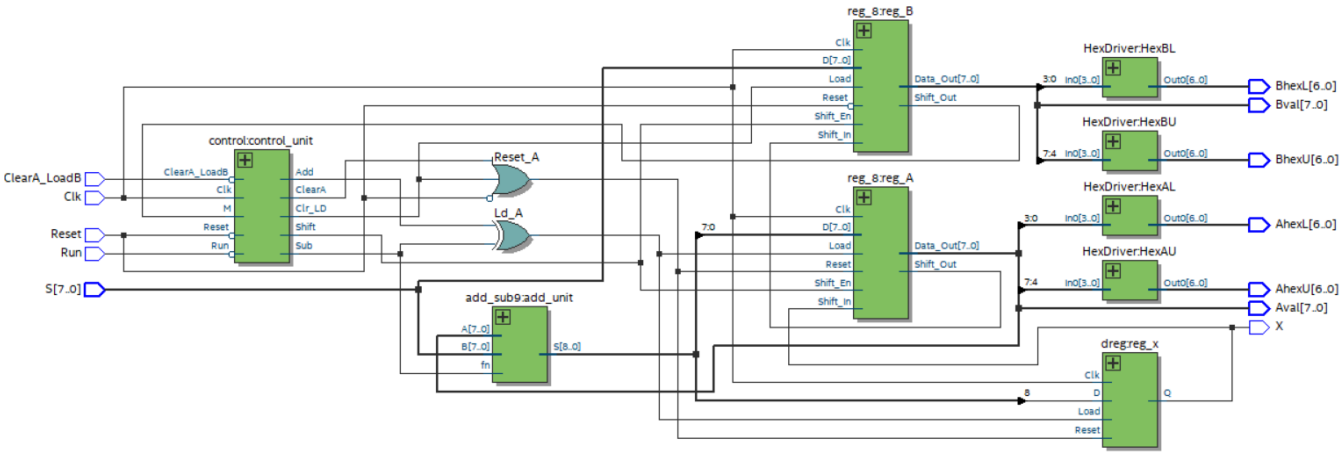

Figure 1: Top Level Block Diagram

3.3 Written Description of .sv Modules

3.3.1 Module 1

Module: full_adder
Inputs: A,B,C
Outputs: [15:0] Sum,CO
Description: A simple 1-bit full adder that takes two operands and one carry in and output the sum and carry out bit.
Purpose: This module servers as a basic module to operate add operation.

3.3.2 Module 2

Module: add_sub9
Inputs: [7:0] A, [7:0] B,fn
Outputs: [8:0] S
Description: A 8-bit adder/subtractor. When fn is set to 0 it acts as a adder and a subtractor that computes A - B otherwise. Then result is a 2's complement number with the 9^th bit the sign bit.
Purpose: This module is used to do one add/subtract operation of the multiplier.

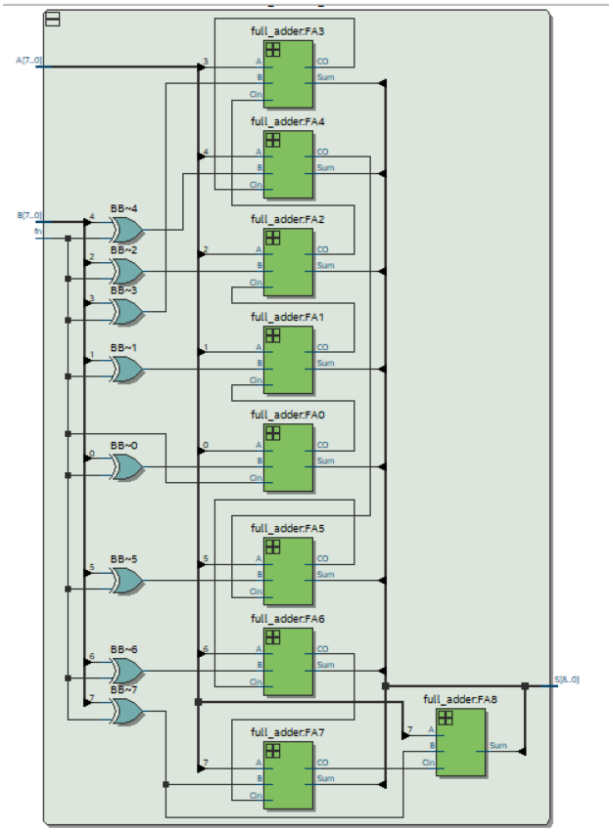

Figure 2: Adder Subtractor

3.3.3 Module 3

Module:reg_8
Inputs: [7:0] D, Clk,Reset, Shift_In, Load, Shift_En
Outputs: [7:0] Data_Out, Shift_Out
Description: A 8 bit synchronous shift register. When Load is high, Data of D is loaded into the register. When Shift_En is high, Shift_In Data will be loaded into the leftmost bit of the register and rightmost bit is shifted out into Shift_Out. All operations on rising edge of Clk. **Purpose**: This module is used to store data of operands of A and B of the multiplier.

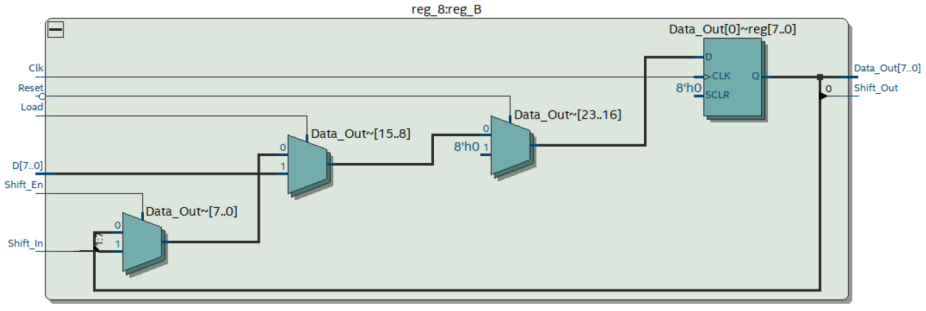

Figure 3: Register

3.3.4 Module 4

Module:dreg
Inputs: D, Clk, Load, Reset
Outputs: Q
Description: A 1-bit flip-flop that loads value of D if Load is high on rising edge of Clk. Otherwise, keep the original value. **Purpose**: This module is used to store bit X in he multiplier.

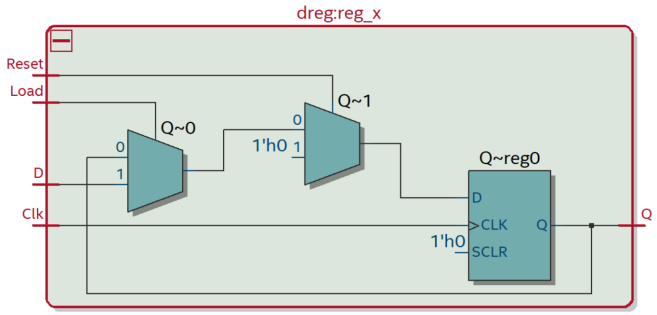

Figure 4: Flip-Flop

3.3.5 Module 5

Module: control
Inputs: Run, ClearA_LoadB, Reset, M, Clk
Outputs: Clr_LD, ClearA, Shift, Add, Sub
Description: A finite state machine that acts the control unit of the multiplier. Based on User's command of Run, Reset, ClearA_LoadB and bit 0 of B(M), set according output to registers or adders. It controls whether the register should be cleared or shifted and whether add_sub9 should do add operation or subtraction operation. **Purpose**: This module is used to control when to do operations and how many operations are required for the multiplier.

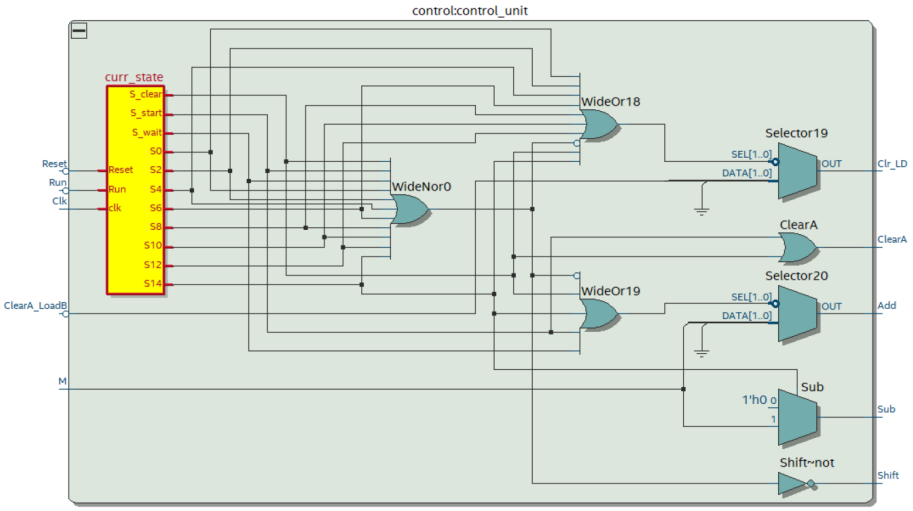

Figure 5: Control Unit

3.3.6 Module 6

Module:HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: A hex driver that transform a 4-bit number into a hex 7-segment display. **Purpose**: To display hex value on FPGA.

3.3.7 Module 7

Module: lab5_toplevel
Inputs: [7:0] S, Run, ClearA_LoadB, Reset, Clk
Outputs: [6:0] AhexU, [6:0] Ahex, [6:0] BhexU, [6:0] BhexL. [7:0] Aval, [7:0] Bval, X
Description: The toplevel of the multiplier. It consists of a control unit, a adder/subtractor, two registers storing A,B, a flip-flop storing X and hexdrivers to display value for FPGA. Inputs are commands from user and output includes hex value for the FPGA as well as XAB value for debug purpose.
Purpose: This module connects all usb-modules together and act as the 8-bit multiplier.
The figure of toplevel multiplier is already shown in figure 1

3.4 State Diagram for Control Unit

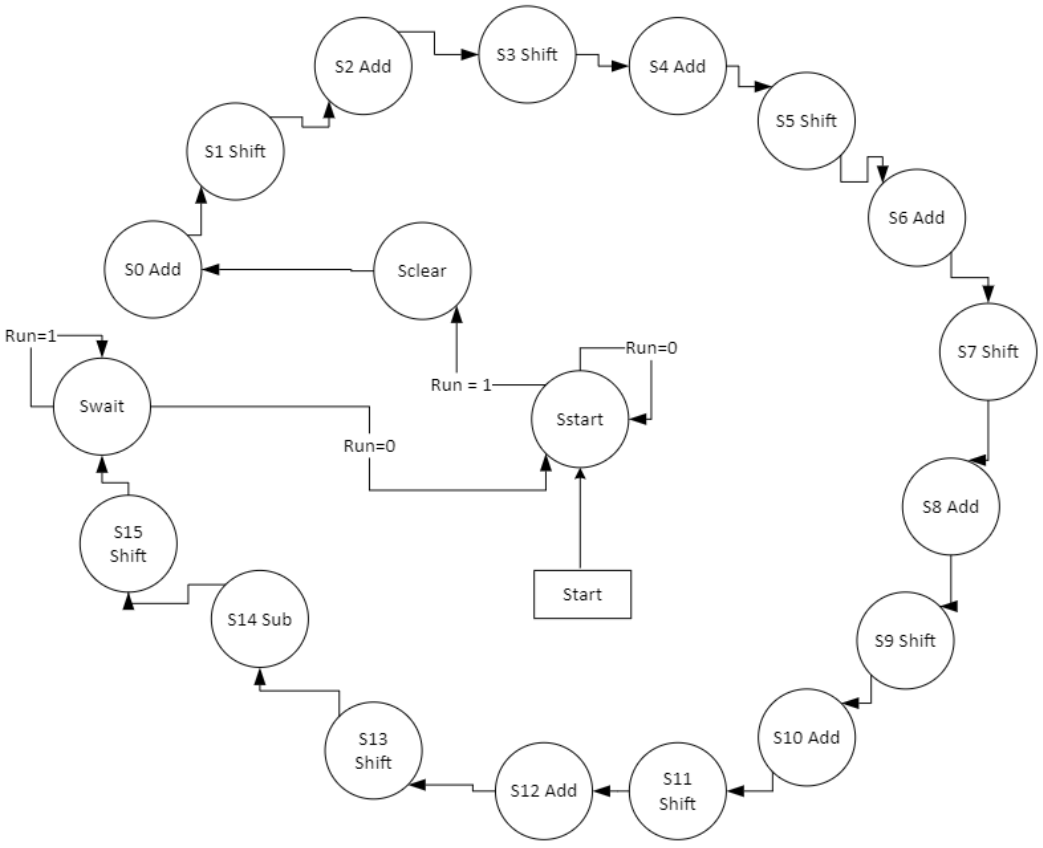

Figure 6: State Machine

Note that there's a arrow from all states to S_{start} state when *Reset* is set to high. It is ignored here for simplicity.

4 Annotated pre-lab simulation waveforms

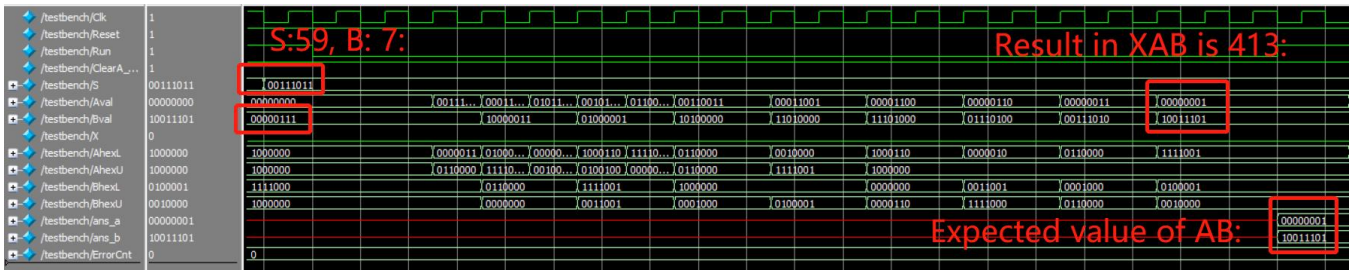

Figure 7: Wavefrom (+*+)

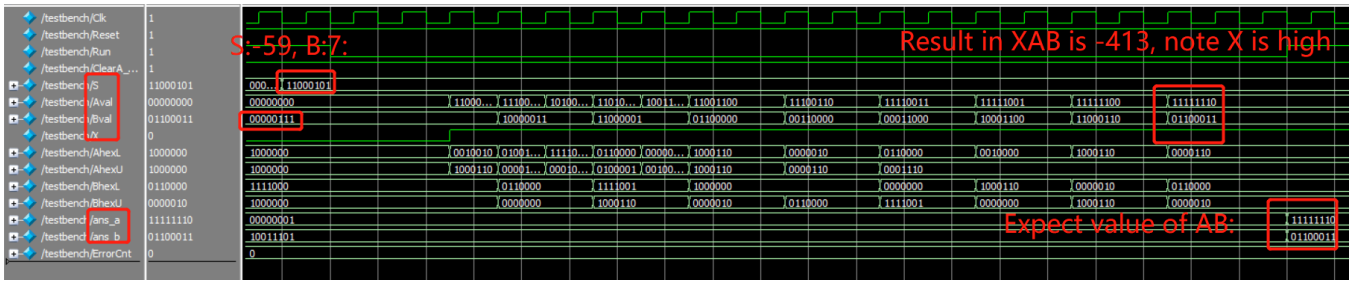

Figure 8: Wavefrom (+*-)

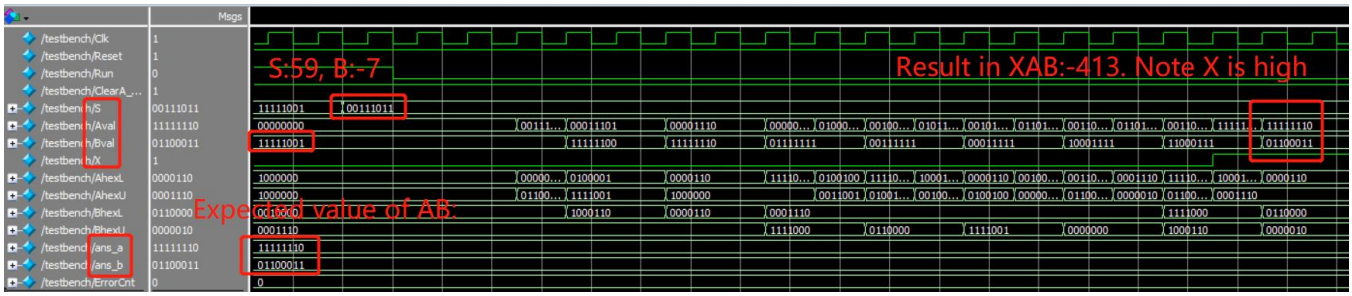

Figure 9: Wavefrom (-*+)

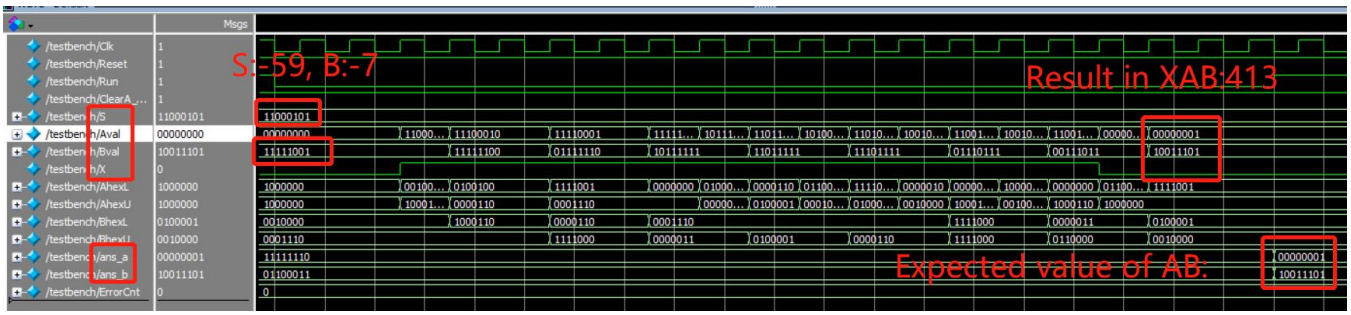

Figure 10: Wavefrom (-*-)

Figure 7 to Figure 10 shows the simulation waveform. Operands and Result are annotated and one can also see the intermediate add/sub and shift operations.

5 Answers to post-lab questions

1. **Design statistics table**

LUT	95
DSP	0
Memory(BRAM)	0
Flip-Flop	36
Frequency(Mhz)	85.08
Static Power(mW)	98.53
Dynamic Power(mW)	2.26
Total Power(mW)	153.48

Table 2: Design Statistics Table

To decrease the number of LUTs, we can use Mealy machine instead of Moore machine. Mealy machine requires less states so it requires less logical gates. But it can be more complicated to implement a Mealy machine. Besides, in our design, we use ripple carry adder to implement addition/subtraction operations. Alternatively, carry-lookahead adder or carry select adder can me used to increase the maximum frequency.

2. **What is the purpose of the X register. When does the X register get set/cleared?**

X register is used to preserve the sign bit. After some addition operation, it's possible that the most significant bit of the result is 0 while there's a carry out bit 1. If X is not used, this sign bit 1 will be lost. The X register get set/cleared at the same time as the register A. More specifically, X get set when addition/subtraction is operated. X get cleared when $ClearA\_LoadB$ button is set, $Reset$ button is set or in clear state S_{clear}.

3. **What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?**

As the data in XA is reset every time after multiplication, it means that the result of continuous multiplication must be represented in a 8-bit number. If the result exceeds the limit, the result of the algorithm will fail.

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

Advantages: The add and shift algorithm makes full use of 2-bit number and can be easily implemented in a processor. Also, only two registers and one flip-flop are used. All intermediate data is stored in these two registers and it saves lots of space compared to directly implementing pencil-and-paper method on processor.

Disadvantages: the algorithm is less intuitive that pencil-and-paper method. And once you get the result of multiplication, you can't see the operands anymore on DE-2 board. Also, it has limitation on consecutive multiplications as we discussed in previous question.

6 Conclusion

6.1 Discussion of functionality of our design

Our design has great support for all kinds of 8-bit multiplication within 17 clock cycles. It works well for all cases where operands have signs (+*+), (+*-), (-*+), (-*-). And it also supports consecutive multiplication as long as the result of last multiplication can be stored in a 8-bit register(B). All the

functionality key *Run*, *Reset*, *ClearA_LoadB* works well. But our design can still be improved. For example, we use lots of states in our Moore machine and it's possible to reduce some states using Mealy machine.

6.2 Difficulty in the lab and possible improvement

One important thing that we didn't realize when we first implement the multiplier is that buttons on DE-2 board are all **active low**, which means when need to complement input signals. This is important but it seems that it isn't mentioned in the lab manual. Overall, the lab manual is detailed-written and explain jobs explicitly.