ECE 385

Spring 2016

Experiment #5

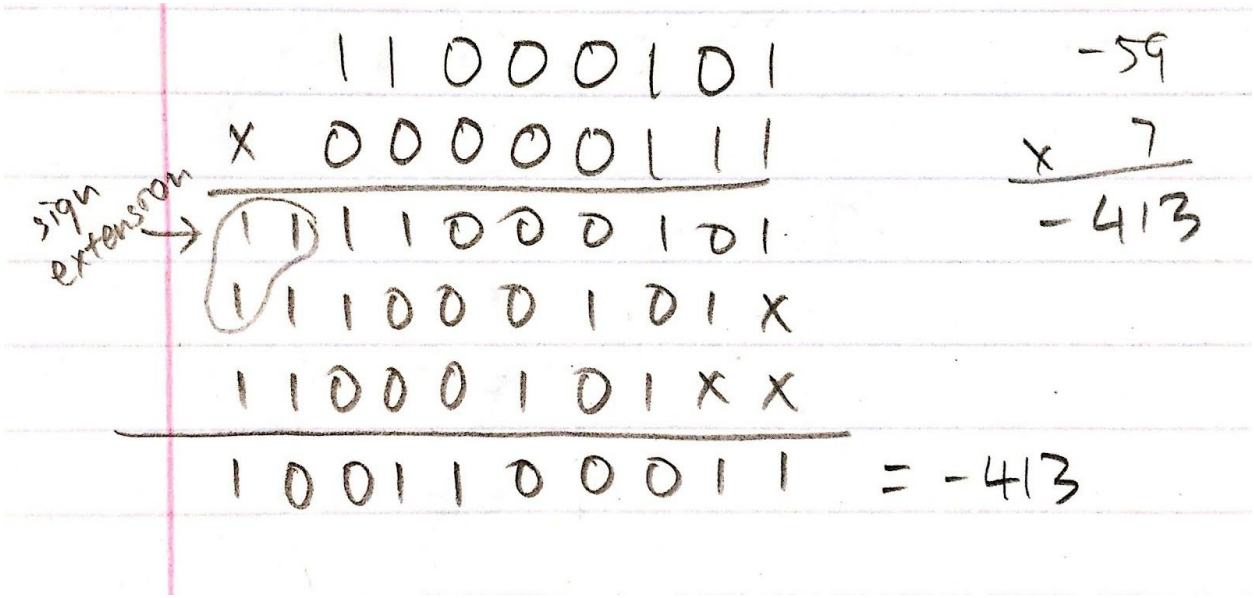
An 8-Bit Multiplier in SystemVerilog

Samuel Lim, Fergal Lonergan
Section ABB / Friday 11AM
Conor, Debjit

Introduction

The objective of this lab was to design a multiplier for two 8-bit 2's compliment numbers in SystemVerilog. The lab also required us to run the multiplier on the DE2 FPGA board. This required the use of different modules such as the 8-bit register, a 9-bit adder, control unit, etc. Using all of these together formed the 8-bit 2's compliment multiplier.

8-Bit Multiplication Example



Written Description

Our circuit is composed of two registers, the 9-bit adder, the X-bit flip flop, and a control unit. The control unit consists of a state machine that has 18 states. The first state is the "reset" state, waiting for the "Run" switch to start the cycle. The next 16 states are eight pairs of adding and shifting states (for this report, let's call them states Y and Z). In Y, if the rightmost bit of B is 1, then it takes the result of the 9-bit adder and loads it into register A. Then it goes on to state Z. In Z, it shifts registers A and B by one bit. This happens 8 times (hence 16 states) because there are 8 bits we are shifting. The last Y state changes the function of the 9-bit adder to subtract, because if the most significant bit of B initially was 1, the last addition must be a subtraction. The 18th state prevents any more loading or shifting.

The flip-flop X is just a sign extension for when shifting registers A and B. If the product is currently negative, then bit-1 must be shift left into register A when shifting. However, the value in X must only change when A is actually being loaded by the control unit. Therefore the control unit also loads X when the rightmost bit of B is 1.

Written purpose and operation of each module

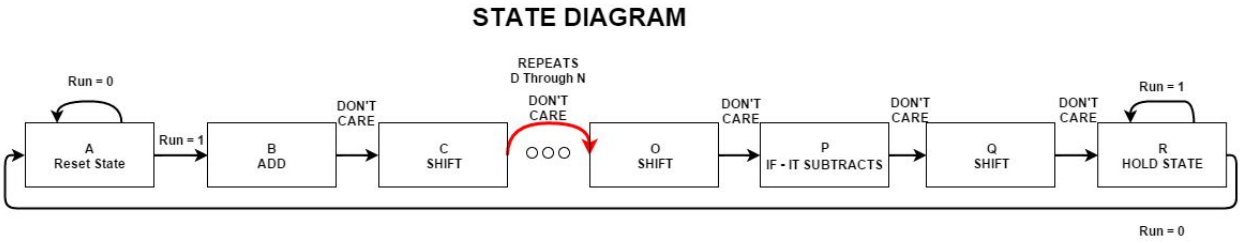
1. full_adder
    a. inputs: x, y, z
    b. outputs: s, c
    c. Purpose of this module is to take add two numbers and the carry bit (if necessary) of the previous (lesser) adder. This module is used by the 9-bit adder.
2. adder9

    a. inputs: A[7:0], B[7:0], fn
    b. outputs: Sum[8:0]
    c. Purpose of this module is to take two 8-bit numbers and either add them or subtract them. The leftmost bit in Sum is just a sign extension of the sum, which is why it is 9 bit. The input fn controls whether the numbers are added or subtracted. When you subtract two 2's compliment numbers, you can negate the second number and add the two numbers. To negate a 2's compliment number, you must invert every bit then add 1. This process is used in this module. Regardless of what fn is, B is initally XOR'd to an 8 bits of fn. (If fn is 1, then 11111111. If fn is 0, then 00000000). Then, fn is inputted to the rightmost adder's carry-in bit. So by now you can see that if fn is 1, then B is inverted by XOR'ing with 11111111, is added 1 by the carry-in bit in the rightmost adder. If fn is 0, none of this happens. The 9th bit is just a copy of the 8th bit because it is a sign extension. The 9th bit is connected to the input D of the flip flop while the other 8 bits are connected to D of register A.
3. Dreg
    a. inputs: Clk, Load, Reset, D
    b. outputs: Q
    c. Purpose of this module is to hold the sign extension bit for register A. The input D is the 9th bit of the 9-bit adder. Load is 1 when register A is being loaded. This allows the correct bit to be shifted into A. The output Q is connected to the shift bit into register A.
4. reg_8
    a. inputs: Clk, Reset, Shift_In, Load, Shift_En, D[7:0]
    b. outputs: Shift_Out, Data_Out[7:0]
    c. Purpose of this module is to store 8-bits of values. This can shift right at every rising edge of clock if Shift_En is 1. It can also load values into all 8 bits from D if Load is 1. Data_Out is the value of the 8-bit "word", and Shift_Out is the rightmost bit that is the first to be shifted out. Shift_In is the bit that would go in as the leftmost bit if the register were to be shifted. This module creates registers that can be controlled by higher level modules such as register_unit.
5. register_unit
    a. inputs: Clk, Reset, A_In, ClearA_LoadB, Shift_En, Ld_A, D_A[7:0], D_B[7:0]
    b. outputs: B_Out, A[7:0], B[7:0]
    c. Purpose of this module is to create two register for A and B. Reset resets both registers, ClearA_LoadB resets only A and loads D_B into register B. B_Out is used as Mval to see whether or not register A should be loaded from the output of the adder. Inputs Ld_A and Shift_En are mainly controlled from the control unit in the previously mentioned states Y and Z to load and shift the registers.
6. control
    a. inputs: Clk, Reset, ClearA_LoadB, Run, Mval
    b. outputs: Shift_En, fn, Ld_A
    c. Purpose of this module is to control when and how many times A and B are shifted. This module also consists of the state machine previously mentioned in "Written Description". This control module controls whether the 9-bit adder is

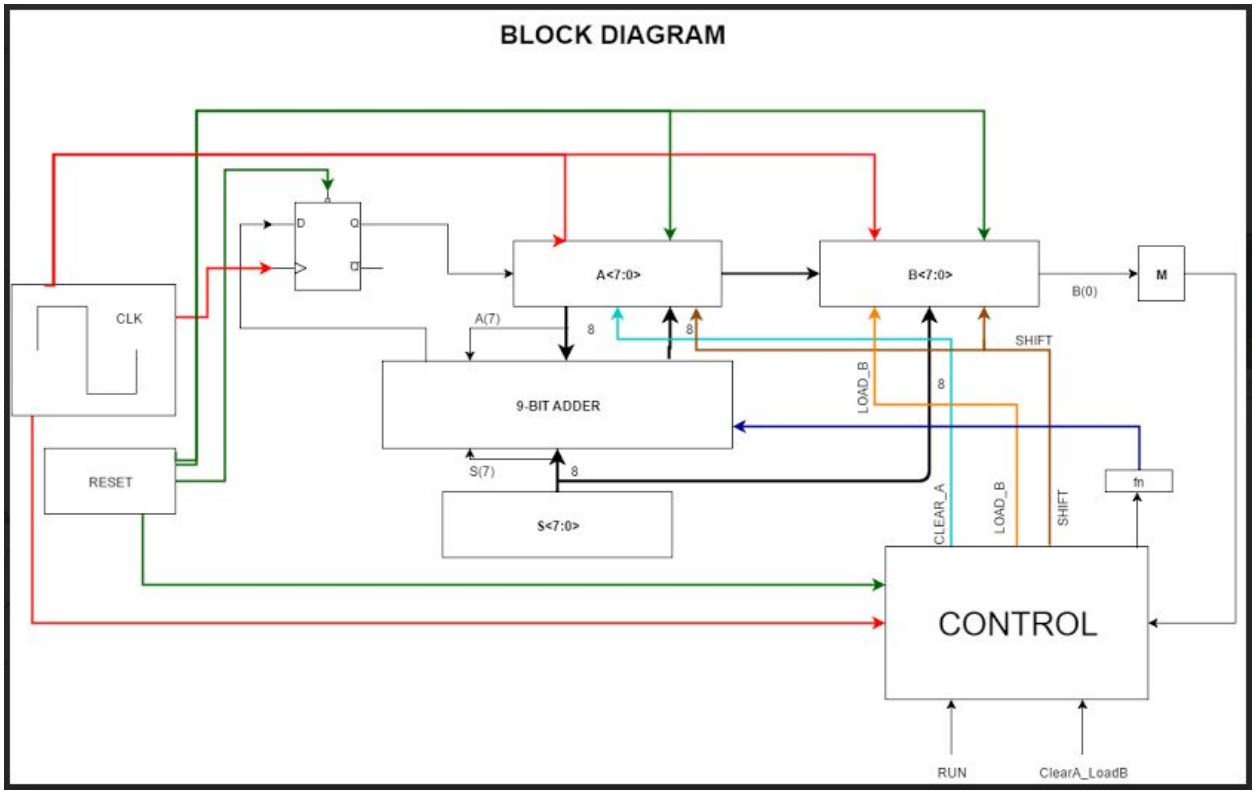
adding or subtracting (fn), whether the registers are loading or shifting (Ld_A and Shift_En), and whether the machine is even computing anything at all. Reset clears both registers. ClearA_LoadB resets register A and loads the number from the switches into B. Run executes the multiplication process. Mval tells the control unit if it should load the sum into A or not.

7. multiplier
    a. inputs: Clk, Reset, Run, ClearA_LoadB, S[7:0]
    b. outputs: AhexL, AhexU, BhexL, BhexU
    c. Purpose of this module is to serve as the toplevel entity of the lab. It creates the registers, the control unit, the flip-flop, and the adder. It “wires” them up so that they can be used as a unit that multiplies two numbers. The product of the two numbers are displayed through the hexdrivers using outputs AhexL, AhexU, BhexL, and BhexU.

State Diagram for the Controller

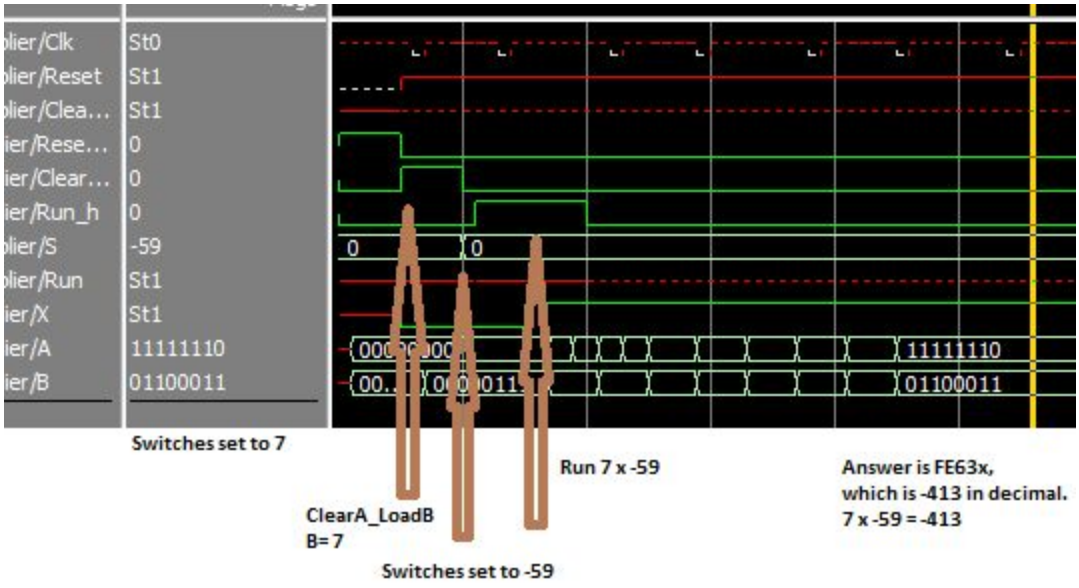


Schematic Block Diagram

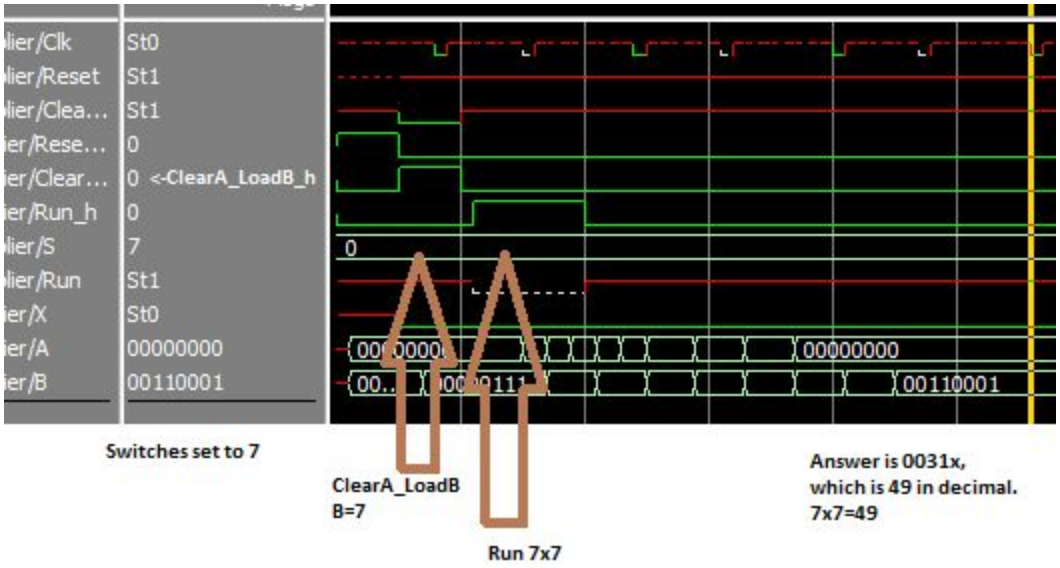


Annotated Pre-Lab Simulation Waveforms

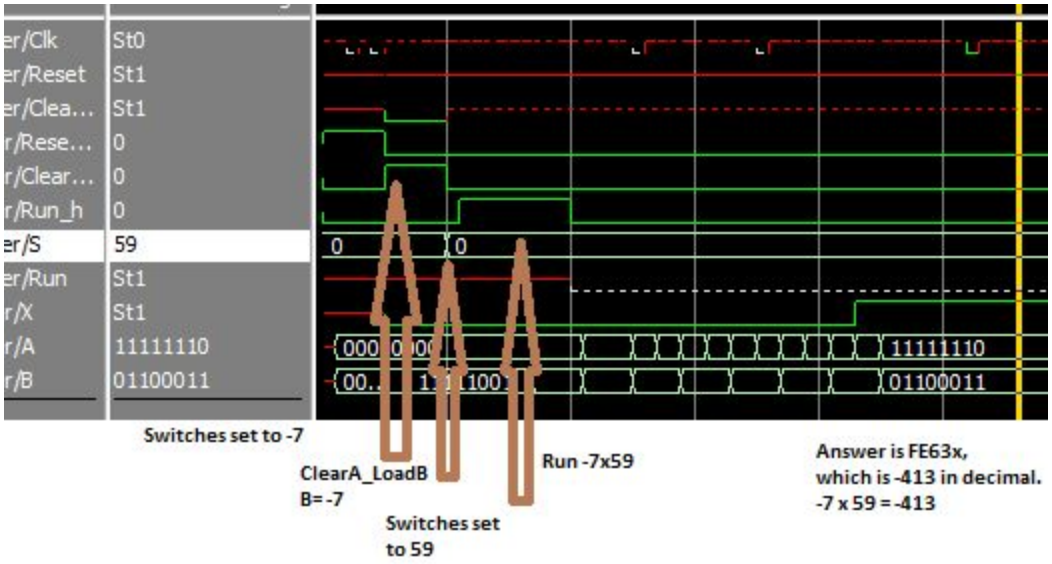
+*+



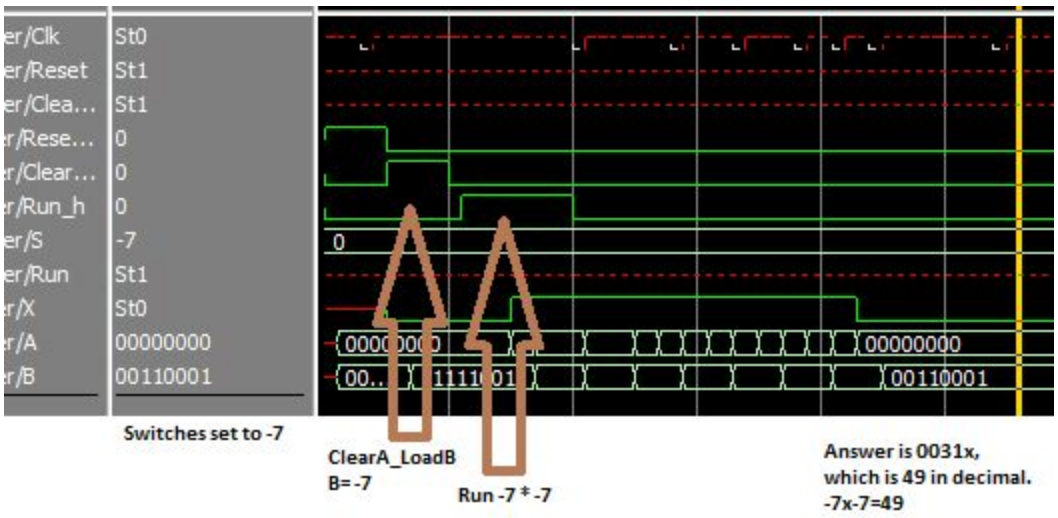
+*-


lier/Clk
lier/Reset
lier/Clea...
ier/Rese...
ier/Clear...
ier/Run_h
lier/S
lier/Run
ier/X
ier/A
ier/B
St0
St1
St1
0
0 <-ClearA_LoadB_h
0
7
St1
St0
00000000
00110001
0
00000000
00110001
Switches set to 7
ClearA_LoadB
B=7
Run 7x7
Answer is 0031x,
which is 49 in decimal.
7x7=49

-*+


er/Clk
er/Reset
er/Clea...
r/Rese...
r/Clear...
r/Run_h
er/S
er/Run
r/X
r/A
r/B
St0
St1
St1
0
0
0
59
St1
St1
11111110
01100011
0
0
11111110
01100011
Switches set to -7
ClearA_LoadB
B=-7
Switches set
to 59
Run -7x59
Answer is FE63x,
which is -413 in decimal.
-7 x 59 = -413

-*-



Post Lab Questions

- Design Statistics Table

LUT	87
DSP	0
Memory (BRAM)	0
Flip-Flop	1
Frequency	229.31 MHz
Static Power	98.49 mW
Dynamic Power	0
Total Power	139.02 mW

- How we might optimize our design
    - If we could use the SystemVerliog arithmetic operations, we could've decreased total gate count. Our 9-bit adder could've just been done without using full_adder's.

Conclusion

In conclusion our design worked exactly as desired.
We created quite a few sub modules which we believed improved the readability and efficiency of our design. We had some difficulties early on with design of our controller and misjudging the amount of states we needed.
We also believed that it might be possible for us to compress two states into one, ie add then shift in the one state. However, we noticed quite quickly that this doesn't work as you need two separate clock edges to complete those operations. We also originally programmed our X bit incorrectly so we were getting incorrect values for our multiplication as the signs were incorrect.