ECE 385

Fall 2023

Lab 4: 8 bit multiplier

Alex Yu, Corey Yu
GL/ 13:45-14:00
Gene Lee

1. Introduction

In this lab, we created a multiplier that can multiply two 8 bit numbers. The two 8 bit numbers can be either positive or negative (2's complement), and this machine will output the result with the correct result with the sign signal. This machine takes input from the switch on the Urbana board; it is used to load B register and act as the A register input when calculating the result. (By clicking the load button the register will load the switch onto the B, which will show up on the hex display.) The result will be output to the hex display after the run button is pressed. This machine can also perform a series of calculations, using the last result as the input and continue the calculation. There is also an overflow bit that lights up if there is an overflow.

2. Pre-lab question

S=11000101, B=00000111

Function	**X**	**A**	**B**	**M**	**Comments for the next step**
ClearA, LoadB, Reset	0	0000 0000	00000111	1	M is one so add S to A
Add	1	11000101	00000111	1	Shift XAB right by one bit after add complete
Shift	1	11100010	10000011	1	Add
Add	1	10100111	10000011	1	Shift
Shift		11010011	11000001	1	Add
Add	1	10011000	11000001	1	Shift
Shift	1	11001100	01100000	0	M=0, shift
Shift	1	11100110	00110000	0	M=0, shift
Shift	1	11110011	00011000	0	M=0, shift
Shift	1	11111001	10001100	0	M=0, shift
Shift	1	11111100	11000110	0	M=0, shift
Shift	1	11111110	01100011	1	8 shifts complete, stop

3. Written description and diagrams of multiplier circuit

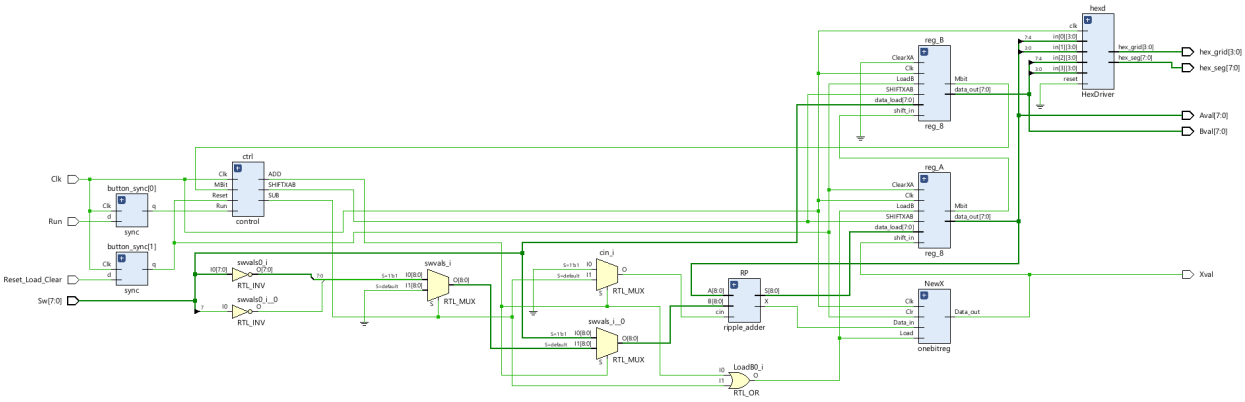
a. Summary of operation

i. Explain in words how operands are loaded, how the multiplier computes its result, how the result is stored, etc.

To multiply two numbers C, D first load C on the switches then hit Reset_Load_Clear (Button 0). This will load X into register B. Then load D into the switches and hit Run (Button 1). This will perform the operation following the implemented algorithm.

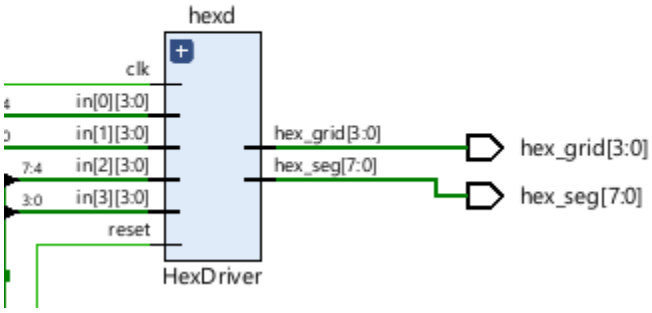
The LSB of register B is checked; if it's 1 then S and A are sign extended, summed together and the MSB of the result is stored in a one-bit register X, and the other 8 bits in register A, if it's 0 then add-enable is set low and the next state proceeds.. This is the first state of the algorithm. Then XAB is arithmetically right-shifted; this is the other operation in the algorithm. The shift is akin to adding a zero to the least significant digit when multiplying decimal numbers on paper with the standard algorithm. The cycle of add then shift occurs 7 times. On the final add state, the LSB is again checked. However this is now the sign bit of X (a 1 means C was negative, not a 2^n value), so if it's 1 then S is subtracted from A rather than added, and if it's 0 then no operation occurs. Then the final shift operation occurs.

Top Level Block Diagram



Written Description of .sv Modules

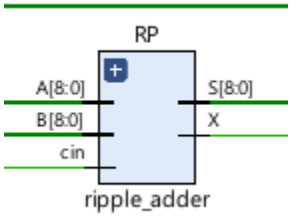
Module: HexDriver.sv



Inputs: Clk, reset, [3:0] in[4], [3:0] nibble
Outputs: [7:0] hex_seg, [3:0] hex_grid, [7:0] hex

Description: The hex driver takes in the data in the binary format and outputs the number to hex that can be shown on the LED segment display.
Purpose: The LED segment display doesn't take in numbers in binary so we need this module to translate it to display.

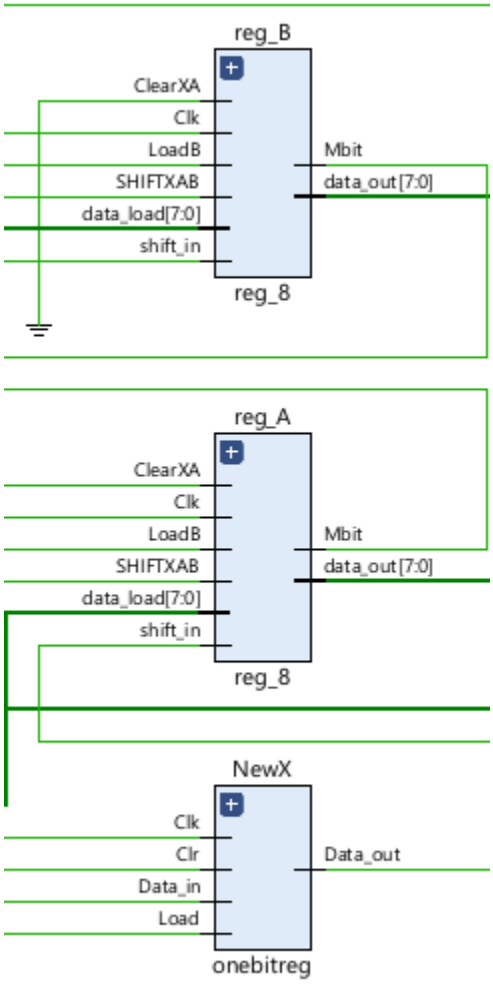
Module: ripple_adder.sv(Module: ADDER4, Module: full_adder)



input [15:0] A, B,
input cin,
output [15:0] S,
output cout

Description: the ripple_adder module contains two submodules full_adder and ADDER4. The full_adder is a one bit full adder that takes in one bit input and does the calculation only on a single bit. In the ADDER4 module, we call the full_adder 4 times, to calculate a total of 4 bits. Finally, in the actual ripple adder, we call the 4 bit adder (ADDER4 module) 4 times to get a 16 bit result in total.
Purpose: We need this module to implement the ripple adder, and the reason why we wrote two submodules instead of just calling one bit adder four times is that it will make the debugging faster and the code more efficient.

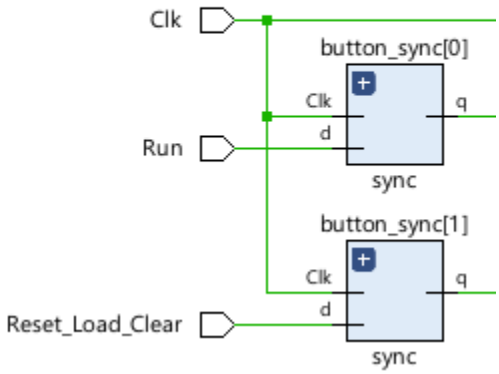
Module: reg_8.sv



input	logic	Clk, Reset, Load, SHIFTXAB, ClearXA, LoadB, shift_in
input	logic	[7:0] data_load
output	logic	[7:0] Data_Out

Description: This is the register that stores data in register A and register B. It can clear the data when it receives the reset signal, and load the data when the load button is pressed. It can also shift in and shift out data using the shift enable bit (SHFTXAB)
Purpose: We need the register module to store data, load data, clear data, and shift data from register A to register B.

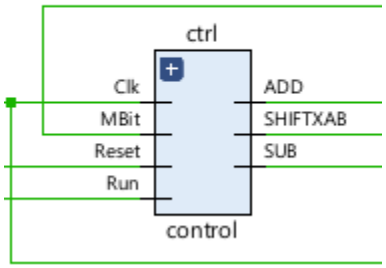
Module: Synchronizer.sv



Inputs: Clk, Reset, d
Outputs: q

Description: This acts like a flip flop that takes an asynchronous signal and makes it synchronous. The asynch input is fed into the flip flop and outputs the data on the next clock cycle.
Purpose: This module forces the entire design to be synchronous, or that every module runs on the same clock cycle.

Module: Control.sv



input logic Clk, Reset, Run, MBit,
output logic ADD, SUB, SHIFTXAB, ClearXA, LoadB

Description: This is a twenty state control unit that goes through every single state of adding, shifting, and checking if the run button is pressed.

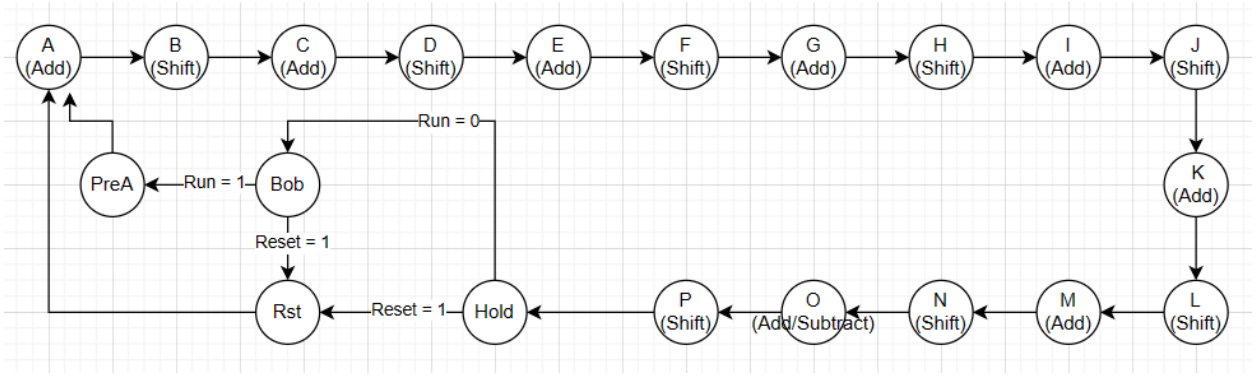
Purpose: We need the control unit to make sure the number in the register A and B is properly added and shifted. Because the shift and add operations are constantly changing, we need to output a different control signal for each state; moreover, after the whole operation is done, we also need a few extra states to make sure the machine doesn't run infinitely and output the wrong result. It has to stop after one cycle.

Module: adder_toplevel.sv
input Clk, Reset_Clear, Run,
input [7:0] Sw,
output logic sign_LED,
output Xval,
output logic [3:0] hex_grid,
output logic [7:0] hex_seg,
output logic [7:0] Aval,
output logic [7:0] Bval

Description: This is the top level module that takes the input data from the switch, calls the modules with the input data, and outputs the results from the module output to the hex display.
Purpose: This module organizes data between all the sub modules and wraps everything. We need this module to control which module output goes into which module input, basically linking all the modules together.

You may insert expanded RTL diagrams of each individual module here if it is legible.

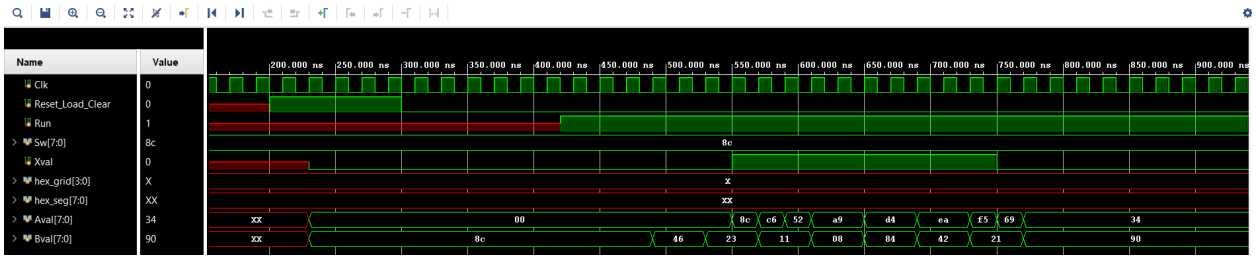
State Diagram for Control Unit



Annotated pre-lab simulation waveforms

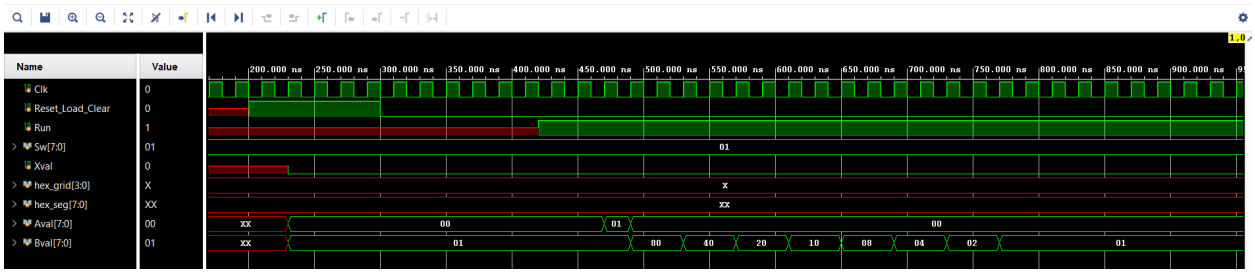
In these 4 cases Aval is the A register, the B val is the B register, and the output is across both A register and B register, where A is the first 8 bits and B is the last 8 bits. In the test bench we first pressed reset and then ran.

Both Negative:



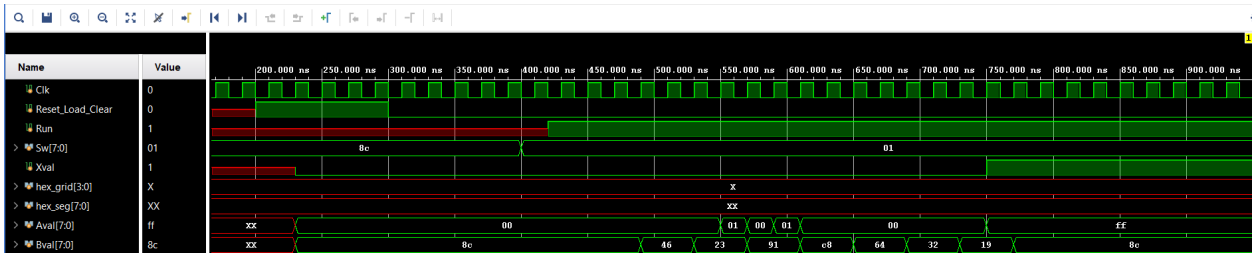
Output ^

Both Positive:



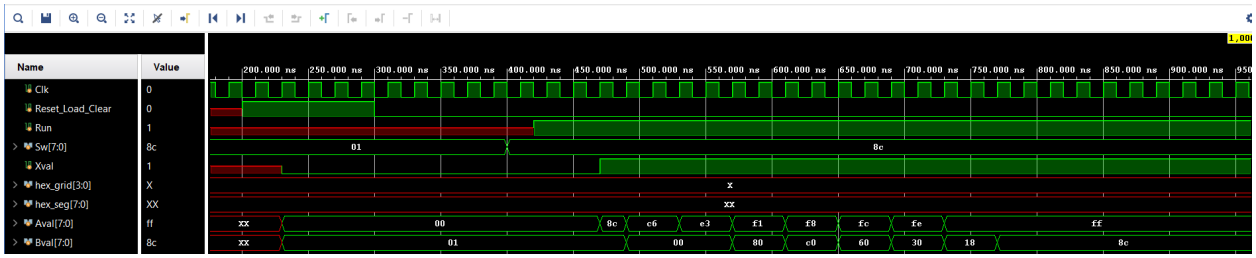
Output ^

A positive, B negative:



Output ^

B positive, A negative:

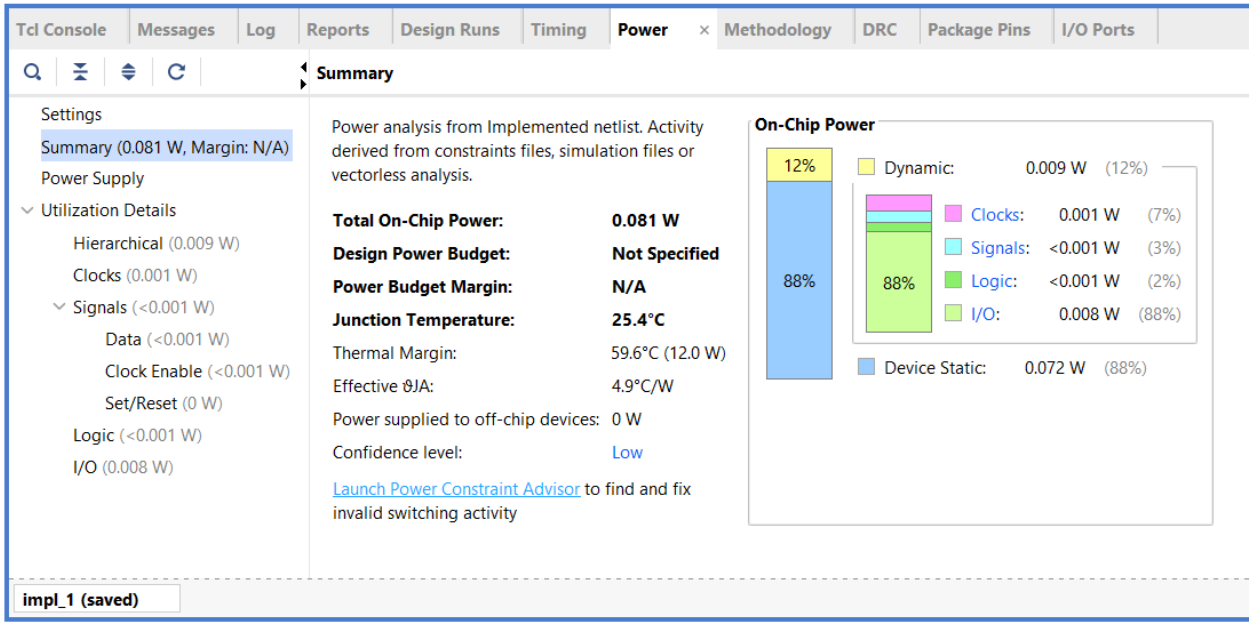


Output ^

Answers to post-lab questions

- **Fill in the table shown in 5.6 with your design's statistics.**

	Multiplier
LUT	77
DSP	0
Memory (BRAM)	0
Flip Flop	64
Frequency	0.24600
Static Power	0.07128W
Dynamic Power	0.00972W
Total Power	0.081W



- 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.

We can increase the maximum frequency by using a faster adder. In this case we are using the ripple adder, which is not necessarily the fastest adder according to lab 3. Changing it to select adder or carry lookahead adder could increase the maximum frequency.

- What is the purpose of the X register?

The X Register is to hold the sign bit of the result of addition between A sign extended to 9 bits and S sign extended to 9 bits.

- When does the X register get set/cleared?

X gets cleared after a multiplication cycle completes and run or reset is hit again

- What would happen if you used the carry out of an 8 bit adder instead of output of a 9 bit adder for X?

Say A = -S; A=1100 0000, S = 0100 0000. The carry out of an 8-bit adder is 1 (which is incorrect as the sign bit should be 0), whereas the MSB of the 9-bit adder is 0.

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

The implemented algorithm will fail when the output has significant information in bits [15:8]. As the multiplier is an 8x8 bit multiplier, when the multiplier is hit again the information in bits [15:8] will be lost resulting in a garbage answer. It will basically fail when it overflows, and the hex can't display the accurate result.

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

The advantage is that the algorithm uses less registers and logic gates as the implemented algorithm is tighter on storage. You only have to keep track of 2 8 bit values A, B and a one bit value X. As opposed to the eight 8 bit numbers to keep track of in the paper and pencil method. However, the disadvantage of this method (algorithm) is that we will need to have up to 20 states to control each add and shift operation. For the paper and pencil method, it is possible to just use all the different registers in a few states.

6. Conclusion

- **Discuss functionality of your design. If parts of your design did not work, discuss what could be done to fix it.**

  In this lab, we had the problem of an inferred latch when we demo. The reason why there is an inferred latch is we didn't cover all the cases in the control unit. In the control unit, we did a lot of if statement logics for the output and state transitions; however, we didn't manage to cover all the cases, and the way we wrote the default case was wrong. This made Vivado generate the inferred latch when it should be a mux. We need to make sure to define the outputs for all the conditions next time.

- **Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right, so it does not get changed.**

This lab is probably more vague than all the other ones since no files are provided and the only thing given is an algorithm. A suggestion is to specify what X is meant for.