ECE 385

Fall 2022

Experiment #4

Lab 4

An 8-bit Multiplier in System Verilog

Haocheng Yang((hy38)

Yicheng Zhou(yz69)

Introduction:

We designed and implemented a multiplier for 8-bit 2's complement multiplications. On the FPGA board 8 switches are used for loading in multiplicand, and result will be stored and displayed in 2 registers.

Prelab Question:

Process of 11000101*00000111 in a table like the example

Function	X	A	B	M	comments for next step
Clear A LOAD b	0	00000000	00000111	1	ADD
ADD	1	11000101	00000111	1	SHIFT
SHIFT	1	11100010	10000011	1	ADD
ADD	1	10100111	10000011	1	SHIFT
SHIFT	1	11010011	11000001	1	ADD
ADD	1	10011000	11000001	1	SHIFT
SHIFT	1	11001100	01100000	0	SHIFT
SHIFT	1	11100110	00110000	0	SHIFT
SHIFT	1	11110011	00011000	0	SHIFT
SHIFT	1	1111001	10001100	0	SHIFT
SHIFT	1	11111100	11000110	0	SHIFT
SHIFT	1	11111110	01100011	1	8th shift done

Table 1 table of pre-lab question

Description and Diagram:

Summary of operation:

Step 1: initializing program on FPGA

Step 2: input the first multiplier in 2's complement binary format using switch 0 to 7.

Step 3: Run Clear-A-Load-B by clicking Key 0, input is now stored in register B and Displayed in hex

Step 4: input the second multiplier in 2's complement binary format using switch 0 to 7.

Step 5: Run the program by pressing Execute button, result will be displayed as a 4 digits hex value on FPGA

Step 6_1: for consecutive calculations, input the next multiplier using switch 0 to 7 and click execute button, result will be displayed in 4-digit hex.

Step 6_2: for ending the program, lick Reset button, This will clear the registers for next operation

Top Level Block Diagram:

The RTL generated graph of top level block diagram.:

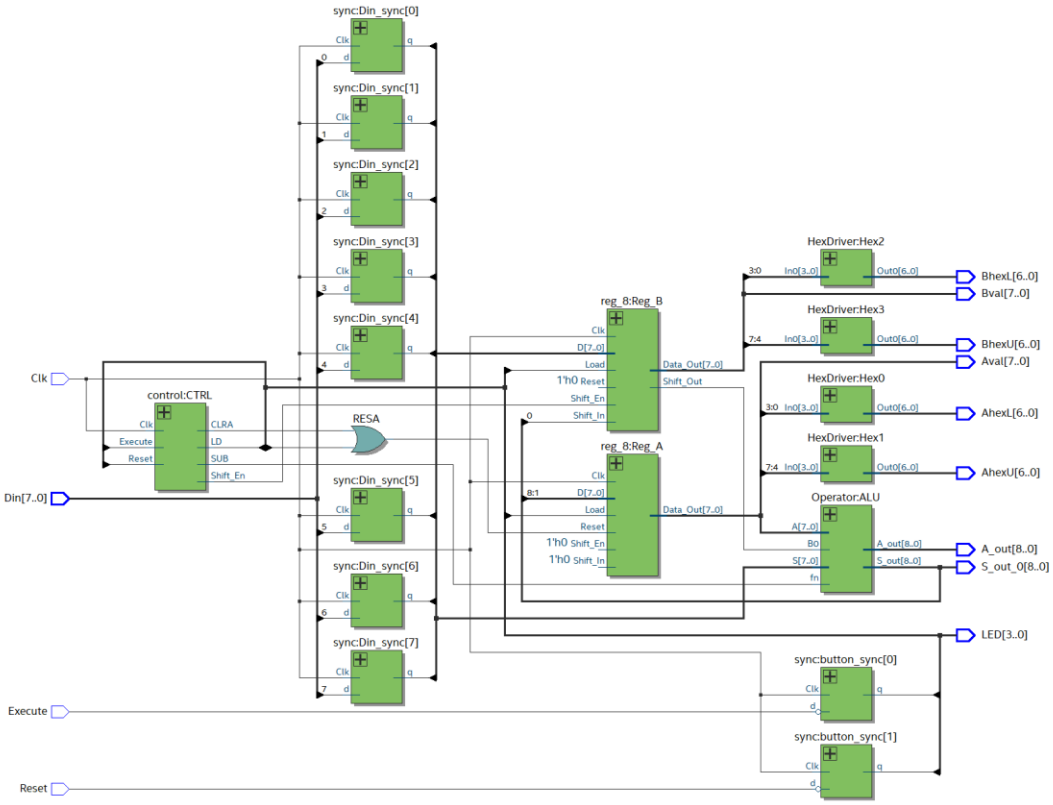

Figure 1: top level block diagram

Description of .sv modules:

a.) Module: synchronizers.sv
Inputs: Clk, d,
Output: q
Description: this module is used to synchronize user input to signals in the program. It will load button inputs to various programs at ever positive clock edge
Purpose: This is used to achieve shift_en and shift_out functions for rgister units.

b.) Module: Router.sv
Inputs:[1:0] R, A_In, B_In, F_A_B,
Outputs: A_Out, B_Out
Description: this module act like a dual 4:1 mux that routes signals to register units, the select signal for both mux(s) is R(2-bit), and it takes in the rest as inputs and outputs the selected signal.
Purpose: this is used to route the correct signal to the correct register for operations.

c.) Module: Reg_8.sv:
Inputs: Clk, Reset, Shift_In, Load, Shift_En,[7:0] D
Outputs: Shift_Out, [7:0] Data_Out.
Description: this is the register unit that based on input select signals(Reset, Shift_In, Load, Shift_En) will select the correct data to load into data_out.
Purpose: this module is used to achieve reset (loading 8'b0) and shifting operations for the registers, basically registers' control unit.

d.) Module: Register_unit.sv:
Inputs: Clk, Reset, A_In, B_In, Ld_A, Ld_B, [7:0]D,
Outputs: A_out, B_out, [7:0]A, [7:0]B,
Description: this is the registers for storing data, there are 2 8-bit registers.
Purpose: the register unit used in to processer.

e.) Module: Porcessor.sv:
Inputs: Clk, Reset, Execute, [7:0] Din
Outputs: [3:0] LED, [7:0]Aval, [7:0]Bval, [6:0]Ahexl, [6:0]Bhexl, [6:0]AhexU, [6:0]BhexU, [8:0] A_out, [8:0]S-out_0
Description: This is the top level entity that connects all modules mentioned here. It is the center for the port-port connection and prepares the final outputs to be displayed.
Purpose: this is the central module that connect and controls all modules, generating output to be translated by HexDriver.

f.) Module: Operator.sv:
Inputs: [7:0]S, [7:0]A,fn,b0
Outputs: [8:0]S_out, [8:0]A_out
Description: This module takes in S and A, along with command signal, to perform extension and call on adder 9 to perform adding operation.
Purpose: extending S and A to 9-bits and call on adder9.sv.

g.) Module: HexDriver.sv:
Inputs: [3:0]In0

Outputs:[6:0] Out0
Description: This module converts 4bit-binry input and convert it to 7 bit for hex-display.
Purpose: This module converts 4bit biary numbers from registers into hexadecimal so LEDs can display outputs.

h.)Module: Control.sv:
Inputs: Clk, Reset, Execute
Outputs: Shift_En, SUB, CLRA, LD
Description: This module is the control unit which is implemented as a Moore machine. There are 10 states in total. This module take in user inputs (Reset, Execute) and clock signal and run through predesigned states to issue Shift_En, SUB, CLRA, LD and halting program.
Purpose: This module is used to enable registers to perform shift, load operation for reg A and B in our system.

i.) Module: compute.sv:
Inputs: [2:0] F, A_In, B_In,
Outputs: A_Out, B_Out, F_A_B
Description: This is an implementation of 8:1 Mux, used to assign correct logic operation to F_A_B, and assign A & B inputs to outputs.
Purpose: This is used to determine A and B outputs for our system.

j.) Module: adder9.sv:
Inputs: [8:0] S9, [8:0]A9, fn.

Outputs: [8:0] S_out
Description: This module has two submodules, one of which is to perform xor and and/or bitwise logic to produce c and s per bit, the second of which is to call on the first submodule and "assemble" the bits by the first submodule and produce the 9-bit output.
Purpose: This module is to implement the S logic for each extended S(s)

k.)Module: 9_bit_adder.sv:
Inputs:[8:0] A, [8:0]B, c_in
Outputs: [8:0]S_out, x
Description: this module calls on adder9.sv
Purpose: this module is just a port from main system to the adder9 function.

State Diagram for Control Unit:

The state diagram are shown below

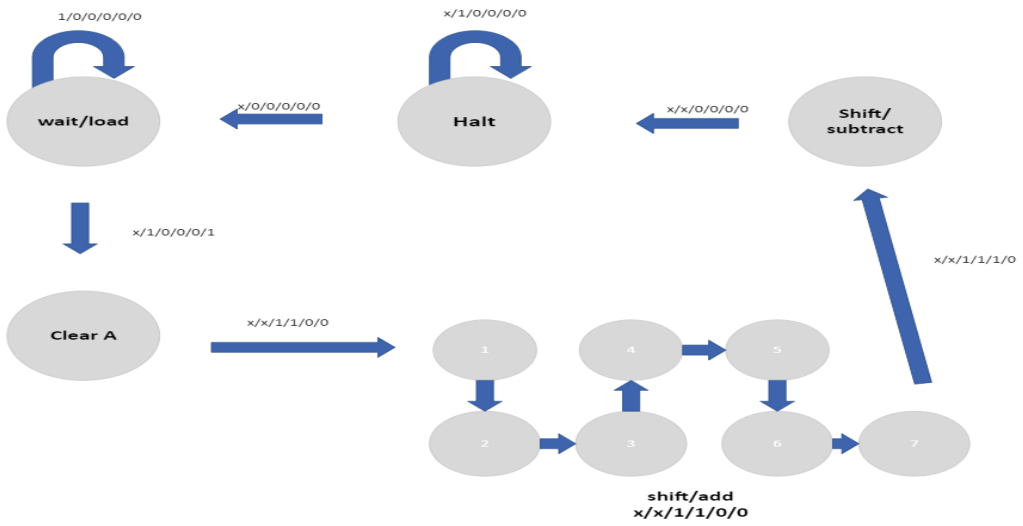

Figure 2 state diagram for control unit moore machine

Annotated pre-lab simulation waveforms:

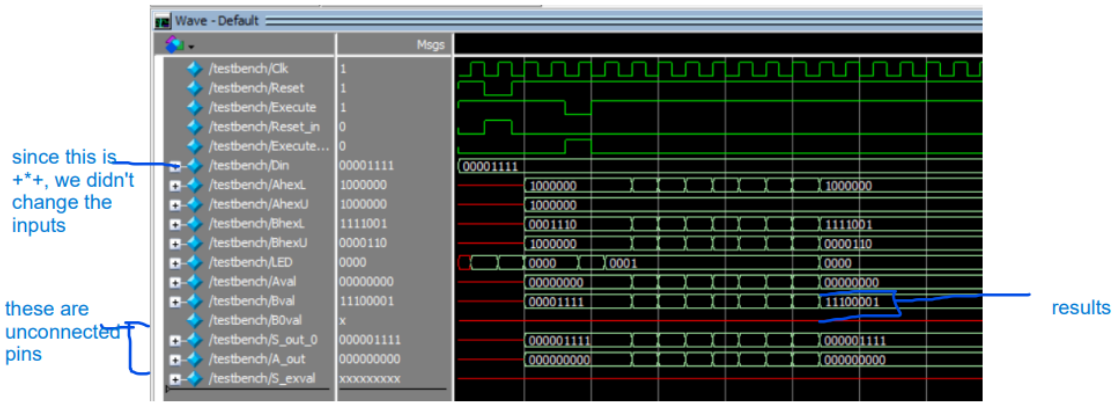

Figure 3, +*+ waveform

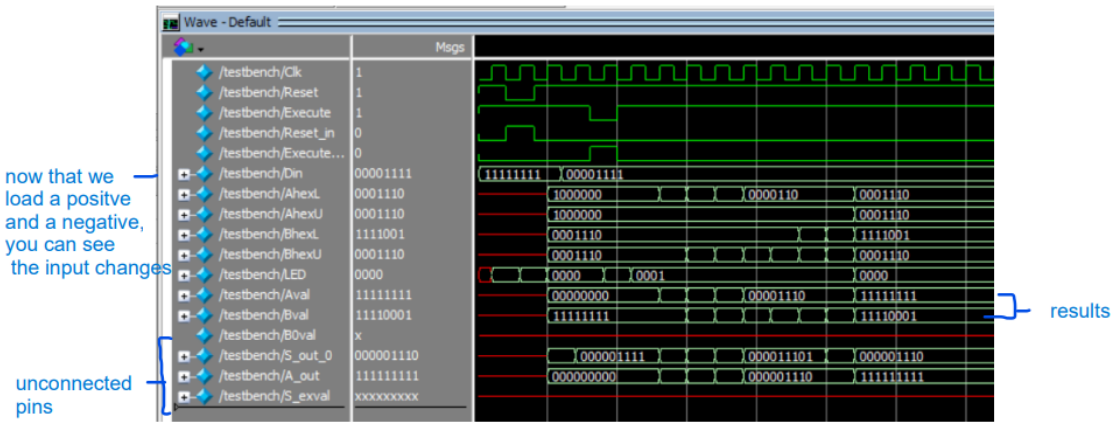

Figure 4, -*+ waveform

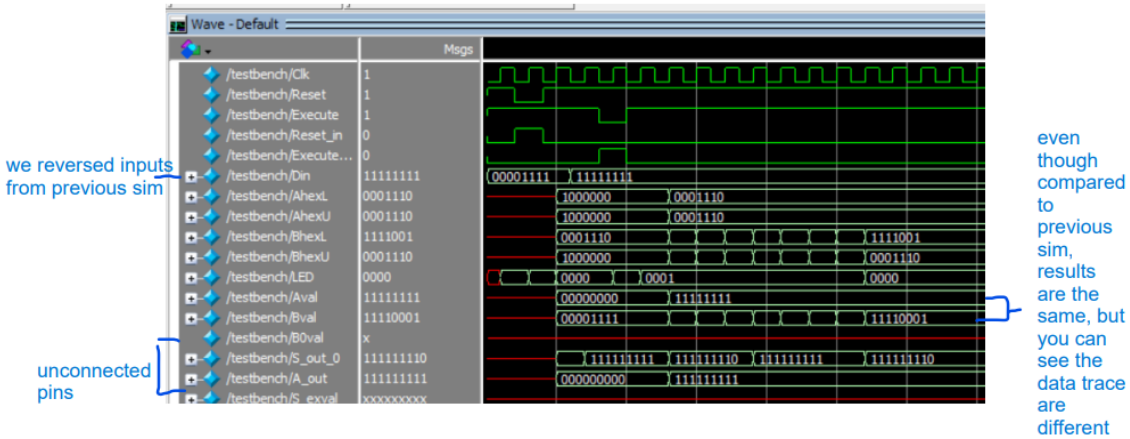

Figure 5, +*- waveform

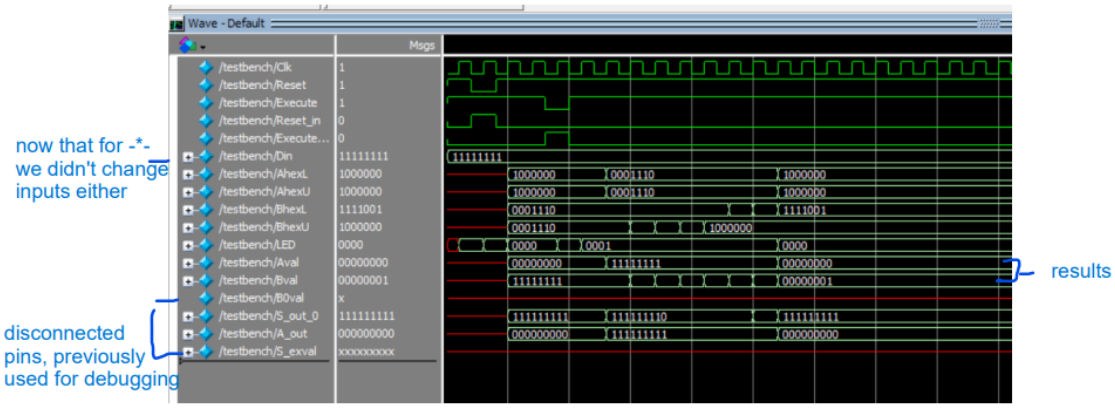

Figure 6, -*- waveform

Answers to post-lab question:

a.)_

LUT	84
DSP	0
Memory (BRAM)	0/1677312(0%)
Flip-Flop	36
Frequency	213.04 MHz
Static Power	89.94mW
Dynamic Power	0
Total Power	98.75mW

The adder used in this case is a ripple-carry implementation, with knowledge of previous labs, we can increase clock speed by switching that to carry-select adder.

b.)1. What is the purpose of the X register? When does the X register get set/cleared?

The X register is to sign-extend values, especially for A, so that the sign of register A will not change after a shift, X register would also stop data loss in overflow situations.

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

The carry out of an 8-bit adder does not represent the sign of the result, instead just an indication of overflow, and the result would be wrong since these are different values.

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

Since the design is for 8bits operations, we truncate the most significant 8 bit from 16bit result, thus if the result from previous operation exceeds 8-bit data will be lost and 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?

The implemented one only requires 2 register which offers less complexity.

Conclusion:

This lab gave us deeper understanding of the utilization and implications of bit-extension in algorithm. One example of this is that during our initial completion, we are testing for 3f*3, and we kept getting FFB9 instead of 02B9 which is what we initially expected. Then after much “debugging” to no avail, we went back

and re-read the documentation and repeated trace, and then realized that FFB9 is the right output all along.