Zhihao Wang (zhihaow6), Laurenz Nava (lfnava2)

Introduction

The multiplier circuit will take two 8-bit 2's complement inputs from the switches. After clearing the A and B registers, the first input is the multiplier and will be stored in the B register. Then the switches are flipped to the value of the multiplicand. Once the run button is pressed, the multiplication begins. Depending on the value in B, the multiplicand is either added to and stored in XA, or XAB is right-shifted once, where X shifts into the most significant bit of A, the least significant bit of A is shifted into the most significant bit of B, and the least significant bit of B is dropped. After 8 total shifts, the multiplication is finished and the resulting number stored in AB is the product of A and B

Pre-lab Questions

Function	X	A 1100 0101	B	M	Comments for next step
Clear A, Load B, Reset	0	0000 0000	0000 0111	1	$M = 1 \rightarrow S + A$
ADD	1	1100 0101	0000 0111	1	Shift XAB after ADD
SHIFT	1	1110 0010	1000 0011	1	$M = 1 \rightarrow S + A$
ADD	1	1010 0111	1000 0011	1	Shift XAB after ADD
SHIFT	1	1101 0011	1100 0001	1	$M = 1 \rightarrow S + A$
ADD	1	1001 1000	1100 0001	1	Shift XAB after ADD
SHIFT	1	1100 1100	0110 0000	0	$M = 0 \rightarrow$ Shift XAB
SHIFT	1	1110 0110	0011 0000	0	$M = 0 \rightarrow$ Shift XAB
SHIFT	1	1111 0011	0001 1000	0	$M = 0 \rightarrow$ Shift XAB
SHIFT	1	1111 1001	1000 1100	0	$M = 0 \rightarrow$ Shift XAB
SHIFT	1	1111 1100	1100 0110	0	$M = 0 \rightarrow$ Shift XAB
SHIFT	1	1111 1110	0110 0011	1	8th Shift Complete, 16-bit product done

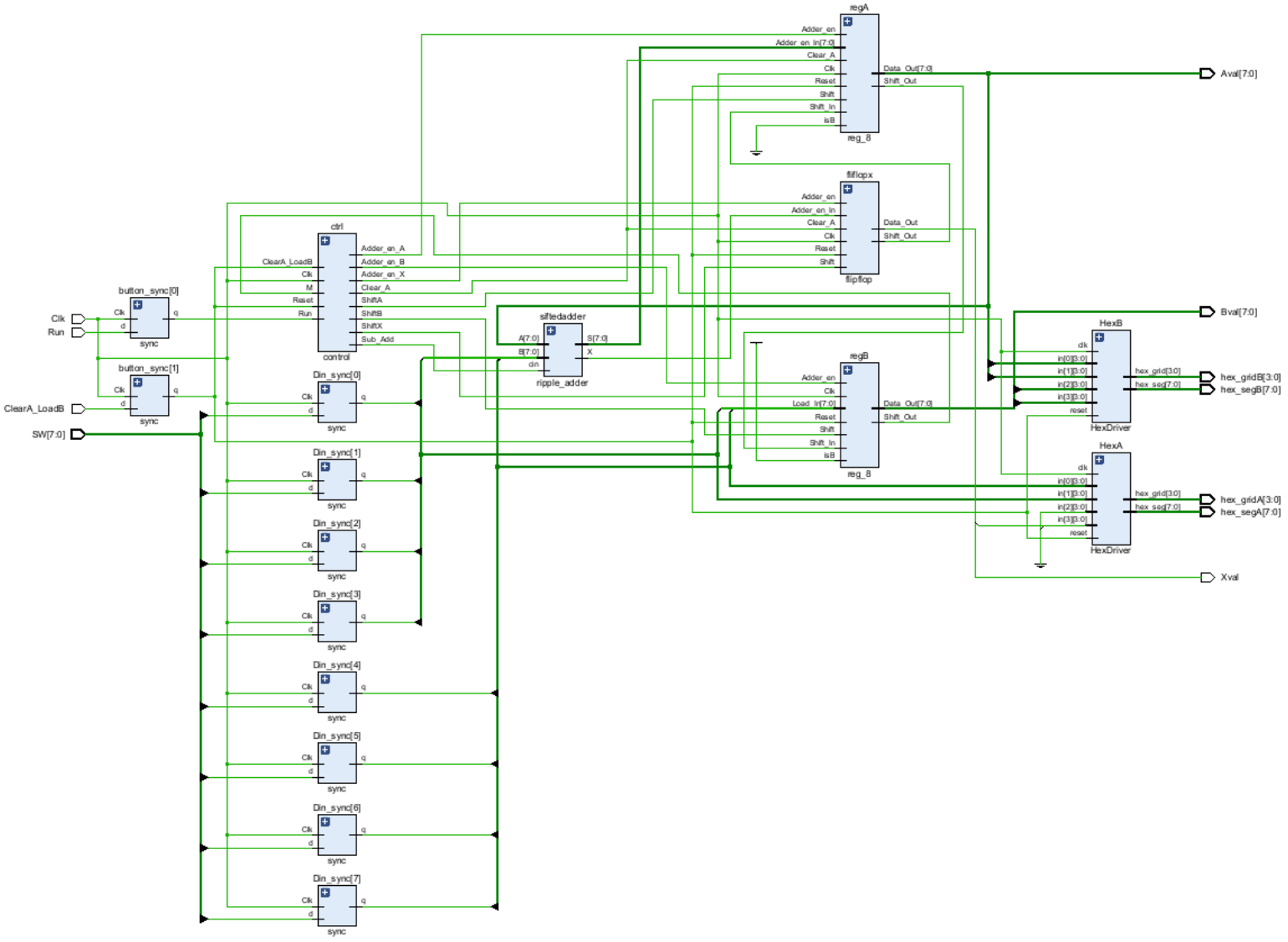
Written Description and Diagrams of Multiplier Circuit

Summary of Operations

The operation of the 8-bit multiplier begins at the inputs. The switches are flipped to a binary number and the reset button is pressed to clear XA, load the switches into B, which is the multiplier, and reset the state to the beginning. Next, the switches are flipped

to the value of the multiplicand and the run button is pressed to begin the multiplication operation. The next step is determined by M, which is the least significant bit in B. If M is 0, then the bits in XAB, the combined bits in that order, are all right-shifted once, dropping the least significant bit. If M is 1, the switch value, or multiplicand, is added to A. If A becomes negative, X becomes 1. Then the bits in XAB are all right-shifted once. If the most significant bit of the original B is 1, so after 7 right-shifts M is 1, the multiplicand is subtracted from A. This only occurs if B is negative. This repeats until all the bits in the original B are dropped, in other words when the right-shift has occurred exactly 8 times.

Top Level Block Diagram



Written Description of SV files

Module: processor.sv
Inputs: Clk, ClearA_LoadB, Run, [7:0] SW
Outputs: [7:0] Aval (Debug), [7:0] Bval (Debug), Xval, [7:0] hex_segA, [3:0] hex_gridA, [7:0] hex_segB, [3:0] hex_gridB
Description: Creates registers for A and B, and a flip flop for X, sorts the data to go to the proper place, and displays the values of A and B
Purpose: Perform the logic described above and display the inputs and products to user

Module: reg_8.sv
Inputs: Clk, Reset, Shift_In, Shift, Adder_en, isB, Clear_A, [7:0] Load_In, [7:0] Adder_en_In
Outputs: Shift_Out, [7:0] Data_Out
Description: Synchronous shift register, if Reset is high, isB is loaded with Load_In, if not isB register cleared. If Shift is high, add Shift_In from left.
Purpose: Store B, the multiplier, XAB, the intermediate sum, and AB, the final product

Module: flipflop, reg_8.sv
Inputs: Clk, Reset, Shift_In, Shift, Adder_en, Clear_A, Load_In, Adder_en_In
Outputs: Shift_Out, Data_Out
Description: Single bit storage, able to shift in and out bits
Purpose: Store X bit for calculations

Module: control.sv
Inputs: Clk, Run, ClearA_LoadB, Reset, M
Outputs: Shift_A, Adder_en_A, Shift_B, Adder_en_B, Shift_X, Adder_en_X, Sub_Add, Clear_A
Description: This is a positive edge Clk triggered to set the next state. ClearA_LoadB and Reset to clear registers. M decides the next state (Add/Sub or Shift). Decides the bit to shift in for X based on sum XA (1 if negative, otherwise 0), for A from X, and for B from A.
Purpose: Manage states for timing and order. Determines next state and bits to shift in

Module: single_cra, ripple_adder.sv
Inputs: A, B, Control, Ci
Outputs: Co, S
Description: Adds A, B, Ci to produce S and Co, if Control high converts sum to difference
Purpose: Compute sum or difference of 1-bit numbers

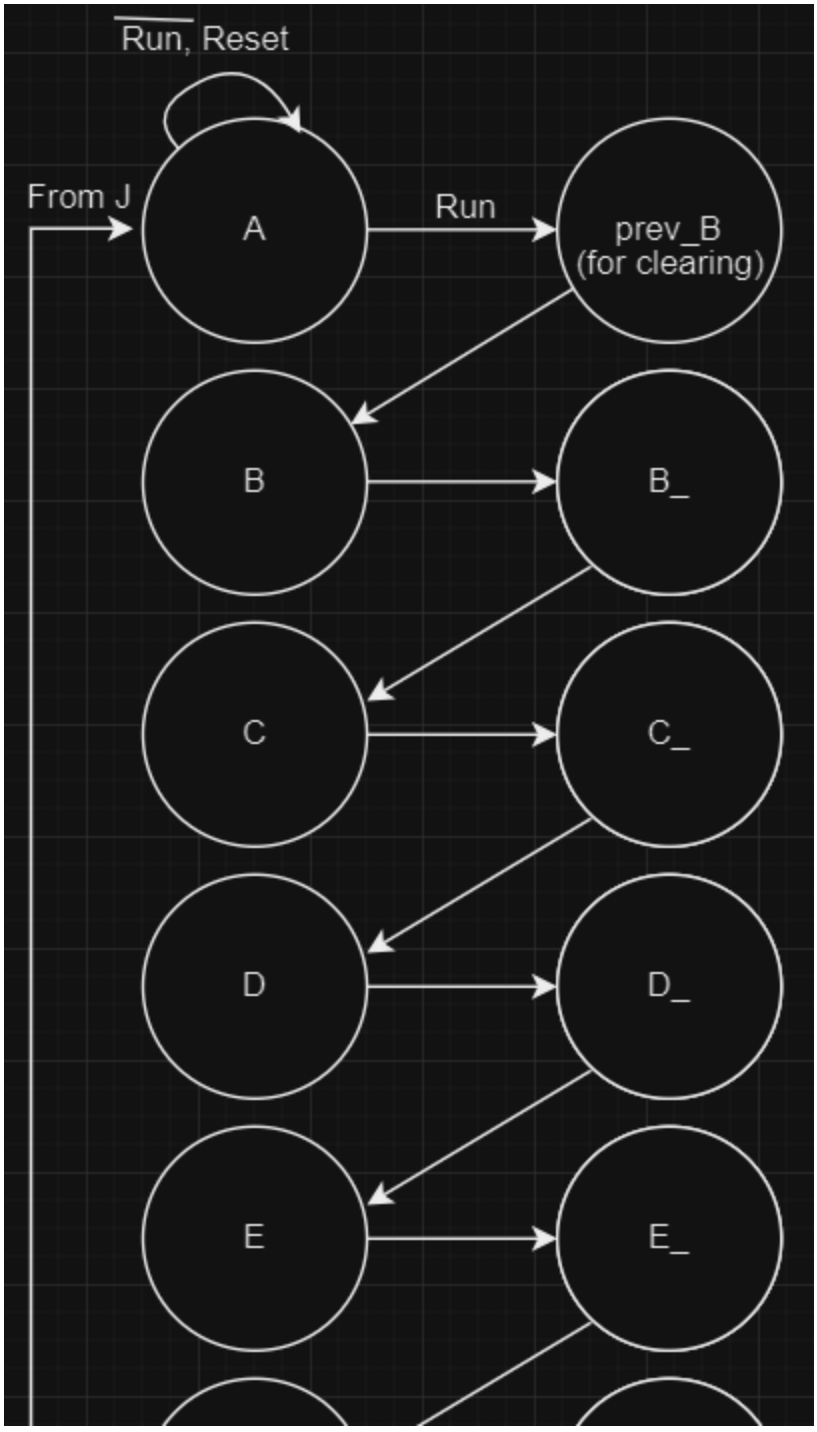
Module: ripple_adder.sv
Inputs: [7:0] A, [7:0] B, cin
Outputs: [7:0] S, X, cout
Description: Cascades 8 single_cra to produce 8-bit S, and one more to produce X
Purpose: Compute sum XA from A and SW

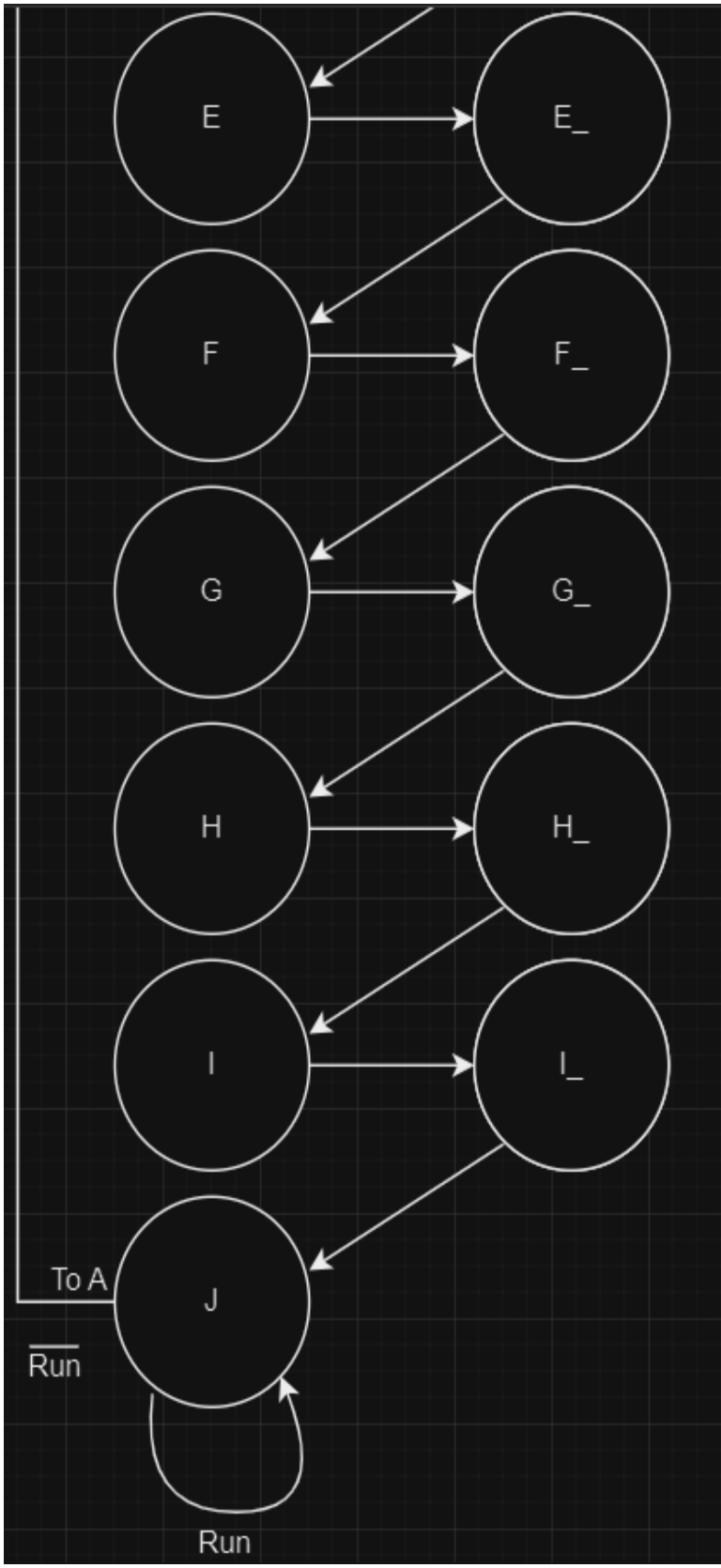
Module: Hex Driver, hex.sv
Inputs: clk, reset, [3:0] in[4]
Outputs: [7:0] hex_seg, [3:0] hex_grid
Description: Convert input data into displayable data, then display on 7 segment display
Purpose: Display input data on 7 segment display

Module: nibble to hex, hex.sv
Inputs: [3:0] nibble
Outputs: [7:0] hex
Description: Turns the 4-bit nibble input into an 8-bit output readable by the hex segments to display the hexadecimal value of the input
Purpose: Convert binary input data into displayable on 7 segment display output data

Module: sync, Synchronizers.sv
Inputs: Clk, d
Outputs: q
Description: Synchronous update of q to d
Purpose: Synchronize inputs from physical devices, switches and buttons, to internal timings

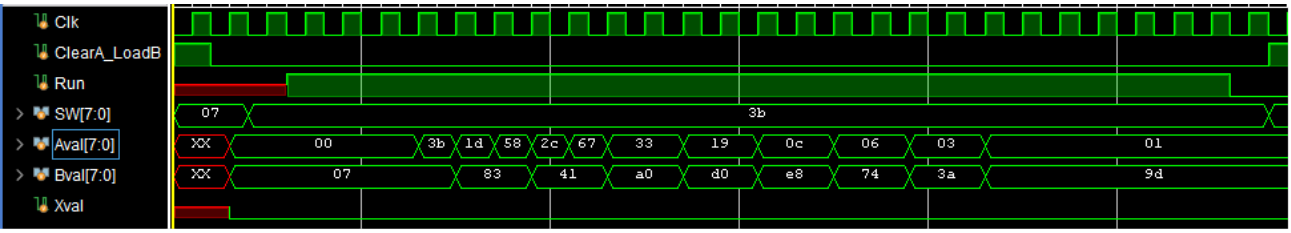
State Diagram for Control Unit



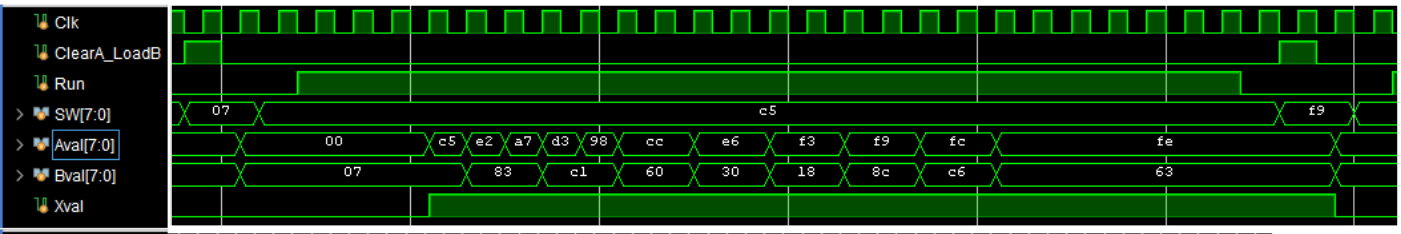


All unmarked state transitions occur
on Clk cycle.
All states will point back to A if **Reset = 1**.
We did not draw in the FSM graph since
Our graph is too compact.

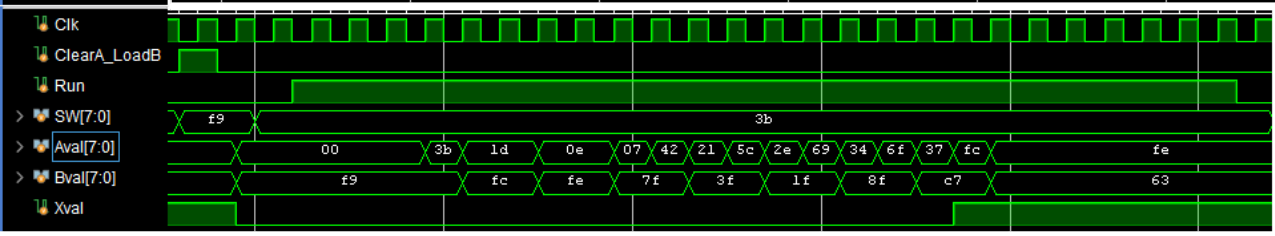
Annotated Pre-lab Simulation Waveforms



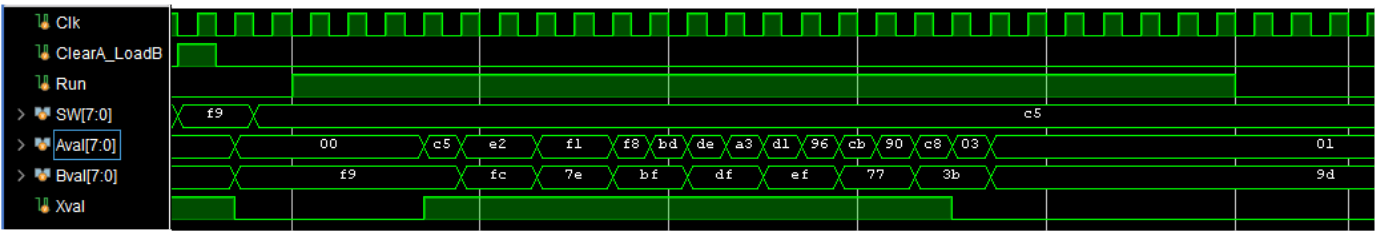
7 × 59 = 413
07 × 3B = 0190



7 × -59 = -413
07 × C5 = FE63



-7 × 59 = -413
F9 × 3B = FE63



-7 × -59 = 413
F9 × C5 = 0190

Post-lab Questions

LUT	79
DSP	0
Memory (BRAM)	0
Flip-Flop	71
Latches	0
Frequency	213.858 MHz (5.324 ns WNS)
Static Power	0.072 W
Dynamic Power	0.023 W
Total Power	0.094 W*

*I know the total power doesn't add up, it's just what was shown on the power report, must be something with the rounding

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

The X register is to determine if the value in XA is negative or not. At the beginning of the operation, X is cleared with A and is set to 0. X can then be set to 1 if the value in the switches, or the multiplicand is negative, or kept at zero if the multiplicand is positive. If at the final bit of B there is a 1 to indicate a SUB, and the multiplicand is positive, X is set to 1, or if the multiplicand is negative, X is set to 0.

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

It is possible for X to not be the correct value if an 8-bit adder is used in place of a 9-bit adder. For example, for the first ADD of a negative multiplicand, there will not be a carry out (1000 0000 + 0000 0000 = 1000 0000, no carry out). This is not correct because in this case, X should be 1 to indicate that XA is now negative.

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

Because the output numbers are represented in 2's complement 16-bit numbers, whose range is [-32768, 32767], but the input numbers are represented in 2's complement 8-bit numbers, whose range is [-128, 127], using the output as the next input only works if the output number is still within the input number range. For example, successive Multiplications of -1 to 1 and back to -1 are okay because it stays within the input range, but Continuous multiplications of 2 will overflow once the output is 128, as this can only requires 17-bits for 2's complement representation (1x00), and will be truncated to x00, which is just 0.

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

One way to consider the advantages of the FPGA implementation over the pencil-and-paper Method is that the FPGA does not need to compute the intermediate products since it is a computer and will always add perfectly, so the design does not need to store those extraneous products to then sum up later. Let's make an assumption on the length of the operand to be 'n'. The total time complexity for calculating all the intermediate values will be O(N^2). And we must spend another O(N^2) for adding them up. It is a two-pass algorithm. Further the space complexity will also be O(N^2). For the current strategy, the worst time complexity will be O(N^2) for adding them when all bits of B are 1. But it is a one-pass algorithm. Also for the space complexity it will be down to O(N).

One potential disadvantage is that, by the nature of the inputs and outputs, there is a limited range of numbers where the FPGA implementation is applicable, while the pencil-and-paper method can be expanded for as large as the inputs and outputs need to be.

Conclusions

The design of the multiplier works well in all specified use cases, so no issues there. The lab manual in its current state for the design and understanding of the lab, and, we personally believe, should not be changed too much.