8-bit 2's Complement Multiplier

1. Introduction:

In this lab, we implemented an 8-bit 2's complement binary multiplier that takes two 8-bit numbers and outputs a 16-bit number. Since it's a 2's complement multiplier, it's capable of multiplying positive times positive, negative times positive, positive times negative, and negative times negative numbers, as well as consecutive multiplications. We implemented this multiplier through a series of additions and shifts, which are determined by specific signals such as M and X, as shown in Table 1 below.

2. Demonstration of Multiplication Process:

Compute 11000101 * 00000111

Function	X	A	B	M	Next Step Comments
Clear A, LoadB, Reset	0	0000 0000	*00000111*	1	M=1, multiplicand will be added to A from switches
ADD	1	1100 0101	*00000111*	1	Shift XAB by one bit after ADD
SHIFT	1	1110 0010	1 *0000011*	1	Add switches to A since M=1
ADD	1	1010 0111	1 *0000011*	1	Shift XAB by one bit after ADD
SHIFT	1	1101 0011	11 *000001*	1	Add switches to A since M=1
ADD	1	1001 1000	11 *000001*	1	Shift XAB by one bit after ADD
SHIFT	1	1100 1100	011 *00000*	0	Do not add switches to A since M=0, only Shift XAB
SHIFT	1	1110 0110	0011 *0000*	0	Do not add switches to A since M=0, only Shift XAB
SHIFT	1	1111 0011	00011 *000*	0	Do not add switches to A since M=0, only Shift XAB
SHIFT	1	1111 1001	100011 *00*	0	Do not add switches to A since M=0, only Shift XAB
SHIFT	1	1111 1100	1100011 *0*	0	Do not add switches to A since M=0, only Shift XAB
SHIFT	1	1111 1110	01100011	1	8th shift done. Stop. 16-bit product in AB.

Table 1: The computation of multiplying (-59, multiplicand) by (7, multiplier)

3. Description:

Summary of Operation

The multiplication process begins by loading register B (the multiplier) with the desired value by setting up the switches on the FPGA board and then pressing the reset button. The reset button loads the value of the switches into register B and clears registers A and X, setting up the registers for a new multiplication cycle. Then, switches, which are the multiplicand, are set to the desired value in order to perform the multiplication operation. By following this process, we can press the run button and expect to see the result of switches times B displayed on the hex displays on the FPGA board, where the left two squares display the value of A, and the right two squares show the value of B, and AB together demonstrates the 16-bit value of the multiplication. Once the run button is pressed, the control unit takes control and performs the multiplication.

The control unit performs multiplication through a series of additions and shifts depending on internal signals, which are M (the least significant bit of register B), reset button, and run button. And outputs signals that control shifting registers X, A, and B (shift_en), loading X, A, and B (add_sub), clearing X and A and loading B (ld_clear), clearing X and A (clear), and lastly controlling the carry in for the 9-bit adder (comp) as shown in Figure 1. The default value for all signals is set to zero.

As soon as the run button is pressed, the control unit moves to the add_0 state, as shown in Figure 3; in all the add states (except add_7), the control unit checks the M bit; if it is equal to 1, the add_sub signal is set to 1; otherwise, it is 0. In the add_7 state, the control unit checks the M bit; if it is equal to 1, the comp and add_sub signals are set to 1; otherwise, they are set to 0. Furthermore, all add states always go to their corresponding shift state where the shift_en signal is set to 1. Moreover, after the last shift state (shift_7), the next state is always the halt state, whereas, as the name implies, all signals are set to 0, so the multiplier is halted. There are two possible transitions from the halt state, which are based on the run button and reset button. If the run button is pressed, the next state is clear, where the clear signal is set to 1 so that registers X and A are cleared and ready to perform consecutive multiplication. Otherwise, if the reset button is pressed, the next state is reset where ld_clear signal is set to 1 so that registers X and A are cleared and register B gets loaded with the switches value so that the multiplier is ready to perform a new multiplication. Otherwise, the control unit stays in the halt state.

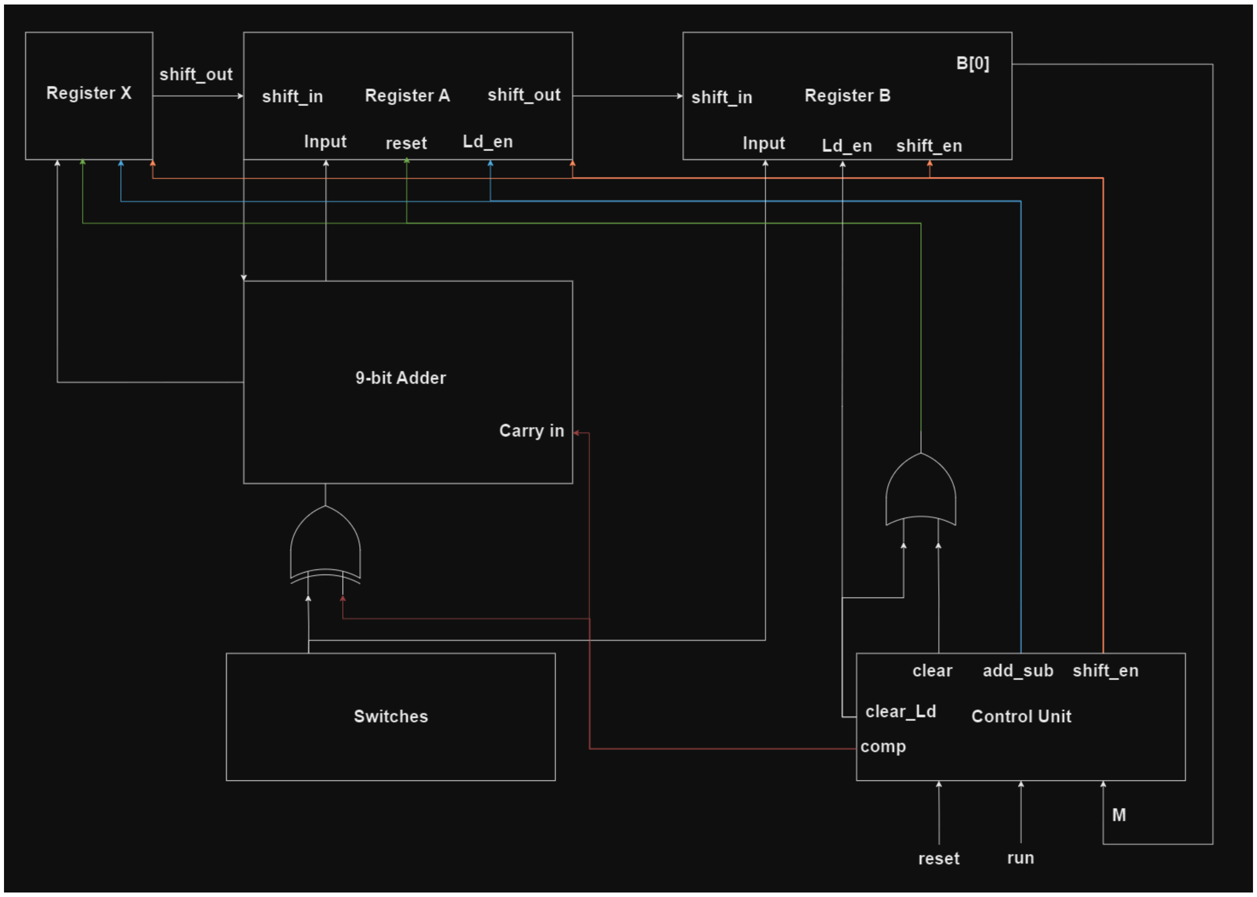

Figure 1: High-level block diagram of the Multiplier

Top Level Block Diagram

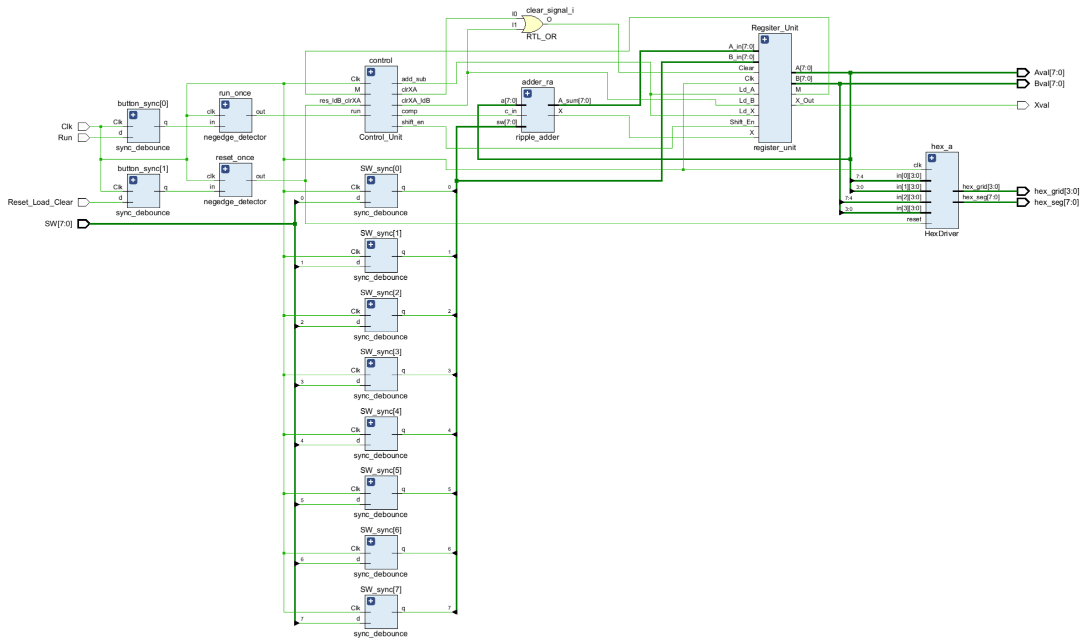

Figure 2: Top Level Block Diagram from Vivado

State Diagram

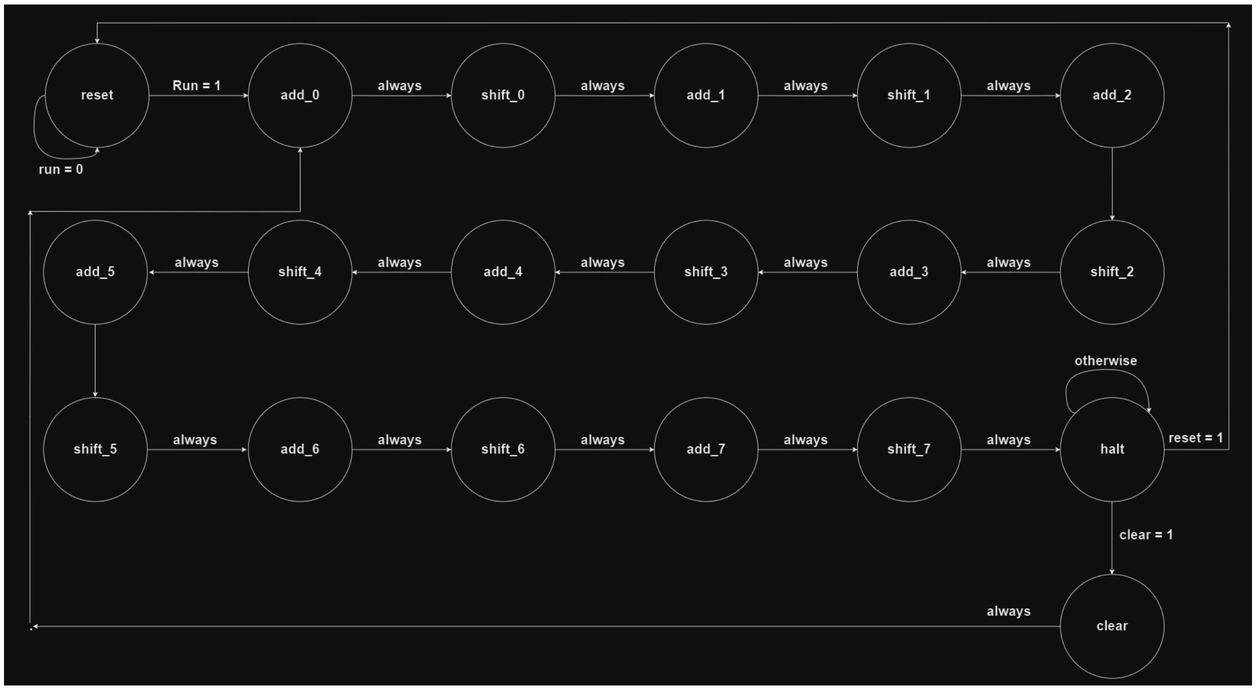

Figure 3: State Transition Diagram for Control Unit

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

The X register preserves the sign of added value to register A, so it acts as a shift in bit for register A whenever it gets shifted. The X register is set whenever an addition or subtraction occurs to register A since X is the result of the operation of register A's MSB with its carry-out as carry-in for X's result. It gets clear whenever Reset is pressed or when a consecutive multiplication occurs.

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

If the input to the X register was wired to the carry-out of the 8'th bit adder, X would not represent the Most Significant Bit (MSB) of A accurately in some cases. For example, if A was zero and S was a negative number, the carry-out of adding 1 and 0 would not be 1, which does not represent 2's complement negative numbers accurately. This leads to errors when performing arithmetic shift right because X does not accurately represent the MSB of A.

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

The limitations of continuous multiplications arise when the product exceeds the representable range of 16 bits in 2’s complement. Since both operands are 8 bits, if the result of the multiplication is larger than what can be stored in 16 bits, the result will overflow, leading to an incorrect outcome.

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

The pencil-and-paper method requires much more memory to store all the subproducts and add them later together, while the implemented method only requires the two input registers with only the addition of the X register. On the other hand, the implemented method is less straightforward and needs much more attention to each step to know when to add or just shift.

4. Conclusion:

In this lab, we designed an 8-bit 2’s complement multiplier using sequential logic on an FPGA. The project allowed us to explore how hardware can handle multiplication operations through a series of controlled shifts and additions. Our design effectively computes the product of signed 8-bit numbers and outputs a 16-bit result, demonstrating the functionality for both positive and negative operands. Through simulation and hardware testing, we verified the accuracy of our design; however, we found that it failed to multiply two large negative numbers (in magnitude). We were able to fix this by looking at the RTL schematic for the X register. We found out that the load enable to the X register was decided by a 2:1 mux where one of the inputs was 1, and the other was the shift enable signal. We fixed this after trying to test the X register without the shift enable signal when we realized we didn’t need one, and the load enable signal was enough.