**Lab Two: Booths Algorithm Implementation Report**

**Part 1:**

The behavioral design was a straightforward implementation of the logic required to perform booths algorithm. Hex Keyboards were used to set the multiplicand (X) and multiplier (Y) and register into their own independent shifter blocks. A Shifter was also used to store the accumulator. The outputs of the accumulator and multiplier are inputs to the provided adder-subtractor. The add-subtractor runs every step and it uses an A/S signal to determine its operation.

The logic to this input to the A/S single is dependent on the least significant bit of the multiplier (now referred to as mLSB) and on the right shifted bit appended to the multiplier (now referred to as Qneg) at the beginning of the algorithm. This appended bit is stored in a D Flip Flop. The negation of Qneg and the mLSB are put through an adder to determine whether it should be a one or a zero. An output in the and gate is 1 if Qneg is 0 and mLSB is 1, and 0 otherwise therefore is will always perform the necessary operation when a load step is encountered in the accumulator.

The load step logic on the accumulator is an XOR of mLSB and Qneg ANDed with the bit determining the load step from the controller, LoadA. When the and gate of ANDed gate turns on it means that it is on a load step and an operation has been performed in the current step which tells the accumulator to update its outputs.

The Shift in to the accumulator at this stage was the output of a second D flip flop that stored the 17th bit carry out of the add/subtract operation. Whenever a shift step occurs this D Flip Flop would shift its output into the accumulator. The output of this flip flop only changed during any step which was an operation and this gets updated whenever the accumulator is also being updated.

All of these parts work with the management of the controller and when the clock reaches its last pulse all outputs are display in hex on a hex display.

**Part 2:**

After staring at booths algorithm for a while, it is possible to see that the 17th bit of the addition/subtraction step does not need to be saved to perform booths algorithm successfully. This is because the 17th bit is always equal to the most significant bit of the partial product from when the previous operation was performed. The equality of this is dependent upon the nature of the shift operation in booths algorithm, namely that it is always an arithmetic right shift which always shifts in the most significant bit of the binary string. Given this the circuit was optimized to not use the 17th bit of the adder-subtractor. Another resulting step to this was to delete the second flip flop storing this extra bit as it did not exist in the algorithms dependent steps. The shift in of the accumulator is then the most significant bit of its own output. The only downside to this is that it does not account for the problem of overflow.

**Part 3:**

1. The input combinations that product correct and incorrect outputs are dependent on the overflow of the additions and subtractions of the partial products which may or may not incorrectly effect what is shifted in. For example, a 4-bit hex representation of this problem would show you that 8000\*8000=C0000000 when the correct product should have been 40000000. Whereas 8000\*0001=8000 as expected. This is because the inputs which give the partial product errors are the ones where overflow occurs.

2. The main driving rules when govern the behavior of the sign bit are dependent upon whether or not an operation is being performed and if an operation is being performed, then determine the value of the new next most significant bit. If there is no operation that is being performed this algorithm must always do an arithmetic shift, this is done by shifting in the most significant bit of the accumulator as was done in part two. If there is an operation being performed there is logic which governs whether or not it is an addition or a subtraction. This logic was determined by the value of the most significant bit of the multiplicand, Qneg and mLSB. If a subtraction step was detected and output to be shifted in is the negation of the most significant bit of the multiplicand, if it is an addition step this output is instead equal to the most significant bit of the multiplicand. This emulators the shift in behavior of booths algorithm for every input. What this reveals about the nature of the error occurring is that there are cases in overflow of the addition step which need special handling.

3. Every output and every input in each step of the algorithm is always going to be a zero so that means the product must also always be zero in each of its outputs at the end as well. This behavior is correctly handled in each part of the circuit. It does not matter whether or not zero is the multiplier or the multiplicand. If it is the multiplier is zero and the multiplicand is nonzero, then there is a noOp on overy step and therefore a zero is shifted in for every zero in the accumulator yielding a result of zero. On the other have if the multiplicand is zero, and the multiplier is nonzero, the partial product of the accumulator will always be zero and therefore a zero is always shifted in which yields a result of zero.

**Part 4:**

Translating the logic works schematic into the Verilog language was straightforward once the nuances of the language were learned. Every wire in my schematic was a wire variable in my language. Each component was translated into a module with corresponding logic to make the module work as intended. The inputs and outputs of the modules were instantiated at with some wires as inputs just as in the logic works schematic. Logic operators such as and, or and xor were also used in the syntax to simulate the necessary values for many of these inputs as well. In this sense, there were virtually no differences. My part 4 implementation compiled and the logic looks correct, but I ran out of time and was not able to test it in Modelism.