The purpose of this fourth homework assignment is to provide an experience for designing a multiplication method, known as the Booth Multiplier.

**System Design**

The Datapath of the *BoothMultiplier* can be found below:



The *Datapath* contains multiple registers. A, M, and Q are 16 bits, and Qr is 1 bit. The last bit of Q and Qr is the 2’bit window of the multiplier (Q) that is being analyzed at each step of the Booth algorithm. The counter is enabled by the Controller, and allows for a total of 16 operations (add/subtract & shift) to occur before the answer is output. Depending on the bit sequence seen, the ALU will take in the multiplicand, M, and will add/subtract it from the current value of A, and then will perform a shift. All operations happen on the positive edge of the clock, as these are all time-dependant operations. The Controller is ultimately responsible for setting all the different control signals. The Verilog logic for the ALU is seen below:

*// Create ALU logic:*

*always @ (posedge clk) begin*

*// If we are in the process of multiplying:*

*if (busy) begin*

*// Perform a case statement on the current Q 2'b window:*

*case ({Q[0], Qr})*

*// If a 01 is seen, add M to A and perform and shift.*

*2'b0\_1: begin*

*res = A + M;*

*{A, Q, Qr} <= {res[15], res, Q};*

*end*

*// If a 10 is seen, add -M to A and perform and shift.*

*2'b1\_0: begin*

*res = A + (~M) + 1;*

*{A, Q, Qr} <= {res[15], res, Q};*

*end*

*// Otherwise, perform no arithmetic and shift.*

*default: {A, Q, Qr} <= {A[15], A, Q};*

*endcase*

*end*

*end*

In order for the Datapath to operate autonomously and for all signals to be set correctly, a simple Controller is developed for this purpose. Note that the Huffman style of programming was not needed here. This is the case because it is assumed that the multiplier will begin operation on the first pulse of *start*, and will continue for a predefined number of clock pulses. It’s important to note that with the LSB bit of A shifting into the MSB of Q with each shift, Q will end up with all right shifted values of A after the 16 *clk* pulses. Therefore, *prod = {A, Q}* at the end, with the sign bit of A always being preserved.

The Controller of the *BoothMultiplier* can be found below:



The Controller consists of a simple three states. The *BoothMultiplier* remains in the `Idle state until a full pulse is seen on *start*. Once that occurs, the `Init state loads all registers with either external input (*mc, mp*) or with 0’s. Once the Controller enters the `Count state, it remains there until it performs the necessary number of operations and shifts. It then returns to the `Idle state and awaits another pulse on *start* for operation to begin once again.

**Verilog Code**

All Verilog code in this Booth Multiplier design, with detailed comments that go into detail regarding the definition of all lines of code, can be found in the *\_Code* folder of the HW 4 ZIP file submitted for this assignment:

*Controller.v, Datapath.v, BoothMultiplier.v, BoothMultiplier \_Tester.v*

**Testbench Confirmation**

A quick testbench was developed in an effort to characterize the 3 different two-complement scenarios that could happen with the *BoothMultipier*:

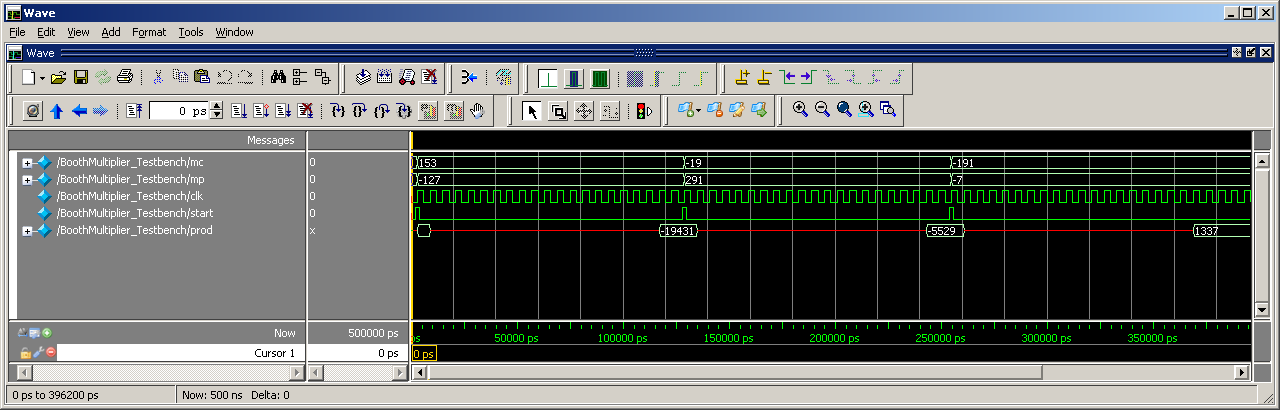
Case 1: Negative \* Positive = Negative

Case 2: Positive \* Negative = Negative

Case 3: Negative \* Negative = Positive

The following Waveform output displays all three cases:

**Waveform Analysis for Booth Multiplier**



**Case 1:** mc = 153 **CORRECT**

mp = -127

prod = -19431 (+ 18 clks: 1 for `Idle, 1 for `Init, 16 for `Count)

**Case 2:** mc = -19 **CORRECT**

mp = 291

prod = -5529 (+ 18 clks: 1 for `Idle, 1 for `Init, 16 for `Count)

**Case 3:** mc = =191 **CORRECT**

mp = -7

prod = 1337 (+ 18 clks: 1 for `Idle, 1 for `Init, 16 for `Count)

Thus, all functionality for the *BoothMultiplier* has been verified.

This concludes the analysis for Homework 04.