Joshua Pollock

EE 310 – Lab 5 Report

NAU, 8 March 2020

Problem Description

In this lab, we have been asked to design a synchronous unsigned 4-bit multiplier. The circuit will use a state machine to complete the multiplication operation without using the built-in multiplication. To do this we will be using two 4-bit inputs, A and B, and multiply them together which will result in an 8-bit register M. This M output will be decoded and then shown on two seven-segment displays. This multiplication operation will follow a 5-state state machine and the positive edge of a clock input. There will also be a reset input to reset the state machine to state 0.

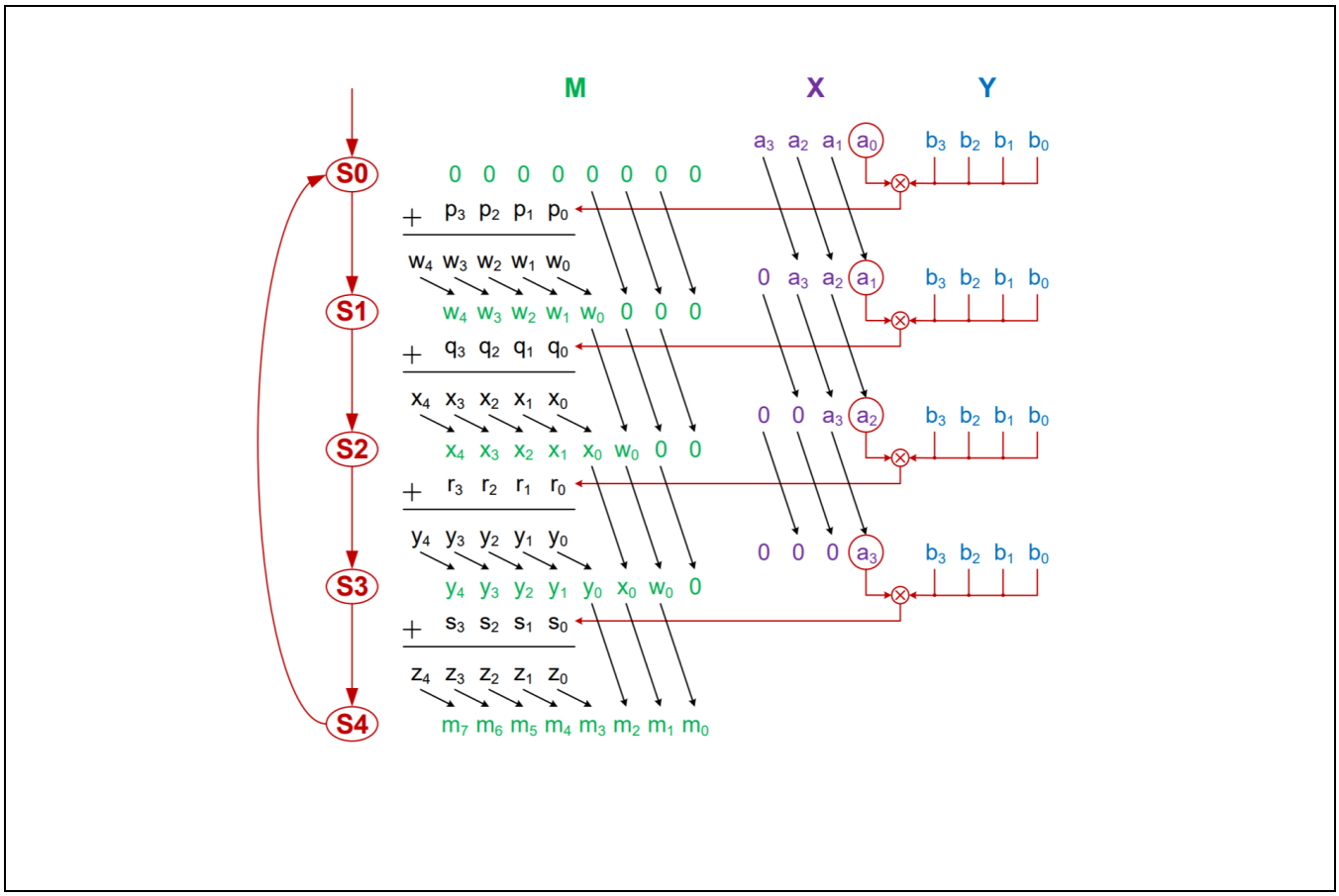

Figure 1. Expected behavior of the circuit

Solution Plan

In order to solve the problem explained above, I used the code from Lab 1 to create the skeleton of a state machine. The zeroth state set M to 0, X to input A, and Y to input B. In state 1, a temporary variable was set to M[7:4] + X[0]*Y. I know that this line has the multiplication operation in it but I couldn't figure out how to get the multiplication working using the other way. Next M[2:0] was set to M[3:1]. This line could be changed into a right shift operation to improve readability (M=M>>1;).M[7:3] was then set to the temporary variable. Lastly X was shifted over by 1 bit. These operations were then repeated for each state (states 1 through 4). Once the statemachine reached state 4, it would reset M to 0, A to X, and B to Y. At the end of each state, the M input would be separated into 2 4-bit numbers and fed into a seven-segment decoder to get the output.

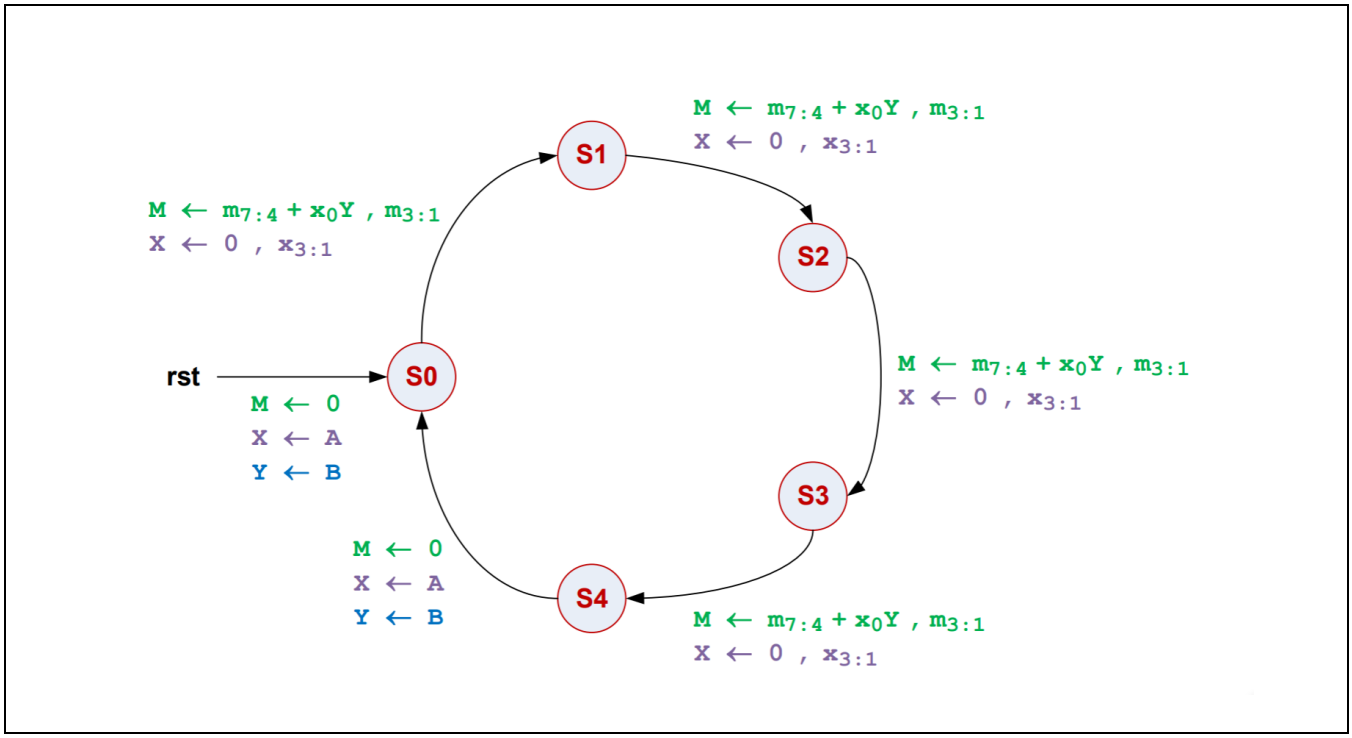

Figure 2. State diagram for the proposed solution

Implementation and Test Plan

I have implemented the solution plan explained above, by first completing the prelim of the lab, creating the main program and a testbench to simulate the program. The testbench simulated a clock and set the A and B inputs to 13 and 11 respectively. This resulted in the output of 143 or 0x8F. Once running the testbench, I saw that the simulation resulted in the correct output and moved on to testing with the FPGA. Compiling and running on the FPGA verified that the code worked for more than the two testbench inputs and I was done with the lab.

During this lab I ran into many errors. Most of it was caused by trying to remove the multiplication from the code. M[7:4] + X[0]*Y was instead M[7:4] + {4{X[0]}}&Y. This makes sense but I was just unable to get it to work with my code. Once I replaced this line with the multiplication operation, everything

worked, and I was able to get the correct outputs. The other confusing thing about this lab was the use of a state machine. Instead of using a state machine, we could have easily used a always for the positive edge of the clock. This way makes sense to me because the state machine does the same thing for all the states but at state 0. Instead of relying on states to detect when it is done, we could simply use a loop. Or if we wanted to synchronize the states to the clock, simply adding a variable counter would also work. This counter could increment by one after each clock cycle until the variable was equal to 4. Then it would reset everything. This is similar to the state machine but does not require as much code.

Overall, my code worked and I was happy with how it turned out.

Lab5.v

```
module Lab5 ( rst , clk , A , B , SS1 , SS0 ) ;

        input rst, clk;
        input [3:0] A,B;
        output reg [6:0] SS1,SS0;

        localparam  [2:0] state____  =  0 ,
                                                state1___  =  1 ,
                                                state_2__  =  2 ,
                                                state__3_  =  3 ,
                                                state___4  =  4 ;
        reg [2:0]  state ;
        reg [7:0]  M;
        reg [4:0] temp;
        reg [3:0] X,Y;
        reg [3:0] SS1_undecoded,SS0_undecoded;

        always @(posedge clk) begin
                if  ( rst )  begin
                        state  = state____ ;
                end
                else  begin
                        case  ( state )
                                state____  :  begin
                                  state  =  state1___ ;

                                end

                                state1___  :  begin
                                  state  =  state_2__ ;
                                end

                                state_2__  :  begin
                                  state  =  state__3_ ;
                                end
```

```
                state__3_  :  begin
                  state  =  state___4 ;
                end

                state___4  :  begin
                  state  =  state____ ;

                end


                default  :  begin
                  state  =  state____ ;
                end
         endcase
 end


case  ( state )
        state____  :  begin
                M=0;
                X=A;
                Y=B;
        end
        state1___  :  begin
                temp=(M[7:4] + (X[0]*Y));
                M[2:0] = M[3:1];
                M[7:3] = temp;
                X=X>>1;
        end
        state_2__  :  begin
                temp=(M[7:4] + (X[0]*Y));
                M[2:0] = M[3:1];
                M[7:3] = temp;
                X=X>>1;
        end
        state__3_  :  begin
                temp=(M[7:4] + (X[0]*Y));
                M[2:0] = M[3:1];
                M[7:3] = temp;
                X=X>>1;

        end
        state___4  :  begin
                temp=(M[7:4] + (X[0]*Y));
                M[2:0] = M[3:1];
```

```
                        M[7:3] = temp;
                        X=X>>1;
                end
        endcase
SS1_undecoded = M[7:4];
SS0_undecoded = M[3:0];
end
        always @(SS1_undecoded) begin
                case  ( SS1_undecoded )
                        4'h0  :  SS1  =  7'b1000000 ;
                        4'h1  :  SS1  =  7'b1111001 ;
                        4'h2  :  SS1  =  7'b0100100 ;
                        4'h3  :  SS1  =  7'b0110000 ;
                        4'h4  :  SS1  =  7'b0011001 ;
                        4'h5  :  SS1  =  7'b0010010 ;
                        4'h6  :  SS1  =  7'b0000010 ;
                        4'h7  :  SS1  =  7'b1111000 ;
                        4'h8  :  SS1  =  7'b0000000 ;
                        4'h9  :  SS1  =  7'b0010000 ;
                        4'hA  :  SS1  =  7'b0001000 ;
                        4'hB  :  SS1  =  7'b0000011 ;
                        4'hC  :  SS1  =  7'b1000110 ;
                        4'hD  :  SS1  =  7'b0100001 ;
                        4'hE  :  SS1  =  7'b0000110 ;
                        4'hF  :  SS1  =  7'b0001110 ;
                endcase
        end
        always @(SS0_undecoded) begin
                case  ( SS0_undecoded )
                        4'h0  :  SS0  =  7'b1000000 ;
                        4'h1  :  SS0  =  7'b1111001 ;
                        4'h2  :  SS0  =  7'b0100100 ;
                        4'h3  :  SS0  =  7'b0110000 ;
                        4'h4  :  SS0  =  7'b0011001 ;
                        4'h5  :  SS0  =  7'b0010010 ;
                        4'h6  :  SS0  =  7'b0000010 ;
                        4'h7  :  SS0  =  7'b1111000 ;
                        4'h8  :  SS0  =  7'b0000000 ;
                        4'h9  :  SS0  =  7'b0010000 ;
                        4'hA  :  SS0  =  7'b0001000 ;
                        4'hB  :  SS0  =  7'b0000011 ;
                        4'hC  :  SS0  =  7'b1000110 ;
                        4'hD  :  SS0  =  7'b0100001 ;
                        4'hE  :  SS0  =  7'b0000110 ;
                        4'hF  :  SS0  =  7'b0001110 ;
                endcase
```

```
                end
endmodule
```

Lab5_tb.v

```
module  Lab5_tb ;

reg  clk_tb , rst_tb ;
wire  [6:0]  SS0_tb,SS1_tb ;
reg [3:0] A_tb,B_tb;

localparam  PER  =  20 ;

Lab5  dut ( rst_tb , clk_tb , A_tb , B_tb , SS1_tb , SS0_tb ) ;

always begin
clk_tb = 0;
#(PER/2) ;
clk_tb = 1;
#(PER/2) ;
end

initial begin
A_tb=13;
B_tb=11;
rst_tb  =  1 ;
#PER ;
rst_tb  =  0 ;
#PER ;
#PER ;
#PER ;
#PER ;
#PER;
$stop ;
end
endmodule
```

baseline_c5gx.v (Only one line added)

```
Lab5 dut ( SW[8] , KEY[0] , SW[7:4] , SW[3:0] , HEX1 , HEX0);
```

Figure 3. Verilog code for the proposed solution

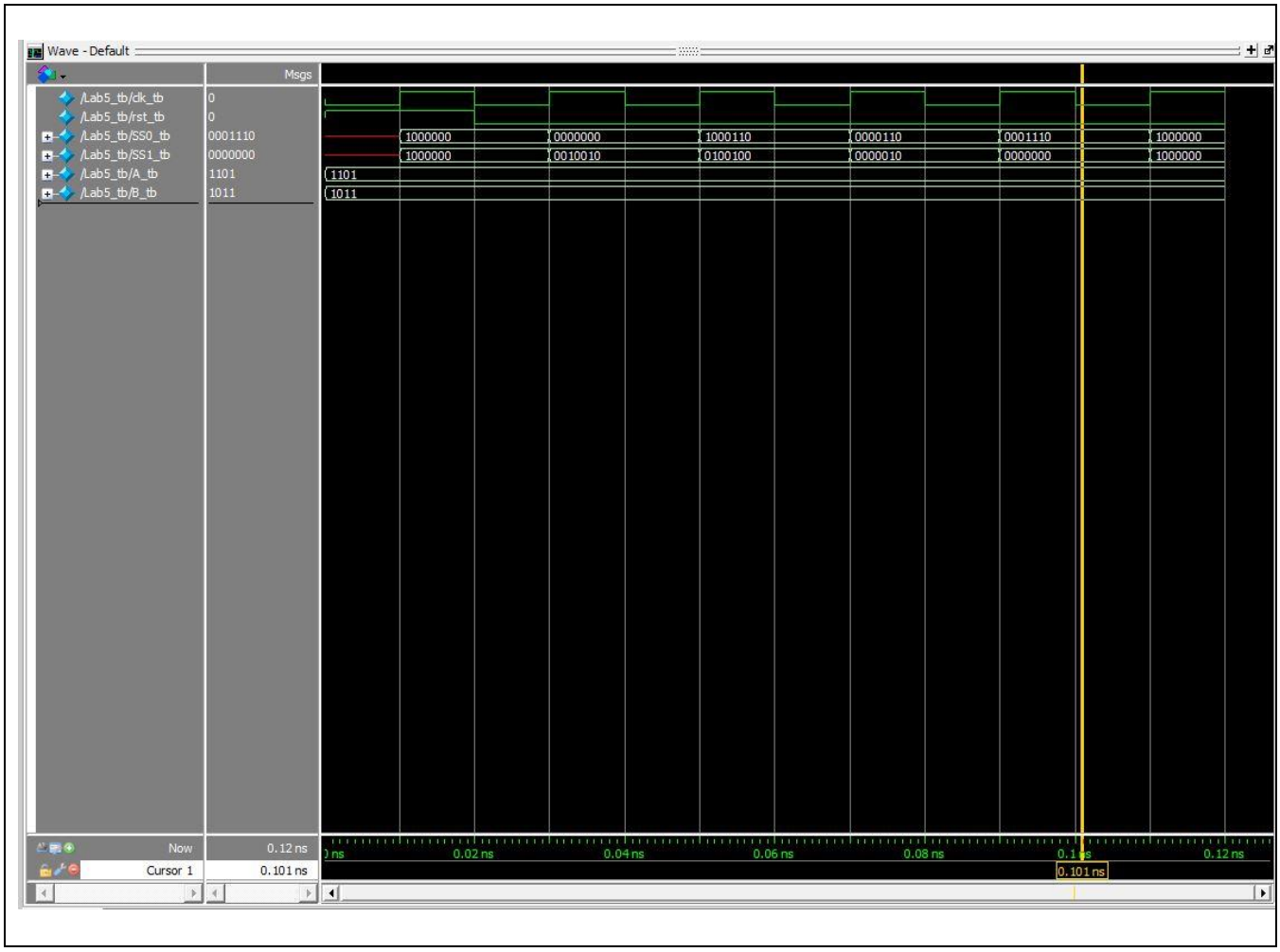

Figure 4. Lab pictures of the running solution