**Assignment 9 (Extra Credit)**

**Unsigned Integer Multiplier Based On Shift Add Algorithm**

Completed by:

Casey Delaney 012272156

Ryota Suzuki 011115039

(Note: there was no cover page uploaded to canvas at the time of submitting our report)

**The items in this lab report are listed in the order shown in assignment 9.**

1. Successful Tasks - All tasks completed on time

* Datapath waveform simulation
* Control unit waveform simulation
* Full multiplier simulation
* FPGA validation

1. **Source Code**

|  |
| --- |
| multiplier.v |
| `timescale 1ns / 1ps  module multiplier(  input clk,  input [3:0] M1,  input [3:0] M2,  input go,  output done,  output [3:0] state,  output [7:0] Result  );    wire L1;  wire SL;  wire L2;  wire SR;  wire sel;  wire L3;  wire R2\_q0;  wire M2\_zero;    datapath dp(  .clk(clk),  .M1(M1),  .M2(M2),  .L1(L1),  .SL(SL),  .L2(L2),  .SR(SR),  .sel(sel),  .L3(L3),    .R2\_q0(R2\_q0),  .M2\_zero(M2\_zero),  .Result(Result)  );    control\_unit cu(  .clk(clk),  .go(go),  .R2(R2\_q0),  .M2\_zero(M2\_zero),    .L1(L1),  .SL(SL),  .L2(L2),  .SR(SR),  .sel(sel),  .L3(L3),  .done(done),  .state(state)  );    endmodule |

|  |
| --- |
| datapath.v |
| module datapath(  input clk,  input [3:0] M1,  input [3:0] M2,  input L1,  input SL,  input L2,  input SR,  input sel,  input L3,    output R2\_q0,  output M2\_zero,  output [7:0] Result  );  wire [7:0] R1\_out;  wire [3:0] R2\_out;  wire [7:0] R3\_out;  wire [7:0] add\_out;  wire [7:0] mux\_out;  shift\_register #(8) R1(  .CLK(clk),  .RST(0),  .SL(SL),  .SR(0),  .LD(L1),  .LeftIn(0),  .RightIn(0),  .D({4'b0000, M1}),  .Q(R1\_out)  );  shift\_register #(4) R2(  .CLK(clk),  .RST(0),  .SL(0),  .SR(SR),  .LD(L2),  .LeftIn(0),  .RightIn(0),  .D(M2),  .Q(R2\_out)  );  shift\_register #(8) R3(  .CLK(clk),  .RST(0),  .SL(0),  .SR(0),  .LD(L3),  .LeftIn(0),  .RightIn(0),  .D(mux\_out),  .Q(R3\_out)  );  NOR N(  .D(R2\_out),  .Q(M2\_zero)  );  adder add(  .A(R1\_out),  .B(R3\_out),  .C(add\_out)  );  mux2 #(8) mux(  .d0(8'b00000000),  .d1(add\_out),  .sel(sel),  .muxOut(mux\_out)  );  assign Result = R3\_out;  assign R2\_q0 = R2\_out[0];  endmodule |

|  |
| --- |
| shift\_register.v |
| module shift\_register #(parameter Data\_width = 4)(CLK, RST, SL, SR, LD, LeftIn, RightIn, D, Q);  input CLK, RST, SL, SR, LD, LeftIn, RightIn;  input [Data\_width-1:0] D;  output reg [Data\_width-1:0] Q;  always @(posedge CLK)  begin  if (RST) Q = 0;  else if (LD) Q = D;  else if (SL) // shift left  begin  Q [Data\_width-1:1] = Q [Data\_width-2:0];  Q [0] = RightIn;  end  else if (SR) // shift right  begin  Q [Data\_width-2:0] = Q [Data\_width-1:1];  Q [Data\_width -1] = LeftIn;  end  else Q [Data\_width-1:0] = Q [Data\_width-1:0];  end  endmodule |

|  |
| --- |
| NOR.v |
| `timescale 1ns / 1ps  module NOR(  input [3 : 0] D,  output Q  );  assign Q = !(D || 4'b0000);  endmodule |

|  |
| --- |
| adder.v |
| `timescale 1ns / 1ps  module adder(  input [7:0] A,  input [7:0] B,  output [7:0] C  );  assign C = (A+B);  endmodule |

|  |
| --- |
| mux2.v |
| module mux2 #(parameter WIDTH = 4)(  input [WIDTH - 1:0] d0,  input [WIDTH - 1:0] d1,  input sel,  output [WIDTH - 1:0] muxOut  );    assign muxOut = (sel) ? d1:d0;    endmodule |

|  |
| --- |
| control\_unit.v |
| `timescale 1ns / 1ps  module control\_unit(  input clk,  input go,  input R2,  input M2\_zero,    output reg L1,  output reg SL,  output reg L2,  output reg SR,  output reg sel,  output reg L3,  output reg done,  output reg [3:0] state = 0  );      always@(negedge clk) begin  if(state == 0) begin  L1 = 0;  SL = 0;  L2 = 0;  SR = 0;  sel = 0;  L3 = 0;  done = 0;  if(go == 1) begin  state = 1;  end  end  else if(state == 1) begin  L1 = 1;  L2 = 1;  L3 = 1;  state = 2;  end  else if(state == 2) begin  if(M2\_zero == 0 && R2 == 1) begin  L1 = 0;  SL = 0;  L2 = 0;  SR = 0;  sel = 1;  L3 = 1;  done = 0;  state = 3;  end  else if(M2\_zero == 0 && R2 == 0)begin  L1 = 0;  SL = 0;  L2 = 0;  SR = 0;  sel = 0;  L3 = 0;  done = 0;  state = 3;  end  else begin  L1 = 0;  SL = 0;  L2 = 0;  SR = 0;  sel = 0;  L3 = 0;  done = 0;  state = 4;  end  end  else if(state == 3) begin  L1 = 0;  SL = 1;  L2 = 0;  SR = 1;  sel = 0;  L3 = 0;  done = 0;  state = 2;  end  else if(state == 4) begin  L1 = 0;  SL = 0;  L2 = 0;  SR = 0;  sel = 0;  L3 = 0;  done = 1;  state = 0;  end  end  endmodule |

|  |
| --- |
| button\_debouncer.v |
| module button\_debouncer #(parameter depth = 16) (  input wire clk, /\* 5 KHz clock \*/  input wire button, /\* Input button from constraints \*/  output reg debounced\_button  );    localparam history\_max = (2\*\*depth)-1;  /\* History of sampled input button \*/  reg [depth-1:0] history;  always @ (posedge clk) begin  /\* Move history back one sample and insert new sample \*/  history <= { button, history[depth-1:1] };    /\* Assert debounced button if it has been in a consistent state throughout history \*/  debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;  end    endmodule |

|  |
| --- |
| clk\_gen.v |
| `timescale 1ns / 1ps  module clk\_gen (  input wire clk100MHz,  input wire rst,  output reg clk\_4sec,  output reg clk\_5KHz  );  integer count1, count2;  always @ (posedge clk100MHz) begin  if (rst) begin  count1 = 0; clk\_4sec = 0;  count2 = 0; clk\_5KHz = 0;  end else begin  if (count1 == 200000000) begin  clk\_4sec = ~clk\_4sec;  count1 = 0;  end  if (count2 == 10000) begin  clk\_5KHz = ~clk\_5KHz;  count2 = 0; end  count1 = count1 + 1;  count2 = count2 + 1;  end  end  endmodule |

|  |
| --- |
| seg\_handler.v |
| module seg\_handler(  input clk100MHz,  input [3:0]led0,  input [3:0]led1,  input [3:0]led2,  input [3:0]led3,  output reg [7:0]led\_out,  output reg [3:0]led\_anode  );  reg clk\_5KHz;  reg [1:0]led\_sel;  reg [3:0]led\_out\_bcd;  integer count1, count2;  always @ (posedge clk100MHz) begin  if(count1 == 200000000) begin  count1 = 0;  end  if (count2 == 10000) begin  clk\_5KHz = ~clk\_5KHz;  count2 = 0;  end  count1 = count1 + 1;  count2 = count2 + 1;  end  always @ (posedge clk\_5KHz) begin  led\_sel = led\_sel + 1;  case(led\_sel)  2'b00: begin  led\_anode = 4'b0111;  led\_out\_bcd = led0;  end  2'b01: begin  led\_anode = 4'b1011;  led\_out\_bcd = led1;  end  2'b10: begin  led\_anode = 4'b1101;  led\_out\_bcd = led2;  end  2'b11: begin  led\_anode = 4'b1110;  led\_out\_bcd = led3;  end  endcase  case(led\_out\_bcd)  4'd0: led\_out = 8'b11000000;  4'd1: led\_out = 8'b11111001;  4'd2: led\_out = 8'b10100100;  4'd3: led\_out = 8'b10110000;  4'd4: led\_out = 8'b10011001;  4'd5: led\_out = 8'b10010010;  4'd6: led\_out = 8'b10000010;  4'd7: led\_out = 8'b11111000;  4'd8: led\_out = 8'b10000000;  4'd9: led\_out = 8'b10010000;  4'b10: led\_out = 8'b01111111;  default: led\_out = 8'b11111111;  endcase  end  endmodule |

|  |
| --- |
| mult\_tb.v |
| `timescale 1ns / 1ps  module mult\_tb;  reg clk\_tb;  reg [3:0] M1\_tb;  reg [3:0] M2\_tb;  reg go\_tb;  wire done\_tb;  wire [3:0] state\_tb;  wire [7:0] Result\_tb;  reg [4:0] i;  reg [4:0] j;  reg [8:0] expected;  integer error=0;  multiplier DUT(  .clk(clk\_tb),  .M1(M1\_tb),  .M2(M2\_tb),  .go(go\_tb),  .done(done\_tb),  .state(state\_tb),  .Result(Result\_tb)  );  task clock;  begin  clk\_tb = 0;  #5;  clk\_tb = 1;  #5;  end  endtask    initial begin  for(i=0; i<=15; i=i+1)begin  for(j=0; j<=15; j=j+1)begin  expected = i\*j;  clk\_tb=0;  M1\_tb = i;  M2\_tb = j;  go\_tb = 0;  clock;  go\_tb = 1;  clock;  go\_tb = 0;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  if(expected!=Result\_tb)begin  error=error+1;  end  end  end  end  endmodule |

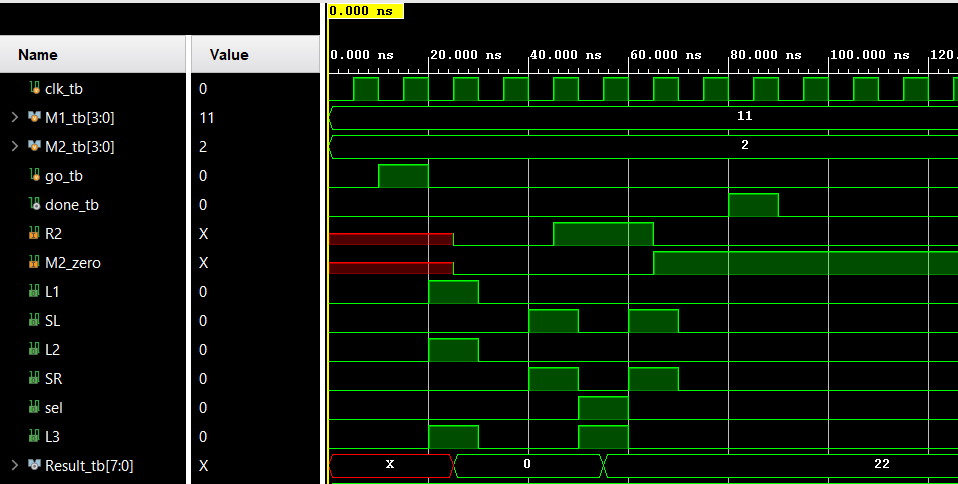
|  |
| --- |
| cu\_tb.v |
| `timescale 1ns / 1ps  module cu\_tb; //testbench for CU  reg clk\_tb;  reg [3:0] M1\_tb;  reg [3:0] M2\_tb;  reg go\_tb;  wire done\_tb;  wire [3:0] state\_tb;  reg [3:0] expected\_state;  integer error=0;  multiplier DUT( //assigning signal to multiplier  .clk(clk\_tb),  .M1(M1\_tb),  .M2(M2\_tb),  .go(go\_tb),  .done(done\_tb),  .state(state\_tb),  .Result(Result\_tb)  );  task clock;  begin  clk\_tb = 0;  #5;  clk\_tb = 1;  #5;  end  endtask    initial begin  clk\_tb=0;  M1\_tb = 11;  M2\_tb = 2;  go\_tb = 0;  expected\_state=0; //initialize expected state  clock;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end  go\_tb = 1;  expected\_state=1; //change with accordance to state transition diagram  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end  go\_tb = 0;  expected\_state = 2;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end    expected\_state=3;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end    expected\_state=2;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end    expected\_state=3;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end    expected\_state = 2;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end    expected\_state = 4;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end    expected\_state = 0;  clock;  if(expected\_state != state\_tb)begin  error=error+1;  end  end  endmodule |

|  |
| --- |
| datapath\_tb.v |
| module datapath\_tb;  reg clk\_tb;  reg [3:0] M1\_tb;  reg [3:0] M2\_tb;    reg L1\_tb;  reg SL\_tb;  reg L2\_tb;  reg SR\_tb;  reg sel\_tb;  reg L3\_tb;    wire R2\_q0\_tb;  wire M2\_zero\_tb;  wire [7:0] Result\_tb;    reg [3:0] state;    reg [8:0] expected;  reg done=0;  integer error=0;    datapath DUT(  clk\_tb,  M1\_tb,  M2\_tb,    L1\_tb,  SL\_tb,  L2\_tb,  SR\_tb,  sel\_tb,  L3\_tb,    R2\_q0\_tb,  M2\_zero\_tb,  Result\_tb  );    task clock;  begin  clk\_tb = 0;  #5;  clk\_tb = 1;  #5;  end  endtask      initial begin  //state 0  M1\_tb = 11;  M2\_tb = 2;  L1\_tb = 0;  SL\_tb = 0;  L2\_tb = 0;  SR\_tb = 0;  sel\_tb = 0;  L3\_tb = 0;  done = 0;  clock;    //state 1  L1\_tb = 1;  L2\_tb = 1;  L3\_tb = 1;  clock;    //state 2  while(done==0)begin  if(M2\_zero\_tb == 0 && R2\_q0\_tb ==1)begin  L1\_tb = 0;//assigning signals to what they should be  SL\_tb = 0;  L2\_tb = 0;  SR\_tb = 0;  sel\_tb = 1;  L3\_tb = 1;  clock;      //state 3  L1\_tb = 0;//assigning signals to what they should be  SL\_tb = 1;  L2\_tb = 0;  SR\_tb = 1;  sel\_tb = 0;  L3\_tb = 0;  clock;    end  else if(M2\_zero\_tb == 0 && R2\_q0\_tb ==0)begin  L1\_tb = 0;//assigning signals to what they should be  SL\_tb = 0;  L2\_tb = 0;  SR\_tb = 0;  sel\_tb = 0;  L3\_tb = 0;  clock;    //state 3  L1\_tb=0;//assigning signals to what they should be  SL\_tb = 0;  L2\_tb = 0;  SR\_tb = 0;  sel\_tb = 1;  L3\_tb = 1;  clock;      end  else begin  L1\_tb = 0;//assigning signals to what they should be  SL\_tb = 0;  L2\_tb = 0;  SR\_tb = 0;  sel\_tb = 0;  L3\_tb = 0;  clock;    //state 4  L1\_tb = 0; //assigning signals to what they should be  SL\_tb = 0;  L2\_tb = 0;  SR\_tb = 0;  sel\_tb = 0;  L3\_tb = 0;  expected = M1\_tb \* M2\_tb;  if(expected != Result\_tb)begin  error = error+1; //adding to error counter if the output is not as expected  end  done = 1;  clock;    end  end  end  endmodule |

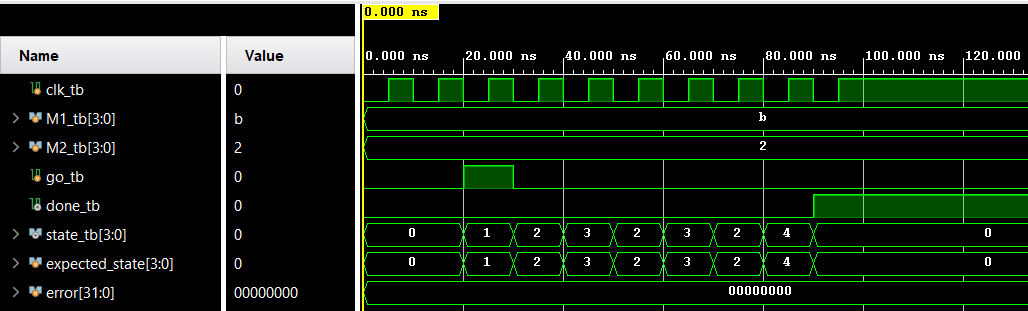
|  |
| --- |
| mult\_fpga.v |
| `timescale 1ns / 1ps  //////////////////////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 05/03/2020 09:45:18 PM  // Design Name:  // Module Name: mult\_fpga  // Project Name:  // Target Devices:  // Tool Versions:  // Description:  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  //////////////////////////////////////////////////////////////////////////////////  module mult\_fpga(  input [3:0]Din1,  input [3:0]Din2,      input clk\_FPGA,  input button\_clk,  input button\_go,  output [7:0]led\_out\_fpga,  output [3:0]led\_anode\_fpga,  output reg done\_LED,  output reg [3:0]Dout1,  output reg [3:0]Dout2  );    wire done\_fpga;  wire fiveKhz\_clk;  wire button\_w;  wire button\_FPGA;  wire DONT\_USE;  wire [7:0] Result\_fpga;  wire [3:0] state\_fpga;      multiplier fpga\_mult(  .clk(button\_FPGA),  .M1(Din1),  .M2(Din2),  .go(button\_go),  .done(done\_fpga),  .state(state\_fpga),  .Result(Result\_fpga)  );  clk\_gen clk (  .clk100MHz(clk\_FPGA),  .rst(1'b0),  .clk\_4sec(DONT\_USE),  .clk\_5KHz(fiveKhz\_clk)  );  button\_debouncer db(  .clk(fiveKhz\_clk),  .button(button\_clk),  .debounced\_button(button\_FPGA)  );  wire [3:0] led0\_w = state\_fpga;  wire [3:0] led3\_w = Result\_fpga % 10;  wire [3:0] led2\_w = ((Result\_fpga % 100) - led3\_w)/10;  wire [3:0] led1\_w = ((Result\_fpga % 1000) - led2\_w)/100;  seg\_handler segs(  .clk100MHz(clk\_FPGA),  .led0(led0\_w),  .led1(led1\_w),  .led2(led2\_w),  .led3(led3\_w),  .led\_out(led\_out\_fpga),  .led\_anode(led\_anode\_fpga)  );  always @ (\*) begin  Dout1 = Din1;  Dout2 = Din2;  done\_LED = done\_fpga;  end  endmodule |

|  |
| --- |
| constraints.xdc |
| # Clock input  set\_property -dict {PACKAGE\_PIN W5 IOSTANDARD LVCMOS33} [get\_ports {clk\_FPGA}];  create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk\_FPGA}];  # Button (debouncer and go)  set\_property -dict {PACKAGE\_PIN U18 IOSTANDARD LVCMOS33} [get\_ports {button\_clk}]; # Center Button  set\_property -dict {PACKAGE\_PIN T17 IOSTANDARD LVCMOS33} [get\_ports {button\_go}]; #Right button  # Input Switches  set\_property -dict {PACKAGE\_PIN V17 IOSTANDARD LVCMOS33} [get\_ports {Din1[0]}]; # Din10  set\_property -dict {PACKAGE\_PIN V16 IOSTANDARD LVCMOS33} [get\_ports {Din1[1]}]; # Din11  set\_property -dict {PACKAGE\_PIN W16 IOSTANDARD LVCMOS33} [get\_ports {Din1[2]}]; # Din12  set\_property -dict {PACKAGE\_PIN W17 IOSTANDARD LVCMOS33} [get\_ports {Din1[3]}]; # Din13  set\_property -dict {PACKAGE\_PIN W15 IOSTANDARD LVCMOS33} [get\_ports {Din2[0]}]; # Din20  set\_property -dict {PACKAGE\_PIN V15 IOSTANDARD LVCMOS33} [get\_ports {Din2[1]}]; # Din21  set\_property -dict {PACKAGE\_PIN W14 IOSTANDARD LVCMOS33} [get\_ports {Din2[2]}]; # Din22  set\_property -dict {PACKAGE\_PIN W13 IOSTANDARD LVCMOS33} [get\_ports {Din2[3]}]; # Din23  # LED Output  set\_property -dict {PACKAGE\_PIN U16 IOSTANDARD LVCMOS33} [get\_ports {Dout1[0]}]; # Dout10  set\_property -dict {PACKAGE\_PIN E19 IOSTANDARD LVCMOS33} [get\_ports {Dout1[1]}]; # Dout11  set\_property -dict {PACKAGE\_PIN U19 IOSTANDARD LVCMOS33} [get\_ports {Dout1[2]}]; # Dout12  set\_property -dict {PACKAGE\_PIN V19 IOSTANDARD LVCMOS33} [get\_ports {Dout1[3]}]; # Dout13  set\_property -dict {PACKAGE\_PIN W18 IOSTANDARD LVCMOS33} [get\_ports {Dout2[0]}]; # Dout20  set\_property -dict {PACKAGE\_PIN U15 IOSTANDARD LVCMOS33} [get\_ports {Dout2[1]}]; # Dout21  set\_property -dict {PACKAGE\_PIN U14 IOSTANDARD LVCMOS33} [get\_ports {Dout2[2]}]; # Dout22  set\_property -dict {PACKAGE\_PIN V14 IOSTANDARD LVCMOS33} [get\_ports {Dout2[3]}]; # Dout23  set\_property -dict {PACKAGE\_PIN P3 IOSTANDARD LVCMOS33} [get\_ports {done\_LED}]; # done LED  # 7 Segment Display  set\_property -dict {PACKAGE\_PIN W7 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[0]}]; # CA  set\_property -dict {PACKAGE\_PIN W6 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[1]}]; # CB  set\_property -dict {PACKAGE\_PIN U8 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[2]}]; # CC  set\_property -dict {PACKAGE\_PIN V8 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[3]}]; # CD  set\_property -dict {PACKAGE\_PIN U5 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[4]}]; # CE  set\_property -dict {PACKAGE\_PIN V5 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[5]}]; # CF  set\_property -dict {PACKAGE\_PIN U7 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[6]}]; # CG  set\_property -dict {PACKAGE\_PIN V7 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[7]}]; # DP  set\_property -dict {PACKAGE\_PIN U2 IOSTANDARD LVCMOS33} [get\_ports {led\_anode\_fpga[0]}]; # AN0  set\_property -dict {PACKAGE\_PIN U4 IOSTANDARD LVCMOS33} [get\_ports {led\_anode\_fpga[1]}]; # AN1  set\_property -dict {PACKAGE\_PIN V4 IOSTANDARD LVCMOS33} [get\_ports {led\_anode\_fpga[2]}]; # AN2  set\_property -dict {PACKAGE\_PIN W4 IOSTANDARD LVCMOS33} [get\_ports {led\_anode\_fpga[3]}]; # AN3 |

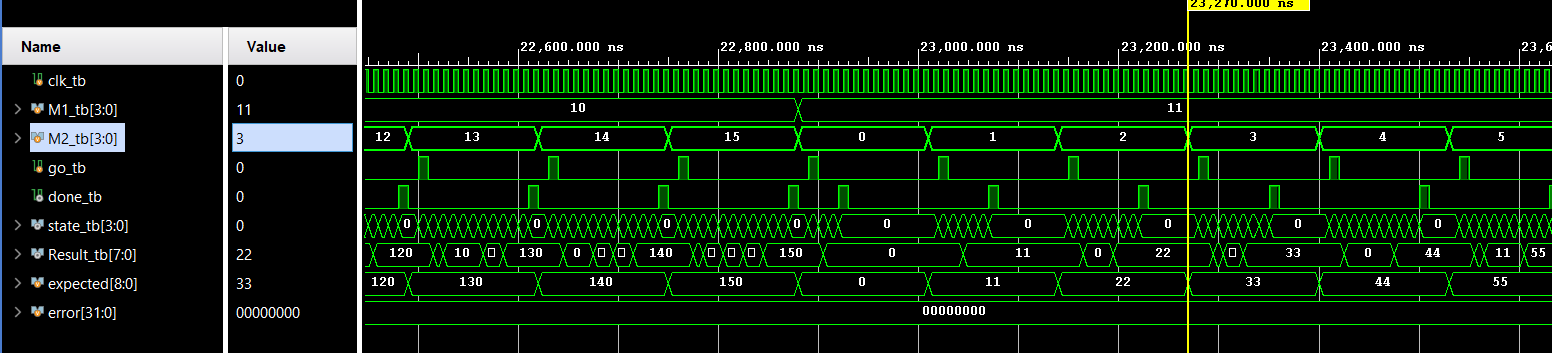
**3) Simulation Results**



*Figure 1: Datapath waveform simulation*



*Figure 2: Control unit waveform simulation*



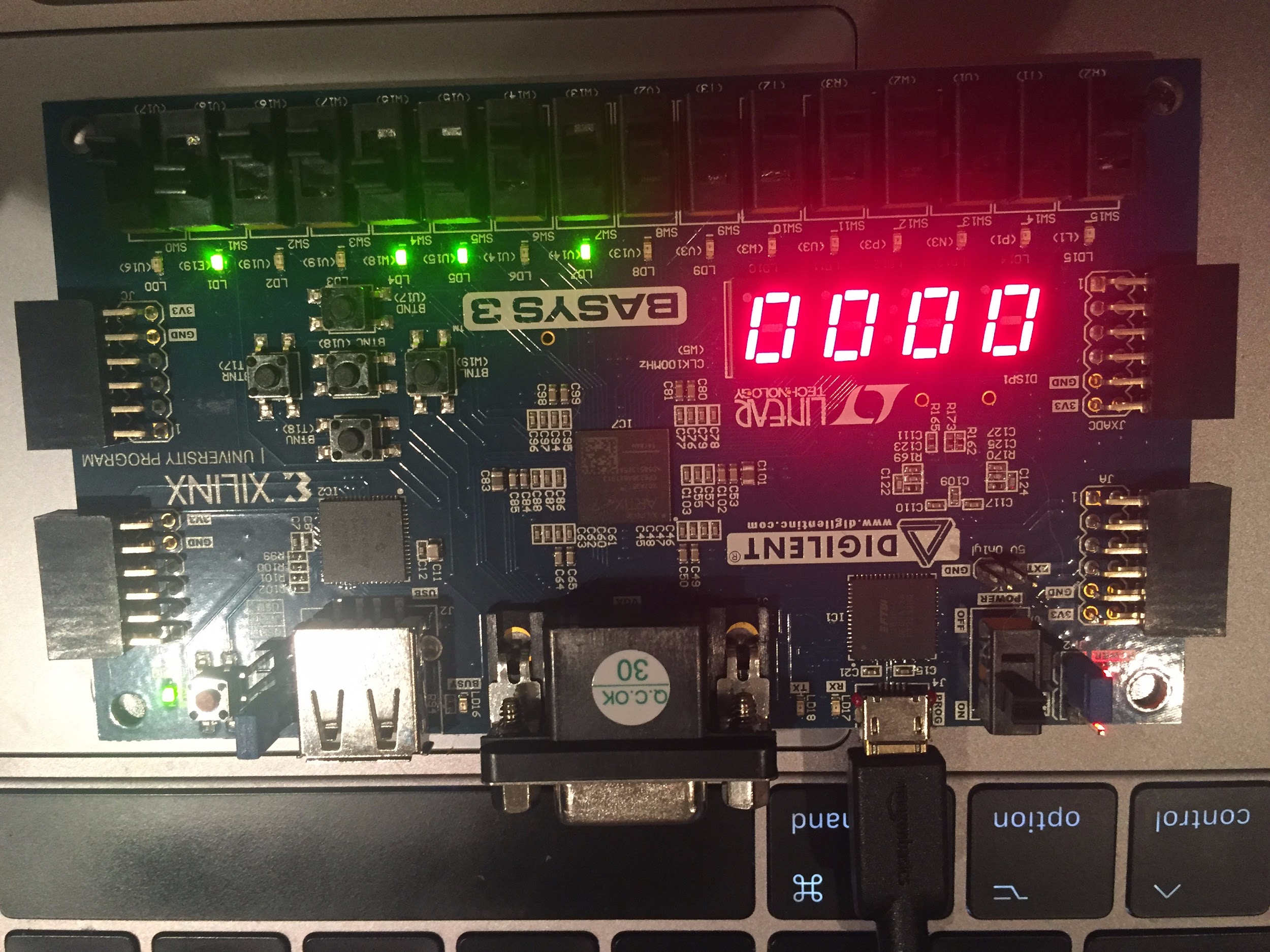
*Figure 3: Full multiplier self checking simulation*

**4) Hardware validation**

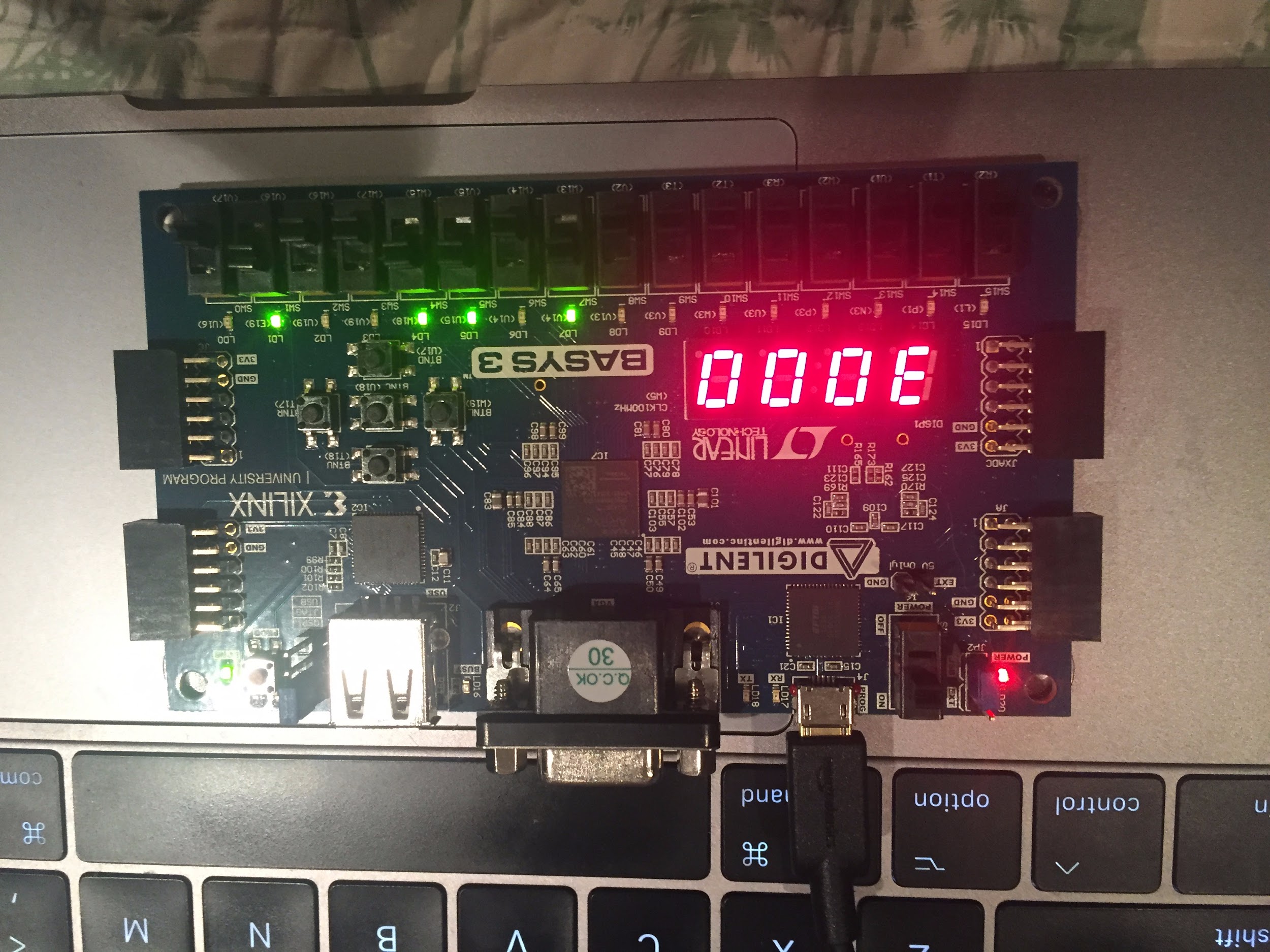
For our FPGA board, the 4 switches to the farthest right represent M1 and the next 4 switches represent M2. On the 7 segment display, the segment farthest to the left represents the current state while the three 7 segments to the right represent the product.

Also, the 4th led from the left represents the done flag, and it is off in this state because the calculation has not finished.

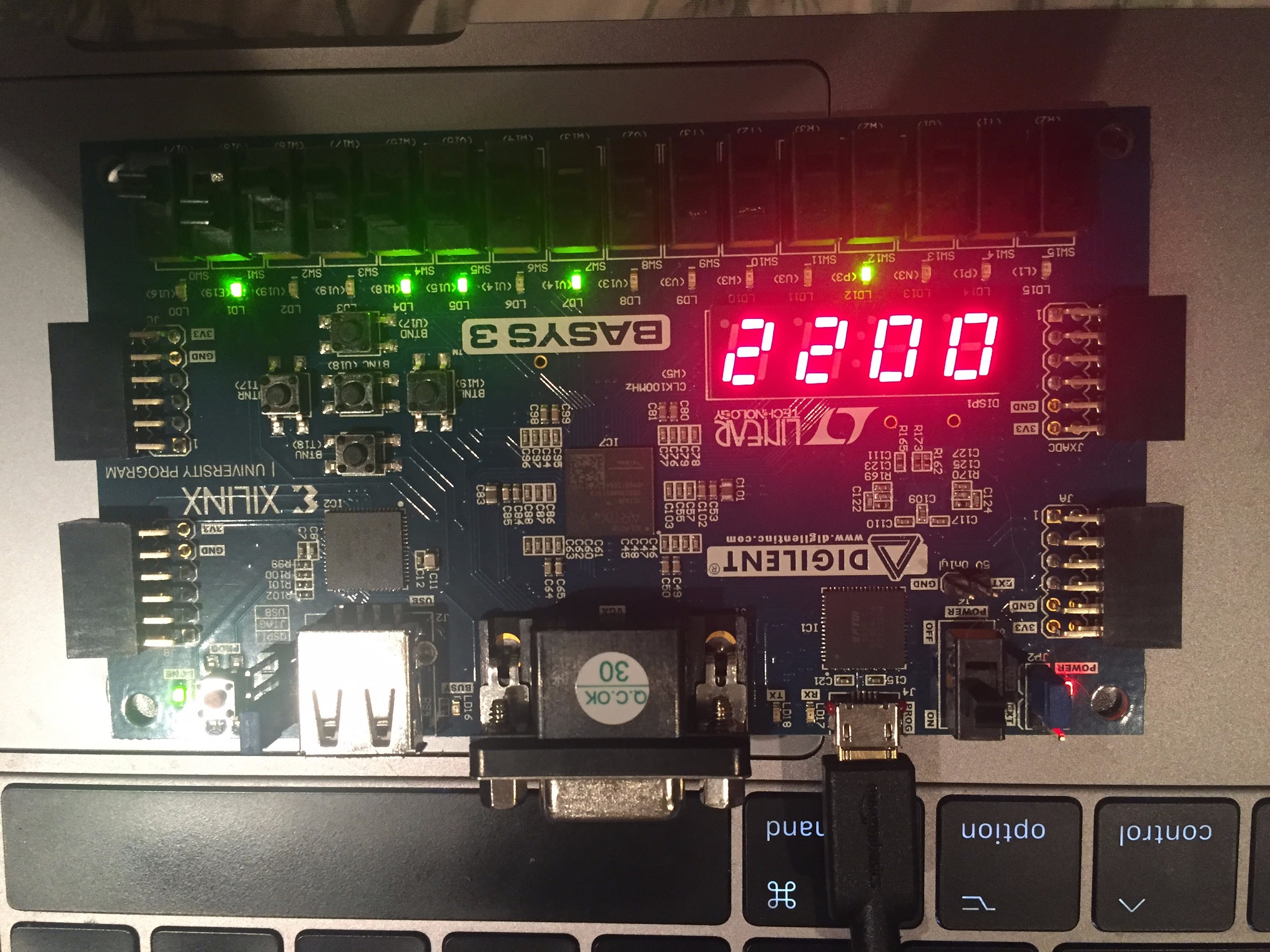
Also, note that the 7seg display will display a value in decimal.



*Figure 4: State 0 before performing 1011 x 0010 calculation*



*Figure 5: State 3 during performing 1011 x 0010 calculation*



*Figure 5: State 0 after performing 1011 x 0010 calculation. Product is 22 from the calculation of 11 x 2*