

**Introduction**

The purpose of this lab is to familiarize ourselves with the hierarchical design methodology and design/verification /validation flow and the EDA tool. Our task was to create a 4-bit adder and run a simulation test via “inferred” design approach which required us to use the “+” sign. Then we designed and ran a 4-bit adder but this time using the carry look ahead method and implementing it on our BASYS3 board. Unlike the ripple carry adder, the carry look ahead adder is used because it removes the propagation delay that would be present in a ripple carry adder. It does this by computing all of the carries at the same time.

**Design Methodology**

When we designed the inferred method, we simply made the adder by using the ‘+’ sign and using an if statement to compare if it was greater than 15. If it was, we added a carry over bit and displayed the rest. However, when we made an adder using the Carry Look Ahead method, we had to compare each bit using an AND function to see if it had any carry over and using a XOR function to calculate the sum. Then we gathered that information and made a Carry Look Ahead module to calculate the 4 bit addition.

To calculate the carry, we used some relations from our adders to create our carryBits.v module, as shown in figure 7.

(1)

(2)

(3)

Utilizing formulas 1, 2 and 3, the carry for all of the bits in the CLA adders can be calculated, as shown below.

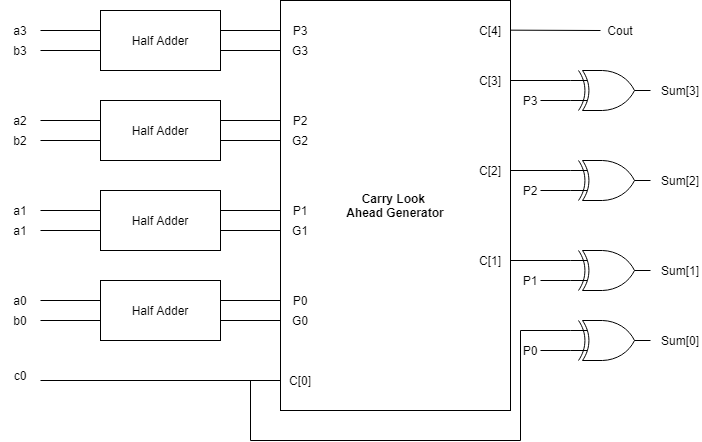
As seen in the equations above, none of the carry values depend on a previous carry value. All of the carries can be calculated in parallel vs in series as in the ripple adder which reduces the propagation delay. Unfortunately, there are some issues that arise with this method. If we were to calculate a 64 bit number, for example, there would be an enormous number of gates to computer all of the carries at the same time, and this would increase the manufacturing costs.

The modules are listed in table 1 below, and it describes the different modules used on this assignment.

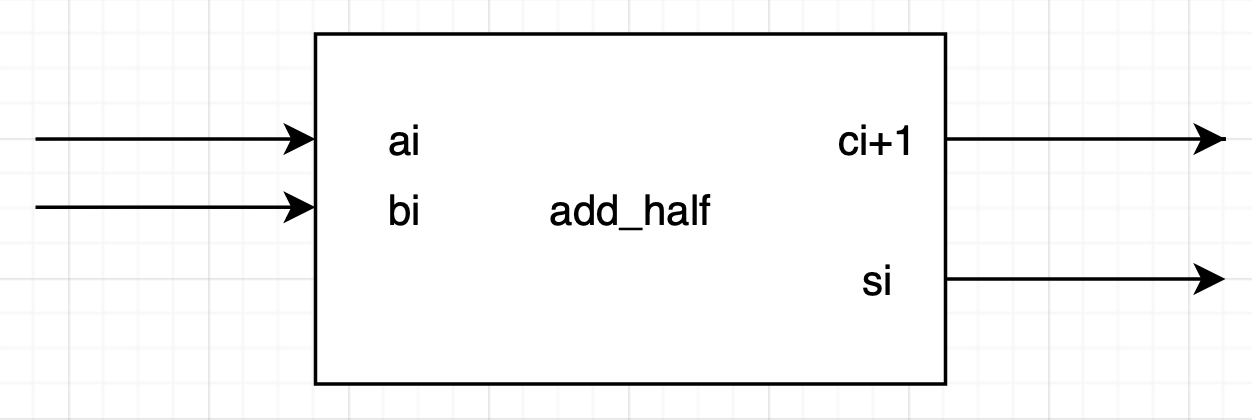
*Table 1. Module descriptions*

|  |  |
| --- | --- |
| **Module** | **Function** |
| cla\_adder.v | Performs a 4 bit addition w/ carry look ahead |
| carryBits.v | Calculates the carry bits in parallel |
| add\_half.v | Calculates sum and carry over |
| bcd\_to\_7seg.v | Displays binary result on 7 seg LED |
| LEDSEL.v | Chooses which 7 seg to display to |
| inferred\_adder.v | Performs a 4 bit addition operation in an “inferred” way |
| inferred\_adder\_self\_checkingtb.v | Check all possible inputs of the inferred adder approach |
| clk\_gen.v | Generates a clock signal to synchronize with |

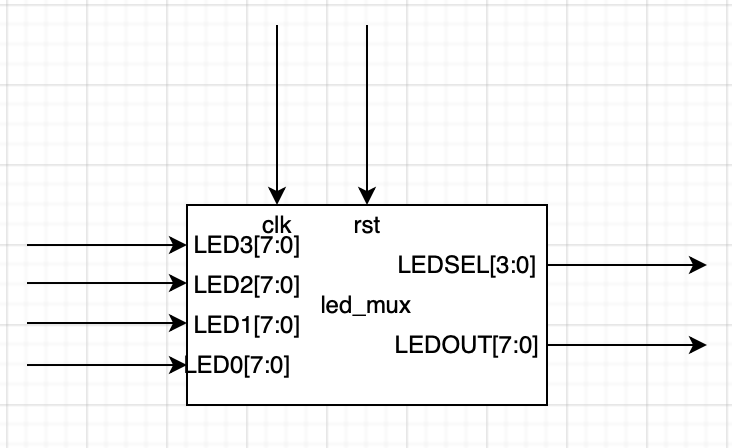
As seen in figure 1, the top level module of the CLA utilizes 4 half bit adders (figure 2), and it computers the carry out in parallel using the carrybits module shown in figure 7.. This is very different from the top level design of the inferred adder module as shown in figure 6. Figures 3 through 5 show the modules used for our 7 segment display.



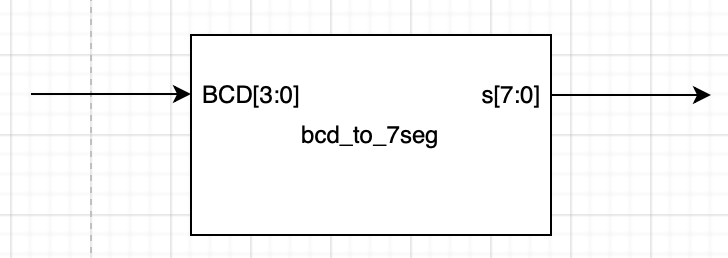
*Figure 1: Carry look ahead structure diagram*



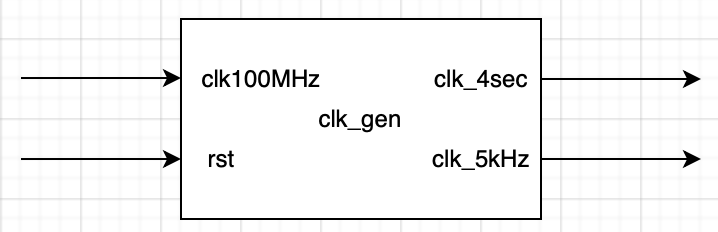
*Figure 2: add\_half structure diagram*



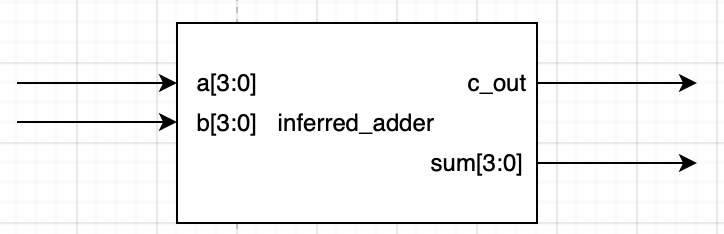
*Figure 3: led\_mux structure diagram*



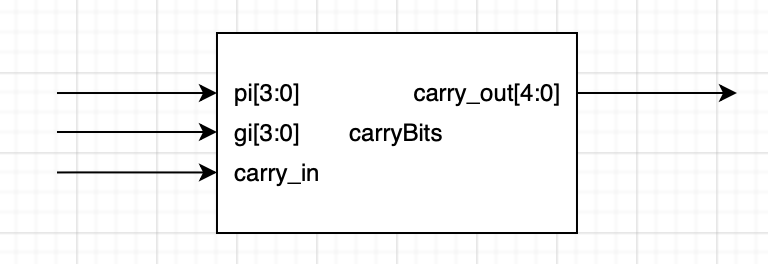
*Figure 4:bcd\_to\_7seg structure diagram*



*Figure 5:clk\_gen structure diagram*

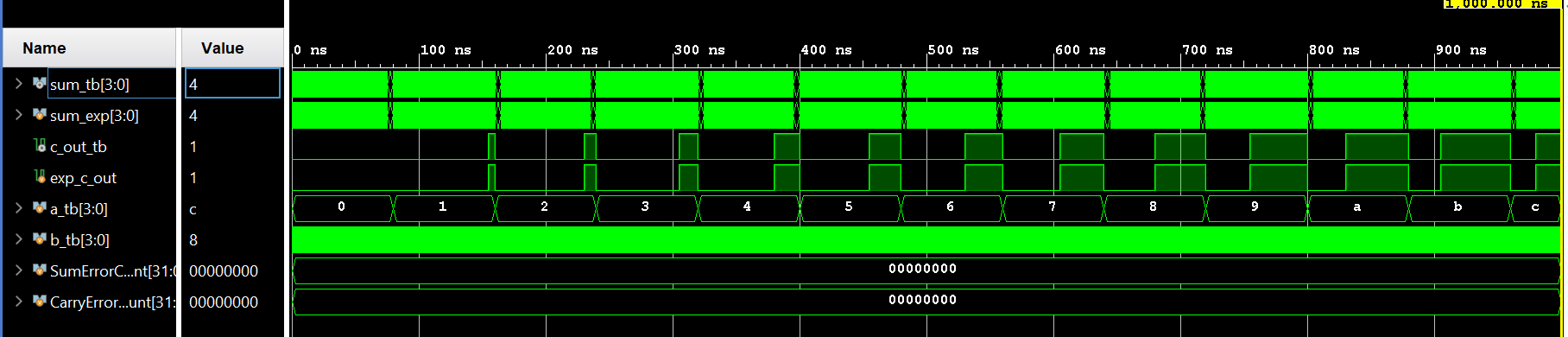


*Figure 6:inferred adder structure diagram*

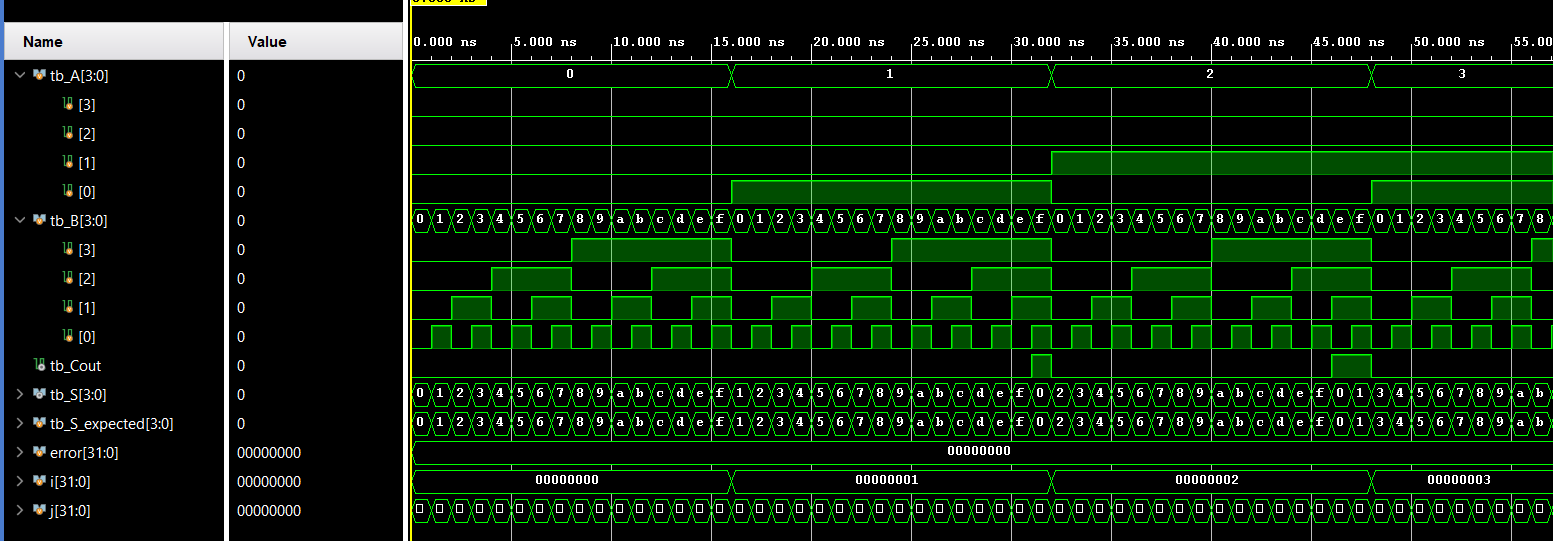


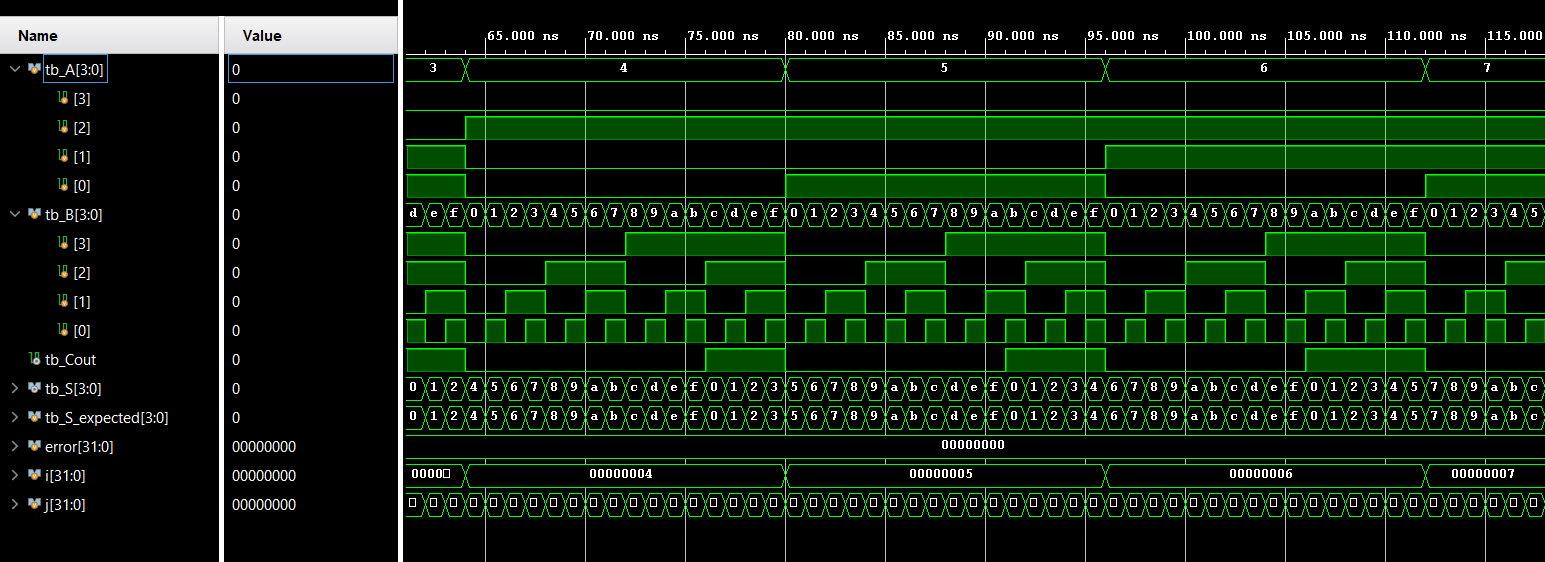
*Figure 7:carry bits structure diagram*

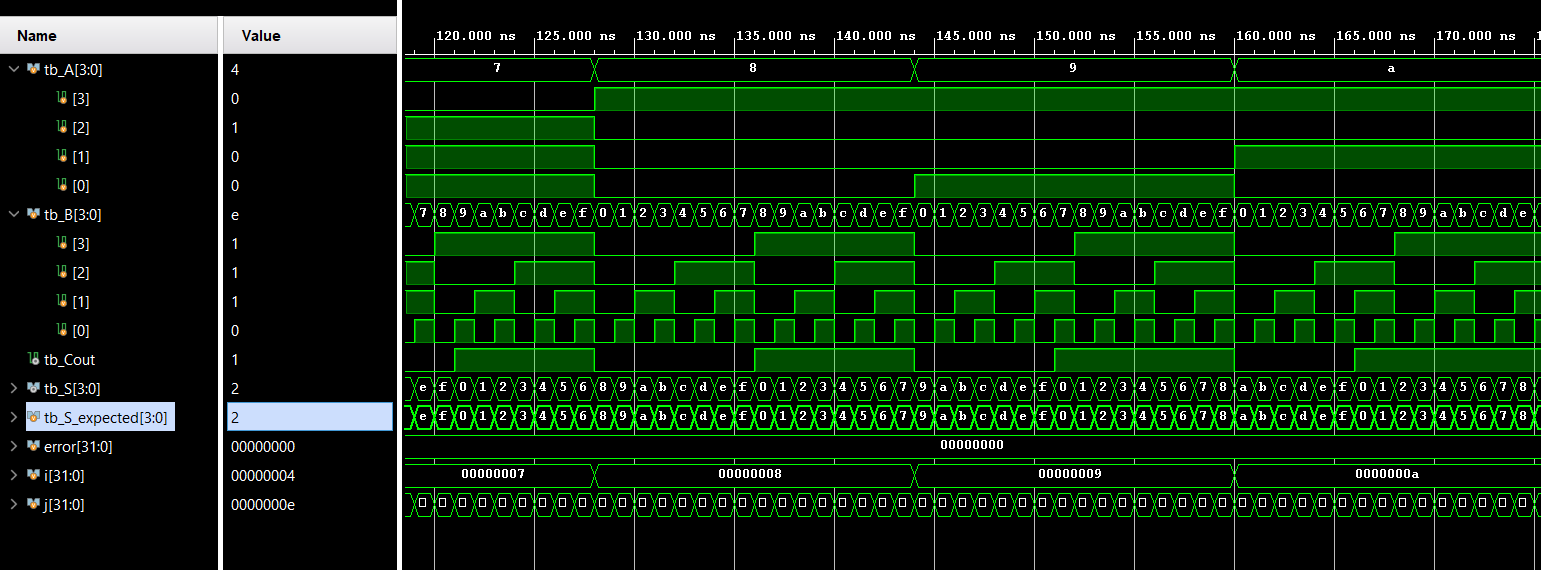
**Simulation results**

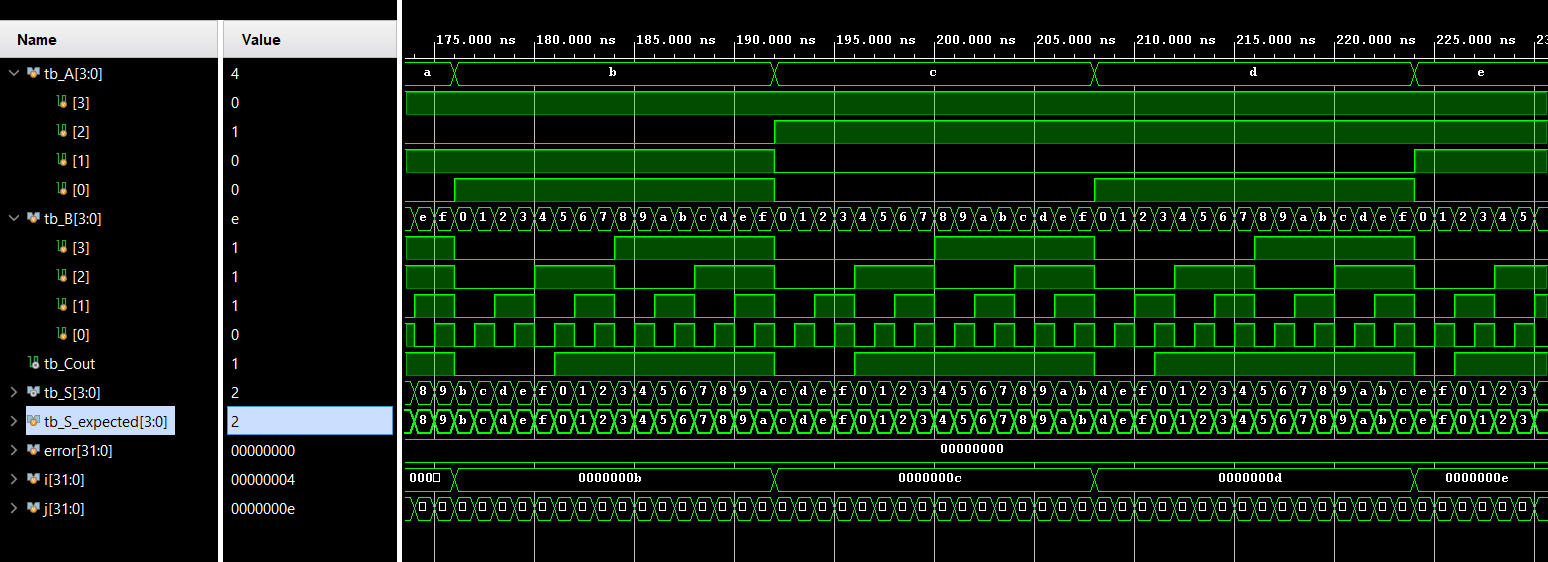


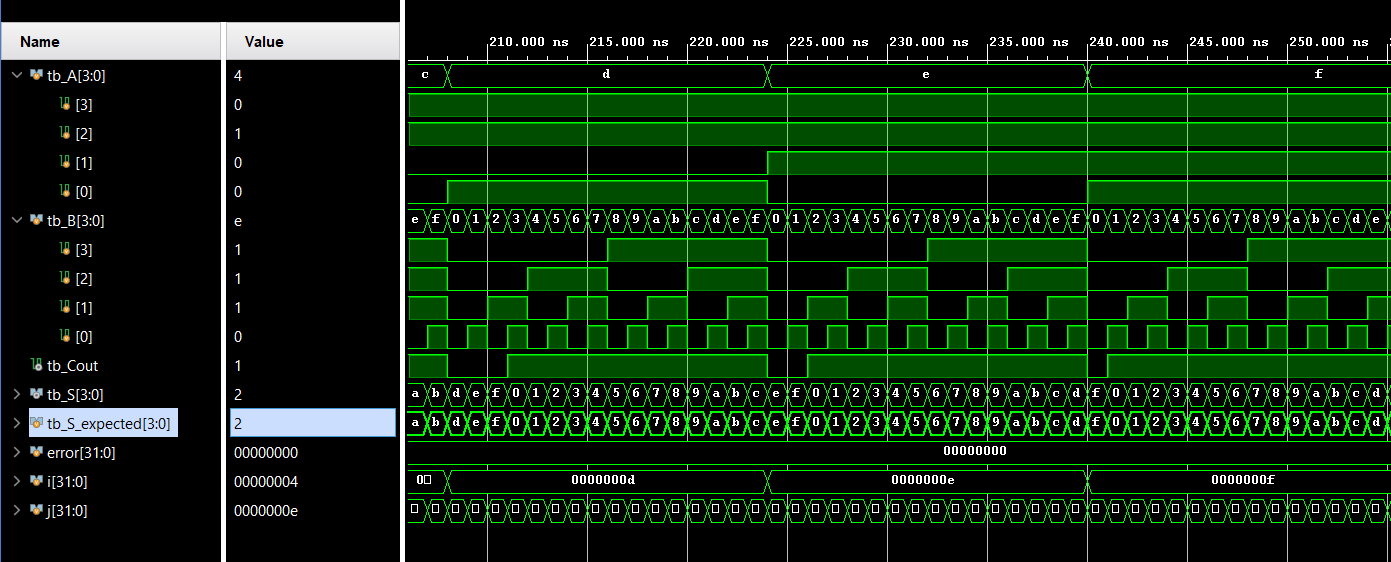
*Figure 7: Inferred method ALU waveform*







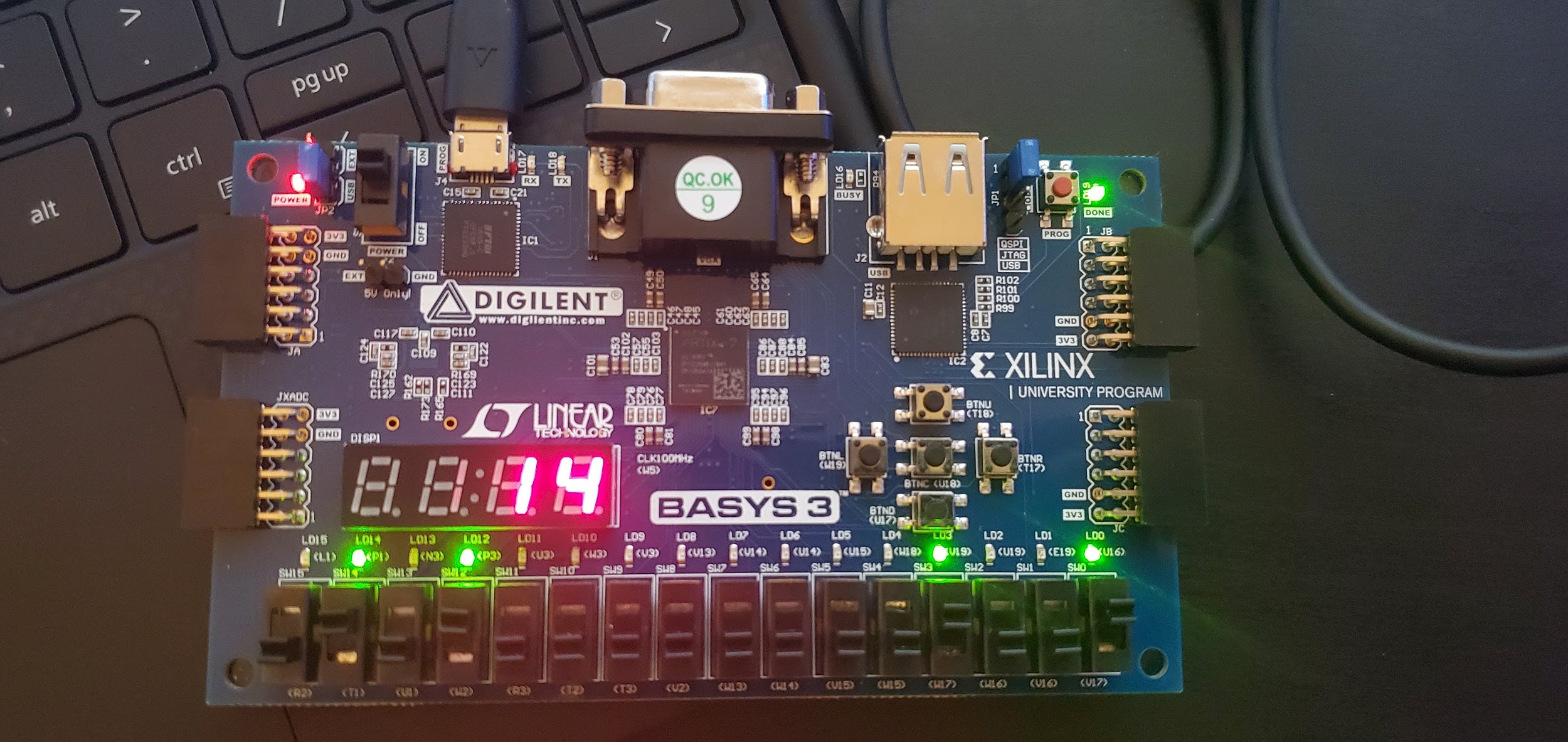
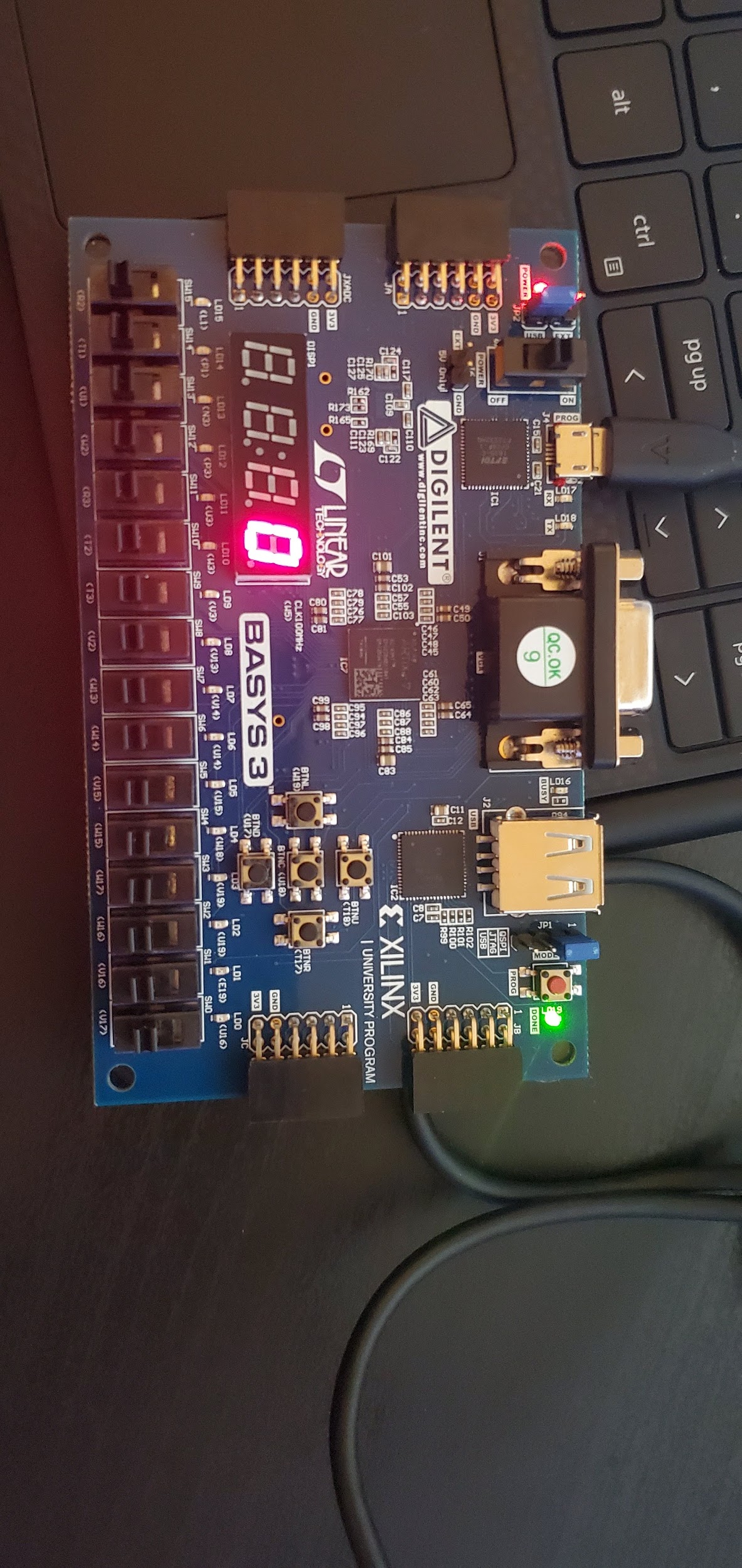




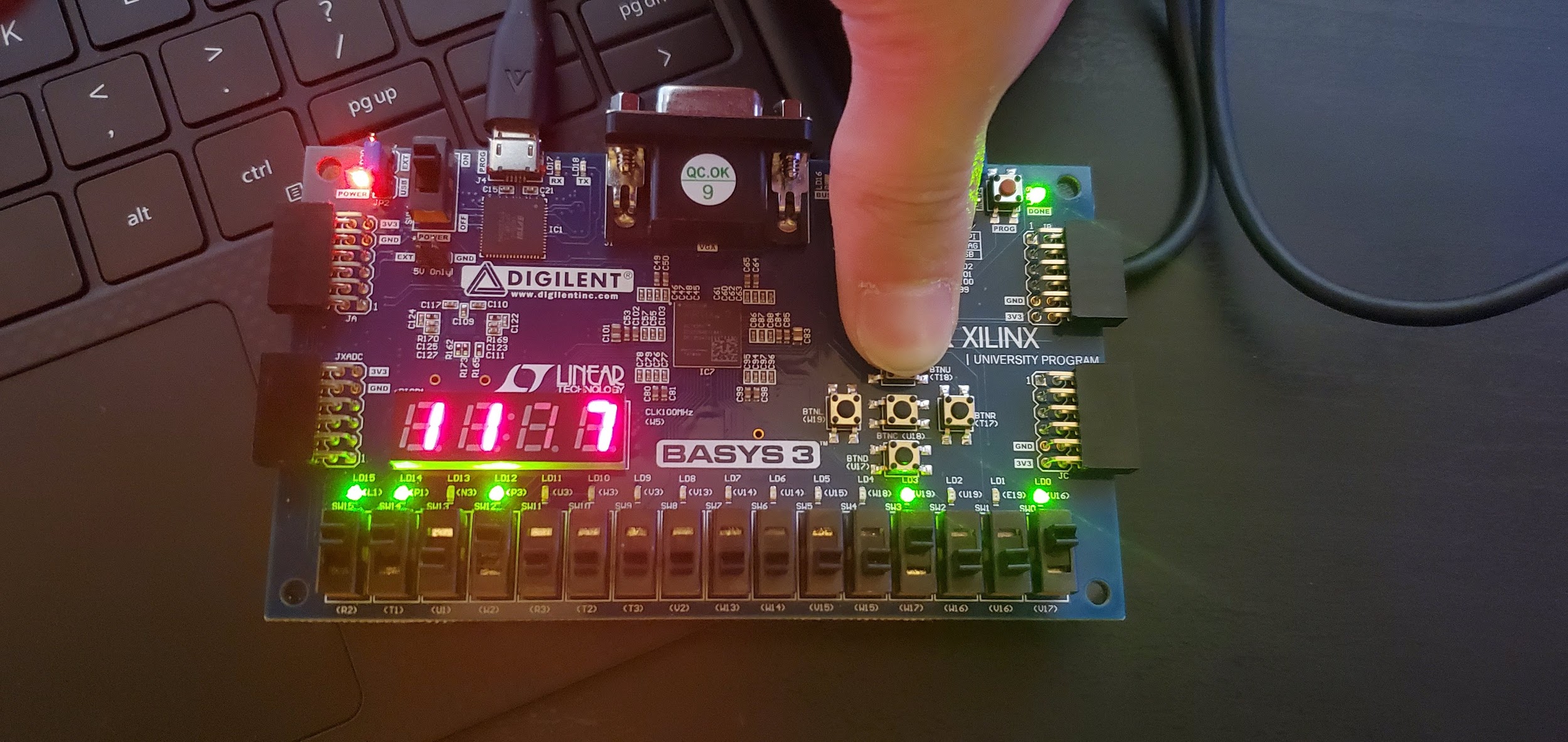
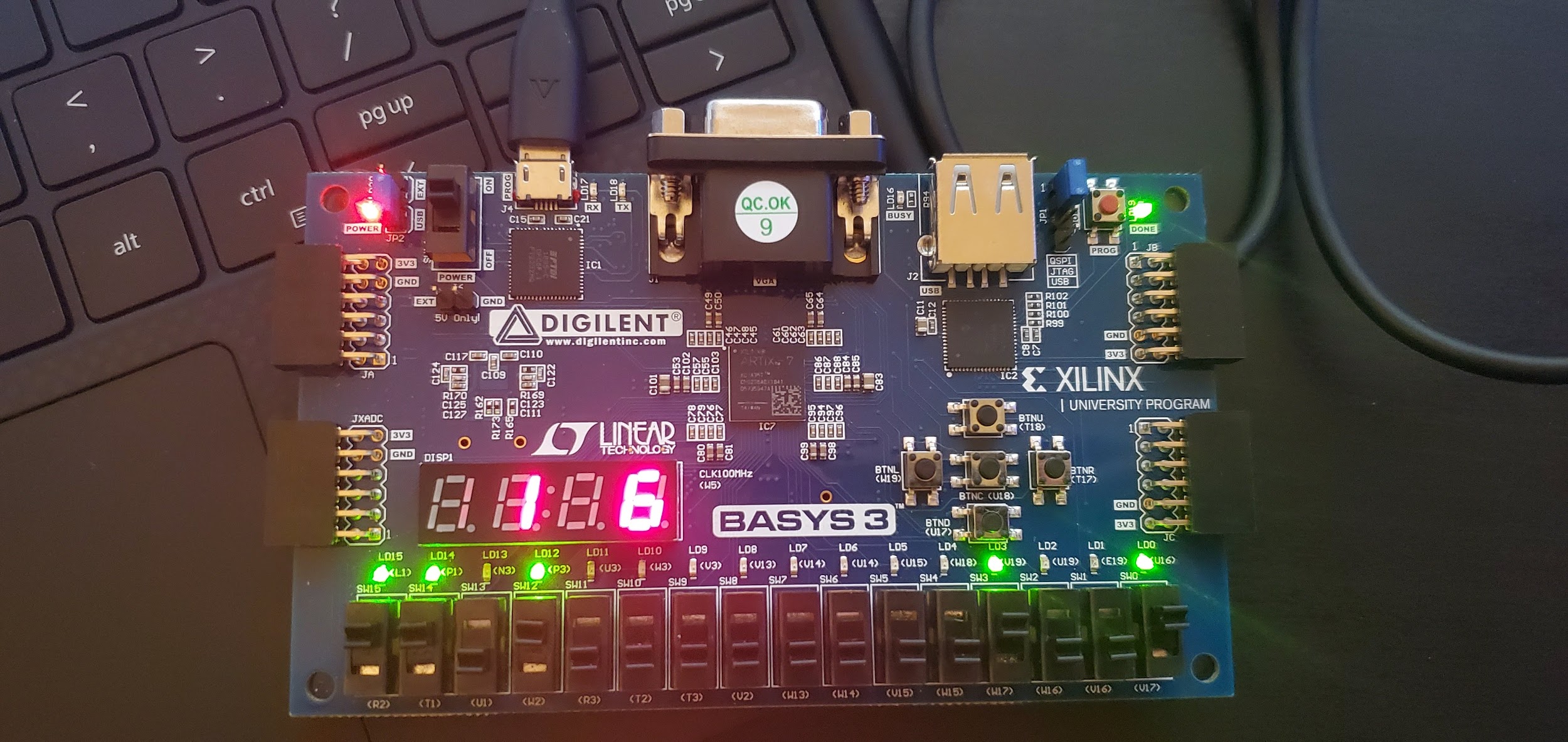
*Figure 8: Carry Look Ahead method ALU waveform*

**FPGA Validation**

When making the constraint file for the BASYS3, we made the left two 7-segment LED show the sum of the two inputs, with a max value of 15. When the sum goes past 16, then the 3rd 7-segment LED will turn on showing the carry over bit since 4 bit can only represent up till 15. The left two LED will show the remainder of the sum. When the carry in button is pressed, we added 1 to the two inputs and the most right LED will turn on, indicating that it is on.



*Figure 9: Left-No switches on, Right-B[2]B[0] and A[3]A[0] on*



*Figure 10: Left-B[3]B[2]B[0] and A[3]A[0] on,, Right-B[3]B[2]B[0] and A[3]A[0]and carry switch on*

**Conclusion**

The purpose of this lab is to familiarize ourselves with the hierarchical design methodology and design/verification /validation flow and the EDA tool, as well as learning about the effect of propagation delay in an adder. We were successful in designing a 4-bit adder and running a simulation test via “inferred” design approach which required us to use the “+” sign. We were also successful designing and running a 4-bit adder using the carry look ahead method and implementing it on our BASYS3 board.

In designing our adder, we learned about the differences between a carry look ahead approach and a ripple carry approach. When we do the CLA method, we can compute all of the carry bits in parallel. Although this is important in reducing propagation delay, its disadvantage is that it requires a lot of gates if we were to make a 64 bit adder. To get the best of both propagation delay and the number of gates, it is possible to make multiple 4 bit CLA adders and then link these together using the ripple carry method to get to 64 bits. This was not a part of the lab, but our design gave us insight towards what we could do in a higher level design. Overall, the lab was successful and we learned how to program a button in our constraint file, which we had not previously done before.

**Appendix**

|  |
| --- |
| inferred\_adder.v |
| module inferred\_adder(sum, c\_out, a, b);  output reg [3:0] sum;  output reg c\_out;  input wire [3:0] a;  input wire [3:0] b;  always @ (a, b)  begin  if(a + b > 15) begin  c\_out = 1;  sum = a + b;  end  else begin  c\_out = 0;  sum = a + b;  end  end  endmodule |

|  |
| --- |
| inferred\_adder\_self\_checkingtb.v |
| `timescale 1ns / 1ps  module self\_checking\_tb;  wire [3:0] sum\_tb;  wire c\_out\_tb;  reg exp\_c\_out;  reg [3:0] sum\_exp;  reg [3:0] a\_tb;  reg [3:0] b\_tb;  inferred\_adder DUT(  .sum(sum\_tb),  .c\_out(c\_out\_tb),  .a(a\_tb),  .b(b\_tb)  );  integer i, j;  integer SumErrorCount;  integer CarryErrorCount;  initial  begin  SumErrorCount = 0;  CarryErrorCount = 0;  for(i = 4'b0000; i <= 4'b1111; i = i + 4'b0001) begin    a\_tb = i;  for(j = 4'b0000; j <= 4'b1111; j = j + 4'b0001) begin  b\_tb = j;    if(sum\_exp[3:0] === sum\_tb[3:0]) begin end  else begin SumErrorCount = SumErrorCount + 1; $display ("sum\_exp =");  $display (sum\_exp);  $display ("sum\_tb = ");  $display (sum\_tb); end  if(exp\_c\_out === c\_out\_tb) begin end  else begin CarryErrorCount = CarryErrorCount + 1; $display ("carry exp =");  $display (exp\_c\_out);  $display ("c\_out\_tb = ");  $display (c\_out\_tb); end  sum\_exp = i + j;  exp\_c\_out = 0;  if(i + j > 15)begin exp\_c\_out = 1; end  #5;  end  end  $display ("Simulation Finished");  $finish;  end  endmodule |

|  |
| --- |
| clk\_gen.v |
| `timescale 1ns / 1ps  module tb\_cla\_adder;  reg [3:0] tb\_A;  reg [3:0] tb\_B;  reg tb\_Cin;  wire tb\_Cout;  wire [3:0] tb\_S;  reg [3:0]tb\_S\_expected;  integer error = 0;  integer i, j;    cla\_adder DUT(.A(tb\_A), .B(tb\_B), .C\_in(tb\_Cin), .C\_out(tb\_Cout), .Sum(tb\_S));  initial begin  for(i = 0; i < 16; i = i + 1) begin  for(j = 0; j < 16; j = j + 1) begin  tb\_Cin = 0;  tb\_A = i;  tb\_B = j;  tb\_S\_expected = i + j;  #1;  if(tb\_S != tb\_S\_expected) begin  $display("Error");  error =+1 ;  end  else begin  end  end  end  end  endmodule |

|  |
| --- |
| clk\_gen.v |
| 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 |

|  |
| --- |
| bcd\_to\_7seg.v |
| module bcd\_to\_7seg (  input wire [3:0] BCD,  output reg [7:0] s  );  always @ (BCD) begin  case (BCD)  4'd0: s = 8'b11000000;  4'd1: s = 8'b11111001;  4'd2: s = 8'b10100100;  4'd3: s = 8'b10110000;  4'd4: s = 8'b10011001;  4'd5: s = 8'b10010010;  4'd6: s = 8'b10000010;  4'd7: s = 8'b11111000;  4'd8: s = 8'b10000000;  4'd9: s = 8'b10010000;  default: s = 8'b01111111;    endcase  end  endmodule |

|  |
| --- |
| LEDSEL.v |
| module LEDSEL (  input wire clk,  input wire rst,  input wire [7:0] LED3,  input wire [7:0] LED2,  input wire [7:0] LED1,  input wire [7:0] LED0,  output wire [3:0] LEDOUT,  output wire [7:0] LEDOUT  );  reg [1:0] index;  reg [11:0] led\_ctrl;  assign {LEDOUT, LEDOUT} = led\_ctrl;  always @ (posedge clk) index <= (rst) ? 2'b0 : (index + 2'd1);  always @ (index, LED0, LED1, LED2, LED3) begin  case (index)  4'd0: led\_ctrl <= {4'b1110, LED0};  4'd1: led\_ctrl <= {4'b1101, LED1};  4'd2: led\_ctrl <= {4'b1011, LED2};  4'd3: led\_ctrl <= {4'b0111, LED3};  default: led\_ctrl <= {8'b1111, 8'hFF};  endcase  end  endmodule |

|  |
| --- |
| cla\_adder.v |
| module cla\_adder(  input [3:0]A,  input [3:0]B,  input C\_in,  output C\_out,  output [3:0] Sum  );  wire [3:0] P;  wire [3:0] G;  wire [4:0] C;  carryBits cBits(.P(P), .G(G), .C\_in(C\_in), .C\_out(C));  Add\_half AH0(.a(A[0]), .b(B[0]), .c\_out(P[0]), .sum(G[0]));  Add\_half AH1(.an(A[1]), .b(B[1]), .c\_out(P[1]), .sum(G[1]));  Add\_half AH2(.a(A[2]), .b(B[2]), .c\_out(P[2]), .sum(G[2]));  Add\_half AH3(.a(A[3]), .b(B[3]), .c\_out(P[3]), .sum(G[3]));  assign Sum[0] = P[0] ^ C[0];  assign Sum[1] = P[1] ^ C[1];  assign Sum[2] = P[2] ^ C[2];  assign Sum[3] = P[3] ^ C[3];  assign C\_out = C[4];  endmodule |

|  |
| --- |
| cla\_adder\_fpga.v |
| `timescale 1ns / 1ps  module cla\_adder\_fpga(  input wire clk100mhz,  input wire cin\_button,  input wire [3:0]inputA,  input wire [3:0]inputB,  output reg [3:0]outputA,  output reg [3:0]outputB,  output wire [7:0]LEDOUT,  output wire [3:0]LEDSEL  );    wire [3:0]sum\_out;  wire carryOut;  wire dontuse;  Wire clk\_5KHz;  Clk\_gen CLK (  .clk100MHz (clk100MHz),  .rst (rst),  .clk\_4sec (dontuse),  .clk\_5KHz (clk\_5KHz)  );  Bcd\_to\_7seg BCD (  .BCD (LEDSEL),  .s (LEDOUT)  );  Led\_mux LED (  .clk (clk\_5KHz),  .rst (rst),  .LED3 (LEDOUT),  .LED2 (LEDOUT),  .LED1 (LEDOUT),  .LED0 (LEDOUT).  .LEDSEL (LEDSEL),  .LEDOUT (LEDOUT)  );  carryBits cBits(  .P(inputA),  .G(inputB),  .C\_in(cin\_button),  .C\_out(carryOut),  );  always @ (inputA, inputB, cin\_button) begin  outputA = inputA;  outputB = inputB;  if(cin\_button) begin  led0 = 4'b0001;  end  else begin  led0 = 4'b1111;  end  if(carryOut) begin  led1 = 4'b0001;  end  else begin  led1 = 4'b1111;  end  if(sum\_out >= 10) begin  led2 = 4'b0001;  led3 = sum\_out - 10;  end  else begin  led2 = 4'b1111;  led3 = sum\_out;  end  end  endmodule |

|  |
| --- |
| carryBits.v |
| module carryBits(  input [3:0]P,  input [3:0]G,  input C\_in,  output[4:0]C\_out  );  assign C\_out[0] = C\_in;  assign C\_out[1] = G[0] | P[0] & C\_in;  assign C\_out[2] = G[1] | P[1] & G[0] | P[1] & P[0] & C\_in;  assign C\_out[3] = G[2] | P[2] & G[1] | P[2] & P[1] & G[0] | P[2] & P[1] & P[0] & C\_in;  assign C\_out[4] = G[3] | P[3] & G[2] | P[3] & P[2] & G[1] | P[3] & P[2] & P[1] & G[0] | P[3] & P[2] & P[1] & P[0] & C\_in;  endmodule |

|  |
| --- |
| Add\_half.v |
| `timescale 1ns / 1ps  module Add\_half(  input a,b;  output c\_out, sum;  );  assign sum = a ^ b;  assign c\_out = a & b;  endmodule |

|  |
| --- |
| cla\_adder\_fpga.xdc |
| # Clock  set\_property -dict {PACKAGE\_PIN W5 IOSTANDARD LVCMOS33} [get\_ports {clk100mhz}];  create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk100mhz}];  # Button  set\_property -dict {PACKAGE\_PIN T18 IOSTANDARD LVCMOS33} [get\_ports {cin\_button}]; #Button - cin  # Switches  set\_property -dict {PACKAGE\_PIN V17 IOSTANDARD LVCMOS33} [get\_ports {inputB[0]}]; # Switch 0  set\_property -dict {PACKAGE\_PIN V16 IOSTANDARD LVCMOS33} [get\_ports {inputB[1]}]; # Switch 1  set\_property -dict {PACKAGE\_PIN W16 IOSTANDARD LVCMOS33} [get\_ports {inputB[2]}]; # Switch 2  set\_property -dict {PACKAGE\_PIN W17 IOSTANDARD LVCMOS33} [get\_ports {inputB[3]}]; # Switch 3  set\_property -dict {PACKAGE\_PIN W2 IOSTANDARD LVCMOS33} [get\_ports {inputA[0]}]; # Switch 4  set\_property -dict {PACKAGE\_PIN U1 IOSTANDARD LVCMOS33} [get\_ports {inputA[1]}]; # Switch 5  set\_property -dict {PACKAGE\_PIN T1 IOSTANDARD LVCMOS33} [get\_ports {inputA[2]}]; # Switch 6  set\_property -dict {PACKAGE\_PIN R2 IOSTANDARD LVCMOS33} [get\_ports {inputA[3]}]; # Switch 7  # LEDs  set\_property -dict {PACKAGE\_PIN U16 IOSTANDARD LVCMOS33} [get\_ports {outputB[0]}]; # LED 0  set\_property -dict {PACKAGE\_PIN E19 IOSTANDARD LVCMOS33} [get\_ports {outputB[1]}]; # LED 1  set\_property -dict {PACKAGE\_PIN U19 IOSTANDARD LVCMOS33} [get\_ports {outputB[2]}]; # LED 2  set\_property -dict {PACKAGE\_PIN V19 IOSTANDARD LVCMOS33} [get\_ports {outputB[3]}]; # LED 3  set\_property -dict {PACKAGE\_PIN P3 IOSTANDARD LVCMOS33} [get\_ports {outputA[0]}]; # LED 4  set\_property -dict {PACKAGE\_PIN N3 IOSTANDARD LVCMOS33} [get\_ports {outputA[1]}]; # LED 5  set\_property -dict {PACKAGE\_PIN P1 IOSTANDARD LVCMOS33} [get\_ports {outputA[2]}]; # LED 6  set\_property -dict {PACKAGE\_PIN L1 IOSTANDARD LVCMOS33} [get\_ports {outputA[3]}]; # LED 7  # 7 Segment Display  set\_property -dict {PACKAGE\_PIN U2 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[0]}]; # mux 0  set\_property -dict {PACKAGE\_PIN U4 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[1]}]; # mux 1  set\_property -dict {PACKAGE\_PIN V4 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[2]}]; # mux 2  set\_property -dict {PACKAGE\_PIN W4 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[3]}]; # mux 3  set\_property -dict {PACKAGE\_PIN W7 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[0]}]; # out 0  set\_property -dict {PACKAGE\_PIN W6 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[1]}]; # out 1  set\_property -dict {PACKAGE\_PIN U8 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[2]}]; # out 2  set\_property -dict {PACKAGE\_PIN V8 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[3]}]; # out 3  set\_property -dict {PACKAGE\_PIN U5 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[4]}]; # out 4  set\_property -dict {PACKAGE\_PIN V5 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[5]}]; # out 5  set\_property -dict {PACKAGE\_PIN U7 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[6]}]; # out 6  set\_property -dict {PACKAGE\_PIN V7 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[7]}]; # out 7 |