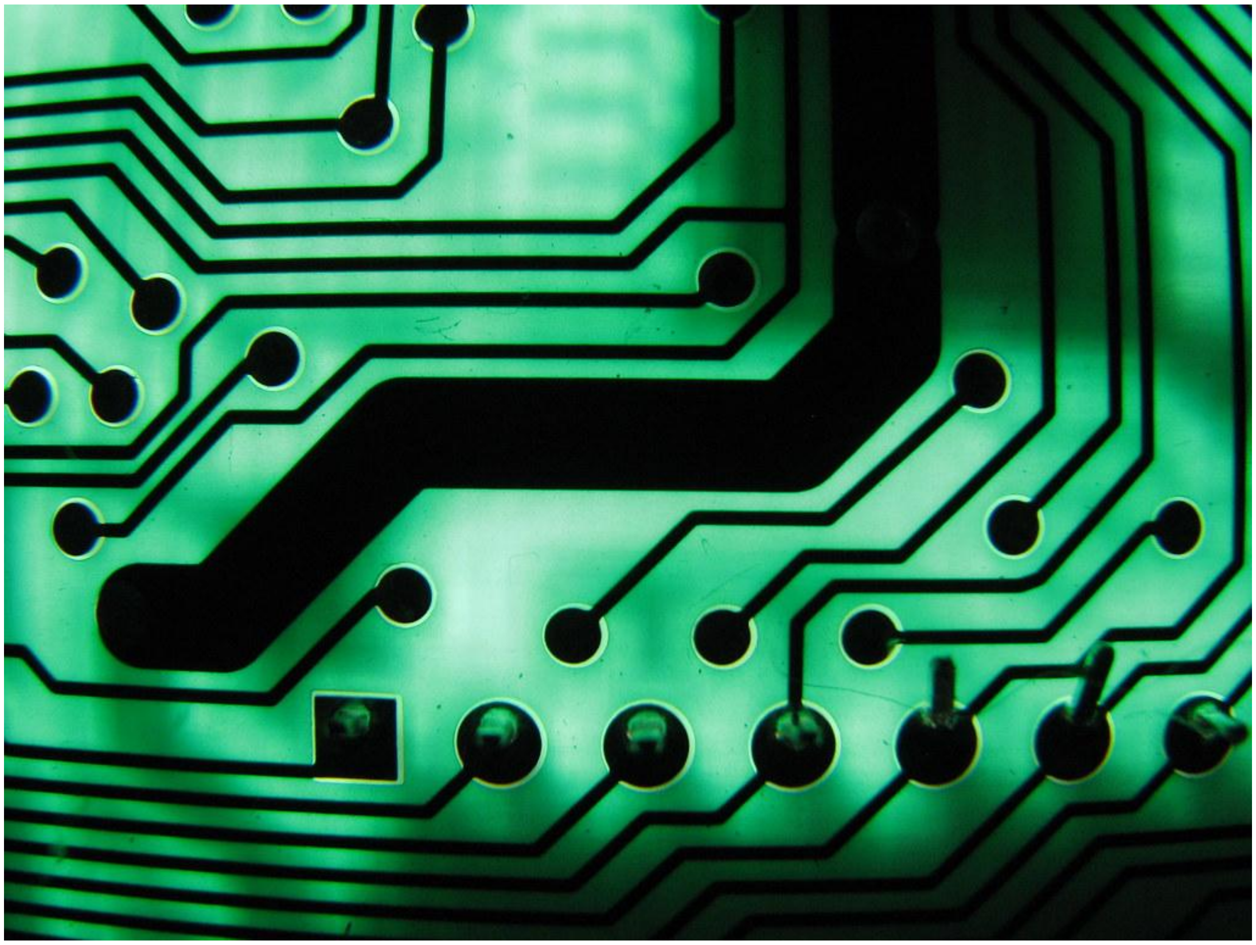
EEC 180 DIGITAL SYSTEMS II

Lab #5
Counter Design



Introduction

The purpose of this lab is to program an FPGA to design a counter.

Tobin Joseph

Results

1 Prelab

Part 1.1 Preliminary design

Part 1.1.1 Block Diagram

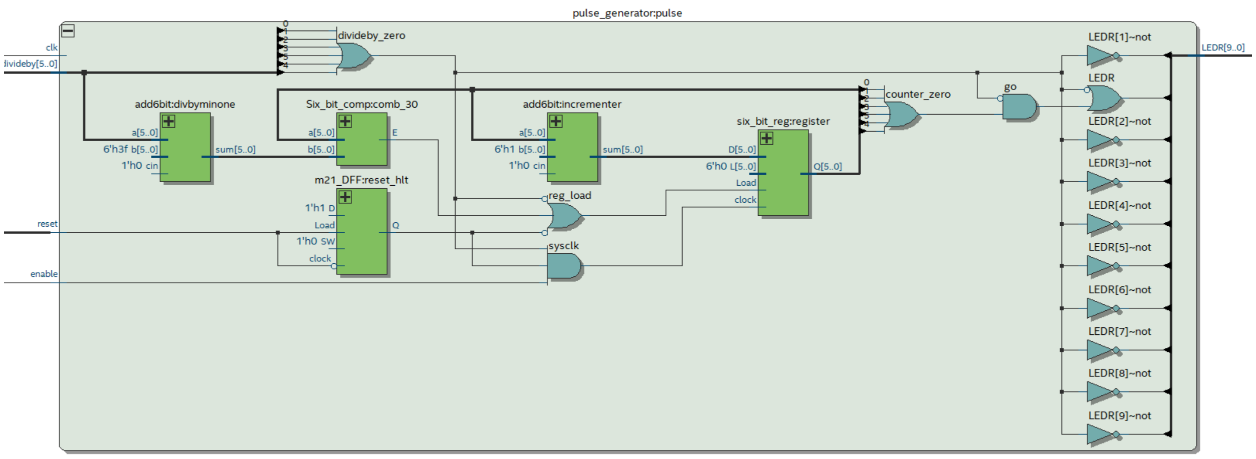


Figure 1.1.1.1 Preliminary block diagram for counter 1.

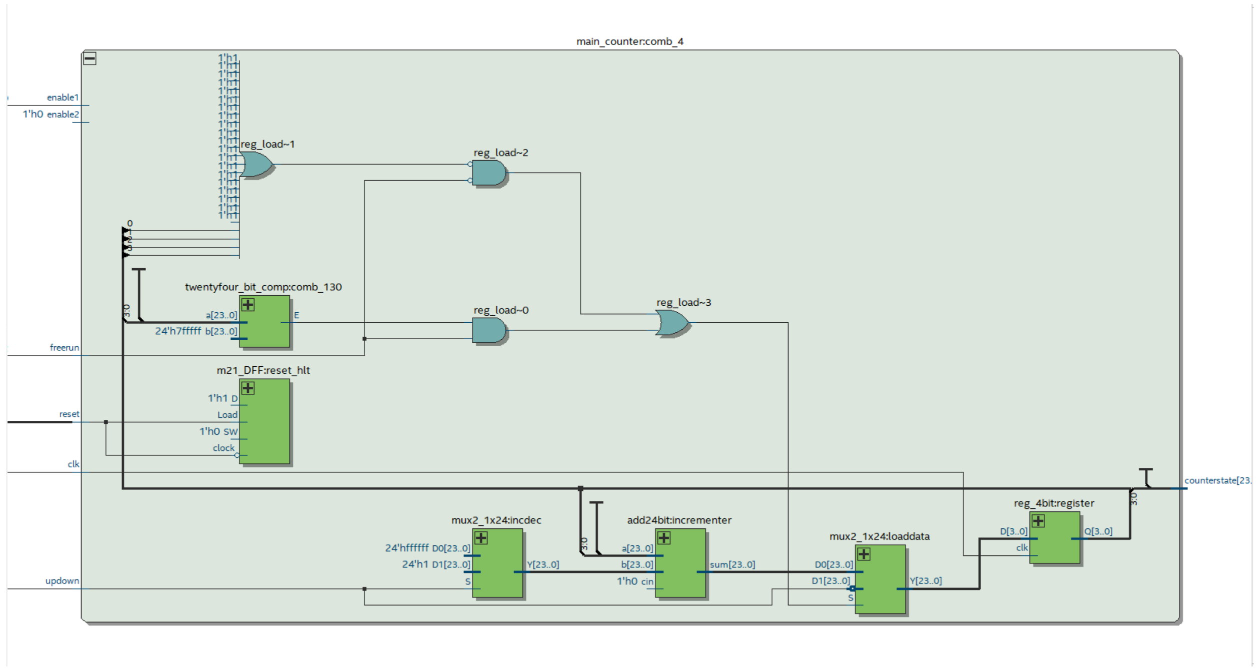


Figure 1.1.1.2 Preliminary block diagram for counter 2.

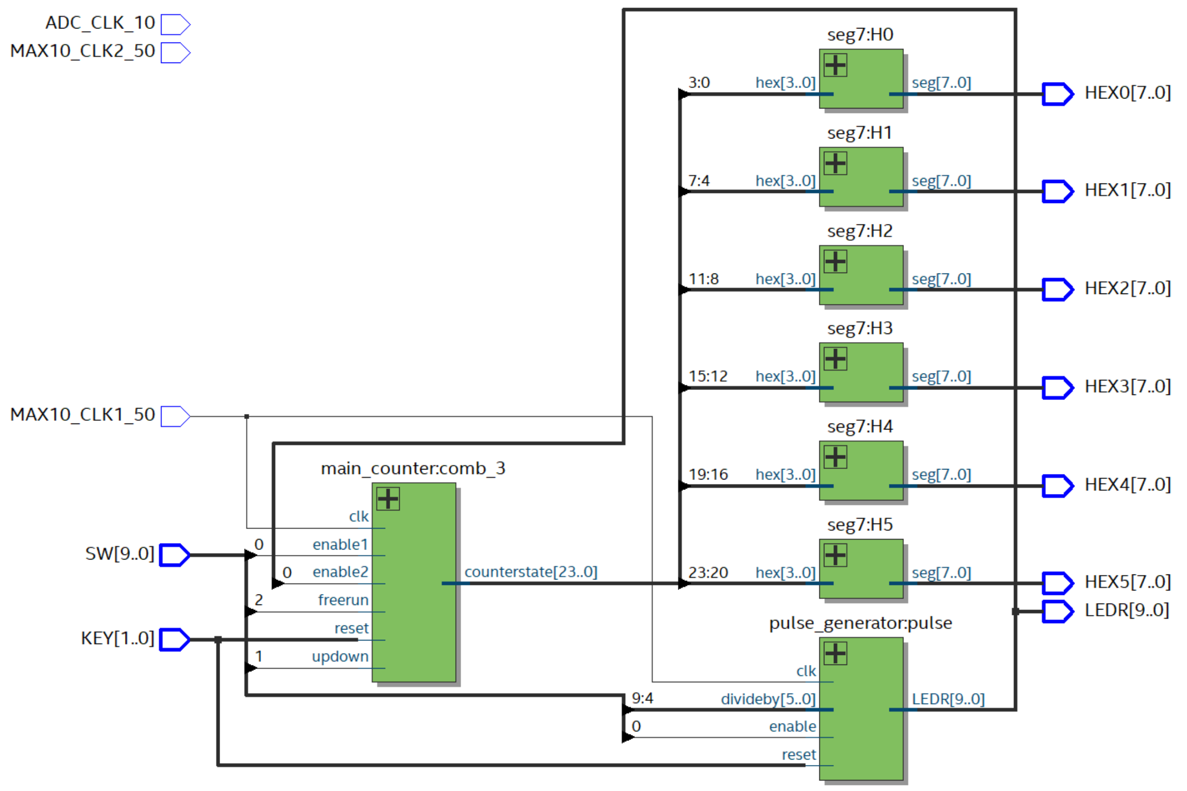


Figure 1.1.1.3 Preliminary block diagram for circuit

Part 1.1.2 Timing Diagram for Counter 1

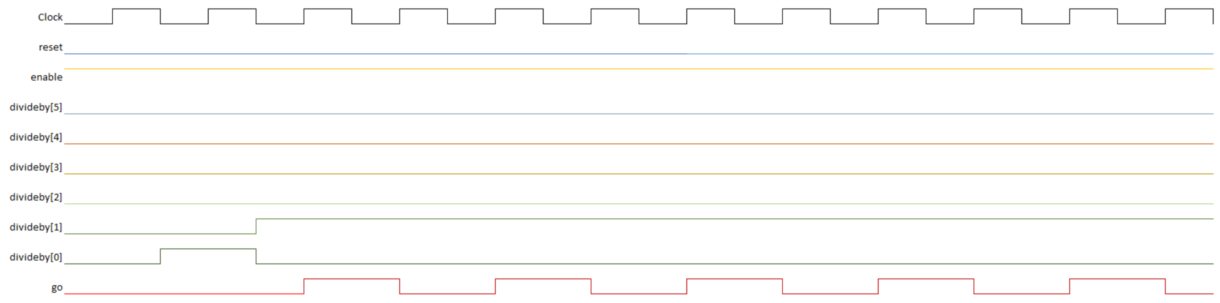


Figure 1.1.2.1 Preliminary timing diagram for counter 1.

Part 1.1.3 Timing Diagram for Counter 2

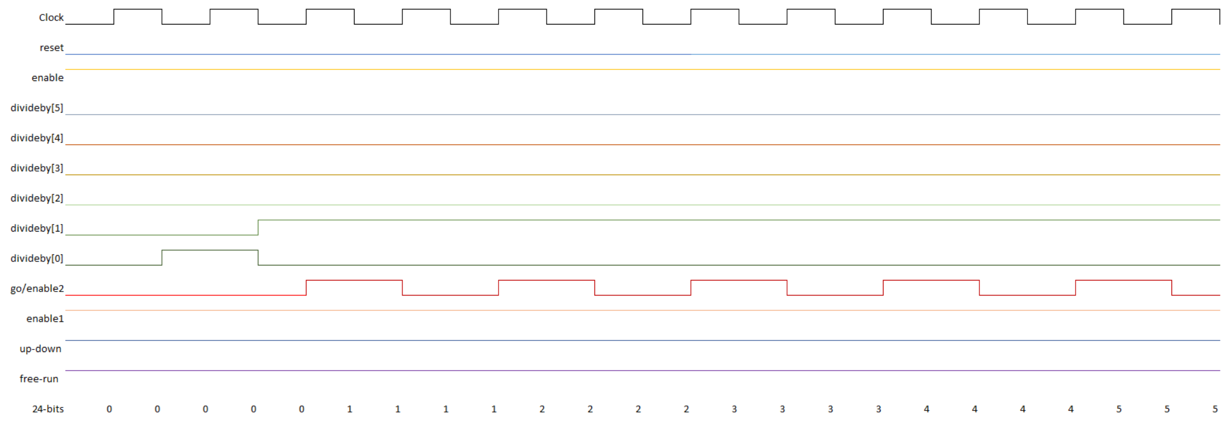


Figure 1.1.3.1 Preliminary timing diagram for counter 2.

Part 1.1.4 Preliminary Verilog Code

```
//Verilog fa.v module.
module fa (in1, in2, cin, sum, cout);

      //Declare inputs, outputs, and internal variables.
      input in1,
                  in2,
                  cin;
      output sum,
                  cout;

            // use Boolean expressions
      assign sum  = (in1 ^ in2) ^ cin;
      assign cout = (in1 & in2) | (in2 & cin) | (cin & in1);
endmodule

//Verilog add6bit.v module.
```

```
module add6bit(a,b,cin,sum,cout);

      //Declare inputs, outputs, and internal variables.
      input [5:0]a,b;
      input cin;
      output wire [5:0]sum;
      output cout;

      // instantiating four full-adder circuits - ripple carry
      fa FA1(a[0],b[0],cin,sum[0],cout1);
      fa FA2(a[1],b[1],cout1,sum[1],cout2);
      fa FA3(a[2],b[2],cout2,sum[2],cout3);
      fa FA4(a[3],b[3],cout3,sum[3],cout4);
      fa FA5(a[4],b[4],cout4,sum[4],cout5);
      fa FA6(a[5],b[5],cout5,sum[5],cout);
endmodule

// Verilog code for rising edge D flip flop
module RE_DFF(D,clk,Q);
      input D; // Data input
      input clk; // clock input
      output reg Q; // output Q

      always @(posedge clk)
      begin
            Q <= D;
      end
endmodule

module mux2_1(Y, D0, D1, S);

      output Y;
      input D0, D1, S;
      wire T1, T2, Sbar;

      not (Sbar, S);
      and (T1, D1, S), (T2, D0, Sbar);
      or (Y, T1, T2);

endmodule



module m21_DFF(Q,D,SW,Load,clock);

      output Q;
      input wire D;
      input wire SW;
      input wire Load;
      input wire clock;

      wire selD;

      mux2_1 mux(selD,D,SW,Load);
      RE_DFF dff(selD,clock,Q);

endmodule



module six_bit_reg(Q,D,L,Load,clock);
      output [5:0]Q;
      input [5:0]D,L;
      input Load, clock;

      wire [5:0]di;

      m21_DFF q0(Q[0],D[0],L[0],Load,clock);
      m21_DFF q1(Q[1],D[1],L[1],Load,clock);
      m21_DFF q2(Q[2],D[2],L[2],Load,clock);
      m21_DFF q3(Q[3],D[3],L[3],Load,clock);
      m21_DFF q4(Q[4],D[4],L[4],Load,clock);
      m21_DFF q5(Q[5],D[5],L[5],Load,clock);
endmodule

module two_bit_comp(G, L, E, a, b);
    input [1:0]a,b;
    output G, L, E;

    assign G = ((a[1])&(~b[1]))|(((a[1])~^(b[1]))&((a[0])&(~b[0])));
    assign L = ((~a[1])&(b[1]))|(((a[1])~^(b[1]))&((~a[0])&(b[0])));
    assign E = (a[1]~^b[1])&(a[0]~^b[0]);
endmodule

module Six_bit_comp(G, L, E, a, b);
    input [5:0] a,b ;
    output G, L, E;

    wire g1, g2, g3, l1, l2, l3, e1, e2, e3;

    two_bit_comp comp1(g3, l3, e3, a[5:4], b[5:4]);
    two_bit_comp comp2(g2, l2, e2, a[3:2], b[3:2]);
    two_bit_comp comp3(g1, l1, e1, a[1:0], b[1:0]);



    assign G = (g3)|(e3&g2)|(e3&e2&g1);
    assign L = (l3)|(e3&l2)|(e3&e2&l1);
    assign E = e1&e2&e3;
endmodule
```

```
module pulse_generator(LEDR, reset, enable, divideby, clk);
      // output and inputs
      output [9:0] LEDR;
      input reset;
      input enable;
      input [5:0]divideby;
      input clk;

      wire [5:0] Q;
      wire [5:0] zero;
      wire [5:0] one;
      wire [5:0] counter_out;
      wire [5:0] minusone, divminone;
      wire sysclk;
      wire reselt_halt, reg_load, divideby_zero, counter_zero, resetbar,
cin0, cin1, cout, go, G, L, E;

      assign one = 6'b000001;
      assign resetbar = ~reset;
      assign cin0 = 1'b0;
      assign zero = 6'b000000;
      assign minusone = 6'b111111;



      // reset input, halts when high
      m21_DFF reset_hlt(reselt_halt,one,z,reset,resetbar);

      // divideby is zero
      assign divideby_zero = (divideby[5] | divideby[4] | divideby[3] |
divideby[2] | divideby[1] | divideby[0]);

      // define sysclk
      // divideby = 000000: Halt counter with value 000000.
```

```
	// enable input, disables operation when low
	// reset input, halts when high
	assign sysclk = (divideby_zero & enable & reselt_halt);

	// counter is equals to divideby -1
	add6bit divbyminone(divideby,minusone,cin0,divminone,cout);
	Six_bit_comp(G, L, E, Q, divminone);

	// counter is zero
	assign counter_zero = Q[5] | Q[4] | Q[3] | Q[2] | Q[1] | Q[0];

	// pulse output go output, is high when the state of the counter is
000000 and the input divideby is not 000000
	assign go = (~divideby_zero) & counter_zero;

	// register load on 1, reset sets count to 000000, counter is equals to
divideby -1, dividedby 000000 Halt counter with value 000000
	assign reg_load = (~reselt_halt | E | ~divideby_zero );

	// create incrementer
	add6bit incrementer(Q,one,cin0,counter_out,cout);

	// create register, on load set to 000000
	six_bit_reg register(Q,counter_out,zero,reg_load,sysclk);

	// set LEDS
	assign LEDR[0] = ~divideby_zero | go;
	assign LEDR[1] = ~divideby_zero;
	assign LEDR[2] = ~divideby_zero;
	assign LEDR[3] = ~divideby_zero;
	assign LEDR[4] = ~divideby_zero;
	assign LEDR[5] = ~divideby_zero;
	assign LEDR[6] = ~divideby_zero;
```

```
      assign LEDR[7] = ~divideby_zero;
      assign LEDR[8] = ~divideby_zero;
      assign LEDR[9] = ~divideby_zero;

endmodule

module four_bit_comp(G, L, E, a, b);
    input [3:0] a,b ;
    output G, L, E;

    wire g1, g2, l1, l2, e1, e2;

    two_bit_comp comp1(g2, l2, e2, a[3:2], b[3:2]);
    two_bit_comp comp2(g1, l1, e1, a[1:0], b[1:0]);

    assign G = (g2)|(e2&g1);
    assign L = (l2)|(e2&l1);
    assign E = e1&e2;
endmodule

module eight_bit_comp(G, L, E, a, b);
    input [7:0] a,b ;
    output G, L, E;

    wire g1, g2, l1, l2, e1, e2;

    four_bit_comp comp1(g2, l2, e2, a[7:4], b[7:4]);
    four_bit_comp comp2(g1, l1, e1, a[3:0], b[3:0]);

    assign G = (g2)|(e2&g1);
    assign L = (l2)|(e2&l1);
    assign E = e1&e2;
endmodule
```

```
module twentyfour_bit_comp(G, L, E, a, b);
    input [23:0] a,b ;
    output G, L, E;

    wire g1, g2, g3, l1, l2, l3, e1, e2, e3;

    eight_bit_comp comp1(g3, l3, e3, a[23:16], b[23:16]);
    eight_bit_comp comp2(g2, l2, e2, a[18:8], b[15:8]);
    eight_bit_comp comp3(g1, l1, e1, a[7:0], b[7:0]);



    assign G = (g3)|(e3&g2)|(e3&e2&g1);
    assign L = (l3)|(e3&l2)|(e3&e2&l1);
    assign E = e1&e2&e3;
endmodule

module add24bit(a,b,cin,sum,cout);

      //Declare inputs, outputs, and internal variables.
      input [23:0]a,b;
      input cin;
      output wire [23:0]sum;
      output cout;

      // instantiating four full-adder circuits - ripple carry
      add6bit FA1(a[5:0],b[5:0],cin,sum[5:0],cout1);
      add6bit FA2(a[11:6],b[11:6],cout1,sum[11:6],cout2);
      add6bit FA3(a[17:12],b[17:12],cout2,sum[17:12],cout3);
      add6bit FA4(a[23:18],b[23:18],cout3,sum[23:18],cout);
endmodule

module mux2_1x4(Y, D0, D1, S);
      output [3:0]Y;
```

```
	input [3:0]D0,D1;
	input S;



	mux2_1 mux0(Y[0],D0[0],D1[0],S);
	mux2_1 mux1(Y[1],D0[1],D1[1],S);
	mux2_1 mux2(Y[2],D0[2],D1[2],S);
	mux2_1 mux3(Y[3],D0[3],D1[3],S);

endmodule

module mux2_1x24(Y, D0, D1, S);
	output [23:0]Y;
	input [23:0]D0,D1;
	input S;

	mux2_1x4 mux0(Y[3:0],D0[3:0],D1[3:0],S);
	mux2_1x4 mux1(Y[7:4],D0[7:4],D1[7:4],S);
	mux2_1x4 mux2(Y[11:8],D0[11:8],D1[11:8],S);
	mux2_1x4 mux3(Y[15:12],D0[15:12],D1[15:12],S);
	mux2_1x4 mux4(Y[19:16],D0[19:16],D1[19:16],S);
	mux2_1x4 mux5(Y[23:20],D0[23:20],D1[23:20],S);

endmodule

module reg_4bit(Q,D,clk);
	output [3:0]Q;
	input [3:0]D;
	input clk;

	RE_DFF dff0(D[0],clk,Q[0]);
	RE_DFF dff1(D[1],clk,Q[1]);
	RE_DFF dff2(D[2],clk,Q[2]);
	RE_DFF dff3(D[3],clk,Q[3]);

endmodule



module reg_24bit(Q,D,clk);
	output [23:0]Q;
	input [23:0]D;
	input clk;

	reg_4bit dff0(D[3:0],clk,Q[3:0]);
	reg_4bit dff1(D[7:4],clk,Q[7:4]);
	reg_4bit dff2(D[11:8],clk,Q[11:8]);
	reg_4bit dff3(D[15:12],clk,Q[15:12]);
	reg_4bit dff4(D[19:16],clk,Q[19:16]);
	reg_4bit dff5(D[23:20],clk,Q[23:20]);

endmodule

module main_counter(counterstate, reset, enable1, enable2, updown, freerun,
clk);
	// output and inputs
	output [23:0] counterstate;
	input reset;
	input enable1;
	input enable2;
	input updown;
	input freerun;
	input clk;

	wire [23:0] Q;
	wire [23:0] zero;
	wire [23:0] one;
	wire [23:0] counter_out, increment_decrement, counter_start, load_data, halfmax;
	wire [23:0] minusone;
	wire sysclk;
	wire reselt_halt, reg_load, counter_zero, resetbar, cin0, cin1, cout, go, G, L, E;
	wire o,z;
	wire updown_bar;

	assign one = 24'b000000000000000000000001;
	assign resetbar = ~reset;
	assign cin0 = 1'b0;
	assign zero = 24'b000000000000000000000000;
	assign minusone = 24'b111111111111111111111111;
	assign halfmax = 24'b011111111111111111111111;
	assign o = 1'b1;
	assign z = 1'b0;

	assign updown_bar = ~updown;
	assign counter_start[0] = updown_bar;
	assign counter_start[1] = updown_bar;
	assign counter_start[2] = updown_bar;
	assign counter_start[3] = updown_bar;
	assign counter_start[4] = updown_bar;
	assign counter_start[5] = updown_bar;
	assign counter_start[6] = updown_bar;
	assign counter_start[7] = updown_bar;
	assign counter_start[8] = updown_bar;
	assign counter_start[9] = updown_bar;
	assign counter_start[10] = updown_bar;
	assign counter_start[11] = updown_bar;
	assign counter_start[12] = updown_bar;
	assign counter_start[13] = updown_bar;
	assign counter_start[14] = updown_bar;
	assign counter_start[15] = updown_bar;
	assign counter_start[16] = updown_bar;
	assign counter_start[17] = updown_bar;
	assign counter_start[18] = updown_bar;
	assign counter_start[19] = updown_bar;
	assign counter_start[20] = updown_bar;
	assign counter_start[21] = updown_bar;
	assign counter_start[22] = updown_bar;
	assign counter_start[23] = updown_bar;

	// reset input, halts when high
	m21_DFF reset_hlt(reselt_halt,one,z,reset,resetbar);

	// define sysclk
	// enable inputs, disables operation when low
	// reset input, halts when high
	assign sysclk = (enable1 & enable2 & reselt_halt);

	// increment or decrement
	mux2_1x24 incdec(increment_decrement, minusone, one, updown);

	// create incrementer
	add24bit incrementer(Q,increment_decrement,cin0,counter_out,cout);

	// half way comparator
	twentyfour_bit_comp(G, L, E, Q, halfmax);

	// register load on 1, reset sets count to 000000, counter is equals to
divideby -1, dividedby 000000 Halt counter with value 000000
	assign reg_load = (freerun & E) | (~freerun & ~(Q[23] | Q[22] | Q[21] |
Q[20] | Q[19] | Q[18] | Q[17] | Q[16] | Q[15] | Q[14] | Q[13] | Q[12] | Q[11]
| Q[10] | Q[9] | Q[8] | Q[7] | Q[6] | Q[5] | Q[4] | Q[3] | Q[2] | Q[1] |
Q[0]));

      // load
      mux2_1x24 loaddata(load_data, counter_out, counter_start, reg_load);
      reg_4bit register(Q,load_data,clk);

      // set counter state
      assign counterstate = Q;

endmodule

module hw2_top(

      //////////// CLOCK //////////
      input                                   ADC_CLK_10,
      input                                   MAX10_CLK1_50,
      input                                   MAX10_CLK2_50,

      //////////// SEG7 //////////
      output                  [7:0]           HEX0,
      output                  [7:0]           HEX1,
      output                  [7:0]           HEX2,
      output                  [7:0]           HEX3,
      output                  [7:0]           HEX4,
      output                  [7:0]           HEX5,

      //////////// KEY //////////
      input                   [1:0]           KEY,

      //////////// LED //////////
      output                  [9:0]           LEDR,
      //////////// SW //////////
      input                  [9:0]        SW
);

//=======================================================
//  REG/WIRE declarations
//=======================================================

wire [23:0]counterstate;
wire go;

assign go = 1'b1;



//=======================================================
//  Structural coding
//=======================================================

pulse_generator pulse(LEDR, KEY[0], SW[0], SW[9:4], MAX10_CLK1_50);
main_counter(counterstate, KEY[0], SW[0], LEDR[0], SW[1], SW[2],
MAX10_CLK1_50);

// display counter
seg7 H0(counterstate[3:0], HEX0);
seg7 H1(counterstate[7:4], HEX1);
seg7 H2(counterstate[11:8], HEX2);
seg7 H3(counterstate[15:12], HEX3);
seg7 H4(counterstate[19:16], HEX4);
seg7 H5(counterstate[23:20], HEX5);

endmodule
```

Figure 1.1.4.1 Preliminary Verilog Code.

Part 1.2 Calculations

Calculate how many seconds are required for counter2 to count these four cases:

1) one least-significant digit increment, and

f = 50Mhz

divideby = $(111111)_b$= 63

f' = f/divideby = 50 Mhz / 63 = 793.650Khz

T = 1/f' = 1/793.650Khz = 0.00000126 s

For divideby

2) through all of its possible values when:

a) divideby = 000001, and

f = 50Mhz

divideby = $(000001)_b$ = 1

f' = f/divideby = 50 Mhz / 1 = 50 Mhz

T = (1/f') x 2^{24}= (1/50 Mhz) x 2^{24} = 0.33554432s

b) divideby = 110010

f = 50Mhz

divideby = $(110010)_b$ = 50

f' = f/divideby = 50 Mhz / 50 = 1 Mhz

T = (1/f') x 2^{24}= (1/1 Mhz) x 2^{24} =16.777216s

2 Diagrams

Part 2.1 Counter 1: Pulse Generator

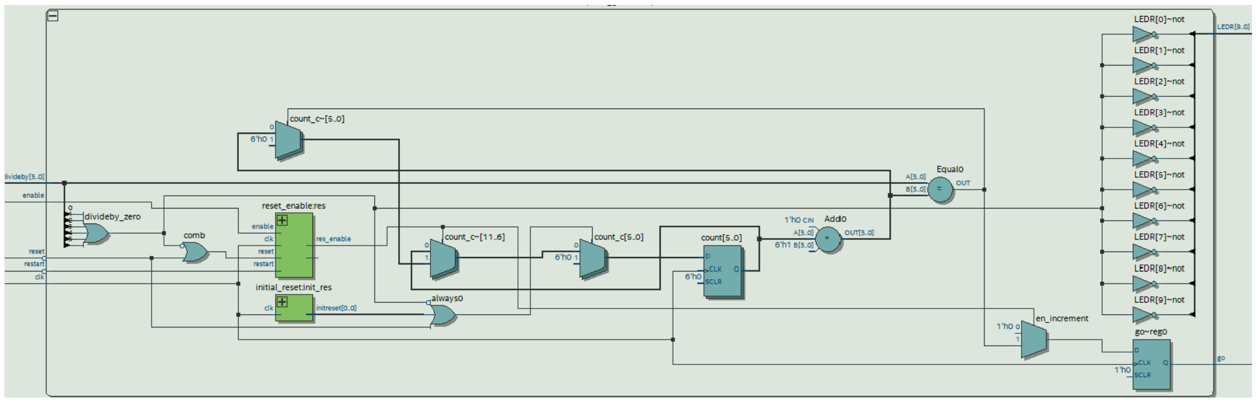


Figure 2.1.1 Pulse generator

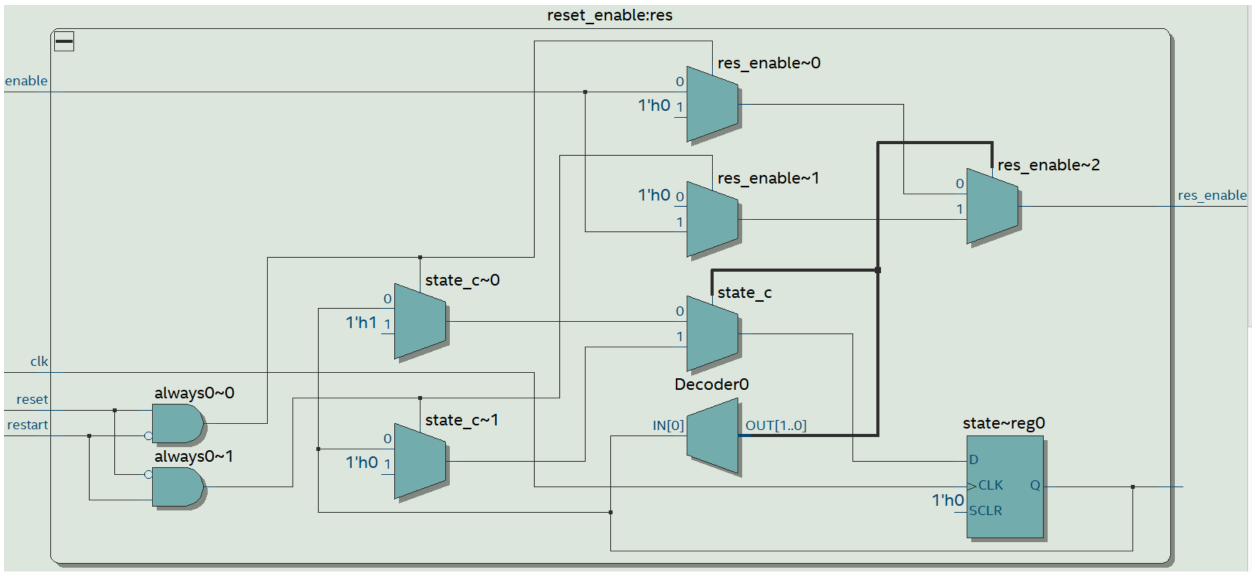


Figure 2.1.2 Reset - Enable subcircuit for Halt function

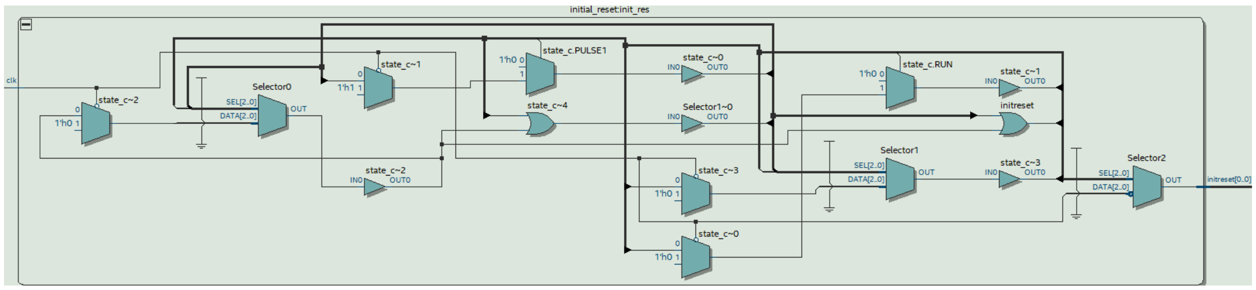


Figure 2.1.1.3 Circuit to generate a pulse at startup for initial reset

Part 2.2 Counter2: Main Counter

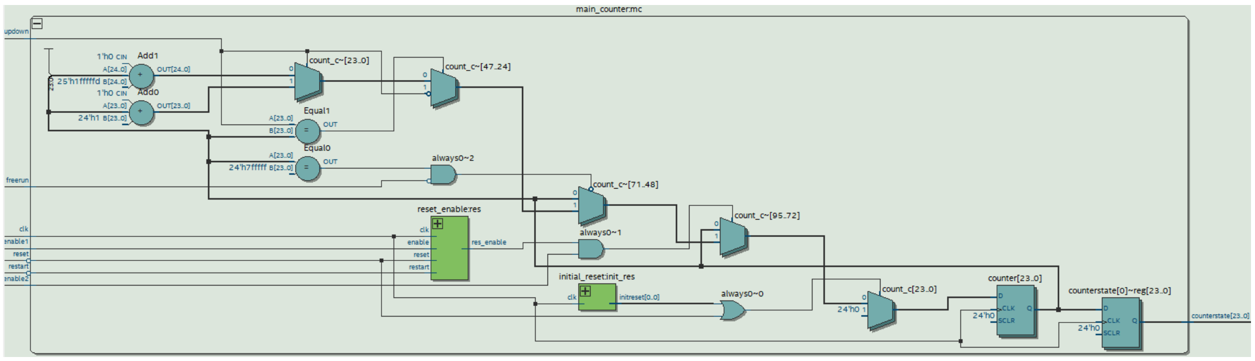


Figure 2.2.1 Main Counter Circuit

Part 2.3 Main Circuit

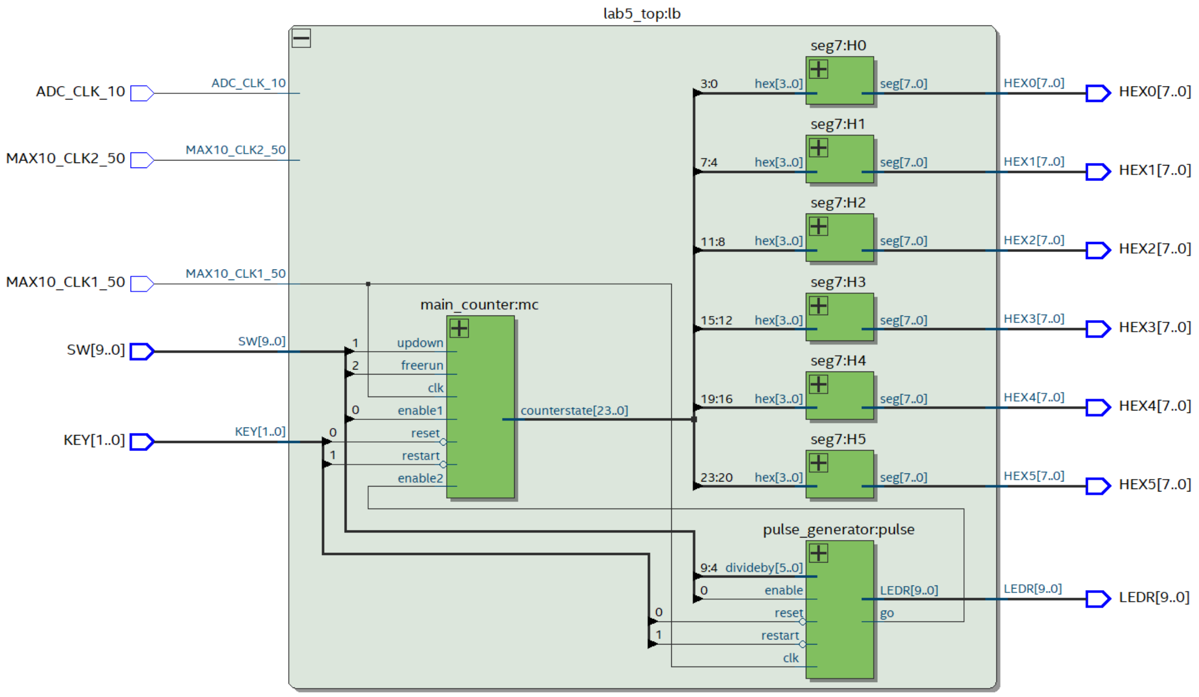


Figure 2.3.1 Main Circuit

3 Verilog Required Hardware

Part 3.1 Top Module for board implementation

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module hw2_top(
//////////// CLOCK //////////
input                                  ADC_CLK_10,
input                                  MAX10_CLK1_50,
input                                  MAX10_CLK2_50,

//////////// SEG7 //////////
output                 [7:0]           HEX0,
output                 [7:0]           HEX1,
output                 [7:0]           HEX2,
output                 [7:0]           HEX3,
output                 [7:0]           HEX4,
output                 [7:0]           HEX5,

//////////// KEY //////////
input                  [1:0]           KEY,

//////////// LED //////////
output                 [9:0]           LEDR,

//////////// SW //////////
input                  [9:0]           SW
);



lab5_top lb(

//////////// CLOCK //////////
ADC_CLK_10,
MAX10_CLK1_50,
MAX10_CLK2_50,

//////////// SEG7 //////////
HEX0,
HEX1,
HEX2,
HEX3,
HEX4,
HEX5,

//////////// KEY //////////
KEY,

//////////// LED //////////
LEDR,

//////////// SW //////////
SW
```

```
);

endmodule
```

Figure 3.1.1 Top Module Code

Part 3.2 Top Module for Lab Implementation

```
module lab5_top(

//////////// CLOCK //////////
input                               ADC_CLK_10,
input                               MAX10_CLK1_50,
input                               MAX10_CLK2_50,

//////////// SEG7 //////////
output                  [7:0]       HEX0,
output                  [7:0]       HEX1,
output                  [7:0]       HEX2,
output                  [7:0]       HEX3,
output                  [7:0]       HEX4,
output                  [7:0]       HEX5,

//////////// KEY //////////
input                   [1:0]       KEY,

//////////// LED //////////
output                  [9:0]       LEDR,

//////////// SW //////////
input                   [9:0]       SW
);



//=======================================================
//  REG/WIRE declarations
//=======================================================

wire [23:0]counterstate;
wire go;




//=======================================================
//  Structural coding
//=======================================================
//reset_enable res(KEY[0], SW[0], MAX10_CLK1_50, res_enable, state);

pulse_generator pulse(LEDR, go, ~KEY[0], ~KEY[1], SW[0], SW[9:4],
MAX10_CLK1_50);
main_counter mc(counterstate, ~KEY[0], ~KEY[1], SW[0], go, SW[1], SW[2],
MAX10_CLK1_50);

// display counter
seg7 H0(counterstate[3:0], HEX0);
seg7 H1(counterstate[7:4], HEX1);
seg7 H2(counterstate[11:8], HEX2);
seg7 H3(counterstate[15:12], HEX3);
seg7 H4(counterstate[19:16], HEX4);
seg7 H5(counterstate[23:20], HEX5);

endmodule
```

Figure 3.2.1 Top Module for Lab Implementation Code

Part 3.3 Initial Reset Circuit

```
module initial_reset(input clk, output reg [0:0]initreset);

parameter [1:0]RUN = 2'b00;
parameter [1:0]HALT = 2'b11;
parameter [1:0]PULSE1 = 2'b01;
parameter [1:0]PULSE2 = 2'b10;

reg [1:0]state_c;
      reg[1:0] state;

reg reset;


always @(*) begin
      // default same state
      state_c = state;

      reset = ~clk;

      case (state)
            RUN: begin
                  if (reset == 1'b1) begin
                        state_c = PULSE1;
                        initreset = 1'b1;
                  end
                  else begin
                        initreset = 1'b0;
                  end
            end
            PULSE1: begin
                  state_c = PULSE2;
                  initreset = 1'b1;
            end
            PULSE2: begin
                  state_c = HALT;
                  initreset = 1'b1;
            end
            HALT: begin
                  state_c = HALT;
                  initreset = 1'b0;
            end
            default: begin
                  state_c = RUN;
                  initreset = 1'b0;
            end
      endcase
end
always @(posedge clk or posedge initreset) begin
      state <= #1 state_c; // reset works even if disabled
end

endmodule
```

Figure 3.3.1 Initial Reset Circuit Code

Part 3.4 Halt Circuit

```
module reset_enable(input reset, restart, enable, clk, output reg res_enable,
output reg state);

parameter RUN = 1'b0;
parameter HALT = 1'b1;

reg state_c;


always @(*) begin
      // default same state
      state_c = state;

      case (state)
```

```
            RUN: begin
                  if (reset == 1'b1 && restart == 1'b0) begin
                        state_c = HALT;
                        res_enable = 1'b0;
                  end
                  else begin
                        res_enable = enable;
                  end
            end
            HALT: begin
                  if (restart == 1'b1 && reset == 1'b0) begin
                        state_c = RUN;
                        res_enable = enable;
                  end
                  else begin
                        res_enable = 1'b0;
                  end
            end
            default: begin
                  state_c = RUN;
                  res_enable = enable;
            end
      endcase
end
always @(posedge clk) begin
      state <= #1 state_c; // reset works even if disabled
end

endmodule
```

Figure 3.4.1 Halt Circuit Code

Part 3.5 Pulse Generator Circuit

```
module pulse_generator(LEDR, go, reset, restart, enable, divideby, clk);
// output and inputs
output reg [9:0]LEDR;
output reg go;
input reset;
input restart;
input enable;
input [5:0]divideby;
input clk;

reg [5:0] count, count_c;
reg en_increment; // next go value
reg divideby_zero;
wire init; // implemented
wire res_enable; // for halt on reset.
wire [1:0]res_state;

initial_reset init_res(clk, init);

reset_enable res(reset | divideby_zero, restart, enable, clk, res_enable,
res_state);

always @(*) begin
      // defaults
      divideby_zero = ~(divideby[5] | divideby[4] | divideby[3] | divideby[2]
| divideby[1] | divideby[0]);
      count_c = count;
      en_increment = 1'b0; // a combinational logic signal

      if (res_enable == 1'b1) begin   // halt on reset and on enable false
            count_c = count + 6'b000001; // "count" is a flip-flop register
            if (count_c == divideby) begin
                  count_c= 6'b000000; // wrap counter back to zero
                  en_increment= 1'b1; // pulse FF enable signal high
            end
      end
      // reset logic (place last to override other logic)
      if (reset == 1'b1 || divideby_zero == 1'b1 || init == 1'b1) begin
            count_c = 6'b00000;
      end
      // set LEDS
      LEDR[0] = divideby_zero;
      LEDR[1] = divideby_zero;
      LEDR[2] = divideby_zero;
      LEDR[3] = divideby_zero;
      LEDR[4] = divideby_zero;
      LEDR[5] = divideby_zero;
      LEDR[6] = divideby_zero;
      LEDR[7] = divideby_zero;
      LEDR[8] = divideby_zero;
      LEDR[9] = divideby_zero;

end
always @(posedge clk) begin
      count <= #1 count_c; // reset works even if disabled
      go <= #5 en_increment;
end
endmodule
```

Figure 3.5.1 Pulse Generator Circuit

Part 3.6 Main Counter Circuit

```
module main_counter(counterstate, reset, restart, enable1, enable2, updown,
freerun, clk);
// output and inputs
output reg [23:0]counterstate;
input reset;
input restart;
input enable1;
input enable2;
input updown;
input freerun;
input clk;


parameter zero = 24'b000000000000000000000000;
parameter one = 24'b000000000000000000000001;

parameter max = 24'b111111111111111111111111;
parameter halfmax = 24'b011111111111111111111111;

reg state_c;
wire init;

reg [23:0]count_c;
reg [23:0]counter;

wire res_enable; // for halt on reset.
wire [1:0]res_state;

initial_reset init_res(clk, init);

reset_enable res(reset, restart, enable1, clk, res_enable, res_state);

always @(*) begin
      // reset logic (place last to override other logic)
      if (reset == 1'b1 || init == 1'b1) begin
            count_c = zero;
      end
      else begin
            // defaults
            count_c = counter; // stay in the same spot

            if (res_enable == 1'b1 && enable2 == 1'b1) begin
                  if (~(freerun == 0 && counter == halfmax)) begin // halt at
halfmax if freerun is zero, restat will get it going again
                        count_c = (updown?(counter + one):(counter - one));
// "count" is a flip-flop register
```

```
                        if (counter == (updown?max:zero)) begin
                              count_c = (updown?zero:max); // wrap counter
back to starting point
                        end
                  end
            end
      end
end
always @(posedge clk) begin
      counter <= #1 count_c;
      counterstate <= #2 counter;
end

endmodule
```

Figure 3.6.1 Main Counter Circuit Code

4 Verilog Testbench

Part 4.1 Test Bench for Initial Reset

```
//`timescale 1ps/1ps
module tb_init_res();

reg clk;

wire res_init;


initial_reset ires(clk, res_init);


// Run clock
initial begin
clk=1'b0;

repeat (400) begin
      #10 clk = ~clk;
end
end

endmodule
```

Figure 4.1.1 Test Bench for Initial Reset Verilog Code

Part 4.2 Test Bench for Halt Circuit

```
//`timescale 1ps/1ps
module tb_reset_enable();

reg clk;

reg [9:0] SW;
reg reset;
reg restart;
wire enable;
reg divideby_zero;
wire res_state;
reg res_zero;


// Run clock
initial begin
clk=1'b0;
reset = 1'b0;
restart = 1'b0;
SW = 10'b0000111111;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
            repeat (400) begin
      #10 clk = ~clk;
end
end

reset_enable res(res_zero, restart, SW[0], clk, enable, res_state);

// change switches and Load
initial begin
SW = 10'b0000111111;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250;
SW = 10'b0001011111;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250;
SW = 10'b0001011110;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250;
```

```
SW = 10'b0001011111;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
# 250
SW = 10'b0000000000;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250;
SW = 10'b0001011001;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250
reset = 1'b1;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250
reset = 1'b0;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250
reset = 1'b0;
restart = 1'b1;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
#250
reset = 1'b0;
restart = 1'b0;
divideby_zero = ~(SW[5] | SW[4] | SW[3] | SW[2] | SW[1] | SW[0]);
res_zero = reset | divideby_zero;
end
endmodule
```

Figure 4.2.1 Test Bench for Halt Circuit Verilog Code

Part 4.3 Test Bench for Pulse Generator Circuit

```
//`timescale 1ps/1ps
module tb_pulse_generator();

reg clk;
wire [9:0] LEDR;
reg [9:0] SW;
reg reset;
reg restart;
wire go;


pulse_generator pgen(LEDR, go, reset, restart, SW[0], SW[9:4], clk);
```

```
// Run clock
initial begin
clk=1'b0;
reset = 1'b0;
restart = 1'b0;
            repeat (400) begin
      #10 clk = ~clk;
end
end

// change switches and Load
initial begin
SW = 10'b0000111111; // start at a divide by = 3
#250;
SW = 10'b0001011111; // jump to a divideby = 5
# 250
SW = 10'b0000000000; // divideby = 0, this will halt the circuit
#250
SW = 10'b0001011111;
reset = 1'b0;
restart = 1'b1;   // restart counting
#250
SW = 10'b0001011111;
reset = 1'b0;
restart = 1'b0;
#250;
SW = 10'b0001011110; // disable the circuit
#250;
SW = 10'b0001011111; // re-enable the circuit
#250;
SW = 10'b0001011001; // switch to count downwards
#250
reset = 1'b1;  // reset the circuit
#250
reset = 1'b0;
#250
reset = 1'b0;
restart = 1'b1;  // restart the circuit again
#250
reset = 1'b0;
restart = 1'b0;

end
endmodule
```

Figure 4.3.1 Test Bench for Pulse Generator Circuit Verilog Code

Part 4.4 Testbench for Main Counter Circuit

```
//`timescale 1ps/1ps
module tb_main_counter();

reg clk;
wire [9:0] LEDR;
reg [9:0] SW;
reg reset;
reg restart;
wire go;

wire [23:0]counterstate;



pulse_generator pgen(LEDR, go, reset, restart, SW[0], SW[9:4], clk);

main_counter mc(counterstate, reset, restart, SW[0], go,  SW[1], SW[2], clk);


// Run clock
initial begin
clk=1'b0;
reset = 1'b0;
restart = 1'b0;
            repeat (400) begin
      #10 clk = ~clk;
end
end

// change switches and Load
initial begin
SW = 10'b0000111111; // start at a divide by = 3
#250;
SW = 10'b0001011111; // jump to a divideby = 5
# 250
SW = 10'b0000000000; // divideby = 0, this will halt the circuit
#250
SW = 10'b0001011111;
reset = 1'b0;
restart = 1'b1;   // restart counting
#250
SW = 10'b0001011111;
reset = 1'b0;
restart = 1'b0;
#250;
```

```
SW = 10'b0001011110; // disable the circuit
#250;
SW = 10'b0001011111; // re-enable the circuit
#250;
SW = 10'b0001011001; // switch to count downwards
#400
reset = 1'b1;  // reset the circuit
#250
reset = 1'b0;
#250
reset = 1'b0;
restart = 1'b1;  // restart the circuit again
#250
reset = 1'b0;
restart = 1'b0;

end
endmodule
```

Figure 4.4.1 Test Bench for Main Counter Circuit Verilog Code

Part 4.5 Testbench for Main Module Circuit

```
//`timescale 1ps/1ps
module tb_lab5_top();

reg clk;
wire [9:0] LEDR;
reg [9:0] SW;
reg [1:0] KEY;

wire             [7:0]        HEX0;
wire             [7:0]        HEX1;
wire             [7:0]        HEX2;
wire             [7:0]        HEX3;
wire             [7:0]        HEX4;
wire             [7:0]        HEX5;

lab5_top lb(

//////////// CLOCK //////////
clk,
clk,
clk,
//////////// SEG7 //////////
HEX0,
HEX1,
HEX2,
HEX3,
HEX4,
HEX5,

//////////// KEY //////////
KEY,

//////////// LED //////////
LEDR,

//////////// SW //////////
SW
);
// Run clock
initial begin
clk=1'b0;
KEY = 2'b11;
            repeat (400) begin
      #10 clk = ~clk;
end
end

// change switches and Load
initial begin
SW = 10'b0000111111; // start at a divide by = 3
#250;
SW = 10'b0001011111; // jump to a divideby = 5
# 250
SW = 10'b0000000000; // divideby = 0, this will halt the circuit
#250
SW = 10'b0001011111;
KEY = 2'b01;
//reset = 1'b0;
//restart = 1'b1; // restart counting
#250
SW = 10'b0001011111;
KEY = 2'b11;
//reset = 1'b0;
//restart = 1'b0;
#250;
SW = 10'b0001011110; // disable the circuit
#250;
SW = 10'b0001011111; // re-enable the circuit
```

```
#250;
SW = 10'b0001011001; // switch to count downwards
#400
KEY = 2'b10;
//reset = 1'b1;  // reset the circuit
#250
KEY = 2'b11;
//reset = 1'b0;
#250
KEY = 2'b01;
//reset = 1'b0;
//restart = 1'b1;  // restart the circuit again
#250
KEY = 2'b11;
//reset = 1'b0;
//restart = 1'b0;

end
endmodule
```

Figure 4.5.1 Test Bench for Main Module Verilog Code

Part 4.6 Testbench for Main Module Circuit Count to Half Max Value

```
/`timescale 1ps/1ps
module tb_lab5_top_half();


reg clk;
wire [9:0] LEDR;
reg [9:0] SW;
reg reset;
reg restart;
wire go;

wire [23:0]counterstate;


pulse_generator pgen(LEDR, go, reset, restart, SW[0], SW[9:4], clk);

main_counter mc(counterstate, reset, restart, SW[0], go,  SW[1], SW[2], clk);


// Run clock
initial begin
clk=1'b0;
reset = 1'b0;
```

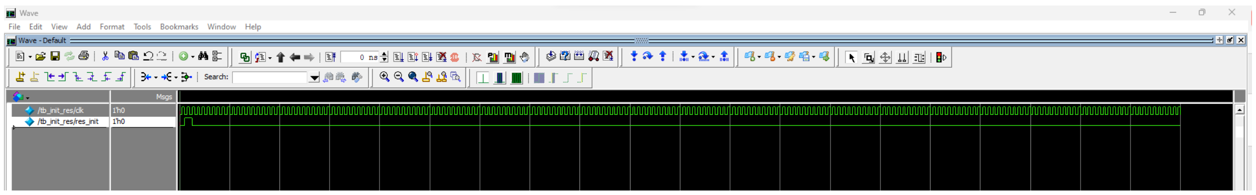
```
restart = 1'b0;
            repeat (33554440) begin
      #10 clk = ~clk;
end
end

// change switches and Load
initial begin
SW = 10'b0000101011; // start at a divide by = 2, stop at half counting up
$display("in = %b  , out = %h", SW, counterstate);
#100
$display("in = %b  , out = %h", SW, counterstate);
#335544300;
$display("in = %b , out = %h", SW, counterstate);
end
endmodule
```

Figure 4.6.1 Testbench for Main Module Circuit Count to Half Max Value Verilog Code

5 Testbench Results

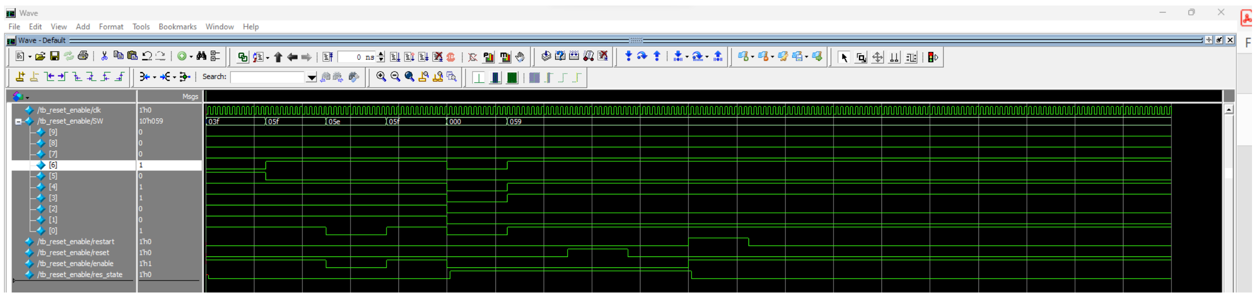
Part 5.1 Test Bench for Initial Reset Timing Diagram



Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_init_res/clk 1'h0
/tb_init_res/res_init 1'h0

Figure 5.1.1 Test Bench for Initial Reset Timing Diagram

Part 5.2 Test Bench for Halt Circuit Timing Diagram



Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_reset_enable/clk 1'h0
/tb_reset_enable/SW 10'h059
[9] 0
[8] 0
[7] 0
[6] 1
[5] 0
[4] 1
[3] 1
[2] 0
[1] 0
[0] 1
/tb_reset_enable/restart 1'h0
/tb_reset_enable/reset 1'h0
/tb_reset_enable/enable 1'h1
/tb_reset_enable/res_state 1'h0
03f
05f
05e
05f
000
059

Figure 5.2.1 Test Bench for Halt Reset Timing Diagram

Part 5.3 Test Bench for Pulse Generator Timing Diagram

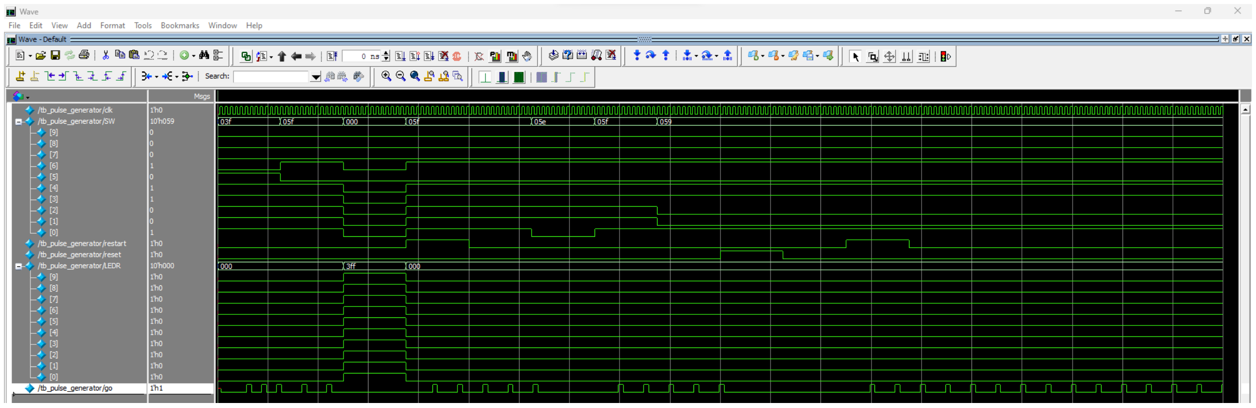


Figure 5.3.1 Test Bench for Pulse Generator Timing Diagram

Part 5.4 Test Bench for Main Counter Timing Diagram

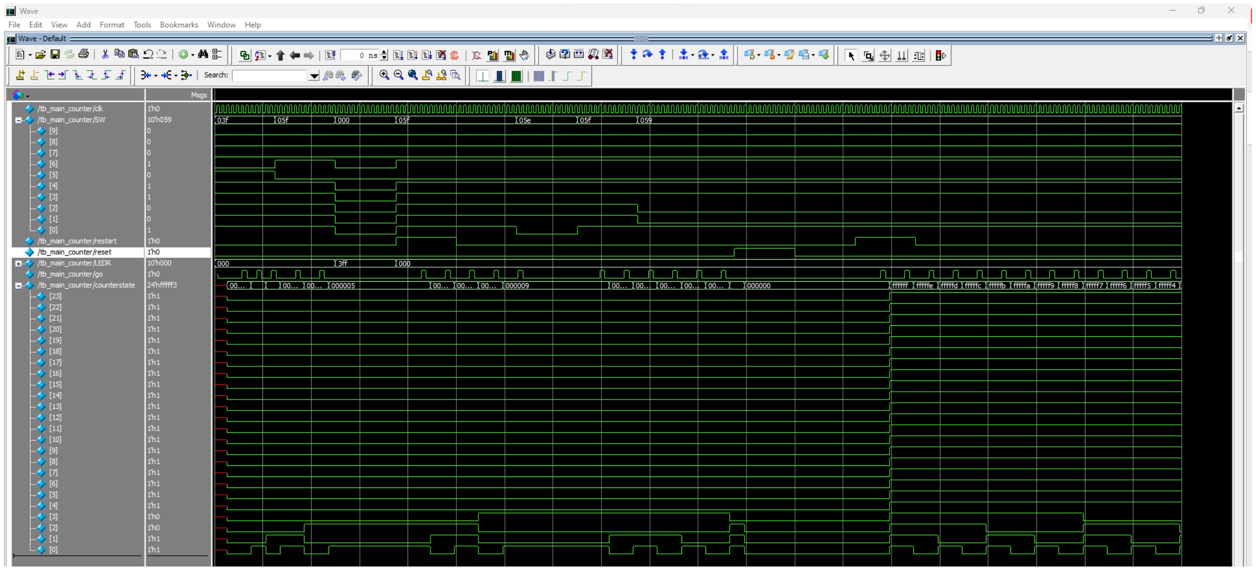


Figure 5.4.1 Test Bench for Main Counter Timing Diagram

Part 5.5 Test Bench for Top Module Including 7 Segment Display Drivers Timing Diagram

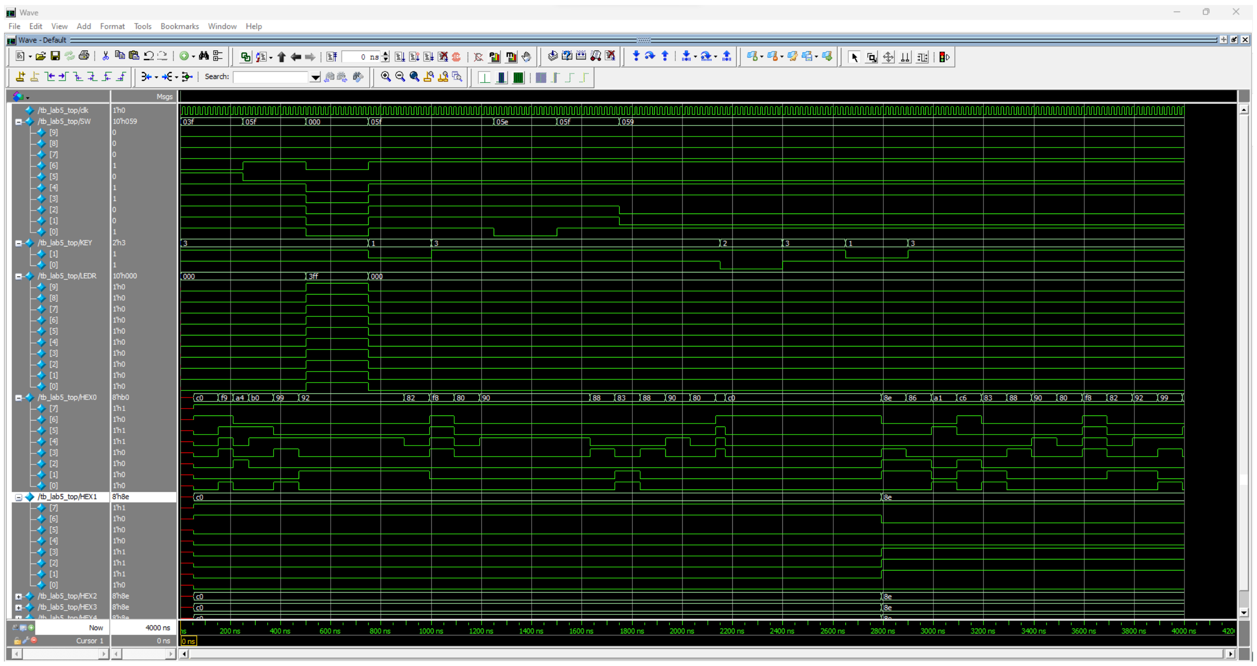


Figure 5.5.1 Test Bench for Top Module Including 7 Segment Display Drivers Timing Diagram

Part 5.6 Test Bench for Top Module Counting to Half Max Value Results

```
# in = 0000101011  , out = xxxxxx
# in = 0000101011  , out = 000000
# in = 02b , out = 7fffff
```

Figure 5.5.1 Test Bench for Top Module Counting to Half Max Value Results

Discussion

In the design, we added an input signal called *restart* so that we could restart the circuit after a reset. We needed to create a circuit to issue one pulse on system startup to do a reset on the circuit to put it in a known state, as we were getting lines in an unknown state without it (the 24-bit counters for example) despite having default values configured. We were able to test all the modules we did on their functionality. The hardware implementation worked as expected a te end.

Conclusion

In this design, the test benches were crucial for debugging the design as they provided visibility of what was happening on the modules that other modules depended upon. Most of the test benches we could prove with a timing diagram except the one where we put it to count to half as that one took a lot of time to run. Testing the counter in the full range of numbers was faster when we implemented the Verilog Code in the board.