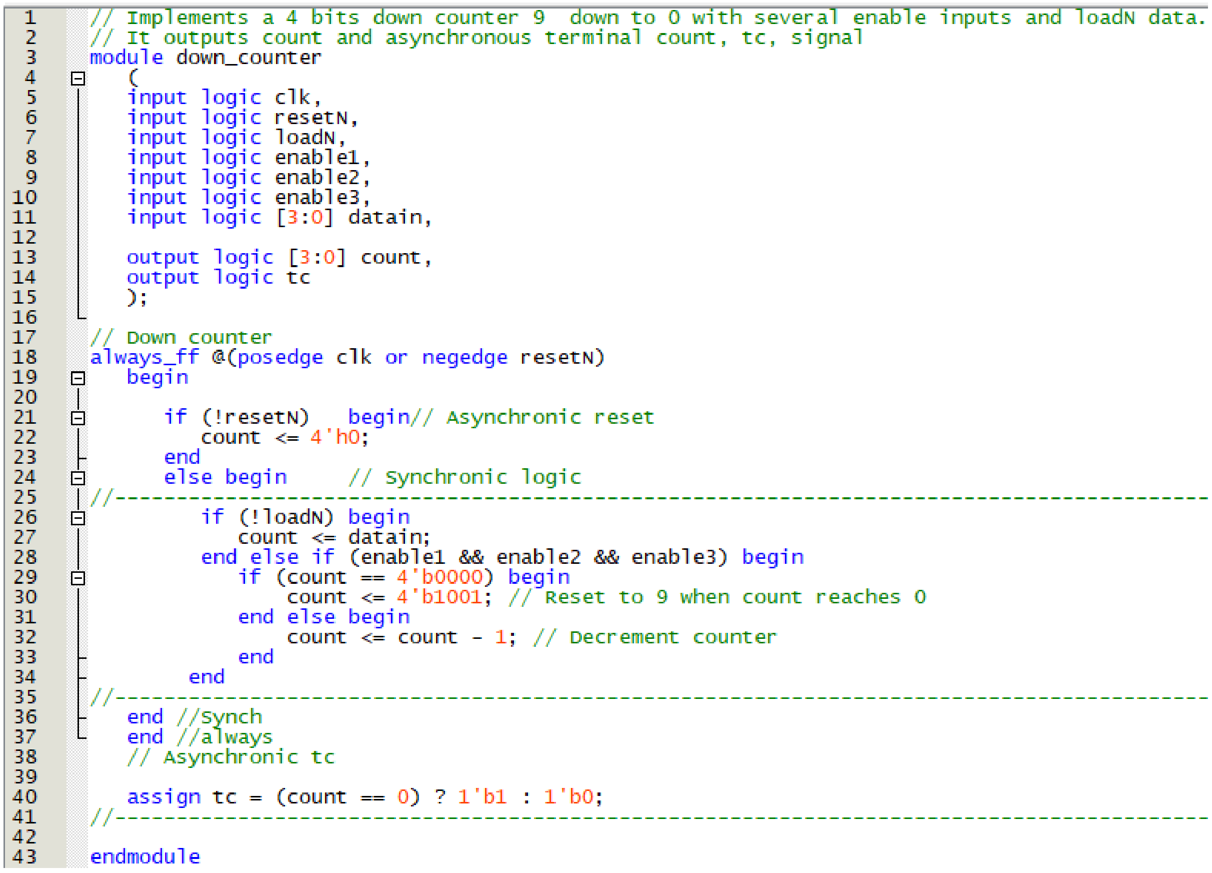
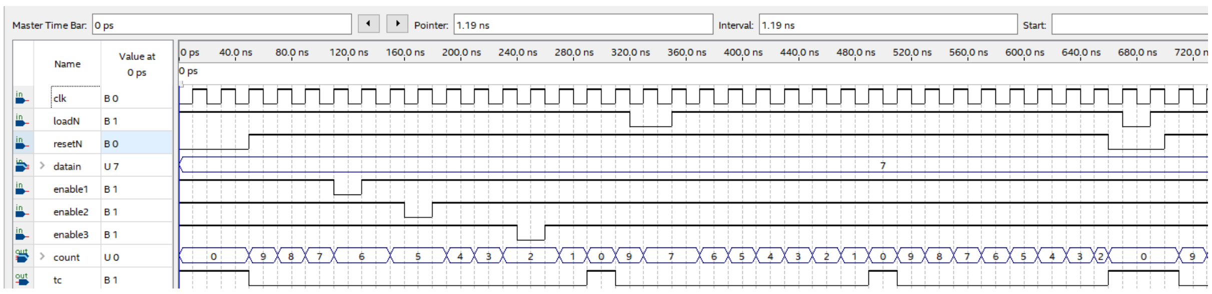
Description: In this project, I will build a bomb using a state machine, a down counter, and lights on the fpga.

First we need to create a down counter:

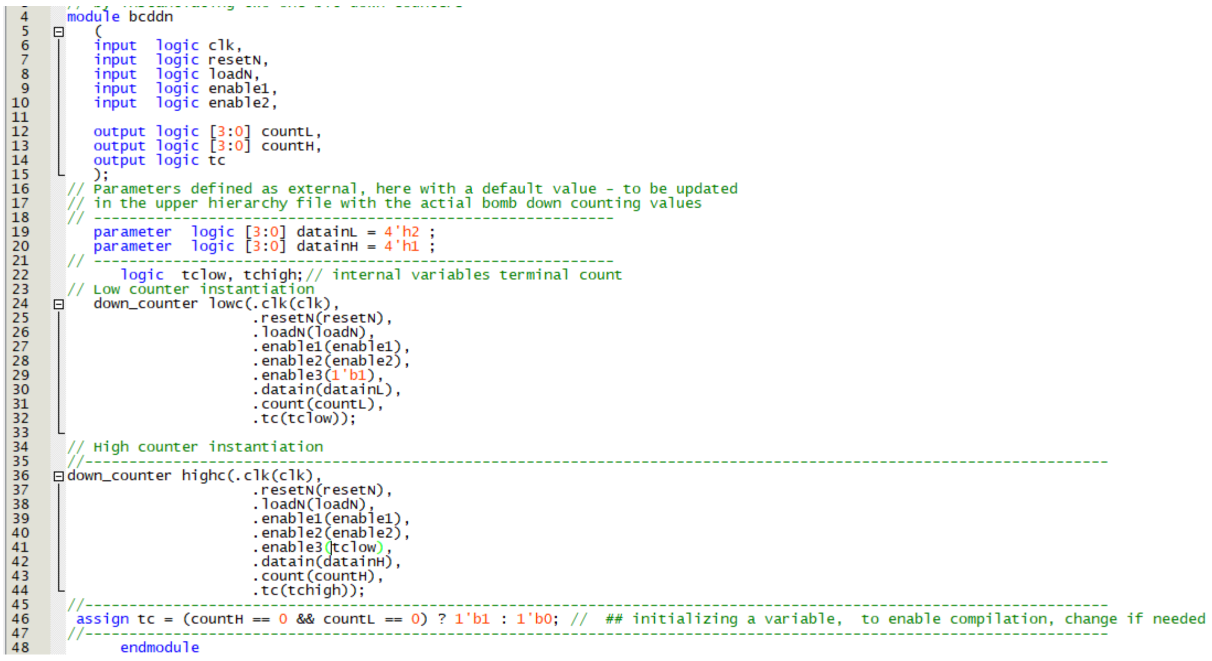


```
// Implements a 4 bits down counter 9  down to 0 with several enable inputs and loadN data.
// It outputs count and asynchronous terminal count, tc, signal
module down_counter
   (
   input logic clk,
   input logic resetN,
   input logic loadN,
   input logic enable1,
   input logic enable2,
   input logic enable3,
   input logic [3:0] datain,

   output logic [3:0] count,
   output logic tc
   );

// Down counter
always_ff @(posedge clk or negedge resetN)
   begin

      if (!resetN)    begin// Asynchronic reset
         count <= 4'h0;
      end
      else begin     // Synchronic logic
//----------------------------------------------------------------------------------------
         if (!loadN) begin
            count <= datain;
         end else if (enable1 && enable2 && enable3) begin
            if (count == 4'b0000) begin
               count <= 4'b1001; // Reset to 9 when count reaches 0
            end else begin
               count <= count - 1; // Decrement counter
            end
         end
//----------------------------------------------------------------------------------------
   end //synch
   end //always
   // Asynchronic tc

   assign tc = (count == 0) ? 1'b1 : 1'b0;
//----------------------------------------------------------------------------------------

endmodule
```

Check with waveform:

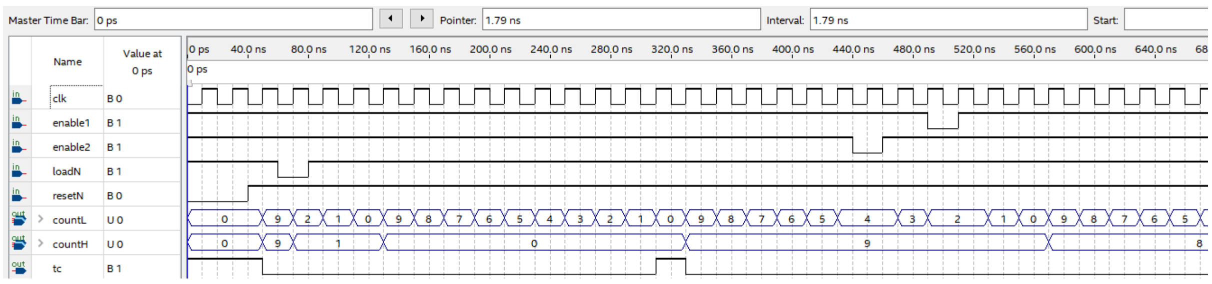


Next step is to design a down counter with 2 digits:

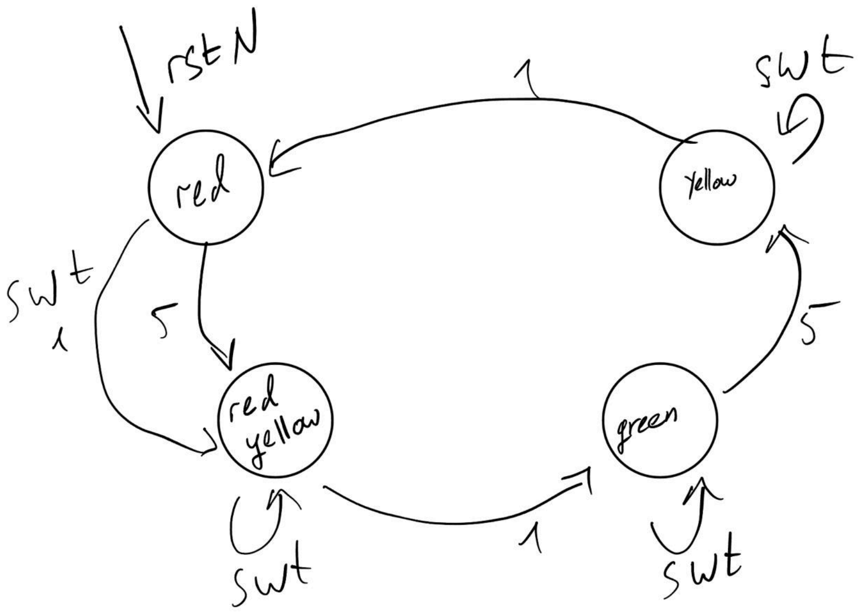


```
module bcddn
    (
    input  logic clk,
    input  logic resetN,
    input  logic loadN,
    input  logic enable1,
    input  logic enable2,

    output logic [3:0] countL,
    output logic [3:0] countH,
    output logic tc
    );
// Parameters defined as external, here with a default value - to be updated
// in the upper hierarchy file with the actial bomb down counting values
// -----------------------------------------------------------
    parameter  logic [3:0] datainL = 4'h2 ;
    parameter  logic [3:0] datainH = 4'h1 ;
// -----------------------------------------------------------
      logic  tclow, tchigh;// internal variables terminal count
// Low counter instantiation
    down_counter lowc(.clk(clk),
                      .resetN(resetN),
                      .loadN(loadN),
                      .enable1(enable1),
                      .enable2(enable2),
                      .enable3(1'b1),
                      .datain(datainL),
                      .count(countL),
                      .tc(tclow));

// High counter instantiation
//------------------------------------------------------------------------------------------------------------
down_counter highc(.clk(clk),
                      .resetN(resetN),
                      .loadN(loadN),
                      .enable1(enable1),
                      .enable2(enable2),
                      .enable3(tclow),
                      .datain(datainH),
                      .count(countH),
                      .tc(tchigh));
//------------------------------------------------------------------------------------------------------------
 assign tc = (countH == 0 && countL == 0) ? 1'b1 : 1'b0; //  ## initializing a variable,  to enable compilation, change if needed
//------------------------------------------------------------------------------------------------------------
      endmodule
```

Check with waveform:



The next step is to create a traffic light, which will allow us to use the light on the bomb as a signal before the explosion.
The FSM:



The code:

```
module ramzor
        (
        input logic clk, // 50 MHz clock
        input logic resetN,
        input logic switchN, // pedestrian switch to change the light

        output logic redLight,    //  output to the red lamp
        output logic yellowLight, //  output to the yellow lamp
        output logic greenLight           //  output to the green lamp '
  );



//--------------------------------------------------------
 // localparam SIMULATION_MODE_ON = 0 ;  //  Run Mode : unmask the line for real run
   localparam SIMULATION_MODE_ON = 1 ; //  simulation mode : unmask the line for
simulation run
//--------------------------------------------------------
// state machine and parameters declaration
        enum logic [1:0] {s_red, s_red_yellow, s_green, s_yellow} SMramzor; // state
machine
        logic [3:0] counter; // count down second timer

  logic t_sec; // A short pulse, once every second.
        assign  timerEnded = (counter == 4'b0) ;

//---------------------------------------------------------------------------------------------------------

        localparam logic [3:0] RED_OR_GREEN_TIME = 'd5;
        localparam logic [3:0] YELLOW_TIME = 'd1;

        localparam logic LED_ON = 1'b1;
        localparam logic LED_OFF = 1'b0;


//-------------------------------------------------------------------------------------------
// Instance of 1Hz sub module as a time counter for the state machine

one_sec_counter #(.SIMULATION_MODE(SIMULATION_MODE_ON)) one_sec_counter
(.clk(clk),
                                                                .resetN(resetN),
                                                                .turbo(0),
                                                                .one_sec(t_sec) );
```

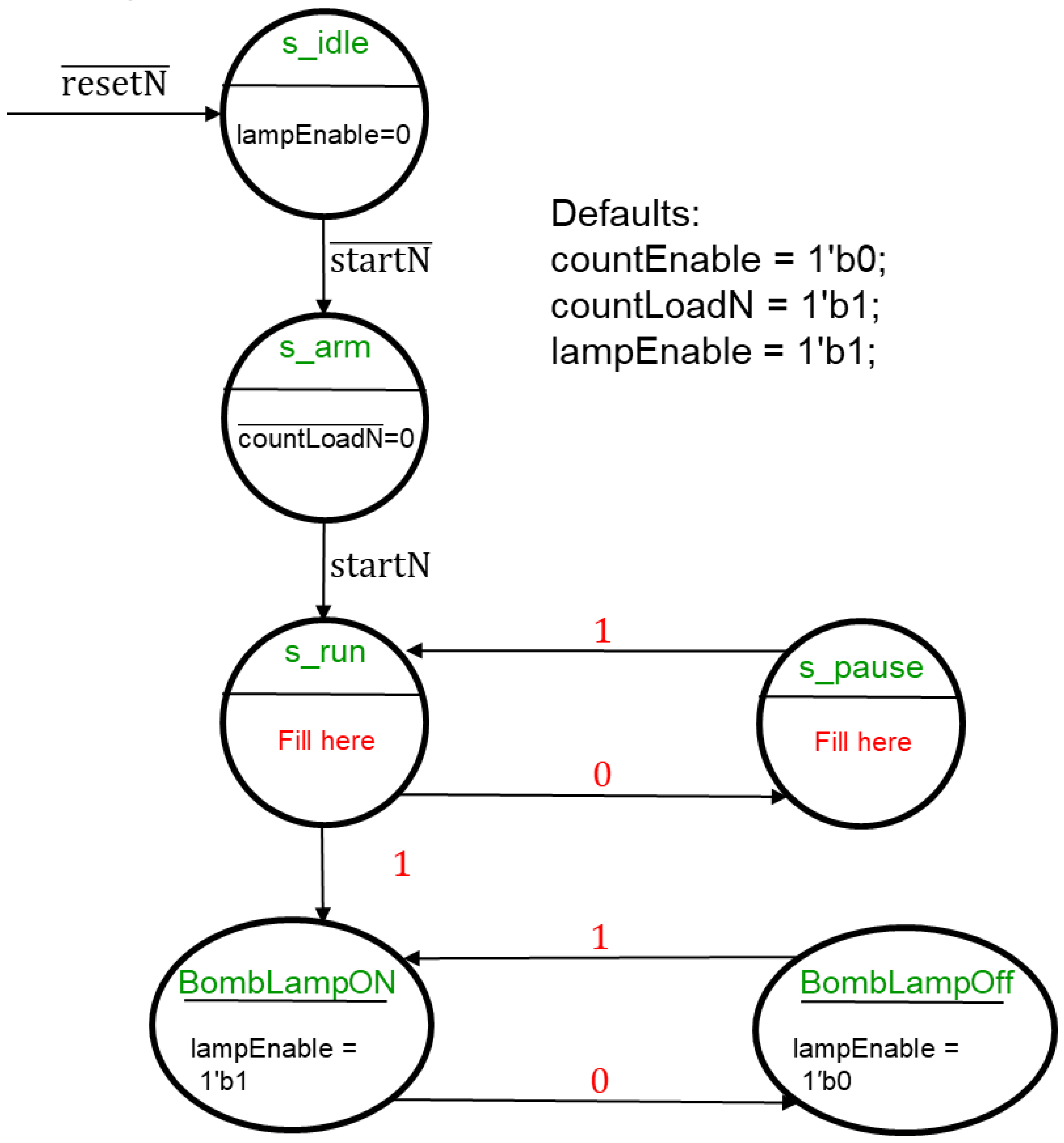
```
//---------------------------------------------------------------------------------------------------------------

 //hexss
//---------------------------------------------------------------------------------------------------------------

//   syncronous code,  executed once every clock to update the current state and outputs

always_ff @(posedge clk or negedge resetN) // State machine logic ////
   begin

   if ( !resetN ) begin // Asynchronic reset, initialize the state machine
                        SMramzor <= s_red;
                        counter  <= RED_OR_GREEN_TIME;
                        redLight  <= 1'b1 ;
                        yellowLight <=1'b0 ;
                        greenLight <= 1'b0 ;




            end // asynch
            else begin                                         // Synchronic logic of the state machine; once
every clock

//---------------------------------------------------------------------------------------------------------------
                        if ( t_sec ) begin  // perform once every timer pulse ( every 0.1 sec )

                                    if ( !timerEnded )  // timer didn't finish
                                                counter <= counter - 4'b1; // decremnt the counter

                        end
//---------------------------------------------------------------------------------------------------------------
            // state machine

                        // default outputs
                        redLight  <= 1'b0 ;
                        yellowLight <=1'b0 ;
                        greenLight <= 1'b0 ;

                        case ( SMramzor )

                                                                //Note: the implementation of the red Yellow
state is already given you as an example

                        //          ===========
                                                                s_red_yellow: begin
                        //          ===========
                                                                            redLight  <= 1'b1 ;
```

```
                                                        yellowLight <=1'b1 ;

                                                        if ( timerEnded )  // timer finished
                                                                begin
                                                                        SMramzor <= s_green;
//next state is green
                                                                        counter <=
RED_OR_GREEN_TIME; // reload counter with the next value.
                                                                end // if

                                                end // s_red_yellow

//-----------------------------------------------------------------------------------------------------------------
            s_green: begin
                                      greenLight <= 1'b1;
                                                 if ( timerEnded )
                                                   begin
                                                           SMramzor <= s_yellow; // next state
is yellow
                                                                 counter <= YELLOW_TIME; //
the timer now is 1sec
                                                         end
                                                end
//-----------------------------------------------------------------------------------------------------------------
            s_yellow: begin
                                      yellowLight <= 1'b1;
                                                 if ( timerEnded )
                                                   begin
                                                           SMramzor <= s_red; // next state is
red
                                                                 counter <=
RED_OR_GREEN_TIME; // the timer now is 5sec
                                                         end
                                                end
//-----------------------------------------------------------------------------------------------------------------
            s_red: begin
                                      redLight <= 1'b1;
                                                 if (!switchN) begin
                                                   SMramzor <= s_red_yellow; // next state is
yellow
                                                         counter <= YELLOW_TIME; // the timer
now is 1sec
                                                 end
                                                 else if (timerEnded)
                                                   begin
                                                           SMramzor <= s_red_yellow; // next
state is yellow
```

```
                                                                          counter <= YELLOW_TIME; //
the timer now is 1sec
                                                                 end
                                                       end
//-------------------------------------------------------------------------------------------------------------------


                  //                    =========
                                               default : begin
                  //          =======
                                                                        SMramzor <= s_red;  //next state
                                                       end // default

         endcase
         end // if reset



end // always_ff state machine ////////////////////////////////



endmodule
```

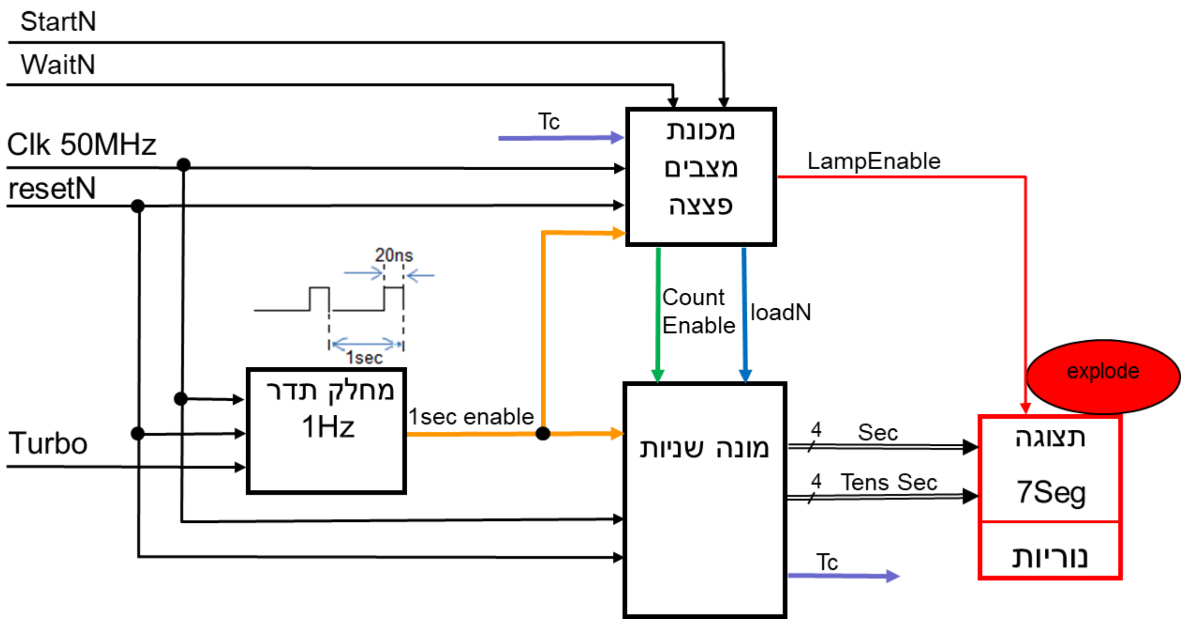
// end code here

Bomb Diagram State:



Defaults:
countEnable = 1'b0;
countLoadN = 1'b1;
lampEnable = 1'b1;

Block Diagram:



Bomb Verilog Code:

```
// Implements the state machine of the bomb mini-project

module bomb
          (
          input logic clk,
          input logic resetN,
          input logic startN,
          input logic waitN,
          input logic OneSecPulse,
          input logic timerEnd,

          output logic countLoadN,
          output logic countEnable,
          output logic lampEnable
  );

//----------------------------------------------------------------------------------------

// state machine decleration
 enum logic [2:0] {s_idle, s_arm, s_run, s_pause, s_lampOff, s_lampOn} SMbomb;

//----------------------------------------------------------------------------------------
//  syncronous code:  executed once every clock to update the current state
always @(posedge clk or negedge resetN)
  begin

  if ( !resetN ) begin // Asynchronic reset
```

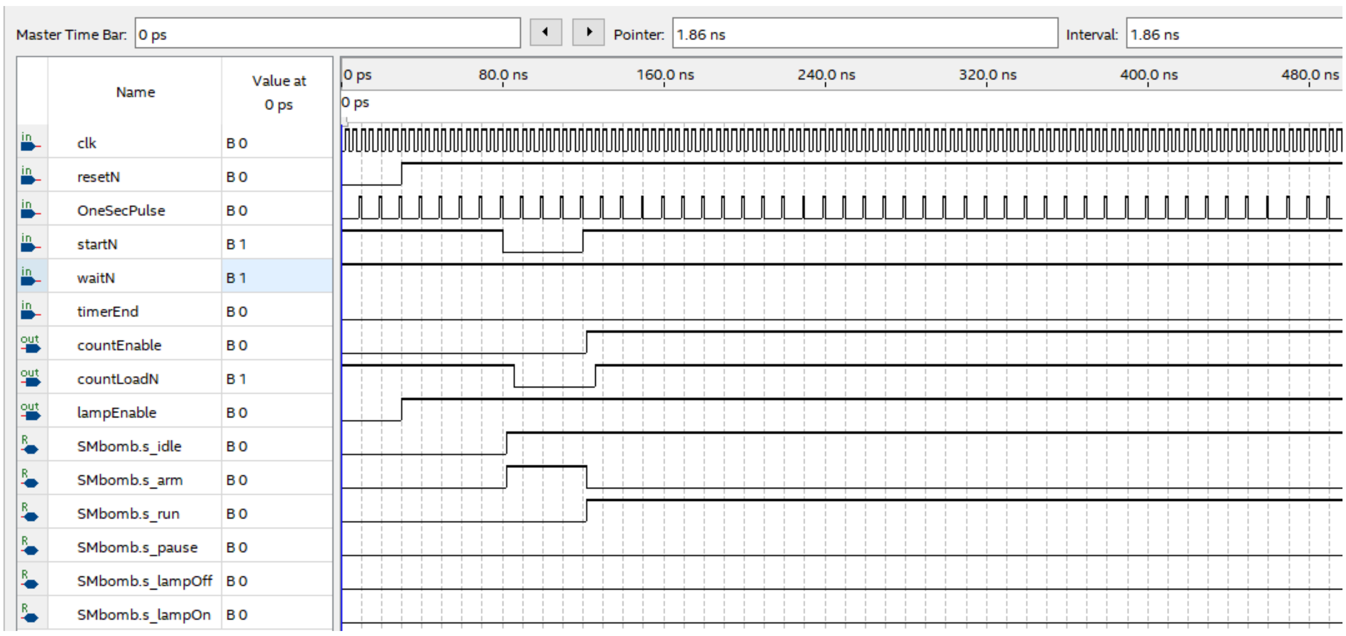
```
                SMbomb <= s_idle;
                countEnable <= 1'b0;
                countLoadN  <= 1'b1;
                lampEnable  <= 1'b0;

                end


        else            // Synchronic logic FSM
                begin
                // default outputs
                countEnable <= 1'b0;
                countLoadN  <= 1'b1;
                lampEnable  <= 1'b1;


                case (SMbomb) // logically defining what is the next state, and the ouptput

//      ======
                        s_idle: begin
//      ======
                                if (startN == 1'b0)
                                        SMbomb <= s_arm;
                        end // idle
//----------------------------------------------------------------------------------------------------------------
s_arm: begin
        countLoadN <= 1'b0; // Load countdown timer
        if (startN) begin // Wait for startN to be released
           SMbomb <= s_run;
           countEnable <= 1'b1; // Start countdown
        end else begin
           SMbomb <= s_arm;
        end
      end

      s_run: begin
        countEnable <= 1'b1; // Continue countdown
        if (!waitN) begin
           SMbomb <= s_pause;
           countEnable <= 1'b0; // Pause countdown
        end else if (timerEnd) begin
           SMbomb <= s_lampOff; // Timer ended
        end else begin
           SMbomb <= s_run;
        end
      end

      s_pause: begin
```

```
   countEnable <= 1'b0; // Countdown is paused
   if (waitN) begin // Wait for waitN to be released
      SMbomb <= s_run;
      countEnable <= 1'b1; // Resume countdown
   end else begin
      SMbomb <= s_pause;
   end
end

s_lampOff: begin
   lampEnable <= 1'b0; // Turn off lamp
   if (OneSecPulse) begin // Blink display
      SMbomb <= s_lampOn;
   end else begin
      SMbomb <= s_lampOff;
   end
end

s_lampOn: begin
   lampEnable <= 1'b1; // Turn on lamp
   if (OneSecPulse) begin // Continue blinking
      SMbomb <= s_lampOff;
   end else begin
      SMbomb <= s_lampOn;
   end
end
```

```
//--------------------------------------------------------------------------------------------------------------------
//                    =========
                               default : begin
//          =======
                                              SMbomb <= s_idle;
                              end // default

                    endcase
          end //else

          end // always sync

          //  outputs logic

endmodule
```

// end code here

Waveform Simulation:

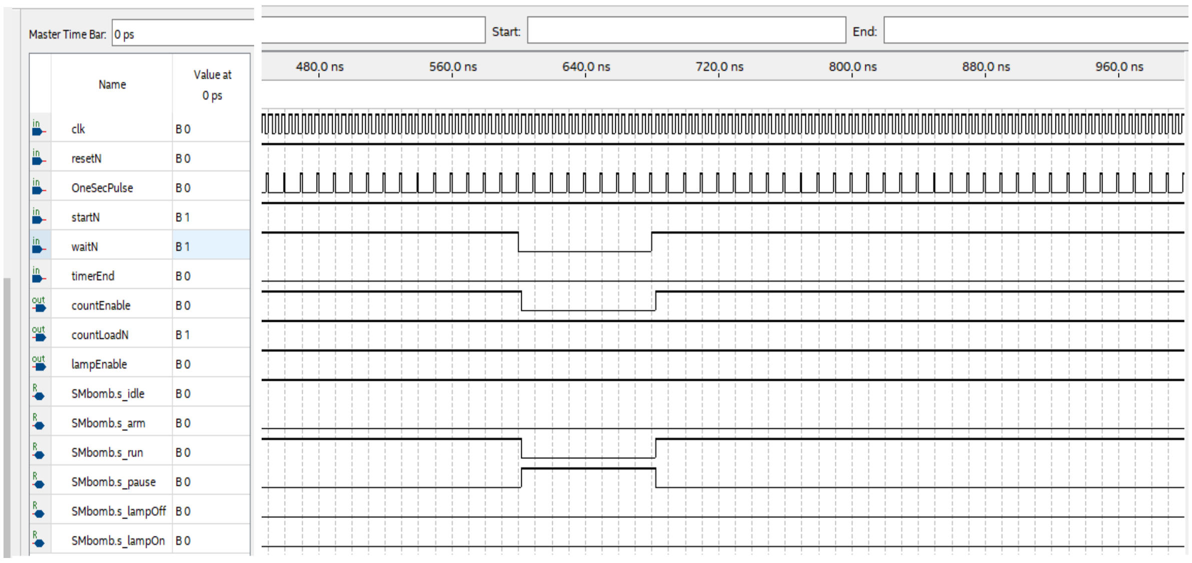


Master Time Bar: 0 ps

Pointer: 1.86 ns

Interval: 1.86 ns

Name	Value at 0 ps
clk	B 0
resetN	B 0
OneSecPulse	B 0
startN	B 1
waitN	B 1
timerEnd	B 0
countEnable	B 0
countLoadN	B 1
lampEnable	B 0
SMbomb.s_idle	B 0
SMbomb.s_arm	B 0
SMbomb.s_run	B 0
SMbomb.s_pause	B 0
SMbomb.s_lampOff	B 0
SMbomb.s_lampOn	B 0



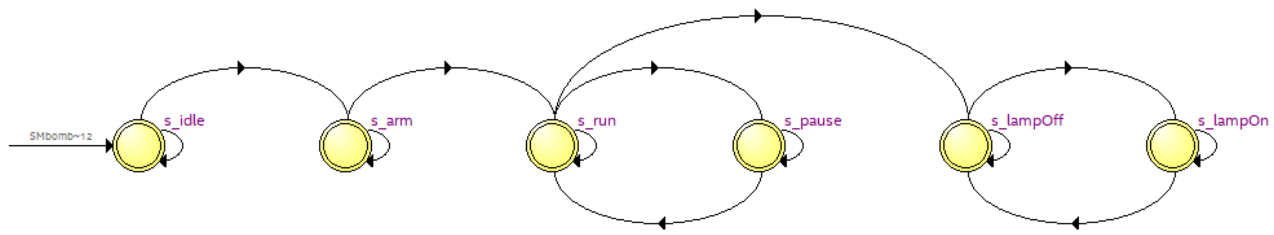
Master Time Bar: 0 ps

Start:

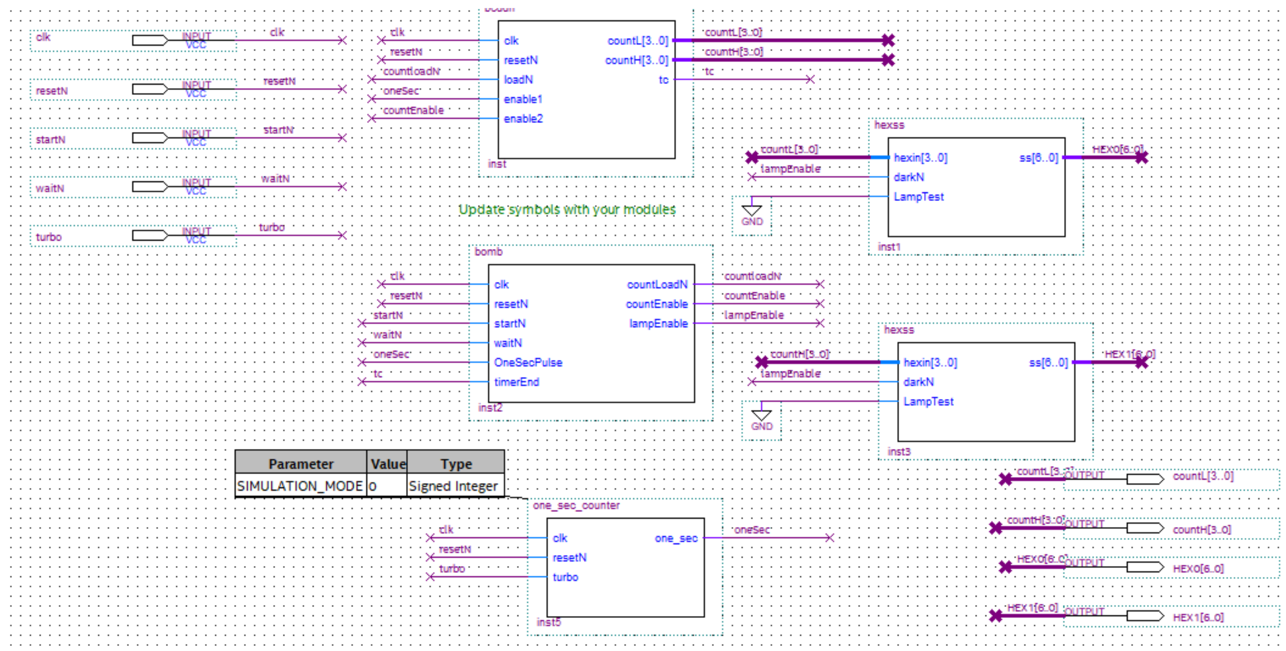
End:

Name	Value at 0 ps
clk	B 0
resetN	B 0
OneSecPulse	B 0
startN	B 1
waitN	B 1
timerEnd	B 0
countEnable	B 0
countLoadN	B 1
lampEnable	B 0
SMbomb.s_idle	B 0
SMbomb.s_arm	B 0
SMbomb.s_run	B 0
SMbomb.s_pause	B 0
SMbomb.s_lampOff	B 0
SMbomb.s_lampOn	B 0

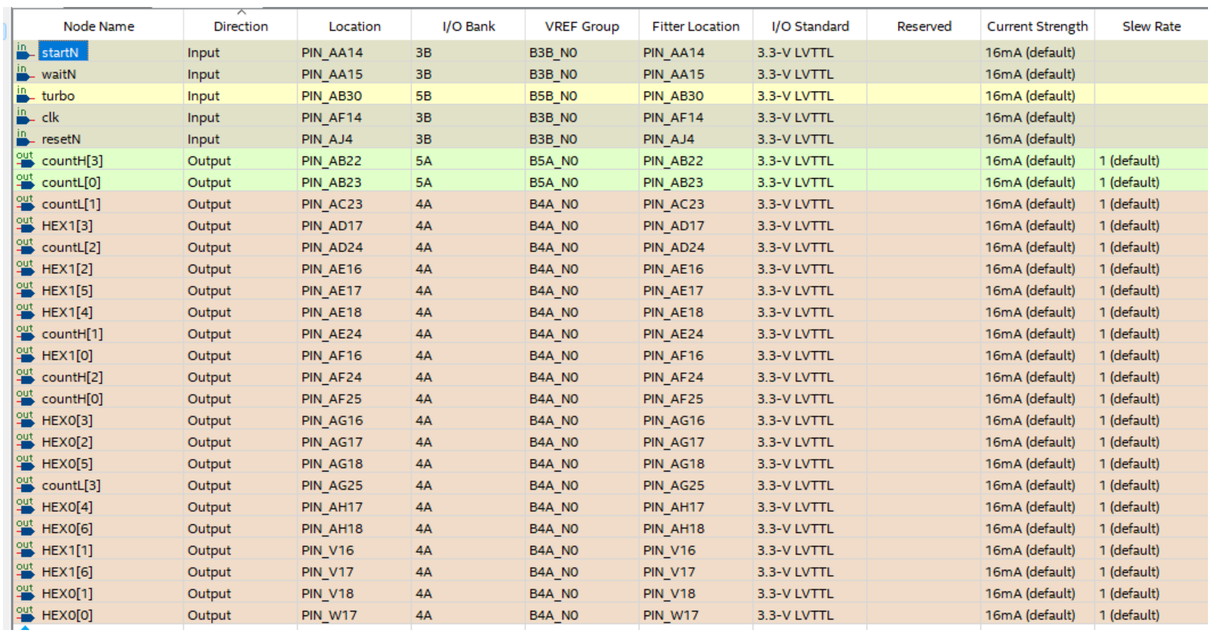
Quartus FSM For The Design:



Top Hierarchy For The Project:



Compilation on the FPGA:



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
startN	Input	PIN_AA14	3B	B3B_N0	PIN_AA14	3.3-V LVTTL		16mA (default)	
waitN	Input	PIN_AA15	3B	B3B_N0	PIN_AA15	3.3-V LVTTL		16mA (default)	
turbo	Input	PIN_AB30	5B	B5B_N0	PIN_AB30	3.3-V LVTTL		16mA (default)	
clk	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	3.3-V LVTTL		16mA (default)	
resetN	Input	PIN_AJ4	3B	B3B_N0	PIN_AJ4	3.3-V LVTTL		16mA (default)	
countH[3]	Output	PIN_AB22	5A	B5A_N0	PIN_AB22	3.3-V LVTTL		16mA (default)	1 (default)
countL[0]	Output	PIN_AB23	5A	B5A_N0	PIN_AB23	3.3-V LVTTL		16mA (default)	1 (default)
countL[1]	Output	PIN_AC23	4A	B4A_N0	PIN_AC23	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[3]	Output	PIN_AD17	4A	B4A_N0	PIN_AD17	3.3-V LVTTL		16mA (default)	1 (default)
countL[2]	Output	PIN_AD24	4A	B4A_N0	PIN_AD24	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[2]	Output	PIN_AE16	4A	B4A_N0	PIN_AE16	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[5]	Output	PIN_AE17	4A	B4A_N0	PIN_AE17	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[4]	Output	PIN_AE18	4A	B4A_N0	PIN_AE18	3.3-V LVTTL		16mA (default)	1 (default)
countH[1]	Output	PIN_AE24	4A	B4A_N0	PIN_AE24	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[0]	Output	PIN_AF16	4A	B4A_N0	PIN_AF16	3.3-V LVTTL		16mA (default)	1 (default)
countH[2]	Output	PIN_AF24	4A	B4A_N0	PIN_AF24	3.3-V LVTTL		16mA (default)	1 (default)
countH[0]	Output	PIN_AF25	4A	B4A_N0	PIN_AF25	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[3]	Output	PIN_AG16	4A	B4A_N0	PIN_AG16	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[2]	Output	PIN_AG17	4A	B4A_N0	PIN_AG17	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[5]	Output	PIN_AG18	4A	B4A_N0	PIN_AG18	3.3-V LVTTL		16mA (default)	1 (default)
countL[3]	Output	PIN_AG25	4A	B4A_N0	PIN_AG25	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[4]	Output	PIN_AH17	4A	B4A_N0	PIN_AH17	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[6]	Output	PIN_AH18	4A	B4A_N0	PIN_AH18	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[1]	Output	PIN_V16	4A	B4A_N0	PIN_V16	3.3-V LVTTL		16mA (default)	1 (default)
HEX1[6]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[1]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA (default)	1 (default)
HEX0[0]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA (default)	1 (default)