EEET2162 – Advanced Digital Design

Laboratory 4 – Finite State Machine Design

Conor Christensen – s3815282

Thursday 1430

Introduction

Finite State Machines (FSM) are powerful machines that as the name suggests contains a limited number of states. Theoretically, most all PCs and computing devices are all finite state machines as they can only exist within a finite set of states during operation. However, on the scale these complex processing machine runs this is not equivalent to the FSM investigated in this laboratory. FSMs are typically designed in two fashions, Mealy and Moore machines, both of which are to be considered in the FSM design in this laboratory. The key difference between the two FSM designs is the system output dependencies. In the case of a Moore machine the outputs are decided entirely by the current state of the FSM, whereas in a Mealy machine it is both the state and the input that determine the system outputs.

The investigation requires the design of an alarm system that takes input from an arming device and motion sensors, and outputs to a siren and strobe light. Given the DE10-Nano FPGA used, these inputs and outputs will be replaced respectively with switches and LEDs.

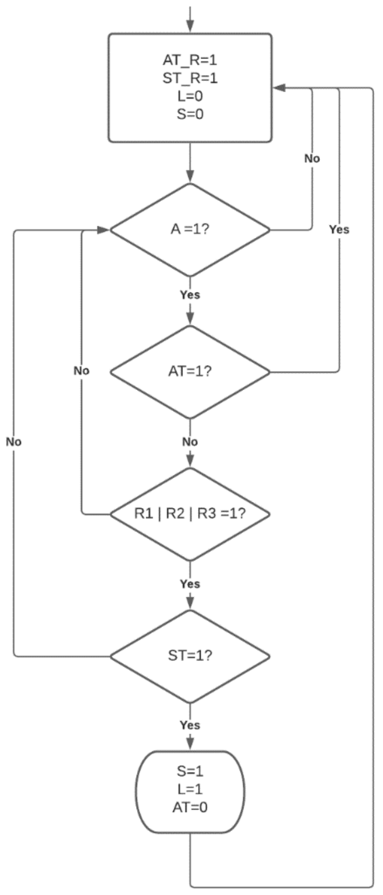
This investigation relies on time sensitive routines, operating in the range of 1Hz oscillators and 5-20 second timers. Digital devices typically run at speeds undetectable to humans, with the DE10-Nano board operating using a clock of 50MHz (a period of 20ns). This laboratory introduces the concept of counters and will require their implementation to operate the timing routines and effectively step-down the internal 50MHz clock.

It will be shown through the results and discussion that this investigation proved challenging and initially some concepts surrounding the FSM design were misunderstood. However, significant learning surrounding these concepts has been made although the timeframe of the assignment did not allow for a revised implementation of the alarm system.

Results and Discussion

The alarm system uses Input/Output (I/O) on the FPGA to control its behaviour. However, the system requirements outlined in the laboratory require the output behaviour to be dependant on temporal parameters, these being a five second alarm-arm delay and a twenty second fixed duration for the activated alarm. These additional timing routines create additional internal inputs and changing variables outside of the direct I/O that are to be considered in the FSM design.

The alarm system was initially considered as a Moore FSM. Multiple draft algorithmic state machine (ASM) diagrams were sketched, and it was quickly realised during their development that the complexity of the system may require conditional outputs. The initial Moore implementations were assuming a number of states equal the number of combinations of internal and external inputs. The complexity and size of the resulting ASM was extensive and a Mealy implementation was considered instead. It was hypothesised that the alarm system could be reduced to a binary FSM, with only two output states of either the alarm being on or off. The two output states would be decided conditionally on the inputs, and the timers used to manage the timing of the alarm triggering would behave as inputs. Using LucidChart an ASM was constructed and can be seen in Figure 1.



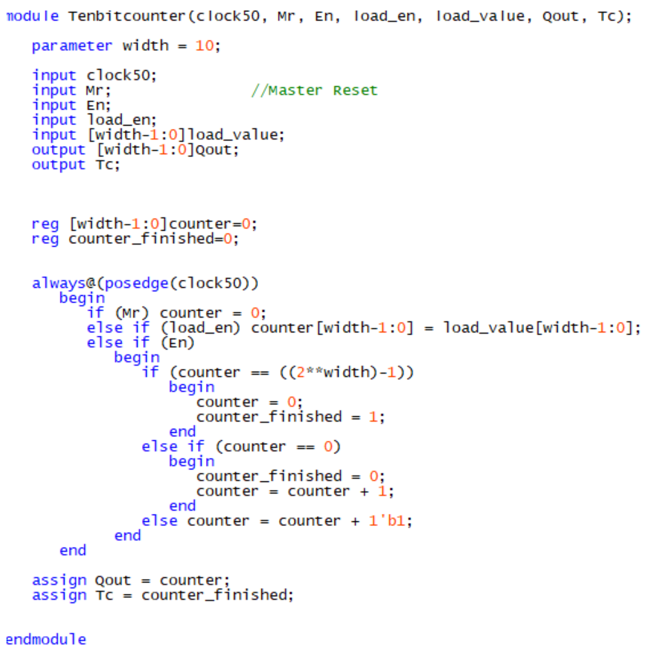
Variables

- AT - Alarm Timer. Alarm timer is the boolean value indicating if the twenty second timer is complete. The alarm may ring while it is low, but not while it is high (unless panic activated).
- ST - Set Timer. Set timer is the boolean value indicating if the ten second timer is complete. If set timer is low, the alarm will not ring (unless panick activated).
- AT_R = Alarm Timer Reset. This indicates amomentary timer reset for the 20 second timer. Internal - can be set.
- ST_R = Alarm Timer Reset. This indicates amomentary timer reset for the 5 second timer. Internal - can be set.
- S = Siren. Siren is the boolean value for if the siren is activated. Internal - can be set.
- L = Light (strobe). L is the boolean value for if the strobe light is activated. Internal - can be set.
- R1/R2/R3 = Room 1, 2, and 3 respectively. This are the boolean value for the sensors in each room. A sensor is high if movement is detected. Input - can't be set.
- A = Armed. This is the boolean value for the armed state of the machine (High for armed state, low for disarmed). Input - can't be set.

Figure 1. ASM for alarm system.

The revised ASM in Figure 1 shows an initial state where the alarm is off, and then a conditional output state where the alarm is triggered. It is to be noted that the panic function was not yet considered in this design as it was thought to be easier to implement a panic override of the whole system through Verilog later. During implementation, the designed FSM would later be realised as an oversimplified attempt at a Mealy interpretation.

It was understood that the FSM would be implemented in the top-level module, so firstly work was conducted on the counting and timing modules. From the lecture material a variable bit counter was coded using Verilog HDL, with a parameter set within the module dictating the bit-width of the counter. Initially a ten-bit wide module was created as the figure counted to of $2^{10} = 1024$ was deemed useful due to its effective use as a value approximate to $10^3 = 1000$. The Verilog implementation of the counter can be seen below in Figure 2.

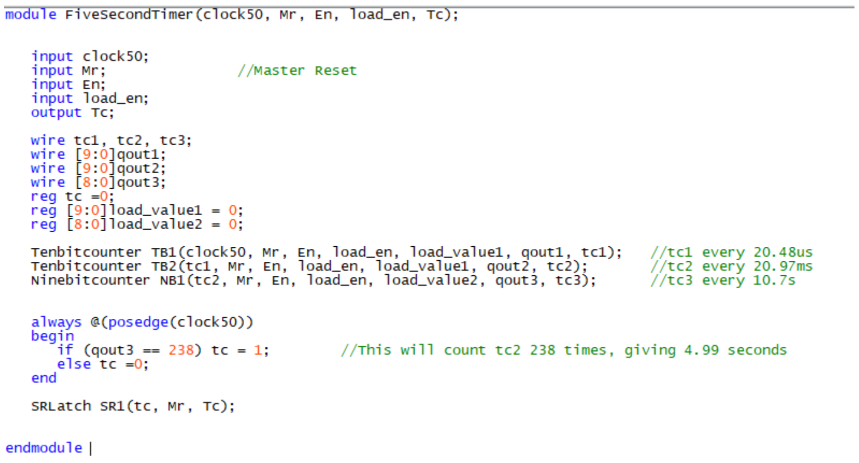


```
module Tenbitcounter(clock50, Mr, En, load_en, load_value, Qout, Tc);

   parameter width = 10;

   input clock50;
   input Mr;              //Master Reset
   input En;
   input load_en;
   input [width-1:0]load_value;
   output [width-1:0]Qout;
   output Tc;


   reg [width-1:0]counter=0;
   reg counter_finished=0;


   always@(posedge(clock50))
      begin
         if (Mr) counter = 0;
         else if (load_en) counter[width-1:0] = load_value[width-1:0];
         else if (En)
            begin
               if (counter == ((2**width)-1))
                  begin
                     counter = 0;
                     counter_finished = 1;
                  end
               else if (counter == 0)
                  begin
                     counter_finished = 0;
                     counter = counter + 1;
                  end
               else counter = counter + 1'b1;
            end
      end

   assign Qout = counter;
   assign Tc = counter_finished;

endmodule
```

Figure 2. Ten-bit counter module.

The loading functionality in the ten-bit counter module (*load_en* and *load_value* inputs) was inherited from the lecture example and, while initially considered redundant, was retained for potential unanticipated use. The Module in Figure 2 shows two outputs Q_{out} and Tc which represent a live representation of the current count and a completion flag respectively. Therefore, Tc would pulse once every 1024 cycles of the input clock.

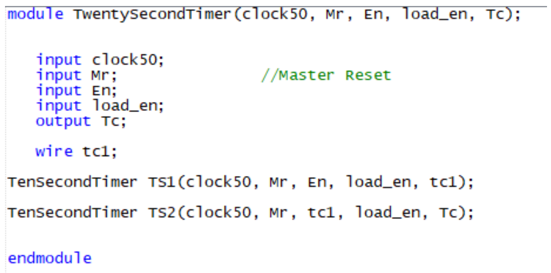
Using the input clock with a 50MHz frequency, the counter modules can take this as input and step down the frequency of the clock by a factor of 1024 with their output. A five second timer was developed to be used for the alarm arm-timer using a cascade of two ten-bit counters and one nine-bit counter. This module is seen in Figure 3 below.



```
module FiveSecondTimer(clock50, Mr, En, load_en, Tc);


   input clock50;
   input Mr;              //Master Reset
   input En;
   input load_en;
   output Tc;

   wire tc1, tc2, tc3;
   wire [9:0]qout1;
   wire [9:0]qout2;
   wire [8:0]qout3;
   reg tc =0;
   reg [9:0]load_value1 = 0;
   reg [8:0]load_value2 = 0;

   Tenbitcounter TB1(clock50, Mr, En, load_en, load_value1, qout1, tc1);   //tc1 every 20.48us
   Tenbitcounter TB2(tc1, Mr, En, load_en, load_value1, qout2, tc2);       //tc2 every 20.97ms
   Ninebitcounter NB1(tc2, Mr, En, load_en, load_value2, qout3, tc3);      //tc3 every 10.7s


   always @(posedge(clock50))
   begin
      if (qout3 == 238) tc = 1;        //This will count tc2 238 times, giving 4.99 seconds
      else tc =0;
   end

   SRLatch SR1(tc, Mr, Tc);

endmodule |
```

Figure 3. Five second timer module.

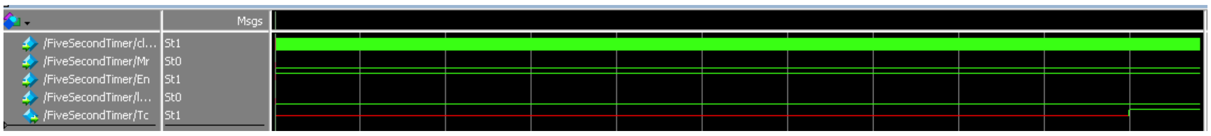
The timer effectively reduces the input clock rate by 10^6 with the two cascading ten-bit counters, and the nine-bit counter is used to manage the smaller count produced by the new 20.97ms stepped-down clock. An always block was used to control the internal tc (timer complete) register, monitoring the qout3 value which held the current count of the nine-bit counter. When the value of qout3 reached the calculated value required for 5 seconds (238) tc was set high. The tc register was then used to control an SR latch (as seen in laboratory three) to control the module output Tc. The latch was used to latch the output unless the timer was decided to be reset, where then all counters and Tc would be reset. A ten second timer module was implemented with near identical code, however qout3 was compared with the number 477 instead. These ten second timers were then cascaded to form a twenty second timer for the alarm triggered timer as seen in Figure 4.



```
module TwentySecondTimer(clock50, Mr, En, load_en, Tc);

   input clock50;
   input Mr;              //Master Reset
   input En;
   input load_en;
   output Tc;

   wire tc1;

TenSecondTimer TS1(clock50, Mr, En, load_en, tc1);

TenSecondTimer TS2(clock50, Mr, tc1, load_en, Tc);

endmodule
```

Figure 4. Twenty second timer module.

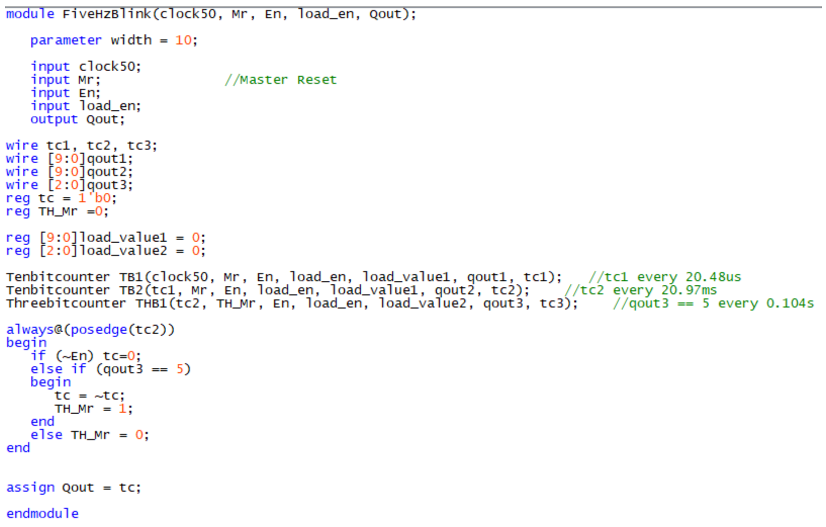
With the alarm trigger and arming timers complete the five second timer was simulated on ModelSim with the results shown in Figure 5. Given the time required to simulate the twenty second timer, the timer was assumed operational with the successful simulation of the five second timer due to the shared code.



```
                           Msgs
/FiveSecondTimer/cl...     St1
/FiveSecondTimer/Mr        St0
/FiveSecondTimer/En        St1
/FiveSecondTimer/l...      St0
/FiveSecondTimer/Tc        St1
```

Figure 5. Five second timer ModelSim Simulation (vertical lines at 0.5s intervals).

With the successful simulation of the timer module, it was next decided to construct the 5Hz blinking module. Using the parametrised counters a cascading counter was developed to step down the clock using two ten-bit and one three-bit counter. The module developed is seen in Figure 6 below.



```
module FiveHzBlink(clock50, Mr, En, load_en, Qout);

   parameter width = 10;

   input clock50;
   input Mr;              //Master Reset
   input En;
   input load_en;
   output Qout;

wire tc1, tc2, tc3;
wire [9:0]qout1;
wire [9:0]qout2;
wire [2:0]qout3;
reg tc = 1'b0;
reg TH_Mr =0;

reg [9:0]load_value1 = 0;
reg [2:0]load_value2 = 0;

Tenbitcounter TB1(clock50, Mr, En, load_en, load_value1, qout1, tc1);   //tc1 every 20.48us
Tenbitcounter TB2(tc1, Mr, En, load_en, load_value1, qout2, tc2);    //tc2 every 20.97ms
Threebitcounter THB1(tc2, TH_Mr, En, load_en, load_value2, qout3, tc3);    //qout3 == 5 every 0.104s

always@(posedge(tc2))
begin
   if (~En) tc=0;
   else if (qout3 == 5)
   begin
      tc = ~tc;
      TH_Mr = 1;
   end
   else TH_Mr = 0;
end


assign Qout = tc;

endmodule
```

Figure 6. Five Hz module.

A 10Hz signal was monitored every time the qout3 register reached 5. The always block used to handle the (qout3 ==5) event was carefully chosen to trigger on the positive edge of tc2. This was important as the (qout3 == 5) event would be held for multiple edges of the faster ticking clock signals (clock50 or tc1) which would produce undesirable behaviour in the module. Using the correct edge input on the always block allowed for the tc register to invert its value every 0.1 seconds, providing a cycle period of 0.2 seconds (5Hz). The module was simulated, and the results can be seen in Figure 7 below.

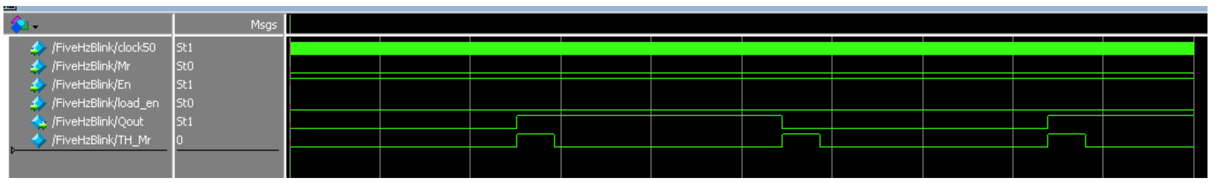
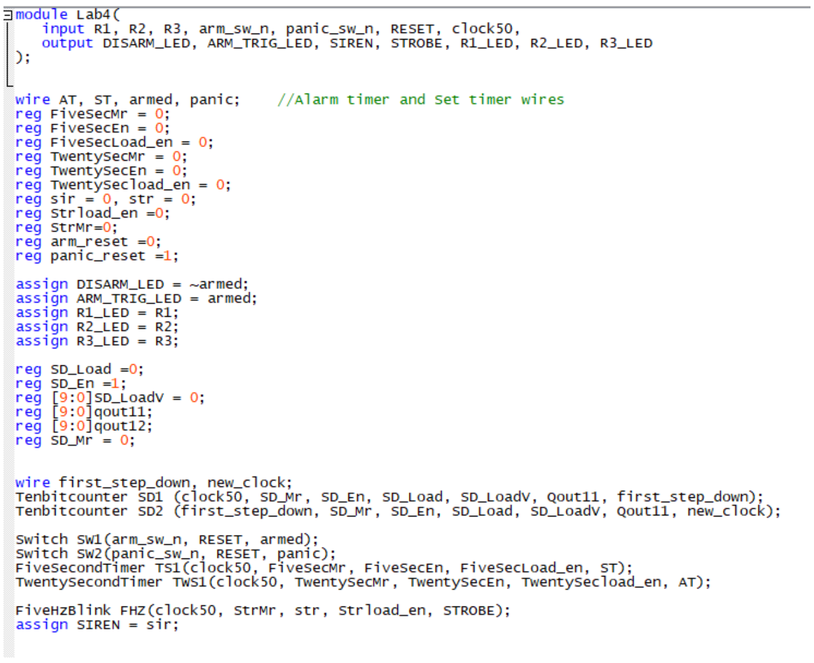


Figure 7. Five Hz module ModelSim simulation (vertical lines at 0.5s intervals).

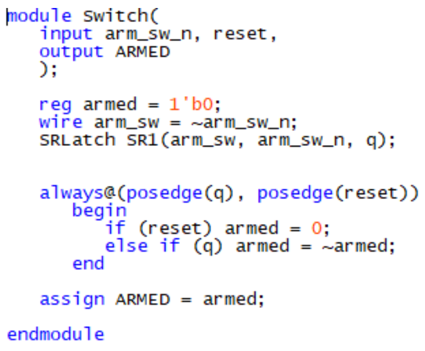
In Figure 7 it can be seen the Qout variable successfully cycles at a rate of 5Hz with a 50% duty cycle.



```
module Lab4(
    input R1, R2, R3, arm_sw_n, panic_sw_n, RESET, clock50,
    output DISARM_LED, ARM_TRIG_LED, SIREN, STROBE, R1_LED, R2_LED, R3_LED
);

wire AT, ST, armed, panic;     //Alarm timer and Set timer wires
reg FiveSecMr = 0;
reg FiveSecEn = 0;
reg FiveSecLoad_en = 0;
reg TwentySecMr = 0;
reg TwentySecEn = 0;
reg TwentySecload_en = 0;
reg sir = 0, str = 0;
reg Strload_en =0;
reg StrMr=0;
reg arm_reset =0;
reg panic_reset =1;

assign DISARM_LED = ~armed;
assign ARM_TRIG_LED = armed;
assign R1_LED = R1;
assign R2_LED = R2;
assign R3_LED = R3;

reg SD_Load =0;
reg SD_En =1;
reg [9:0]SD_LoadV = 0;
reg [9:0]qout11;
reg [9:0]qout12;
reg SD_Mr = 0;


wire first_step_down, new_clock;
Tenbitcounter SD1 (clock50, SD_Mr, SD_En, SD_Load, SD_LoadV, Qout11, first_step_down);
Tenbitcounter SD2 (first_step_down, SD_Mr, SD_En, SD_Load, SD_LoadV, Qout11, new_clock);

Switch SW1(arm_sw_n, RESET, armed);
Switch SW2(panic_sw_n, RESET, panic);
FiveSecondTimer TS1(clock50, FiveSecMr, FiveSecEn, FiveSecLoad_en, ST);
TwentySecondTimer TWS1(clock50, TwentySecMr, TwentySecEn, TwentySecload_en, AT);

FiveHzBlink FHZ(clock50, StrMr, str, Strload_en, STROBE);
assign SIREN = sir;
```

Figure 8. Lab4 module initialisation and sub-module instantiation.

Figure 8 above shows the Verilog used to initialise the top-level module of the alarm system. Registers to carry information between modules are all initialised to their required values to ensure the system boots correctly. Then the 50Hz clock input is stepped down using two ten-bit counters to a 50Hz clock to run the upcoming always block. This was done to ensure the always block performed at a speed that would assist expected behaviour in the module. We then see the instantiation of the five and twenty second timer modules, along with two modules named switch. Finally, the five hertz blinker is instantiated to output to the STROBE LED output with 'str' register input to enable, and the SIREN LED output is assigned to the 'sir' register.



```
module Switch(
    input arm_sw_n, reset,
    output ARMED
    );

    reg armed = 1'b0;
    wire arm_sw = ~arm_sw_n;
    SRLatch SR1(arm_sw, arm_sw_n, q);


    always@(posedge(q), posedge(reset))
        begin
            if (reset) armed = 0;
            else if (q) armed = ~armed;
        end

    assign ARMED = armed;

endmodule
```

Figure 9. Switch module.

Figure 9 shows the Verilog code for the switch module. This module was used to handle the input from the key buttons controlling both the arm and panic functions of the alarm system. A reset input is implemented to assist the overall system reset along with an input of the active low key-switch. This switch module initialises and resets to output low (not armed/panic) and is used to invert the output at the press of the key switch. Using an always block triggered only on the input positive edge, the key switch input avoids repeated cycling. An SR latch could have been implemented to further avoid bouncing of the output which would have improved the design implemented.

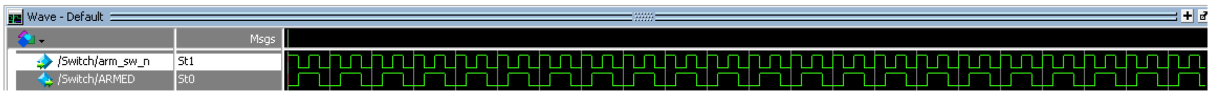
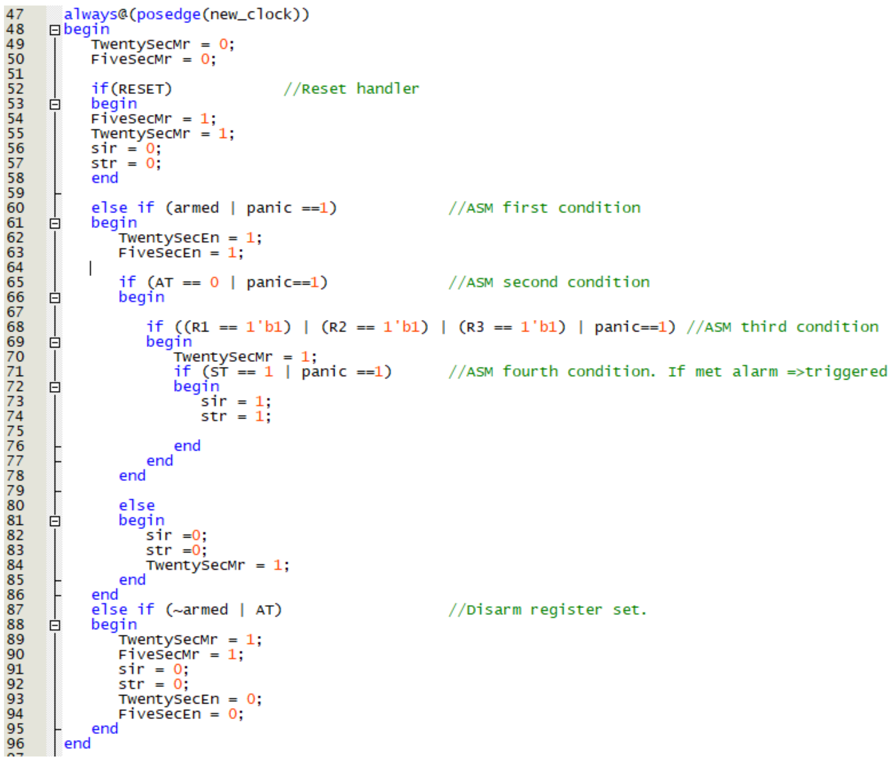


Figure 12. Switch module ModelSim waveform.

The switch module was tested in ModelSim for correct functionality and the waveform is seen in Figure 12 above. The waveform shows an alternating output at half the rate the input is pressed, confirming the desired module behaviour.



```
always@(posedge(new_clock))
begin
    TwentySecMr = 0;
    FiveSecMr = 0;

    if(RESET)               //Reset handler
    begin
    FiveSecMr = 1;
    TwentySecMr = 1;
    sir = 0;
    str = 0;
    end

    else if (armed | panic ==1)             //ASM first condition
    begin
        TwentySecEn = 1;
        FiveSecEn = 1;
    |
        if (AT == 0 | panic==1)             //ASM second condition
        begin

            if ((R1 == 1'b1) | (R2 == 1'b1) | (R3 == 1'b1) | panic==1) //ASM third condition
            begin
                TwentySecMr = 1;
                if (ST == 1 | panic ==1)    //ASM fourth condition. If met alarm =>triggered
                begin
                    sir = 1;
                    str = 1;

                end
            end
        end

        else
        begin
            sir =0;
            str =0;
            TwentySecMr = 1;
        end
    end
    else if (~armed | AT)                   //Disarm register set.
    begin
        TwentySecMr = 1;
        FiveSecMr = 1;
        sir = 0;
        str = 0;
        TwentySecEn = 0;
        FiveSecEn = 0;
    end
end
```

Figure 11. Always block routine in Lab4 module.

In Figure 11 we see the latter part of the Lab4 module comprised entirely of an always block acting on the positive edge of the new_clock (50Hz). After assessing the reset input, a series of 'if' conditions are used to match the conditional expressions seen in the ASM in Figure 1. To allow for the panic override each condition is ORed with the output of the panic switch module. After all the conditions are met, the siren and strobe light enable registers are set high and the alarm is in the second hypothesised 'ON' state. The implementation of the always block has some alterations from the ASM, including the AT_R =1 statement in the ASM (TwentySecMr in Verilog) being moved to before the ST = 1? Condition.

What is seen in Figure 11 is not, in fact a state machine. The always block contains a failed Mealy machine with no conditionality and no clearly defined states. With a higher degree of planning, and a better system design, this could have been avoided. However, the resulting deployment on the DE10-Nano board did prove to have most of the required functionality of the alarm. Only two unexpected behaviours were found. The first was that the 5Hz blinking strobe could be set high even when the alarm was no longer triggered or disarmed. This has a 50% chance of occurring due to the 5Hz blink only being controlled by the enable input, allowing it to be disabled while outputting high, which simply disables the cycling of the Module not the output. Through some modification of the Master Reset input register instantiated in the top-level module this could have been fixed, however, time constraints restricted these modifications before the deployment deadline. The second unexpected behaviour came from inconsistencies in the timing functions for the alarm set and the alarm trigger. Through hands on testing on the DE10-Nano FPGA it was discovered that these timers were not resetting nor enabling at the correct times, and they appeared to always cycle meaning the delay time depended on what time you caught the timer at on its cycle. For example, the 5second timer could be almost instant or take a maximum of 10 seconds if caught with a changing input on the start/end of its cycle. This second unexpected behaviour could have been fixed if the correct implementation of either a Mealy or Moore FSM was designed.

Conclusion

This investigation was very insightful, requiring a high level of design and implementation to get the correct alarm system behaviour. It was learned that the attempt at the Mealy FSM was incomplete and required more design, however this did not become entirely apparent to the researcher until the deployment of the Verilog on the FPGA. On reflection, more time should have been spent on designing the system's ASM and more investigation into Mealy and Moore machines should have been made to ensure an effective FSM design. However, the Alarm system did meet most functionality requirements, showing a strong grasp on the concepts of timing modules, clock stepping, and latching in Verilog. The investigation provided valuable learning for future FPGA projects and will be of excellent experience for the upcoming project assessment.