LABORATORY_5

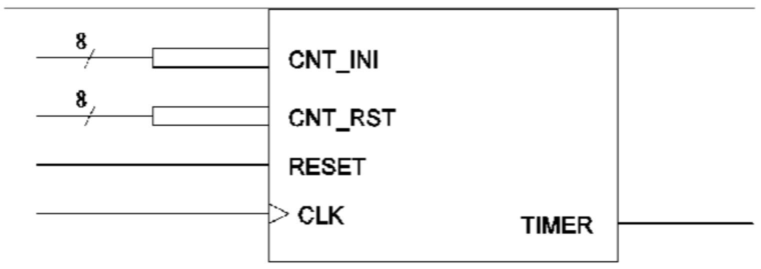
Timer design - laser surgery system

Hannah Hwang | SID : 862419233 | CS120A Section 025
Adithya Chander | SID : 862429692 | CS120A Section 025
Calvin Hong | SID : 862416275 | CS120A Section 025

Code : https://edaplayground.com/x/FHp8

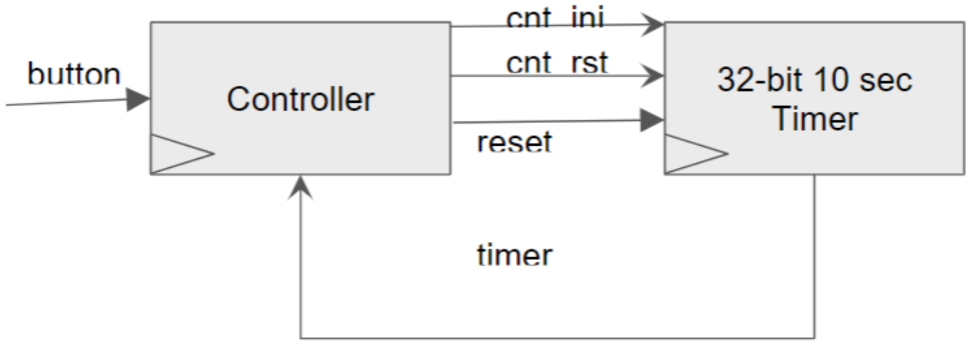
Overview

In part A of the lab, we are told about the implementation of a special purpose timer. We were given the specification that the action of timers is based on the system clock's time division – built as a counter where the most significant bit (MSB) controls the output. [figure 1.] shows how to implement a clock where we can set the initial value(cnt_int) and reset value(cnt_rst) of a timer in order to manipulate how often it ticks. Depending on these values, the number of necessary bits will also differ.



[figure 1.] Timer Structure and Function

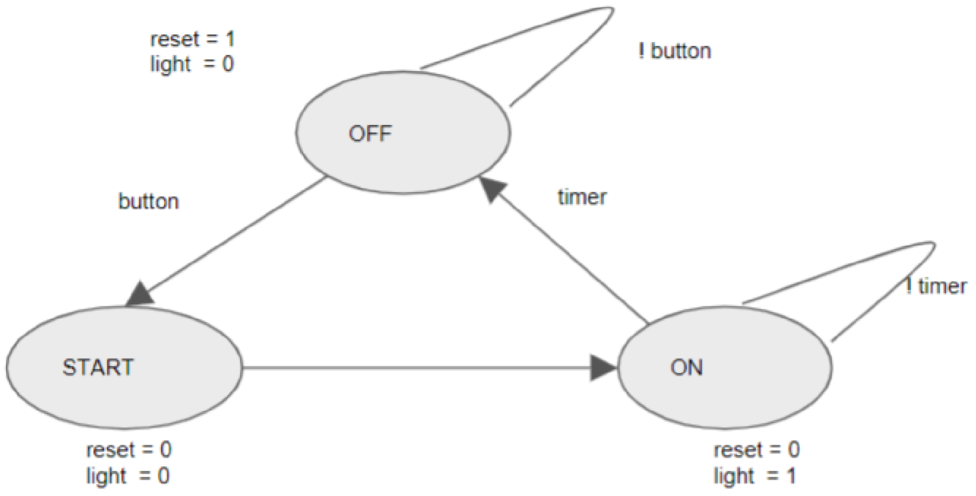
In part B of the lab, we are told about implementation of the timer specified in part A in a laser surgery system. We were given the finite state machine (FSM) diagram of the system as shown in [figure 2.] as well as the diagram of the controller(FSM in [figure 2.]) and timer.



[figure 3.] Timer - Laser Surgery System - Controller and Timer

In part C of the lab, we actually got to implement the systems previously described in Verilog. We were provided with the code for the timer and its sub-modules and had to complete the laser_surgery_sys module (controller). We did so by taking the provided FSM diagram and coding the behaviors of each state to match the FSM, and had button and clock inputs that would help determine when the FSM started and stopped, as well as transition.

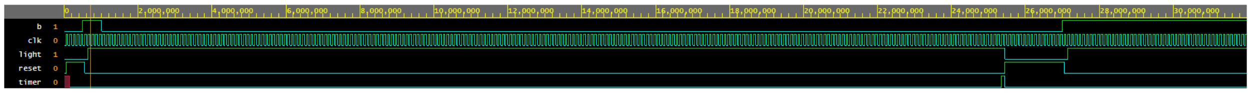
Analysis



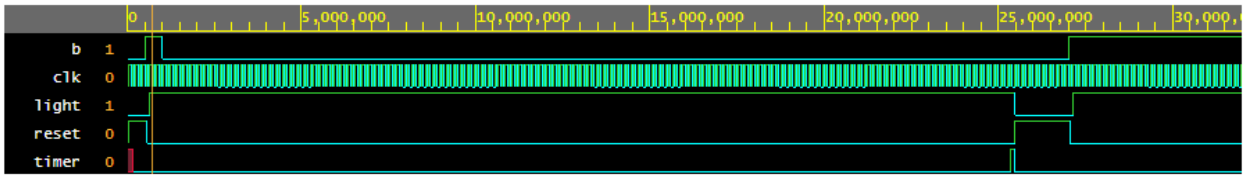
[figure 2.] Timer - Laser Surgery System FSM

Records

Output Waveform:



Output Waveform (zoomed in):



Design.sv

```
// Code your design here
`timescale 1ns/1ps

module laser_surgery_sys #(parameter NBITS = 16)
(       input wire b,
        input wire clk,
      output reg light);

  reg reset;

  wire timer;


  reg [1:0] current_state;
  reg [1:0] next_state;
  wire [NBITS-1:0] cnt_ini ;
  wire [NBITS-1:0] cnt_rst ;

  // seq logic
always @(posedge clk) begin
      current_state = next_state ;
end

  // comb logic


assign cnt_ini = 32'h0000 ;
assign cnt_rst = 32'h00F9 ;

  // comb logic -fsm
localparam OFF = 2'b00 ;
localparam START = 2'b01 ;
localparam ON = 2'b10 ;

always @( * ) begin
  case (current_state)
      OFF : begin
      light = 1'b0;
      reset = 1'b1;
            if (b == 1'b1) begin
                  next_state = START ;
            end
      else begin
                  next_state = OFF ;
            end
      end

      START : begin
      light = 1'b0;
        reset = 1'b0;
        next_state = ON;
      end

    ON: begin
      // code for state transition
      light = 1'b1;
      reset = 1'b0;
            if(timer == 1'b1) begin
          next_state = OFF;
        end
      else begin
              next_state = ON;
            end
      end
    default: begin
        light = 1'b0 ;
      reset = 1'b0 ;
      next_state = OFF ;
      end

endcase
end
```

Testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1ns/1ps

module test;
wire light;
reg clk = 1'b0;
reg b = 1'b0;

// include clock with custom frequency

// instantiate device under test
    laser_surgery_sys uut(.b(b),
        .clk(clk),
        .light(light));

  always #50 clk <= !clk;
```

```
    initial begin
      $dumpfile("dump.vcd"); $dumpvars;
      #500
      b = 1'b1;
      #500
      b = 1'b0;
      #26000
      b = 1'b1;
      #5000
      $finish;
    end
endmodule
```

Discussion

The system works according to the provided specifications, and performs as expected in regards to the FSM and controller diagrams that we were provided. The states transition from a default state (before any actions occur) to a “START” state, then to an “ON” state, which either transitions to another “ON” state or to an “OFF” state, depending on a timer input. If it transitions to the “OFF” state, the only way to transition out of it is to use a button input to transition it back to “START”, which we did using an input wire and an extra test case in the test bench that resent a button input signal once the time reached the point of 25,000,000 picoseconds.

We encountered multiple problems with this lab, including a number of waveform output errors, and code errors themselves, since the lab manual was painfully vague in explaining the way that we should be implementing the things it was asking to. Especially in the case of the testbench, where we were told to use a clock with a custom frequency, that we weren’t shown how to do, nor provided any resources that we could use to learn how to do it properly. However, we were able to resolve a majority of these errors by rewriting the code to use a (working) clock signal, although we aren’t sure if it’s correct or not, due to the aforementioned lack of explanation.

I think the only way we could improve this system would be to have some kind of loop (or a second clock input) to continuously send button input signals after some interval of time. That way we wouldn’t have to hard-code each button input to occur after a delay of X amount, and instead the button inputs would occur on their own and we’d be able to just analyse the waveforms.

Conclusion

The purpose of this lab was to learn about a special purpose timer which we implemented into the provided schematics of a laser surgery system. We learned how to customize the ticking frequency of the clock by setting the cnt_ini (initial) and cnt_rst (reset) variables. The provided laser surgery system specified a 25 microsecond timer which we calculated appropriate cnt_ini and cnt_rst values for. Additionally, we had to use a button and the aforementioned clock to calculate when the state transitions would actually occur. Using the provided code, FSM specifications of the system, and our calculations, the system was successfully implemented in verilog.

Questions

[Part A]

1.Assuming we will set ***cnt_init = 0***, what should ***cnt_rst*** be in order to achieve a tick every 10 seconds using the 25MHz internal clock(CLK) of the FPGA?

Clock frequency : 25MHz = 25 * 10^6 Hz
Total # cycles = clock frequency * tick interval = 25 * 10^6 * 10 = 25 * 10^7 = 250,000,000
Since cnt_init = 0, cnt_rst = 250,000,000 - 1 =
249,999,999(32’b1110111001101011001001111111) in order to achieve a tick every 10 seconds with the given specifications.

cnt_rst would have to be assigned a value of 249,999,999.