LABORATORY # 5
LAB MANUAL

Timer Design - Laser Surgery System

Objectives

1. Design of counters, synthesis and implementation;
2. Usage of internal “clock” signal to drive CLK inputs of flip-flops;
3. Design of special purpose timers

Equipment

- PC or compatible

Software

- EDA Playground Software

Parts

- N/A

Timer Design - Laser Surgery System

Part A: First, we will implement a special purpose timer.

Specification

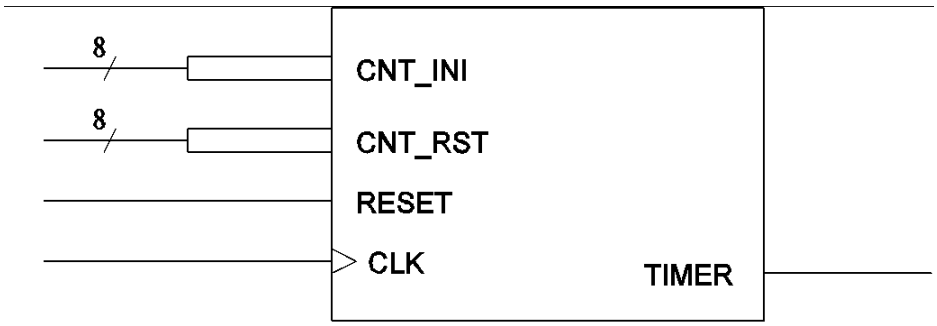

Figure 1. Timer Structure and Function

The action of timers is based on system clock's time division. It is built as a counter whose most significant bit (MSB) controls the output (a tick for example). An 8-bit counter will switch its MSB from 0 to 1 only once per 256 internal clock counts (from 0 to 255). But what if we need a timer that creates a tick every 250 counts (from 0 to 249)?
One way to implement this is by using the timer shown in Figure 1 which counts from **cnt_ini** to **cnt_rst**. In this case we can have **cnt_ini** = 0 (8'b00000000), and **cnt_rst** = 249 (8'b11111001). The timer will count from 0 to 249, at which point it will output a tick (**timer** = 1'b1) before counting again starting from 0. Additionally, the **reset** signal can be used at any time to reset the count back to **cnt_ini**.

Question: 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?

Bear in mind the value of **cnt_ini** and **cnt_rst** will determine the number of bits you will need to represent them. In our case they will be 32 bits wide instead of 8 bits wide as shown in Figure 1.

Part B: Now, we are going to use the timer developed in part A for the purpose of implementing the schematics of a laser surgery system (as described in the textbook). Figure 2 shows the controller (FSM) of the proposed system.

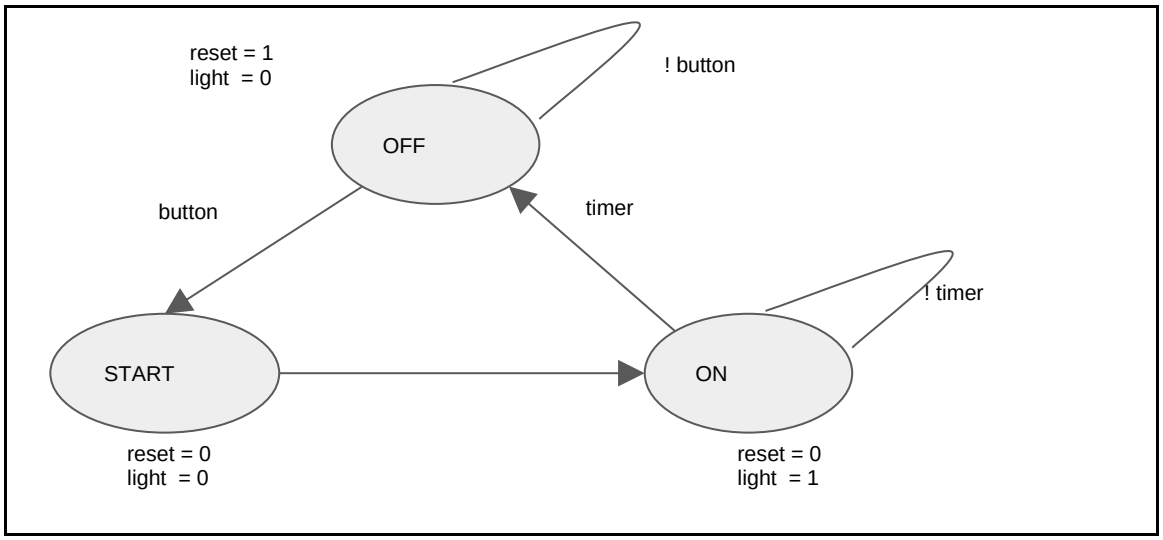

Figure 2. Timer - Laser Surgery System FSM

As shown in figure 2, the laser surgery system starts in the state OFF. In this state, the laser light is low, and the reset signal is high. When the user presses the start button, the system advances to the state START. In this state the laser light continues to be low and reset is set to low. Notice that by setting the reset signal to low the timer is started. In the next clock cycle, the system advances to the state ON. In this state, the laser light is set to high while the reset signal is kept at low. Moreover, the laser surgery system continues in this state until the timer goes high.

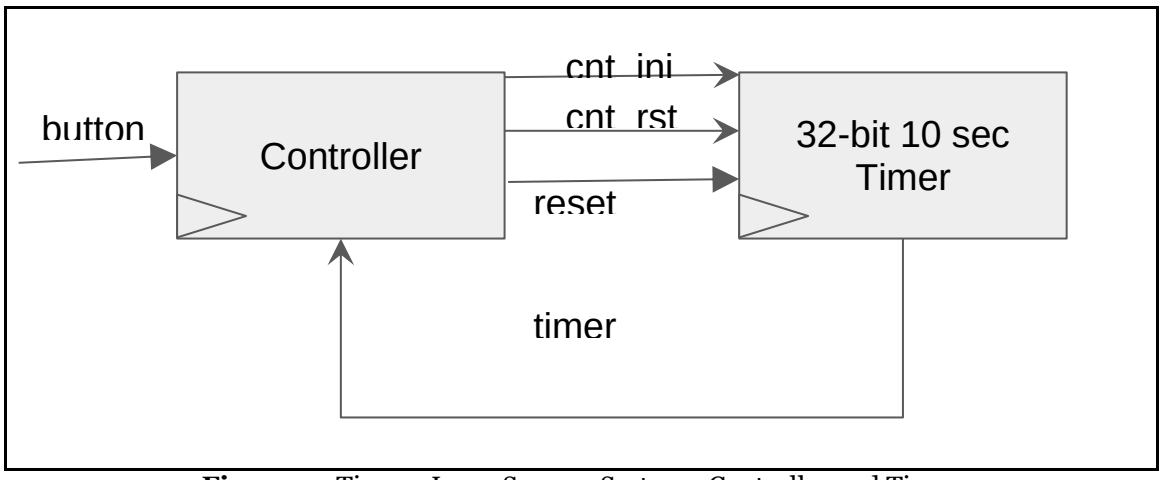

Figure 3. Timer - Laser Surgery System - Controller and Timer

Figure 3 shows the diagram of the controller along with the timer. The controller is the FSM shown in figure 2. The timer is the system developed in part A.

Part C: In this part of the lab, your task is to develop a structural verilog implementation of the systems described in part A and B. The main module of your solution should have the following parameters and ports. **You're only required to complete the laser_surgery_sys module (controller)**. Verilog code for the timer and its sub-modules are given to you. You are of course encouraged to use your own timer code if you like.

Note: When using EDA, the compiler cannot handle large counts, therefore it advised that you do not use a 10sec timer. Suggested value of cnt_rst should not exceed 1000. Using this value, how will you define a 25microsecond timer?

Hint: You will need to define a clock pulse in the testbench, what is the relation between this clock's frequency and the count?

```
`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 ;

// -------------------------------------
// Sequential logic
// -------------------------------------

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

// -------------------------------------
// Comb. Logic
// -------------------------------------

assign cnt_ini = 32'h0000 ;
assign cnt_rst = YOUR VALUE; // 25 μsecs (Custom clock frequency)

// -------------------------------------
// 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
                    // your code for state transition
                end

        ON:     begin
                    // your code for state transition
                end

        default:  begin
                    light = 1'b0 ;
                    reset = 1'b0 ;
                    next_state = OFF ;
                end

endcase

end

// ------------------------------------
// Timer instantiation
// ------------------------------------

timer_st #( .NBITS(NBITS) ) timerst (
                .timer(timer),
                .clk(clk),
                .reset(reset) ,
                .cnt_ini(cnt_ini) ,
                .cnt_rst(cnt_rst)
        );

endmodule
```

In addition, the following set of modules are given.

```
module flopr #(  parameter NBITS = 16 )(
  input clk,
        input reset,
        input [NBITS-1:0] cnt_ini,
        input [NBITS-1:0] nextq,
        output[NBITS-1:0] q
);

reg [NBITS-1:0] iq ;

always @(posedge clk) begin
  if (reset) begin
        iq <= cnt_ini ;
  end
  else begin
        iq <= nextq;
 end
end

assign q = iq  ;

endmodule
```

```
module comparatorgen_st #(  parameter NBITS = 16 )(
output wire r ,
input wire[NBITS-1:0] a ,
input wire[NBITS-1:0] b  );

wire [NBITS-1:0] iresult ;

genvar k ;
generate
for (k=0; k < NBITS; k = k+1)
begin : blk
  xor c1 (iresult[k], a[k], b[k] ) ;
end
endgenerate
```

```
// Reduction plus negation
assign r = ~(|iresult);

endmodule
```

```
module fulladder_st(
output wire r,
output wire cout,
input wire a,
input wire b,
input wire cin
) ;

assign r = (a ^ b) ^ (cin) ;
assign cout = (a & b) | ( a & cin ) | ( b & cin ) ;

endmodule
```

```
module addergen_st #(  parameter NBITS = 16 )(

output wire[NBITS-1:0] r ,
output wire cout  ,
input wire[NBITS-1:0] a ,
input wire[NBITS-1:0] b ,
input wire cin  ) ;

wire [NBITS:0] carry;

assign carry[0]= cin ;

genvar k ;
generate
for (k=0; k < NBITS; k = k+1)
begin : blk
  fulladder_st FA (
            .r(r[k]),
             .cout(carry[k+1]),
```

```
            .a(a[k]),
            .b(b[k]),
            .cin(carry[k]) ) ;
end
endgenerate

assign cout = carry[NBITS] ;
endmodule
```

```
module adder #(   parameter NBITS = 16  )(
input [NBITS-1:0] q ,
input [NBITS-1:0] cnt_ini ,
input [NBITS-1:0] cnt_rst ,
output[NBITS-1:0] nextq,
output tick
);

wire same ;
wire[NBITS-1:0] inextq;

// ---------------------------------------------
// inextq = q + 1 ;
// ---------------------------------------------

addergen_st #(.NBITS(NBITS))
nextval     ( .r(inextq),          // Next value
                                   .cout(),              // Carry out - Don't use
                                   .a(q),                // Current value
                                   .b(16'b0000_0001),     // Plus One
                                   .cin(16'b0000_0000) ) ; // No carry in

// ---------------------------------------------
// Are inextq and cnt_rst equal ?
// ---------------------------------------------

comparatorgen_st #(.NBITS(NBITS))
comparator (
                    .r(same) ,
                    .a(inextq),
```

```
                .b(cnt_rst) );

// ------------------------------------------------------------
// If they are the same produce a tick and set the value for nextq
// ------------------------------------------------------------

assign tick = (same) ? 'd1 : 'd0 ;
assign nextq = (same) ? cnt_ini : inextq ;

endmodule
```

```
module timer_st #(
  parameter NBITS = 32
)
(
   output wire timer ,
   input wire  clk ,
   input wire  reset,
   input [NBITS-1:0] cnt_ini ,
   input [NBITS-1:0] cnt_rst

);

wire [NBITS-1:0] q ;
wire [NBITS-1:0] qnext ;

// Compute the next value

adder #( .NBITS(NBITS ) )
     c1 (.q(q),
          .cnt_ini(cnt_ini),
          .cnt_rst(cnt_rst),
          .nextq(qnext),
          .tick(timer) );

// Save the next state
flopr #( .NBITS(NBITS ) )
     c2 (.clk(clk),
```

```
        .reset(reset),
         .cnt_ini(cnt_ini),
         .nextq(qnext),
         .q(q) );

endmodule
```

Demonstration

testbench.sv

```
`timescale 1ns / 1ps
module test;

  wire light;
  reg clk = 1'b0;
  reg b = 1'b0;
  // Include clock with your custom frequency here

  // Instantiate device under test
  laser_surgery_sys uut(.b(b),
                        .clk(clk),
                        .light(light));
  initial #50000 $finish;
  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    #500
    b = 1'b1;
    #500
    b = 1'b0;
  end
endmodule
```

Procedures

N/A

Presentation and Report

Must be presented according to the general EE/CS 120A lab guidelines posted in iLearn.

Prelab

1. Review the Chapter 3 Lecture
2. Try to answer all the questions, prepare logic truth tables, do all necessary computations