

EECS120A Logic Design
Department of Electrical Engineering
University of California – Riverside

Laboratory #6
EECS 120 A
Spring 2020

LABORATORY # 6
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
- Digilent’s Basys Spartan-3E FPGA Evaluation Board

Software

- Xilinx ISE Design Software Suite
- ModelSim XE III modeling software
- Digilent’s Adept ExPort Software

Parts

- N/A

Timer Design - Laser Surgery System

Part A: In this development, we will implement the schematics for a special purpose 1 second timer.

Specification

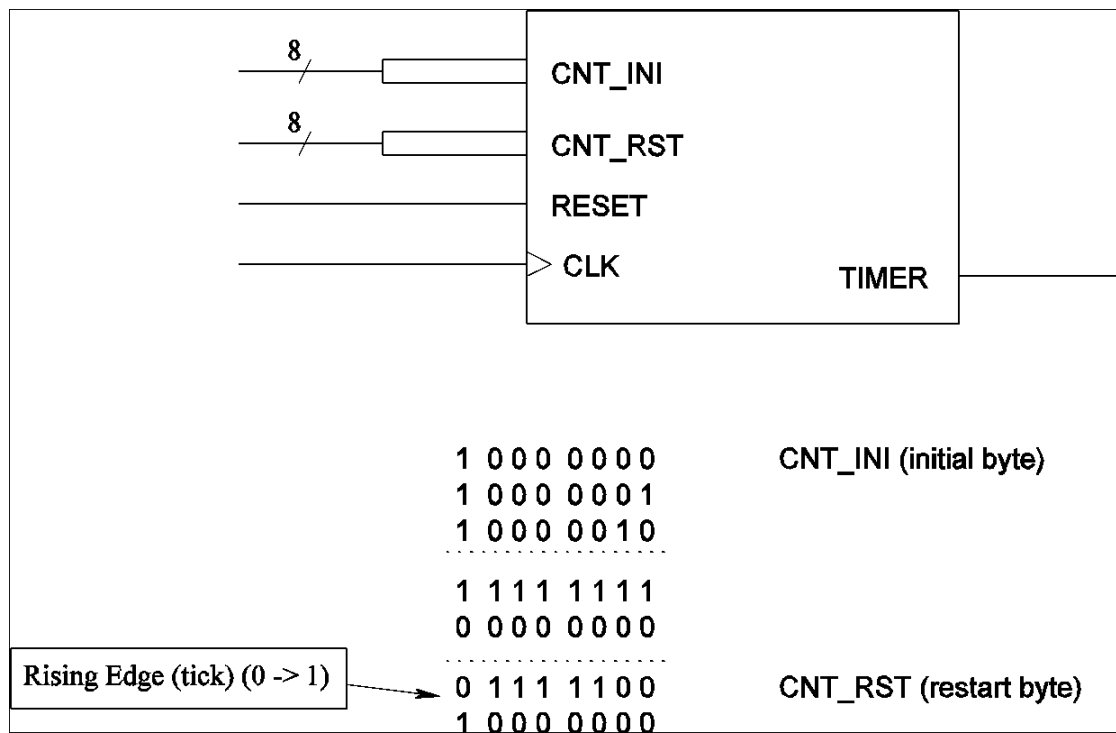


Figure 1. Timer Structure and Function

The action of timers is based on system clock's time division. It is built as a counter whose 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) starting from MSB=1? From this description we can infer the required timer block diagram as shown in Figure 1. In figure 1 notice that the number of signals in the ports **cnt_ini** and **cnt_rst** has to be modified: the higher the frequency of the clock in the board, the more signals in these ports.

The task is to create and implement a timer which uses 50 MHz internal clock (CLK) and output a timer tick every second.

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 text book). In order to make the system easy to test, first, we have to change the timer so as to make it to tick every 10 seconds. Figure 2 shows the controller (FSM) of the proposed system.

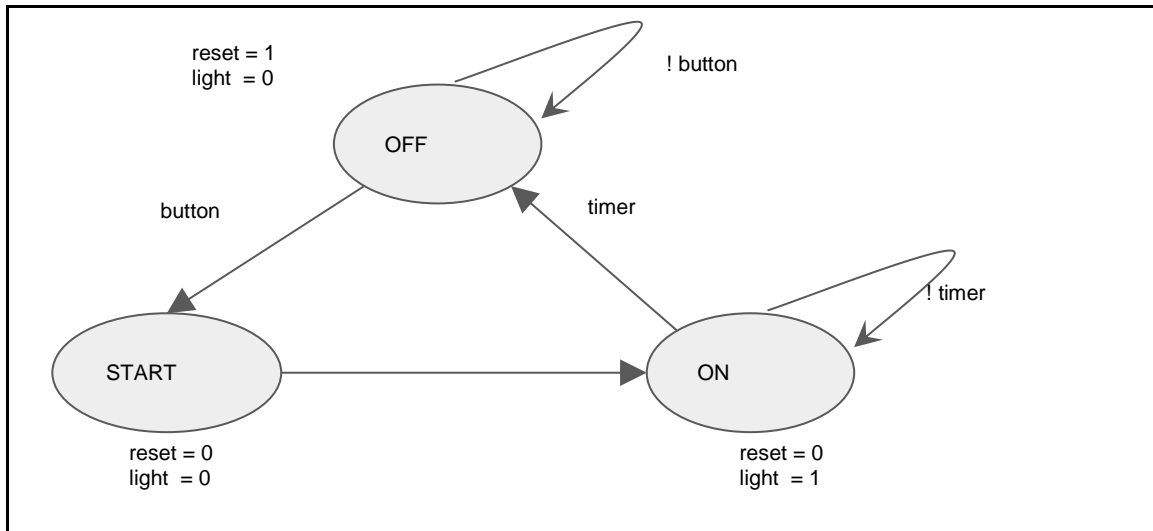


Figure 2. Timer - Laser Surgery System FSM

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 resets is set to low as well. 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 high while the reset signal is low. Moreover, the laser surgery system continues in this state until the timer goes high.


```

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; // 10 secs ( 25 MHZ internal clock )

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

localparam OFF  = 2'b00 ;
localparam START = 2'b01 ;
localparam ON   = 2'b10 ;

always @( current_state ) begin

case (current_state)

    OFF : begin
            // your code for state transition
        end

    START : begin
            // your code for state transition
        end

    ON:    begin
            // your code for state transition
        end
endcase
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
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] irestult ;

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

// Reduction plus negation
assign r = ~(irestult);

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 ;
// -----

adder gen_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 ?
// -----

comparator gen_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 : 'do ;
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

```

Question and Report

1. Show the simulation result from testbench
2. How can we change the light on time?(For example, 20s or 100s?)