EE/CS120A Logic Design
Department of Electrical Engineering
University of California – Riverside

Mini-Term Project
Prof. Kelly Downey
Winter 2019

Mini-Term Project
4-way light controller

Dan Murphy; *862*

Project Overview

The goal of this mini-project is to create a four-way traffic light controller through an FPGA BASYS 2 board. This traffic light controller simulates a simplified stop light controller where the light states negate the “slow” (i.e. yellow light) state and turn signals. Each light will illuminate for three seconds.

Part 1:
By mapping out the logic of the four-way traffic light system, we were able to create a HLSM, a data-path and a controller for this system. These design visualizations are provided below to portray the logic of the Boolean traffic-light system.

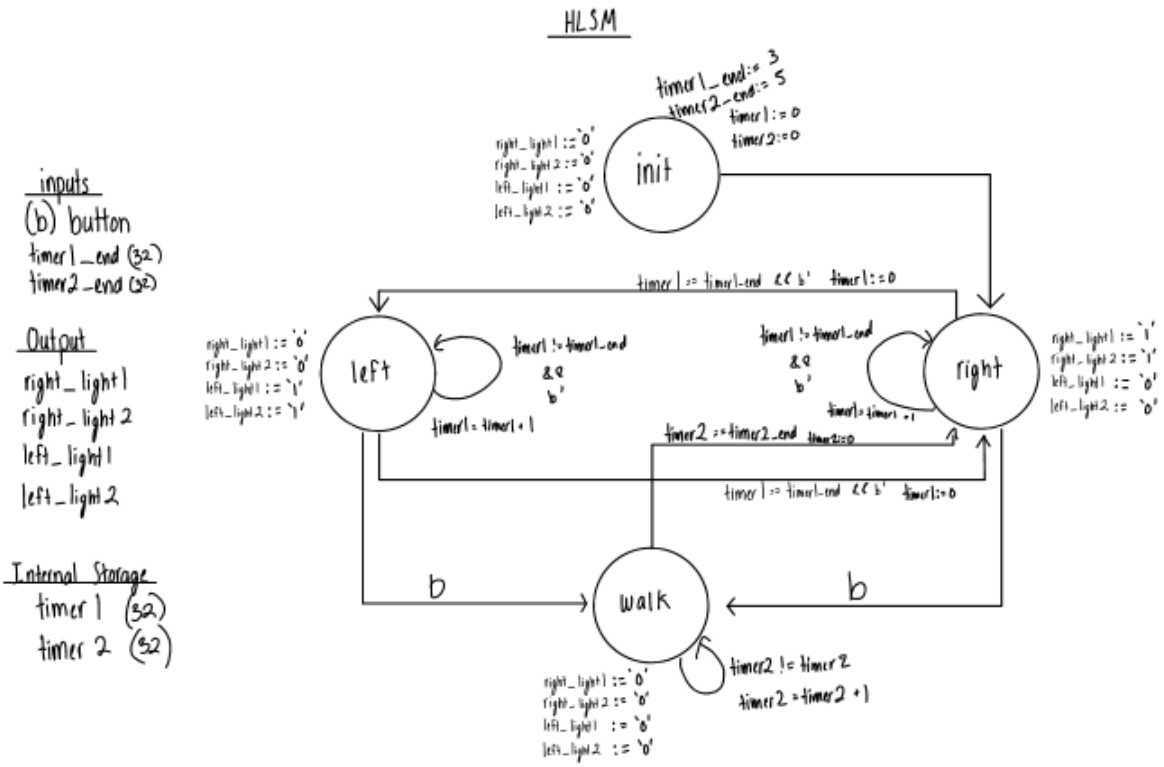
Part 2:
Building off of part 1, we implemented the logic of the four-way traffic light system into Verilog code. This Verilog code allows us to simulate within Xilinx to output to our FPGA BASYS 2 board. The board outputs the different lights as well as a crosswalk button from a “pedestrian”.

Part 3:
To ensure the timing is correct on the four-way traffic light system, the 50 MHz clock source is utilized. Due to the clock source being set to 50 MHz, the timer ticks at 3 seconds per light and 5 seconds for the crosswalk button.

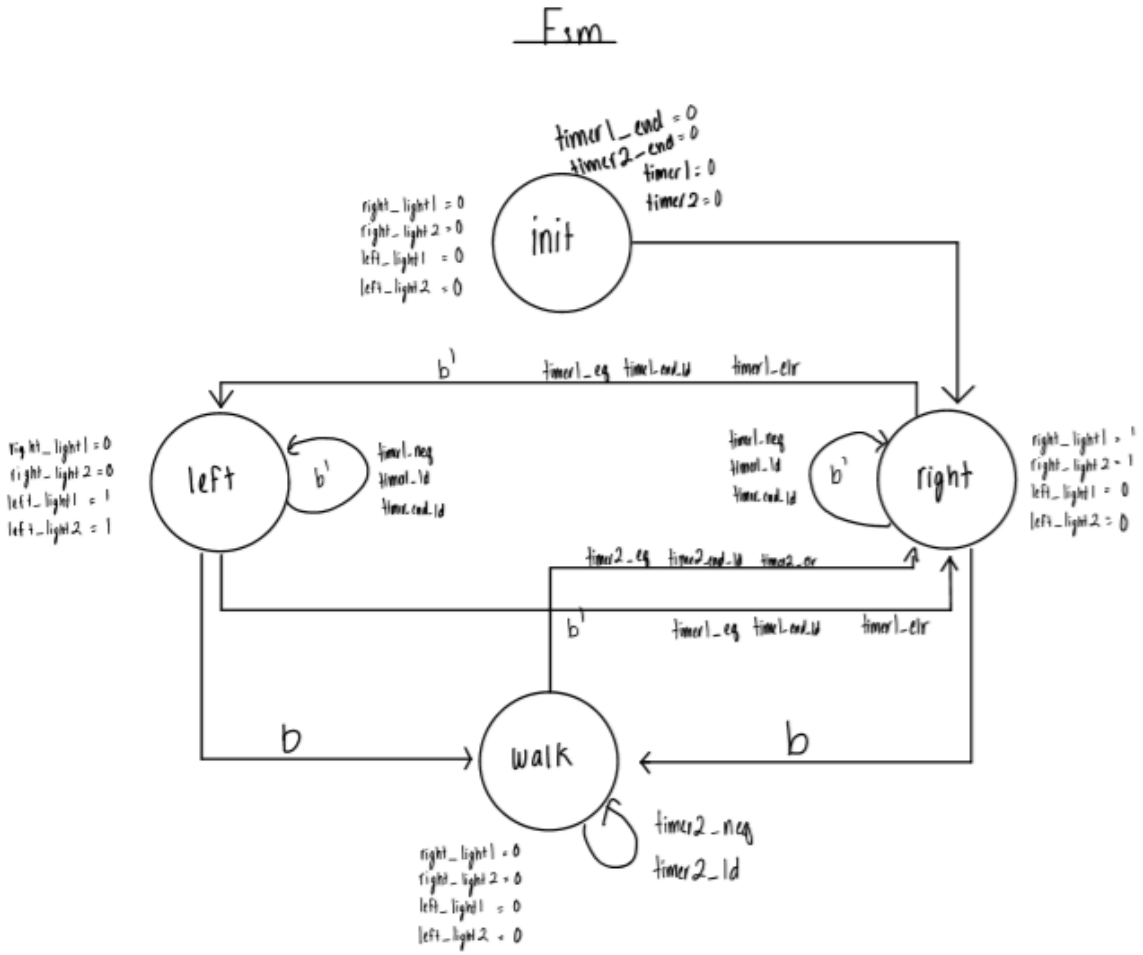
Design Constraints

The FPGA BASYS 2 board can only illuminate green LEDs for purposes of this project. Due to this restriction, when an LED is illuminated (green), this represents a [green] traffic light (e.g. “go”); when an LED is not illuminated, this represents a [red] traffic light (e.g. “stop”). For the cross-walk implementation, the way this is portrayed is through all LEDs turned off (which represents all lights as [red] e.g. “stop”) thus allowing a pedestrian to cross the street.

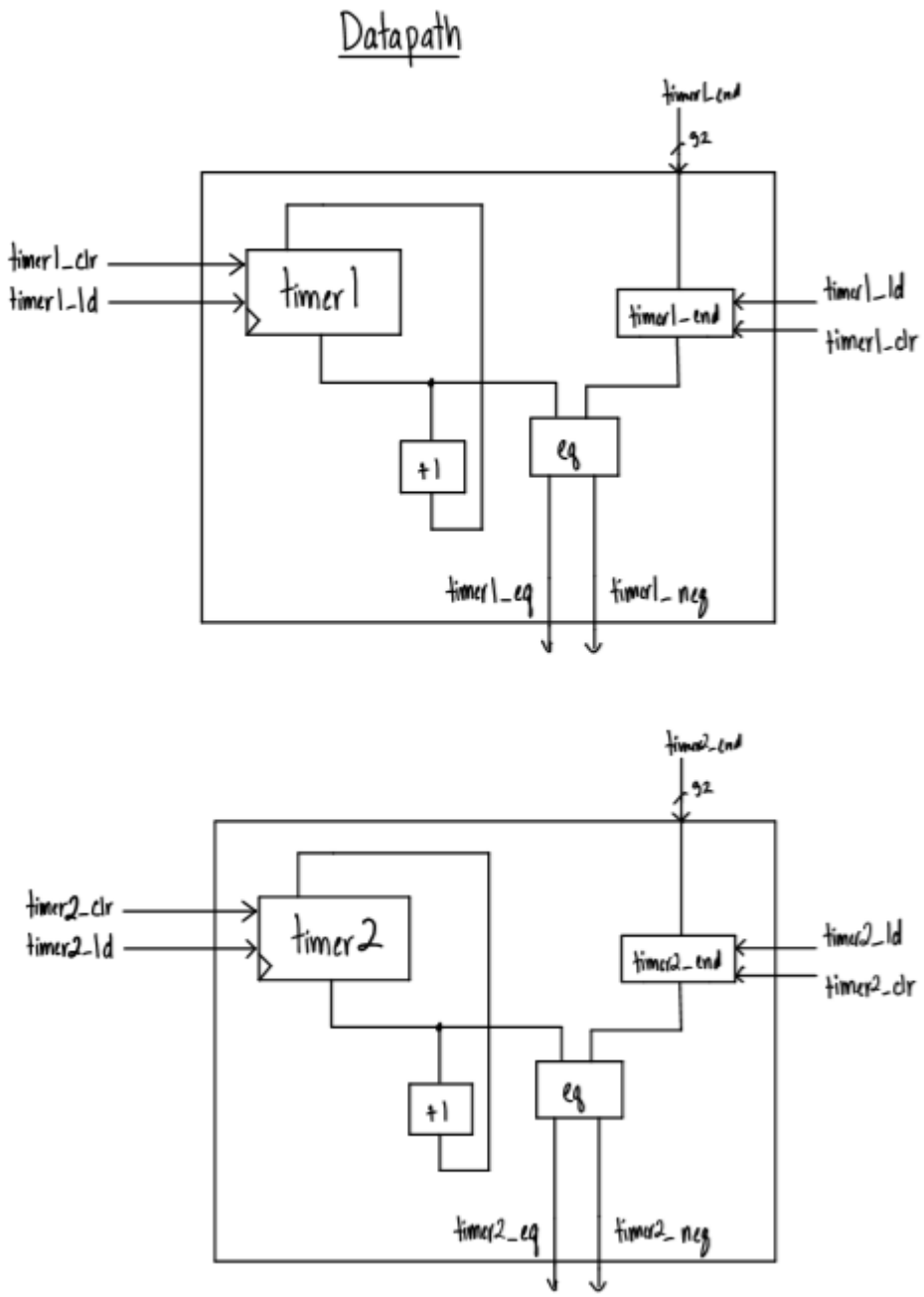
High-Level State Machine



Finite State Machine



Datapath


Datapath
timer1_end
32
timer1_clr
timer1_ld
timer1
timer1_end
timer1_ld
timer1_clr
+1
eq
timer1_eq
timer1_neq
timer2_end
32
timer2_clr
timer2_ld
timer2
timer2_end
timer2_ld
timer2_clr
+1
eq
timer2_eq
timer2_neq

Verilog Code

```
`timescale 1ns / 1ps


// Same name as Lab 6 because we copied and paste the for the timer
module laser_surgery_sys #( parameter NBITS = 32 ) (
input wire b ,
input wire clk ,
output reg right_light1 ,
output reg right_light2 ,
output reg left_light1  ,
output reg left_light2
);


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

wire timer;
wire walk_timer;
reg reset_count;
reg reset_walk;

reg [1:0] current_state = 2'b00;
reg [1:0] next_state = 2'b00;

// State initialization -------------------
localparam RIGHT = 2'b00;
localparam WALK = 2'b10;
localparam LEFT = 2'b01;

// Traffic timer initialization of 32-bit register (i.e. [n-1:0]) -----------------------
reg [31:0] cnt_ini = 32'd0;
reg [31:0] cnt_rst = 32'd300000000;

// Walk timer initialization of 32-bit register (i.e. [n-1:0]) -----------------------
reg [31:0] cnt_iniw = 32'd0;
reg [31:0] cnt_wlk = 32'd500000000;

always @(posedge clk) begin
        current_state <= next_state;
end // end of local scope -----------------

// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
//    --- SWITCH STATEMENT --- SWITCH STATEMENT --- SWITCH STATEMENT ---
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
always @( current_state or b or timer) begin
```

```
case(current_state)                         // Switch statement
        RIGHT : begin                       // RIGHT case --------------------------
                next_state = RIGHT;         // Re-initialize values
                reset_count = 0;
                reset_walk = 1;
                right_light1 = 1'b1;
                right_light2 = 1'b1;
                left_light1 = 1'b0;
                left_light2 = 1'b0;
                next_state = timer ? LEFT : RIGHT;
                if(1) begin                 // IF statement -------------------
                        if(b) begin         // Nested if ---
                        next_state = WALK;  // Assign cross-walk to the next state
                        end                 // End of local scope
                        else begin          // ELSE statement ---
                        next_state = LEFT;  // Assign left light to the next state
                        end                 // End of local scope
                end                         // End of local scope
                else                        // ELSE statement ---
                        next_state = RIGHT; // Assign right light to the next state
                end                         // End of local scope

        // ====================================================
        LEFT : begin                        // LEFT case --------------------------
                next_state = LEFT;          // Re-initializing values
                reset_count = 0;
                reset_walk = 1;
                right_light1 = 1'b0;
                right_light2 = 1'b0;
                left_light1 = 1'b1;
                left_light2 = 1'b1;
                next_state = timer ? RIGHT : LEFT;
                if(1) begin                 // IF statement -------------------
                        if(b) begin         // Nested if ---
                        next_state = WALK;  // Assign cross-walk to the next state
                        end                 // End of local scope
                        else begin          // ELSE statement ---
                        next_state = RIGHT; // Assign right light to the next state
                        end                 // End of local scope
                end                         // End of parent scope
                else                        // ELSE statement ---
                        next_state = LEFT;  // Assign left light to the next state
                end                         // End of local scope

        // ====================================================
        // When button is pressed, cross-walk starts after prev state finishes ----

        WALK : begin                        // WALK case --------------------------
                reset_count = 1;            // Re-initializing values
                next_state = WALK;
```

```
                        reset_walk = 0;
                        right_light1 = 1'b0;
                        right_light2 = 1'b0;
                        left_light1 = 1'b0;
                        left_light2 = 1'b0;
                        next_state = walk_timer ? RIGHT : WALK;
                        end                             // End of local WALK scope --------

                // ================================================
                default : begin                         // DEFAULT case ----------------------
                        reset_count = 1'b1;             // Re-initializing values
                        right_light1 = 1'b0;
                        right_light2 = 1'b0;
                        left_light1 = 1'b0;
                        left_light2 = 1'b0;
                        next_state = RIGHT;
                        end                             // End of local DEFAULT scope ---
        endcase // End of switch statement ------------------------------------------------------
end             // End of local scope

// ---------------------------------------
// Timer instantiation
// ---------------------------------------
timer_st #( .NBITS(NBITS) ) timerst (
.timer(timer),
.clk(clk),
.reset(reset) ,
.cnt_ini(cnt_ini) ,
.cnt_rst(cnt_rst)
);


// Instantiating a new timer for the walk timer ------------------
timer_st #( .NBITS(NBITS) ) timerw (
.timer(walk_timer),
.clk(clk),
.reset(reset_walk) ,
.cnt_ini(cnt_iniw) ,
.cnt_rst(cnt_wlk)
);
endmodule

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 ;
```

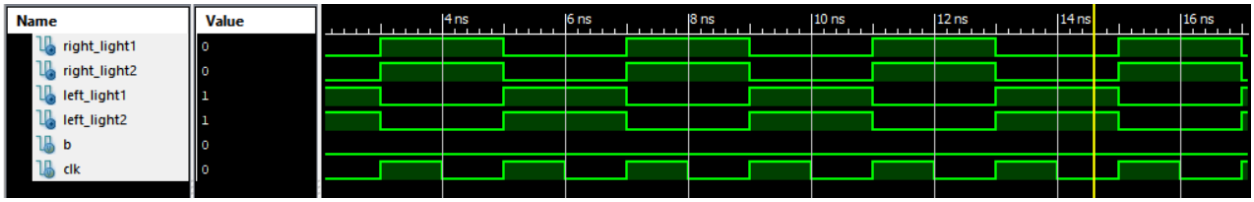
```
endmodule


// TIMER ---------------------------------------------------------------------

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

Testbench Code

```
‘timescale 1ns / 1ps

module test2;

        // Inputs
        reg b;
        reg clk;
```

```
	// Outputs
	wire right_light1;
	wire right_light2;
	wire left_light1;
	wire left_light2;

	// Instantiate the Unit Under Test (UUT)
	laser_surgery_sys uut (
		.b(b),
		.clk(clk),
		.right_light1(right_light1),
		.right_light2(right_light2),
		.left_light1(left_light1),
		.left_light2(left_light2)
	);

	initial begin
		// Initialize Inputs

		b = 0;
		while(1) begin

		if(clk == 0) begin
		clk = 1;
		end
		else begin
		clk = 0;
		end
		#1;
		end

	end

endmodule ////////////////////////////////////////////////////////////////////////////////////////////////////
```

Timing Diagram



YouTube Demo

https://www.youtube.com/watch?v=_vPp8j59lSM

Issues

The most prevalent issue stems from the cryptic errors within the IDE. Debugging efficiently only goes so far until compiler errors provide a broad scope as to where the error lies. Initially, initializing variables proved to be a challenge due to the auto-generated code.

Contingencies

In order to prevent complete frustration and inherent disaster, we decided to keep record of our mapped-out logic on an iPad's file viewer. This allowed us to reference and manipulate logical steps into our code as well as to determine if our diagrams visuals were flawed or not. In regards to debugging cryptic errors within the IDE, line-by-line code & logic evaluation pointed out the flaws. Once realizing the auto-generated code is not what we needed, we went back to our previous, functional build and quickly realized the mistake that was made. The functional code was ported over to the current build and the initialization issue vanished.

Potential Improvements

Improvements can always be made in all systems. In this particular system, I believe that implementing the logic behind the [yellow] "slow" light. In addition to this, a sensor for vehicles can be implemented as well to better simulate an actual four-way traffic stop scenario.

Conclusion

This project is intended to emulate the design and functionality of a four-way traffic light control system. Through the implementation of the HLSM, FSM and data-path, our Verilog code has been synchronized as one and correctly simulates the simplified four-way traffic stop. Our FPGA BASYS 2 board is able to output LEDs to show when traffic lights are [green] on (e.g. "go") and [red] off (e.g. "stop"). Each respective light sequence remains illuminated for 3 seconds each. When the button is pressed, all lights are [red] off (e.g. "stopped") for 5 seconds which allows for the pedestrian to cross the street safely. All necessary functionality is present and output yielded correct simulated results.