CS/EE 120A Mini Project Report

Name: Micah Galos
Email: mgalo001@ucr.edu
Partner's Name: Ronny Anariva
Mini Project Title: Four-way Traffic Light Controller

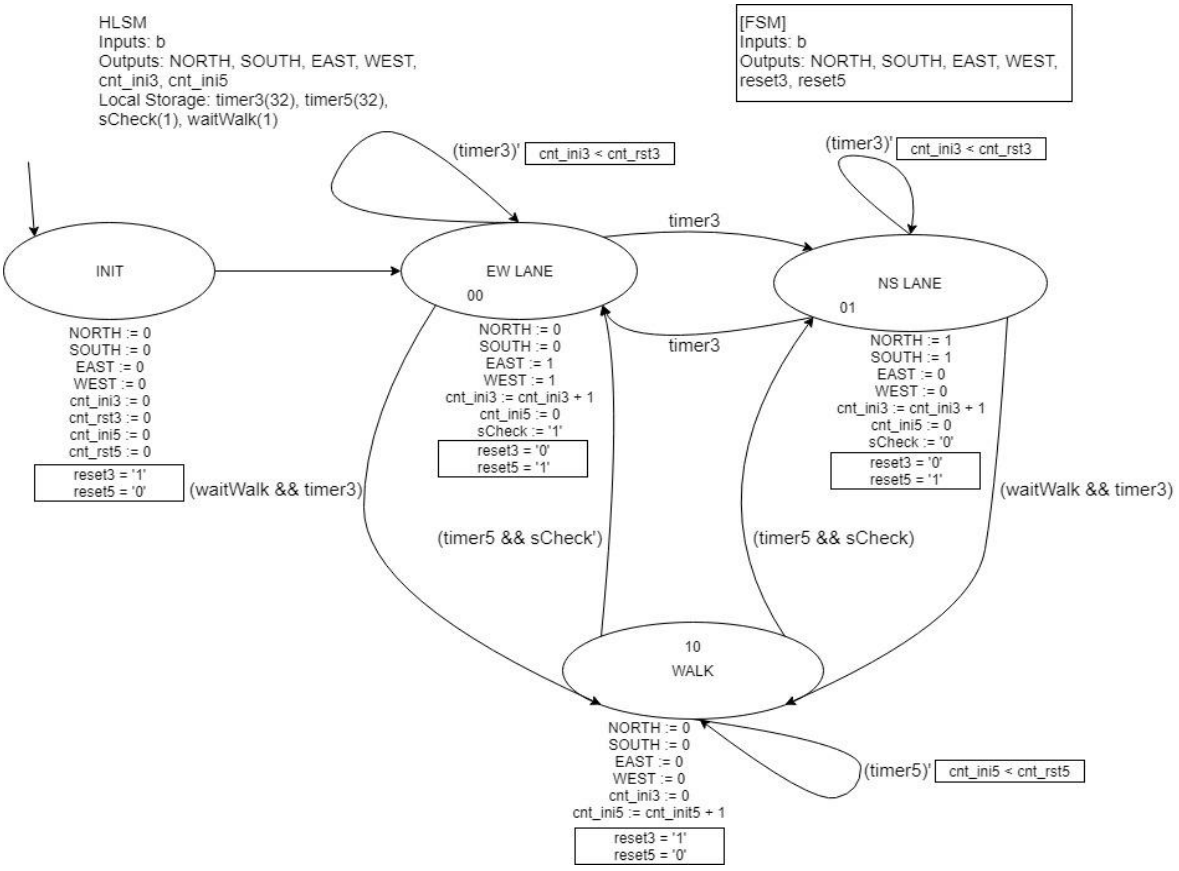
1. Detailed specification of the project

- Purpose:
  Create a design for a 4-way traffic light controller from our Basys 2 boards. From the design process, implementation and problem solving it gives us a small representation to simulate a real-world application in Verilog.

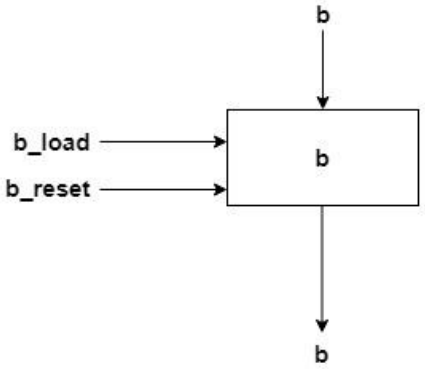
- Explanation of any design constraints
  There are the four lights that correspond to the horizontal and vertical lanes of a four-way street intersection: North, South, East, and West. The North and South lane being the vertical while East and West is the horizontal. These are the output registers in the design process while the input wires are represented by the button press (b) and clock (clk). In addition, we added several registers to keep track of the state transitions going from the North-South lanes to the East-West and the Walk state in which the user presses the button as the cycle of either North-South or East-West lanes are over. The first of the registers are the reset timers which tells us when the clock cycle is over for the alternating lanes for three seconds as well as the timer when the traffic lights are in the crosswalk state for five seconds. The “check” registers are called sCheck, which checks the previous state transitioning to the crosswalk. If we transition to the walk state from the North-South lanes, then the sCheck holds a zero while the East-West holds a one. For the button press, I assigned it to a register called waitWalk, which holds a ‘1’ when a button has been pressed within either lane states. Most of the overall Verilog code was inspired from the laser eye surgery lab, which I utilized for this design.

2. System Design, Architecture and Performance

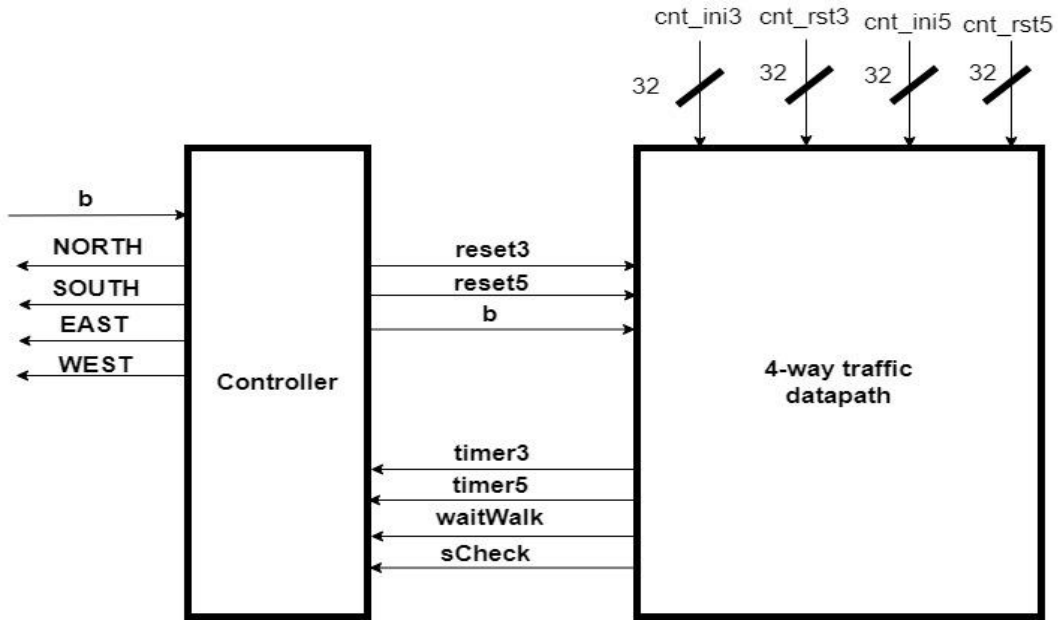
HLSM and FSM



Four-way Traffic Datapath



Four-way Traffic Controller



- Circuit schematic

Four-way Traffic Constraints

```
# clock pins for basys board
//Input:
NET "clk" LOC= "B8";
NET "b" LOC= "G12";

//Output:
NET "NORTH" LOC= "M5";
NET "SOUTH" LOC= "M11";
NET "EAST" LOC= "P7";
NET "WEST" LOC= "P6";
```

Behavioral Module Code

```
`timescale 1ns / 1ps

module fourway_traffic_bh #(parameter NBITS = 32)(
    input wire b, input wire clk,
    output reg NORTH, output reg SOUTH,
    output reg EAST, output reg WEST
    );

        // Check registers
        reg sCheck = 1'b0;
        reg waitWalk = 1'b0;

        // State Transitions
        reg [1:0] curr;
        reg [1:0] next;

        // 3 Seconds
        reg reset3 = 1'b0;
```

```
        wire timer3;
        wire [NBITS-1:0] cnt_ini3;
        wire [NBITS-1:0] cnt_rst3;

        // 5 Seconds
        reg reset5 = 1'b0;
        wire timer5;
        wire [NBITS-1:0] cnt_ini5;
        wire [NBITS-1:0] cnt_rst5;

// ---------------------------------------
// Comb. Logic
// ---------------------------------------
assign cnt_ini3 = 32'D0000 ;
assign cnt_rst3 = 32'D150000000; // 3 secs ( 25 MHZ internal clock )

assign cnt_ini5 = 32'D0000 ;
assign cnt_rst5 = 32'D250000000; // 5 secs

// ---------------------------------------
// Comb. Logic - FSM
// ---------------------------------------
localparam EWLANE = 2'b00;
localparam NSLANE = 2'b01;
localparam WALK = 2'b10;

// ---------------------------------------
// Sequential logic
// ---------------------------------------
always @(posedge clk)
        begin
        curr <= next;

        // Check for button press
        if(reset3)
                begin
                waitWalk = 1'b0;
                end

        if(b)
                begin
                waitWalk = 1'b1;
                end

        // Check for moving from lane to lane
        if(curr == NSLANE)
                sCheck = 1'b0;

        if(curr == EWLANE)
                sCheck = 1'b1;

        end

always @(curr or b or timer3)
        begin
                case (curr)
                default:
```

```
                        begin
                                NORTH = 1'b0;
                                SOUTH = 1'b0;
                                EAST = 1'b0;
                                WEST = 1'b0;
                                reset3  = 1'b1;
                                reset5 = 1'b0;
                                next = NSLANE;
                        end

                        EWLANE:
                                begin
                                        NORTH = 1'b0;
                                        SOUTH = 1'b0;
                                        EAST = 1'b1;
                                        WEST = 1'b1;
                                        reset3  = 1'b0;
                                        reset5 = 1'b1;
                                        if(1)
                                                begin
                                                        if(waitWalk)
                                                                begin
                                                                next = WALK;
                                                                end
                                        else
                                                begin
                                                next = NSLANE;
                                                end
                                        end // if(timer3)
                                        else
                                                begin
                                                next = EWLANE;
                                                        end
                                end // EWLANE state

                        NSLANE:
                                begin
                                        NORTH = 1'b1;
                                        SOUTH = 1'b1;
                                        EAST = 1'b0;
                                        WEST = 1'b0;
                                        reset3  = 1'b0;
                                        reset5 = 1'b1;
                                        if(1)
                                                begin
                                                        if(waitWalk)
                                                                begin
                                                                next = WALK;
//moves on to the WALK state
                                                                end
                                                        else
                                                                begin
                                                                next = EWLANE;
//moves on to the OFF state
                                                                end
                                                end // if(timer3)
                                        else
```

```
                                        begin
                                        next = NSLANE; // stays in the
loop
                                        end
                        end // NSLANE state

                WALK:
                        begin
                                NORTH = 1'b0;
                                SOUTH = 1'b0;
                                EAST = 1'b0;
                                WEST = 1'b0;
                                reset3  = 1'b1;
                                reset5 = 1'b0;
                                        if(1)
                                                begin
                                                        if(sCheck)
                                                                begin
                                                                next = NSLANE;
                                                                end
                                                        else
                                                                begin
                                                                next = EWLANE;
                                                                end
                                        end // if(timer5)
                                        else
                                                begin
                                                next = WALK;
                                                end
                        end // WALK state
        endcase // case(curr)
end // always @(curr)

// ---------------------------------------
// Timer instantiation
// ---------------------------------------
timer_st #( .NBITS(NBITS) ) timerst3(
        .timer(timer3), .clk(clk), .reset(reset3),
        .cnt_ini(cnt_ini3), .cnt_rst(cnt_rst3)
);


timer_st #( .NBITS(NBITS) ) timerst5(
        .timer(timer5), .clk(clk), .reset(reset5),
        .cnt_ini(cnt_ini5), .cnt_rst(cnt_rst5)
);

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 ;
```

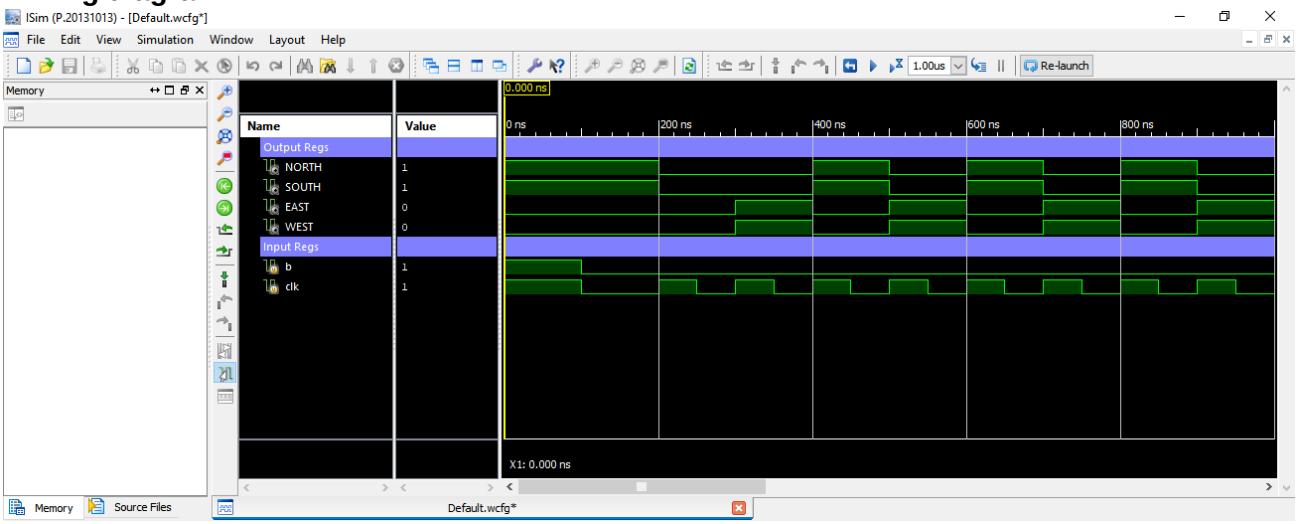
```
assign nextq = (same) ? cnt_ini : inextq ;
endmodule
//##############################################################################


//##############################################################################
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
//##############################################################################
```

Testbench Verilog Code

```
`timescale 1ns / 1ps

module fourway_traffic_tb;

	// Inputs
	reg b;
	reg clk;

	// Outputs
	wire NORTH;
	wire SOUTH;
	wire EAST;
	wire WEST;

	// Instantiate the Unit Under Test (UUT)
	fourway_traffic_bh uut (
		.b(b), .clk(clk),
		.NORTH(NORTH), .SOUTH(SOUTH),
		.EAST(EAST), .WEST(WEST)
```

```
	);

	initial
		begin
		// Initialize Inputs
		b = 1'b1;
		clk = 1'b1;
		#50;

		b = 1'b1;
		clk = 1'b0;
		#50;

		b = 1'b0;
		clk = 1'b1;
		#50;

		b = 1'b0;
		clk = 1'b0;
		#50;
		end

	always #50 clk = ~clk;
endmodule
```

Timing diagram



Truth Table

	Inputs					Outputs							
	S1	S0	b	timer3	timer5	n1	n0	reset3	reset5	North	South	East	West
NSLANE	0	0	0	0	0	0	0	0	1	1	1	0	0
	0	0	0	0	1	0	1	0	1	1	1	0	0
	0	0	0	1	0	0	0	0	1	1	1	0	0
	0	0	0	1	1	0	1	0	1	1	1	0	0
	0	0	1	0	0	0	0	0	1	1	1	0	0
	0	0	1	0	1	0	0	0	1	1	1	0	0
	0	0	1	1	0	1	0	0	1	1	1	0	0
	0	0	1	1	1	1	0	0	1	1	1	0	0
EWLANE	0	1	0	0	0	0	1	0	1	0	0	1	1
	0	1	0	0	1	0	1	0	1	0	0	1	1
	0	1	0	1	0	0	0	0	1	0	0	1	1
	0	1	0	1	1	0	0	0	1	0	0	1	1
	0	1	1	0	0	0	1	0	1	0	0	1	1
	0	1	1	0	1	0	1	0	1	0	0	1	1
	0	1	1	1	0	1	0	0	1	0	0	1	1
	0	1	1	1	1	1	0	0	1	0	0	1	1
WALK	1	0	0	0	0	1	0	1	0	0	0	0	0
	1	0	0	0	1	0	1	1	0	0	0	0	0
	1	0	0	1	0	1	0	1	0	0	0	0	0
	1	0	0	1	1	0	1	1	0	0	0	0	0
	1	0	1	0	0	1	0	1	0	0	0	0	0
	1	0	1	0	1	0	1	1	0	0	0	0	0
	1	0	1	1	0	1	0	1	0	0	0	0	0
	1	0	1	1	1	0	1	1	0	0	0	0	0

- Link to youTube video showing your board's performance

3. Detailed description of the problems and technical issues encountered especially the ones that resulted in system re-design and/or modifications

There were no technical issues that came about when the Basys 2 board were involved. Most of the problems occurred in the Verilog code itself, but we were just missing several parameters and registers when generating the bit files to show that the LEDs on the board were outputting the alternating lights in our FSM logic as well as the transition to the crosswalk. The testbench on the other hand, I felt very dumbfounded when writing the code. Since the behavioral module was reliant on the button press and the clock, the initialized inputs to simulate the timing diagram seemed so easy. Initially, I overthought the coding process because there were a couple of registers to consider in the Verilog code. It was not until I attempted to simulate the phases when the button and clock were on and off Is when I started to notice the testbench being very simple.

4. Conclusions

Overall, by using the material and knowledge I have gained from the past labs. The four-way traffic controller follows the provided specifications. I do not recall in my implementations any exceptions or special cases where I came across a problem that relied on a roundabout method of working the application on the Basys 2 boards. If I were to hypothetically improve the design of this traffic light, then I would connect the FPGA board onto a separate breadboard and hook up with two red and green LEDs to simulate the stop and go. Alternatively, what I could have done in my original work was using seven segment LED display. I could have let the LEDs display the word stop or go and have shown that as well in the demo.