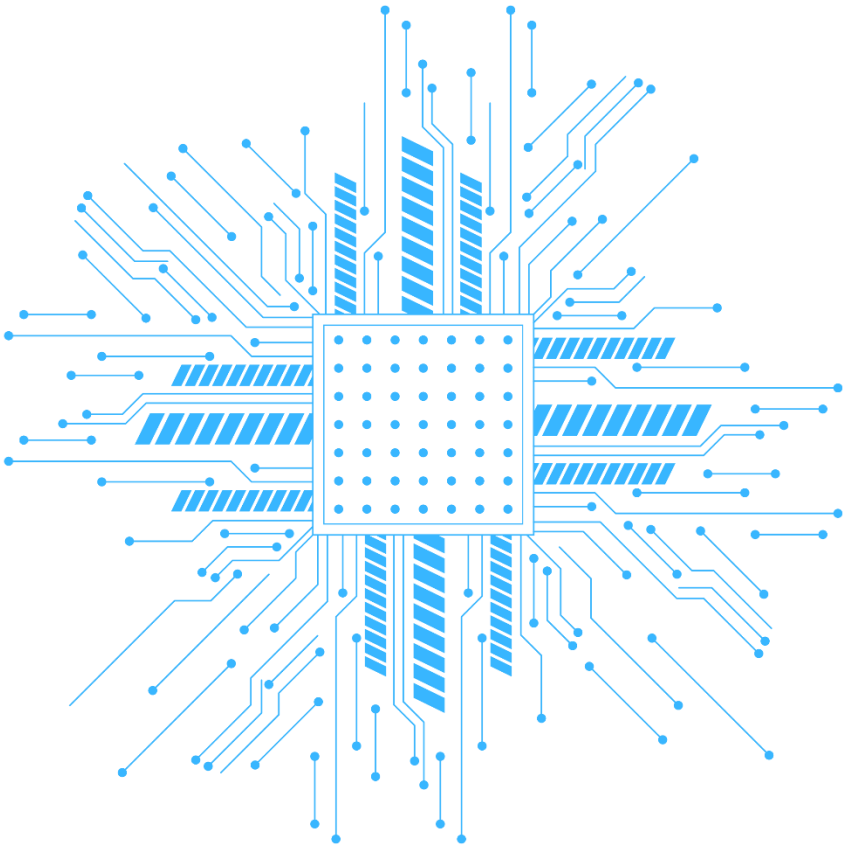
Designing and Testing a Traffic Controller with a Timer

Represented by: Mahmous Ismail Mahmoud

Using Modelsim and Vivado

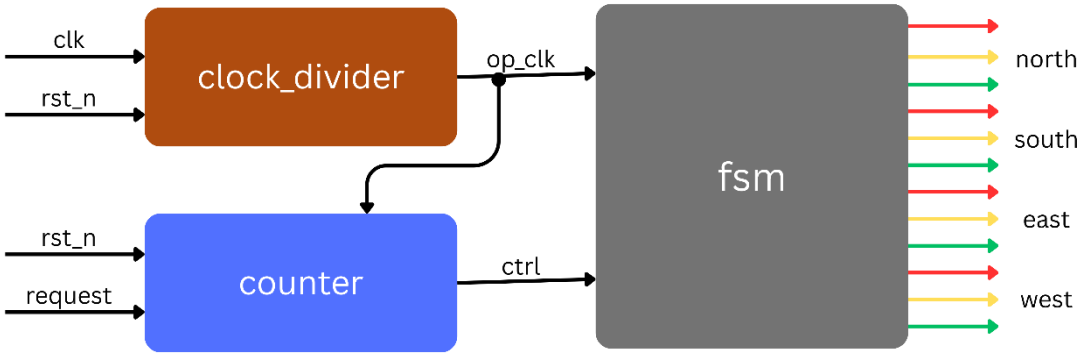


System Specification:

The system manages the operation of traffic signals at a four-way intersection. Each direction has three lights (Green, Yellow, Red), and the system cycles through them with programmable timing intervals, 10 seconds for green, and 3 seconds for yellow. It also offers an option for pedestrian requests that extends the green phase by 5 seconds.

System Design:

The system consists of three main blocks.



Clock divider:

Makes the clock period of the system equal to 1 second, instead of 10 nanoseconds.

Counter:

The system's timer, using **op_clk** (a clock with a period of 1 second), counts up to 18 seconds in the maximum case. Its main functionality is controlling the finite state machine, with the signal **ctrl.** With the **request** signal (which represents the pedestrian request), the counter works in two case:

1. Pedestrian request = 0: counts up to 12 divided by: 0 to 9 ctrl = 1, 10 to 12 ctrl = 0.
2. Pedestrian request = 1: counts up to 12 divided by: 0 to 14 ctrl = 1, 15 to 17 ctrl = 0.

FSM:

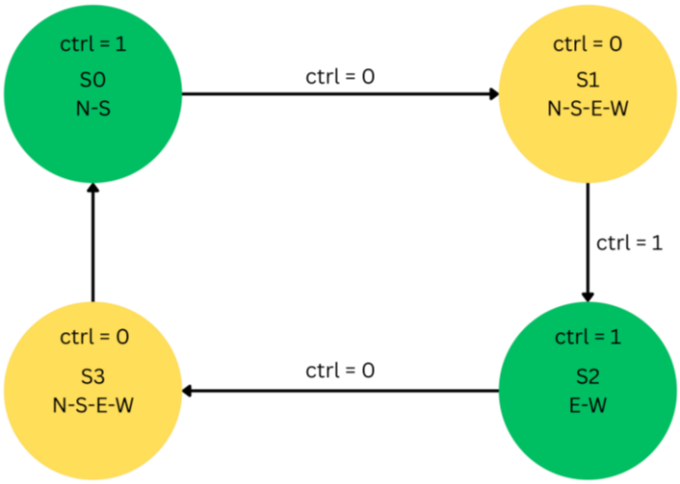
Controls the lights of each way, using the **ctrl** signal from the counter. States using mealy:

S0: North-south green west-east red.

S1: All is yellow for 3 seconds.

S2: West-east green north-south red.

S4: All is yellow for 3 seconds.



RTL Codes:

Clock divider:

```
module clock_divider (
    input clk,  // system clock
    input rst_n,  // system reset (asynchronous active-low reset)
    output reg op_clk  // operation clock
);

reg [26: 0] i;  // counter

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        i <= 27'b0;
    else
        i <= i + 27'b1;
end

always @ (*) begin
    if (!rst_n)
        op_clk = clk;
    else
        op_clk = ~i[26];
end
endmodule
```

Counter:

```
module counter (
    input clk,
    input rst_n,
    input request,
    output reg ctrl
);

reg [4:0] counts;
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n )
        counts <= 5'b0;
    else if (request) begin
        if (counts == 5'b10001)
            counts <= 5'b0;
        else
            counts <= counts + 1'b1;
    end
    else begin
        if (counts == 5'b01100)
            counts <= 5'b0;
        else
            counts <= counts + 1'b1;
    end
end

always @ (*) begin
    if (request) begin
        ctrl = ~(counts[4] | &counts[3:0]);
    end
    else begin
        ctrl = ~(&counts[3:2] | (counts[3] & counts[1]));
    end
end
endmodule
```

FSM:

```
module fsm (
    input ctrl,
    input clk,
    input rst_n,
    output reg north_green,
    output reg north_red,
    output reg north_yellow,
    output reg south_green,
    output reg south_red,
    output reg south_yellow,
    output reg east_green,
    output reg east_red,
    output reg east_yellow,
    output reg west_green,
    output reg west_red,
    output reg west_yellow
);

localparam s0 = 2'b0, s1 = 2'b1, s2 = 2'b10, s3 = 2'b11;
reg [1:0] cs, ns;

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        cs <= s0;
    else
        cs <= ns;
end

always @ (*) begin
    north_green = 1'b0;
    north_red = 1'b0;
    north_yellow = 1'b0;
    south_green = 1'b0;
    south_red = 1'b0;
    south_yellow = 1'b0;
    east_green = 1'b0;
    east_red = 1'b0;
    east_yellow = 1'b0;
    west_green = 1'b0;
    west_red = 1'b0;
    west_yellow = 1'b0;
```

```
    case (cs)
        s0: begin
            north_green = 1'b1;
            south_green = 1'b1;
            east_red = 1'b1;
            west_red = 1'b1;
            if (ctrl)
                ns = s0;
            else
                ns = s1;
        end
        s1: begin
            north_yellow = 1'b1;
            south_yellow = 1'b1;
            west_yellow = 1'b1;
            east_yellow = 1'b1;
            if (ctrl)
                ns = s2;
            else
                ns = s1;
        end
        s2: begin
            east_green = 1'b1;
            west_green = 1'b1;
            north_red = 1'b1;
            south_red = 1'b1;
            if (ctrl)
                ns = s2;
            else
                ns = s3;
        end
        s3: begin
            north_yellow = 1'b1;
            south_yellow = 1'b1;
            west_yellow = 1'b1;
            east_yellow = 1'b1;
            if (ctrl)
                ns = s0;
            else
                ns = s3;
        end
    endcase
end
endmodule
```

Top:

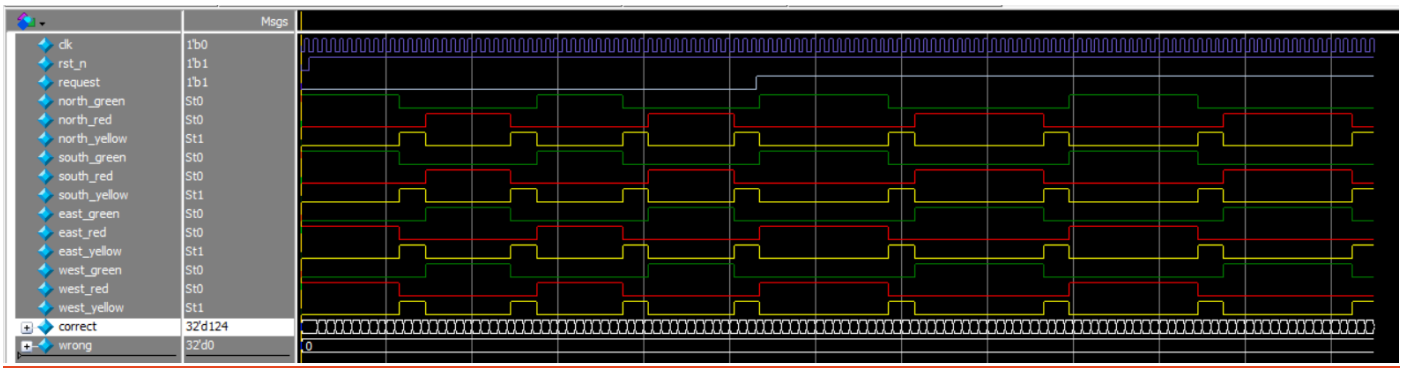
```
module top (
    input clk,
    input rst_n,
    input request,
    output _north_green_,
    output _north_red_,
    output _north_yellow_,
    output _south_green_,
    output _south_red_,
    output _south_yellow_,
    output _east_green_,
    output _east_red_,
    output _east_yellow_,
    output _west_green_,
    output _west_red_,
    output _west_yellow_
);

wire [4:0] counts;
wire ctrl;
wire op_clk;

clock_divider clk_div (
    .clk(clk),
    .op_clk(op_clk),
    .rst_n(rst_n)
);

counter counter_block (
    .clk(op_clk),
    .rst_n(rst_n),
    .request(request),
    .ctrl(ctrl),
    .counts(counts)
);

fsm fsm_block (
    .ctrl(ctrl),
    .clk(op_clk),
    .rst_n(rst_n),
    .north_green(_north_green_),
    .north_red(_north_red_),
    .north_yellow(_north_yellow_),
    .south_green(_south_green_),
    .south_red(_south_red_),
    .south_yellow(_south_yellow_),
    .east_green(_east_green_),
    .east_red(_east_red_),
    .east_yellow(_east_yellow_),
    .west_green(_west_green_),
    .west_red(_west_red_),
    .west_yellow(_west_yellow_)
);
endmodule
```

Testbench:

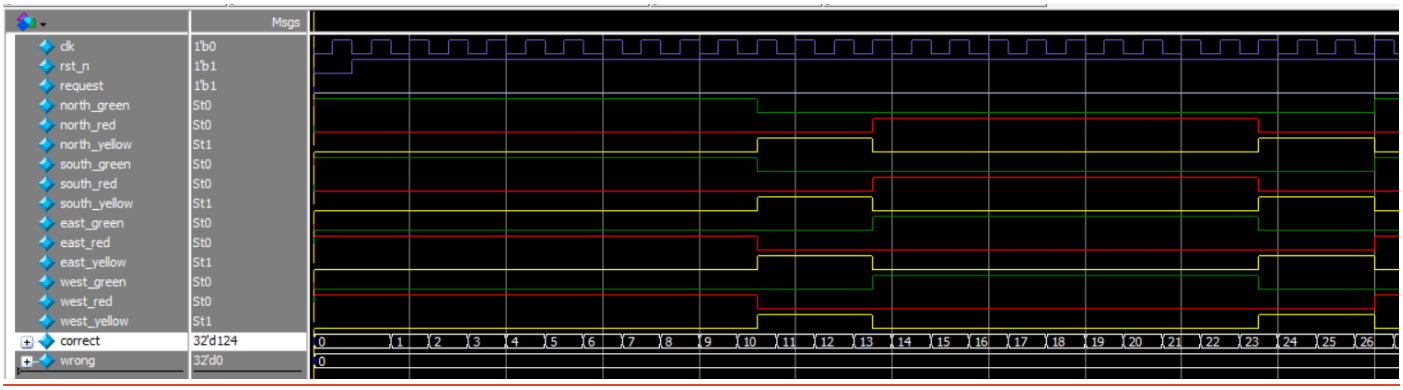
```
module testbench ();

reg clk;
reg rst_n;
reg request;
wire north_green;
wire north_red;
wire north_yellow;
wire south_green;
wire south_red;
wire south_yellow;
wire east_green;
wire east_red;
wire east_yellow;
wire west_green;
wire west_red;
wire west_yellow;

integer correct, wrong;

top DUT (.*);

initial begin
    clk = 0;
    forever
    #10 clk = ~clk;
end

task delay_10_cycles_NS;  // checks for north-south state.
    repeat (10) begin
        @(negedge clk);
        if (!(north_green && south_green && east_red && west_red)) begin
            $display ("Error in north-south state");
            wrong = wrong + 1;
        end
        else
            correct = correct + 1;
    end

endtask

task delay_10_cycles_EW;  // checks for east-west state.
    repeat (10) begin
        @(negedge clk);
        if (!(east_green && west_green && north_red && south_red)) begin
            $display ("Error in east-west state");
            wrong = wrong + 1;
        end
        else begin
            correct = correct + 1;
        end
    end
endtask

task delay_3_cycles;  // checks for waiting state.
    repeat (3) begin
        @(negedge clk);
        if (!(north_yellow && south_yellow && east_yellow && west_yellow)) begin
            $display ("Error in waiting state");
            wrong = wrong + 1;
        end
        else begin
            correct = correct + 1;
        end
    end
endtask

task delay_15_cycles_NS;  // checks for north-south state.
    repeat (15) begin
        @(negedge clk);
        if (!(north_green && south_green && east_red && west_red)) begin
            $display ("Error in north-south state");
            wrong = wrong + 1;
        end
        else begin
            correct = correct + 1;
        end
    end
endtask

task delay_15_cycles_EW;  // checks for east-west state.
    repeat (15) begin
        @(negedge clk);
        if (!(east_green && west_green && north_red && south_red)) begin
            $display ("Error in east-west state");
            wrong = wrong + 1;
        end
        else begin
            correct = correct + 1;
        end
    end
endtask

initial begin
    rst_n = 0;
    correct = 0;
    wrong = 0;
    request = 0;
    @(negedge clk);
    rst_n = 1;
    delay_10_cycles_NS();
    delay_3_cycles();
    delay_10_cycles_EW();
    delay_3_cycles();
    delay_10_cycles_NS();
    delay_3_cycles();
    delay_10_cycles_EW();
    delay_3_cycles();
    request = 1;
    delay_15_cycles_NS();
    delay_3_cycles();
    delay_15_cycles_EW();
    delay_3_cycles();
    delay_15_cycles_NS();
    delay_3_cycles();
    delay_15_cycles_EW();
    delay_3_cycles();
    $display("Correct checks : %d , wrong checks : %d" , correct , wrong);
    $stop;
end
endmodule
```

Results from waveform:

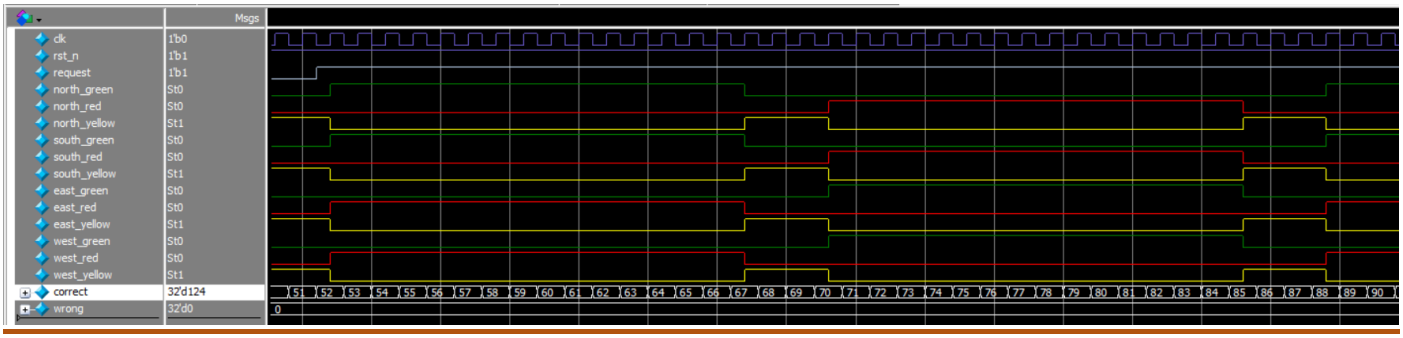
Full wavefrom:



Waveform for full cycle with pedestrian request = 0:



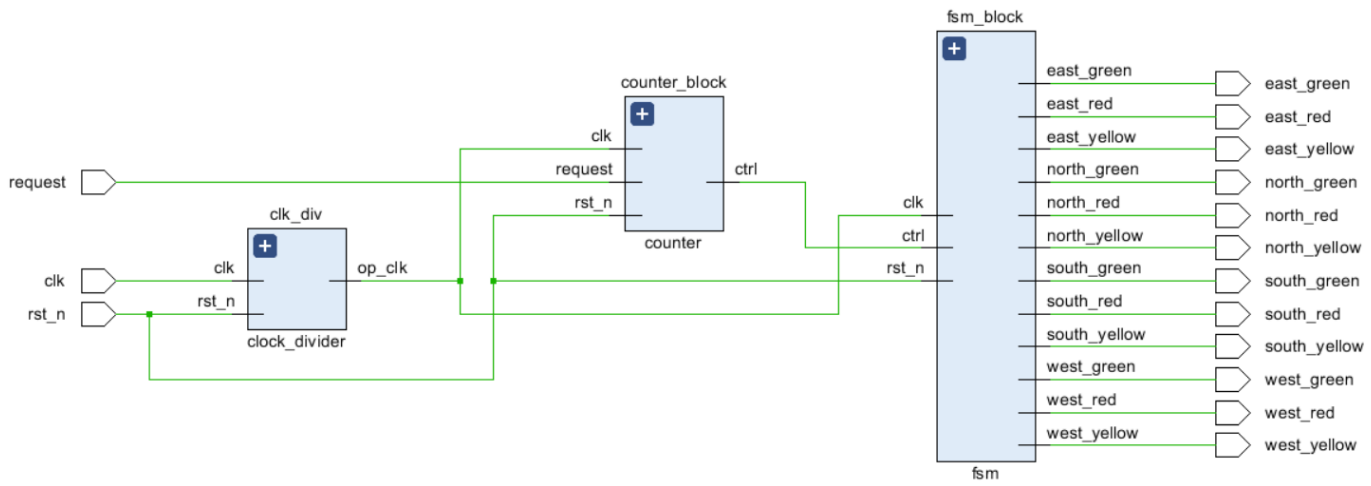
Waveform for full cycle with pedestrian request = 1:



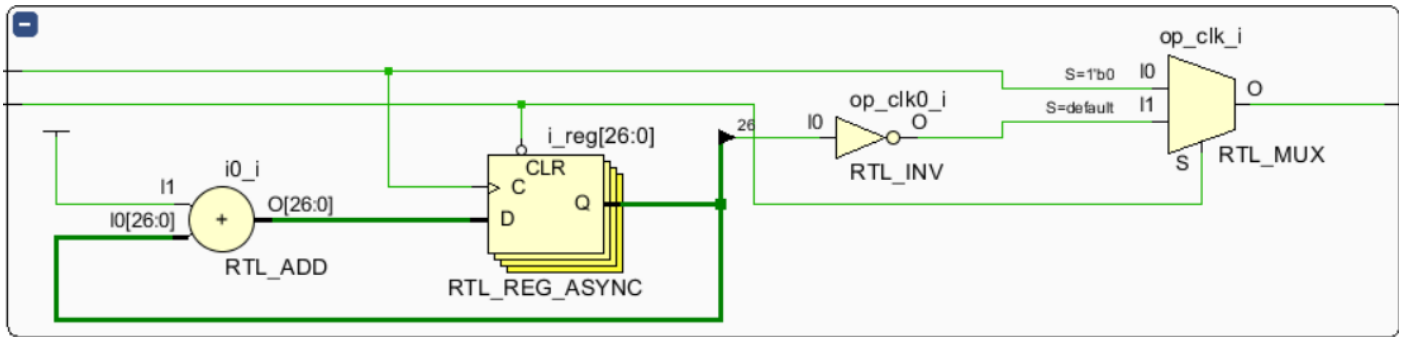
Transcript after testbench:

```
# Top level modules:
#       testbench
# End time: 06:29:39 on Sep 05,2025, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" work.testbench -coverage
# Start time: 06:29:40 on Sep 05,2025
# ** Note: (vsim-3812) Design is being optimized...
# ** Note: (vopt-143) Recognized 1 FSM in module "fsm(fast)".
# Loading sv_std.std
# Loading work.testbench(fast)
# Loading work.top(fast)
# Loading work.counter(fast)
# Loading work.fsm(fast)
# Correct checks :          124 , wrong checks :          0
# ** Note: $stop    : testbench.sv(119)
#    Time: 2500 ps  Iteration: 1  Instance: /testbench
# Break in Module testbench at testbench.sv line 119
```

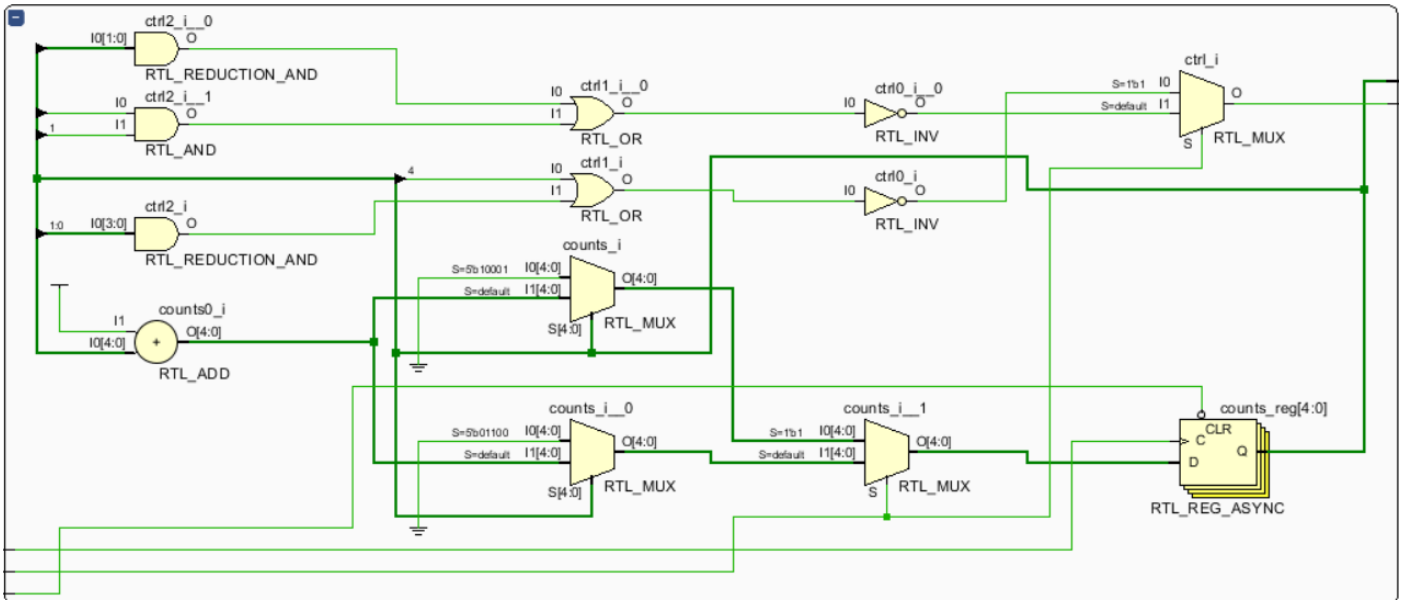
Block-level schematic:



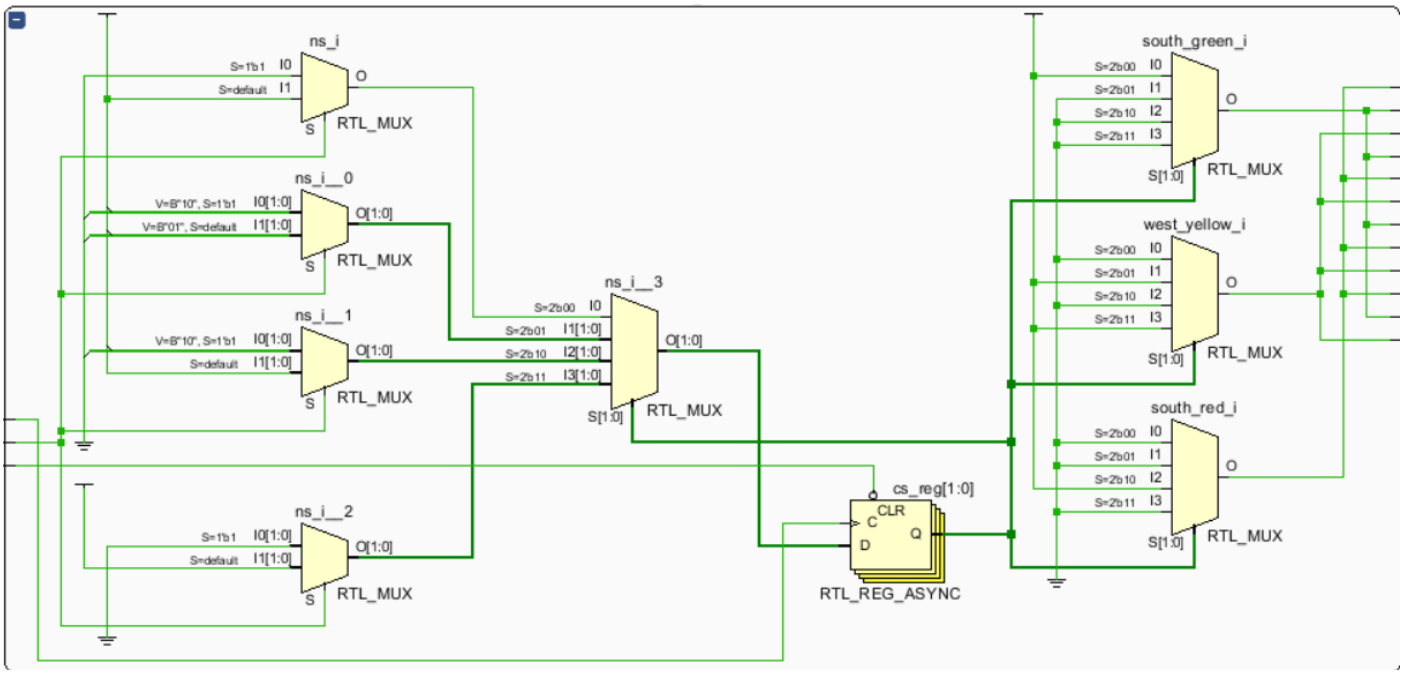
Clock divider schematic:



Counter schematic:



FSM schematic:



Synthesis timing report:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.986 ns	Worst Hold Slack (WHS):	0.131 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	34	Total Number of Endpoints:	34	Total Number of Endpoints:	35

All user specified timing constraints are met.

Synthesis power report:

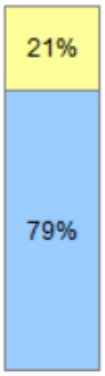
Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.078 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	74.6°C (14.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

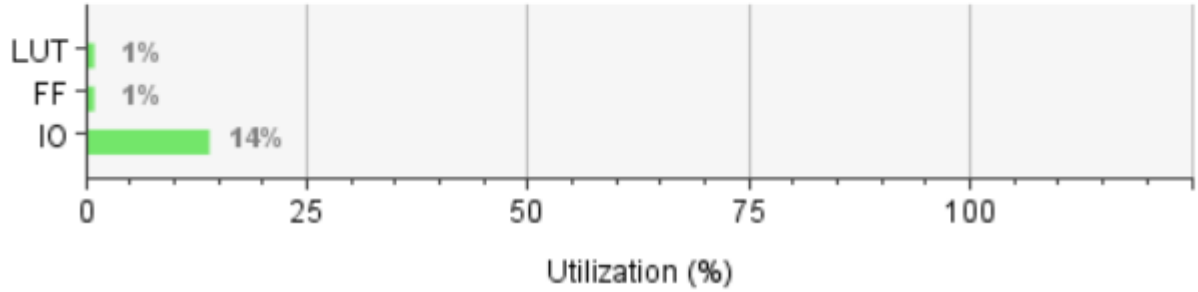


Dynamic:	0.016 W	(21%)
Clocks:	0.001 W	(6%)
Signals:	<0.001 W	(1%)
Logic:	<0.001 W	(1%)
I/O:	0.015 W	(92%)
Device Static:	0.062 W	(79%)

Synthesis utilization report:

Summary

Resource	Utilization	Available	Utilization %
LUT	13	20800	0.06
FF	34	41600	0.08
IO	15	106	14.15



Implementation timing report:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.988 ns	Worst Hold Slack (WHS):	0.033 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3824	Total Number of Endpoints:	3808	Total Number of Endpoints:	2139

All user specified timing constraints are met.

Implementation power report:

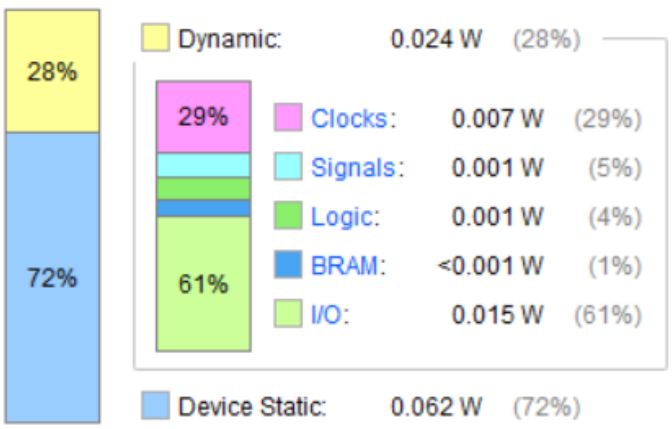
Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.086 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	74.6°C (14.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

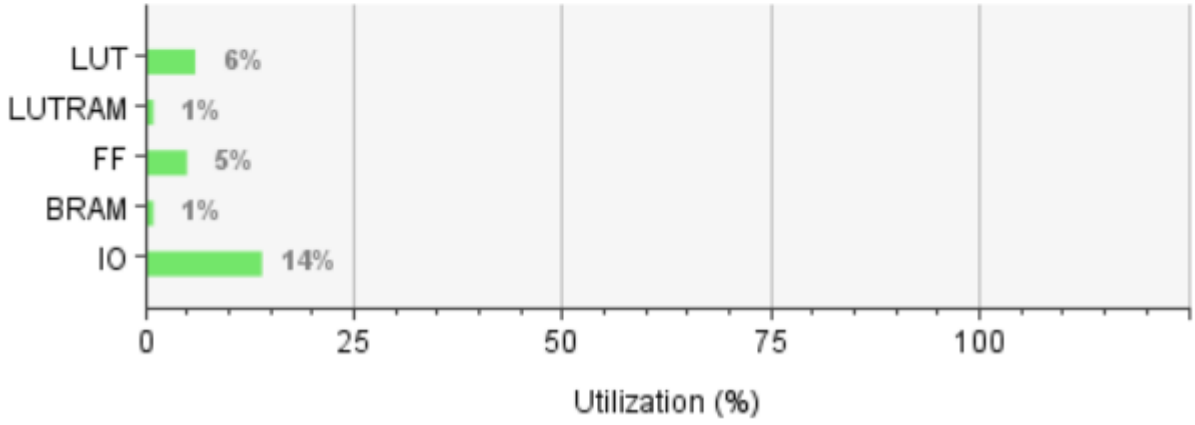
On-Chip Power



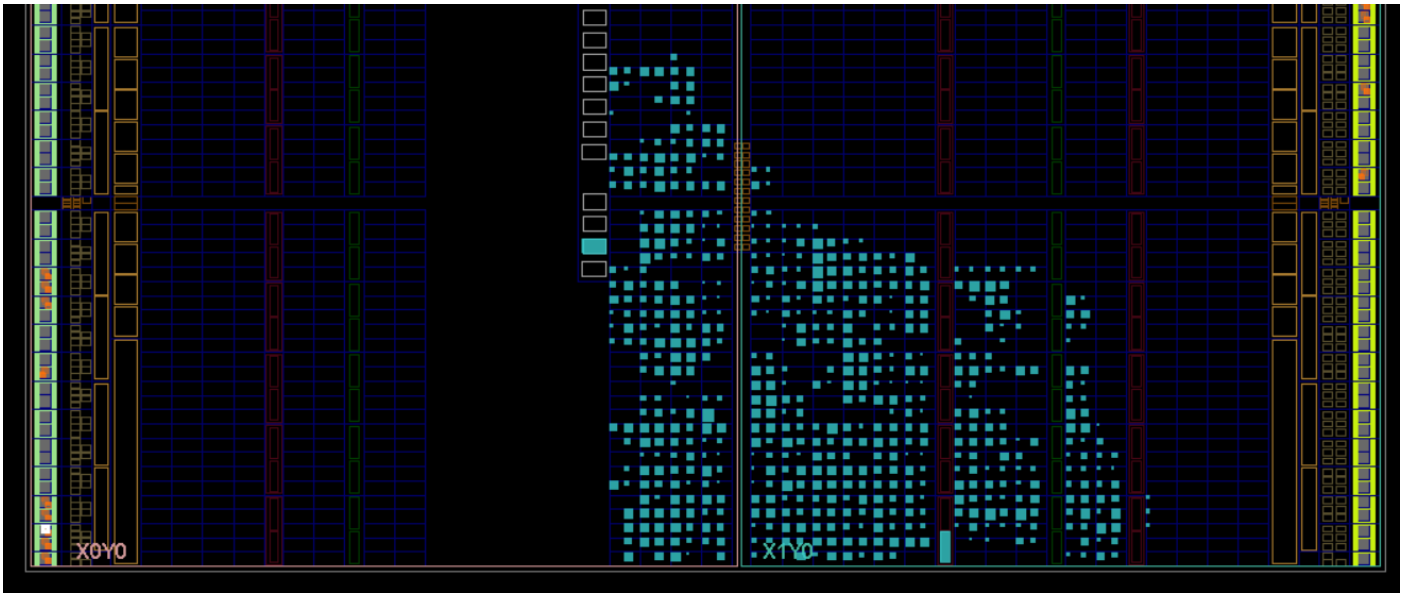
Implementation utilization report:

Summary

Resource	Utilization	Available	Utilization %
LUT	1250	20800	6.01
LUTRAM	108	9600	1.13
FF	1954	41600	4.70
BRAM	0.50	50	1.00
IO	15	106	14.15



Implementation:



GitHub Repository:

https://github.com/MahmoudIsmail47/Traffic-controller-with-a-timer