Description

In this lab, we will implement a chain of flip-flops in order to make a shift register. The shift register will use the clock divider circuit from Lab 7 with just slight modifications. We will then use this shift register to shift the 16 LED lights on the Basys3 Board from right to left and back and forth at the frequency of 2.98Hz. In part 2, we will then implement another shift register, in order to shift the LED lights back and forth in different patterns based on one of the three push button inputs on the Basys3 Board.

Clock Divider

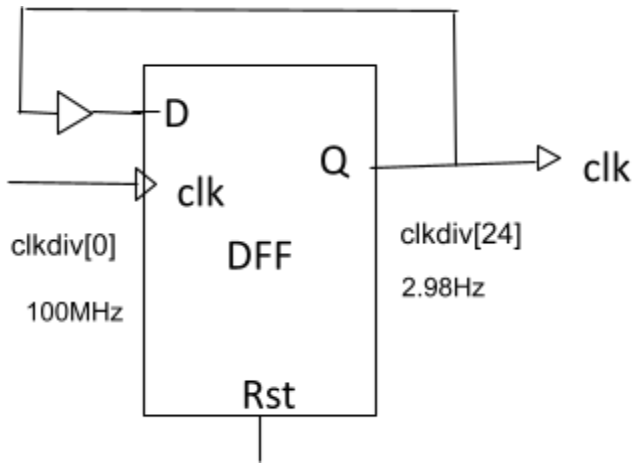

Figure 1.1: Black box diagram of the Clock Divider showing input and output ports.

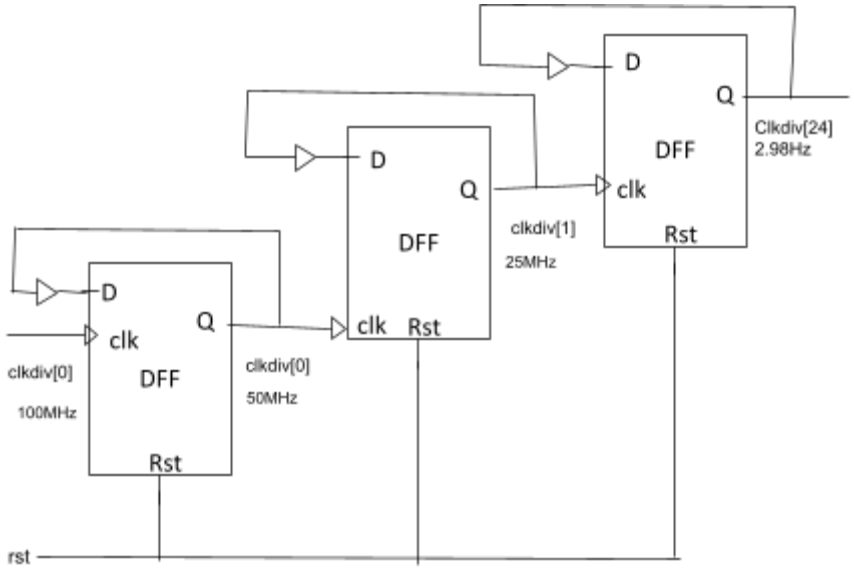

Figure 1.2: Structural Model for the clock divider. Shows the consecutive D flip-flops.

Shift Register

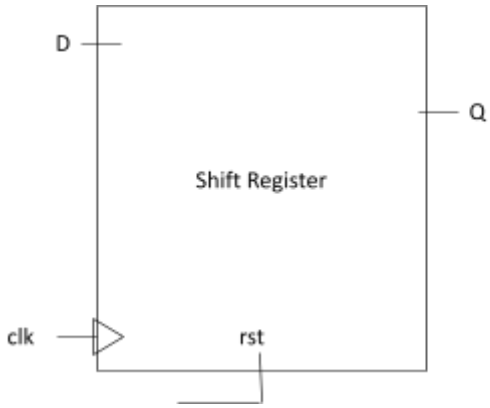

Figure 1.3: Black Box diagram for the Shift Register showing the inputs and outputs.

LED Shifter (Lab 8 Top-Level)

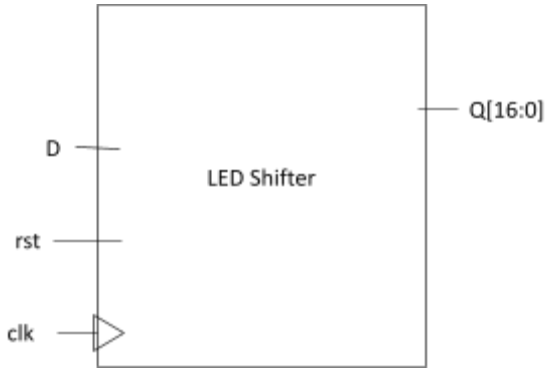

Figure 1.3: Black Box diagram for the LED Shifter showing the inputs and outputs.

Circuit Schematics

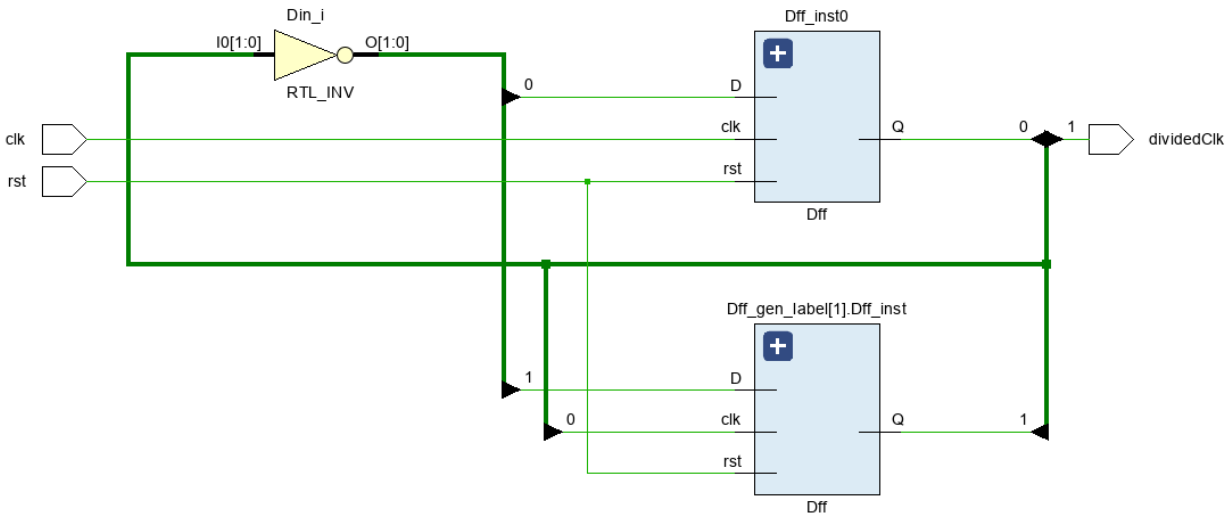

Figure 1.4: Screenshot of the Clock Divider circuit schematic after running the synthesis

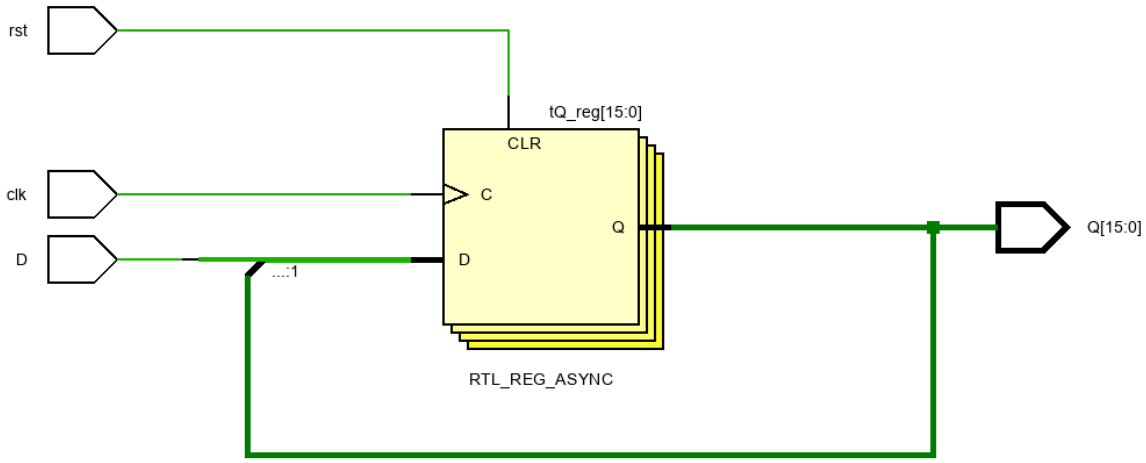

Figure 1.5: Screenshot of the Shift Register circuit schematic obtained from the synthesis

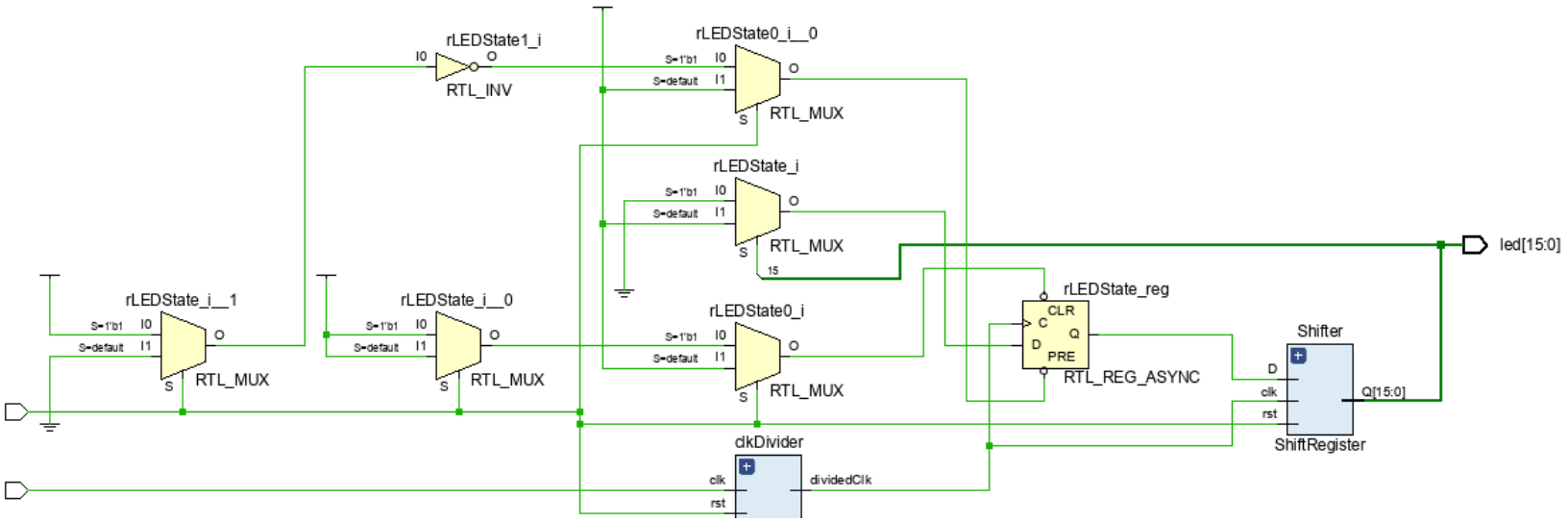

Figure 1.6: Screenshot of the Lab 8 Top-Level (LED Shifter) circuit schematic obtained from the synthesis

Simulation Results

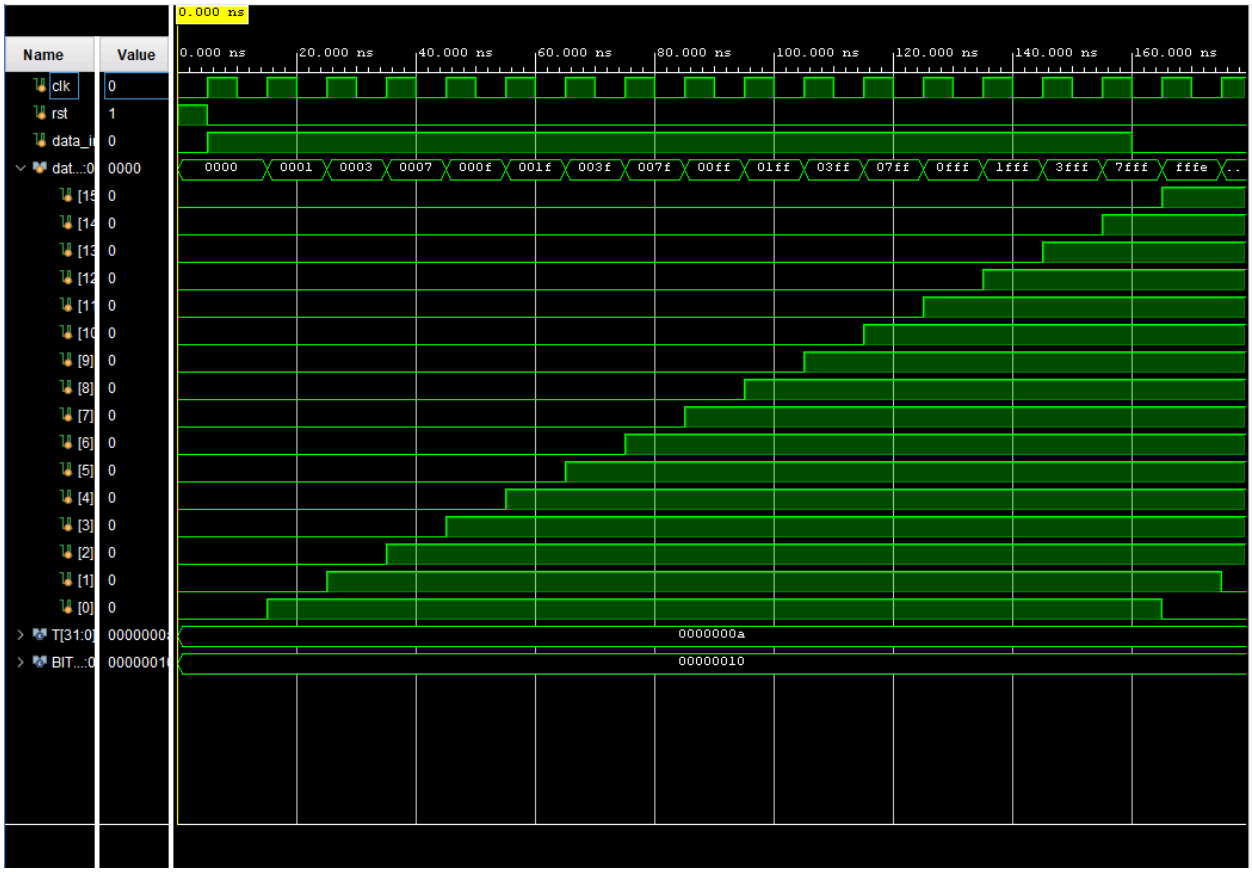

Figure 1.7: Screenshot of the timing diagram of the shift register capturing all test cases. Shows how the data_in values propagate through the shift register over time.

The simulation results as expected. The shift register works as it should and produces the expected output values for the given input combinations.

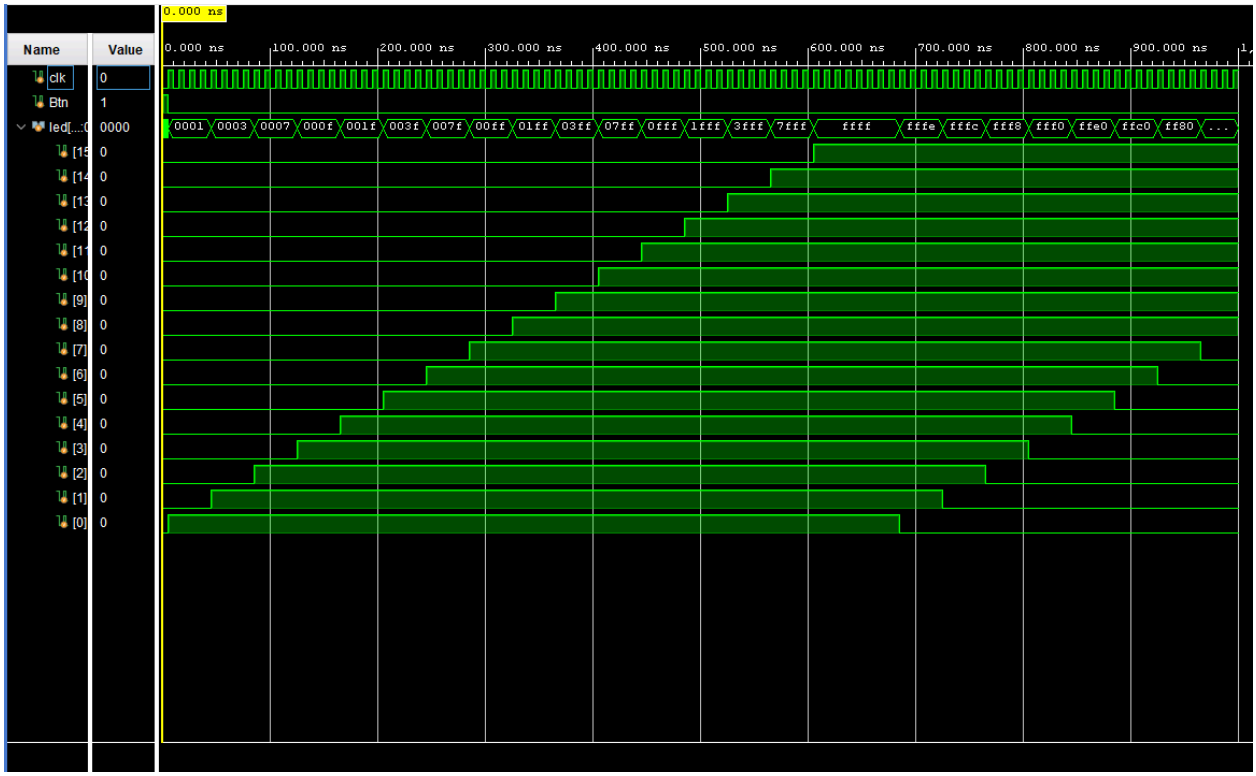

Figure 1.8: Screenshot of the timing diagram of the top level module capturing all test cases. Shows how the data_in values propagate through the shift register and LEDS over time.

Simulation worked as expected and shows the values propagating in a pattern. It looks very similar to the shift register simulation, and this is because it is essentially doing the same thing, however it takes in a "Btn" input which determines how the data_in values propagate.

Source Code

Clock Divider Design Code:

```
module clk_divider #(parameter Divisions = 1)(
   input clk,
   input rst,
   output dividedClk
   );
   logic [Divisions:0] Din;
   logic [Divisions:0] clkdiv;
   Dff Dff_inst0 (.D(Din[0]), .Q(clkdiv[0]), .rst(rst), .clk(clk));
   genvar i;
   generate
   for (i = 1; i < Divisions+1; i = i+1)
      begin : Dff_gen_label
      Dff Dff_inst (.clk(clkdiv[i-1]), .rst(rst), .D(Din[i]), .Q(clkdiv[i]));
   end
   endgenerate;
   assign Din = ~ clkdiv;
   assign dividedClk = clkdiv[Divisions];
endmodule
```

Shift Register Design Code:

```
module ShiftRegister #(parameter BITS = 16)(
input clk,
input rst,
input D,
output [BITS - 1:0] Q
);
//internal signals
logic [BITS-1:0] tQ;
logic [BITS-1:0] next_output;
//Circuit operates on rising edge of clk and synchronous rst
always_ff @ (posedge clk, posedge rst)
if (rst)
tQ <= 0;
else
tQ <= next_output;
//Update the shift register contents by shifting left
assign next_output = {tQ[BITS - 2:0], D};
```

```
//Update the shift register output
assign Q = tQ;
endmodule
```

LED Shifter Design Code:

```
module LEDShifter #(parameter LED_BITS = 16)
(
input clk,
input Btn,
output [LED_BITS - 1:0] led
);
logic dividedClk;
logic [LED_BITS - 1:0] wled;
logic rLEDState = 1;
clk_divider #(24) clkDivider (.clk(clk), .rst(Btn),
.dividedClk(dividedClk));
ShiftRegister Shifter (.clk(dividedClk), .rst(Btn), .D(rLEDState),
.Q(wled));
always_ff @ (posedge dividedClk, posedge Btn)
if (Btn)
rLEDState = 1'b1;
else if (wled[LED_BITS - 1] == 1'b1)
rLEDState <= 1'b0;
else
rLEDState <= 1'b1;
assign led = wled;
endmodule
```