LABS 5 AND 6

UART Transmitter and Receiver

BY: HAYDEN HUTSELL AND ZAK ROWLAND

DIGITAL SYSTEM DESIGN I

Submitted in partial fulfillment for the requirements of Digital System Design I (CST 231)

Contents

Abstract

In these labs, the Xilinx® Spartan 6™ XC6SLX9 FPGA was used to generate a UART transmitter and receiver that operates at 9,600 baud or 38,400 baud. The main goal for the transmitter was to send a 10-character message to putty, and for the receiver to display a received ascii character from putty as a hex value on the Lumex LDD-E2802RD SSEG (seven segment display). The most challenging part of the lab was timing, as even minor errors can lead to unexpected results. Using test benches to see what the data was doing at specific times proved to be helpful with debugging, however, it proved less useful on the receiver portion as timing was tricky with rounding errors. These labs required the use of state machines which were sketched before coding to provide an outline to work from. This allowed for more efficient coding and debugging. The goal of the labs was achieved.

Introduction

The primary objective of the labs was to create a UART transmitter and receiver using the Spartan 6 FPGA. The data was transferred using a USB to UART converter to/from putty. Developing a 2-option clock divider that can be toggled using a button, and creating a state machine that transmits a 10-character message was the goal of lab 5. The individual character data transmitted included a start bit, a character in binary (8-bits), an even parity bit, and two stop bits. Lab 6 introduced oversampling by multiplying the togglable divider created in Lab 5, which was manipulated to be 8X the original frequencies. This allows for midbit sampling of received data to ensure accuracy. If oversampling is not utilized, clock skew could cause unexpected behavior. A state machine was created for received data transmitted via UART. The data was received and then displayed on the SSEG. These labs introduced the importance of timing when sending or receiving data and strengthened our debugging and testing benching skills.

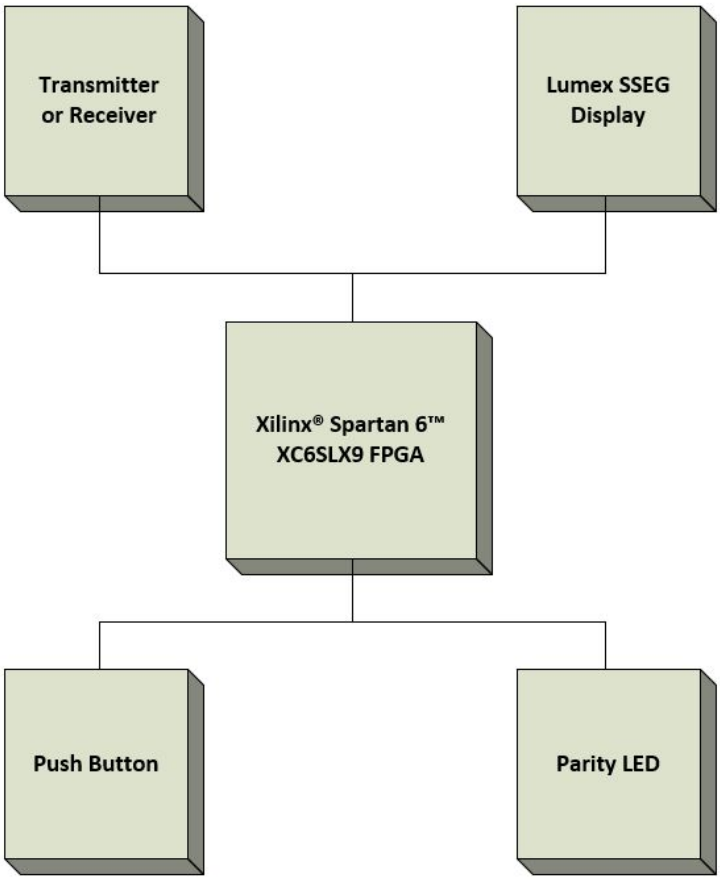

Figure 0: System Block Diagram

Design

The following sections discuss the physical and synthesized hardware design used in this lab.

Physical hardware design

The seven segment provided for this lab was the Lumex LDD-E2802RD. It is configured as common anode. In the data sheet, the recommended voltage is 2.2V. Since the Xilinx FPGA provides 3.3V, and drives a 10 mA current, the following calculation was done to determine the resistors required for the 2.2V cathodes on the seven segment display.

$$\frac{1.1V}{.010\ mA} = 110 = R$$

Figure 1: Seven Segment Resistance Equation

Since 110 ohm resistors were not available, 100 ohm resistors were used. This difference shouldn't noticeably affect any components. After the appropriate resistor was selected, they were wired to the Lumex's pins 1, 2, 3, 4, 5, 7, 8, and 10. Pins 6 and 9 were connected without resistors as the display is common anode. Then, these were wired to Group A's header pins. See figure 3.

Both switches being used were connected to an internal 10K pull-up resistor in the FPGA.

Pin 4 of header group A was used as the serial transmitter or receiver pin.

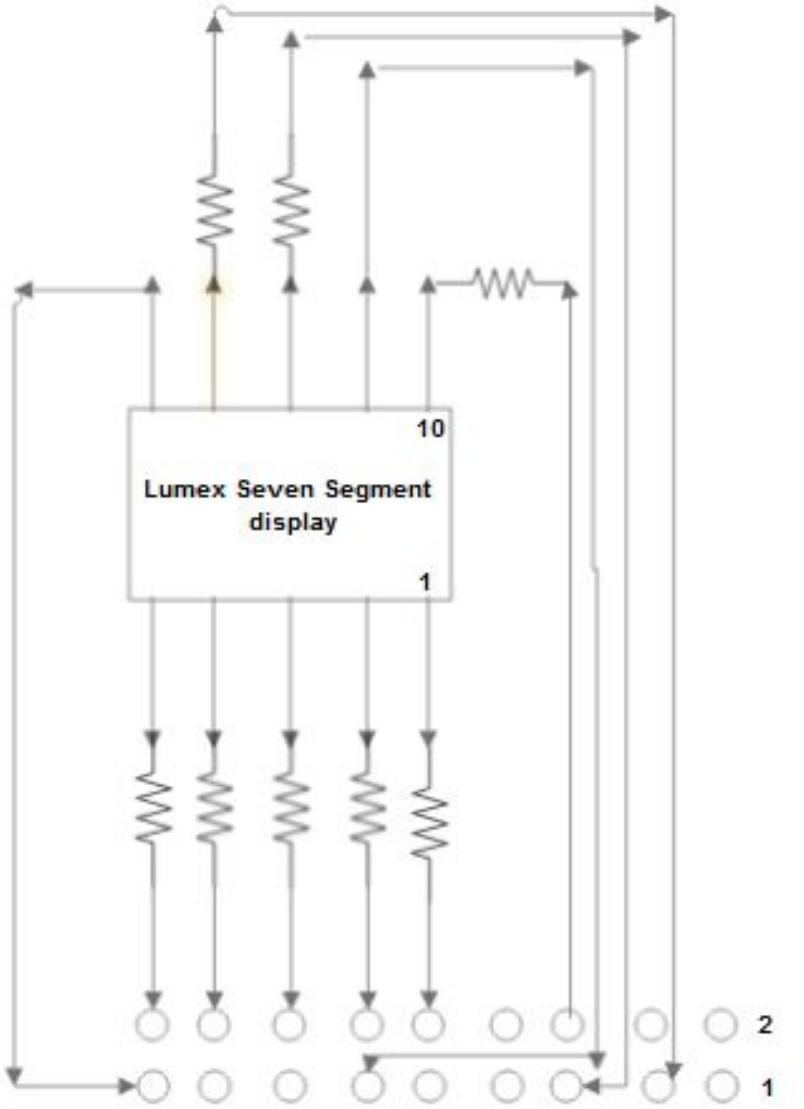

Figure 2: Seven Segment Display Wiring

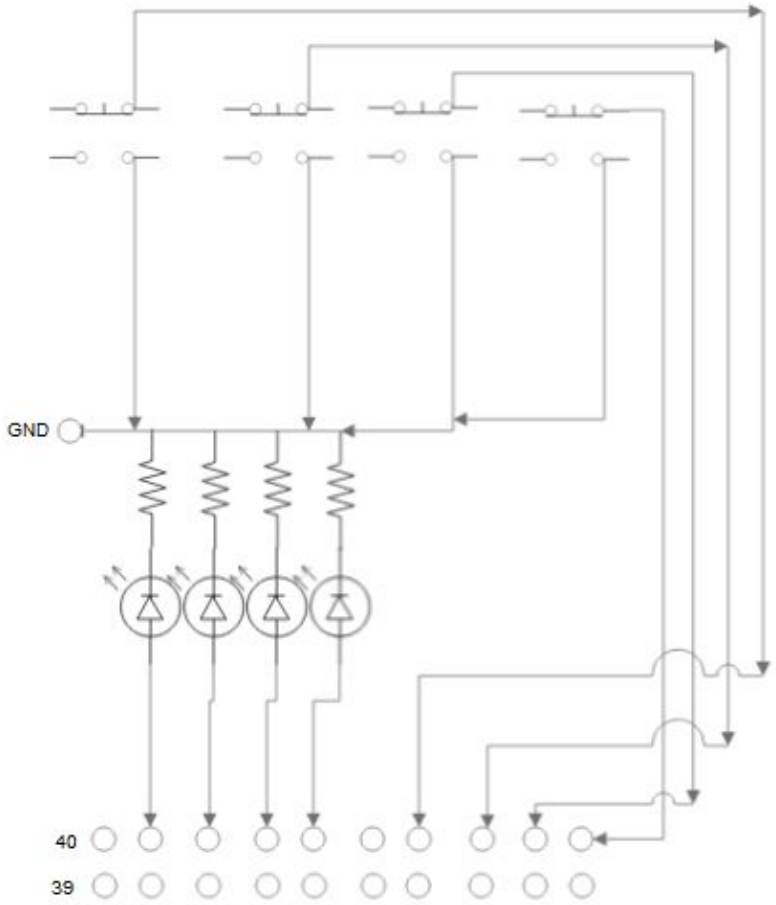

Figure 3: External Button and LED (one of each used) Wiring Diagram

Synthesized hardware design

Design Structure

There are a handful of modules in these labs. Block diagrams are provided in Appendix G, and RTL screenshots are provided in Appendix A.

Lab 5 Modules

This section summarizes the modules used in Lab 5.

top_lab5.v

This module instantiates the three other modules and connects them accordingly. The message to be transmitted is stored in reverse in this module as an 80 bit hex value, which can be easily changed. No other code is used in this module besides instantiating other modules and connecting them.

clkdiv_48mhz_9600or38400hz.v

This module takes the system clock running at 48MHz, converts, and outputs a 9600Hz or 38400Hz clock signal. A push button is used to toggle between the two signals. This was implemented by incrementing a counter by 1, and inverting the output clock signal when the counter reached 2,499 for 9600Hz, or 624 for 38400Hz. It was determined the counter needed to count 2,500 or 625 values from the calculations below in figure 4 and 5. Oscilloscope screenshots are provided in Appendix H.

$$\frac{48MHz}{9600Hz} = 5,000\ , \quad \frac{5,000}{2\,(50\,\%\ \ duty\ cycle)} = 2,500$$

Figure 4: 48MHz to 9600Hz Clock Divider Calculation

$$\frac{48MHz}{38400Hz} = 1,250\ , \quad \frac{1,250}{2\,(50\,\%\ \ duty\ cycle)} = 625$$

Figure 5: 48MHz to 38400Hz Clock Divider Calculation

tx_message.v

This module receives 8 bits of data and outputs a start bit, the received data, two stop bits, and a parity bit. Once a "go" signal is received, the shifting of data begins. The completed UART frame is sent to a receiver, like a terminal program or another FPGA. A state machine is used to accomplish this design, the diagram can be found in Appendix F.

tx_driver.v

This module receives a reversed 10 character message, and upon button press, it outputs a single character at a time to the tx_message module. Once a character is ready, the go signal is set to '1' to inform other modules that the data is ready. There is a 12 cycle delay between character transmission, this is necessary to control the timing. A state machine is used to accomplish this design, the diagram can be found in Appendix F.

Lab 6 Modules

This section summarizes the modules used in Lab 6.

Lab6_top.v

This module instantiates the seven other modules and connects them accordingly. No other code is used in this module besides instantiating other modules and connecting them.

clkdiv_receiver8x_76800or307200hz.v

This module takes the system clock running at 48MHz, converts, and outputs a 76,800Hz or 307,200Hz clock signal. A push button is used to toggle between the two signals. These frequencies were chosen by multiplying the original frequencies by 8 (oversampling.) This was implemented by incrementing a counter by 1, and inverting the output clock signal when the counter reached 311 for 76,800Hz, or 77 for 307200Hz. It was determined the counter needed to count 312 or 78 values from the calculations below in figure 6 and 7. A fractional count cannot be reached, so the values were rounded down. The difference from rounding should not have an effect on the final result. Oscilloscope screenshots are provided in Appendix H.

$$\frac{48MHz}{76800Hz} = 625 \ , \ \frac{625}{2\,(50\,\%\ duty\ cycle)} = 312.5$$

Figure 6: 48MHz to 76,800Hz Clock Divider Calculation

$$\frac{48MHz}{307.2KHz} = 156.25\ ,\ \frac{156.25}{2\,(50\,\%\ \ duty\ cycle)} = 78.125$$

Figure 7: 48MHz to 307,200Hz Clock Divider Calculation

rx.v

This module receives a UART frame serially from a transmitter that includes a start bit, 8 bits of data, 2 stop bits, and an even parity bit. The frame is parsed to allow the data to be output to a decoder for display. A parity LED is illuminated if the received parity bit is not equal to the calculated parity bit, this indicates corrupt data. This module uses multiple different delay states to control timing of data reception. A state machine is used to accomplish this design, the diagram can be found in Appendix F.

clk_divider_48mhz_to_500hz.v

This module takes the system clock running at 48MHz, converts, and outputs a 500Hz clock signal. This was implemented by incrementing a counter by 1, and inverting the output clock signal when the counter reached 47,999. It was determined the counter needed to count 48,000 values from the calculation below in figure 8.

$$\frac{48MHz}{500Hz} = 96,000\ ,\ \frac{96,000}{2\,(50\,\%\ \ duty\ cycle)} = 48,000$$

Figure 8: 48MHz to 500Hz Clock Divider Calculation

SSEG_output.v

This module is functionally a mux. The two decoded binary coded decimal digits are the inputs, as well as the 500Hz clock. Each clock cycle, the mux's output switches between digits. This, coupled with the anode selector, achieves the multiplex required to display two solid digits.

SSEG_sel.v

Like the previous module, this module is functionally a mux. Each clock cycle, the output (which is the anodes on the SSEG) that is receiving power is swapped. This, coupled with the cathode selector, achieves the multiplex required to display two solid digits.

bcd_decoder.v

There are two binary coded decimal decoders that take the values from the receiver, and each decoder decodes one digit to a value which would display the number correctly on the SSEG display. Both decoders output the decoded digit to the SSEG output module.

Simulation and Testing

Lab 5 was the simpler of the two labs. A state machine was developed to send a 10-character message to a putty terminal. If the button wasn't pressed, the led wouldn't light and it would use a baud rate of 9600hz. The message was hardcoded into a reg and the state machine would send putty the go bit, the character, a parity bit, and a stop bit. Every time the second button was pressed, the message would be sent.

Lab 6 took more debugging time because of the complexities of supersampling at 8x or 16x. If the button is pressed, the led is lit and the receiver supersamples at 16x. A state machine was developed to read a serial UART input from a putty terminal and output the ascii hex character to the SSEG. The state machine implemented delays to sample the 9600 baud rate input from putty mid-bit for improved accuracy. The rx module waits for a go bit, then starts sampling the data mid-bit. Before every bit the receiver samples, it must delay an appropriate amount to hit the mid bit of the serial input.

Problems

There were many technicalities with both parts of the lab. The main issues with the transmitter were small typos and reg sizes not declared to the right size. Another issue was bad logic with the states, like not changing states when it should or going to the wrong one entirely. The biggest problem was why putty was showing it was receiving the incorrect data. Upon analysis, the message was being sent LSB first, and we were also reversing the register with the message in it, so the message was backward. Lab 6 was much more time intensive, as timing was very difficult to debug. The testbench proved useful in dialing in our delay values and verifying it was, in fact, processing the serial input correctly. One bug was sequentially evaluating 2 variables and testing the result, so the test wouldn't use the intended values. Moving this to the output arc of the previous state fixed this issue. Another issue was timing on receiving the parity bit. To correctly receive the parity bit, a delay of 3 cycles less than expected was needed to receive the mid-bit value. After figuring this out, the SSEG correctly displaying the ascii hex character inputted from putty.

Results and Conclusion

In this lab, the main goal was to create a UART transmitter and receiver, with the transmitter sending a 10-character message to putty, and the receiver receiving and displaying the correct ascii character from putty. The goal was achieved. Debugging and planning out the state machines were crucial elements in this lab, as well as previous elements in the SSEG lab. The most difficult aspect was debugging and timing, as test benching can only do so much, and timing was basically guessing and checking for the parity state. Also, it was time consuming finding the small errors within the code that wasn't immediately obvious. The state machines were sketched out before development began, and since previously coded modules were available for use, most of the time spent was on the transmitter and receiver, saving time and debugging. All parts of both labs were completed successfully.

Appendix A

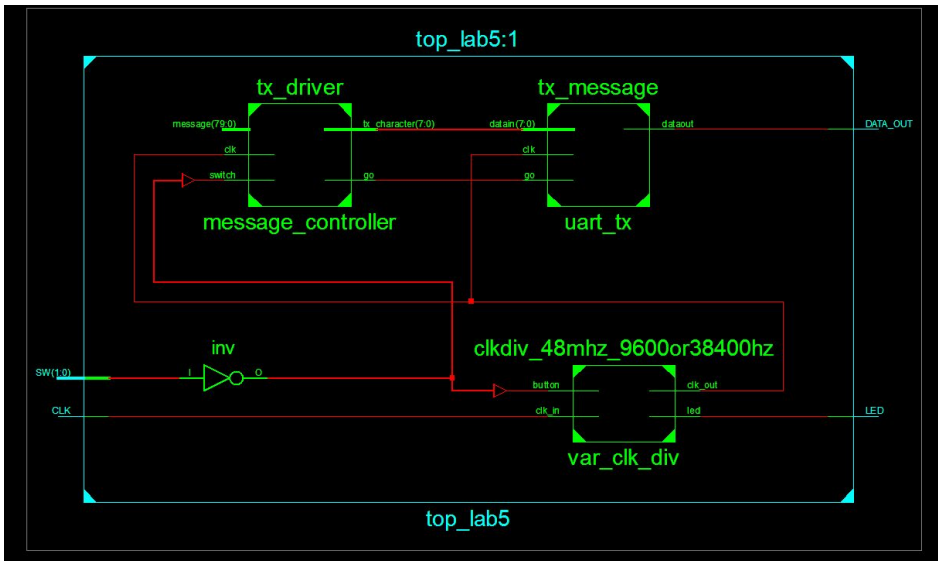

Figure 9: RTL Diagram for top_lab5.v

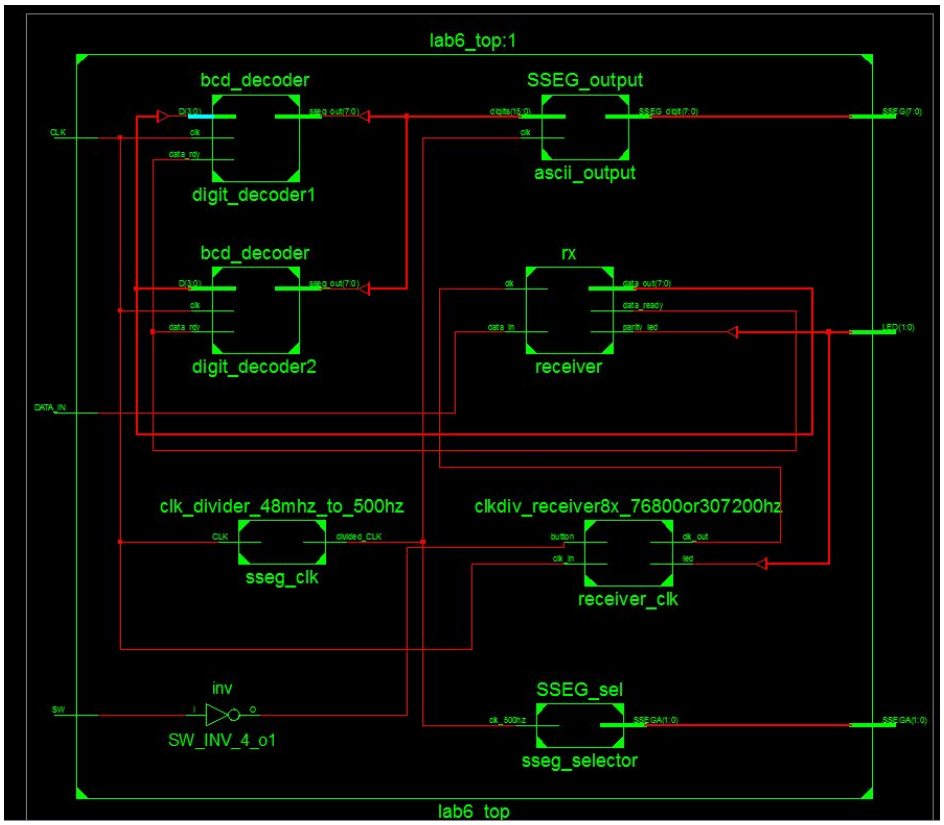

Figure 10: RTL Diagram for lab6_top.v

Appendix B

Component/Pin	Group A Header Pin	FPGA Pin
Onboard Led [0]	-	119
Onboard Led [1]	-	118
Onboard Led [2]	-	117
Onboard Led [3]	-	116
Onboard Led [4]	-	115
Onboard Led [5]	-	114
Onboard Led [6]	-	112
Onboard Led [7]	-	111
Onboard switch [0]	-	124
Onboard switch [1]	-	123
Onboard switch [2]	-	122
Onboard switch [3]	-	120
Blue Led [0]	38	131
Blue Led [1]	36	133
Blue Led [2]	34	137
Blue Led [3]	32	139

Switch [0]	28	1
Switch [1]	26	5
Switch [2]	24	7
Switch [3]	22	9
Seven Segment Pin [1]	10	26
Seven Segment Pin [2]	12	23
Seven Segment Pin [3]	14	21
Seven Segment Pin [4]	16	16
Seven Segment Pin [5]	18	14
Seven Segment Pin [6]	17	15
Seven Segment Pin [7]	3	35
Seven Segment Pin [8]	5	33
Seven Segment Pin [9]	11	24
Seven Segment Pin [10]	6	32

Table 1: Pin Mappings

Appendix C

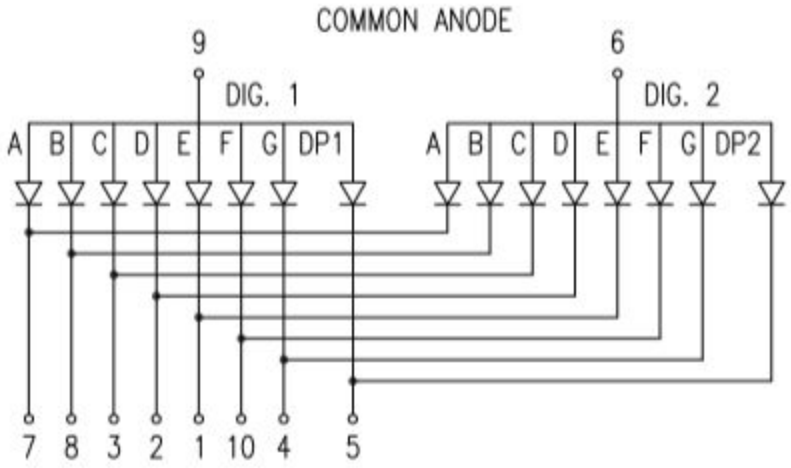

Figure 11: Pin Mapping for Seven Segment Display

Appendix D

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Hayden Hutsell / Zak Rowland
//
// Create Date:    15:24:12 02/07/2019
// Design Name:
// Module Name:    top_lab5
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module top_lab5(
     input CLK,
      input [1:0] SW,
      output DATA_OUT,
      output LED
     );
      wire [7:0] tx_char;
      reg [79:0] transmit_message  = 80'h6e6f6974616e65706176; //"vapenation" reversed
      wire clk_div;
      wire go_tx;



clkdiv_48mhz_9600or38400hz var_clk_div(.clk_in(CLK), .clk_out(DATA_OUT), .button(~SW[0]), .led(LED));
 //variable clock divider depending on if button is pressed
tx_message uart_tx(.clk(clk_div),.go(go_tx), .datain(tx_char), .dataout(DATA_OUT));
 //outputs 8bit data sent from tx_char
tx_driver message_controller(.clk(clk_div), .switch(~SW[1]), .message(transmit_message), .tx_character(tx_char), .go (go_tx));
 //gets fed a message, outputs it one character at a time to tx module
endmodule
```

Figure 12: top_lab5.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Hayden Hutsell and Zak Rowland
//
// Create Date:    15:22:19 02/14/2019
// Design Name:
// Module Name:    lab6_top
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module lab6_top(
    input CLK,
    input SW,
    input DATA_IN,
    output [1:0] LED, //one is on for variable clk, other is on for parity
    output [7:0] SSEG,
    output [1:0]SSEGA  //sseg anode
    );

wire clk_div_uart; //wire for uart
wire clk_div_sseg; //wire for multiplexing sseg
wire [7:0] data_received; //wire to send the serial data to the sseg
wire data_rdy; //data ready flag for sseg
wire [15:0] digit_out; //digits to send to sseg


clkdiv_receiver8x_76800or307200hz receiver_clk(.clk_in(CLK),.button(~SW),.led(LED[0]),.clk_out(clk_div)); //variable clk divider
rx receiver(.clk(clk_div), .data_in(DATA_IN), .parity_led(LED[1]), .data_out(data_received), .data_ready(data_rdy)); //receiver state machine module
clk_divider_48mhz_to_500hz sseg_clk(.CLK(CLK), .divided_CLK(clk_div_sseg)); //clk divider for sseg multiplexing
SSEG_sel sseg_selector(.clk_500hz(clk_div_sseg), .SSEGA(SSEGA));//multiplexes anodes
SSEG_output ascii_output(.clk(clk_div_sseg), .digits(digit_out), .SSEG_digit(SSEG)); //multiplexes cathodes
bcd_decoder digit_decoder1(.D(data_received[3:0]), .sseg_out(digit_out[7:0]), .data_rdy(data_rdy), .clk(CLK)); //decoder for digit 1
bcd_decoder digit_decoder2(.D(data_received[7:4]), .sseg_out(digit_out[15:8]), .data_rdy(data_rdy), .clk(CLK)); //decoder for digit 2

endmodule
```

Figure 13: lab6_top.v

Appendix E

```
NET "CLK" LOC = P126;
TIMESPEC TS_CLK = PERIOD "CLK" 100 MHz HIGH 50%;

NET "SW[0]" LOC = P1;
NET "SW[1]" LOC = P5;
NET "SW[2]" LOC = P7;
NET "SW[3]" LOC = P9;
NET "SW[4]" LOC = P124;
NET "SW[5]" LOC = P123;

NET "SW[0]" PULLUP;
NET "SW[1]" PULLUP;
NET "SW[2]" PULLUP;
NET "SW[3]" PULLUP;
NET "SW[4]" PULLUP;
NET "SW[5]" PULLUP;

NET "SSEG[0]" LOC = P26;
NET "SSEG[1]" LOC = P23;
NET "SSEG[2]" LOC = P21;
NET "SSEG[3]" LOC = P16;
NET "SSEG[4]" LOC = P14;
NET "SSEGA[0]" LOC = P15;
NET "SSEG[5]" LOC = P35;
NET "SSEG[6]" LOC = P33;
NET "SSEGA[1]" LOC = P24;
NET "SSEG[7]" LOC = P32;
```

Figure 14: UCF File

Appendix F

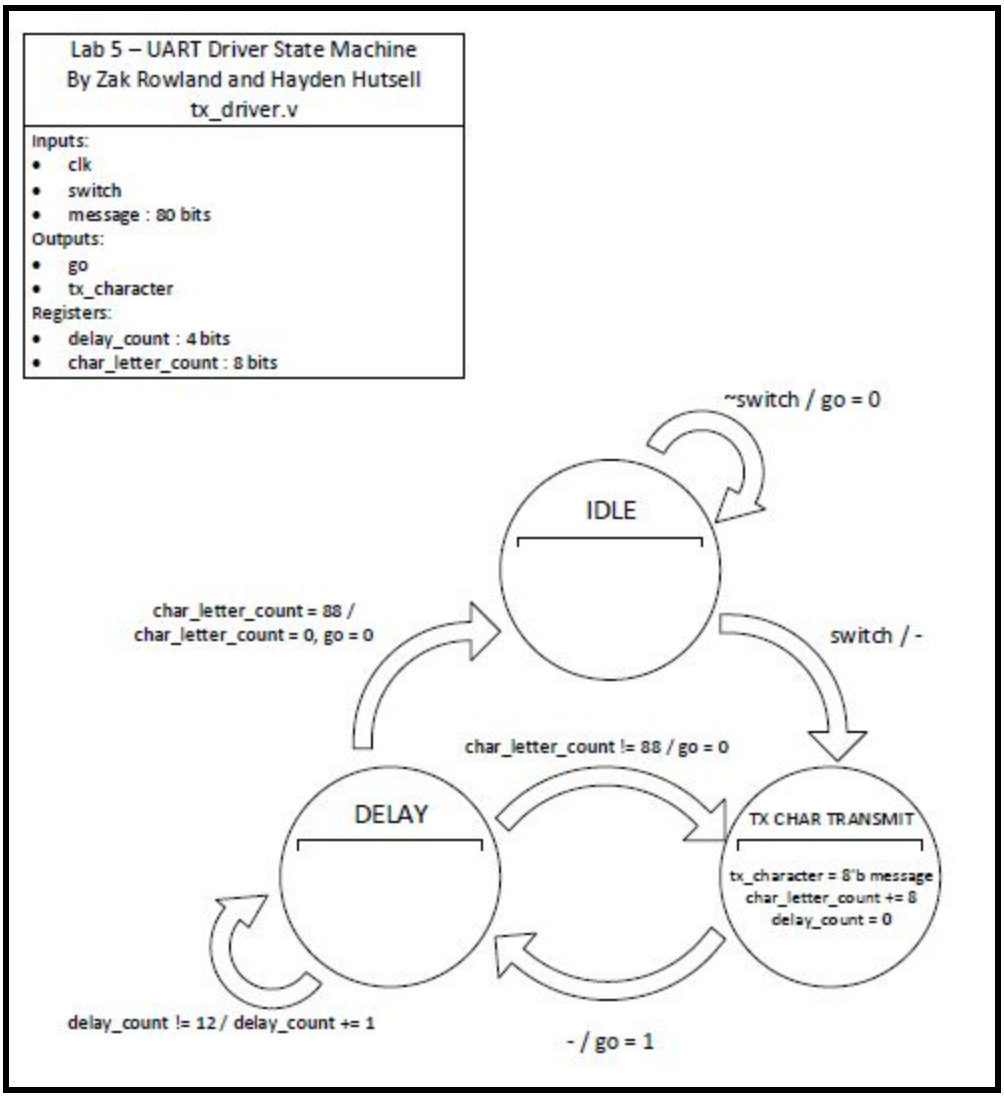

Figure 15: State machine for tx_driver

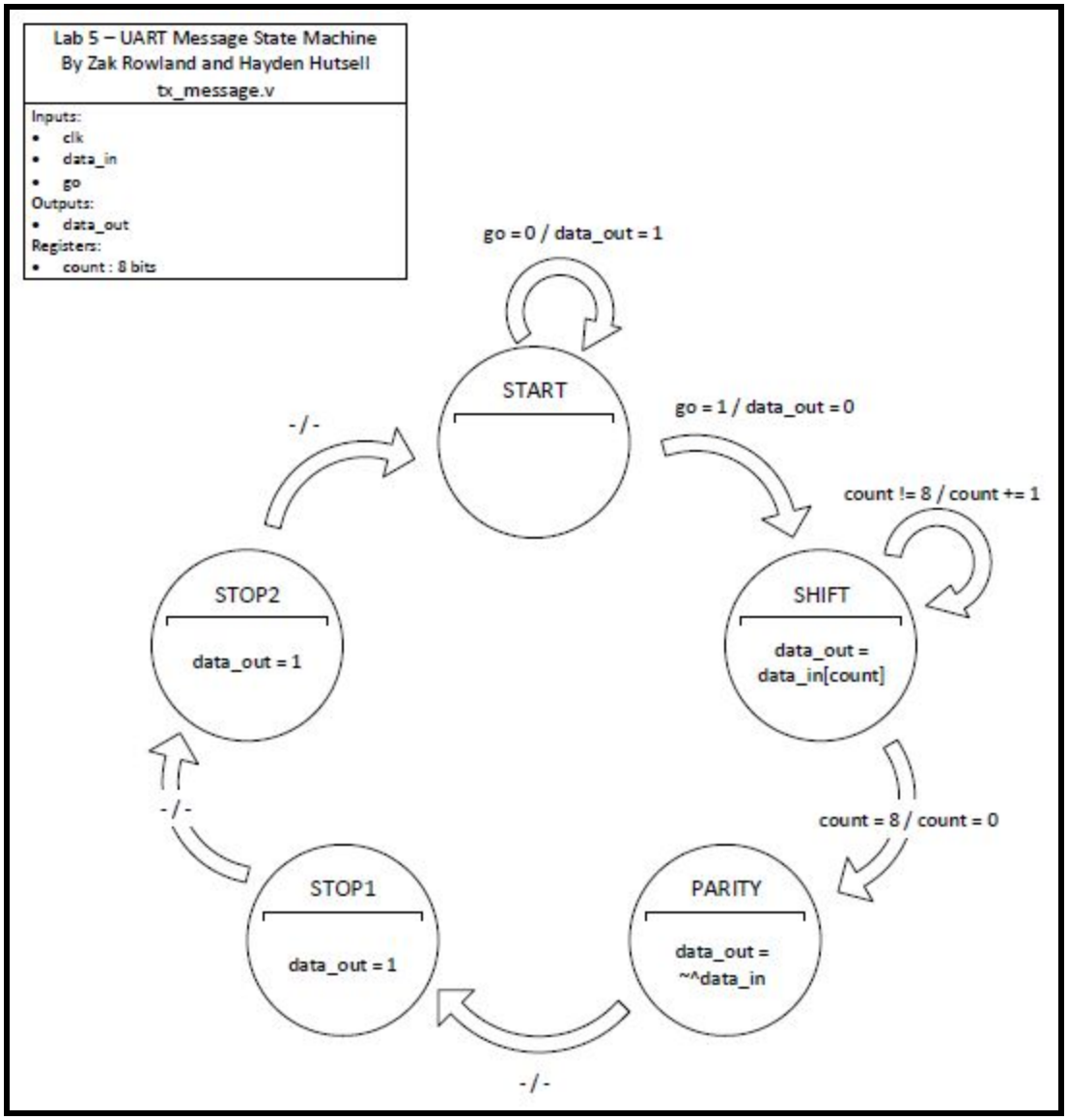

Figure 16: State machine for tx_message

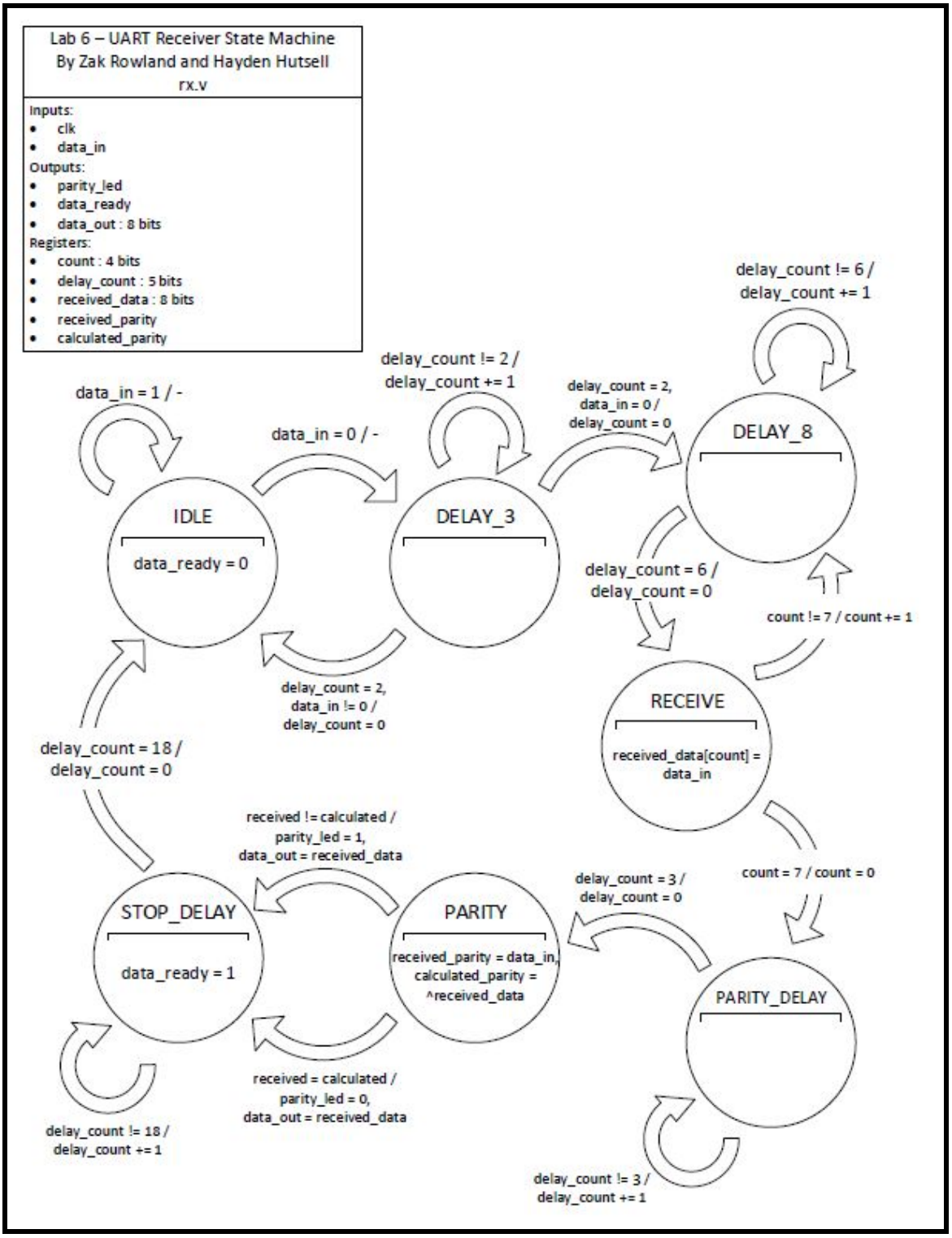

Figure 17: State machine for rx

Appendix G

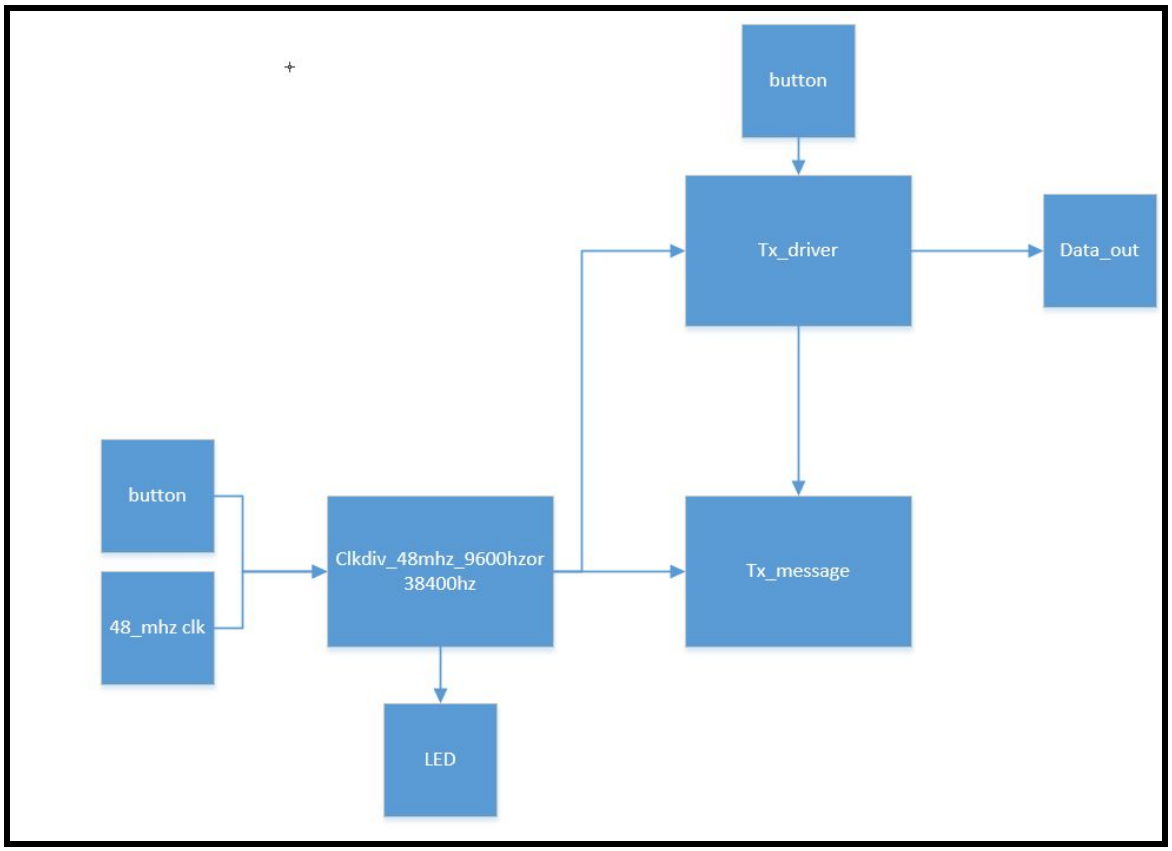

Figure 18: Block diagram for Lab 5

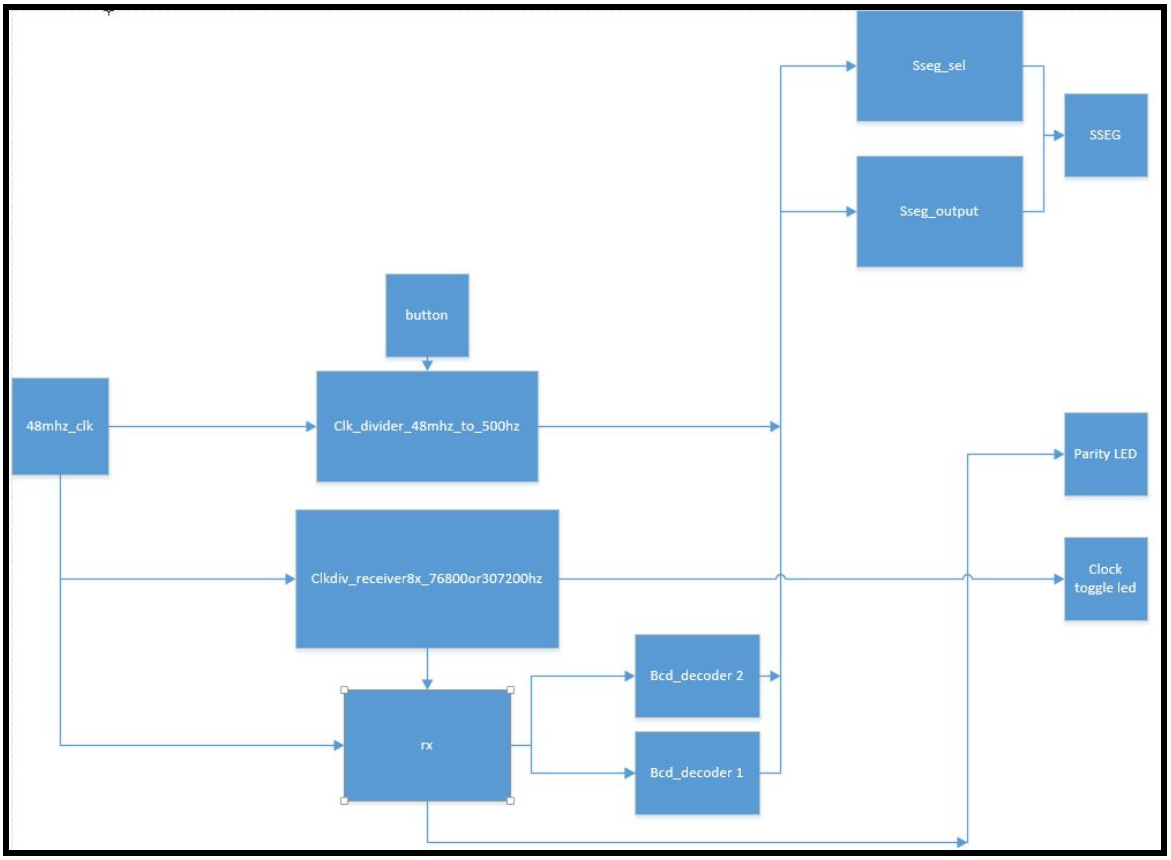

Figure 19: Block diagram for Lab 6

Appendix H

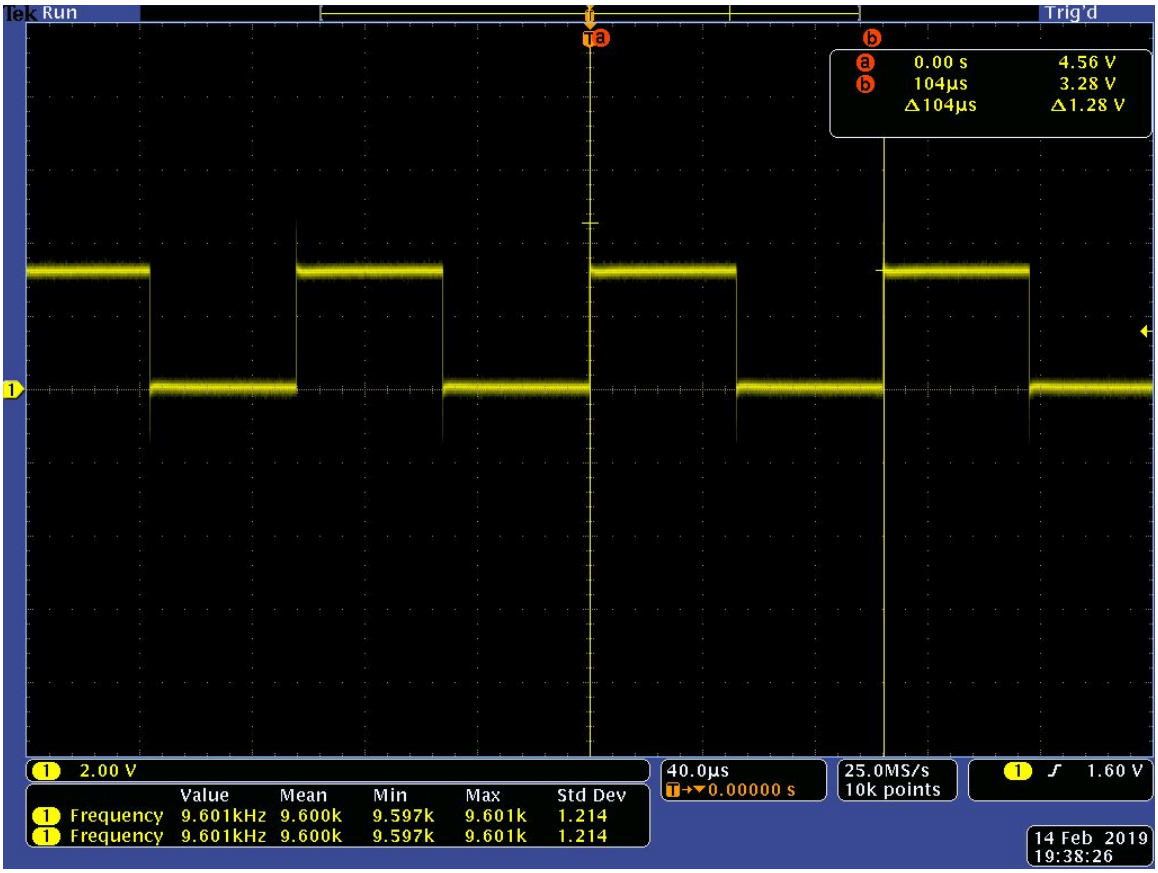

Figure 20: Clock signal for 9,600Hz from Lab 5

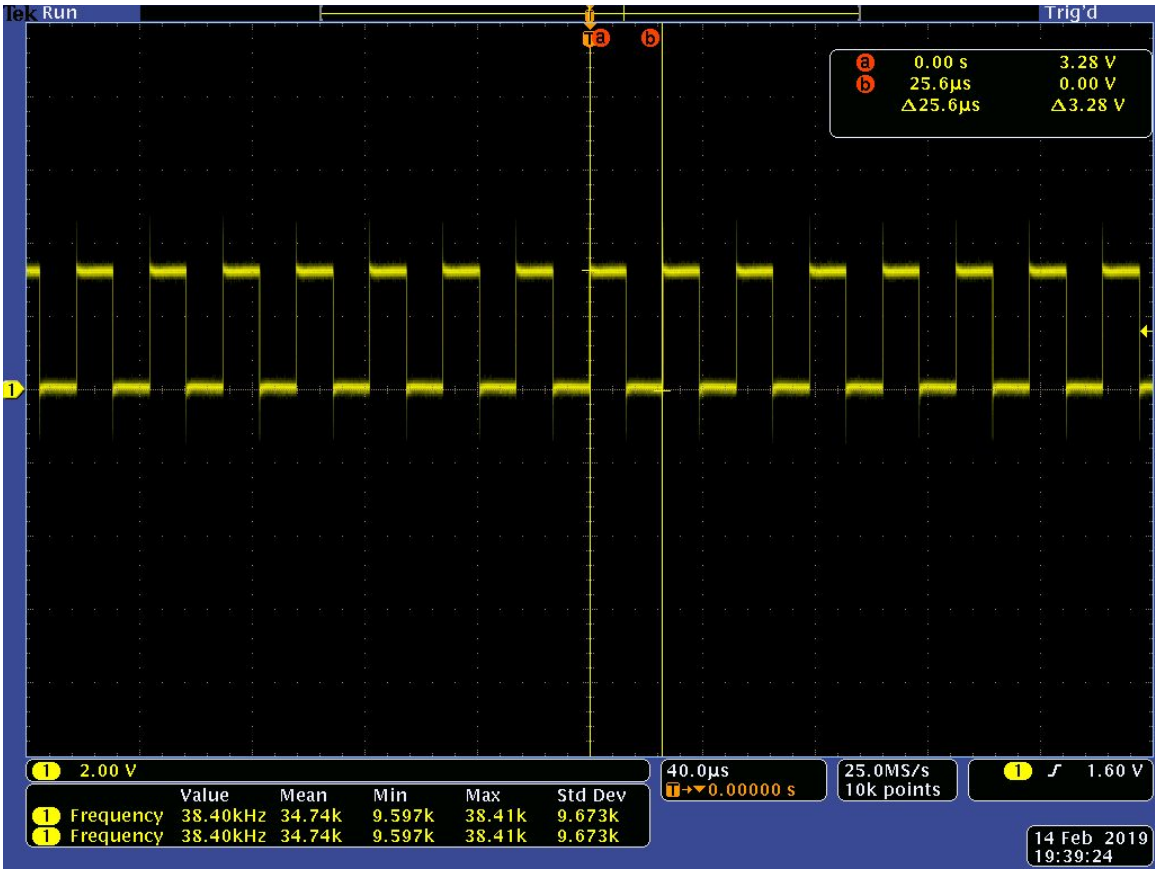

Figure 21: Clock signal for 38,400Hz from Lab 5

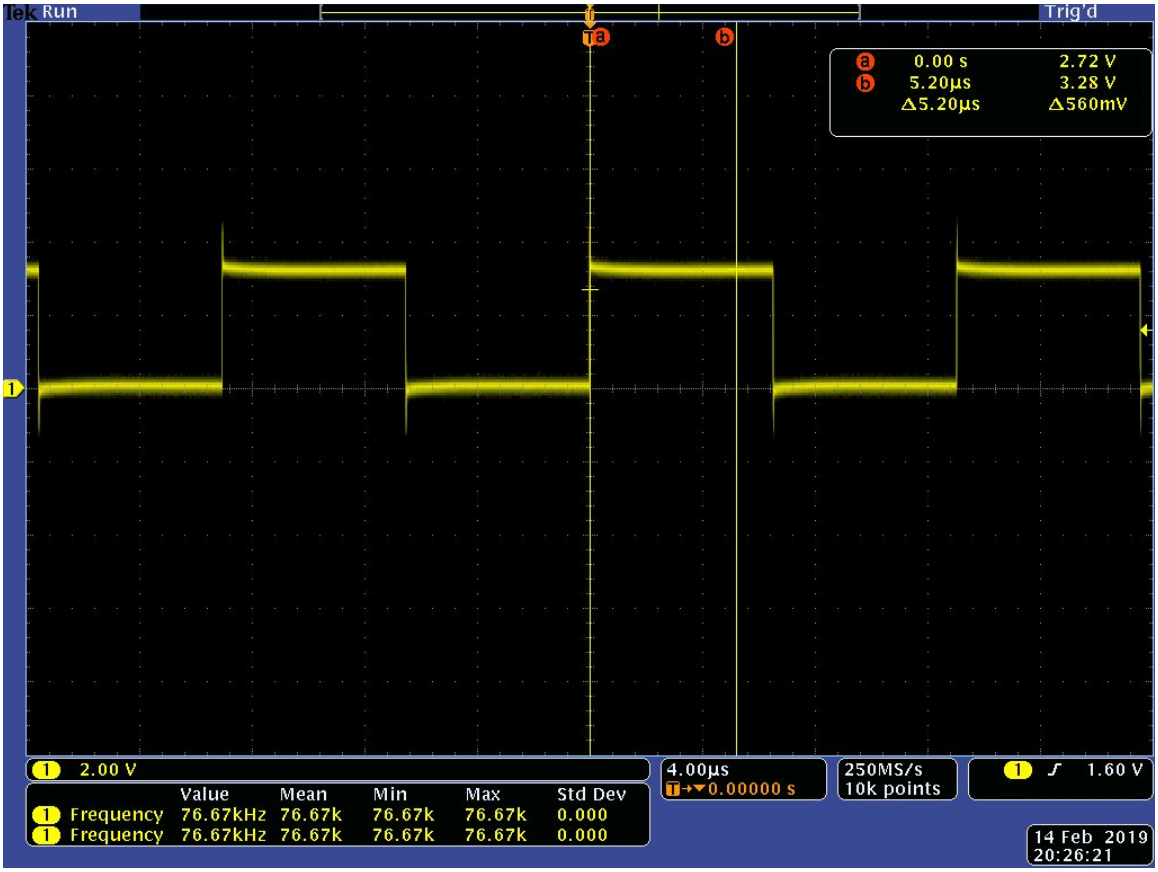

Figure 22: Clock signal for 76,800Hz from Lab 6

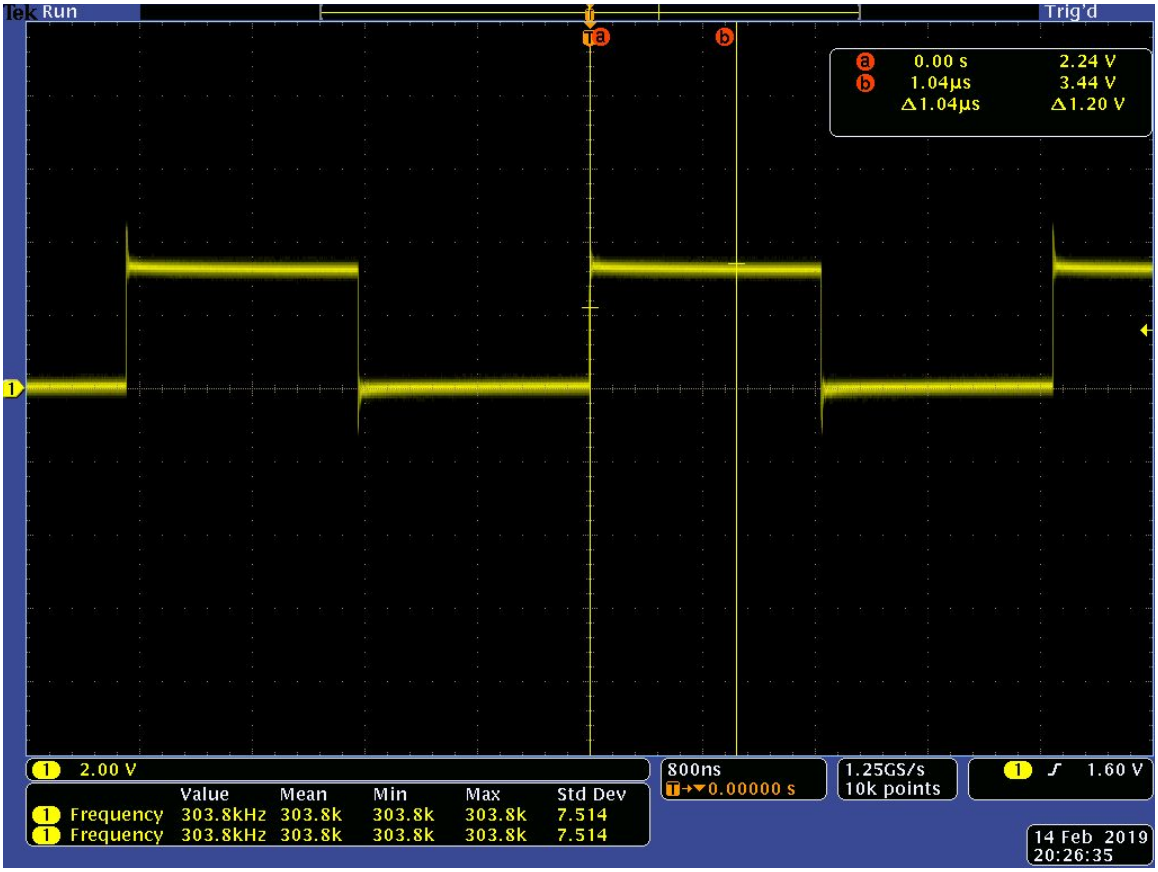

Figure 23: Clock signal for 307,200Hz from Lab 6

References

[1] MA and SS, "LDD-E2802RD," Lumex, 2013.
[2] A. Hogen and K. Pintong, "OwlBoard v3.4 Users Guide," Oregon Tech, Klamath Falls, 2018.
[3] "CPLDXAPP805 (v1.0)," Xilinx, 2005.