Name: Colton Beery

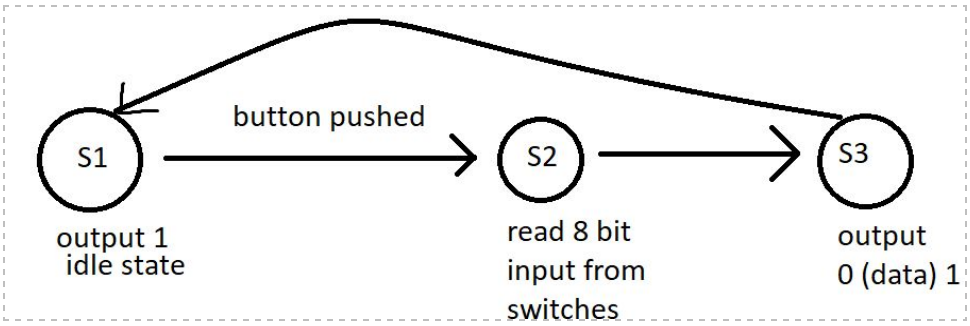
RedID: 821625071

Lab Name: UART (Tx)

Lab Summary:

What did I do to complete lab:

I watched the video on blackboard about UART Tx and then drew out a first version of my state machine logic, as shown below



Then I started trying to write out a basic state machine using this code. This didn't really work well, though and I had to make a lot of changes to get it to even output anything. After all those changes, I got it to output garbage, which was better than nothing. Every time a problem came up, I google searched what might be causing that, got a new problem, google searched, etc.

What challenges did I have:

- Was getting no data output because the board didn't like the way that I was trying to read the data and then concatenate it with the start and stop bits into a single register titled transmit.
- Laptop BSOD'd from trying to synthesize my code. Apparently using Vivado's internal debugging tools sometimes causes problems for… reasons. Still not sure why they suddenly caused problems when they worked fine for hours before that.
- Swapped back and forth between for loops and if loops several times, because I was having trouble getting if loops to output anything and I thought that outputting something that seemed like garbage was probably better than outputting nothing. Turns out my if loops were just set up wrong, because I had to use if loops in my final version.

What did I learn from the lab:

- For loops in verilog
    - Are synthesized by essentially copy/pasting the for loop over and over for every value of , so my 2 nested for loops were essentially making 8*10415 = 83320 statements.
    - do not execute sequentially, so my code was running 83320 iterations of the loop and only outputting the final result

Github link:

https://github.com/gameguy95/UART_TX

Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: SDSU
// Engineer: Colton Beery
//
// Create Date: 02/20/2019 08:59:18 AM
// Revision Date: 3/6/2019 12:01 AM
// Module Name: UART_TX
// Project Name: UART
// Target Devices: Basys3
//
// Description: Tx: The simpler of the two. When an 8 bit value is loaded into a
//              register using the 8 DIP switches for the number and a push button
//              for the "load" signal, it shifts the byte out in asynchronous
//              serial format (initially at 9600 bits per second, later at an
//              arbitrary, programmable data rate).  That begins with a start bit (0),
//              followed by the 8 data bits LSB first, and a stop (1) bit.
//
// Dependencies: Basys3_Master_Customized.xdc
//
// Revision History
// Current Revision: 0.20
// Changelog in Changelog.txt
//
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module UART_TX(
    input [7:0] IO_SWITCH, //IO Dipswitches; up = 1
    input IO_BTN_C,         //IO Pushbutton (Center); pushed = 1
    input clk,              //Master clock signal
    output reg [7:0] JA,     //PMOD JA; port JA1 used as TX pin
    output wire [7:0] IO_LED     //IO LED's for debugging data
//    output wire [9:0] IO_LED     //IO LED's for debugging transmit
    );

    /* State Machine Parameters */
    reg [1:0] state = 0;
    parameter idle = 2'b00; //State 0 = idle
    parameter start = 2'b01; //State 1 = start bit
```

```
    parameter out = 2'b10;  //State 2 = output
    parameter stop = 2'b11; //State 3 = stop bit

    /* Data and transmission parameters */
    reg [7:0] data = 0;         //data input
    reg [3:0] bit = 0;                      //bit number currently being transmitted

    parameter max_counter = 10415;          // this should give 9600 baud
    reg [13:0] counter = 0;                 //counter for baud rate generation; currently hardcoded to 14 bits for 9600 baud

    /* LED Debugging */
    assign IO_LED = data; //LEDs used to check if data is read successfully
//    assign IO_LED = transmission; //LEDs used to check if data is read successfully

    always @(posedge clk) begin

        /* Current State Logic */
          case(state)
            idle: begin
                JA[0] <= 1; //When idle, assert idle bit (1)
                /* Read Logic */
                if (IO_BTN_C) begin
                    data <= IO_SWITCH[7:0];     //read switches 1-8, where 0 is LSB
//                    transmission = {stop_bit, data, start_bit}; //transmission is Start->data (lsb first)->Stop
                    //IO_LED = transmission;
                    state <= start;
                    bit = 0;
                end
            end

            /* start bit */
            start: begin
                if (counter < max_counter) begin //if counter hasn't reached 10415 yet, transmit start bit
                        JA[0] <= 0;
                        counter <= counter + 1;
                    end else begin //when counter reaches 10415, done transmitting start bit, go to data
                        counter <= 0;
                        state <= out;
                end
            end

            /* Data transmission */
            out: begin
                    if (bit <= 7) begin // If there's still more bits to transmit
                        if (counter < max_counter) begin //if counter hasn't reached 10415 yet, transmit
                            JA[0] <= data[bit];
                            counter <= counter + 1;
                        end else begin //reset counter when it reaches 10415, and go to next bit
                            counter <= 0;
                            bit <= bit + 1;
```

```
                end
            end else begin //when you run out of bits to transmit, reset counter and bit
                counter <= 0;
                bit <= 0;
                state <= stop;
            end
        end

        /* stop bit */
        stop: begin
            if (counter < max_counter) begin //if counter hasn't reached 10415 yet, transmit stop bit
                JA[0] <= 1;
                counter <= counter + 1;
            end else begin //when counter reaches 10415, done transmitting, go back to idle
                counter <= 0;
                state <= idle;
            end
        end
    endcase
  end
endmodule
```