```
module ioTest (input  M_CLOCK,
                                        input  [3:0] IO_PB,      // IO Board Pushbutton Switches
                                    input  [7:0] IO_DSW,     // IO Board Dip Switchs
                                        output [3:0] F_LED,      // FPGA LEDs
                                    output [7:0] IO_LED,  // IO Board LEDs
                                        output [3:0] IO_SSEGD,    // IO Board Seven Segment Digits

                                        output [7:0] IO_SSEG,     //7=dp, 6=g, 5=f,4=e, 3=d,2=c,1=b, 0=a
                                        output IO_SSEG_COL,       // Seven segment column
                                        output DEC_POINT,
                                        output reg POINT);

        //===================================
        //Variable declaration

assign IO_SSEG_COL = 1; // deactivate the column displays
assign DEC_POINT = 1;   // deactivate the the decimal point of the seven segment
assign IO_SSEG = 8'b11111111; // deactivate the seven segment display
assign IO_LED = 8'b00000000; // deactivate the IO board leds
assign IO_SSEGD = 4'b1111; // deactivate the decimal points
assign F_LED = 4'b0000; // deactivate the fpga board leds



reg [1:0] state; // state to change --> 0,1,2
reg [2:0] count = 0; //counts 0-7, used for sending [7:0]data
reg [7:0] data = 0; // data bits given by the dip switches

        // generate new clock
reg [12:0] clk_count = 0; //counter up to 8192, enough to cover the new 50Mhz/9600 clock
```

```
reg new_clk = 0; //clock starts at 0


always @(posedge M_CLOCK) begin //count each posedge of the fpga clock
	if (clk_count < 5208) begin //count until M_CLOCK reaches 5208
		clk_count <= count_reg + 1;
	end
	else begin //once the counter value is reached, reset counter and toggle new_clk
		clk_count <= 0;
		new_clk <= ~new_clk; //toggles from 0 to 1, or 1 to 0
		//used in the always loops as the new clock speed
  end
end


//I'm still not sure why this didn't work, reading the push button in its own always block
//always @(posedge new_clk) begin
//	if (IO_PB[0] == 0) begin //if the pushbutton is down
//		data <= IO_DSW; //the 8 dip switch positions are loaded into the data
//		state <= 0; //the state is changed from original to 0, or 2 to 0 if the case statements have already been executed
//	end
//end



always @(posedge new_clk) begin //using the generated clock speed
	if (IO_PB[0] == 0) begin //if the pushbutton is down
		data <= IO_DSW; //the 8 dip switch positions are loaded into the data
		state <= 0; //the state is changed from original to 0, or 2 to 0 if the case statements have already been executed
	end
```

```
        else begin
                case (state)
                        0 : begin //state zero
                                        POINT <= 0; //send the start bit of 0 first, before sending data
                                        state <= 1; //change state
                                end
                        1 : begin //state 1, sends the data to the pin 1 bit at a time, using the counter
                                        POINT <= data[count]; //send the bits

                                        if (count == 7) begin //check to see if all 8 bits have been sent,
0-7
                                                state <= 2; //if transfer complete, change to next state
                                        end
                                        else begin //if the transfer is not complete, increment counter
to send next bit
                                                count <= count + 1;
                                                state <= 1; //keep current state
                                        end
                                end

                        2 : begin
                                        POINT <= 1; //send end bit
                                        count <= 0; //reset counter
                                end
                        default : POINT <= 1; //keep the output high
                endcase
        end
end
```

```
endmodule
```