ENGR 378 Digital Systems Design

# Exp. 5: Keyboard Interface

Submitted by:

Björn Franzén
Kristian Lauszus

Oct. 16, 2014

Demonstration……..…..____/5

Report: Presentation ……...____/2

Problem analysis…….…….____/2

Design work…….____/5

Results……..……..____/5

Conclusion/Discussion…….…….____/1

Total……..…….____/ 20

# Problem Analysis

In this lab we needed to read the input from a keyboard and output it to four 7-segment displays. Our approach to modularize it according to the picture shown in our pre-lab report.

# Hardware Design

We decided to create two modules, one for interpreting data from the keyboard and one top-module for displaying the transmission value on the 7-segment display.

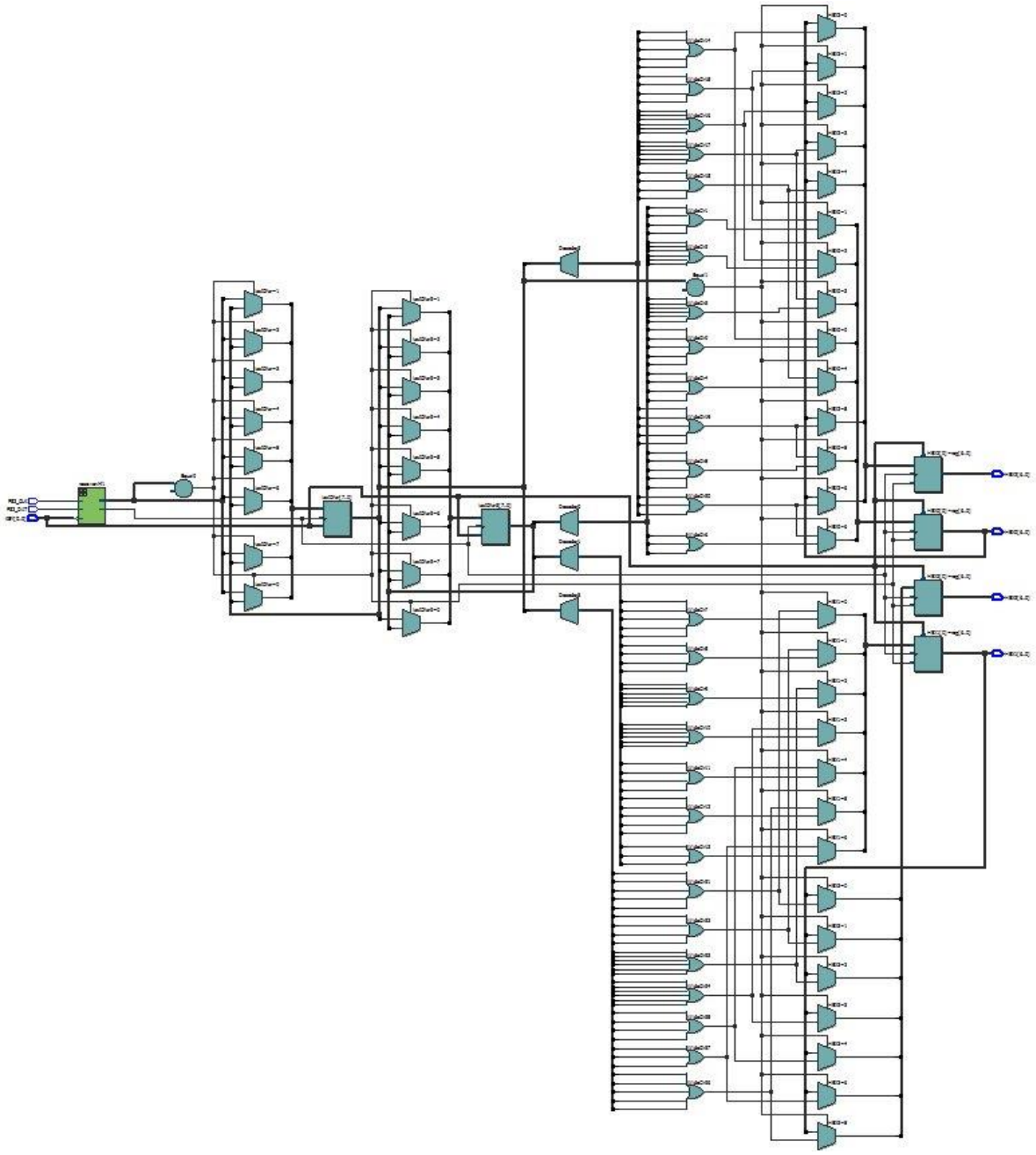


*Figure 1 – Hardware synthesis of the top module.*

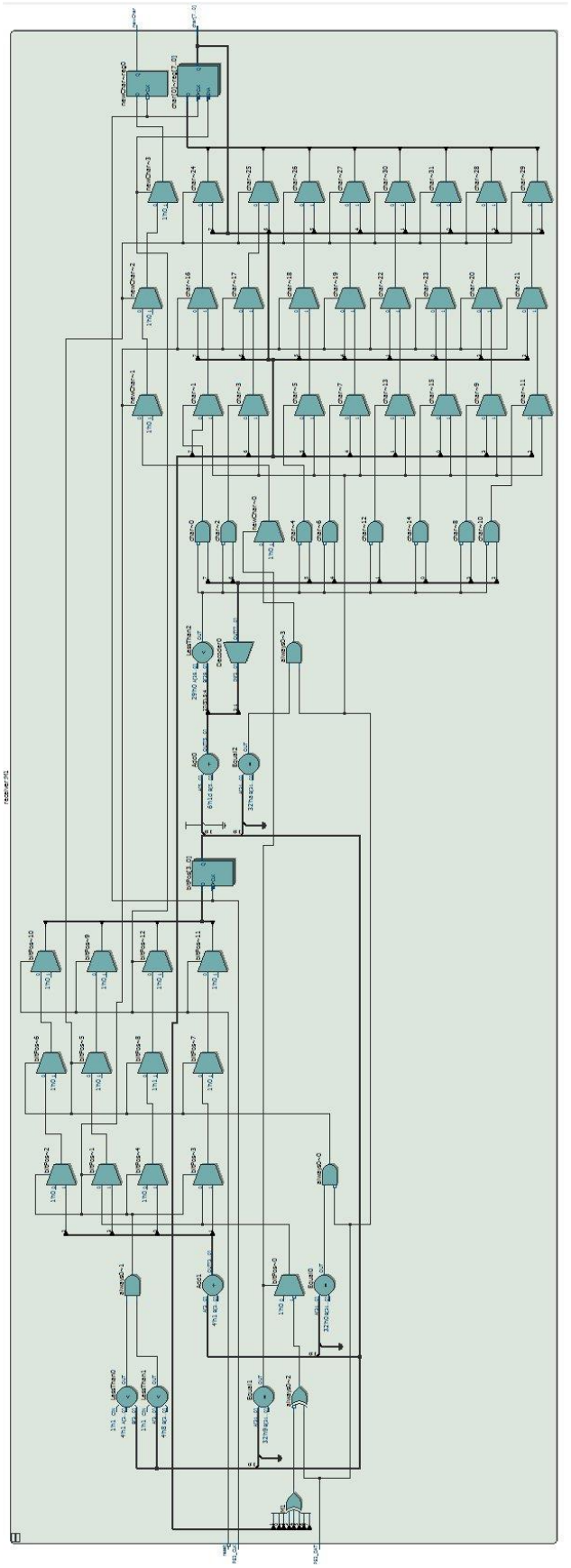


*Figure 2 - Hardware synthesis of the receiver module.*

# Verilog Modelling

The receiver module consists of one always statement that is triggered on a negative edge on the PS/2 clock signal. The basic implementation is a state machine where the state is determined by the current position in the transmission sequence. First it checks that the start bit is high, then it reads the following 8-bits into a buffer and checks the parity afterwards. Finally it checks the stop-bit. If all passes it will indicate that a new transmission is received by setting an output high.

The top-module will trigger on the positive edge of this output and checks for the stop sequence. If true then it will shift the last transmission in from the right, on the 7-segment displays. There is a special case when pressing the arrow keys, as these send out two transmissions instead of one, as described in the lab manual. We solved this by checking for the special character, 0xE0, that indicates a 16-bit key-code. If that happens it will show the first transmission on the two rightmost displays and 0xE0 on the two leftmost displays.

Finally we implemented a reset signal that clears the displays and resets the receiver module.

# Results/Verification

First we created a test bench to verify that our receiver module worked as expected. The test bench simulated an arbitrary transmission of 0x55. The waveform can be seen in Figure 3.

After we confirmed the receiver module, we wrote a test bench for the entire design. We did this by simulating a release of the “up arrow” key. As can be seen in Figure 4, everything worked as expected.


*Figure 3 - Simulation waveform of the receiver module.*

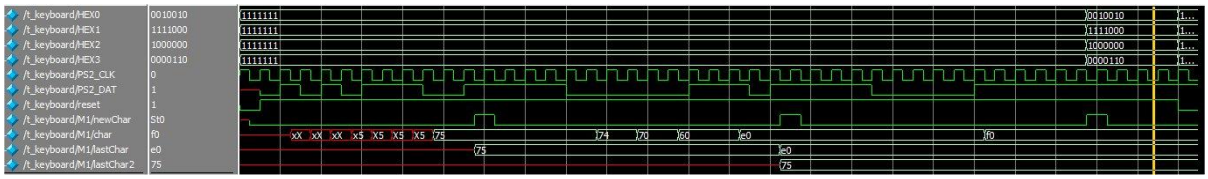
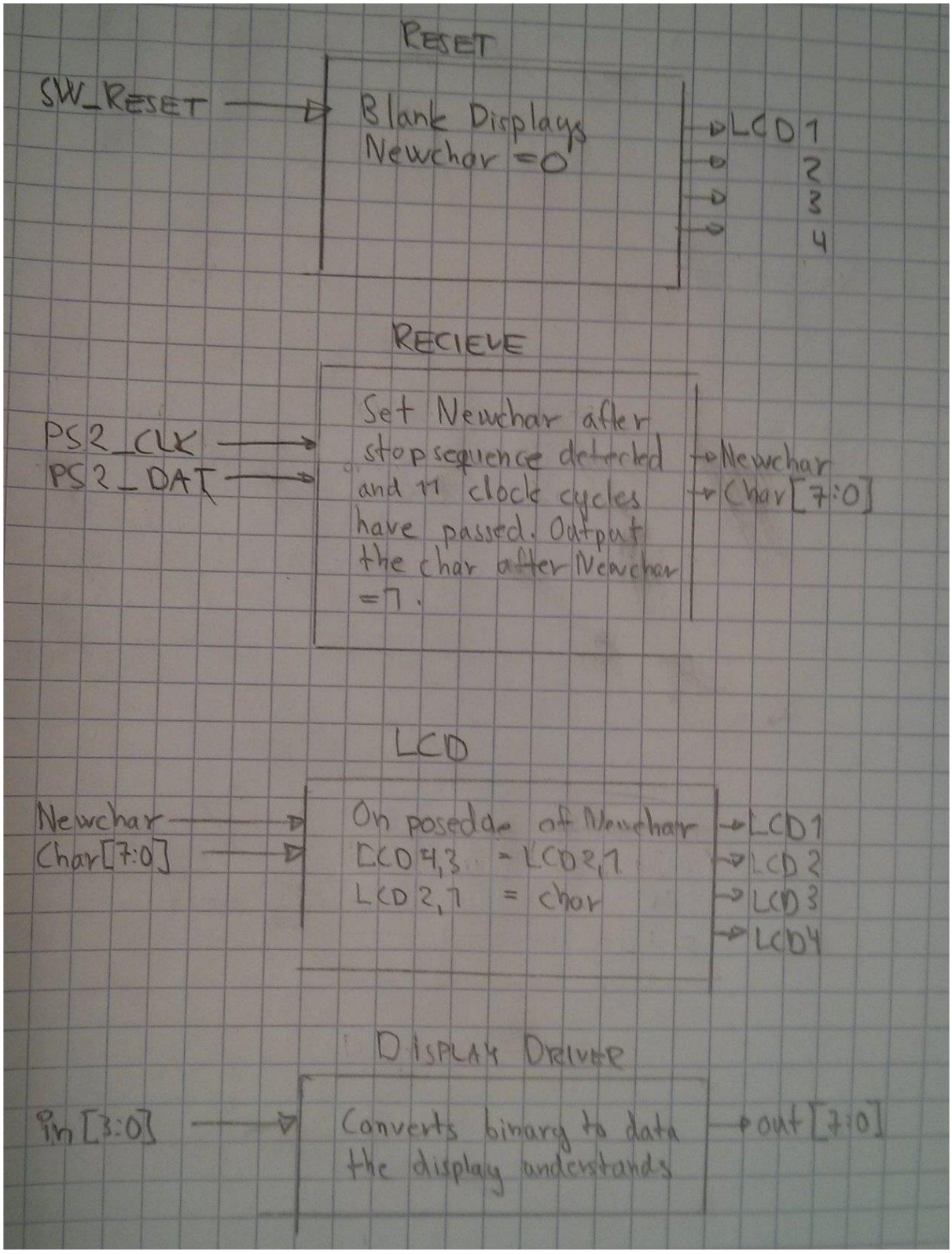



*Figure 4 - Simulation waveform of entire design.*

# Conclusions and Discussion

We found that it was good to have a slightly larger project this time so that we could break down the design into different parts and divide the work between us. This actually created fewer problems for us, in relation to previous labs and allowed us to do this entire lab during one session.

## Pre-lab




## Work Breakdown

The receiver module was created by Kristian and the top module by Björn.

# HDL Source Code

```
module keyboard(
        output reg [6:0] HEX0, HEX1, HEX2, HEX3, // 7-segment displays
        input  PS2_CLK, PS2_DAT, // The clock and data signals from the PS/2 device
        input [0:0] KEY // Used as reset signal
 );
        wire newChar;
        wire [7:0] char;
        reg [7:0] lastChar, lastChar2;

        receiver M1(.newChar(newChar), .char(char), .PS2_CLK(PS2_CLK),
.PS2_DAT(PS2_DAT), .reset(!KEY[0]));

        always @ (posedge newChar or negedge KEY[0]) begin
                if (!KEY[0]) begin
                        HEX0 = 7'b1111111; // OFF
                        HEX1 = 7'b1111111; // OFF
                        HEX2 = 7'b1111111; // OFF
                        HEX3 = 7'b1111111; // OFF
                end
                else if (char == 8'hF0) begin // This will ensure that we only
display the last value once the button is released
                        if (lastChar == 8'hE0) begin // Make sure that it display
the entire key-code of the arrows and other buttons that sends a 16-bit
transmission
                                HEX2 = display_driver(lastChar[3:0]);
                                HEX3 = display_driver(lastChar[7:4]);
                                HEX0 = display_driver(lastChar2[3:0]);
                                HEX1 = display_driver(lastChar2[7:4]);
                        end
                        else begin
                                HEX2 = HEX0; // Shift value to next two displays
                                HEX3 = HEX1;
                                HEX0 = display_driver(lastChar[3:0]);
                                HEX1 = display_driver(lastChar[7:4]);
                        end
                end
                else begin
                        lastChar2 = lastChar; // Save the char before last char
                        lastChar = char; // Save last char
                end
        end

        function [6:0] display_driver;
                input [3:0] in;
                case (in)
                        4'b0000 : display_driver = 7'b1000000; // 0
                        4'b0001 : display_driver = 7'b1111001; // 1
                        4'b0010 : display_driver = 7'b0100100; // 2
                        4'b0011 : display_driver = 7'b0110000; // 3
                        4'b0100 : display_driver = 7'b0011001; // 4
                        4'b0101 : display_driver = 7'b0010010; // 5
                        4'b0110 : display_driver = 7'b0000011; // 6
                        4'b0111 : display_driver = 7'b1111000; // 7
                        4'b1000 : display_driver = 7'b0000000; // 8
                        4'b1001 : display_driver = 7'b0011000; // 9

                        4'b1010 : display_driver = 7'b0001000; // A
                        4'b1011 : display_driver = 7'b0000000; // B (8)
                        4'b1100 : display_driver = 7'b1000110; // C
                        4'b1101 : display_driver = 7'b1000000; // D (0)
                        4'b1110 : display_driver = 7'b0000110; // E
                        4'b1111 : display_driver = 7'b0001110; // F

                        default : display_driver = 7'b1111111; // OFF
                endcase
        endfunction
```

```
endmodule

module t_keyboard;
	wire [6:0] HEX0, HEX1, HEX2, HEX3; // 7-segment displays
	reg  PS2_CLK, PS2_DAT; // The clock and data signals from the PS/2 device
	reg reset; // Used as reset signal

	keyboard M1(.PS2_CLK(PS2_CLK), .PS2_DAT(PS2_DAT), .KEY(reset), .HEX0(HEX0),
.HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3));

	initial begin
		PS2_CLK = 1; reset = 1; // Set initial values
	end

	always #5 PS2_CLK = !PS2_CLK; // Generate clk signal

	// Test by releasing up arrow
	always begin
		reset = 0;
		#10 reset = 1;

		// Send 0x75
		PS2_DAT = 0; // Start bit
		#10 PS2_DAT = 1; // D0
		#10 PS2_DAT = 0; // D1
		#10 PS2_DAT = 1; // D2
		#10 PS2_DAT = 0; // D3
		#10 PS2_DAT = 1; // D4
		#10 PS2_DAT = 1; // D5
		#10 PS2_DAT = 1; // D6
		#10 PS2_DAT = 0; // D7
		#10 PS2_DAT = 0; // Parity bit
		#10 PS2_DAT = 1; // Stop bit

		// Send E0
		#50 PS2_DAT = 0; // Start bit
		#10 PS2_DAT = 0; // D0
		#10 PS2_DAT = 0; // D1
		#10 PS2_DAT = 0; // D2
		#10 PS2_DAT = 0; // D3
		#10 PS2_DAT = 0; // D4
		#10 PS2_DAT = 1; // D5
		#10 PS2_DAT = 1; // D6
		#10 PS2_DAT = 1; // D7
		#10 PS2_DAT = 0; // Parity bit
		#10 PS2_DAT = 1; // Stop bit

		// Send F0
		#50 PS2_DAT = 0; // Start bit
		#10 PS2_DAT = 0; // D0
		#10 PS2_DAT = 0; // D1
		#10 PS2_DAT = 0; // D2
		#10 PS2_DAT = 0; // D3
		#10 PS2_DAT = 1; // D4
		#10 PS2_DAT = 1; // D5
		#10 PS2_DAT = 1; // D6
		#10 PS2_DAT = 1; // D7
		#10 PS2_DAT = 1; // Parity bit
		#10 PS2_DAT = 1; // Stop bit

		#50 reset = 0;
		#10 $stop;
	end

endmodule

module receiver(
		output reg newChar,
```

```
                output reg[7:0] char,
                input PS2_CLK, PS2_DAT, reset
        );
        reg[3:0] bitPos; // Current bit position in transmission
        wire parityCheck;

        xor M1(parityCheck, char[0], char[1], char[2], char[3], char[4], char[5],
char[6], char[7]);

        always @ (negedge PS2_CLK)
        begin
                newChar = 0;
                if (reset) bitPos = 0; // Reset bit position
                else if (bitPos == 0 & PS2_DAT == 0) bitPos = 1; // Check start
bit
                else if (bitPos >= 1 & bitPos <= 8) begin
                        char[bitPos - 1] = PS2_DAT;
                        bitPos = bitPos + 1'd1;
                end
                else if (bitPos == 9) begin // Check parity
                        if (PS2_DAT == ~parityCheck)
                                bitPos = 10; // Go to stop bit state
                        else
                                bitPos = 0; // Error in transmission
                end
                else if (bitPos == 10 & PS2_DAT == 1) begin // Check stop bit
                        newChar = 1;
                        bitPos = 0;
                end
                else bitPos = 0; // If everything fails, then there must be an
error in the transmission
        end

endmodule

module t_receiver;
        wire newChar;
        wire [7:0] char;
        reg PS2_CLK, PS2_DAT, reset;

        receiver M1(.newChar(newChar), .char(char), .PS2_CLK(PS2_CLK),
.PS2_DAT(PS2_DAT), .reset(reset));

        initial begin
                PS2_CLK = 1; reset = 0; // Set initial values
        end

        always #5 PS2_CLK = !PS2_CLK; // Generate clk signal

        always begin
                reset = 1;
                #10 reset = 0;
                PS2_DAT = 0; // Start bit
                #10 PS2_DAT = 1; // D0
                #10 PS2_DAT = 0; // D1
                #10 PS2_DAT = 1; // D2
                #10 PS2_DAT = 0; // D3
                #10 PS2_DAT = 1; // D4
                #10 PS2_DAT = 0; // D5
                #10 PS2_DAT = 1; // D6
                #10 PS2_DAT = 0; // D7
                #10 PS2_DAT = 1; // Parity bit
                #10 PS2_DAT = 1; // Stop bit
                #10 $stop;
        end

endmodule
```