

Logic Analyzer Display System

Using Pmod OLEDrgb

**Submitted by:**

KANGULA JAI LAKSHMI

Roll no. – 160123735011

Branch – ECE (E1)

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

July 2025

# Abstract

This project presents the design and implementation of a miniature, FPGA-based logic analyzer display system using the Spartan-7 FPGA (XC7S50CSG324-1) boolean board and the Pmod OLEDrgb module. The objective was to create a system that could capture and visualize digital logic signals in real time on a small-scale OLED display without an external computer. Logic analyzers are essential tools for observing digital waveforms and understanding signal transitions. By using a compact 96x64 OLED display (SSD1331), this system can visualize real-time logic signals in a compact and user-friendly format. The project uses Verilog HDL for core logic and Vivado for synthesis and implementation. Key functionalities of the system include signal capture, SPI(Serial Peripheral Interface) communication with the OLED, and waveform rendering. This implementation acts as an educational and practical embedded systems application that aids in the analysis of digital systems. This report delves into the details of the project’s hardware and its Verilog logic, along with focusing on the system architecture and flow of information. It will give the reader a brief overview of the project's details.

**Aim**

The logic analyzer display project aimed to interface the Spartan-7 FPGA and the Pmod OLEDrgb module to display and analyze the signal samples.

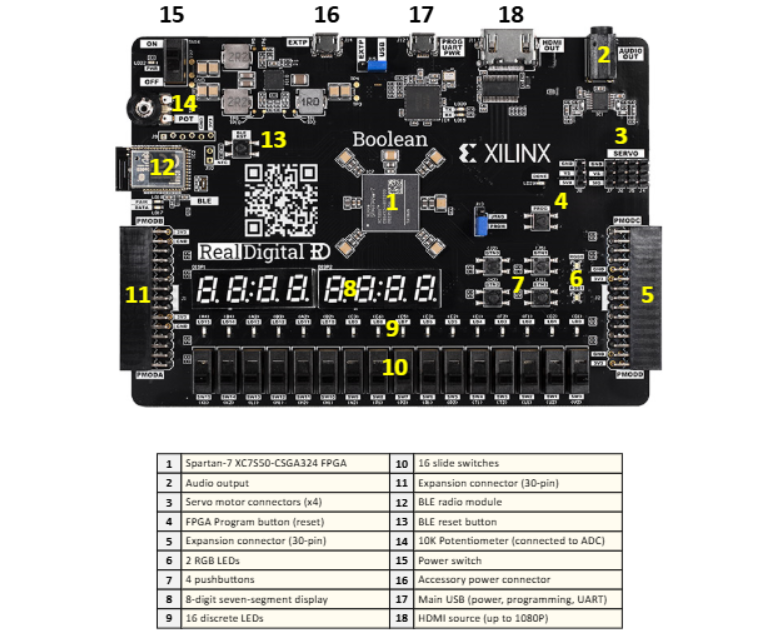
# Introduction

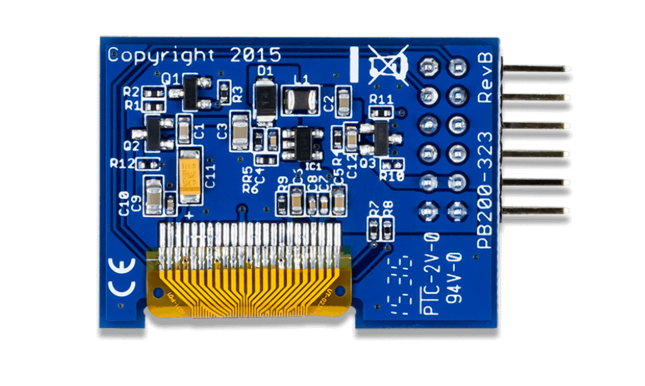
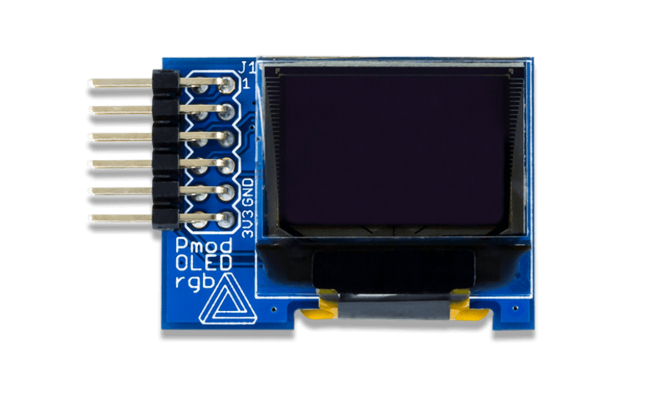
In modern digital systems, logic analyzers are critical devices that allow engineers and developers to visualize and debug signal behavior. This project aims to develop a compact and functional logic analyzer display system using anar Xilinx Spartan-7 FPGA (XC7S50CSG324-1) and a Pmod OLEDrgb OLED module. The basic outcome is to provide a simple yet effective method to monitor logic levels in real-time on a small RGB display without the need for external computers or expensive debugging tools.  
  
The Spartan-7 FPGA offers flexibility and high-speed digital design capability, while the Pmod OLEDrgb display with its 96x64 pixel resolution, provides a compact medium to visualize digital signals. This system can capture 4-bit input logic signals, process them using Verilog logic, and then output the waveform data to the OLED display using SPI communication. There are a few features that are also embedded such as switching between test and real-time signals, controlling the speed, and freezing the signals. This project finds its use in academic learning, embedded system debugging, and as a baseline for larger digital analyzer systems.

# System Specifications and Tools Used

The project utilizes the following hardware and software components:  
  
• FPGA Board: Xilinx Spartan-7 XC7S50CSG324-1  
• Display Module: Digilent Pmod OLEDrgb (based on SSD1331 controller)  
• Display Resolution: 96 x 64 pixels  
• Color Format: 16-bit RGB (5-6-5)  
• Programming Language: Verilog HDL  
• Development Software: Xilinx Vivado 2020.2 or higher  
• Communication Interface: SPI ( Serial Peripheral Interface)  
• Input Channels: 4-bit digital signal input  
  
These specifications ensure a compact, real-time visualization system with low hardware overhead.

# Hardware Overview

1. Spartan-7 FPGA (XC7S50CSG324-1) boolean board
2. Pmod OLEDrgb Display Module

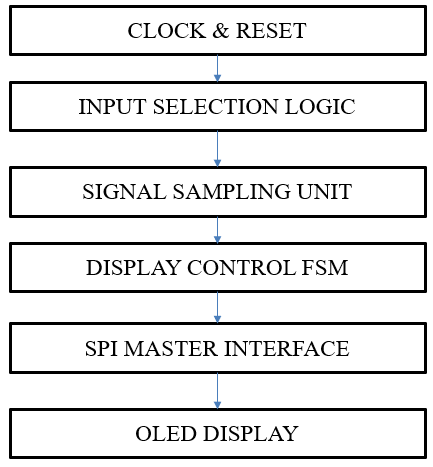


1. PMOD C Connector
2. Vivado Design Suite

# 4.Methodology

# System Design and Architecture

The logic analyzer display system is designed using a modular architecture comprising the following components:  
  
1. Top\_module (also has) – Samples 4-bit input signals at the clock rate and stores them temporarily for rendering.  
2. Logic\_display\_pixel\_gen – Translates signal samples into waveform pixel coordinates.  
3. SPI Master – Handles data transmission over SPI to the OLED.  
4. SSD1331\_INIT – Manages initialization and graphics commands to SSD1331.  
  
These components are integrated through a top-level control module that synchronizes capture, process, and display operations. The following block diagram illustrates the high-level system design:



The design operates on a 100 MHz system clock and updates the display at user-selected speeds. The waveform rendering algorithm ensures scrolling effects with background refresh logic.

# Verilog Implementation

The project is developed entirely using Verilog HDL. Each major component of the system is written as a separate module and instantiated in the top-level design. The major modules include:  
  
• logic\_display\_pixel\_gen.v – Signals samples from the external are converted to waveform pixel coordinates.  
• spi\_master.v – Handles SPI communication with proper clock division and synchronization.  
• ssd1331\_init.v – Sends initialization commands and drawing instructions to SSD1331.  
• top\_logic\_analyzer.v – Integrates all components and manages display logic.  
  
The design ensures pipelining of data to the OLED for real-time rendering. Below is a snippet of the top module declaration along with all the other modules.

**1.TOP MODULE**

**module top\_logic\_analyzer**(  
 input clk,

input rst,

input [3:0] logic\_in\_external,

input speed\_switch,

input mode\_select, // 1 = test signal mode, 0 = external input

input freeze\_button, // 1 = freeze test signal counter

output sclk,

output mosi,

output cs,

output dc,

output res\_n,

output vccen,

output pmoden

);

// SPI/Init Wiring

wire spi\_done, init\_done;

wire [7:0] spi\_data;

wire spi\_start;

// Sample buffer (96 pixels wide)

reg [3:0] sample\_mem [0:95];

integer j;

// Sampling rate

reg [15:0] sample\_counter = 0;

reg [15:0] sample\_rate = 16'd50000;

// SPI drawing

wire sclk\_int, mosi\_int, cs\_int, dc\_init, res\_n\_init;

reg draw\_dc = 0;

reg draw\_spi\_start = 0;

reg [7:0] draw\_spi\_data = 0;

reg drawing = 0;

// FSM coordinates

reg [6:0] x = 0;

reg [5:0] y = 0;

reg [4:0] state = 0;

// Pixel color

wire [15:0] pixel\_color;

// Internal counter-based test signal generator

reg [31:0] counter = 0;

always @(posedge clk) begin

if (!freeze\_button) // freeze when button is held

counter <= counter + 1;

end

wire [3:0] logic\_test;

assign logic\_test[0] = counter[24]; // CH0 - slow

assign logic\_test[1] = counter[23]; // CH1 - medium

assign logic\_test[2] = counter[22]; // CH2 - faster

assign logic\_test[3] = counter[21] ^ counter[20]; // CH3 - dynamic XOR

// Select input source

wire [3:0] logic\_in;

assign logic\_in = (mode\_select) ? logic\_test : logic\_in\_external;

// Sample and scroll logic

always @(posedge clk) begin

sample\_rate <= (speed\_switch) ? 16'd10000 : 16'd50000;

sample\_counter <= sample\_counter + 1;

if (sample\_counter >= sample\_rate) begin

sample\_counter <= 0;

for (j = 0; j < 95; j = j + 1)

sample\_mem[j] <= sample\_mem[j + 1];

sample\_mem[95] <= logic\_in;

end

end

// OLED Init FSM

ssd1331\_init oled\_init (

.clk(clk),

.rst(rst),

.spi\_data(spi\_data),

.spi\_start(spi\_start),

.spi\_done(spi\_done),

.dc(dc\_init),

.res\_n(res\_n\_init),

.init\_done(init\_done)

);

// SPI Master

spi\_master spi (

.clk(clk),

.rst(rst),

.data\_in(drawing ? draw\_spi\_data : spi\_data),

.start(drawing ? draw\_spi\_start : spi\_start),

.sclk(sclk\_int),

.mosi(mosi\_int),

.cs(cs\_int),

.done(spi\_done)

);

// Pixel color generator (you must use the version for 4 channels)

logic\_display\_pixel\_gen pixel\_gen (

.sample((x < 8) ? 4'b0000 : sample\_mem[x - 8]),

.x(x),

.y(y),

.pixel\_color(pixel\_color)

);

// OLED Drawing FSM (1 column per frame)

always @(posedge clk or posedge rst) begin

if (rst) begin

state <= 0;

x <= 0; y <= 0;

draw\_spi\_start <= 0;

draw\_dc <= 0;

drawing <= 0;

draw\_spi\_data <= 0;

end else if (init\_done) begin

draw\_spi\_start <= 0;

case (state)

0: begin

draw\_spi\_data <= 8'h15;

draw\_spi\_start <= 1;

draw\_dc <= 0;

drawing <= 1;

state <= 1;

end

1: if (spi\_done) begin

draw\_spi\_data <= x;

draw\_spi\_start <= 1;

state <= 2;

end

2: if (spi\_done) begin

draw\_spi\_data <= x;

draw\_spi\_start <= 1;

state <= 3;

end

3: if (spi\_done) begin

draw\_spi\_data <= 8'h75;

draw\_spi\_start <= 1;

state <= 4;

end

4: if (spi\_done) begin

draw\_spi\_data <= 0;

draw\_spi\_start <= 1;

state <= 5;

end

5: if (spi\_done) begin

draw\_spi\_data <= 63;

draw\_spi\_start <= 1;

state <= 6;

end

6: if (spi\_done) begin

draw\_dc <= 1;

y <= 0;

state <= 7;

end

7: begin

draw\_spi\_data <= pixel\_color[15:8];

draw\_spi\_start <= 1;

state <= 8;

end

8: if (spi\_done) begin

draw\_spi\_data <= pixel\_color[7:0];

draw\_spi\_start <= 1;

state <= 9;

end

9: if (spi\_done) begin

if (y < 63) begin

y <= y + 1;

state <= 7;

end else begin

y <= 0;

x <= (x + 1) % 96;

state <= 0;

end

end

default: state <= 0;

endcase

end

end

// Outputs

assign sclk = sclk\_int;

assign mosi = mosi\_int;

assign cs = cs\_int;

assign dc = drawing ? draw\_dc : dc\_init;

assign res\_n = res\_n\_init;

assign vccen = 1'b1;

assign pmoden = 1'b1;

endmodule

**2. LOGIC\_DISPLAY\_PIXEL\_GEN:**

module logic\_display\_pixel\_gen(  
    input [3:0] sample,  
    input [6:0] x,  
    input [5:0] y,  
    output reg [15:0] pixel\_color  
);  
  
    // RGB565 colors  
    localparam COLOR\_RED    = 16'hF800;  // CH0  
    localparam COLOR\_GREEN  = 16'h07E0;  // CH1  
    localparam COLOR\_BLUE   = 16'h001F;  // CH2  
    localparam COLOR\_YELLOW = 16'hFFE0;  // CH3  
    localparam COLOR\_WHITE  = 16'hFFFF;  
    localparam COLOR\_BLACK  = 16'h0000;  
  
    always @(\*) begin  
        // Label zones on the left (x = 0-7)  
        if (x < 8) begin  
            if (y < 16)        pixel\_color = COLOR\_RED;    // CH0 stripe  
            else if (y < 32)   pixel\_color = COLOR\_GREEN;  // CH1 stripe  
            else if (y < 48)   pixel\_color = COLOR\_BLUE;   // CH2 stripe  
            else               pixel\_color = COLOR\_YELLOW; // CH3 stripe  
        end  
        // Waveform zone  
        else begin  
            if (y < 16)  
                pixel\_color = sample[0] ? COLOR\_RED : COLOR\_BLACK;  
            else if (y < 32)  
                pixel\_color = sample[1] ? COLOR\_GREEN : COLOR\_BLACK;  
            else if (y < 48)  
                pixel\_color = sample[2] ? COLOR\_BLUE : COLOR\_BLACK;  
            else  
                pixel\_color = sample[3] ? COLOR\_YELLOW : COLOR\_BLACK;  
        end  
    end  
endmodule

**3.SPI MASTER:**

module spi\_master(  
// SPI master with proper 8-bit transfer (Mode 0, SCLK idle low)  
    input clk,  
    input rst,  
    input [7:0] data\_in,  
    input start,  
    output reg sclk,  
    output reg mosi,  
    output reg cs,  
    output reg done  
);  
    reg [3:0] bit\_cnt;  
    reg [7:0] shift\_reg;  
    reg [1:0] state;  
    localparam IDLE=0, LOAD=1, TRANSFER=2, FINISH=3;  
  
    always @(posedge clk or posedge rst) begin  
        if (rst) begin  
            state <= IDLE;  
            sclk <= 0; // Mode 0: idle low  
            mosi <= 0;  
            cs   <= 1;  
            done <= 0;  
            bit\_cnt <= 0;  
            shift\_reg <= 0;  
        end else begin  
            case(state)  
                IDLE: begin  
                    sclk <= 0;  
                    cs <= 1;  
                    done <= 0;  
                    if (start) begin  
                        cs <= 0;  
                        shift\_reg <= data\_in;  
                        bit\_cnt <= 4'd7;  
                        state <= LOAD;  
                    end  
                end  
                LOAD: begin  
                    mosi <= shift\_reg[7];  
                    state <= TRANSFER;  
                end  
                TRANSFER: begin  
                    sclk <= 1;  
                    state <= FINISH;  
                end  
                FINISH: begin  
                    sclk <= 0;  
                    shift\_reg <= {shift\_reg[6:0],1'b0};  
                    if (bit\_cnt == 0) begin  
                        cs <= 1;  
                        done <= 1;  
                        state <= IDLE;  
                    end else begin  
                        bit\_cnt <= bit\_cnt - 1;  
                        state <= LOAD;  
                    end  
                end  
            endcase  
        end  
    end  
endmodule

**4.SSD1331\_INIT**

module ssd1331\_init(  
// SSD1331 OLED initialization FSM (NUM\_CMDS = 37)  
    input wire clk,  
    input wire rst,  
    output reg [7:0] spi\_data,  
    output reg spi\_start,  
    input wire spi\_done,  
    output reg dc,  
    output reg res\_n,  
    output reg init\_done  
);  
    parameter S\_IDLE     = 3'd0;  
    parameter S\_RESET    = 3'd1;  
    parameter S\_WAIT     = 3'd2;  
    parameter S\_SEND\_CMD = 3'd3;  
    parameter S\_WAIT\_SPI = 3'd4;  
    parameter S\_DONE     = 3'd5;  
  
    reg [2:0] state;  
    parameter NUM\_CMDS = 37;  
    reg [7:0] init\_cmds [0:NUM\_CMDS-1];  
    reg [5:0] cmd\_idx;  
    reg [19:0] rst\_cnt; // 100MHz  
    parameter RST\_HOLD = 20'd200\_000; // 2ms @ 100MHz  
  
    initial begin  
        init\_cmds[0]  = 8'hAE; // Display off  
        init\_cmds[1]  = 8'hA0; // Set re-map & color depth  
        init\_cmds[2]  = 8'h72; // RGB color  
        init\_cmds[3]  = 8'hA1; // Set display start line  
        init\_cmds[4]  = 8'h00;  
        init\_cmds[5]  = 8'hA2; // Set display offset  
        init\_cmds[6]  = 8'h00;  
        init\_cmds[7]  = 8'hA4; // Normal display  
        init\_cmds[8]  = 8'hA8; // Set multiplex ratio  
        init\_cmds[9]  = 8'h3F;  
        init\_cmds[10] = 8'hAD; // Set master config  
        init\_cmds[11] = 8'h8E;  
        init\_cmds[12] = 8'hB0; // Power save  
        init\_cmds[13] = 8'h0B;  
        init\_cmds[14] = 8'hB1; // Phase 1 & 2 period  
        init\_cmds[15] = 8'h31;  
        init\_cmds[16] = 8'hB3; // Display clock div  
        init\_cmds[17] = 8'hF0;  
        init\_cmds[18] = 8'h8A; // Precharge  
        init\_cmds[19] = 8'h64;  
        init\_cmds[20] = 8'h8B; // Precharge   
        init\_cmds[21] = 8'h78;  
        init\_cmds[22] = 8'h8C; // Precharge   
        init\_cmds[23] = 8'h64;  
        init\_cmds[24] = 8'hBB; // Precharge level  
        init\_cmds[25] = 8'h3A;  
        init\_cmds[26] = 8'hBE; // VCOMH  
        init\_cmds[27] = 8'h3E;  
        init\_cmds[28] = 8'h87; // Master current  
        init\_cmds[29] = 8'h06;  
        init\_cmds[30] = 8'h81; // Contrast A (Red)  
        init\_cmds[31] = 8'h91;  
        init\_cmds[32] = 8'h82; // Contrast B (Green)  
        init\_cmds[33] = 8'h50;  
        init\_cmds[34] = 8'h83; // Contrast C (Blue)  
        init\_cmds[35] = 8'h7D;  
        init\_cmds[36] = 8'hAF; // Display ON  
    end  
    always @(posedge clk or posedge rst) begin  
        if (rst) begin  
            state     <= S\_RESET;  
            cmd\_idx   <= 0;  
            spi\_start <= 0;  
            spi\_data  <= 0;  
            dc        <= 0;  
            res\_n     <= 0;  
            init\_done <= 0;  
            rst\_cnt   <= 0;  
        end else begin  
           case (state)  
                S\_RESET: begin  
                    res\_n   <= 0;  
                    rst\_cnt <= rst\_cnt + 1;  
                    if (rst\_cnt > RST\_HOLD) begin  
                        res\_n   <= 1; // Release reset  
                        rst\_cnt <= 0;  
                        state   <= S\_WAIT;  
                    end  
                end  
                S\_WAIT: begin  
                    rst\_cnt <= rst\_cnt + 1;  
                    if (rst\_cnt > RST\_HOLD) begin  
                        rst\_cnt <= 0;  
                        state   <= S\_SEND\_CMD;  
                    end  
                end  
                S\_SEND\_CMD: begin  
                    if (cmd\_idx < NUM\_CMDS) begin  
                        spi\_data  <= init\_cmds[cmd\_idx];  
                        spi\_start <= 1;  
                        dc        <= 0;  
                        state     <= S\_WAIT\_SPI;  
                    end else begin  
                        state     <= S\_DONE;  
                        init\_done <= 1;  
                    end  
                end  
                S\_WAIT\_SPI: begin  
                    spi\_start <= 0;  
                    if (spi\_done) begin  
                        cmd\_idx <= cmd\_idx + 1;  
                        state   <= S\_SEND\_CMD;  
                    end  
                end  
                S\_DONE: begin  
                end  
                default: state <= S\_IDLE;  
            endcase  
        end  
    end  
endmodule

# Hardware Connections and the Constraint File:

HARDWARE CONNECTIONS :

|  |  |
| --- | --- |
| CS | R4 |
| MOSI | T3 |
| NC | N5 |
| SCK | N4 |
| GND | - |
| VCC | - |
| D/C | L5 |
| RES | M4 |
| VCCEN | K4 |
| PMODEN | L4 |
| GND | - |
| VCC | - |

**CONSTRAINTS FILE :**

set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in[1]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in[0]}]  
  
set\_property PACKAGE\_PIN U2 [get\_ports {logic\_in[1]}]  
set\_property PACKAGE\_PIN V2 [get\_ports {logic\_in[0]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  
set\_property IOSTANDARD LVCMOS33 [get\_ports cs]  
set\_property IOSTANDARD LVCMOS33 [get\_ports dc]  
set\_property IOSTANDARD LVCMOS33 [get\_ports mosi]  
set\_property IOSTANDARD LVCMOS33 [get\_ports pmoden]  
set\_property IOSTANDARD LVCMOS33 [get\_ports res\_n]  
set\_property IOSTANDARD LVCMOS33 [get\_ports rst]  
set\_property IOSTANDARD LVCMOS33 [get\_ports sclk]  
set\_property IOSTANDARD LVCMOS33 [get\_ports vccen]  
set\_property PACKAGE\_PIN F14 [get\_ports clk]  
set\_property PACKAGE\_PIN R4 [get\_ports cs]  
set\_property PACKAGE\_PIN L5 [get\_ports dc]  
set\_property PACKAGE\_PIN T3 [get\_ports mosi]  
set\_property PACKAGE\_PIN L4 [get\_ports pmoden]  
set\_property PACKAGE\_PIN M4 [get\_ports res\_n]  
set\_property PACKAGE\_PIN J1 [get\_ports rst]  
set\_property PACKAGE\_PIN N4 [get\_ports sclk]  
set\_property PACKAGE\_PIN K4 [get\_ports vccen]  
  
set\_property DRIVE 12 [get\_ports sclk]  
  
  
set\_property PACKAGE\_PIN J2 [get\_ports speed\_switch]  
set\_property IOSTANDARD LVCMOS33 [get\_ports speed\_switch]  
  
set\_property PACKAGE\_PIN U1 [get\_ports {logic\_in[2]}]  
set\_property PACKAGE\_PIN T2 [get\_ports {logic\_in[3]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in[3]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in[2]}]  
  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in\_external[3]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in\_external[2]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in\_external[1]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports {logic\_in\_external[0]}]  
set\_property IOSTANDARD LVCMOS33 [get\_ports freeze\_button]  
set\_property IOSTANDARD LVCMOS33 [get\_ports mode\_select]  
set\_property PACKAGE\_PIN V2 [get\_ports {logic\_in\_external[3]}]  
set\_property PACKAGE\_PIN U2 [get\_ports {logic\_in\_external[2]}]  
set\_property PACKAGE\_PIN U1 [get\_ports {logic\_in\_external[1]}]  
set\_property PACKAGE\_PIN T2 [get\_ports {logic\_in\_external[0]}]  
set\_property PACKAGE\_PIN T1 [get\_ports mode\_select]  
set\_property PACKAGE\_PIN R2 [get\_ports freeze\_button]

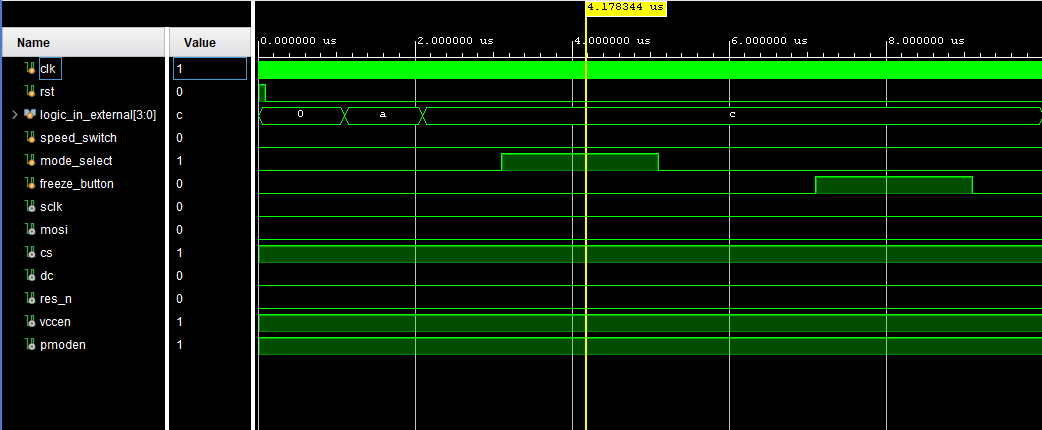
# Results

# Waveform Visualization :

TESTBENCH AND THE SIMULATION WAVEFORM :

module top\_logic\_analyzer\_tb;  
    // Inputs  
    reg clk;  
    reg rst;  
    reg [3:0] logic\_in\_external;  
    reg speed\_switch;  
    reg mode\_select;  
    reg freeze\_button;  
  
    // Outputs  
    wire sclk;  
    wire mosi;  
    wire cs;  
    wire dc;  
    wire res\_n;  
    wire vccen;  
    wire pmoden;  
  
    // Instantiate the DUT  
    top\_logic\_analyzer uut (  
        .clk(clk),  
        .rst(rst),  
        .logic\_in\_external(logic\_in\_external),  
        .speed\_switch(speed\_switch),  
        .mode\_select(mode\_select),  
        .freeze\_button(freeze\_button),  
        .sclk(sclk),  
        .mosi(mosi),  
        .cs(cs),  
        .dc(dc),  
        .res\_n(res\_n),  
        .vccen(vccen),  
        .pmoden(pmoden)  
    );  
  
    // Clock generation: 100 MHz  
    initial clk = 0;  
    always #5 clk = ~clk;  
    // Stimulus  
    initial begin  
        // Start with reset  
        rst = 1;  
        mode\_select = 0;  
        freeze\_button = 0;  
        logic\_in\_external = 4'b0000;  
        speed\_switch = 0;  
  
        #100;  
        rst = 0;  
        // External signals active  
        #1000 logic\_in\_external = 4'b1010;  
        #1000 logic\_in\_external = 4'b1100;  
  
        // =Toggle mode\_select HIGH=  
        #1000 mode\_select = 1;  
        #2000 mode\_select = 0;  
  
        // === Press freeze\_button ===  
        #2000 freeze\_button = 1;  
        #2000 freeze\_button = 0;  
        // === Toggle again ===  
        #2000 mode\_select = 1;  
        #2000 freeze\_button = 1;  
        #2000 mode\_select = 0;  
        #2000 freeze\_button = 0;  
        #10000;  
        $finish;  
    end  
endmodule

SIMULATION WAVEFORMS :



1. **Conclusion :**

The successful implementation of a logic analyzer display system using the Spartan-7 FPGA (XC7S50CSG324-1) and the Pmod OLEDrgb module shows the power of FPGA-based embedded design. This project has achieved its primary goal: to develop a compact and standalone system capable of capturing and visually representing digital signals in real-time without the support of an external computer or display interface.

By using Verilog HDL, the system efficiently captures 4-bit input signals, processes them in a controlled pipeline, and transmits the resulting waveform data to a 96x64 RGB OLED display via SPI. The design incorporates a clean and modular architecture that separates logic capture, waveform rendering, and SPI communication—each of which was tested.

Future enhancements to the system could involve expanding the number of input channels, using higher-resolution displays for better waveform detail, using frequency generator for producing variety of signals in real time, integrating USB or SD card data logging, or introducing on-screen menus for selecting logic channels, zoom level, or display modes.

In conclusion, the project provides a solid foundation for future work in embedded signal visualization, bridging the gap between academic learning and practical hardware implementation. It proves that with careful design, even minimal hardware resources can be leveraged to create robust, real-time digital tools.

1. **References :**
2. <https://www.realdigital.org/doc/02013cd17602c8af749f00561f88ae21> - The FPGA board manual was referred.
3. [pmodoledrgb\_rm-934951.pdf](https://www.mouser.com/datasheet/2/690/pmodoledrgb_rm-934951.pdf) – Pmod OLEDrgb module reference manual.
4. Various pre-existing GitHub projects related to Pmod OLEDrgb module integration.