3EC503CC24-Computer Architecture

Special Assignment Report

Implementation and Evaluation of UART Communication on FPGA

21BEC062-Aum Makwana

22BEC076-Neel Patel

Table of Contents:

ABSTRACT:

This report presents a comprehensive study on the implementation and evaluation of Universal Asynchronous Receiver/Transmitter (UART) communication between a Field Programmable Gate Array (FPGA) and an ESP32 microcontroller. The integration of these technologies facilitates bidirectional serial communication, which is essential for various embedded applications. The UART protocol's simplicity and versatility make it an ideal choice for interfacing different devices. This study details the design, implementation, and testing phases, demonstrating the effectiveness of UART communication on FPGA platforms. The results indicate reliable data transmission and reception, showcasing the potential for broader applications in embedded systems.

KEYWORDS: UART, FPGA, Communication Protocol, Serial Transmission, ESP32, Embedded Systems

LITERATURE SURVEY:

The UART protocol has been widely adopted in digital communication due to its asynchronous nature, allowing devices to communicate without a shared clock signal. This feature is particularly beneficial in environments where devices operate at different clock speeds. Recent advancements in FPGA technology have enabled the implementation of UART communication in various applications, including IoT devices and embedded systems.

Several studies have explored the integration of UART with FPGAs. For instance, research by Mughal (2019) and Mughal (2020) demonstrated the practical implementation of UART transmit and receive modules on FPGA platforms. These studies highlighted the importance of UART in facilitating communication between microcontrollers and FPGAs, emphasizing its role in enhancing system interoperability.

LIMITATIONS AND DRAWBACKS OF CURRENT TECHNOLOGY:

Despite the advantages of UART communication, several limitations exist:

- **Speed Constraints**: UART operates at lower speeds compared to other protocols like SPI and I2C, which may not be suitable for high-speed applications.
- **Limited Distance**: The effective communication distance is limited, typically to a few meters, which can restrict its use in larger systems.
- **No Error Detection**: Standard UART does not include built-in error detection mechanisms, making it less reliable in noisy environments.

PROPOSED SOLUTION AND METHODOLOGY:

To address the limitations of UART communication, this study proposes a robust implementation strategy using FPGA technology. The methodology includes:

1. **Designing the Top Module:** A Verilog-based top module is created to manage UART communication, parameterized for clock frequency and baud rate.
2. **Transmitter and Receiver Modules:** Separate modules for transmission and reception are developed, ensuring efficient data handling.
3. **Integration with ESP32:** The ESP32 microcontroller is utilized as a bridge for data exchange, connecting the FPGA to external devices.
4. **Testing and Validation:** The system is tested using the Arduino IDE's serial terminal to verify data transmission and reception.

BLOCK DIAGRAM:

TTL & RTL Of UART RX:

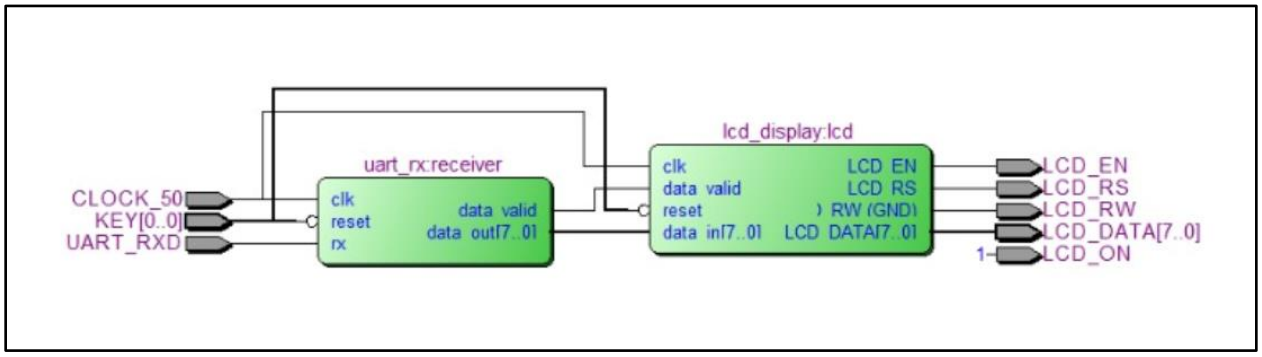


Fig.1 - RTL

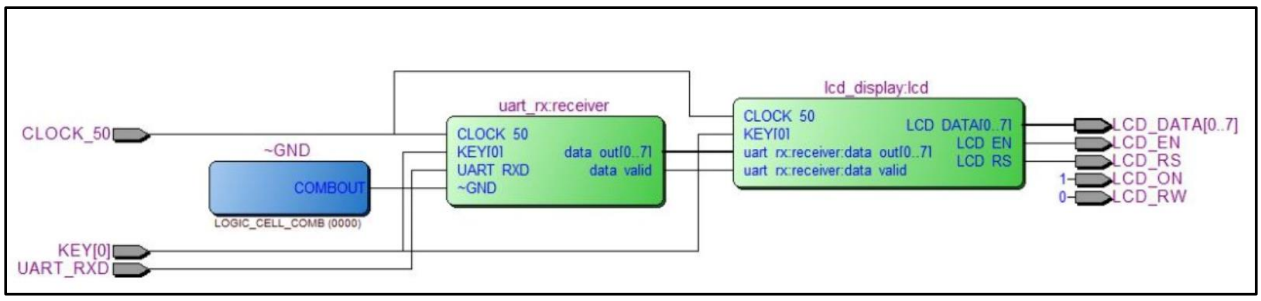


Fig.2 – RTL

TTL & RTL Of UART TX:

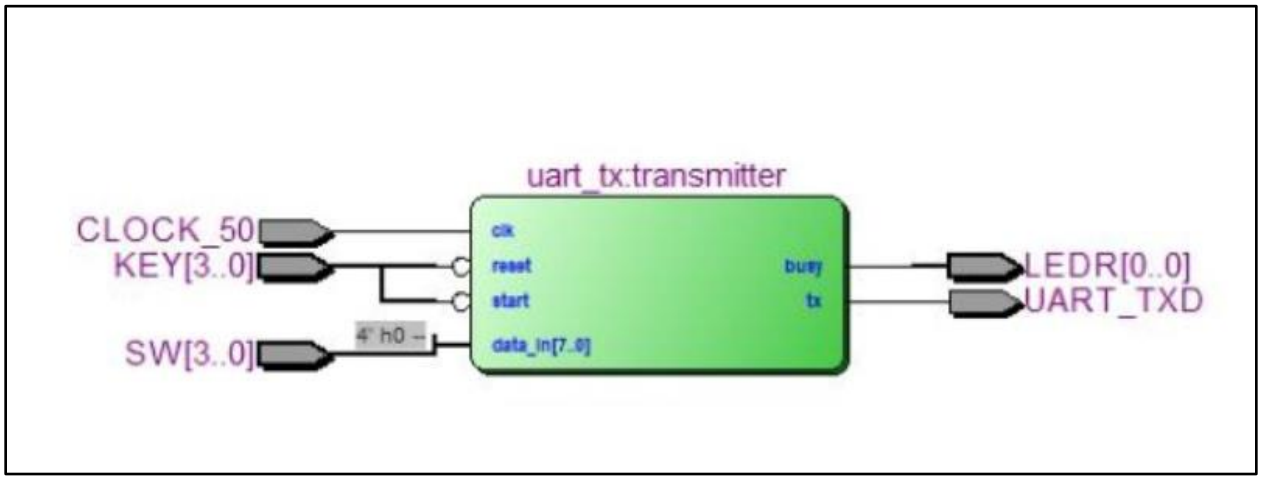


Fig.1 – RTL

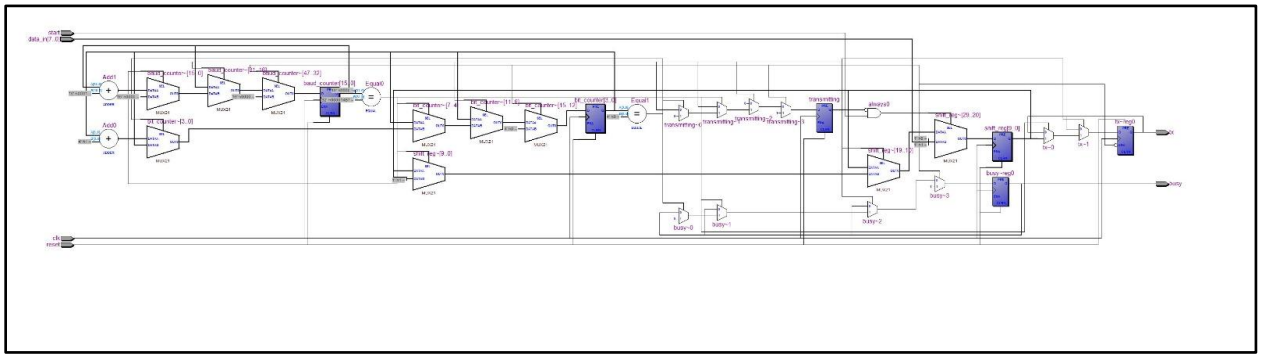


Fig.2 – RTL (Expanded view)

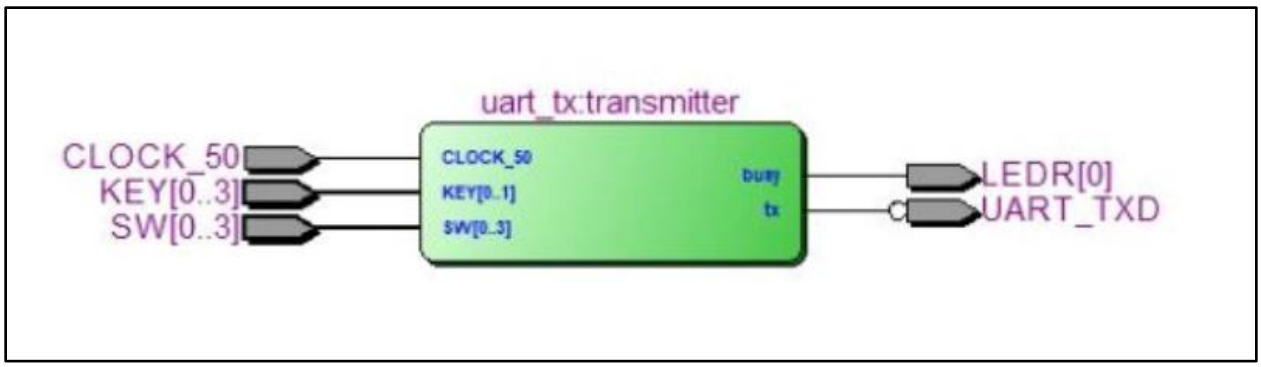


Fig.3 - TTL

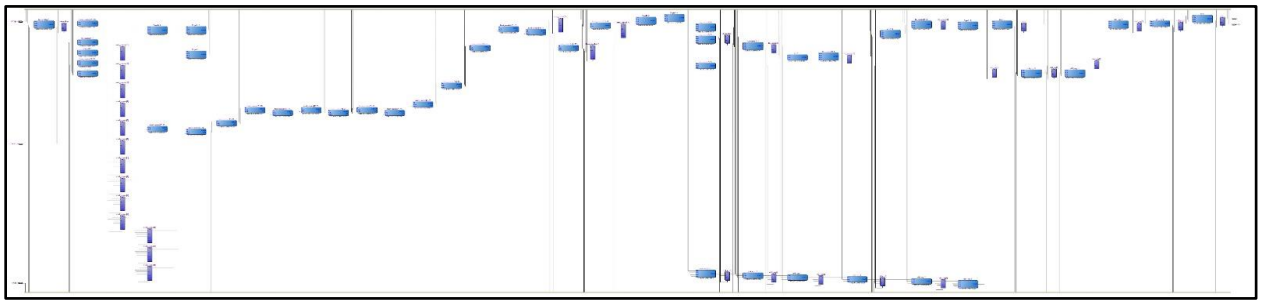
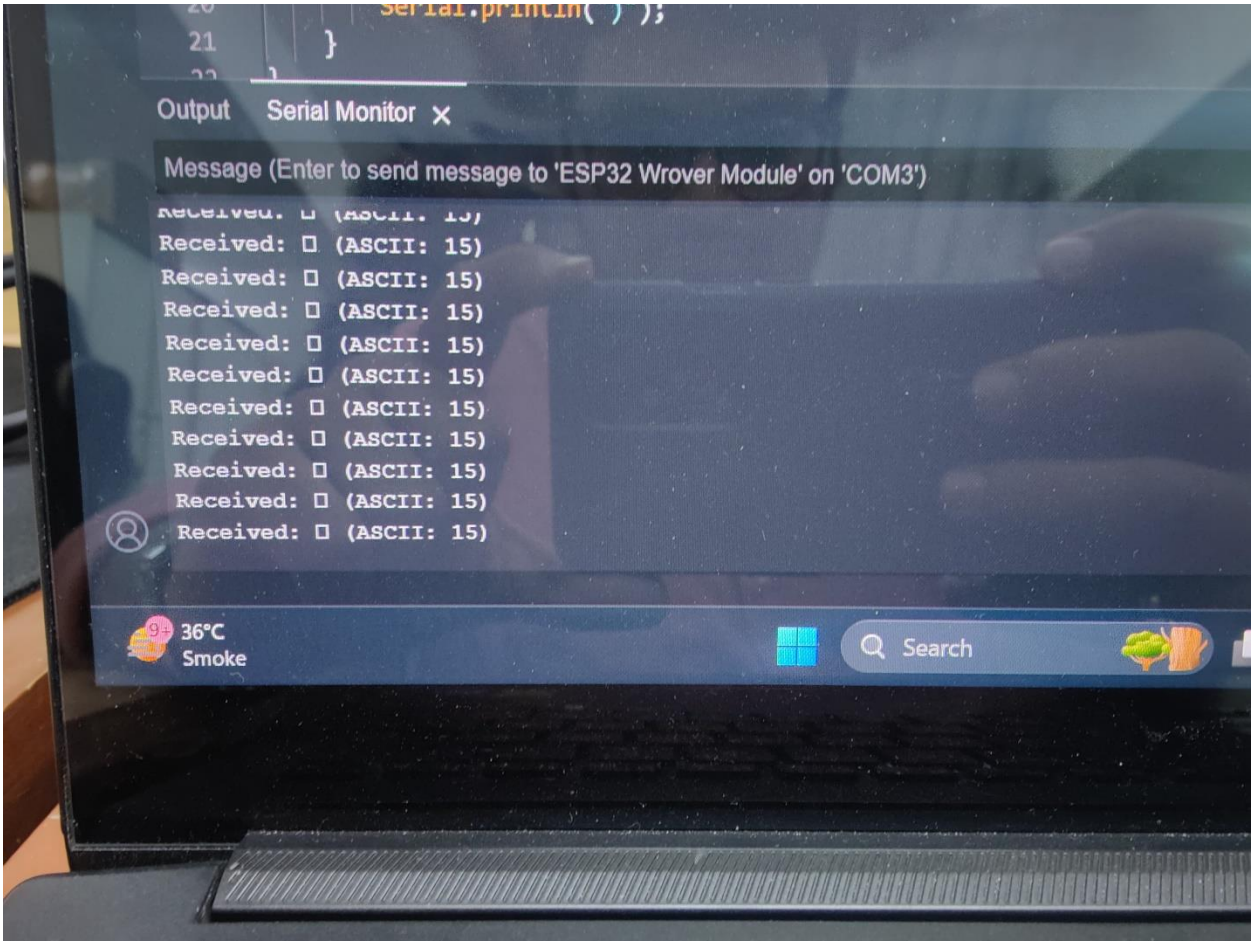
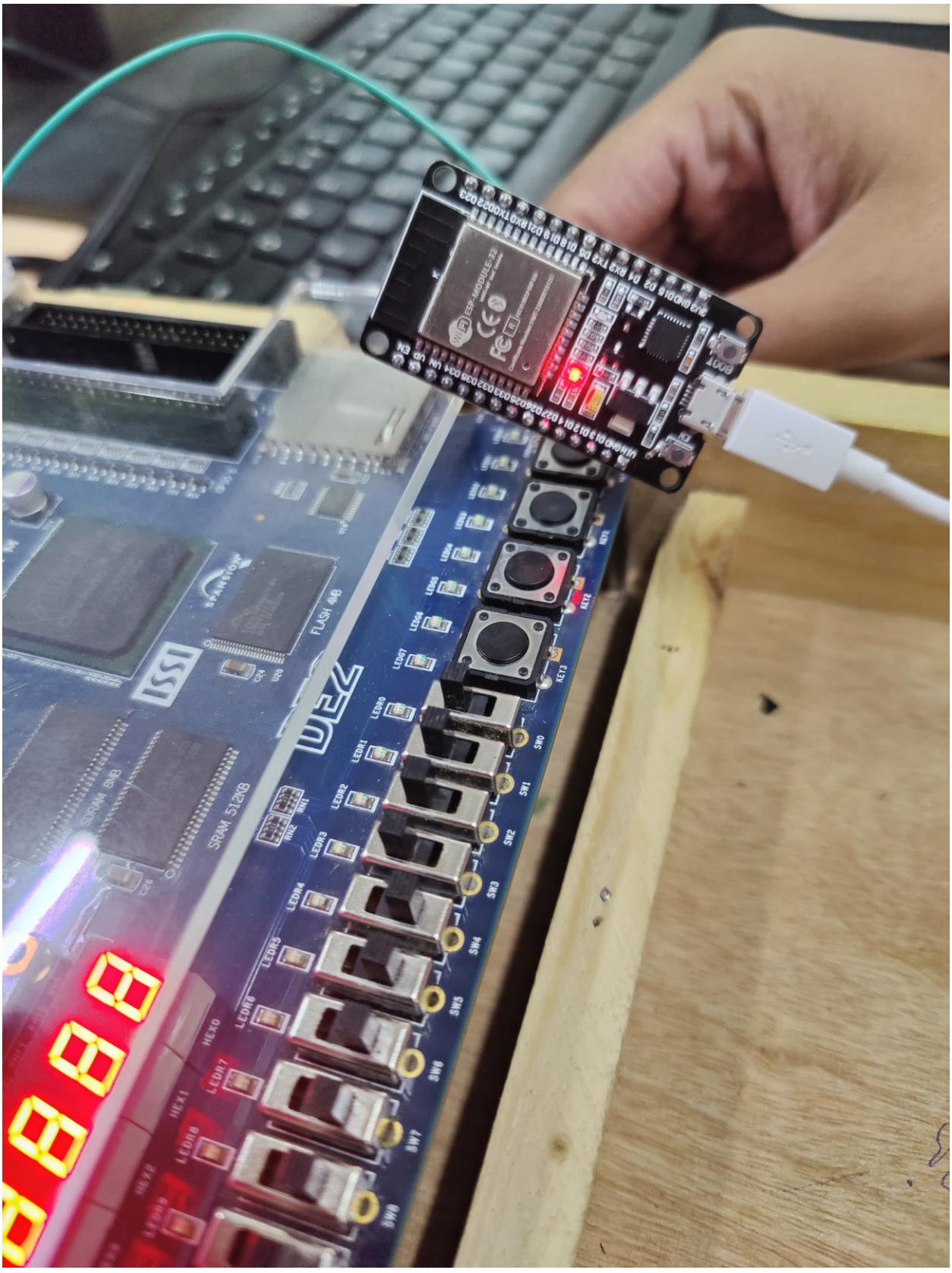


Fig.4 – TTL (Expanded view)

SIMULATION AND ANALYSIS:




ESP-WROOM-32
ISSI
SPANSION
FLASH 4MB
SRAM 512KB
SDRAM 8MB

CONCLUSION:

The study successfully demonstrates the implementation and evaluation of UART communication on FPGA platforms. The results highlight the reliability and effectiveness of this communication protocol in facilitating data exchange between FPGAs and microcontrollers. As embedded systems continue to evolve, the integration of UART communication will play a crucial role in enhancing connectivity and interoperability among devices.

REFERENCES:

1. Prof. Mughal, "UART TxD on FPGA," 2019. [Online]. Available: https://youtu.be/Fms2Qwkbu1g.
2. Prof. Mughal, "UART RxD on FPGA," 2020. [Online]. Available: https://youtu.be/XpfEHPg5AxU.

CODE:

Code for FPGA to Receive Data

```
module uart_rx (
    input wire clk,          // 50MHz clock
    input wire reset,        // Reset (active high)
    input wire rx,           // UART RX input from ESP32
    output reg [7:0] data_out,// Received byte
    output reg data_valid    // Pulse when data is ready
);
    parameter CLOCK_FREQ = 50000000;
    parameter BAUD_RATE = 115200;
    parameter BAUD_TICK = CLOCK_FREQ / BAUD_RATE;  // ~434
    reg [15:0] baud_counter;
```

```
reg [3:0] bit_counter;
reg [9:0] shift_reg;
reg receiving;
always @(posedge clk or posedge reset) begin
    if (reset) begin
        baud_counter <= 0;
        bit_counter <= 0;
        shift_reg <= 0;
        receiving <= 0;
        data_out <= 0;
            data_valid <= 0;
    end
    else begin
        if (!receiving && !rx) begin  // Start bit detected (low)
            receiving <= 1;
            baud_counter <= BAUD_TICK / 2;  // Sample mid-bit
            bit_counter <= 0;
        end
        else if (receiving) begin
            if (baud_counter == BAUD_TICK - 1) begin
                baud_counter <= 0;
                shift_reg <= {rx, shift_reg[9:1]};  // Shift in bits
                bit_counter <= bit_counter + 1;
                if (bit_counter == 9) begin  // 1 start + 8 data + 1 stop
                    receiving <= 0;
                    if (shift_reg[9] == 1) begin  // Valid stop bit
                        data_out <= shift_reg[8:1];
                        data_valid <= 1;
```

```
                end
            end
        end
        else begin
            baud_counter <= baud_counter + 1;
            data_valid <= 0;
        end
    end
    else data_valid <= 0;
  end
 end
endmodule
module lcd_display (
    input wire clk,
    input wire reset,
    input wire [7:0] data_in,
    input wire data_valid,
    output reg [7:0] LCD_DATA,  // LCD data bus
    output reg LCD_RS,         // Register Select
    output reg LCD_EN,         // Enable
    output reg LCD_RW          // Read/Write (0 for write)
);
    reg [31:0] counter;        // Timing counter
    reg [3:0] state;           // FSM state
    reg [15:0] display_data;   // Store number to display (up to 4 digits)
    // LCD timing parameters (50MHz clock)
    parameter T40US = 2000;    // 40us delay
    parameter T160US = 8000;   // 160us delay
```

```
initial begin
    LCD_DATA = 8'h00;
    LCD_RS = 0;
    LCD_EN = 0;
    LCD_RW = 0;
    counter = 0;
    state = 0;
    display_data = 0;
end
always @(posedge clk or posedge reset) begin
    if (reset) begin
        LCD_DATA <= 8'h00;
        LCD_RS <= 0;
        LCD_EN <= 0;
        LCD_RW <= 0;
        counter <= 0;
        state <= 0;
        display_data <= 0;
    end
    else begin
        if (data_valid && data_in >= 8'h30 && data_in <= 8'h39) begin
            // Accumulate digits (assuming single-digit input for simplicity)
            display_data <= (display_data * 10) + (data_in - 8'h30);
        end
        case (state)
            0: begin  // Initialization: Function Set (8-bit, 2 lines)
                LCD_RS <= 0;
                LCD_DATA <= 8'h38;
```

```
    LCD_EN <= 1;
    counter <= counter + 1;
    if (counter == T40US) state <= 1;
end
1: begin
    LCD_EN <= 0;
    counter <= 0;
    state <= 2;
end
2: begin  // Display On, Cursor Off
    LCD_RS <= 0;
    LCD_DATA <= 8'h0C;
    LCD_EN <= 1;
    counter <= counter + 1;
    if (counter == T40US) state <= 3;
end
3: begin
    LCD_EN <= 0;
    counter <= 0;
    state <= 4;
end
4: begin  // Clear Display
    LCD_RS <= 0;
    LCD_DATA <= 8'h01;
    LCD_EN <= 1;
    counter <= counter + 1;
    if (counter == T160US) state <= 5;
end
```

```
5: begin
    LCD_EN <= 0;
    counter <= 0;
    state <= 6;
end
6: begin  // Display Number (e.g., first digit)
    LCD_RS <= 1;
    LCD_DATA <= (display_data / 1000) ? (8'h30 + (display_data / 1000)) : 8'h20;
    LCD_EN <= 1;
    counter <= counter + 1;
    if (counter == T40US) state <= 7;
end
7: begin
    LCD_EN <= 0;
    counter <= 0;
    state <= 8;
end
8: begin  // Second digit
    LCD_RS <= 1;
    LCD_DATA <= (8'h30 + ((display_data % 1000) / 100));
    LCD_EN <= 1;
    counter <= counter + 1;
    if (counter == T40US) state <= 9;
end
9: begin
    LCD_EN <= 0;
    counter <= 0;
    state <= 10;
```

```
        end
        10: begin  // Third digit
            LCD_RS <= 1;
            LCD_DATA <= (8'h30 + ((display_data % 100) / 10));
            LCD_EN <= 1;
            counter <= counter + 1;
            if (counter == T40US) state <= 11;
        end
        11: begin
            LCD_EN <= 0;
            counter <= 0;
            state <= 12;
        end
        12: begin  // Fourth digit
            LCD_RS <= 1;
            LCD_DATA <= (8'h30 + (display_data % 10));
            LCD_EN <= 1;
            counter <= counter + 1;
            if (counter == T40US) state <= 13;
        end
        13: begin
            LCD_EN <= 0;
            counter <= 0;
            state <= 6;  // Loop back to update display
        end
        default: state <= 0;
    endcase
end
```

```
    end
endmodule
module de2_top (
    input CLOCK_50,
    input [0:0] KEY,       // KEY[0] for reset
    input UART_RXD,        // Pin D25
    output [7:0] LCD_DATA,
    output LCD_RS,
    output LCD_EN,
    output LCD_RW,
    output LCD_ON          // LCD power
);
    wire reset = ~KEY[0];
    wire [7:0] rx_data;
    wire data_valid;
    assign LCD_ON = 1;  // Always on
    uart_rx receiver (
        .clk(CLOCK_50),
        .reset(reset),
        .rx(UART_RXD),
        .data_out(rx_data),
        .data_valid(data_valid)
    );
    lcd_display lcd (
        .clk(CLOCK_50),
        .reset(reset),
        .data_in(rx_data),
        .data_valid(data_valid),
```

```
        .LCD_DATA(LCD_DATA),
        .LCD_RS(LCD_RS),
        .LCD_EN(LCD_EN),
        .LCD_RW(LCD_RW)
    );
Endmodule
```

Code for ESP32 to transmit data

```
#define BAUD_RATE 115200
void setup() {
    // Initialize Serial (UART0) for Serial Monitor
    Serial.begin(BAUD_RATE);
    // Initialize UART1 for transmitting to FPGA (TX1 on GPIO10, RX disabled)
    Serial1.begin(BAUD_RATE, SERIAL_8N1, -1, 10);  // RX = -1 (unused), TX = GPIO10
    delay(100);
    Serial.println("ESP32 Ready");
    Serial.println("Enter a number to send to FPGA:");
}
void loop() {
    // Check if data is available from Serial Monitor
    if (Serial.available()) {
        String input = Serial.readStringUntil('\n');  // Read until newline
        input.trim();  // Remove leading/trailing whitespace
        if (input.length() > 0) {
            // Send the input string via UART1 to FPGA
```

```
            Serial1.print(input);
            Serial1.print('\n');  // Add newline as delimiter
            // Echo to Serial Monitor
            Serial.print("Sent to FPGA: ");
            Serial.println(input);
        }
    }
}
```

Code for FPGA to transmit data

```
module uart_tx (
    input wire clk,           // 50MHz system clock (DE2 default)
    input wire reset,         // Reset signal (active high)
    input wire start,         // Start transmission signal
    input wire [7:0] data_in, // 8-bit data to transmit
    output reg tx,            // UART TX output to ESP32
    output reg busy           // Transmission in progress flag
);
    // Parameters for 115200 baud rate with 50MHz clock
    parameter CLOCK_FREQ = 50000000;    // 50MHz
    parameter BAUD_RATE = 9600;
    parameter BAUD_TICK = CLOCK_FREQ / BAUD_RATE;  // ~434 cycles per bit
    // Internal registers
    reg [15:0] baud_counter;    // Counter for baud rate timing
    reg [3:0] bit_counter;      // Count transmitted bits (0-9)
    reg [9:0] shift_reg;        // Shift register (start + 8 data + stop)
    reg transmitting;           // Transmission state
```

```
// Initial values
initial begin
    tx = 1'b1;              // Idle state is high
    busy = 1'b0;
    baud_counter = 16'b0;
    bit_counter = 4'b0;
    shift_reg = 10'b0;
    transmitting = 1'b0;
end

// UART Transmission Logic
always @(posedge clk or posedge reset) begin
    if (reset) begin
        tx <= 1'b1;
        busy <= 1'b0;
        baud_counter <= 16'b0;
        bit_counter <= 4'b0;
        shift_reg <= 10'b0;
        transmitting <= 1'b0;
    end
    else begin
        if (!transmitting && start) begin
            // Start new transmission
            shift_reg <= {1'b1, data_in, 1'b0};  // Stop bit, data, start bit
            transmitting <= 1'b1;
            busy <= 1'b1;
            baud_counter <= 16'b0;
            bit_counter <= 4'b0;
```

```
        end
        else if (transmitting) begin
            if (baud_counter == BAUD_TICK - 1) begin
                // Time to shift next bit
                tx <= shift_reg[0];
                shift_reg <= {1'b1, shift_reg[9:1]};  // Shift right, pad with 1
                baud_counter <= 16'b0;

                if (bit_counter == 4'd9) begin
                    // Finished transmitting all bits
                    transmitting <= 1'b0;
                    busy <= 1'b0;
                end
                else begin
                    bit_counter <= bit_counter + 1;
                end
            end
            else begin
                baud_counter <= baud_counter + 1;
            end
        end
    end

endmodule
// Top module
module de2 (
    input CLOCK_50,          // 50MHz clock on DE2
```

```
    input [3:0] KEY,          // KEY[0] for reset (active low on DE2)
    input [3:0] SW,           // Switches for data input
    output [0:0] LEDR,        // LED to show busy status
    output UART_TXD           // UART TX pin on DE2
);
    wire reset = ~KEY[0];     // KEY is active low, so invert
    wire [7:0] data = {4'b0, SW};  // Use 4 switches as data
    // Instantiate UART transmitter
    uart_tx transmitter (
        .clk(CLOCK_50),
        .reset(reset),
        .start(~KEY[1]),      // Use KEY[1] as start signal
        .data_in(data),
        .tx(UART_TXD),
        .busy(LEDR[0])
    );
Endmodule
```

Code for ESP32 to recieve data

```
#define BAUD_RATE 9600
void setup() {
    // Initialize UART2 with RX on GPIO16 (RX2) and TX disabled (no TX pin needed)
    Serial2.begin(BAUD_RATE, SERIAL_8N1, 16, -1);  // RX2 = GPIO16, TX = -1 (unused)
    delay(100);  // Give time for serial to stabilize
    // Use Serial (UART0) for output to terminal
    Serial.begin(BAUD_RATE);
    Serial.println("ESP32 Ready to Receive from FPGA on RX2 (GPIO16)");
}
void loop() {
```

```
    if (Serial2.available()) {  // Check UART2 (RX2) for data

        char receivedChar = Serial2.read();

        Serial.print("Received: ");
        Serial.print(receivedChar);      // Print character

        Serial.print(" (ASCII: ");

        Serial.print((int)receivedChar); // Print ASCII value

        Serial.println(")");
    }
}
```