Project Report:

Title:

“UART Design and Communication between PC and SP601 using USB-UART Bridge”

1. Objectives of the Project:

- To design and implement UART communication between a PC and the SP601 FPGA evaluation board.
- To use a USB-UART bridge for data transmission between the devices.
- To design custom transmitter and receiver modules in Verilog.
- To validate the design through a functional testbench simulation.

2. Introduction:

Universal Asynchronous Receiver-Transmitter (UART) is a widely used serial communication protocol for data exchange between devices. Unlike synchronous communication, UART does not require a shared clock signal. Instead, it uses start and stop bits for synchronization.

In this project, we aim to establish UART communication between a PC and an SP601 evaluation board, which is based on the Xilinx Spartan-6 FPGA, using a USB-UART bridge. This design will help demonstrate how UART communication works, utilizing a USB-to-UART converter to interface between the FPGA and the PC.

3. System Components:

- **SP601 Evaluation Board:**
  - **FPGA:** Xilinx Spartan-6
  - **UART Core:** Custom design or pre-built IP Core (e.g., Xilinx UARTLite)
- **PC:**
  - Running a terminal emulator (e.g., TeraTerm, PuTTY) or a custom application for UART communication.
- **USB-UART Bridge:**
  - A USB-to-UART converter (e.g., FTDI FT232, Silicon Labs CP210x) to translate USB signals to UART-compatible signals.

Download the PuTTY from Here

Download the TeraTerm from Here

Download the CP210x from Here

4. Design Steps:

4.1. Hardware Connections

- Connect the PC and SP601 board using the USB-UART bridge:
  - **TX (PC) ↔ RX (SP601)**
  - **RX (PC) ↔ TX (SP601)**

  - **GND ↔ GND**
- Ensure that the USB-UART bridge is recognized by the PC, and the appropriate drivers (e.g., CP210x) are installed.

4.2. UART Core on FPGA

- **Custom UART Design:**
  - Implement the UART protocol using Verilog/VHDL, including logic for baud rate generation, start/stop bits, and parity checks.
  - Use an internal clock (e.g., 50 MHz or 100 MHz) and divide it to generate the desired baud rate (e.g., 9600 or 115200 bps).
- **Pre-built UART IP Core (Optional):**
  - Use Xilinx's UARTLite core, configure it to match the baud rate, data bits, and parity settings of the PC, and map the IP to the FPGA pins for TX, RX, and clock.

4.3. UART Configuration

- Match the UART settings on both the PC and FPGA:
  - **Baud Rate:** 9600 or 115200 bps
  - **Data Bits:** 8
  - **Stop Bits:** 1
  - **Parity:** None
- Set the UART clock frequency using a clock divider or a Digital Clock Manager (DCM).

5. Transmitter Module:

The transmitter module sends data from the FPGA to the PC. The module design in Verilog is as follows:

```
module tx (
  input        reset,
  input        txclk,
  input        ld_tx_data,
  input  [7:0] tx_data,
  input        tx_enable,
  output reg   tx_out,
  output reg   tx_empty
);
  reg [7:0] tx_reg;
  reg [3:0] tx_cnt;

  always @ (posedge txclk or posedge reset) begin
    if (reset) begin
      tx_reg    <= 0;
      tx_empty  <= 1;
      tx_out    <= 1;
      tx_cnt    <= 0;
    end else begin
      if (ld_tx_data) begin
        if (tx_empty) begin
```

```
          tx_reg   <= tx_data;
          tx_empty <= 0;
        end
      end
      if (tx_enable && !tx_empty) begin
        tx_cnt <= tx_cnt + 1;
        if (tx_cnt == 0) tx_out <= 0; // Start bit
        if (tx_cnt > 0 && tx_cnt < 9) tx_out <= tx_reg[tx_cnt - 1]; //
Data bits
        if (tx_cnt == 9) begin
          tx_out <= 1; // Stop bit
          tx_cnt <= 0;
          tx_empty <= 1;
        end
      end
      if (!tx_enable) tx_cnt <= 0;
    end
  end
endmodule
```

<u>6. Receiver Module:</u>

The receiver module receives data from the PC and processes it in the FPGA. The design is as follows:

```
module rx (
  input         reset,
  input         rxclk,
  input         rx_enable,
  input         rx_in,
  input         uld_rx_data,
  output reg [7:0] rx_data,
  output reg    rx_empty
);
  reg [7:0] rx_reg;
  reg [3:0] rx_sample_cnt;
  reg [3:0] rx_cnt;
  reg       rx_busy;
  reg       rx_d1, rx_d2;

  always @ (posedge rxclk or posedge reset) begin
    if (reset) begin
      rx_reg        <= 0;
      rx_data       <= 0;
      rx_sample_cnt <= 0;
      rx_cnt        <= 0;
      rx_empty      <= 1;
      rx_d1         <= 1;
      rx_d2         <= 1;
      rx_busy       <= 0;
    end else begin
      rx_d1 <= rx_in;
      rx_d2 <= rx_d1;
      if (uld_rx_data) begin
        rx_data <= rx_reg;
        rx_empty <= 1;
      end
      if (rx_enable) begin
        if (!rx_busy && !rx_d2) begin
          rx_busy <= 1;
          rx_sample_cnt <= 1;
          rx_cnt <= 0;
        end
        if (rx_busy) begin
          rx_sample_cnt <= rx_sample_cnt + 1;
          if (rx_sample_cnt == 7) begin
            if ((rx_d2 == 1) && (rx_cnt == 0)) begin
              rx_busy <= 0;
            end else begin
              rx_cnt <= rx_cnt + 1;
              if (rx_cnt > 0 && rx_cnt < 9) rx_reg[rx_cnt - 1] <=
rx_d2;
              if (rx_cnt == 9) begin
                rx_busy <= 0;
                rx_empty <= 0;
              end
            end
          end
        end
      end
      if (!rx_enable) rx_busy <= 0;
    end
  end
endmodule
```

7. Top Module:

The top module integrates both the transmitter and receiver modules. It connects the signals between them and handles the overall communication.

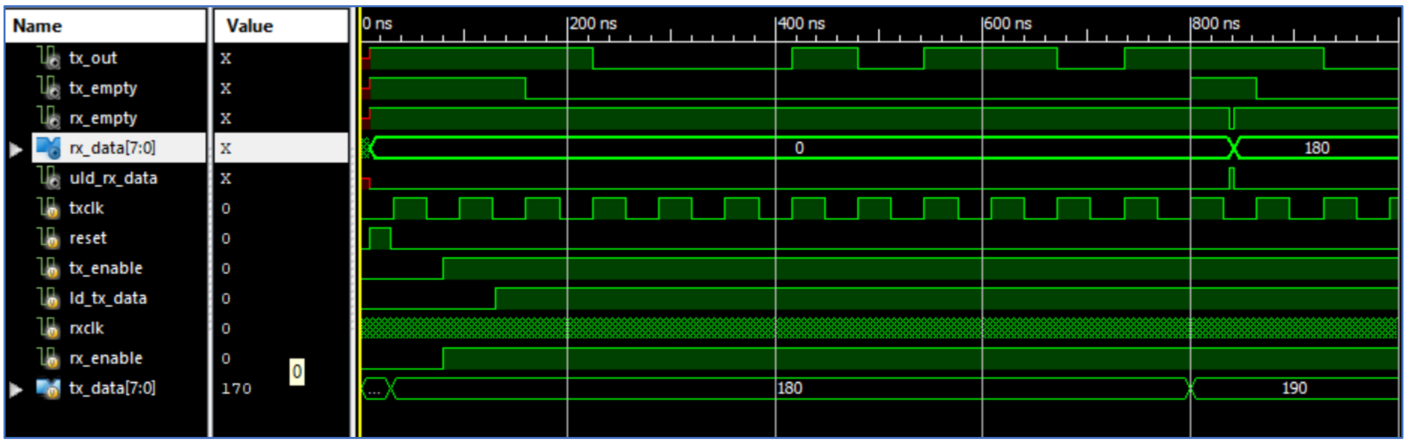
```
module uart (
  input        reset,
  input        txclk,
  input        ld_tx_data,
  input  [7:0] tx_data,
  input        tx_enable,
  output       tx_out,
  output       tx_empty,
  input        rxclk,
  input        uld_rx_data,
  output [7:0] rx_data,
  input        rx_enable,
  input        rx_in,
  output       rx_empty
);
  tx tx_inst (
    .reset(reset),
    .txclk(txclk),
    .ld_tx_data(ld_tx_data),
    .tx_data(tx_data),
    .tx_enable(tx_enable),
    .tx_out(tx_out),
    .tx_empty(tx_empty)
  );

  rx rx_inst (
    .reset(reset),
    .rxclk(rxclk),
    .rx_enable(rx_enable),
    .rx_in(rx_in),
    .uld_rx_data(uld_rx_data),
    .rx_data(rx_data),
    .rx_empty(rx_empty)
  );
endmodule
```

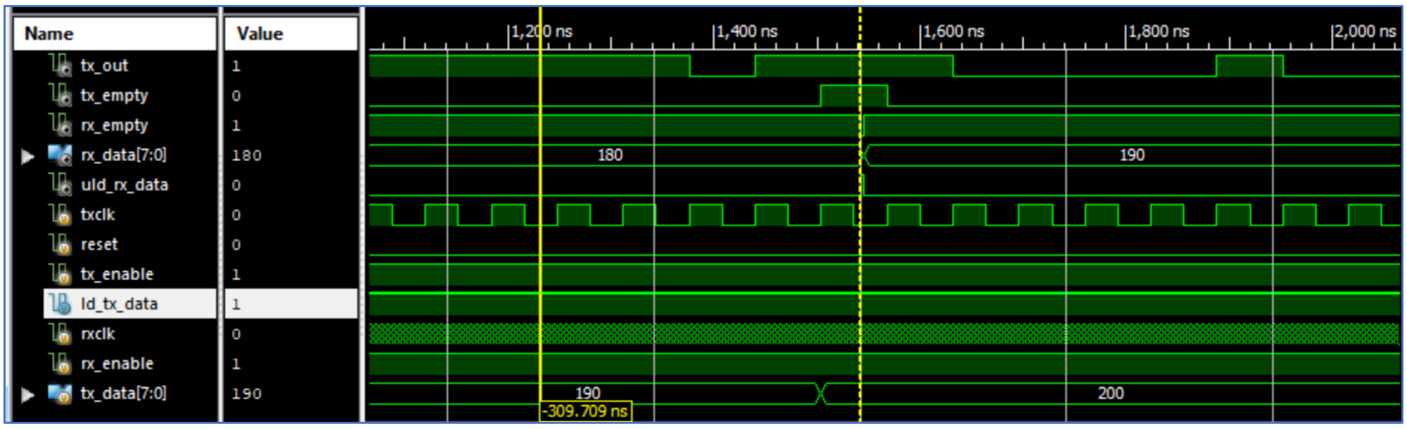
8. Testbench:

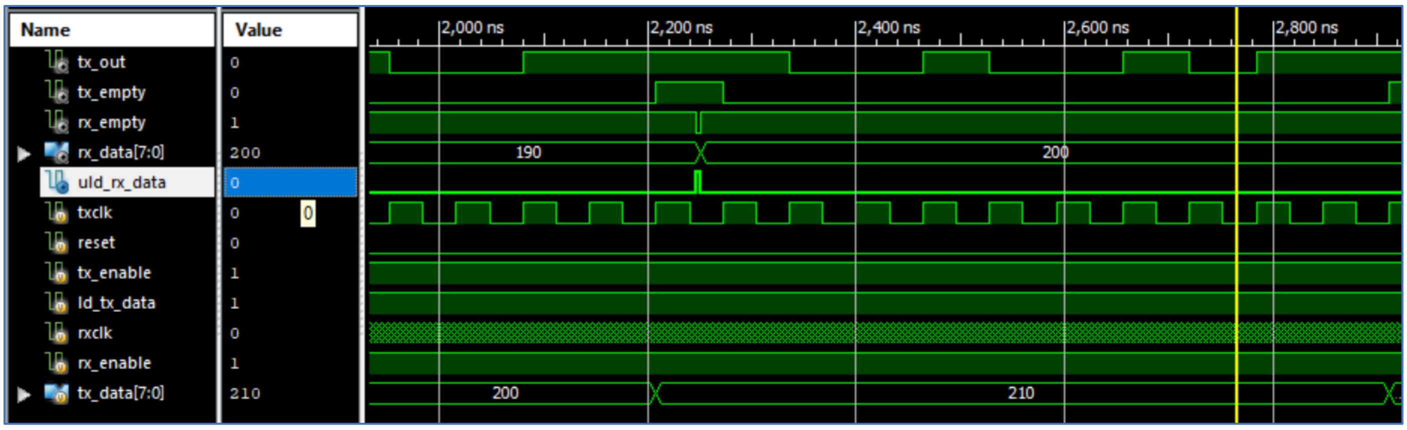
The testbench validates the functionality of the UART design. It generates clock signals and tests the transmission and reception of data.

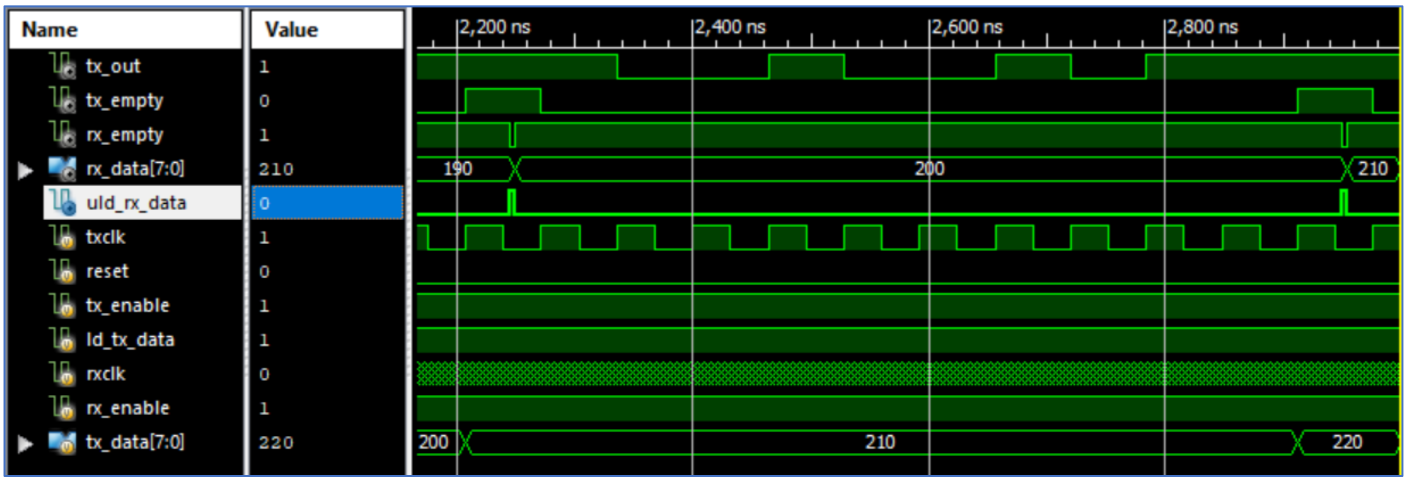
```
module uart_testbench();
  reg txclk, reset, tx_enable, ld_tx_data, rxclk, rx_enable;
  wire tx_out, tx_empty, rx_empty;
  reg [7:0] tx_data;
  wire [7:0] rx_data;
  wire uld_rx_data;

  uart uut (
    .reset(reset),
    .txclk(txclk),
    .ld_tx_data(ld_tx_data),
    .tx_data(tx_data),
    .tx_enable(tx_enable),
    .tx_out(tx_out),
    .tx_empty(tx_empty),
    .rxclk(rxclk),
    .uld_rx_data(uld_rx_data),
    .rx_data(rx_data),
    .rx_enable(rx_enable),
    .rx_in(tx_out),
    .rx_empty(rx_empty)
  );

  assign uld_rx_data = ~rx_empty;

  always @(reset, tx_empty) begin
    if (reset) begin
      tx_data = 170;
    end else if (tx_empty) begin
      tx_data = tx_data + 10;
    end
  end

  initial begin
    txclk = 0;
    forever #32 txclk = ~txclk;
  end
  initial begin
    rxclk = 0;
    forever #2 rxclk = ~rxclk;
  end

  initial begin
    reset = 0;
    ld_tx_data = 0;
    tx_enable = 0;
    rx_enable = 0;
    tx_data = 170;
    #10;
    reset = 1;
    #20;
    reset = 0;

    #50;
    tx_enable = 1;
    rx_enable = 1;

    #50;
    ld_tx_data = 1;
    #400;
  end
endmodule
```

9. Simulation Results:









9. Conclusion:

This project successfully demonstrates UART communication between a PC and the SP601 evaluation board using a USB-UART bridge. The custom-designed Verilog modules for both the transmitter and receiver allow efficient serial communication. Through the testbench, the system's functionality was verified, ensuring accurate data transmission and reception. It can be seen in the above simulation that the transmitted data (Tx) is correctly received and displayed by the receiver (Rx). This confirms that the design is functioning as expected, and the project should work effectively when implemented on the FPGA.

This project can be extended for real-world applications, including remote sensor data communication, embedded systems, and other FPGA-based designs. Future work could explore implementing error correction mechanisms, handshaking protocols, and interfacing with additional peripherals.