Indian Institute of Technology Bombay

EE 705 VLSI Design Lab

Project 1

Implementation of UART Lite Module

Team Members:

Bala Murugan S (24M1173)
Lakshmideep Chowdary M (24M1150)
Juhi Bharti (24M1144)

Course Instructor:

Prof. Laxmeesha Somappa

1 Design of UART_Lite Module

This module consists of an AXI4-Lite Slave Interface for accessing registers and facilitating data transfer between two devices. The following sections discusses the module definition and provide an explanation of its input and output signals.

1.1 Block Diagram of the UART_Lite Module

The AXI4-Lite signals on one side of this module are used to transmit and receive data from peripheral devices. These handshake signals follow the AXI4-Lite protocol.

The module consists of four registers: Receive FIFO, Transmit FIFO, Status, and Control registers. These registers are accessed using the following hexadecimal addresses: `00, 04, 08, 0C` respectively. Among these, the Receive FIFO and Status registers are read-only, meaning no write operations are permitted. Similarly, the Transmit FIFO and Control registers are write-only; any read operation on these registers results in `cfg_rdata_o = 32'b0`. The following image shows the block diagram of the given module,

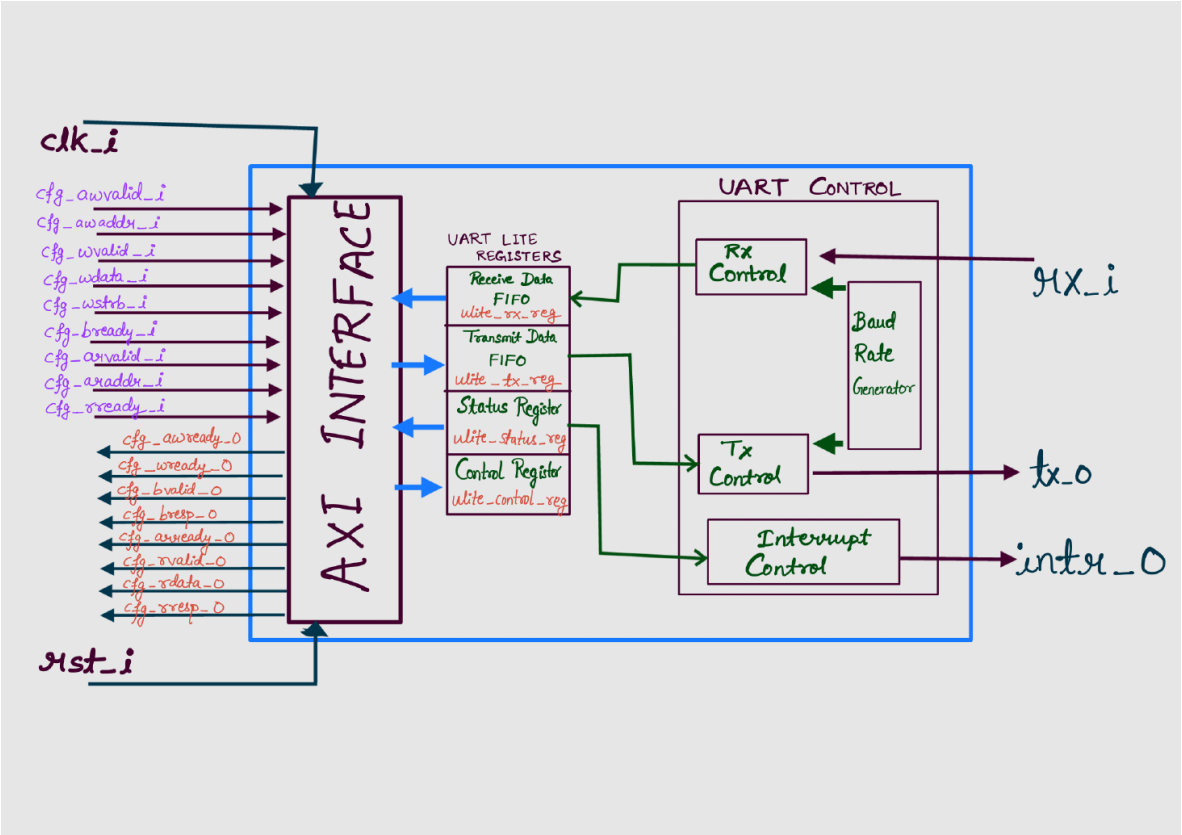

Figure 1: Block Diagram of UART_Lite

Baud Rate Generator

The UART_Lite module also consists of a Baud Rate generator which is used to synchronize data transmission and reception. The baud rate determines the speed of communication and is derived from the system clock. The baud rate generator divides the input clock frequency to produce the desired baud rate for serial communication. The division factor is configured based on a preset value, ensuring compatibility with standard baud rates such as 9600, 115200, etc. Baud rate in the code is fixed at 115200 for a system clock of 125MHz but it is parameterized in the module so different baud rates can also be employed. The generated baud clock controls the timing of data bits in the Transmit and Receive FIFOs. It ensures that each bit is transmitted at the correct interval, preventing data corruption. The baud rate generator's accuracy is critical in maintaining proper synchronization between the UART transmitter and receiver.

Use of Interrupt pin

The `intr_o` signal is connected to the Master device to indicate that a new unread data has been received i.e., the RX FIFO is filled and contains Valid Data. But the intrerrupt is not enabled by default. It should be enabled by explicitly writing `5'b10000` to the Control Register since the 5th index contains the Interrupt Enable Pin (IE), this is held high, The 1st index and the 2nd index are for RST_TX and RST_RX which are used for clearing RX FIFO and TX FIFO.

Function of tx_o and rx_i pins

The tx_o pin of one UART_lite module is connected with the rx_i pin of the other UART_lite module. The following image shows how the uart_lite module can be deployed and how it communicates with the other blocks,

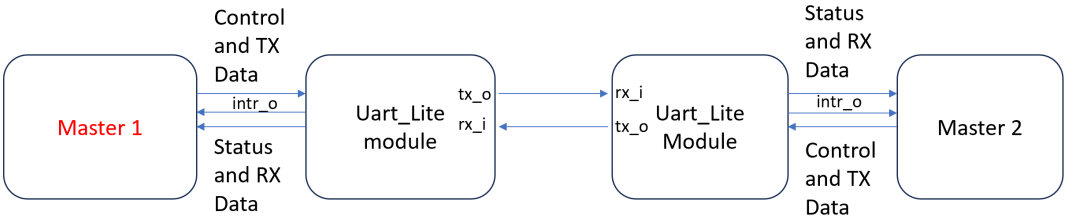

Figure 2: UART_Lite module with other blocks

1.2 Baud Rate Generator Code

Before discussing the details of UART_Lite Module, the baud rate generator code should be discussed. In this code, for a required baud rate, the system clock's frequency is used and based on that it will generate a baud clk internally, through which the tx and rx transmission happens. The system clk frequency and the required baud rate can be passed as parameters from the uart_lite module code.

```
module baud_rate_gen #(
    parameter SYSTEM_CLK_FREQ = 125_000_000, // 125 MHz system clock
    parameter OUT_CLK_FREQ = 115200        // Desired baud rate
)(
    input wire clk,      // System clock input
    output reg out_clk   // Generated baud clock output
);
localparam integer HALF_PERIOD = SYSTEM_CLK_FREQ / (2 * OUT_CLK_FREQ);
reg [31:0] counter = 32'b0;
reg flag = 1'b0;
always @(posedge clk) begin
    if (counter >= HALF_PERIOD - 1) begin
        counter <= 32'b0;
        out_clk <= ~out_clk;
    end
    else if (flag == 1'b0) begin
        out_clk <= 1;
        counter <= counter + 1;
        flag <= 1'b1;
    end
    else begin
        counter <= counter + 1;
    end
end
endmodule
```

2 UART_LITE CODE

This section includes the code for UART_Lite module and its explanation. In the below code the signal `clk_i` represents the system clk, and `rst_i` represents the reset signal. This signal is used to initialize the registers and outputs when `rst_i` is held `HIGH`. The signals `cfg_awaddr_i` and its respective valid and data signals are similar to AXI 4 lite, the `cfg_awvalid_i` should be held `HIGH` meaning that a valid address is placed on the bus. With the address the respective data is also written using `cfg_wdata_i` signal and the respective `cfg_wvalid_i` is also held `HIGH`. As already discussed write operation is permitted only for transmit FIFOs and Control Register. Similar to the write operation, read operation is also executed. The transmission and reception happens at the positive edges of `baud_clk`. The `tx_o` signal is always 1, when transmit FIFO is filled, it transmits a 0 for one baud_clk period, indicating a start of transmission, then every bit

starting from MSB is sent one by one at each edges. After completing the data transfer it again goes back to 1. In the receiver side, the `rx_i` detects the transmission when it receives 0 and receives every bit one by one from the transmitter and stores it in the RX FIFO. This RX FIFO can be read by placing proper address in the `cfg_araddr_i` bus. The following code implements the above discussed explanation.

```
module uart_lite (
    input            clk_i,
    input            rst_i,
    input            cfg_awvalid_i,
    input  [31:0]    cfg_awaddr_i,
    input            cfg_wvalid_i,
    input  [31:0]    cfg_wdata_i,
    input  [3:0]     cfg_wstrb_i,
    input            cfg_bready_i,
    input            cfg_arvalid_i,
    input  [31:0]    cfg_araddr_i,
    input            cfg_rready_i,
    input            rx_i,
    output reg         cfg_awready_o,
    output reg         cfg_wready_o,
    output reg         cfg_bvalid_o,
    output       [1:0] cfg_bresp_o,
    output reg         cfg_arready_o,
    output reg         cfg_rvalid_o,
    output reg [31:0]  cfg_rdata_o,
    output       [1:0] cfg_rresp_o,
    output reg         tx_o,
    output reg         intr_o
);


  // Data registers for RX and TX

  reg [7:0]  ulite_rx_reg;        // Holds received data
  reg [7:0]  ulite_tx_reg;        // Holds data to be transmitted
  reg [4:0]  ulite_status_reg;    // Holds status bits
  reg [4:0]  ulite_control_reg;   // Holds control bits

  `define ULITE_CONTROL_IE       4
  `define ULITE_CONTROL_RST_RX   1
  `define ULITE_CONTROL_RST_TX   0

  `define ULITE_STATUS_IE        4
  `define ULITE_STATUS_TXFULL    3
  `define ULITE_STATUS_TXEMPTY   2
  `define ULITE_STATUS_RXFULL    1
  `define ULITE_STATUS_RXVALID   0

  `define ULITE_RX       8'h00
  `define ULITE_TX       8'h04
```

```
`define ULITE_STATUS   8'h08
`define ULITE_CONTROL  8'h0C
reg [2:0] tindex;
reg       tcount;
reg [2:0] rindex;
reg       rcount;
reg txdone;
reg rxdone;
assign cfg_bresp_o = 2'b00;
assign cfg_rresp_o = 2'b00;
wire baud_clk;

// UART TX Shifting Process (operates on baud_clk)

always @(posedge baud_clk) begin
 if (rst_i) begin
  tx_o <= 1;
  tindex <= 3'd7;
  tcount <= 1'b0;
  txdone <= 1'b0;
 end

 else if (ulite_status_reg[`ULITE_STATUS_TXEMPTY] == 1'b0) begin
    if (tcount == 1'b0) begin
      tx_o   <= 1'b0;    // Send start bit (logic 0)
      tcount <= 1'b1;
    end
    else begin
      tx_o   <= ulite_tx_reg[tindex]; // Transmit MSB
      if (tindex == 3'd0) begin
        txdone <= 1'b1;
        tindex <= 3'd7;
        tcount <= 1'b0;
      end
      else begin
        tindex <= tindex - 1;
      end
    end
  end
  else begin
    tx_o   <= 1'b1; //stop bit and maintains 1
    txdone <= 1'b0;
    tindex <= 3'd7;
    tcount <= 1'b0;
  end
// UART RX Sampling Process (operates on baud_clk)
if (cfg_wdata_i[1])
          ulite_rx_reg <= 8'b0;  // Reset RX FIFO
        else
          ulite_rx_reg <= ulite_rx_reg;
if (rst_i) begin
      rcount           <= 1'b0;
```

```
      rindex           <= 3'd7;
      ulite_rx_reg  <= 8'b0;
      rxdone           <= 1'b0;
  end
  else begin
     if (rx_i != 1'b1 && rcount == 1'b0) begin

       // Detect start bit (rx_i goes low)
       rcount <= 1'b1;
     end
     else if (rcount == 1'b1 && rindex > 0) begin
       ulite_rx_reg[rindex] <= rx_i;
       rindex <= rindex - 1;
     end
     else if (rindex == 3'd0) begin
       ulite_rx_reg[rindex] <= rx_i;
       rcount <= 1'b0;
       rindex <= 3'd7;
       rxdone <= 1'b1;
     end
     else begin
       rcount <= 1'b0;
       rindex <= 3'd7;
       rxdone <= 1'b0;
     end
   end
end

// AXI-Lite Write TX Buffer

always @(posedge clk_i) begin
  if (rst_i) begin
    ulite_tx_reg             <= 8'b0;
    ulite_status_reg         <= 5'b00100;
    ulite_control_reg        <= 5'b00000;
    cfg_awready_o            <= 1'b0;
    cfg_wready_o             <= 1'b0;
    cfg_bvalid_o             <= 1'b0;
  end
  else if (cfg_awvalid_i && cfg_wvalid_i) begin
    case (cfg_awaddr_i[7:0])

      `ULITE_TX: begin

        // Accept new TX data only if transmitter is idle.

        if (ulite_status_reg[`ULITE_STATUS_TXEMPTY] == 1'b1) begin
          cfg_awready_o <= 1'b1;
          cfg_wready_o  <= 1'b1;
          ulite_tx_reg  <= cfg_wdata_i[7:0];    // Latch TX data
          ulite_status_reg[`ULITE_STATUS_TXEMPTY] <= 1'b0;
          ulite_status_reg[`ULITE_STATUS_TXFULL]  <= 1'b1;
```

```
        end
        else begin
          cfg_awready_o <= 1'b0;
          cfg_wready_o  <= 1'b0;
        end
      end

      `ULITE_CONTROL: begin
        cfg_awready_o <= 1'b1;
        cfg_wready_o  <= 1'b1;
        ulite_control_reg[`ULITE_CONTROL_IE]     <= cfg_wdata_i[4];
        ulite_control_reg[`ULITE_CONTROL_RST_RX] <= cfg_wdata_i[1];
        ulite_control_reg[`ULITE_CONTROL_RST_TX] <= cfg_wdata_i[0];
        if (cfg_wdata_i[4])
          ulite_status_reg[`ULITE_STATUS_IE] <= 1;
        else
          ulite_status_reg[`ULITE_STATUS_IE] <= 0;
        if (cfg_wdata_i[0]) begin
          ulite_tx_reg <= 8'b0;  // Reset TX FIFO
          ulite_status_reg[`ULITE_STATUS_TXEMPTY] <= 1;
          ulite_status_reg[`ULITE_STATUS_TXFULL]  <= 0;
        end
        else begin
          ulite_tx_reg <= ulite_tx_reg;  // Reset TX FIFO
        end
      end
      default: begin
        cfg_awready_o <= 1'b0;
        cfg_wready_o  <= 1'b0;
        cfg_bvalid_o  <= 1'b0;
      end
    endcase
    if (cfg_bready_i == 1'b1)
      cfg_bvalid_o <= 1'b1;
    else
       cfg_bvalid_o <= 1'b0;
  end
  else if (txdone) begin
      ulite_status_reg[`ULITE_STATUS_TXEMPTY] <= 1; // Mark as idle
      ulite_status_reg[`ULITE_STATUS_TXFULL]  <= 0;
  end
  else begin
    cfg_awready_o <= 1'b0;
    cfg_wready_o  <= 1'b0;
    cfg_bvalid_o  <= 1'b0;
  end
  if (rst_i) begin
    cfg_rvalid_o   <= 1'b0;
    cfg_rdata_o    <= 32'b0;
    cfg_arready_o  <= 1'b0;
    intr_o <= 1'b0;
  end
```

```
    else if (cfg_arvalid_i) begin
      cfg_arready_o <= 1'b1; // Always ready for a read address
      case (cfg_araddr_i[7:0])

        `ULITE_RX: begin
          if (cfg_rready_i == 1'b1 &&
              ulite_status_reg[`ULITE_STATUS_RXFULL] == 1'b1) begin
            cfg_rvalid_o <= 1'b1;
            cfg_rdata_o  <= {24'b0, ulite_rx_reg}; // Read RX data
            ulite_status_reg[`ULITE_STATUS_RXFULL]  <= 1'b0;
            ulite_status_reg[`ULITE_STATUS_RXVALID] <= 1'b0;
            intr_o <= 0;
          end
          else begin
            cfg_rvalid_o <= 1'b0;
          end
        end

        `ULITE_STATUS: begin
          if (cfg_rready_i == 1'b1) begin
            cfg_rvalid_o <= 1'b1;
            cfg_rdata_o  <= {24'b0, ulite_status_reg};
          end
          else begin
            cfg_rvalid_o <= 1'b0;
          end
        end

        `ULITE_CONTROL: begin
          if (cfg_rready_i == 1'b1) begin
            cfg_rvalid_o <= 1'b1;
            cfg_rdata_o  <= {27'b0, ulite_control_reg};
          end
          else begin
            cfg_rvalid_o <= 1'b0;
          end
        end
        default: begin
          cfg_rdata_o  <= 32'b0;
        end
      endcase
    end
    else if (rxdone) begin
        ulite_status_reg[`ULITE_STATUS_RXFULL]  <= 1;
        ulite_status_reg[`ULITE_STATUS_RXVALID] <= 1;
        if (ulite_status_reg[`ULITE_STATUS_IE] == 1'b1&
        (~ulite_status_reg[`ULITE_STATUS_RXFULL]))
             intr_o <= 1'b1;
        else
           intr_o <= 1'b0;
    end
    else begin
```

```
      cfg_rvalid_o  <= 1'b0;
      cfg_rdata_o   <= 32'b0;
      cfg_arready_o <= 1'b0;
      intr_o <= 1'b0;
    end
  end

  // Baud Rate Generator Instantiation

  baud_rate_gen #(
    .SYSTEM_CLK_FREQ(125_000_000),
    .OUT_CLK_FREQ(115200)
  ) u_baud (
    .clk(clk_i),
    .out_clk(baud_clk)
  );
endmodule
```

2.1 Testbench Code

Then we wrote a testbench to verify my design. The testbench first sends the 8 bit Hexadecimal data `E9` bit by bit through `rx_i`, then the transmission operation is checked by writing `32'd55` into the transmit FIFO. After this the Control Register is modified such that to enable interrupt signal so that functionality of interrrupt can also be verified. After that status register is read and the RX FIFO is also read to check whether the data is properly received through `rx_i`.

```
`timescale 1ns / 1ps
module uart_lite_tb;
// Testbench signals

reg clk_i;
reg rst_i;
// AXI-Lite Inputs

reg cfg_awvalid_i;
reg [31:0] cfg_awaddr_i;
reg cfg_wvalid_i;
reg [31:0] cfg_wdata_i;
reg [3:0] cfg_wstrb_i;
reg cfg_bready_i;
reg cfg_arvalid_i;
reg [31:0] cfg_araddr_i;
reg cfg_rready_i;
reg rx_i;
// AXI-Lite Outputs

wire cfg_awready_o;
wire cfg_wready_o;
wire cfg_bvalid_o;
```

```
wire [1:0] cfg_bresp_o;
wire cfg_arready_o;
wire cfg_rvalid_o;
wire [31:0] cfg_rdata_o;
wire [1:0] cfg_rresp_o;
wire tx_o;
wire intr_o;
// Instantiate UART Lite Module
uart_lite dut (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .cfg_awvalid_i(cfg_awvalid_i),
    .cfg_awaddr_i(cfg_awaddr_i),
    .cfg_wvalid_i(cfg_wvalid_i),
    .cfg_wdata_i(cfg_wdata_i),
    .cfg_wstrb_i(cfg_wstrb_i),
    .cfg_bready_i(cfg_bready_i),
    .cfg_arvalid_i(cfg_arvalid_i),
    .cfg_araddr_i(cfg_araddr_i),
    .cfg_rready_i(cfg_rready_i),
    .rx_i(rx_i),
    .cfg_awready_o(cfg_awready_o),
    .cfg_wready_o(cfg_wready_o),
    .cfg_bvalid_o(cfg_bvalid_o),
    .cfg_bresp_o(cfg_bresp_o),
    .cfg_arready_o(cfg_arready_o),
    .cfg_rvalid_o(cfg_rvalid_o),
    .cfg_rdata_o(cfg_rdata_o),
    .cfg_rresp_o(cfg_rresp_o),
    .tx_o(tx_o),
    .intr_o(intr_o)
);
// Clock Generation
always #4 clk_i = ~clk_i; // 125MHz clk(8ns period)
initial begin
    // Initialize Inputs
    clk_i = 0;
    rst_i = 1;
    cfg_awvalid_i = 0;
    cfg_awaddr_i = 0;
    cfg_wvalid_i = 0;
    cfg_wdata_i = 0;
    cfg_wstrb_i = 4'b1111;
    cfg_bready_i = 0;
    cfg_arvalid_i = 0;
    cfg_araddr_i = 0;
    cfg_rready_i = 0;
    rx_i = 1; // Idle state for UART RX
    // Reset the DUT
    #20 rst_i = 0;
    #8000 rx_i = 0; // start bit
    #8000 rx_i = 1;
```

```
    #8000 rx_i = 1;
    #8000 rx_i = 1;
    #8000 rx_i = 0;
    #8000 rx_i = 1;
    #8000 rx_i = 0;
    #8000 rx_i = 0;
    #8000 rx_i = 1;
    #8000 rx_i = 1; // stop bit
    // Write TX Register (0x04) with data 0x55 ('U' ASCII)
    #10 cfg_awvalid_i = 1;
        cfg_awaddr_i = 32'h04;
        cfg_wvalid_i = 1;
        cfg_wdata_i = 32'h00000055;
        cfg_bready_i = 1;
    // Wait for write response
    wait (cfg_bvalid_o);
    #1000 cfg_awvalid_i = 0;
        cfg_wvalid_i = 0;
        cfg_bready_i = 0;
    // Control register
    #10 cfg_awvalid_i = 1;
        cfg_awaddr_i = 32'h0C;
        cfg_wvalid_i = 1;
        cfg_wdata_i = 32'd16;
        cfg_bready_i = 1;
    // Wait for write response
    wait (cfg_bvalid_o);
    #1000 cfg_awvalid_i = 0;
        cfg_wvalid_i = 0;
        cfg_bready_i = 0;
        cfg_wdata_i = 32'd0;
    // Read Status Register (0x08)
    #10 cfg_arvalid_i = 1;
        cfg_araddr_i = 32'h08;
        cfg_rready_i = 1;
    // Wait for read response
    wait (cfg_rvalid_o);
    #1000 cfg_arvalid_i = 0;
        cfg_rready_i = 0;
    #10 cfg_arvalid_i = 1;
        cfg_araddr_i = 32'h0000;
        cfg_rready_i = 1;
    // Wait for read response
    wait (cfg_rvalid_o);
    #1000 cfg_arvalid_i = 0;
        cfg_rready_i = 0;
    // End simulation
    #10 $finish;
end
endmodule
```

3 Post Implementation Timing Simulation

The following image shows the output waveform of the above testbench code after the design is implemented. The status read operation shows `1B` in `cfg_rdata_o` signal which means that TX FIFO is full, RX FIFO is FULL and contains VALID data, and interrupt is also enabled. Then the receiver operation can be verified by reading the RX FIFO which displays the data `E9` that we sent through `rx_i` initially. The transmission operation can be verified by looking at the `tx_o` signal which sends bit by bit (TX FIFO contains 32'd55) at each `baud_clk` edge, which is shown in the second image.

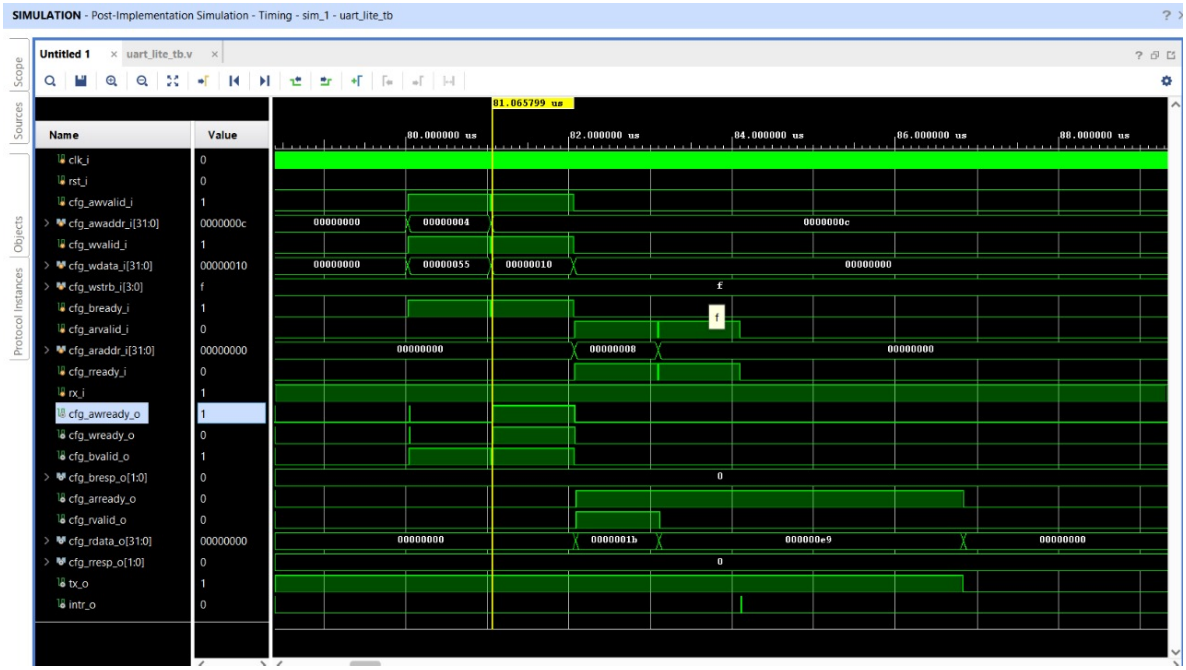

Figure 3: Post Implementation Timing Simulation

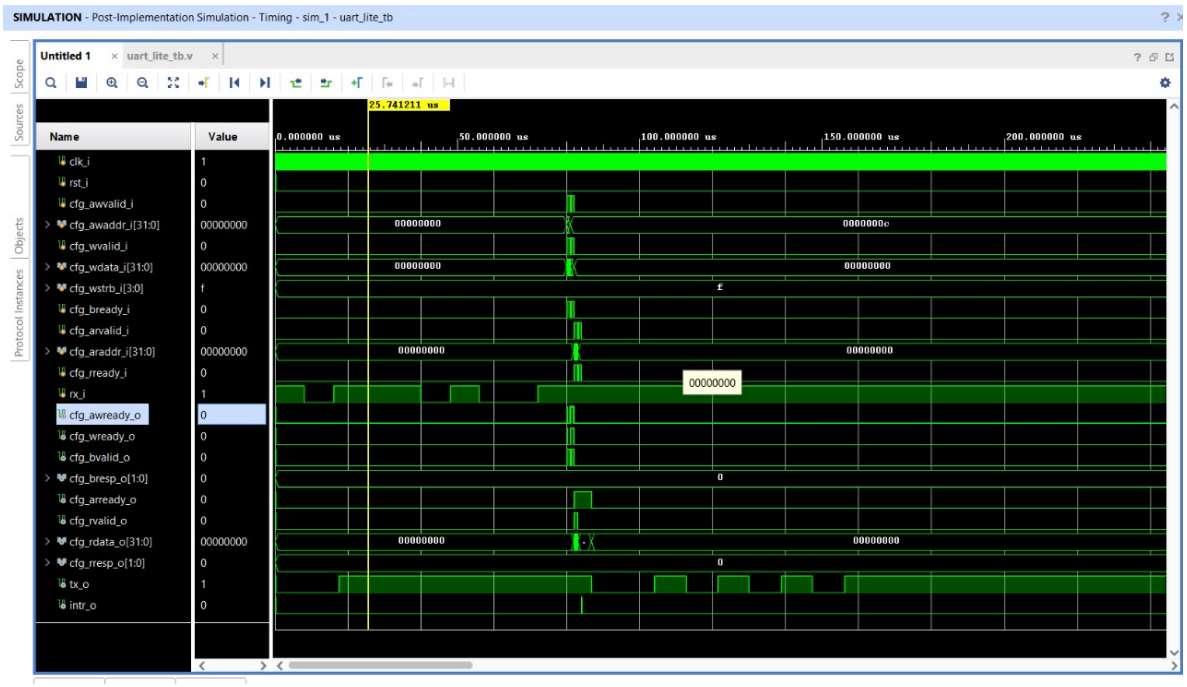

Figure 4: Zoomed out to show transmission Operation

4 Conclusion

Thus, the design of the UART_Lite module has been successfully achieved, synthesized, and implemented. The Post-Implementation Timing Simulation has been executed to verify the functionality and timing characteristics of the design. This implementation demonstrates efficient communication using the AXI4-Lite interface, enabling seamless data transfer between peripherals. The parameterized baud rate generator provides flexibility for various data rates, ensuring adaptability to different system requirements.