SILICONCLUSTER-2025-UART TRANSMITTER

1.Abstract

The uart_tx module implements a parameterizable **8-N-1 asynchronous serial transmitter** in Verilog-2005

2.Key features:

- Minimal I/O count (10 inputs, 2 outputs)
- Internal Power-On Reset (eliminates external reset pin)
- Fully parameterizable clock and baud rate
- Efficient finite-state machine (FSM) design
- Resource utilization well within ≤500 standard cells

2.1 Technical Specifications

Parameter	Value
HDL Language	Verilog-2005
Inputs	clk(1),tx_start(1),tx_data[7:0](8) -10 total
Outputs	tx,tx_busy
Default Clock (CLK_FREQ)	50MHz
Default Baud Rate (BAUD_RATE)	115200 bps
Data Format	8 data bits, No parity, 1 stop bit
Reset	Internal, active-high, 8-cycle POR
Max Area	Fits 150 µm × 150 µm (≤500 std cells)
Technology	SkyWater 130 nm CMOS

3. System Architecture

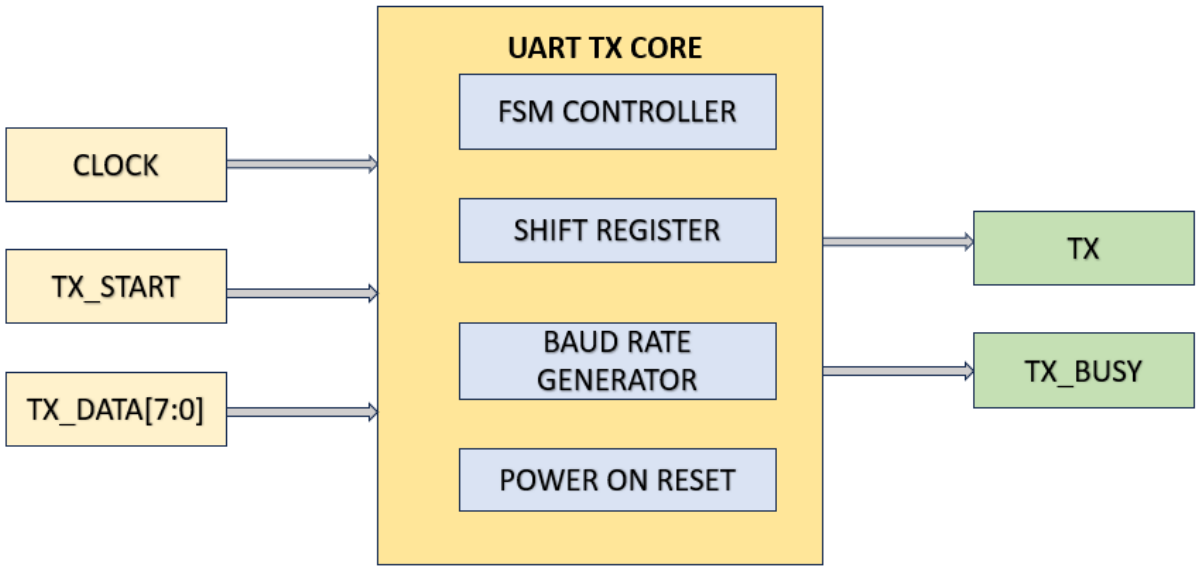
3.1 Functional Overview

- The transmitter is composed of four main subsystems:
- Power-On Reset (POR) — Ensures deterministic startup state.
- Baud Rate Generator — Divides system clock to serial bit rate.
- Shift Register — Loads parallel byte and shifts bits out LSB-first.
- FSM Controller — Orchestrates transmit sequence (IDLE → START → DATA → STOP).

3.2 State Machine

State Name	Function
S_IDLE	Idle line high, waiting for tx_start
S_START	Sends start bit (0)
S_DATA	Shifts and sends 8 data bits (LSB-first)
S_STOP	Sends stop bit (1) and returns to IDLE

3.3 Block Diagram:



4. Port Description

Port Name	Direction	Width	Description
clk	Input	1	System clock
tx_start	Input	1	Transmission request trigger
tx_data	Input	8	Byte to be transmitted
tx	Output	1	Serial TX line
tx_busy	Output	1	Busy status indicator

5. Baud Rate Calculation

- The baud interval is “CLKS_PER_BIT = CLK_FREQ / BAUD_RATE”
- For default 50 MHz and 115 200 bps: CLKS_PER_BIT ≈ 434

6. Testbench

- A simplified testbench (uart_tx_tb.v) validates module behavior.

Test Sequence:

- Waits ~8 cycles after power-on (POR delay).
- Sends byte 0xA5, waits for tx_busy low.
- Sends byte 0x3C.
- Dumps waveform to uart_tx.vcd for validation.

7. Compliance with Silicluster Requirements

Requirement	Specification	Status
I/O Limit	≤10 inputs + ≤10 outputs	10 inputs, 2 outputs
Area Limit	≤150 µm × 150 µm, ≤500 std cells	Estimated <400 cells
Language	Verilog-2005	Compliant
Reset	Optional external reset	Internal POR
Tech Node	SkyWater 130 nm	Compatible

8. Simulation Results

- **EDA**: Edaplayground
- **Source code URL**: https://edaplayground.com/x/bCGt
- **Simulation URL:**https://edaplayground.com/w/x/BUQ

- Start bit: Low for 1 baud period
- Data bits: Sent in correct order
- Stop bit: High for 1 baud period

---

9. Conclusion

- The uart_tx module is a simple and reliable UART transmitter designed for the Silicluster 2025 project.
- It meets all technical requirements, uses minimal resources, and requires no external reset.
- The design, testbench, and documentation are ready for submission and integration into the collective chip project.

---

10. Appendix: Source Code

UART_RX_DUT.V

```
module uart_tx #(
   parameter integer CLK_FREQ  = 50000000, // Hz
   parameter integer BAUD_RATE = 115200
)(
   input       clk,        // 1 clock input
   input       tx_start,   // 1 control input
   input  [7:0] tx_data,    // 8 data inputs  -> total inputs = 10
   output reg   tx,         // serial line (idle = 1)
   output reg   tx_busy     // high during transmission
);


   // ------------------------------------------------------------
   // Power-On Reset (POR): hold internal reset for a few cycles
   // ------------------------------------------------------------
   reg [3:0] por_cnt = 4'd0;
   wire     por_done = por_cnt[3];     // becomes 1 after 8 cycles
   wire     rst_i    = ~por_done;      // internal active-high reset

   always @(posedge clk) begin
      if (!por_done)
         por_cnt <= por_cnt + 4'd1;
   end


   // ------------------------------------------------------------
   // Baud timing
   // ------------------------------------------------------------
```

```
localparam integer CLKS_PER_BIT = CLK_FREQ / BAUD_RATE;

reg [15:0] clk_count;
reg [2:0]  bit_index;    // 0..7
reg [7:0]  tx_shift_reg;

// FSM
localparam [1:0]
   S_IDLE  = 2'b00,
   S_START = 2'b01,
   S_DATA  = 2'b10,
   S_STOP  = 2'b11;

reg [1:0] state;

// Outputs default
// (We fully drive them in the sequential block below.)

// ----------------------------------------------------------------
// Sequential logic
// ----------------------------------------------------------------
always @(posedge clk) begin
   if (rst_i) begin
      state      <= S_IDLE;
      tx         <= 1'b1;   // idle line high
      tx_busy    <= 1'b0;
      clk_count  <= 16'd0;
```

```
    bit_index   <= 3'd0;
    tx_shift_reg<= 8'd0;
  end else begin
    case (state)
      S_IDLE: begin
        tx     <= 1'b1;
        tx_busy <= 1'b0;
        clk_count <= 16'd0;
        if (tx_start) begin
          tx_shift_reg <= tx_data;
          state       <= S_START;
          tx_busy     <= 1'b1;
        end
      end

      S_START: begin
        tx <= 1'b0; // start bit (0)
        if (clk_count < CLKS_PER_BIT-1)
          clk_count <= clk_count + 16'd1;
        else begin
          clk_count <= 16'd0;
          bit_index <= 3'd0;
          state    <= S_DATA;
        end
      end

      S_DATA: begin
```

```
        tx <= tx_shift_reg[bit_index]; // LSB-first
        if (clk_count < CLKS_PER_BIT-1)
          clk_count <= clk_count + 16'd1;
        else begin
          clk_count <= 16'd0;
          if (bit_index < 3'd7)
            bit_index <= bit_index + 3'd1;
          else
            state <= S_STOP;
        end
      end

      S_STOP: begin
        tx <= 1'b1; // stop bit (1)
        if (clk_count < CLKS_PER_BIT-1)
          clk_count <= clk_count + 16'd1;
        else begin
          state    <= S_IDLE;
          tx_busy  <= 1'b0;
        end
      end

      default: begin
        state <= S_IDLE;
      end
    endcase
  end
```

```
    end

endmodule
```

TESTBENCH.v

```
`timescale 1ns/1ps
module uart_tx_tb;
    reg       clk = 1'b0;
    reg       tx_start;
    reg [7:0]  tx_data;
    wire      tx, tx_busy;

    // 50 MHz clock
    always #10 clk = ~clk;

    uart_tx #(.CLK_FREQ(50000000), .BAUD_RATE(115200)) dut (
        .clk(clk),
        .tx_start(tx_start),
        .tx_data(tx_data),
        .tx(tx),
        .tx_busy(tx_busy)
    );

    initial begin
        $dumpfile("uart_tx.vcd");
        $dumpvars(0, uart_tx_tb);
```

```
        // Wait for internal POR to complete (~8 cycles)
        tx_start = 1'b0;
        tx_data  = 8'h00;
        #(200);  // safe margin

        // Send 0xA5
        tx_data  = 8'hA5;
        tx_start = 1'b1; #(20); tx_start = 1'b0;
        wait (!tx_busy); #(200);

        // Send 0x3C
        tx_data  = 8'h3C;
        tx_start = 1'b1; #(20); tx_start = 1'b0;
        wait (!tx_busy); #(200);

        $finish;
    end
endmodule
```