**SILICONCLUSTER-2025-UART TRANSMITTER**

# 1.Abstract

The uart\_tx module implements a parameterizable **8-N-1 asynchronous serial transmitter** in Verilog-20052.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**

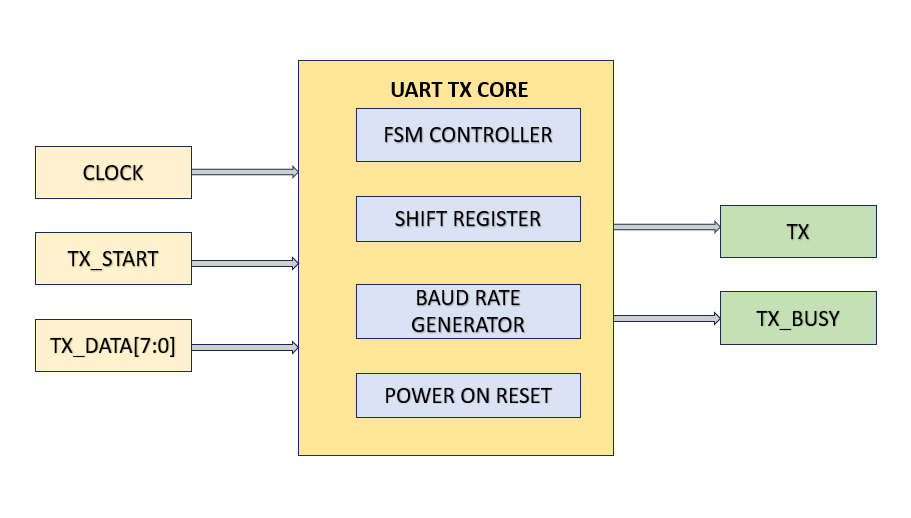
## 3.1 Functional Overview

* 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:

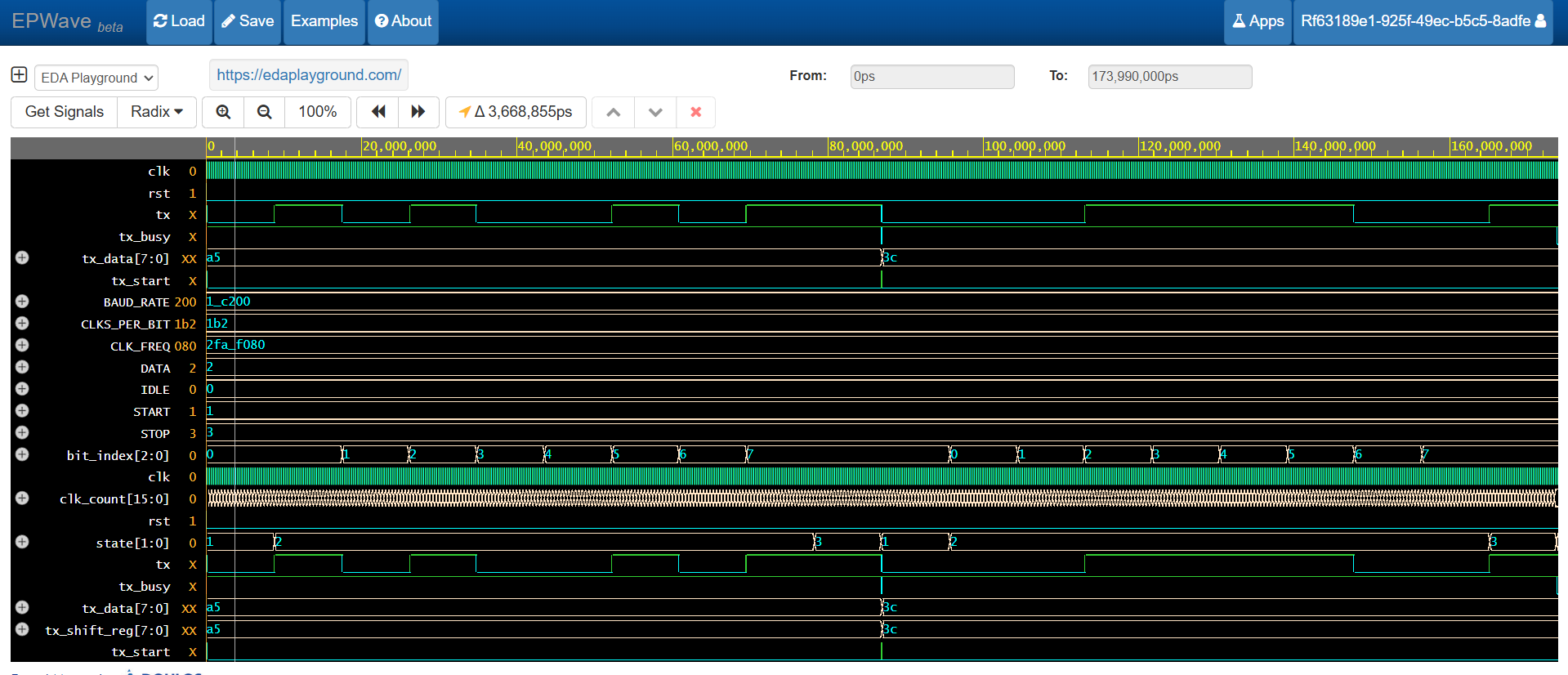
* Waits ~8 cycles after power-on (POR delay).
* Sends byte 0xA5, waits for tx\_busy low.
* Sends byte 0x3C.
* Dumps waveform to uart\_tx.vcd for validation.

# 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