**SILICONCLUSTER-2025-UART RECEIVER**

# **1.Abstract**

## The design and verification of an 8-N-1 UART Receiver implemented in Verilog-2005. The module operates at 50 MHz system clock, receiving asynchronous serial data at 115,200 baud. It complies fully with Silicluster 2025 requirements, including ≤10 outputs, single clock domain, and internal power-on reset. The design performs input synchronization, mid-bit sampling, and framing error detection.

## **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. Technical Specifications**

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Clock Frequency | 50 MHz |
| Baud Rate | 115,200 bps |
| Protocol Format | 8 data bits, No parity, 1 stop bit |
| Inputs | clk, rx |
| Outputs | rx\_data[7:0], rx\_valid, framing\_error (total 10 outputs) |
| Reset | Internal Power-On Reset |
| Clock Domains | Single |

# **3. System Architecture**

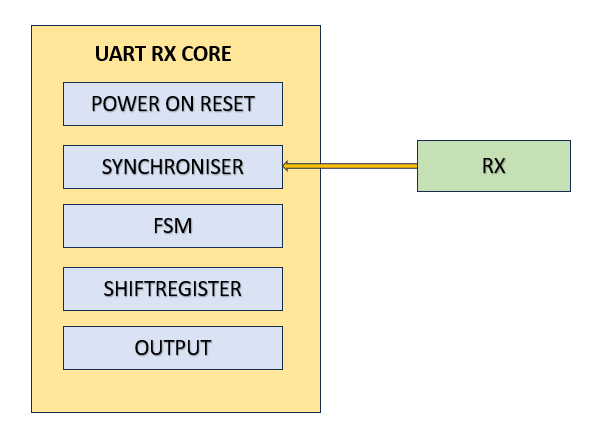
## **3.1 Functional Overview**

* 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:**



* **Power-On Reset (POR):** Internally holds module in reset for ~8 cycles after power-up.
* **Input Synchronization:** Two-stage flip-flop synchronizer ensures metastability protection on asynchronous rx input.
* **Finite State Machine (FSM):** Four states — S\_IDLE (waiting for start bit), S\_START (validating start bit), S\_DATA (capturing eight data bits), S\_STOP (validating stop bit).
* **Bit Sampling:** Samples each bit near the midpoint of its bit period using counters derived from CLK\_FREQ/BAUD\_RATE.
* **Framing Error Detection:** Flag set if stop bit is not high.

# **4. Testbench**

* Testbench sends two bytes: 0xA5 and 0x3C.
* rx\_valid asserts one clock cycle upon byte reception.
* rx\_data matches sent bytes exactly.
* framing\_error remains low for valid frames.
* Waveform dumps verify timing and data correctness.

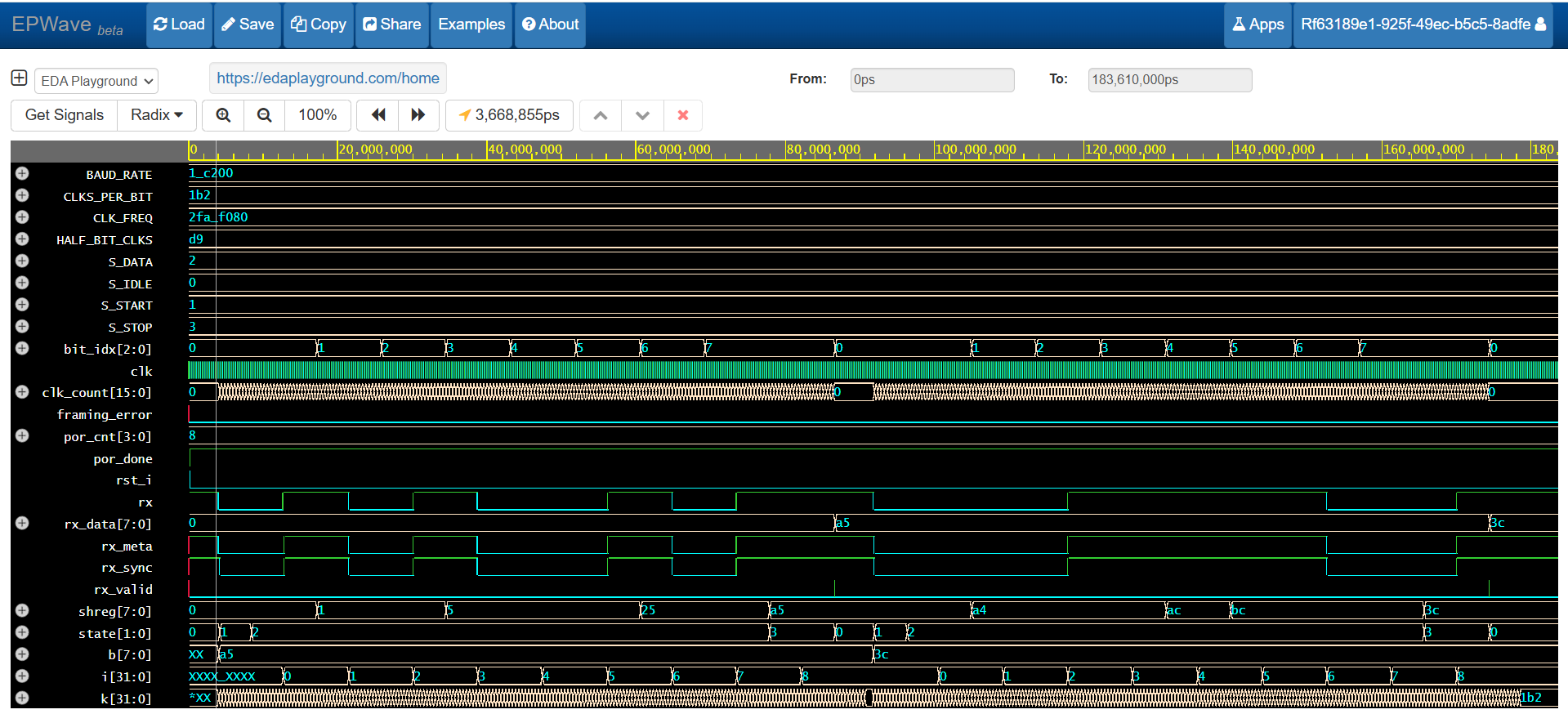
# **5. Compliance with Silicluster Requirements**

|  |  |
| --- | --- |
| **Requirement** | **Status** |
| ≤ 10 outputs | Yes |
| Single clock domain | Yes |
| No loops in RTL | Yes |
| Parameterizable frequency | Yes |
| Parameterizable baud rate | Yes |
| Functional testbench included | Yes |

|  |  |  |
| --- | --- | --- |
| **Component** | **Approximate Quantity** | **Gate Equivalent** |
| Flip-Flops (FF) | 32 | 128 gates |
| Counters & Adders | 2 | 60 gates |
| Comparators | 2 | 40 gates |
| FSM Logic | 10 | 30 gates |
| Total | — | 258 gates |

# **6. Simulation Results**

* **EDA**: Edaplayground
* **Source code URL**: <https://edaplayground.com/x/ghDc>
* **Simulation URL:**<https://edaplayground.com/w/x/EYr>



* **Testbench** sends two known bytes: 0xA5 and 0x3C.
* rx\_valid pulses exactly once per byte.
* rx\_data matches transmitted values.
* framing\_error remains low for correct frames.
* Waveforms confirm correct sampling, byte assembly, and timing alignment.

# **7. Conclusion**

* The UART RX module meets all functional and guideline requirements.
* It is compact (~258 gates), robust to metastability, and easy to integrate with the UART TX design to form a full-duplex UART system.

# **8. Appendix: Source Code**

**UART\_RX\_DUT.v:**

`timescale 1ns/1ps

// uart\_rx.v — Verilog-2005, 8-N-1 UART receiver, ≤10 outputs, 1 clk

// Inputs: clk, rx

// Outputs: rx\_data[7:0], rx\_valid, framing\_error (total outputs = 10)

module uart\_rx #(

parameter integer CLK\_FREQ = 50000000, // Hz

parameter integer BAUD\_RATE = 115200

)(

input clk, // single clock

input rx, // serial input

output reg [7:0] rx\_data, // received byte

output reg rx\_valid, // 1 clk pulse when byte ready

output reg framing\_error // asserted if stop bit not '1'

);

// ------------------------------------------

// Internal power-on reset (no external reset)

// ------------------------------------------

reg [3:0] por\_cnt = 4'd0;

wire por\_done = por\_cnt[3];

wire rst\_i = ~por\_done;

always @(posedge clk) begin

if (!por\_done) por\_cnt <= por\_cnt + 4'd1;

end

// ------------------------------------------

// Parameters for timing

// ------------------------------------------

localparam integer CLKS\_PER\_BIT = CLK\_FREQ / BAUD\_RATE; // ~434 @ 50MHz/115200

localparam integer HALF\_BIT\_CLKS = CLKS\_PER\_BIT/2;

// ------------------------------------------

// 2-FF synchronizer for 'rx'

// ------------------------------------------

reg rx\_meta, rx\_sync;

always @(posedge clk) begin

rx\_meta <= rx;

rx\_sync <= rx\_meta;

end

// ------------------------------------------

// FSM

// ------------------------------------------

localparam [1:0]

S\_IDLE = 2'b00,

S\_START = 2'b01,

S\_DATA = 2'b10,

S\_STOP = 2'b11;

reg [1:0] state;

reg [15:0] clk\_count; // wide enough for CLKS\_PER\_BIT

reg [2:0] bit\_idx; // 0..7

reg [7:0] shreg;

// ------------------------------------------

// Sequential logic

// ------------------------------------------

always @(posedge clk) begin

if (rst\_i) begin

state <= S\_IDLE;

clk\_count <= 16'd0;

bit\_idx <= 3'd0;

shreg <= 8'd0;

rx\_data <= 8'd0;

rx\_valid <= 1'b0;

framing\_error <= 1'b0;

end else begin

rx\_valid <= 1'b0; // default (one-cycle pulse when byte completes)

case (state)

S\_IDLE: begin

framing\_error <= 1'b0;

clk\_count <= 16'd0;

bit\_idx <= 3'd0;

if (rx\_sync == 1'b0) begin // start edge

state <= S\_START;

clk\_count <= 16'd0;

end

end

// Wait half a bit, then re-sample to confirm a valid start bit.

S\_START: begin

if (clk\_count < HALF\_BIT\_CLKS) begin

clk\_count <= clk\_count + 16'd1;

end else begin

if (rx\_sync == 1'b0) begin

// Good start bit → move to data; align to full bit periods

clk\_count <= 16'd0;

bit\_idx <= 3'd0;

state <= S\_DATA;

end else begin

// False start

state <= S\_IDLE;

end

end

end

// Sample each data bit at the middle of its bit window

S\_DATA: begin

if (clk\_count < CLKS\_PER\_BIT-1) begin

clk\_count <= clk\_count + 16'd1;

end else begin

clk\_count <= 16'd0;

shreg[bit\_idx] <= rx\_sync; // LSB-first

if (bit\_idx < 3'd7) begin

bit\_idx <= bit\_idx + 3'd1;

end else begin

state <= S\_STOP;

end

end

end

// Sample stop bit; if not '1', flag framing error

S\_STOP: begin

if (clk\_count < CLKS\_PER\_BIT-1) begin

clk\_count <= clk\_count + 16'd1;

end else begin

clk\_count <= 16'd0;

rx\_data <= shreg;

rx\_valid <= 1'b1;

framing\_error <= (rx\_sync != 1'b1);

state <= S\_IDLE;

end

end

default: state <= S\_IDLE;

endcase

end

end

endmodule

**TESTBENCH.v**

`timescale 1ns/1ps

module uart\_rx\_tb;

// Parameters matching TX

localparam integer CLK\_FREQ = 50000000;

localparam integer BAUD\_RATE = 115200;

localparam integer CLKS\_PER\_BIT = CLK\_FREQ / BAUD\_RATE; // ~434 at 50MHz

reg clk = 1'b0;

reg rx = 1'b1; // idle high

wire [7:0] rx\_data;

wire rx\_valid;

wire framing\_error;

// 50 MHz clock (20 ns period)

always #10 clk = ~clk;

// DUT

uart\_rx #(.CLK\_FREQ(CLK\_FREQ), .BAUD\_RATE(BAUD\_RATE)) dut (

.clk(clk),

.rx(rx),

.rx\_data(rx\_data),

.rx\_valid(rx\_valid),

.framing\_error(framing\_error)

);

// Task to transmit one UART frame on 'rx' (start + 8 data + stop)

task send\_byte;

input [7:0] b;

integer i, k;

begin

// start bit

rx = 1'b0;

for (k=0; k<CLKS\_PER\_BIT; k=k+1) @(posedge clk);

// data bits LSB-first

for (i=0; i<8; i=i+1) begin

rx = b[i];

for (k=0; k<CLKS\_PER\_BIT; k=k+1) @(posedge clk);

end

// stop bit

rx = 1'b1;

for (k=0; k<CLKS\_PER\_BIT; k=k+1) @(posedge clk);

end

endtask

initial begin

$dumpfile("uart\_rx.vcd");

$dumpvars(0, uart\_rx\_tb);

// idle a bit (also lets internal POR finish)

repeat (200) @(posedge clk);

// Send two bytes that we used in TX tests

send\_byte(8'hA5); // expect rx\_data=0xA5

repeat (50) @(posedge clk);

send\_byte(8'h3C); // expect rx\_data=0x3C

repeat (50) @(posedge clk);

// Check results (simple console prints)

@(posedge clk);

$display("Last rx\_data=0x%0h, rx\_valid=%0d, framing\_error=%0d", rx\_data, rx\_valid, framing\_error);

// Let waves settle then finish

repeat (200) @(posedge clk);

$finish;

end

endmodule