UART Specifications

Draft

Contents

Index **1**

1 Introduction **3**

2 UART (Universal Asynchronous Receiver Transmitter) **3**

2.1 UART Specifications . . . 3
2.2 UART Top Diagram . . . 4
2.3 Overview . . . 4
2.4 UART Microarchitecture . . . 4
2.5 UART Top Module – Port Description . . . 5
2.6 baud_rate_generator – Specifications . . . 5
2.7 baud_rate_generator – Port Description . . . 5
2.8 transmitter – Specifications . . . 5
2.9 transmitter – Port Description . . . 6
2.10 receiver – Specifications . . . 6
2.11 receiver – Port Description . . . 6
2.12 RTL Snippet . . . 6

3 Appendix A: RTL Modules - UART **8**

Appendix A: UART RTL Modules **8**

3.1 baud_rate_generator - module . . . 8
3.2 receiver - module . . . 8
3.3 transmitter - module . . . 11
3.4 uart top - module . . . 13
3.5 uart top testbench - module . . . 14

List of Figures

1 Diagram UART Top. . . . 4
2 UART uA. . . . 4

Glossary

- **MSB (Most Significant Bit)** — The bit in a binary number with the highest positional value. For example, in an 8-bit value, bit 7 is the MSB.
- **LSB (Least Significant Bit)** — The bit in a binary number with the lowest positional value. For the same 8-bit value, bit 0 is the LSB.
- **Tile** — A modular RTL block that implements a specific functionality and follows a standardized interface for integration into larger chip architectures. Designed for reuse and scalability in SoC designs.
- **Register** — A storage element in hardware that holds a value across clock cycles. Registers are used to store inputs, outputs, intermediate results, and control signals.
- **FSM (Finite State Machine)** — A logic-based control structure that transitions between a finite set of states based on inputs and internal conditions.
- **UART (Universal Asyncrhonous Receiver TRansmitter)** -

Draft

1 Introduction

The following sections present the main specifications of the UART module.

- An overview of its functionality.
- Port conventions and signal descriptions.
- RTL code snippets for reference.
- A diagram illustrating its internal structure and operation.

2 UART (Universal Asynchronous Receiver Transmitter)

2.1 UART Specifications

The UART module implements an asynchronous serial communication interface with the following characteristics:

- Communication mode: Asynchronous
- Oversampling: 16x
- Frame format:
  - 1 Start bit
  - 8 Data bits (LSB first)
  - 1 Stop bit
  - No parity
- Baud rate: Configurable through the baud rate generator
- Full-duplex internal structure (independent transmitter and receiver)

2.2 UART Top Diagram

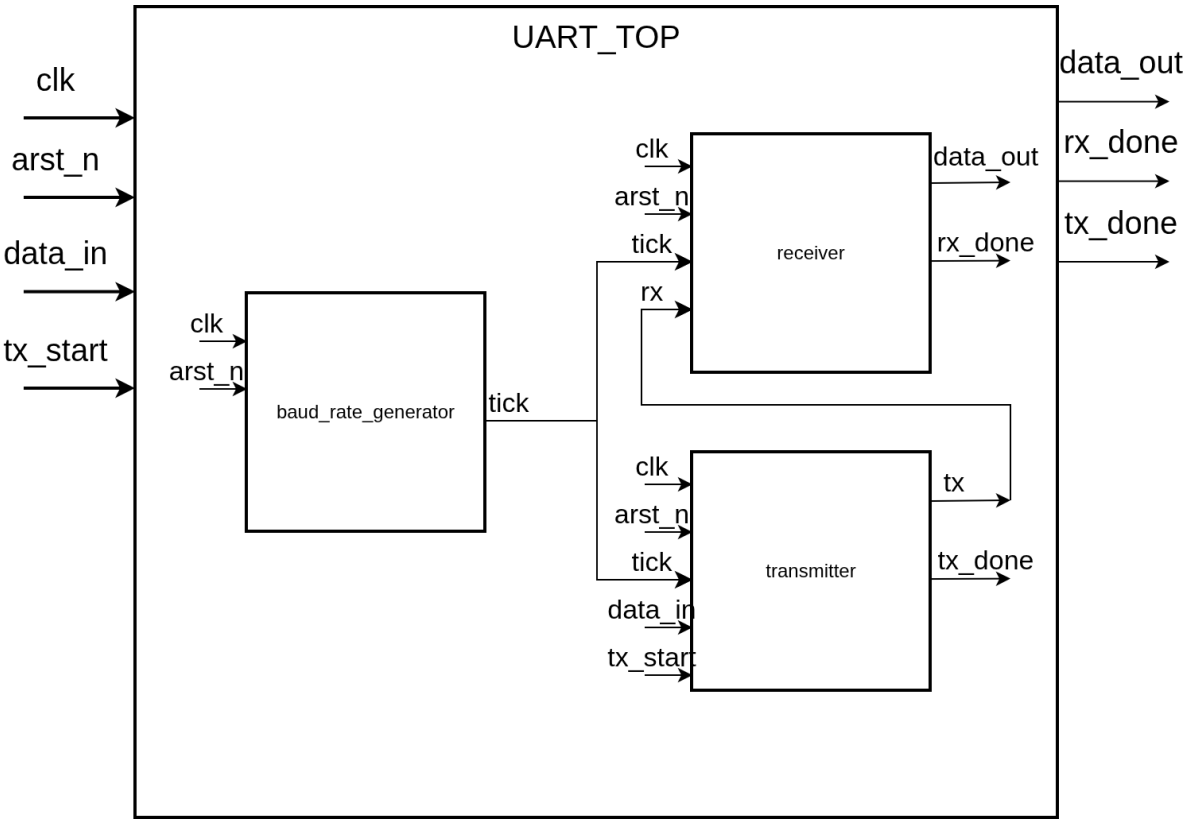

Figure 1: Diagram UART Top.

2.3 Overview

2.4 UART Microarchitecture

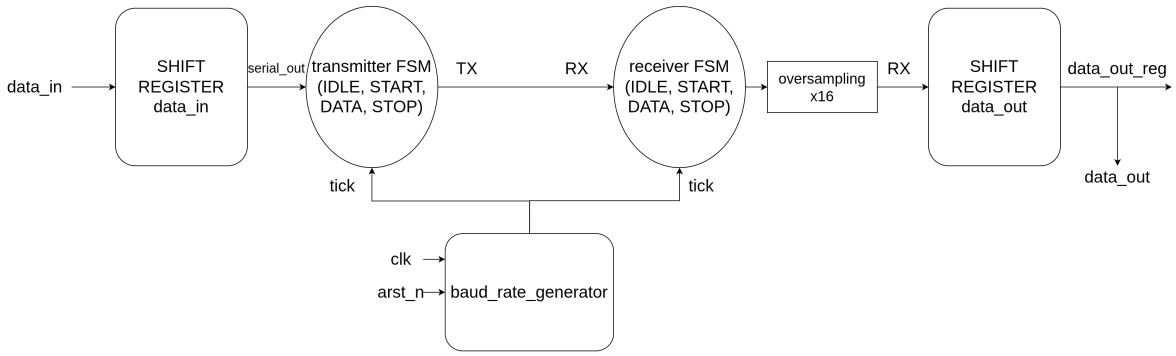

Figure 2: UART uA.

2.5 UART Top Module – Port Description

Port	Width	Direction	Description
clk	1	Input	System clock input. Used as the main clock domain for the UART logic.
arst_n	1	Input	Asynchronous active-low reset. Resets all internal registers and FSMs.
data_in	8	Input	Parallel input data byte to be transmitted by the UART transmitter.
tx_start	1	Input	Control signal to initiate a transmission. When asserted, the byte in data_in is transmitted.
tx_done	1	Output	Indicates that the transmission of the current byte has completed.
rx_done	1	Output	Indicates that a complete byte has been successfully received.
data_out	8	Output	Parallel output data byte received by the UART receiver.

2.6 baud_rate_generator – Specifications

The baud rate generator produces a periodic tick signal used to synchronize the UART transmitter and receiver.

- Oversampling factor: 16x
- Tick frequency: Baud rate × 16
- Configurable through the CLK_FREQ parameter

2.7 baud_rate_generator – Port Description

Port	Width	Direction	Description
clk	1	Input	System clock input.
arst_n	1	Input	Asynchronous active-low reset.
tick	1	Output	Baud tick signal used for UART oversampling (16x baud rate).

2.8 transmitter – Specifications

The transmitter module serializes parallel input data and transmits it following the UART frame format.

- Frame format: 1 start bit, 8 data bits, 1 stop bit
- Data transmission order: LSB first
- Controlled by a finite state machine (FSM)
- Transmission synchronized by a 16x oversampling tick

2.9 transmitter – Port Description

Port	Width	Direction	Description
clk	1	Input	System clock input.
arst_n	1	Input	Asynchronous active-low reset.
data_in	8	Input	Parallel input data byte to be transmitted.
tick	1	Input	Oversampling tick from the baud rate generator.
tx_start	1	Input	Starts the transmission process.
tx	1	Output	Serial data output line.
tx_done	1	Output	Indicates completion of the byte transmission.

2.10 receiver – Specifications

The receiver module deserializes incoming serial data and reconstructs parallel data bytes.

- Frame format: 1 start bit, 8 data bits, 1 stop bit
- Data reception order: LSB first
- Oversampling factor: 16x
- Mid-bit sampling for data stability
- Finite state machine-based control

2.11 receiver – Port Description Draft

Port	Width	Direction	Description
clk	1	Input	System clock input.
arst_n	1	Input	Asynchronous active-low reset.
tick	1	Input	Oversampling tick from the baud rate generator.
rx	1	Input	Serial input data line.
rx_done	1	Output	Indicates successful reception of a complete byte.
data_out	8	Output	Parallel output data byte.

2.12 RTL Snippet

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Manuel Enrique Rivera Acosta
//
// Create Date: 10/28/2025 12:14:17 PM
// Design Name:
// Module Name: uart_top
//////////////////////////////////////////////////////////////////////////////////

module uart_top #(parameter CLK_FREQ = 100_000_000, BYTE_WIDTH = 8) (
    input logic clk,
    input logic arst_n,
    input logic [BYTE_WIDTH - 1 : 0] data_in,
    input logic tx_start,
```

```
    output logic tx_done,
    output logic rx_done,
    output logic [BYTE_WIDTH - 1 : 0] data_out
);

logic tick;
logic serial_data;

baud_rate_generator #(.CLK_FREQ(CLK_FREQ)) baud_rate_generator_i(
    .clk(clk),
    .arst_n(arst_n),
    .tick(tick)
);

transmitter #(.BYTE_WIDTH(BYTE_WIDTH)) transmitter_i(
   .clk(clk),
   .arst_n(arst_n),
   .data_in(data_in),
   .tick(tick),
   .tx_start(tx_start),
   .tx(serial_data),
   .tx_done(tx_done)
);

receiver #(.BYTE_WIDTH(BYTE_WIDTH)) receiver_i(
.clk(clk),
.arst_n(arst_n),
.tick(tick),
.rx(serial_data), // RX CONNECTED TO THE TX FROM THE TRANSMITTER MODULE
.rx_done(rx_done),
.data_out(data_out)
);


endmodule
```

3 Appendix A: RTL Modules - UART

3.1 baud_rate_generator - module

```
`timescale 1ns / 1ps
//
    //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer: Manuel Enrique Rivera Acosta
//
// Create Date: 10/28/2025 11:14:06 AM
// Design Name:
// Module Name: baud_rate_generator
// Project Name:
//
    //////////////////////////////////////////////////////////////////////////////////


module baud_rate_generator #(parameter CLK_FREQ =100_000_000)(
    input  logic       clk,
    input  logic       arst_n,
    output logic       tick
);

logic [31:0] counter_max;
logic [31:0] counter;
                                        Draft
assign counter_max = (CLK_FREQ / (9600 * 16));

always_ff @(posedge clk or negedge arst_n) begin
    if(!arst_n) begin
        tick    <= 1'b0;
        counter <= 32'd0;
    end else begin
        if (counter == counter_max) begin
            counter <= 32'd0;
            tick    <= 1'b1;
        end else begin
            counter <= counter + 1;
            tick    <= 1'b0;
        end
    end
end

endmodule
```

3.2 receiver - module

```
`timescale 1ns / 1ps
//
    //////////////////////////////////////////////////////////////////////////////////

// Company:
```

```
// Engineer: Manuel Enrique Rivera Acosta
//
// Create Date: 10/28/2025 11:54:10 AM
// Design Name:
// Module Name: receiver
//
    //////////////////////////////////////////////////////////////////////////////////


module receiver #(parameter BYTE_WIDTH = 8)(
input logic clk,
input logic arst_n,
input logic tick, // TICK COMING FROM THE BAUD RATE GENERATOR
input logic rx, // INPUT SERIAL COMING FROM THE TRANSMITTER TX
output logic rx_done, // DONE SIGNAL TO INDICATE THAT THE BYTE HAS BEEN
    RECEIVED
output logic [BYTE_WIDTH - 1 : 0] data_out // OUPUT SIGNAL FOR THE BYTE
    RECEIVED
);

localparam BIT_SAMPLING = 15;
localparam HALFBIT_SAMPLING = 7;

logic [4:0] nbits, nbits_next;
logic [4:0] oversampling_count, oversampling_count_next;
logic [BYTE_WIDTH - 1 : 0] data_out_reg, data_out_reg_next;
logic rx_done_next;

typedef enum logic [2:0] {IDLE = 3'b000, START = 3'b001, DATA = 3'b010, STOP =
     3'b011} state_type;

state_type state_reg, state_next;

always_ff @(posedge clk or negedge arst_n) begin
    if(!arst_n) begin
        state_reg <= IDLE;
    end else begin
        oversampling_count <= oversampling_count_next;
        state_reg <= state_next;
        nbits <= nbits_next;
        data_out_reg <= data_out_reg_next;
        rx_done <= rx_done_next;
    end
end

always_comb begin
    oversampling_count_next = oversampling_count;
    nbits_next              = nbits;
    data_out_reg_next       = data_out_reg;
    state_next              = state_reg;
    rx_done_next            = rx_done;

    case(state_reg)
        IDLE: begin
            rx_done_next = 1'b0;
            oversampling_count_next = '0;
```

```
55              data_out_reg_next       = '0;
56              nbits_next              = '0;
57              if (!rx) begin
58                  data_out_reg_next = '0;
59                  state_next = START;
60                          end
61          end
62
63          START: begin
64              if (tick) begin
65                  if (oversampling_count == BIT_SAMPLING) begin
66                      oversampling_count_next = '0;
67                      nbits_next = '0;
68                      data_out_reg_next = '0;
69                      state_next = DATA;
70                  end else begin
71                      oversampling_count_next = oversampling_count + 1'b1;
72                      if (oversampling_count == HALFBIT_SAMPLING) begin
73                          state_next = (!rx) ? START : IDLE;
74                                      end
75                  end
76              end
77          end
78
79          DATA: begin
80              if (tick) begin
81                  if (oversampling_count == BIT_SAMPLING) begin
82                      oversampling_count_next = '0;
83                      if (nbits == (BYTE_WIDTH - 1'b1)) begin
84                          state_next = STOP;
85                      end else begin
86                          nbits_next = nbits + 1'b1;
87                                      end
88                  end else begin
89                      oversampling_count_next = oversampling_count + 1'b1;
90                      if (oversampling_count == HALFBIT_SAMPLING) begin
91                          data_out_reg_next = {rx, data_out_reg[BYTE_WIDTH
                                -1:1]};
92                                      end
93                  end
94              end
95          end
96
97          STOP: begin
98              if (tick) begin
99                  if (rx) begin
100                     if (oversampling_count == BIT_SAMPLING) begin
101                         rx_done_next = 1'b1;
102                         state_next = IDLE;
103                         oversampling_count_next = '0;
104                     end else begin
105                         oversampling_count_next = oversampling_count + 1'b1;
106                     end
107                 end else begin
108                     state_next = IDLE;
109                     oversampling_count_next = '0;
```

```
                end
            end
        end
            default: begin
                    state_next = state_reg;
            end
    endcase
end

assign data_out = data_out_reg;

endmodule
```

3.3 transmitter - module

```
`timescale 1ns / 1ps
//
    //////////////////////////////////////////////////////////////////////////////////

// Company:
// Engineer: Manuel Enrique Rivera Acosta
//
// Create Date: 10/28/2025 12:38:11 PM
// Design Name:
// Module Name: transmitter
// Project Name:                        Draft
//
    //////////////////////////////////////////////////////////////////////////////////



module transmitter #(parameter BYTE_WIDTH = 8)(
    input logic clk, // clock signal
    input logic arst_n, // asynchronous reset
    input logic [BYTE_WIDTH - 1 : 0] data_in, // DATA_IN TO BE SENT TO THE
        RECEIVER MODULE BIT BY BIT THROUGH THE TX
    input logic tick, // TICK COMING FROM THE BAUD RATE GENERATOR
    input logic tx_start, // TX_START SIGNAL TO BEGIN THE BYTE TRANSMISSION
    output logic tx, // SERIAL OUTPUT BIT TO TRANSMIT INTO THE RECEIVER RX
        INPUT SIGNAL
    output logic tx_done // DONE SIGNAL INDICATING THAT THE BYTE HAS BEEN SENT
);

localparam BIT_SAMPLING = 15; // localparam for the oversampling counter of
    the uart x16

logic [BYTE_WIDTH - 1 : 0] shifted_data; // temporal register to store the
    data shifted to the tx
logic [BYTE_WIDTH - 1 : 0] shifted_data_next; // next-state temporal register
    shifted_data

logic [4:0] nbits; // variable to count the number of bits that has been sent
logic [4:0] nbits_next; // next-state nbits variable

```

```
logic [4:0] oversampling_count; // oversampling counter for the uart x16
logic [4:0] oversampling_count_next; // next-state oversampling counter

logic tx_done_next; // next-state tx_done signal (transmission done signal)
logic tx_next;  // next-state tx signal (serial output bit)

typedef enum logic [2:0]  {IDLE = 3'b000, START = 3'b001, DATA = 3'b010, STOP
    = 3'b011} state_type;

state_type state_reg, state_next;

always_ff @(posedge clk or negedge arst_n) begin
    if(!arst_n) begin
        state_reg <= IDLE;
    end else begin
        state_reg <= state_next;
        nbits <= nbits_next;
        oversampling_count <= oversampling_count_next;
        shifted_data <= shifted_data_next;
        tx_done <= tx_done_next;
        tx <= tx_next;
    end
end

always_comb begin
        state_next = state_reg;
        nbits_next = nbits;
        oversampling_count_next = oversampling_count;
        shifted_data_next = shifted_data;
        tx_done_next = tx_done;
        tx_next = tx;

    case(state_reg)

        IDLE: begin
            tx_done_next = 1'b0;
            nbits_next = '0;
            tx_next = 1'b1;
            oversampling_count_next = '0;
            if(tx_start) begin
                shifted_data_next = data_in;
                tx_next = 1'b0;
                state_next = START;
            end
        end

        START: begin
            if(tick) begin
                if(oversampling_count == BIT_SAMPLING) begin
                    oversampling_count_next = '0;
                    state_next = DATA;
                end else begin
                    oversampling_count_next = oversampling_count + 1;
                end
            end
        end
```

```
87
88          DATA: begin
89              tx_next = shifted_data[0];
90              if(tick) begin
91                  if(oversampling_count == BIT_SAMPLING) begin
92                      shifted_data_next = {1'b0, shifted_data[7:1]};
93                      oversampling_count_next = '0;
94                      if(nbits == BYTE_WIDTH - 1) begin
95                          state_next = STOP;
96                      end else begin
97                          nbits_next = nbits + 1'b1;
98                      end
99                  end else begin
100                     oversampling_count_next = oversampling_count + 1;
101                 end
102             end
103         end
104
105         STOP: begin
106             tx_next = 1'b1;
107             if(tick) begin
108                 if(oversampling_count == BIT_SAMPLING) begin
109                     state_next = IDLE;
110                     tx_done_next = 1'b1;
111                 end else begin
112                     oversampling_count_next = oversampling_count + 1;
113                     state_next = STOP;
114                 end
115             end                          Draft
116         end
117         default: begin
118                 state_next = state_reg;
119         end
120     endcase
121 end
122
123 endmodule
```

3.4 uart top - module

```
1  `timescale 1ns / 1ps
2  //
       //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer: Manuel Enrique Rivera Acosta
5  //
6  // Create Date: 10/28/2025 12:14:17 PM
7  // Design Name:
8  // Module Name: uart_top
9  //
       //////////////////////////////////////////////////////////////////////////////////
10
11 module uart_top #(parameter CLK_FREQ = 100_000_000, BYTE_WIDTH = 8) (
```

```
    input logic clk,
    input logic arst_n,
    input logic [BYTE_WIDTH - 1 : 0] data_in,
    input logic tx_start,
    output logic tx_done,
    output logic rx_done,
    output logic [BYTE_WIDTH - 1 : 0] data_out
);

logic tick;
logic serial_data;

baud_rate_generator #(.CLK_FREQ(CLK_FREQ)) baud_rate_generator_i(
    .clk(clk),
    .arst_n(arst_n),
    .tick(tick)
);

transmitter #(.BYTE_WIDTH(BYTE_WIDTH)) transmitter_i(
   .clk(clk),
   .arst_n(arst_n),
   .data_in(data_in),
   .tick(tick),
   .tx_start(tx_start),
   .tx(serial_data),
   .tx_done(tx_done)
);

receiver #(.BYTE_WIDTH(BYTE_WIDTH)) receiver_i(
.clk(clk),
.arst_n(arst_n),
.tick(tick),
.rx(serial_data), // RX CONNECTED TO THE TX FROM THE TRANSMITTER MODULE
.rx_done(rx_done),
.data_out(data_out)
);


endmodule
```


3.5 uart top testbench - module

```
module uart_top_tb;

localparam BYTE_WIDTH = 8;

bit clk;
bit arst_n;
logic [BYTE_WIDTH - 1 : 0] data_in;
logic tx_start;
logic tx_done;
logic rx_done;
logic [BYTE_WIDTH - 1 : 0] data_out;

always #5ns clk = ~clk;
```

```
assign #50ns arst_n = 1'b1;

initial begin
    data_in = '0;
    tx_start = 1'b0;
    repeat (10) begin
        std::randomize(data_in);
        #1ms;
        @(posedge clk);
        tx_start = 1'b1;
        @(posedge clk);
        tx_start = 1'b0;
        @(posedge clk);
        #1ms;
    end
end

initial begin
    #20ms;
    $finish;
end

logic [7:0] data_in_temp;

always_ff @(posedge clk or negedge arst_n) begin
  if (!arst_n)
    data_in_temp <= '0;
  else if (tx_start)
    data_in_temp <= data_in;
end


uart_top uart_top_i(
.clk(clk),
.arst_n(arst_n),
.data_in(data_in),
.tx_start(tx_start),
.tx_done(tx_done),
.rx_done(rx_done),
.data_out(data_out)
);

assert property(
    @(posedge clk)
    disable iff(!arst_n)
    rx_done |-> (data_out == data_in_temp)
);

assert property (
  @(posedge clk)
  disable iff(!arst_n)
  $rose(tx_start) |-> ##[100000:115000] ($rose(tx_done) && (data_out ==
      data_in_temp))
);

assert property (
```

```
69    @(posedge clk)
70    disable iff(!arst_n)
71    $rose(tx_start) |=>  (~uart_top_i.transmitter_i.tx)
72  );
73
74  endmodule
```