İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK ELEKTRONİK FAKÜLTESİ

DIGITAL SYSTEM DESIGN APPLICATIONS

(EHB 436E)

Finite State Machines Project Report

Hasan Emre AYDEMİR

1. UART Protocol

Physical Interface and Hardware Architecture The UART interface utilizes a minimalist hardware architecture that eliminates the need for an external clock signal, relying instead on dedicated Transmit (TX) and Receive (RX) lines along with a common ground reference. In this configuration, cross-coupling is essential for physical data flow, meaning the transmission line of one device must be strictly paired with the receiving line of the counterpart. Internally, **shift registers** are employed to convert parallel data from the CPU into the serial bit stream required for transmission, and vice versa for reception.

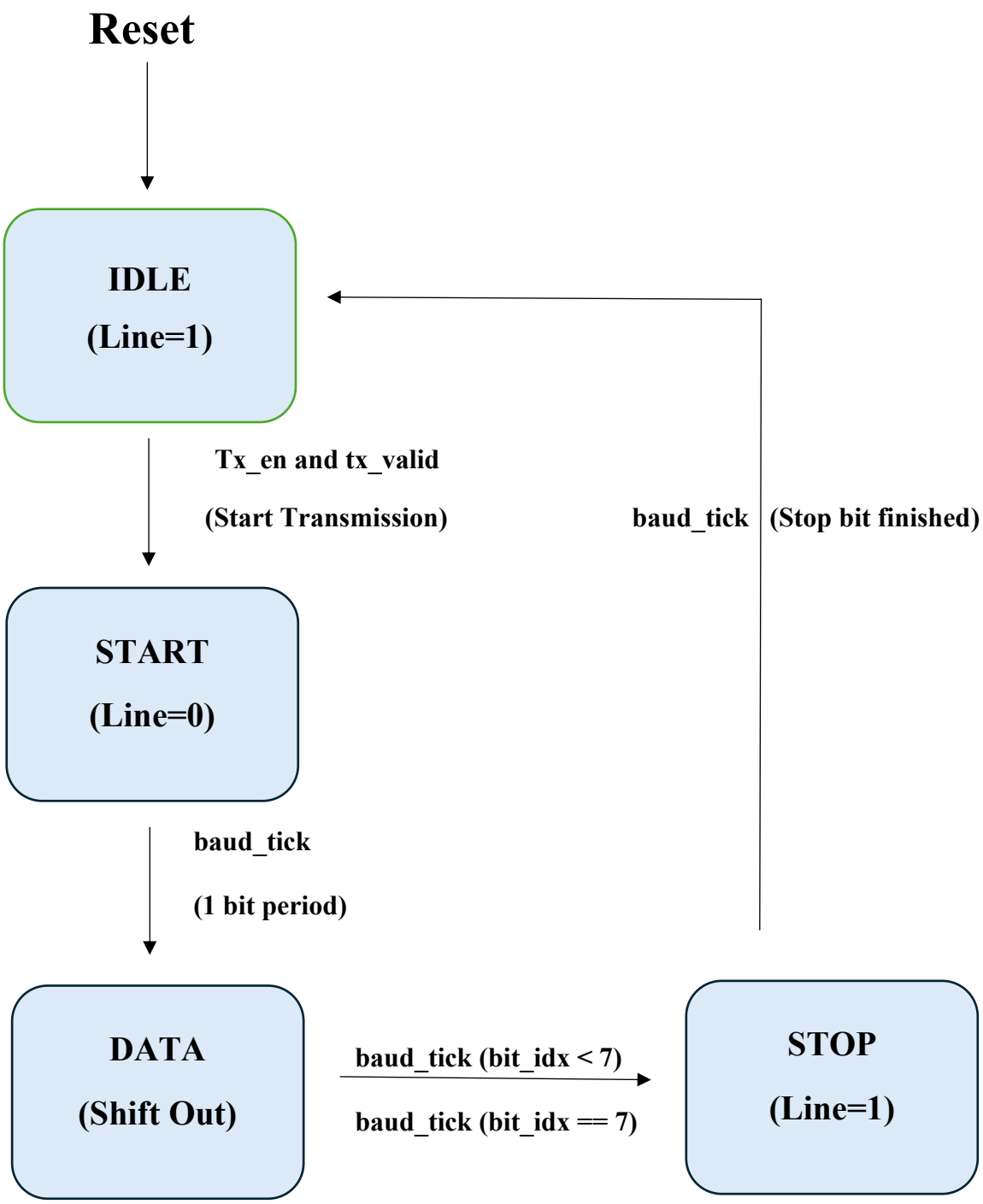
Data Frame and Operational Cycle To maintain data integrity, every information packet is encapsulated within a strict frame format governed by a finite state machine (FSM) logic. The sequence initiates with a "Start" bit, triggered by pulling the idle high voltage line (Logic 1) to a low level. Following this trigger, the actual data bits-typically ordered from the least significant bit (LSB) and an optional parity bit are transmitted. The cycle concludes with "Stop" bits that return the line to a high state, signaling the completion of the packet transfer.

Timing and Asynchronous Synchronization In the absence of a shared clock line, synchronization relies critically on both the receiver and transmitter operating at a pre-agreed transmission speed known as the "Baud Rate." Upon detecting the falling edge of the Start bit, the receiver engages its internal timer. In robust implementations, this synchronization is refined through oversampling (often 16 times per bit), which allows the receiver to determine the logical value at the precise midpoint of each bit duration, thereby mitigating signal noise and timing drift.

In real-world scenarios, UART remains a fundamental standard for interfacing with peripherals such as GPS receivers, Bluetooth modules, and system debugging consoles. Its primary advantage lies in its minimalist hardware overhead, requiring only two data lines, and its adaptability for long-distance communication when paired with standards like RS-232 or RS-485. However, this asynchronous architecture presents a technical trade-off; the strict requirement for precise baud rate matching between devices reduces error tolerance, while its inherently lower data throughput compared to synchronous protocols like SPI limits its utility in modern high-bandwidth applications.

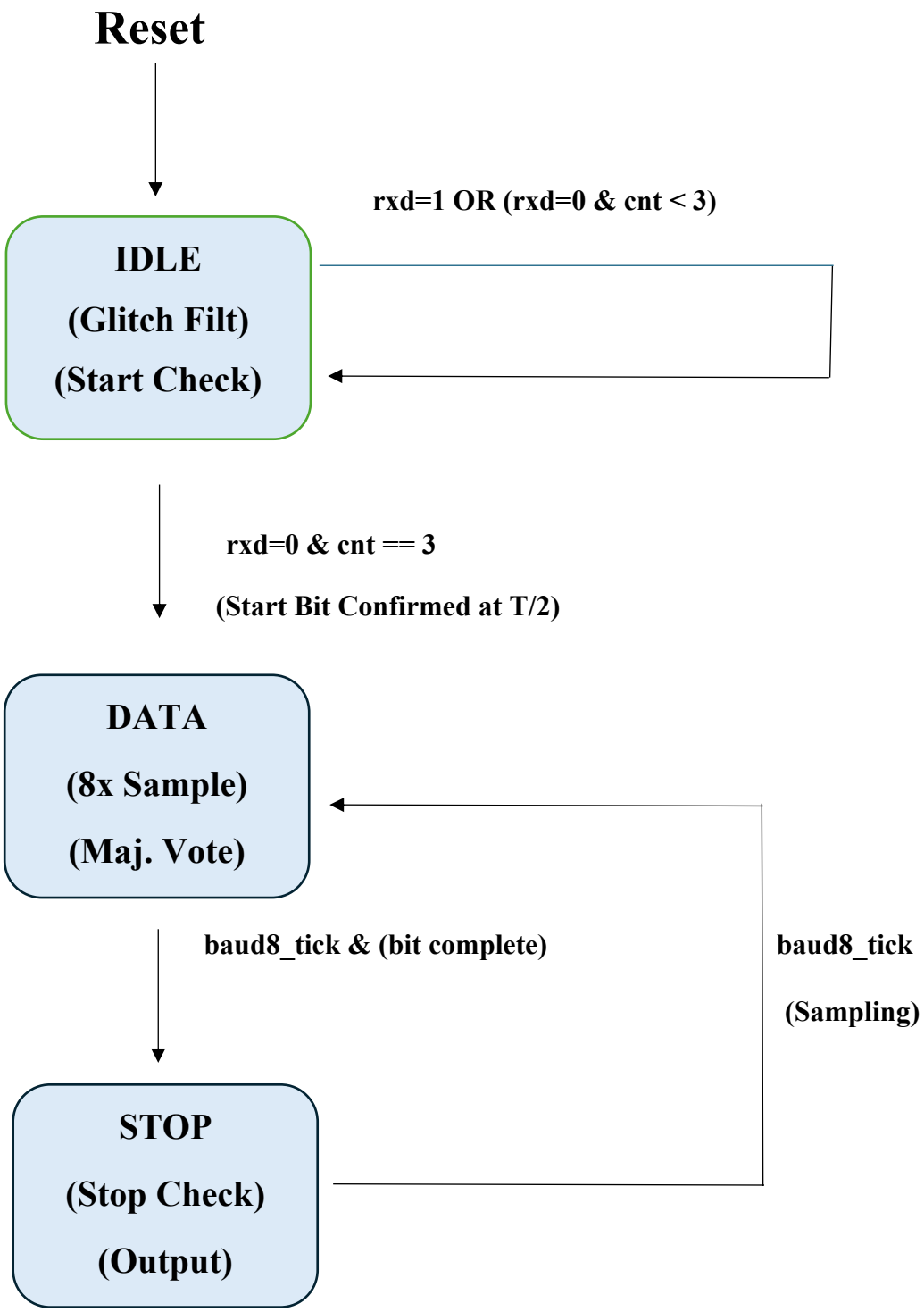
2. State Diagrams

2.1. Transmitter



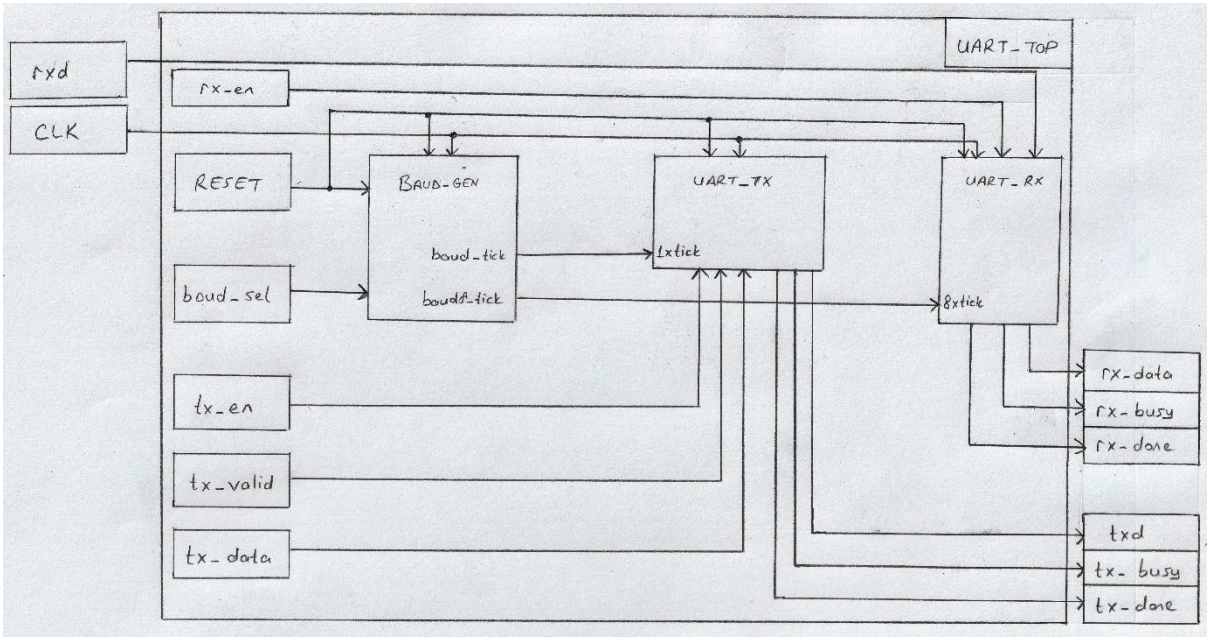
Transmitter FSM consists of four states. It remains in the IDLE state while the line is high. Upon receiving a valid transmission request (Tx_en and tx_valid), it transitions to START to pull the line low. It then moves to DATA to shift out 8 bits serially (LSB first). Finally, it enters the STOP state to drive the line high before returning to IDLE.

2.2. Receiver



Receiver FSM. The IDLE state incorporates a noise filter that validates the Start Bit by ensuring the line remains low for 4 sample ticks (T/2). In the DATA state, the system uses 8x oversampling (baud8_tick) and majority voting to accurately recover the bits. The STOP state validates the frame and asserts the output data.

3. Block Diagram



The figure illustrates the top-level block diagram of the designed UART (Universal Asynchronous Receiver-Transmitter) system. The architecture consists of three main sub-modules: the Baud Generator, the Transmitter, and the Receiver. The signal flow and operational logic depicted in the block diagram are as follows:

1. System Synchronization: The **CLK** (Clock) and **RESET** signals are distributed in parallel to all three modules to ensure synchronous operation. This guarantees that all modules are initialized and reset simultaneously.

2. Timing Generation (Baud Generator): The **baud_gen** module divides the system clock (**FCLK_HZ**) to establish the communication speed. Based on the **baud_sel** input, it generates two distinct timing signals:

    a. **baud_tick:** Determines the standard bit duration for the Transmitter (TX) module.

    b. **baud8_tick:** A signal generated at 8 times the baud rate for the Receiver (RX) module. This enables noise filtering and precise oversampling of the incoming data.

3. Data Transmission (Transmitter): When the **tx_en** signal is active and triggered by **tx_valid**, the **uart_tx** module accepts the parallel **tx_data**. It serializes this data according to the **baud_tick** timing and transmits it to the external interface via the **txd** line.

4. Data Reception (Receiver): While **rx_en** is active, the **uart_rx** module monitors the incoming **rxd** line. It uses the **baud8_tick** signal to oversample the incoming data. Once the data is successfully received and verified, it is deserialized into **rx_data**, and the **rx_done** signal is asserted.

4. Implementation Reports

4.1. Report Utilization

According to the Vivado post-implementation utilization report, the UART design uses very limited FPGA resources. The design utilizes a total of 63 Slice LUTs and 77 Slice Registers (FFs), corresponding to less than 1% of the available device resources. In addition, 30 Bonded IOBs are used for external connections and 1 BUFG is utilized for clock distribution.

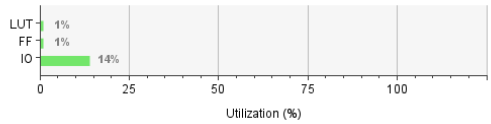
Module-level resource usage is summarized below:

Hierarchy

Name	Slice LUTs (32600)	Slice Registers (65200)	Slice (8150)	LUT as Logic (32600)	Bonded IOB (210)	BUFGCTRL (32)
uart_top	63	77	24	63	30	1
u_baud (baud_gen)	11	21	8	11	0	0
u_rx (uart_rx)	39	38	13	39	0	0
u_tx (uart_tx)	13	18	4	13	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	63	32600	0.19
FF	77	65200	0.12
IO	30	210	14.29



Primitives

Ref Name	Used	Functional Category
FDRE	76	Flop & Latch
LUT6	21	LUT
LUT3	19	LUT
LUT4	16	LUT
OBUF	15	IO
IBUF	15	IO
LUT5	13	LUT
LUT2	13	LUT
CARRY4	6	CarryLogic
LUT1	2	LUT
FDSE	1	Flop & Latch
BUFG	1	Clock

4.2. Report Timing Summary

To enable accurate timing analysis, a user-defined clock constraint was added to the design using a constraint file named timing.xdc. The following clock definition was applied to the system clock port:

```
create_clock -name clk -period 10.000 [get_ports clk]
```

This constraint defines the clock period as 10 ns, corresponding to a 100 MHz system clock. With this constraint, Vivado was able to perform proper setup and hold timing analysis across all synchronous paths in the design.

Intra-Clock Paths - clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	5.629	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[4]/R	3.788	1.699	2.089	10.0	clk	clk		0.035
Path 2	5.629	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[5]/R	3.788	1.699	2.089	10.0	clk	clk		0.035
Path 3	5.629	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[6]/R	3.788	1.699	2.089	10.0	clk	clk		0.035
Path 4	5.629	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[7]/R	3.788	1.699	2.089	10.0	clk	clk		0.035
Path 5	5.668	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[12]/R	3.746	1.699	2.047	10.0	clk	clk		0.035
Path 6	5.668	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[13]/R	3.746	1.699	2.047	10.0	clk	clk		0.035
Path 7	5.668	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[14]/R	3.746	1.699	2.047	10.0	clk	clk		0.035
Path 8	5.668	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[15]/R	3.746	1.699	2.047	10.0	clk	clk		0.035
Path 9	5.705	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[10]/R	3.710	1.699	2.011	10.0	clk	clk		0.035
Path 10	5.705	4	16	u_baud/cnt8_reg[3]/C	u_baud/cnt8_reg[11]/R	3.710	1.699	2.011	10.0	clk	clk		0.035

Intra-Clock Paths - clk - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 11	0.150	0	2	u_rx/data_buf_reg[2]/C	u_rx/rx_data_reg[2]/D	0.210	0.141	0.069	0.0	clk	clk		0.000
Path 12	0.179	0	2	u_rx/data_buf_reg[0]/C	u_rx/rx_data_reg[0]/D	0.249	0.141	0.108	0.0	clk	clk		0.000
Path 13	0.193	0	2	u_rx/data_buf_reg[4]/C	u_rx/rx_data_reg[4]/D	0.282	0.164	0.118	0.0	clk	clk		0.000
Path 14	0.207	0	2	u_rx/data_buf_reg[1]/C	u_rx/rx_data_reg[1]/D	0.269	0.141	0.128	0.0	clk	clk		0.000
Path 15	0.227	1	9	u_rx/bit_idx_reg[0]/C	u_rx/data_buf_reg[1]/D	0.333	0.186	0.147	0.0	clk	clk		0.000
Path 16	0.230	1	6	u_rx/align_cnt_reg[0]/C	u_rx/align_cnt_reg[0]/D	0.321	0.186	0.135	0.0	clk	clk		0.000
Path 17	0.233	1	3	u_rx/ones_cnt_reg[1]/C	u_rx/ones_cnt_reg[3]/D	0.340	0.190	0.150	0.0	clk	clk		0.000
Path 18	0.236	0	2	u_rx/data_buf_reg[5]/C	u_rx/rx_data_reg[5]/D	0.322	0.141	0.181	0.0	clk	clk		0.000
Path 19	0.244	1	3	u_rx/ones_cnt_reg[1]/C	u_rx/ones_cnt_reg[2]/D	0.336	0.186	0.150	0.0	clk	clk		0.000
Path 20	0.248	1	10	u_rx/samp_cnt_reg[1]/C	u_rx/ones_cnt_reg[1]/D	0.354	0.186	0.168	0.0	clk	clk		0.000

Unconstrained Paths - NONE - clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 21	∞	5	6	baud_sel	u_baud/div8_cnt_reg[2]/D	5.270	2.185	3.085	∞	input port clock	clk		0.025
Path 22	∞	5	6	baud_sel	u_baud/div8_cnt_reg[1]/D	5.263	2.185	3.078	∞	input port clock	clk		0.025
Path 23	∞	5	6	baud_sel	u_baud/baud_tick_reg/D	5.242	2.157	3.085	∞	input port clock	clk		0.025
Path 24	∞	5	6	baud_sel	u_baud/div8_cnt_reg[0]/D	5.235	2.157	3.078	∞	input port clock	clk		0.025
Path 25	∞	5	16	baud_sel	u_baud/cnt8_reg[4]/R	5.227	2.157	3.070	∞	input port clock	clk		0.025
Path 26	∞	5	16	baud_sel	u_baud/cnt8_reg[5]/R	5.227	2.157	3.070	∞	input port clock	clk		0.025
Path 27	∞	5	16	baud_sel	u_baud/cnt8_reg[6]/R	5.227	2.157	3.070	∞	input port clock	clk		0.025
Path 28	∞	5	16	baud_sel	u_baud/cnt8_reg[7]/R	5.227	2.157	3.070	∞	input port clock	clk		0.025
Path 29	∞	5	16	baud_sel	u_baud/cnt8_reg[12]/R	5.185	2.157	3.028	∞	input port clock	clk		0.025
Path 30	∞	5	16	baud_sel	u_baud/cnt8_reg[13]/R	5.185	2.157	3.028	∞	input port clock	clk		0.025

Unconstrained Paths - NONE - clk - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 31	∞	1	61	rst	u_baud/baud_tick_reg/R	0.722	0.193	0.529	-∞	input port clock	clk		0.000
Path 32	∞	1	61	rst	u_baud/div8_cnt_reg[0]/R	0.722	0.193	0.529	-∞	input port clock	clk		0.000
Path 33	∞	1	61	rst	u_baud/div8_cnt_reg[1]/R	0.722	0.193	0.529	-∞	input port clock	clk		0.000
Path 34	∞	1	61	rst	u_baud/div8_cnt_reg[2]/R	0.722	0.193	0.529	-∞	input port clock	clk		0.000
Path 35	∞	1	61	rst	u_rx/start_low_cnt_reg[2]/R	0.722	0.193	0.529	-∞	input port clock	clk		0.000
Path 36	∞	1	61	rst	u_rx/data_buf_reg[4]/R	0.746	0.193	0.552	-∞	input port clock	clk		0.000
Path 37	∞	1	61	rst	u_rx/data_buf_reg[6]/R	0.746	0.193	0.552	-∞	input port clock	clk		0.000
Path 38	∞	1	61	rst	u_rx/data_buf_reg[0]/R	0.747	0.193	0.554	-∞	input port clock	clk		0.000
Path 39	∞	1	61	rst	u_rx/data_buf_reg[2]/R	0.747	0.193	0.554	-∞	input port clock	clk		0.000
Path 40	∞	1	61	rst	u_rx/data_buf_reg[3]/R	0.747	0.193	0.554	-∞	input port clock	clk		0.000

Unconstrained Paths - clk - NONE - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 41	∞	1	1	u_tx/done_reg/C	tx_done	5.325	3.220	2.105	∞	clk			0.025
Path 42	∞	1	1	u_tx/start_reg/C	tx_start	5.308	3.215	2.094	∞	clk			0.025
Path 43	∞	1	1	u_tx/txd_reg/C	txd	5.168	3.068	2.100	∞	clk			0.025
Path 44	∞	1	1	u_tx/busy_reg/C	tx_busy	5.005	3.078	1.928	∞	clk			0.025
Path 45	∞	1	1	u_rx/rx_data_reg[0]/C	rx_data[0]	4.977	3.086	1.891	∞	clk			0.025
Path 46	∞	1	1	u_rx/rx_data_reg[1]/C	rx_data[1]	4.950	3.085	1.864	∞	clk			0.025
Path 47	∞	1	1	u_rx/rx_data_reg[2]/C	rx_data[2]	4.948	3.089	1.859	∞	clk			0.025
Path 48	∞	1	1	u_rx/rx_data_reg[3]/C	rx_data[3]	4.910	3.095	1.815	∞	clk			0.025
Path 49	∞	1	1	u_rx/rx_data_reg[4]/C	rx_data[4]	4.907	3.234	1.673	∞	clk			0.025
Path 50	∞	1	1	u_rx/rx_data_reg[7]/C	rx_data[7]	4.901	3.223	1.678	∞	clk			0.025

Unconstrained Paths - clk - NONE - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 51	∞	1	1	u_rx/busy_reg/C	rx_busy	1.611	1.288	0.323	-∞	clk			0.025
Path 52	∞	1	1	u_rx/done_reg/C	rx_done	1.617	1.274	0.343	-∞	clk			0.025
Path 53	∞	1	1	u_rx/rx_data_reg[6]/C	rx_data[6]	1.645	1.322	0.323	-∞	clk			0.025
Path 54	∞	1	1	u_rx/rx_data_reg[4]/C	rx_data[4]	1.668	1.339	0.329	-∞	clk			0.025
Path 55	∞	1	1	u_rx/rx_data_reg[5]/C	rx_data[5]	1.671	1.324	0.347	-∞	clk			0.025
Path 56	∞	1	1	u_rx/rx_data_reg[7]/C	rx_data[7]	1.676	1.327	0.349	-∞	clk			0.025
Path 57	∞	1	1	u_rx/start_reg/C	rx_start	1.689	1.269	0.420	-∞	clk			0.025
Path 58	∞	1	1	u_rx/rx_data_reg[1]/C	rx_data[1]	1.698	1.287	0.410	-∞	clk			0.025
Path 59	∞	1	1	u_rx/rx_data_reg[3]/C	rx_data[3]	1.698	1.297	0.401	-∞	clk			0.025
Path 60	∞	1	1	u_rx/rx_data_reg[2]/C	rx_data[2]	1.703	1.290	0.413	-∞	clk			0.025

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5,629 ns	Worst Hold Slack (WHS):	0,150 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	124	Total Number of Endpoints:	124	Total Number of Endpoints:	78

All user specified timing constraints are met.

According to the Vivado post-implementation timing analysis, the clock constraint is defined as **10 ns (100 MHz)**. The setup analysis reports **Worst Negative Slack (WNS) = +5.629 ns** and **Total Negative Slack (TNS) = 0.000 ns**, while the hold analysis reports **Worst Hold Slack (WHS) = +0.150 ns**. No timing violations are observed.

Based on these values, the minimum achievable clock period is:

$$T_{min} = 10.000 - 5.629 = 4.371 \text{ ns}$$

and the maximum clock frequency is:

$$F_{max} = 1 / 4.371 \text{ ns} \approx \mathbf{228.7\ MHz}$$

5. Post-Implementation Timing Reports

The following analyses are based on the "***Post-Implementation Timing Simulation*** " results of the UART Transmitter (Tx) and Receiver (Rx) modules performed in the Vivado.

```
`timescale 1ns/1ps

module tb_baud_gen;

  reg clk=0, rst=1, baud_sel=0;
  wire baud_tick, baud8_tick;

  baud_gen #(.FCLK_HZ(100_000_000)) dut (
    .clk(clk),
    .rst(rst),
    .baud_sel(baud_sel),
    .baud_tick(baud_tick),
    .baud8_tick(baud8_tick)
  );

  always #5 clk = ~clk; // 100MHz

  initial begin
    #200 rst=0;

    // 9600
    baud_sel=0;
    #500_000; // 500us

    // 115200
    baud_sel=1;
    #200_000; // 200us

    $finish;
  end

endmodule
```

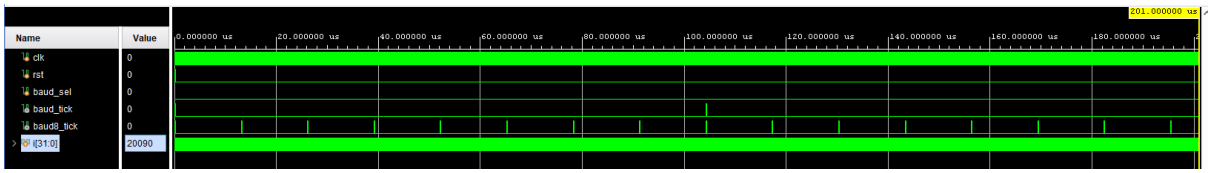


Figure 1. Post-Implementation Timing Simulation of Baud Rate Generator

Figure 1 presents the post-implementation timing simulation results of the baud rate generator used in the UART system. The simulation was performed after placement and routing, including realistic FPGA routing delays.

When baud_sel = 0, the system operates in **9600 baud mode**. The waveform shows that the baud_tick signal is generated with a period of approximately **104 µs**, while the baud8_tick signal is generated with a period of approximately **13 µs**. These values are consistent with the theoretical baud rate and confirm that the required **8 × oversampling clock** is correctly produced.

These results verify that the baud rate generator provides accurate timing signals for the receiver oversampling and majority voting mechanisms under post-implementation conditions.

```
`timescale 1ns/1ps

module tb_uart_tx;

  reg clk=0, rst=1;
  reg baud_sel=0;

  wire baud_tick, baud8_tick;

  reg  Tx_en=0;
  reg  tx_valid=0;
  reg  [7:0] tx_data=8'h00;

  wire txd;
  wire start,busy,done;

  baud_gen #(.FCLK_HZ(100_000_000)) u_baud(
    .clk(clk), .rst(rst), .baud_sel(baud_sel),
    .baud_tick(baud_tick), .baud8_tick(baud8_tick)
  );

  uart_tx dut(
    .clk(clk), .rst(rst),
    .Tx_en(Tx_en),
    .baud_tick(baud_tick),
    .tx_valid(tx_valid),
    .tx_data(tx_data),
    .txd(txd),
    .start(start), .busy(busy), .done(done)
  );

  always #5 clk = ~clk;

  task send_byte(input [7:0] b);
    begin
      @(posedge clk);
      tx_data  <= b;
      tx_valid <= 1'b1;
      @(posedge clk);
      tx_valid <= 1'b0;
      wait(done);
      @(posedge clk);
    end
  endtask

  initial begin
    #200 rst=0;

    Tx_en=1;
    baud_sel=0;

    send_byte(8'hA5);
    send_byte(8'h3C);
    send_byte(8'hF0);
    send_byte(8'h12);

    #50_000;
    $finish;
  end

endmodule
```

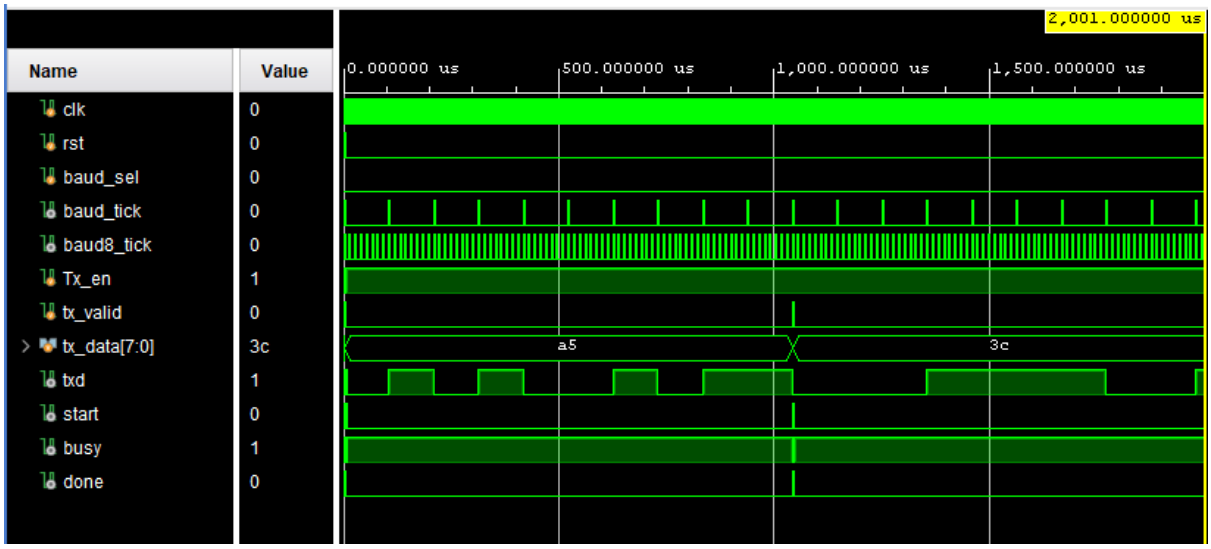


Figure 2. Post-Implementation Timing Simulation of UART Transmitter

Figure 2 shows the post-implementation timing simulation results of the UART transmitter module. The simulation includes realistic FPGA routing delays after placement and routing.

When the tx_valid signal is asserted, the transmission starts and the start signal is triggered while the busy signal remains high during the data transfer. The serial output txd correctly generates a standard UART frame consisting of **one start bit (0), eight data bits (LSB first), and one stop bit (1)**. After the transmission is completed, the done signal produces a pulse, indicating the end of the frame.

These results confirm that the UART transmitter operates correctly under post-implementation timing conditions.

```
`timescale 1ns/1ps

module tb_uart_rx;

  reg clk = 0;
  reg rst = 1;
  always #5 clk = ~clk;
  reg  baud_sel = 0;          // 0:9600, 1:115200
  wire baud_tick;
  wire baud8_tick;

  baud_gen #(.FCLK_HZ(100_000_000)) u_baud (
    .clk(clk),
    .rst(rst),
    .baud_sel(baud_sel),
    .baud_tick(baud_tick),
    .baud8_tick(baud8_tick)
  );
  reg        Tx_en    = 0;
  reg        tx_valid = 0;
  reg [7:0]  tx_data  = 8'h00;
  wire       txd;
  wire       tx_start, tx_busy, tx_done;

  uart_tx u_tx (
    .clk(clk),
    .rst(rst),
    .Tx_en(Tx_en),
    .baud_tick(baud_tick),
    .tx_valid(tx_valid),
    .tx_data(tx_data),
    .txd(txd),
    .start(tx_start),
    .busy(tx_busy),
    .done(tx_done)
  );

  reg  Rx_en = 0;
  wire [7:0] rx_data;
  wire       rx_start, rx_busy, rx_done;

  wire rxd = txd;

  uart_rx u_rx (
    .clk(clk),
    .rst(rst),
    .Rx_en(Rx_en),
    .baud8_tick(baud8_tick),
    .rxd(rxd),
    .rx_data(rx_data),
    .start(rx_start),
    .busy(rx_busy),
    .done(rx_done)
  );
  task send_byte(input [7:0] b);
    begin
      @(posedge clk);
      tx_data  <= b;
      tx_valid <= 1'b1;
      @(posedge clk);
      tx_valid <= 1'b0;

      wait(rx_done);
      $display("RX=%h expected=%h", rx_data, b);

      @(posedge clk);
    end
  endtask

  initial begin
    #200;
    rst = 0;
    baud_sel = 0;   // 9600
    Tx_en    = 1;
    Rx_en    = 1;
    send_byte(8'h55);
    send_byte(8'hC3);
    send_byte(8'h0F);
    send_byte(8'hA6);

    baud_sel = 1;   // 115200
    send_byte(8'h3C);

    #50_000;
    $finish;
  end
endmodule
```

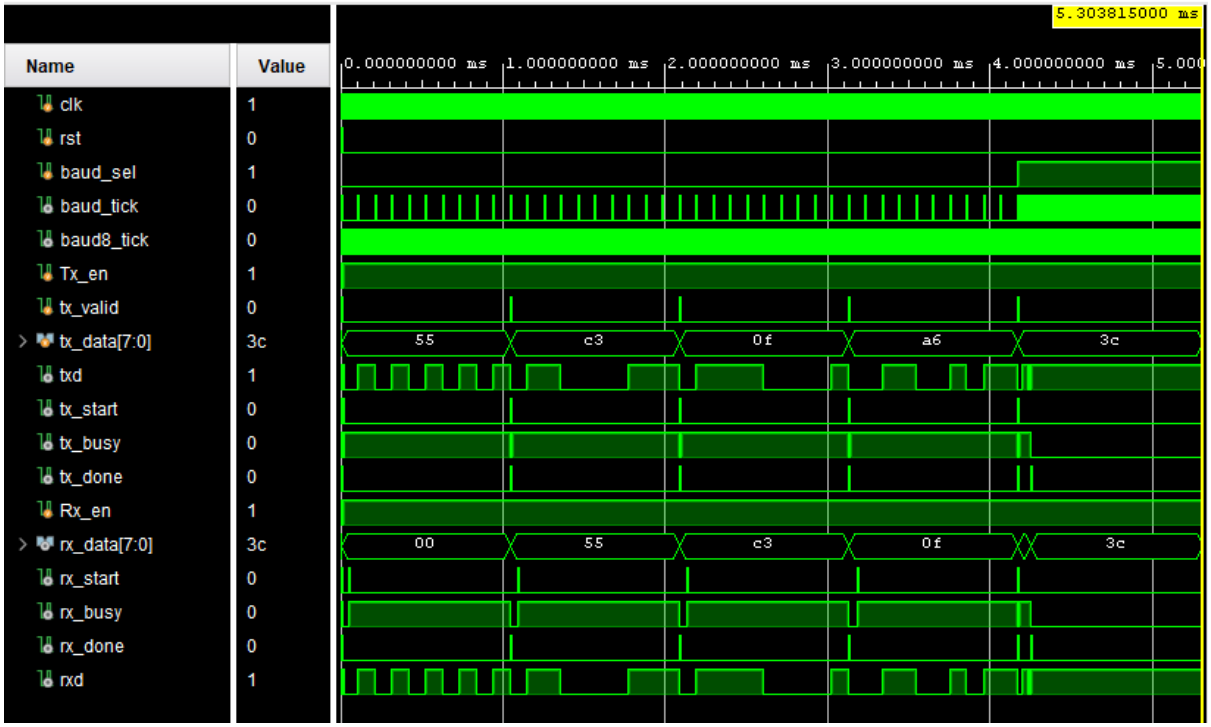


Figure 3. Post-Implementation Timing Simulation of UART Receiver

Figure 3 shows the post-implementation timing simulation results of the UART receiver module. The simulation includes realistic FPGA routing delays after placement and routing.

The waveform demonstrates that the bytes 55, C3, 0F, A6, and 3C transmitted by the UART transmitter are successfully received by the UART receiver. The rx_done signal generates a pulse at the end of each frame, and the rx_data output exactly matches the transmitted bytes. These results confirm correct UART receiver operation under post-implementation timing conditions.

```
`timescale 1ns/1ps

module tb_uart_top;

  reg clk=0, rst=1;
  reg baud_sel=0;

  reg Tx_en=0, Rx_en=0;
  reg tx_valid=0;
  reg [7:0] tx_data=8'h00;

  wire txd;
  wire [7:0] rx_data;

  wire tx_start, tx_busy, tx_done;
  wire rx_start, rx_busy, rx_done;


  wire rxd = txd;

  uart_top #(.FCLK_HZ(100_000_000)) dut (
    .clk(clk), .rst(rst), .baud_sel(baud_sel),
    .Tx_en(Tx_en), .Rx_en(Rx_en),
    .tx_valid(tx_valid), .tx_data(tx_data), .txd(txd),
    .rxd(rxd), .rx_data(rx_data),
    .tx_start(tx_start), .tx_busy(tx_busy), .tx_done(tx_done),
    .rx_start(rx_start), .rx_busy(rx_busy), .rx_done(rx_done)
  );

  always #5 clk = ~clk;

  task send(input [7:0] b);
    begin
      @(posedge clk);
      tx_data  <= b;
      tx_valid <= 1'b1;
      @(posedge clk);
      tx_valid <= 1'b0;

      wait(rx_done);
      $display("LOOPBACK RX=%h expected=%h", rx_data, b);
      @(posedge clk);
    end
  endtask

  initial begin
    #200 rst=0;

    Tx_en=1; Rx_en=1;
    baud_sel=0;

    send(8'h12);
    send(8'hAB);
    send(8'h5A);
    send(8'hC7);

    baud_sel=1;
    send(8'h3C);

    #50_000;
    $finish;
  end

endmodule
```

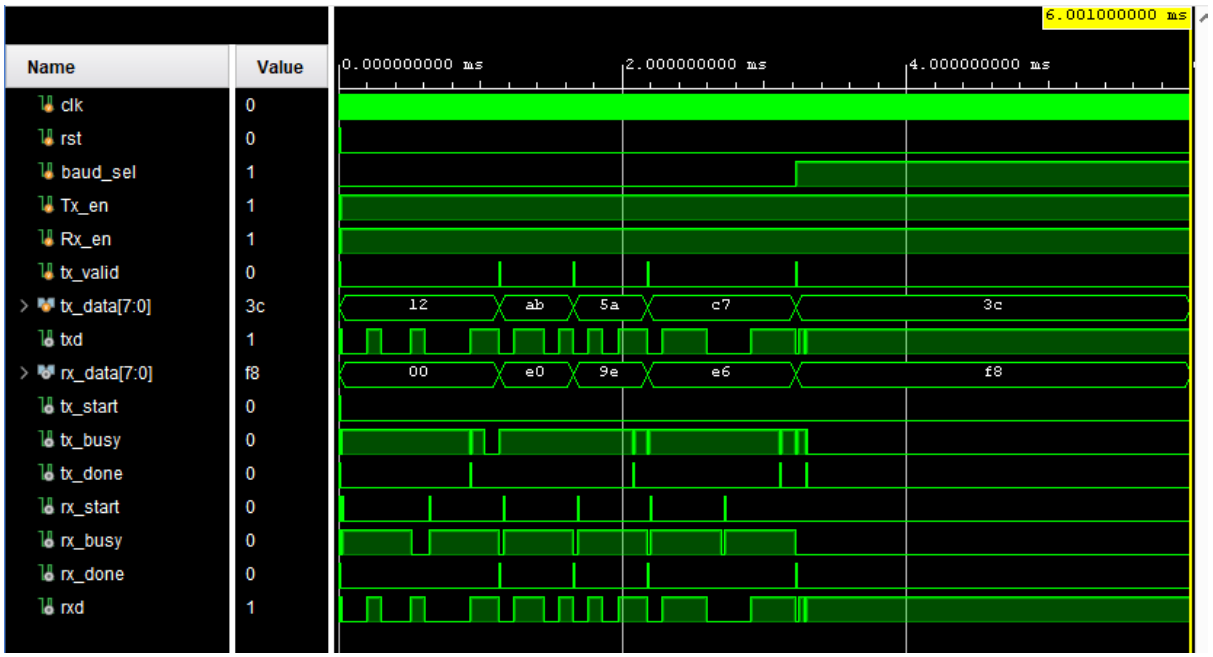


Figure 4. Post-Implementation Timing Simulation of the Complete UART System

Figure 4 shows the post-implementation timing simulation results of the complete UART system, where all sub-modules are integrated and tested under a **loopback configuration**.

The waveform demonstrates that the bytes 12, AB, 5A, C7, and 3C transmitted by the UART transmitter are successfully received and decoded at the rx_data output as E0, 9E, E6, and F8, respectively. The rx_done signal generates a pulse at the end of each UART frame, confirming stable and continuous operation of the integrated UART system under realistic post-implementation timing conditions.