UART DESIGN WITH AMBA APB BUS

Under Supervision of:

Eng. Mohammed Salah

Submitted by:

Ahmed Belal

Table of Contents

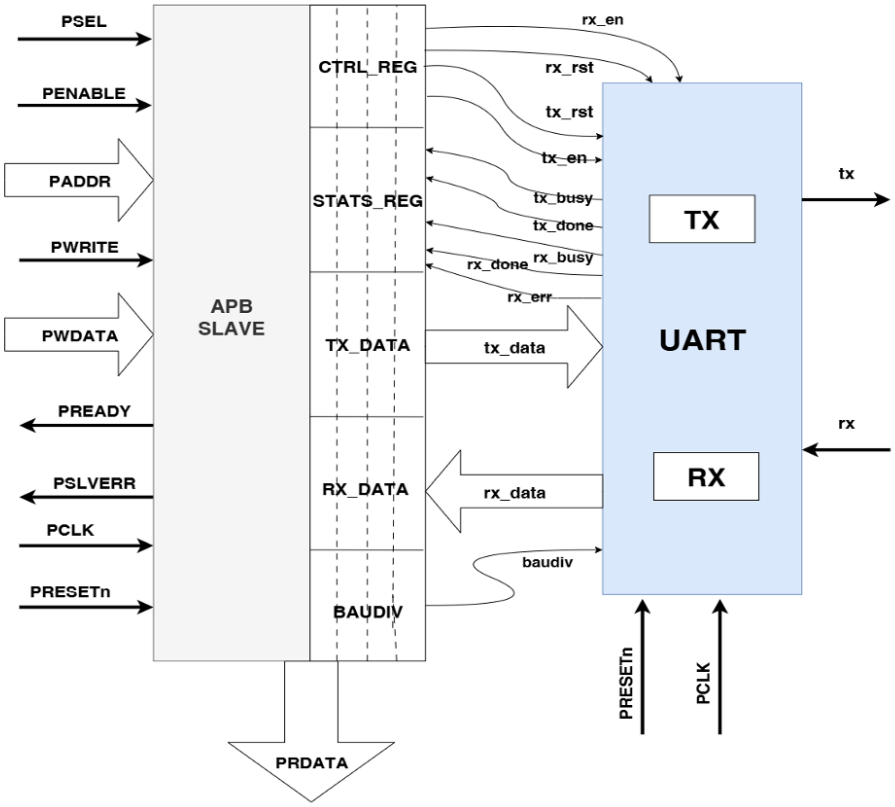
Overview

This project involves the design and implementation of a custom **Universal Asynchronous Receiver Transmitter (UART)** module that is integrated with the **AMBA Advanced Peripheral Bus (APB)** interface. The goal of the design is to create a reusable IP block that can be easily connected as a peripheral in a System-on-Chip (SoC).

The UART module provides reliable **serial communication** by handling data transmission and reception with the addition of start and stop bits. It supports error detection for invalid frames and ensures proper signaling of busy and done states during operation. The APB slave interface allows the UART to be controlled and monitored through memory-mapped registers, enabling configuration of control signals, writing of transmit data, and reading of received data or status.

A **baud rate generator** (Baud module) is included to generate timing pulses based on the system clock, ensuring correct sampling and transmission timing according to the selected baud rate (e.g., 9600 bps).

The design has been verified using a custom testbench that simulates both transmission (TX) and reception (RX) scenarios, ensuring the UART behaves as expected when integrated into a larger system.



APB_Slave Module

1. Purpose

The APB_Slave module acts as the bridge between the APB master (processor/testbench) and the UART module.
It provides:

- Control signals for enabling/resetting the UART transmitter and receiver.
- Registers for writing TX data and reading RX data.
- A status register that reflects UART state (busy, done, error).
- Compliance with the standard APB protocol (IDLE → SETUP → ACCESS).

2. Inputs and Outputs

➢ Inputs (from APB Master)

- **PCLK:** APB clock.
- **PRESETn:** Active-low reset.
- **PSEL:** Slave select (active when APB master selects this peripheral).
- **PENABLE:** Indicates active phase of the APB transaction.
- **PWRITE:** Direction control (1 = Write, 0 = Read).
- **PADDR [31:0]:** Address for register access.
- **PWDATA [31:0]:** Data written from the master.

➢ Outputs (to APB Master)

- **PRDATA [31:0]:** Data returned on read operations.
- **PREADY:** Indicates transfer completion (slave ready).

- Inputs (from UART)
  - **tx_busy:** UART transmitter is busy.
  - **tx_done:** UART transmitter completed sending one frame.
  - **rx_busy:** UART receiver is receiving.
  - **rx_done:** UART receiver finished receiving one frame.
  - **rx_error:** UART detected framing error.
  - **rx_data [7:0]:** Received byte.

- Outputs (to UART)
  - **rx_en:** Enable receiver.
  - **rx_rst:** Reset receiver.
  - **tx_en:** Enable transmitter.
  - **tx_rst:** Reset transmitter.
  - **tx_data [7:0]:** Data to be transmitted.

3. Internal Registers & Their Map

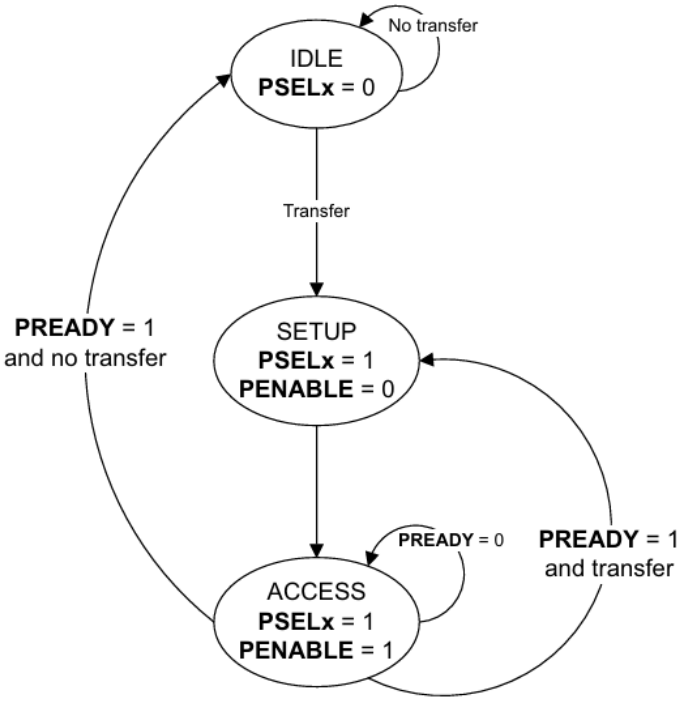
- **CTRL_REG:** Control register. Holds enable/reset bits for TX and RX.
- **STATS_REG:** Status register. Captures UART status signals (busy, done, error).
- **TX_DATA:** Transmit data register. Holds the byte to be sent by UART.
- **RX_DATA:** Receive data register. Stores received byte for APB master to read.

Address	Name	Description
0x0000	CTRL_REG	Contains bits for tx_en , rx_en , tx_rst & rx_rst
0x0001	STATS_REG	Contains bits for rx_busy , tx_busy , rx_done, tx_done & rx_error
0x0002	TX_DATA	UART Tx data
0x0003	RX_DATA	UART Rx data

4. APB State Machine

The APB protocol has three states, represented by cs (current state) and ns (next state):

1. **IDLE:**
    - Waiting for PSEL = 1.
    - Transitions to SETUP when selected.
2. **SETUP:**
    - Address and control signals are valid.
    - If PENABLE = 1, transitions to ACCESS.
3. **ACCESS:**
    - Actual transfer phase.
    - If PWRITE = 1: write to internal register.
    - If PWRITE = 0: read from internal register.
    - PREADY is asserted to indicate completion.
    - Returns to IDLE when PSEL = 0.



5. Control Register Signals (32 Bits)

- Bit **0** → Enable RX.
- Bit **1** → Reset RX.
- Bit **2** → Reset TX.
- Bit **3** → Enable TX.

6. Status Register Signals (32 Bits)

STATS_REG is updated each cycle from UART signals:

- Bit **0** → TX Busy.
- Bit **1** → TX Done.
- Bit **2** → RX Busy.
- Bit **3** → RX Done.
- Bit **4**→ RX Error.

7. TX and RX Data Registers (32 Bits)

- **TX_DATA:** Written by the APB master. Forwarded directly to UART (tx_data).
- **RX_DATA:** Updated whenever rx_done = 1. Makes the last received byte available to APB master.

8. APB SLAVE Code:

```
module APB_Slave(PCLK,PRESETn,PSEL,PENABLE,PWRITE,PADDR,PWDATA,PRDATA,PREADY
                ,tx_busy,tx_done,rx_busy,rx_done,rx_error,rx_data,rx_en
                ,rx_rst,tx_en,tx_rst,tx_data);

parameter IDLE = 2'b00 ;
parameter SETUP = 2'b01 ;
parameter ACCESS = 2'b10 ;
parameter ADDR_CTRL_REG = 32'h00000000 ;
parameter ADDR_STATS_REG = 32'h00000001 ;
parameter ADDR_TX_REG = 32'h00000002 ;
parameter ADDR_RX_REG = 32'h00000003 ;

//MASTER -----> APB SLAVE
input PCLK,PRESETn,PSEL,PENABLE,PWRITE ;
input [31:0] PADDR,PWDATA ;

//APB SLAVE -----> MASTER
output reg [31:0] PRDATA;
output reg PREADY ;

//UART -----> APB SLAVE
input tx_busy,tx_done,rx_busy,rx_done,rx_error ;
input [7:0] rx_data;

//APB SLAVE -----> UART
output rx_en,rx_rst,tx_en,tx_rst ;
output [7:0] tx_data ;

//REGISTERS VALUES
reg [31:0] CTRL_REG,STATS_REG,TX_DATA,RX_DATA ;

//CS & NS
reg [1:0] cs,ns ;

always @(posedge PCLK or negedge PRESETn) begin
    if (~PRESETn)
        cs <= IDLE ;
    else
        cs <= ns ;
end

```

```
42  //NEXT STATE LOGIC
43  always @(*) begin
44
45      case (cs)
46          IDLE: begin
47              if (PSEL)
48                  ns = SETUP ;
49              else
50                  ns = IDLE ;
51          end
52
53          SETUP: begin
54              if (PSEL & PENABLE)
55                  ns = ACCESS ;
56              else
57                  ns = SETUP ;
58          end
59
60          ACCESS: begin
61              if (PSEL & PREADY)
62                  ns = SETUP ;
63              else if (~PSEL)
64                  ns = IDLE ;
65              else
66                  ns = ACCESS ;
67          end
68
69      endcase
70  end
71
```

```
72  //OUTPUT LOGIC
73  always @(posedge PCLK or negedge PRESETn) begin
74
75      if(~PRESETn) begin
76          CTRL_REG <= 0;
77          TX_DATA  <= 0;
78          PREADY   <= 0;
79          PRDATA   <= 0;
80      end
81      else begin
82          case (cs)
83
84              IDLE: begin
85                  PRDATA <= 0 ;
86                  PREADY <= 0 ;
87              end
88
89              SETUP: begin
90                  PRDATA <= 0 ;
91                  PREADY <= 0 ;
92              end
93
94              ACCESS: begin
95                  if (PWRITE) begin
96                      if (PENABLE) begin
97                          PREADY <= 1 ;
98
99                          //to choose which regisiter will take the Write data
100                         if (PADDR == ADDR_CTRL_REG) begin
101                             CTRL_REG <= PWDATA ;
102                         end
103                         else begin
104                             TX_DATA <= PWDATA ;
105                         end
106                     end
107                     else
108                         PREADY <= 0 ;
109                 end
110                 else begin
111
112                     if (PENABLE) begin
113                         PREADY <= 1 ;
114
115                         if (PADDR == ADDR_RX_REG)
116                             PRDATA <= RX_DATA ;
117                         else
118                             PRDATA <= STATS_REG ;
119                     end
120                     else begin
121                         PREADY <= 0 ;
122                     end
123
124                 end
125             end
126         endcase
127     end
128 end
```

```
129
130
131   //CTRL_REG SIGNALS
132       assign rx_en = CTRL_REG[0] ;
133       assign rx_rst = CTRL_REG[1] ;
134       assign tx_rst = CTRL_REG[2] ;
135       assign tx_en = CTRL_REG[3] ;
136
137   //TX_REG SIGNALS
138       assign tx_data = TX_DATA ;
139
140   //STATS_REG SIGNALS
141   always @(posedge PCLK or negedge PRESETn) begin
142       if (~PRESETn)
143           STATS_REG <= 0 ;
144       else begin
145           STATS_REG[0] <= tx_busy ;
146           STATS_REG[1] <= tx_done ;
147           STATS_REG[2] <= rx_busy ;
148           STATS_REG[3] <= rx_done ;
149           STATS_REG[4] <= rx_error ;
150           STATS_REG[31:5] <= 0 ;
151       end
152   end
153
154   //RX_REG SIGNALS
155   always @(posedge PCLK or negedge PRESETn) begin
156       if(~PRESETn)
157           RX_DATA <= 0 ;
158       else if(rx_done) begin
159           RX_DATA[7:0] <= rx_data ;
160       end
161   end
162
163   endmodule
```

UART Module

Imp. Note: The UART module in this project was implemented using a counter-based sequential logic approach instead of the traditional FSM (state machine) method.

1. Purpose

The UART_module is responsible for handling the serial transmission (TX) and serial reception (RX) of data.
It communicates with the APB slave interface, which provides control and data registers. The UART adds start and stop bits to the transmitted data, shifts data out on the TX pin, and samples data on the RX pin at the correct baud intervals.

2. Inputs and Outputs

➢ Inputs

- **PCLK:** System clock used for synchronous logic.
- **PRESETn:** Active-low reset to initialize the module.
- **rx_en:** Enable signal for the receiver.
- **rx_rst:** Reset signal for the receiver.
- **tx_en:** Enable signal for the transmitter.
- **tx_rst:** Reset signal for the transmitter.
- **tx_data [7:0]:** Parallel byte to be transmitted.
- **baud_tick:** Baud clock generated from the baud generator. Used to control when bits are shifted in/out.
- **RX:** Serial data input pin (data received from outside).

➢ Outputs

- **TX:** Serial data output pin (data transmitted out).
- **tx_busy:** High while transmission is in progress.
- **tx_done:** High when transmission of one byte is finished.
- **rx_busy:** High while reception is in progress.
- **rx_done:** High when a full byte is received.
- **rx_error:** High when framing error occurs (wrong start/stop bits).
- **rx_data [7:0]:** Parallel byte received after successful reception.

3. Internal Registers

- **tx_reg [9:0]:** Holds the transmit frame (start bit + 8 data bits + stop bit).
- **tx_counter [3:0]:** Tracks which bit of the frame is currently being transmitted.
- **rx_reg [9:0]:** Holds the received frame (start bit + 8 data bits + stop bit).
- **rx_counter [3:0]:** Tracks how many bits have been sampled during reception.

4. Transmitter Operation (TX Path)

1. **Loading Data:**
    - When tx_busy is low and a new byte is available, tx_reg is loaded with {start bit, tx_data, stop bit}.
2. **Shifting Data Out:**
    - On each baud_tick pulse (while tx_en is high), the transmitter outputs tx_reg[tx_counter] onto the TX pin.
    - tx_counter increments with each baud tick.
3. **Completion:**
    - When tx_counter reaches 0 (all 10 bits sent), tx_done is set high, tx_busy is cleared, and the counter resets.

5. Receiver Operation (RX Path)

1. **Reset/Initialization:**
    - rx_counter is initialized to 9 (waiting for a frame).
2. **Sampling Data:**
    - When rx_en is high and a baud_tick pulse occurs, the UART samples the RX line.
    - Bits are shifted into rx_reg [rx_counter].
    - rx_counter decrements each time.
3. **Completion:**
    - When rx_counter reaches 0, a full frame has been received.
    - rx_data is updated with rx_reg[8:1] (removing start/stop bits).
    - rx_done is set high, rx_busy is cleared, and rx_counter is reset to 9.

6. Error Detection

- After reception, the UART checks the frame:
    - rx_reg[0] must equal start bit (0).
    - rx_reg[9] must equal stop bit (1).
- If either check fails, rx_error is asserted.
- Otherwise, rx_error remains 0.

7. UART Code:

```
module UART_module (PRESETn,PCLK,rx_en,rx_rst,tx_en,tx_rst,tx_data,
                    tx_busy,tx_done,rx_busy,rx_done,rx_error,rx_data,baud_tick,RX,TX);

parameter START_BIT = 0 ;
parameter END_BIT= 1 ;

//CLK and RESET
input PRESETn,PCLK ;

//UART -----> APB SLAVE
input rx_en,rx_rst,tx_en,tx_rst ;
input [7:0] tx_data ;

//APB SLAVE -----> UART
output reg tx_busy,tx_done,rx_busy,rx_done,rx_error ;
output reg [7:0] rx_data;

//BAUD CLK
input baud_tick ;

//UART INPUT
input RX ;

//UART OUTPUT
output reg TX ;

reg [9:0] tx_reg ;
reg [3:0] tx_counter ;
reg [9:0] rx_reg ;
reg [3:0] rx_counter ;

```

```
/*Always Block that give the tx_data from APB SLAVE -----> the UART
,and only updated when the tx_busy is LOW ,also add the START and END_BIT bits */
always @(posedge PCLK or negedge PRESETn) begin
        if (~PRESETn)
            tx_reg <= 0;
        else if (tx_rst)
            tx_reg <= 0;
        else if (~tx_busy)
            tx_reg <= {1'b0,tx_data,1'b1} ;
end


always @(posedge PCLK or negedge PRESETn) begin
    if(~PRESETn) begin
        TX <= 1 ;
        tx_counter <= 9 ;
        tx_busy <= 0 ;
        tx_done <= 0 ;
    end
    else if(tx_rst) begin
        TX <= 1 ;
        tx_counter <= 9 ;
        tx_busy <= 0 ;
        tx_done <= 0 ;
    end
    else begin
        if (tx_en & baud_tick) begin
            TX <= tx_reg[tx_counter] ;
            tx_counter <= tx_counter - 1 ;

            if (tx_counter == 0) begin
                tx_busy <= 0 ;
                tx_done <= 1 ;
                tx_counter <= 9 ;
            end
            else begin
                tx_busy <= 1 ;
                tx_done <= 0 ;
            end
        end
    end
end

```

```
/*Always Block that give the rx_data from UART -----> APB SLAVE
,and only updated when the rx_done is LOW ,also remove the START and END_BIT bits */
always @(posedge PCLK or negedge PRESETn) begin
    if(~PRESETn) begin
        rx_counter <= 9;
        rx_busy <= 0 ;
        rx_done <= 0 ;
        rx_reg <= 0 ;
    end
    else if(rx_rst) begin
        rx_counter <= 9;
        rx_busy <= 0 ;
        rx_done <= 0 ;
        rx_reg <= 0 ;
    end
    else begin
        if (rx_en & baud_tick & !rx_done) begin
            rx_counter <= rx_counter - 1 ;
            rx_reg[rx_counter] <= RX ;

            if (rx_counter == 0) begin
                rx_busy <= 0 ;
                rx_done <= 1 ;
                rx_data <= rx_reg [8:1] ;
                rx_counter <= 9 ;
            end
            else
                rx_busy <= 1 ;
                // rx_done <= 0 ;

        end
    end
end

//PRESET that reset all the outputs and the rx_error signal
always @(posedge PCLK or negedge PRESETn) begin
    if (~PRESETn) begin
        rx_error <= 0 ;
    end
    else if (rx_done && (rx_reg[0] != 1'b1 || rx_reg[9] != 1'b0))
        rx_error <= 1 ;
    else
        rx_error <= 0 ;
end

endmodule
```

Baud Generator Module

1. Purpose

The Baud module generates the baud clock (baud_tick) required for UART transmission and reception.
This baud clock defines the bit period — ensuring each transmitted or received bit is sampled/sent at the correct time.

2. Inputs and Outputs

- Inputs
  - **clk:** System clock (100 MHz in your design).
  - **rst:** Active-low reset.
- Outputs
  - **baud_tick:** Baud clock pulse.
    - Asserted high (1) once every FINAL cycles of the system clock.
    - Used by UART TX and RX FSMs to sample or shift bits.

3. Parameters

- **BAUD:** Desired baud rate (9600 bps).
- **FREQUENCY:** Input clock frequency (100 MHz).

4. Calculation

- FINAL is the number of system clock cycles per UART bit period.
- For 100 MHz / 9600 baud:

$$FINAL = 100{,}000{,}000/9600 \approx 10417$$

- This means one baud tick (baud_tick = 1) occurs every 10,417 system clock cycles.

5. Internal Signals

- **counter:** 20-bit register. Counts system clock cycles up to FINAL - 1.

6. Behavior

- On reset: counter = 0, baud_tick = 0.
- On each system clock cycle:
    - counter increments until it reaches FINAL - 1.
    - When it does:
        - counter resets to 0.
        - baud_tick is pulsed HIGH (1) for one cycle only.
- At all other times, baud_tick = 0.

7. Function in UART System

- **Transmitter:** Uses baud_tick pulses to shift out each bit (start, data, stop) at the correct rate.
- **Receiver:** Uses baud_tick pulses to sample the RX line at each bit boundary.

Thus, baud_tick synchronizes UART communication with the configured baud rate.

8. BAUD Rate

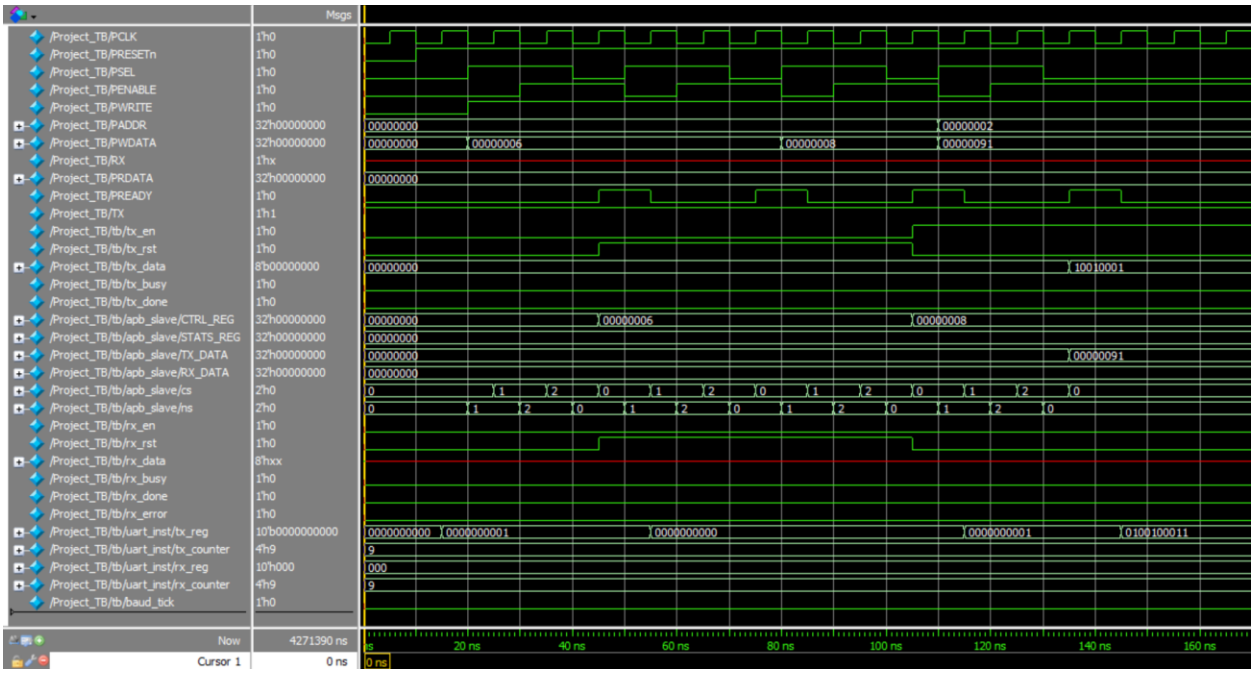
```
1   module Baud(clk,rst,baud_tick);
2
3   parameter BAUD = 9600 ;
4   parameter FREQUENCY = 100000000 ;
5
6   localparam FINAL = (FREQUENCY / (BAUD)) ;
7
8   input clk,rst ;
9   output reg baud_tick ;
10
11  reg [19:0]counter ;
12
13  always @(posedge clk or negedge rst) begin
14      if (~rst) begin
15          counter <= 0 ;
16          baud_tick <=0 ;
17      end
18      else if(counter == FINAL-1) begin
19          counter <= 0 ;
20          baud_tick <= 1 ;
21      end
22      else begin
23          counter <= counter + 1 ;
24          baud_tick <= 0 ;
25      end
26  end
27
28  endmodule
```

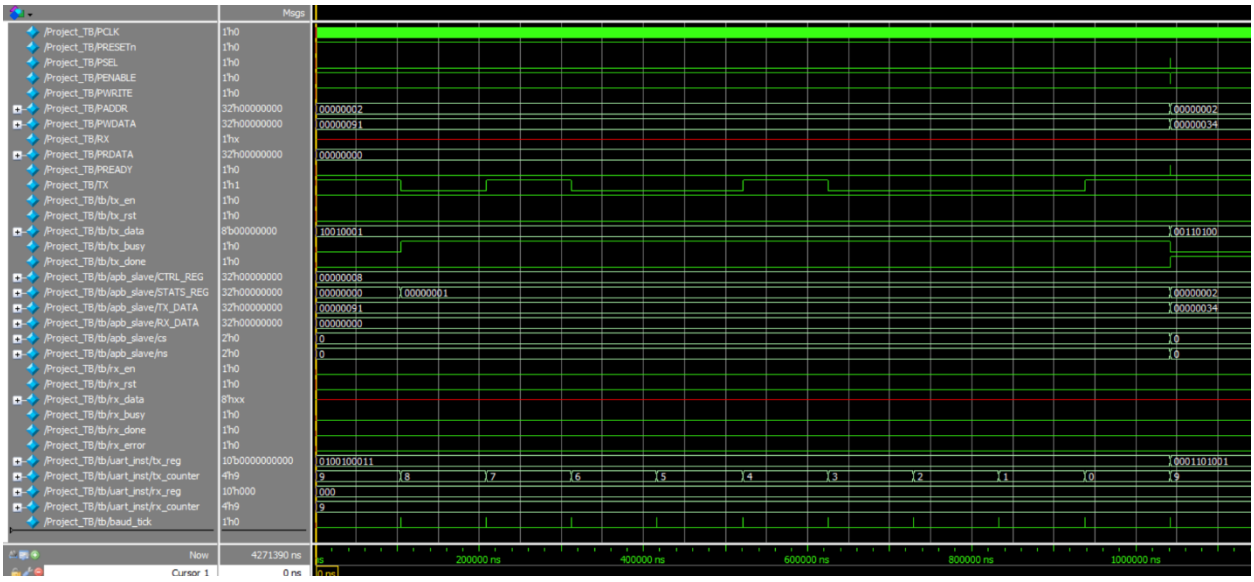
TOP Module

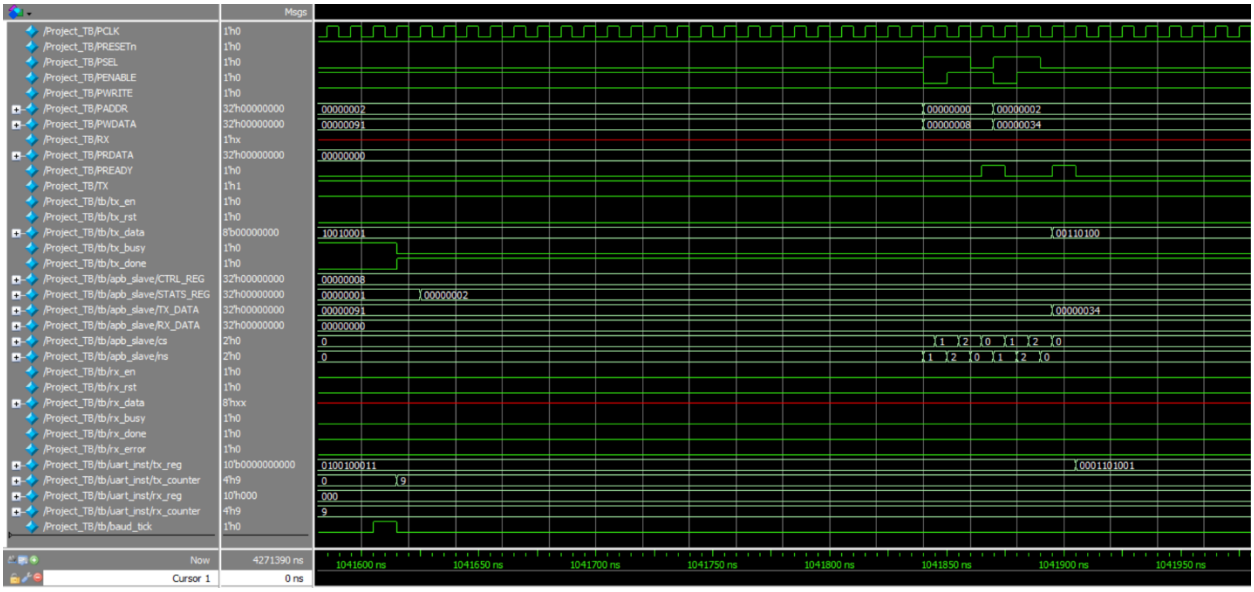
```
module UART_APB_Top (PCLK,PRESETn,PSEL,PENABLE,PWRITE,PADDR,
                     PWDATA,PRDATA,PREADY,RX,TX);

input  PCLK ;           // System clock
input  PRESETn ;        // Active-low reset

// APB interface
input PSEL,PENABLE,PWRITE ;
input [31:0] PADDR ;
input [31:0] PWDATA ;
output [31:0] PRDATA ;
output       PREADY ;

// UART pins
input  RX ;   // UART receive pin
output TX ;  // UART transmit pin

// Internal wires
wire rx_en, rx_rst, tx_en, tx_rst;
wire [7:0] tx_data;
wire [7:0] rx_data;
wire tx_busy, tx_done, rx_busy, rx_done, rx_error;

// Baud clock
wire baud_tick;

    // Instantiate Baud generator
    Baud baud_gen (
        .clk(PCLK),
        .rst(PRESETn),
        .baud_tick(baud_tick)
    );

```

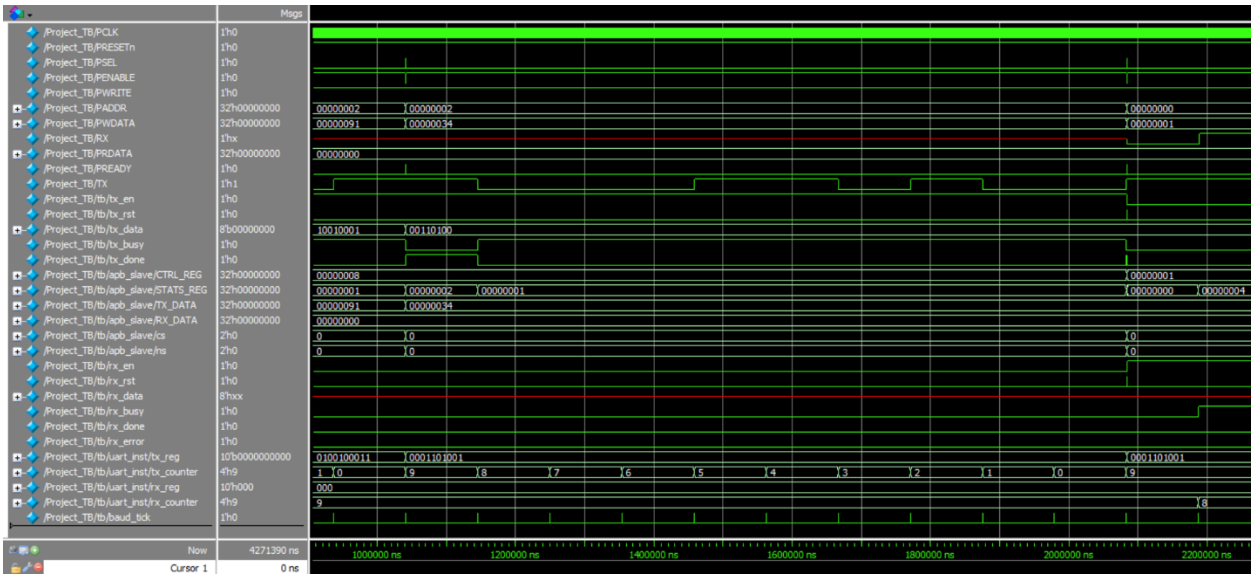
```
    // Instantiate UART
    UART_module uart_inst (
        .PRESETn(PRESETn),
        .PCLK(PCLK),
        .rx_en(rx_en),
        .rx_rst(rx_rst),
        .tx_en(tx_en),
        .tx_rst(tx_rst),
        .tx_data(tx_data),
        .tx_busy(tx_busy),
        .tx_done(tx_done),
        .rx_busy(rx_busy),
        .rx_done(rx_done),
        .rx_error(rx_error),
        .rx_data(rx_data),
        .baud_tick(baud_tick),
        .RX(RX),
        .TX(TX)
    );

    // Instantiate APB Slave
    APB_Slave apb_slave (
        .PCLK(PCLK),
        .PRESETn(PRESETn),
        .PSEL(PSEL),
        .PENABLE(PENABLE),
        .PWRITE(PWRITE),
        .PADDR(PADDR),
        .PWDATA(PWDATA),
        .PRDATA(PRDATA),
        .PREADY(PREADY),

        // UART -> APB
        .tx_busy(tx_busy),
        .tx_done(tx_done),
        .rx_busy(rx_busy),
        .rx_done(rx_done),
        .rx_error(rx_error),
        .rx_data(rx_data),

        // APB -> UART
        .rx_en(rx_en),
        .rx_rst(rx_rst),
        .tx_en(tx_en),
        .tx_rst(tx_rst),
        .tx_data(tx_data)
    );

endmodule
```

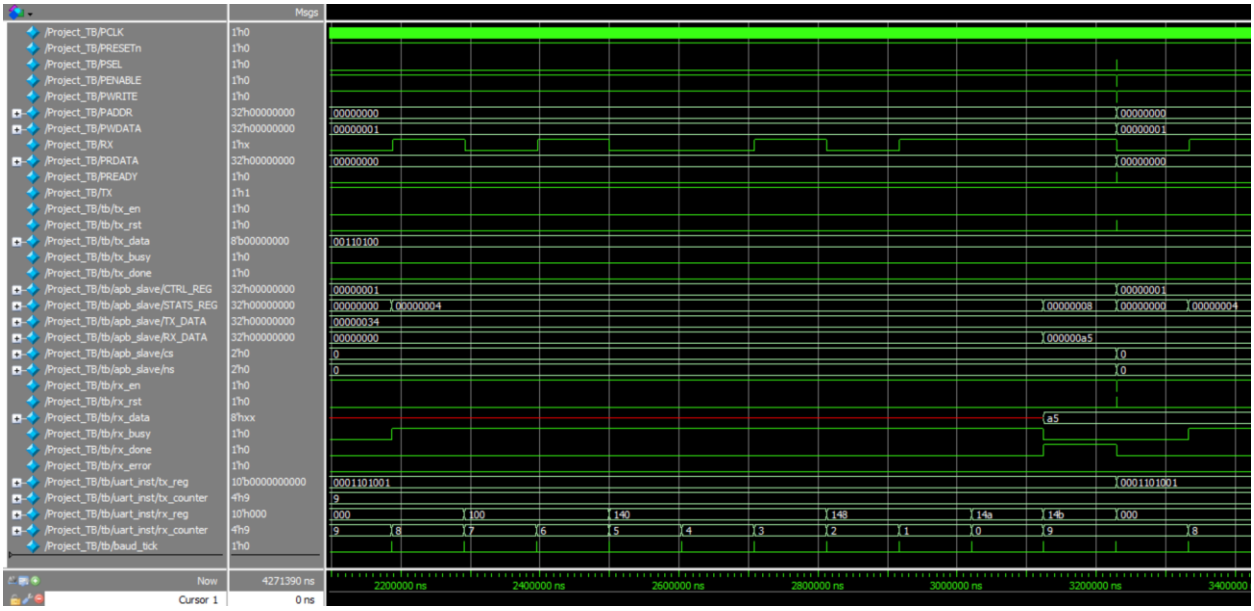
Questa Sim Snippets (TB Code in GITHUB)

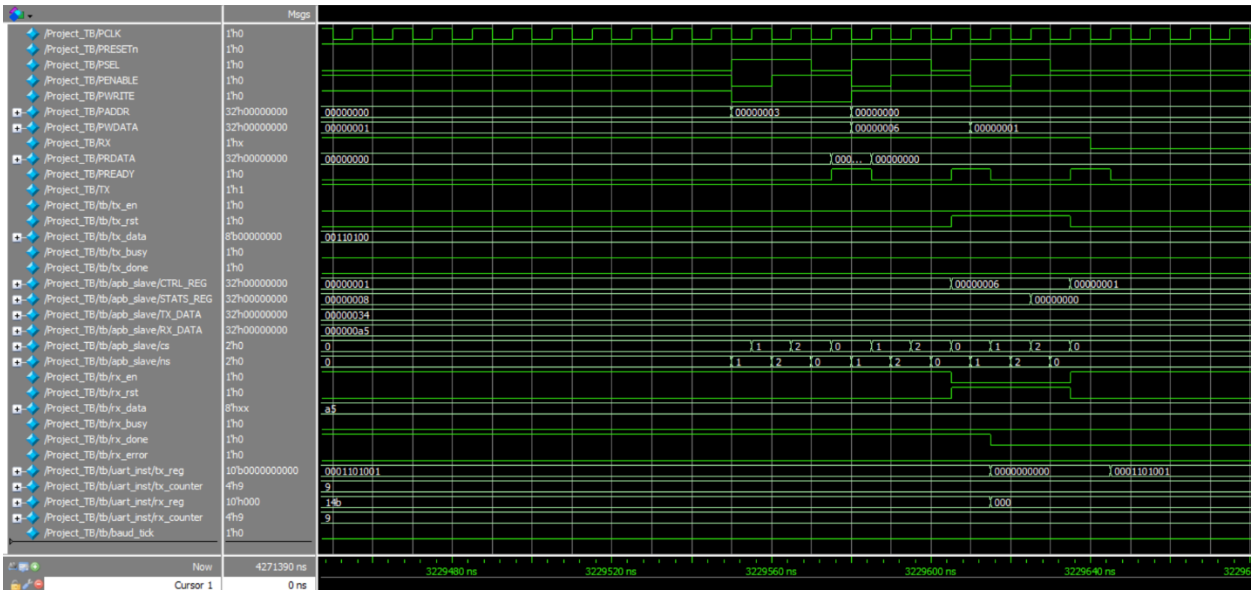


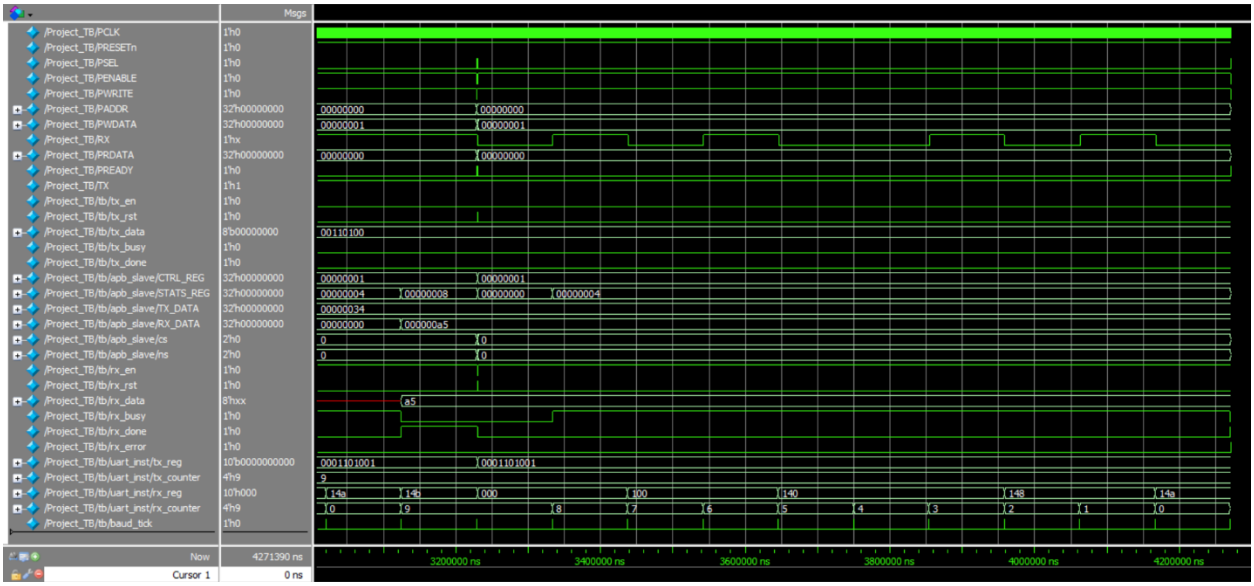


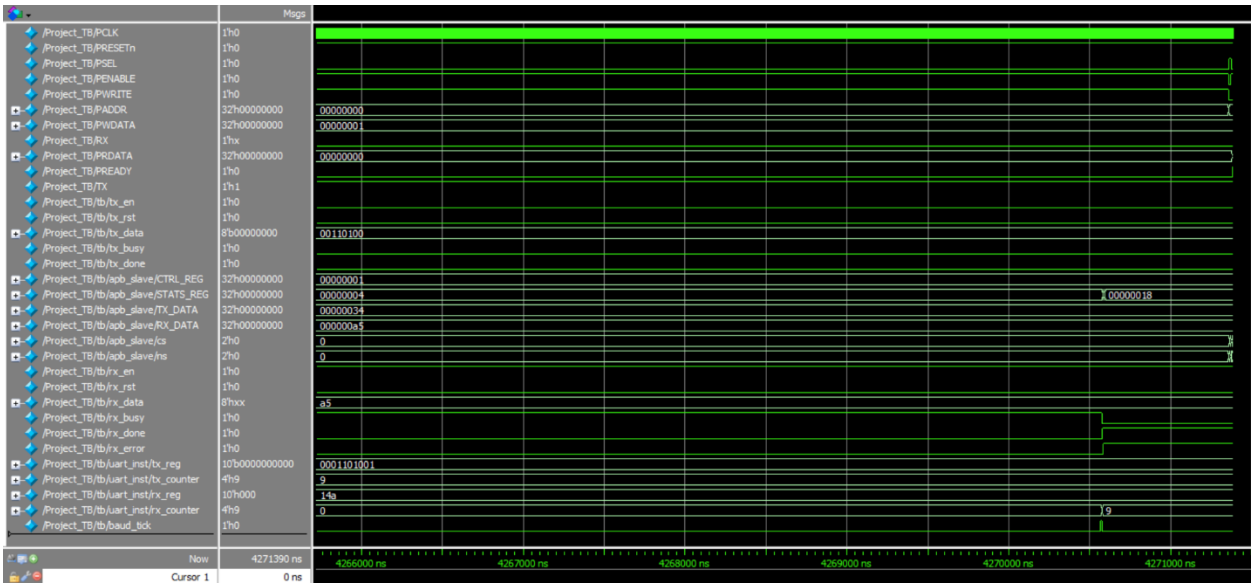

/Project_TB/PCLK
/Project_TB/PRESETn
/Project_TB/PSEL
/Project_TB/PENABLE
/Project_TB/PWRITE
/Project_TB/PADDR
/Project_TB/PWDATA
/Project_TB/RX
/Project_TB/PRDATA
/Project_TB/PREADY
/Project_TB/TX
/Project_TB/tb/tx_en
/Project_TB/tb/tx_rst
/Project_TB/tb/tx_data
/Project_TB/tb/tx_busy
/Project_TB/tb/tx_done
/Project_TB/tb/apb_slave/CTRL_REG
/Project_TB/tb/apb_slave/STATS_REG
/Project_TB/tb/apb_slave/TX_DATA
/Project_TB/tb/apb_slave/RX_DATA
/Project_TB/tb/apb_slave/cs
/Project_TB/tb/apb_slave/ns
/Project_TB/tb/rx_en
/Project_TB/tb/rx_rst
/Project_TB/tb/rx_data
/Project_TB/tb/rx_busy
/Project_TB/tb/rx_done
/Project_TB/tb/rx_error
/Project_TB/tb/uart_inst/tx_reg
/Project_TB/tb/uart_inst/tx_counter
/Project_TB/tb/uart_inst/rx_reg
/Project_TB/tb/uart_inst/rx_counter
/Project_TB/tb/baud_tick
Msgs
Now
Cursor 1
1041600 ns
1041650 ns
1041700 ns
1041750 ns
1041800 ns
1041850 ns
1041900 ns
1041950 ns


/Project_TB/PCLK
/Project_TB/PRESETn
/Project_TB/PSEL
/Project_TB/PENABLE
/Project_TB/PWRITE
/Project_TB/PADDR
/Project_TB/PWDATA
/Project_TB/RX
/Project_TB/PRDATA
/Project_TB/PREADY
/Project_TB/TX
/Project_TB/tb/tx_en
/Project_TB/tb/tx_rst
/Project_TB/tb/tx_data
/Project_TB/tb/tx_busy
/Project_TB/tb/tx_done
/Project_TB/tb/apb_slave/CTRL_REG
/Project_TB/tb/apb_slave/STATS_REG
/Project_TB/tb/apb_slave/TX_DATA
/Project_TB/tb/apb_slave/RX_DATA
/Project_TB/tb/apb_slave/cs
/Project_TB/tb/apb_slave/ns
/Project_TB/tb/rx_en
/Project_TB/tb/rx_rst
/Project_TB/tb/rx_data
/Project_TB/tb/rx_busy
/Project_TB/tb/rx_done
/Project_TB/tb/rx_error
/Project_TB/tb/uart_inst/tx_reg
/Project_TB/tb/uart_inst/tx_counter
/Project_TB/tb/uart_inst/rx_reg
/Project_TB/tb/uart_inst/rx_counter
/Project_TB/tb/baud_tick
Msgs
Now
Cursor 1
1000000 ns
1200000 ns
1400000 ns
1600000 ns
1800000 ns
2000000 ns
2200000 ns

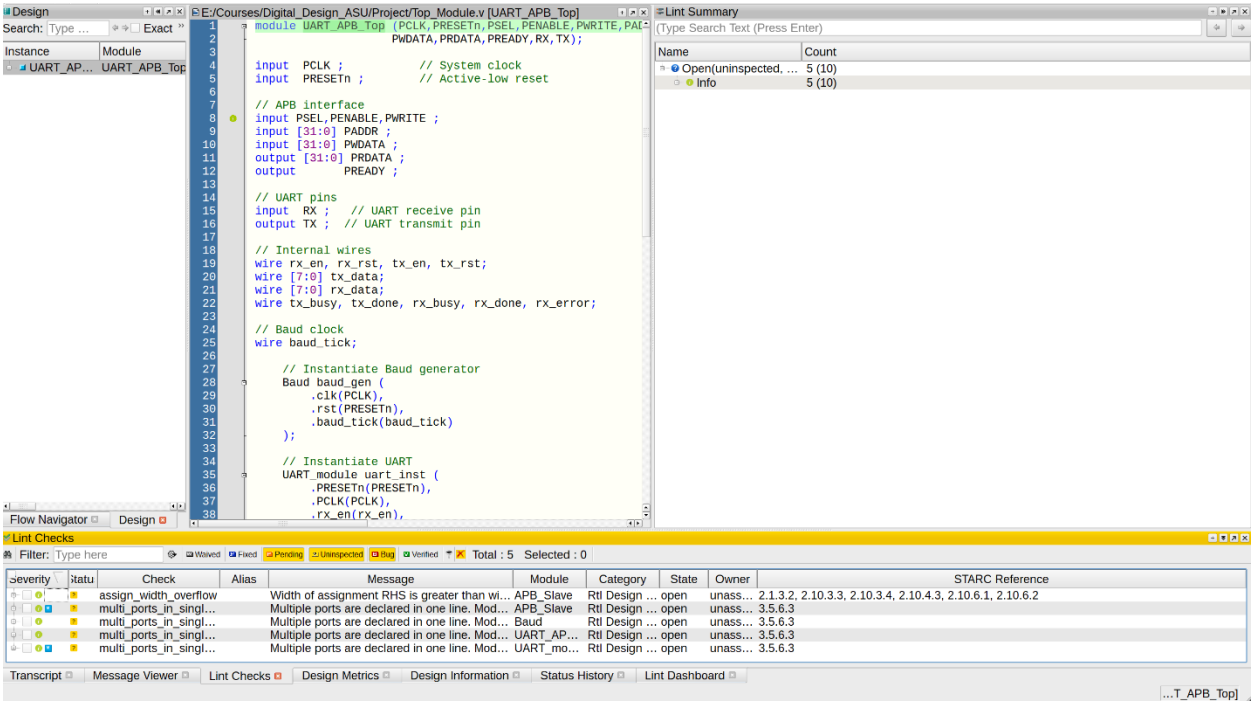

Msgs
/Project_TB/PCLK
/Project_TB/PRESETn
/Project_TB/PSEL
/Project_TB/PENABLE
/Project_TB/PWRITE
/Project_TB/PADDR
/Project_TB/PWDATA
/Project_TB/RX
/Project_TB/PRDATA
/Project_TB/PREADY
/Project_TB/TX
/Project_TB/tb/tx_en
/Project_TB/tb/tx_rst
/Project_TB/tb/tx_data
/Project_TB/tb/tx_busy
/Project_TB/tb/tx_done
/Project_TB/tb/apb_slave/CTRL_REG
/Project_TB/tb/apb_slave/STATS_REG
/Project_TB/tb/apb_slave/TX_DATA
/Project_TB/tb/apb_slave/RX_DATA
/Project_TB/tb/apb_slave/cs
/Project_TB/tb/apb_slave/ns
/Project_TB/tb/rx_en
/Project_TB/tb/rx_rst
/Project_TB/tb/rx_data
/Project_TB/tb/rx_busy
/Project_TB/tb/rx_done
/Project_TB/tb/rx_error
/Project_TB/tb/uart_inst/tx_reg
/Project_TB/tb/uart_inst/tx_counter
/Project_TB/tb/uart_inst/rx_reg
/Project_TB/tb/uart_inst/rx_counter
/Project_TB/tb/baud_tick
Now
4271390 ns
Cursor 1
0 ns
2200000 ns
2400000 ns
2600000 ns
2800000 ns
3000000 ns
3200000 ns
3400000


Msgs
/Project_TB/PCLK
/Project_TB/PRESETn
/Project_TB/PSEL
/Project_TB/PENABLE
/Project_TB/PWRITE
/Project_TB/PADDR
/Project_TB/PWDATA
/Project_TB/RX
/Project_TB/PRDATA
/Project_TB/PREADY
/Project_TB/TX
/Project_TB/tb/tx_en
/Project_TB/tb/tx_rst
/Project_TB/tb/tx_data
/Project_TB/tb/tx_busy
/Project_TB/tb/tx_done
/Project_TB/tb/apb_slave/CTRL_REG
/Project_TB/tb/apb_slave/STATS_REG
/Project_TB/tb/apb_slave/TX_DATA
/Project_TB/tb/apb_slave/RX_DATA
/Project_TB/tb/apb_slave/cs
/Project_TB/tb/apb_slave/ns
/Project_TB/tb/rx_en
/Project_TB/tb/rx_rst
/Project_TB/tb/rx_data
/Project_TB/tb/rx_busy
/Project_TB/tb/rx_done
/Project_TB/tb/rx_error
/Project_TB/tb/uart_inst/tx_reg
/Project_TB/tb/uart_inst/tx_counter
/Project_TB/tb/uart_inst/rx_reg
/Project_TB/tb/uart_inst/rx_counter
/Project_TB/tb/baud_tick
Now
4271390 ns
Cursor 1
0 ns
3229480 ns
3229520 ns
3229560 ns
3229600 ns
3229640 ns


Msgs
/Project_TB/PCLK
/Project_TB/PRESETn
/Project_TB/PSEL
/Project_TB/PENABLE
/Project_TB/PWRITE
/Project_TB/PADDR
/Project_TB/PWDATA
/Project_TB/RX
/Project_TB/PRDATA
/Project_TB/PREADY
/Project_TB/TX
/Project_TB/tb/tx_en
/Project_TB/tb/tx_rst
/Project_TB/tb/tx_data
/Project_TB/tb/tx_busy
/Project_TB/tb/tx_done
/Project_TB/tb/apb_slave/CTRL_REG
/Project_TB/tb/apb_slave/STATS_REG
/Project_TB/tb/apb_slave/TX_DATA
/Project_TB/tb/apb_slave/RX_DATA
/Project_TB/tb/apb_slave/cs
/Project_TB/tb/apb_slave/ns
/Project_TB/tb/rx_en
/Project_TB/tb/rx_rst
/Project_TB/tb/rx_data
/Project_TB/tb/rx_busy
/Project_TB/tb/rx_done
/Project_TB/tb/rx_error
/Project_TB/tb/uart_inst/tx_reg
/Project_TB/tb/uart_inst/tx_counter
/Project_TB/tb/uart_inst/rx_reg
/Project_TB/tb/uart_inst/rx_counter
/Project_TB/tb/baud_tick
Now
Cursor 1
3200000 ns
3400000 ns
3600000 ns
3800000 ns
4000000 ns
4200000 ns


Msgs
/Project_TB/PCLK
/Project_TB/PRESETn
/Project_TB/PSEL
/Project_TB/PENABLE
/Project_TB/PWRITE
/Project_TB/PADDR
/Project_TB/PWDATA
/Project_TB/RX
/Project_TB/PRDATA
/Project_TB/PREADY
/Project_TB/TX
/Project_TB/tb/tx_en
/Project_TB/tb/tx_rst
/Project_TB/tb/tx_data
/Project_TB/tb/tx_busy
/Project_TB/tb/tx_done
/Project_TB/tb/apb_slave/CTRL_REG
/Project_TB/tb/apb_slave/STATS_REG
/Project_TB/tb/apb_slave/TX_DATA
/Project_TB/tb/apb_slave/RX_DATA
/Project_TB/tb/apb_slave/cs
/Project_TB/tb/apb_slave/ns
/Project_TB/tb/rx_en
/Project_TB/tb/rx_rst
/Project_TB/tb/rx_data
/Project_TB/tb/rx_busy
/Project_TB/tb/rx_done
/Project_TB/tb/rx_error
/Project_TB/tb/uart_inst/tx_reg
/Project_TB/tb/uart_inst/tx_counter
/Project_TB/tb/uart_inst/rx_reg
/Project_TB/tb/uart_inst/rx_counter
/Project_TB/tb/baud_tick
Now
Cursor 1
4266000 ns
4267000 ns
4268000 ns
4269000 ns
4270000 ns
4271000 ns

QuestLint Snippets

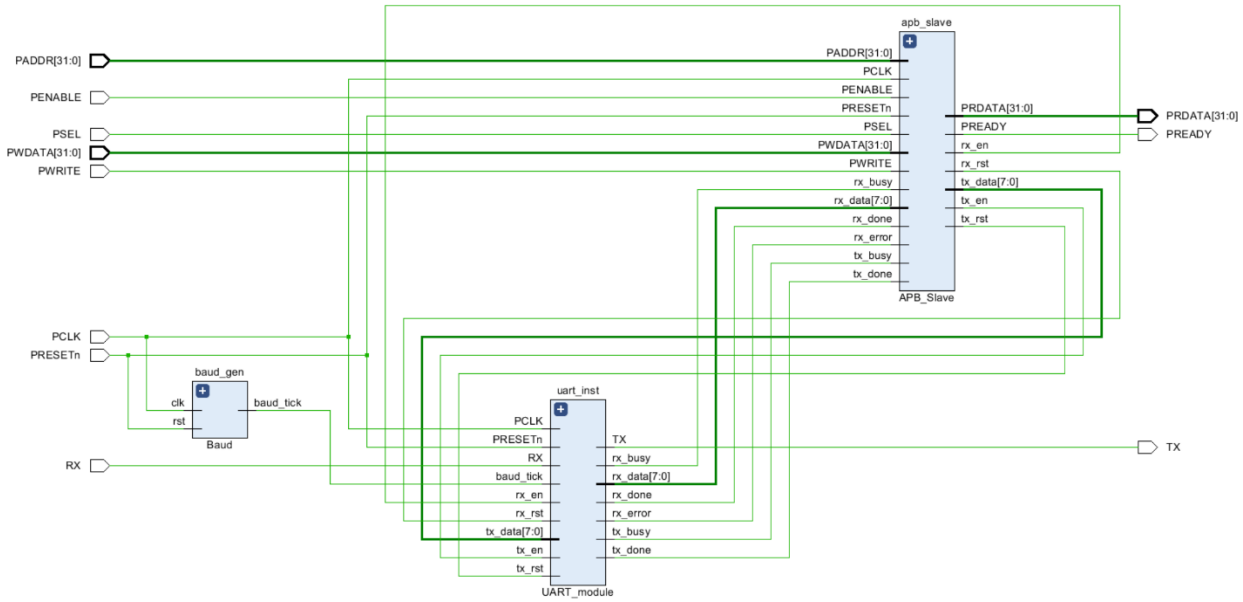


DO File:

```
vlib work
vlog APB_Master.v
vlog Baud.v
vlog UART_module.v
vlog Top_Module.v
vlog Project_TB.v
vsim -voptargs=+acc work.Project_TB
add wave -position insertpoint  \
sim:/Project_TB/PCLK \
sim:/Project_TB/PRESETn \
sim:/Project_TB/PSEL \
sim:/Project_TB/PENABLE \
sim:/Project_TB/PWRITE \
sim:/Project_TB/PADDR \
sim:/Project_TB/PWDATA \
sim:/Project_TB/RX \
sim:/Project_TB/PRDATA \
sim:/Project_TB/PREADY \
sim:/Project_TB/TX
add wave -position insertpoint  \
sim:/Project_TB/tb/tx_en \
sim:/Project_TB/tb/tx_rst \
sim:/Project_TB/tb/tx_data \
sim:/Project_TB/tb/tx_busy \
sim:/Project_TB/tb/tx_done
add wave -position insertpoint  \
sim:/Project_TB/tb/apb_slave/CTRL_REG \
sim:/Project_TB/tb/apb_slave/STATS_REG \
sim:/Project_TB/tb/apb_slave/TX_DATA \
sim:/Project_TB/tb/apb_slave/RX_DATA \
sim:/Project_TB/tb/apb_slave/cs \
sim:/Project_TB/tb/apb_slave/ns
add wave -position insertpoint  \
sim:/Project_TB/tb/rx_en \
sim:/Project_TB/tb/rx_rst \
sim:/Project_TB/tb/rx_data \
sim:/Project_TB/tb/rx_busy \
sim:/Project_TB/tb/rx_done \
sim:/Project_TB/tb/rx_error
add wave -position insertpoint  \
sim:/Project_TB/tb/uart_inst/tx_reg \
sim:/Project_TB/tb/uart_inst/tx_counter \
sim:/Project_TB/tb/uart_inst/rx_reg \
sim:/Project_TB/tb/uart_inst/rx_counter
add wave -position insertpoint  \
sim:/Project_TB/tb/baud_tick
run -all
#quit -sim
```

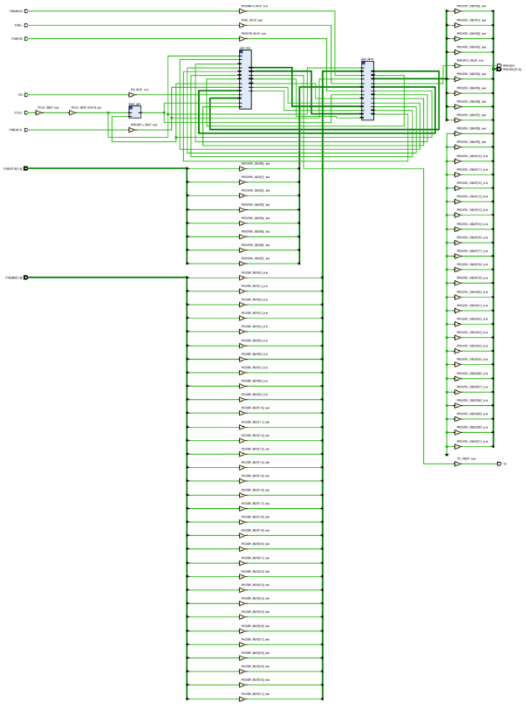
VIVADO

- **Elaboration Schematic:**

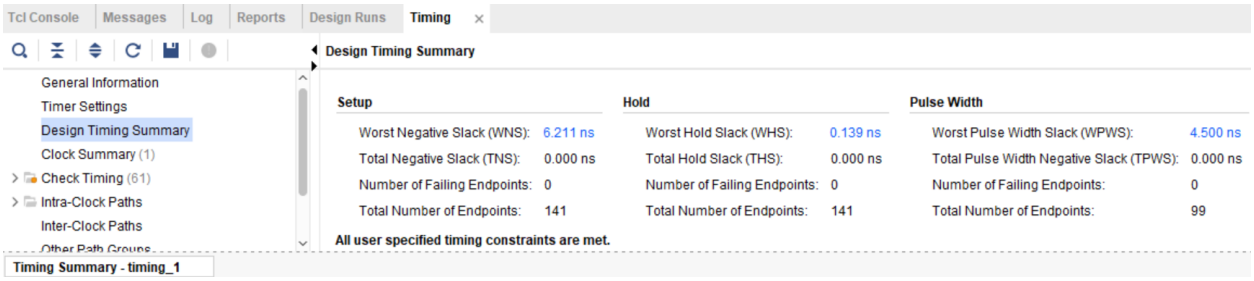


- **Elaboration Massage:**

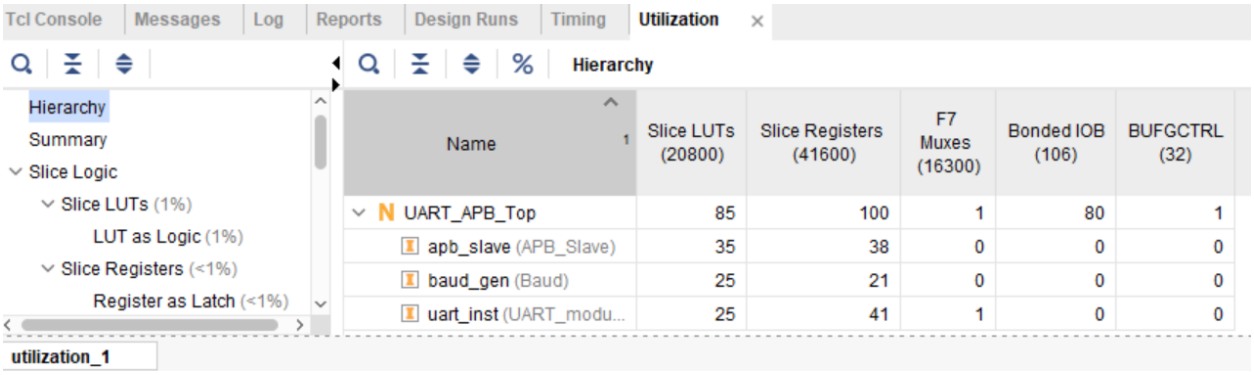
- **Synthesis Schematic:**



- **Synthesis Timing report snippet:**

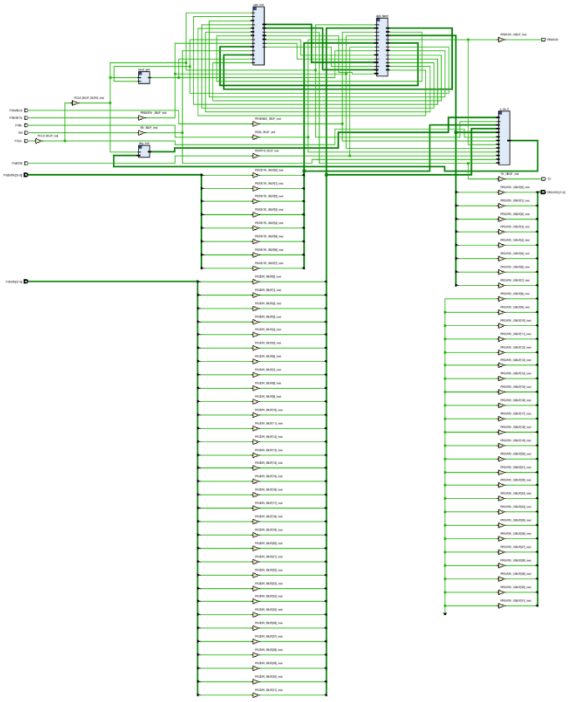


- **Synthesis Utilization report:**

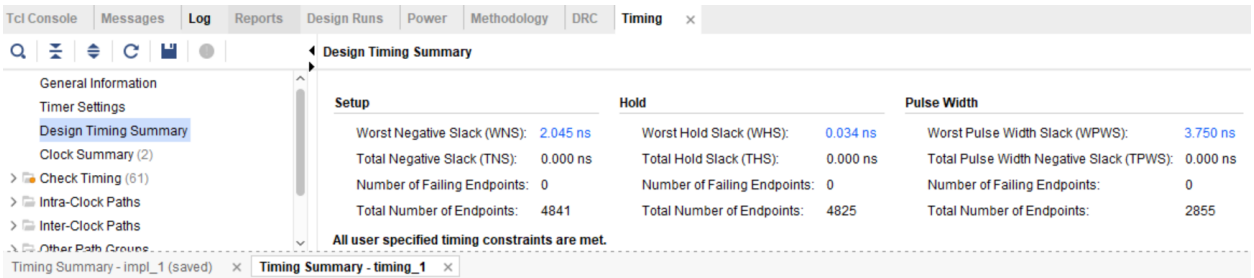


- **Massage after Synthesis:**

- **Implementation Schematic:**



- **Implementation Timing report:**



- **Implementation Utilization report:**

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
N UART_APB_Top	1614	2533	18	776	1436	178	1021	2	80	2	1
apb_slave (APB_Slave)	35	38	0	19	35	0	11	0	0	0	0
baud_gen (Baud)	25	21	0	14	25	0	21	0	0	0	0
dbg_hub (dbg_hub)	475	727	0	217	451	24	316	0	0	1	1
u_ila_0 (u_ila_0)	1055	1706	17	529	901	154	646	2	0	0	0
uart_inst (UART_modu...	24	41	1	13	24	0	18	0	0	0	0

- **Massage after Implementation:**

- **FPGA device snippet:**