



---

# UART DESIGN WITH AMBA APB BUS

---



**Under Supervision of:**

**Eng. Mohammed Salah**

**Submitted by:**

**Ahmed Belal**

## Table of Contents

<b>Overview</b> .....	4
<b>APB_Slave Module</b> .....	5
1. Purpose.....	5
2. Inputs and Outputs .....	5
3. Internal Registers & Their Map.....	6
4. APB State Machine .....	7
5. Control Register Signals (32 Bits).....	8
6. Status Register Signals (32 Bits) .....	8
7. TX and RX Data Registers (32 Bits) .....	8
8. APB SLAVE Code: .....	9
<b>UART Module</b> .....	13
1. Purpose.....	13
2. Inputs and Outputs .....	13
3. Internal Registers.....	14
4. Transmitter Operation (TX Path).....	14
5. Receiver Operation (RX Path).....	15
6. Error Detection.....	15
7. UART Code: .....	16
<b>Baud Generator Module</b> .....	19
1. Purpose.....	19
2. Inputs and Outputs .....	19
3. Parameters.....	19
4. Calculation .....	19
5. Internal Signals .....	20
6. Behavior .....	20
7. Function in UART System .....	20
<b>TOP Module</b> .....	22
<b>Questa Sim Snippets (TB Code in GITHUB)</b> .....	24

<b>QuestLint Snippets .....</b>	28
<b>DO File: .....</b>	29
<b>VIVADO.....</b>	30

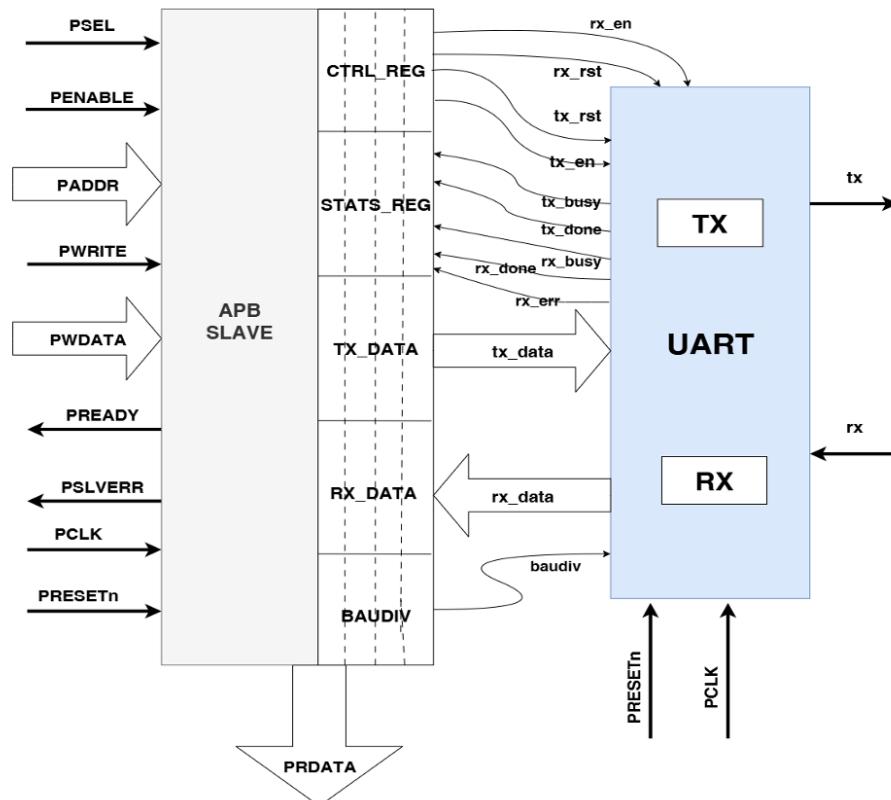
## Overview

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:

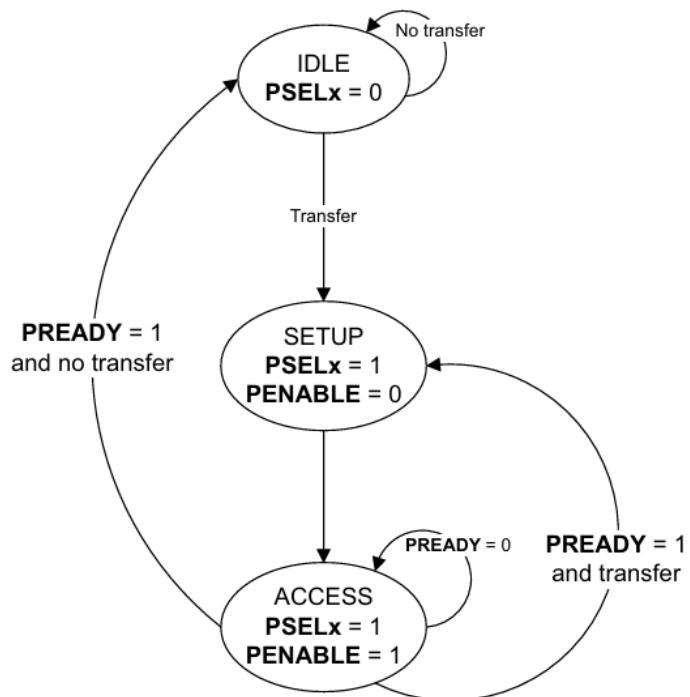
- Waiting for PSEL = 1.
- Transitions to SETUP when selected.

### 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:

```
1  module APB_Slave(PCLK,PRESETn,PSEL,PENABLE,PWRITE,PADDR,PWDATA,PRDATA,PREADY
2  |  |  |  |  ,tx_busy,tx_done,rx_busy,rx_done,rx_error,rx_data,rx_en
3  |  |  |  |  ,rx_rst,tx_en,tx_rst,tx_data);
4
5  parameter IDLE = 2'b00 ;
6  parameter SETUP = 2'b01 ;
7  parameter ACCESS = 2'b10 ;
8  parameter ADDR_CTRL_REG = 32'h00000000 ;
9  parameter ADDR_STATS_REG = 32'h00000001 ;
10 parameter ADDR_TX_REG = 32'h00000002 ;
11 parameter ADDR_RX_REG = 32'h00000003 ;
12
13 //MASTER -----> APB SLAVE
14 input PCLK,PRESETn,PSEL,PENABLE,PWRITE ;
15 input [31:0] PADDR,PWDATA ;
16
17 //APB SLAVE -----> MASTER
18 output reg [31:0] PRDATA;
19 output reg PREADY ;
20
21 //UART -----> APB SLAVE
22 input tx_busy,tx_done,rx_busy,rx_done,rx_error ;
23 input [7:0] rx_data;
24
25 //APB SLAVE -----> UART
26 output rx_en,rx_rst,tx_en,tx_rst ;
27 output [7:0] tx_data ;
28
29 //REGISTERS VALUES
30 reg [31:0] CTRL_REG,STATS_REG,TX_DATA,RX_DATA ;
31
32 //CS & NS
33 reg [1:0] cs,ns ;
34
35 always @(posedge PCLK or negedge PRESETn) begin
36   if (~PRESETn)
37     |  cs <= IDLE ;
38   else
39     |  cs <= ns ;
40 end
41
```

```
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 register 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.
- **BCLK:** 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 BCLK 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 BCLK 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:

```
1  module UART_module (PRESETn,PCLK,rx_en,rx_rst,tx_en,tx_rst,tx_data,
2  |  |  |  tx_busy,tx_done,rx_busy,rx_done,rx_error,rx_data,BCLK,RX,TX);
3
4  parameter START_BIT = 0 ;
5  parameter END_BIT= 1 ;
6
7  //CLK and RESET
8  input PRESETn,PCLK ;
9
10 //UART -----> APB SLAVE
11 input rx_en,rx_rst,tx_en,tx_rst ;
12 input [7:0] tx_data ;
13
14 //APB SLAVE -----> UART
15 output reg tx_busy,tx_done,rx_busy,rx_done,rx_error ;
16 output reg [7:0] rx_data;
17
18 //BAUD CLK
19 input BCLK ;
20
21 //UART INPUT
22 input RX ;
23
24 //UART OUTPUT
25 output reg TX ;
26
27 reg [9:0] tx_reg ;
28 reg [3:0] tx_counter ;
29 reg [9:0] rx_reg ;
30 reg [3:0] rx_counter ;
31
```

```

32  /*Always Block that give the tx_data from APB SLAVE -----> the UART
33 ,and only updated when the tx_busy is LOW ,also add the START and END_BIT bits */
34 always @(posedge PCLK or negedge PRESETn) begin
35     if (~PRESETn)
36         tx_reg <= 0;
37     else if (tx_rst)
38         tx_reg <= 0;
39     else if (~tx_busy)
40         tx_reg <= {1'b0,tx_data,1'b1} ;
41 end
42
43
44 always @(posedge PCLK or negedge PRESETn) begin
45     if(~PRESETn) begin
46         TX <= 1 ;
47         tx_counter <= 9 ;
48         tx_busy <= 0 ;
49         tx_done <= 0 ;
50     end
51     else if(tx_rst) begin
52         TX <= 1 ;
53         tx_counter <= 9 ;
54         tx_busy <= 0 ;
55         tx_done <= 0 ;
56     end
57     else begin
58         if (tx_en & BCLK) begin
59             TX <= tx_reg[tx_counter] ;
60             tx_counter <= tx_counter - 1 ;
61
62             if (tx_counter == 0) begin
63                 tx_busy <= 0 ;
64                 tx_done <= 1 ;
65                 tx_counter <= 9 ;
66             end
67             else begin
68                 tx_busy <= 1 ;
69                 tx_done <= 0 ;
70             end
71         end
72     end
73 end
74

```

```

75  /*Always Block that give the rx_data from UART -----> APB SLAVE
76  ,and only updated when the rx_done is LOW ,also remove the START and END_BIT bits */
77  always @(posedge PCLK or negedge PRESETn) begin
78      if(~PRESETn) begin
79          rx_counter <= 9;
80          rx_busy <= 0 ;
81          rx_done <= 0 ;
82          rx_reg <= 0 ;
83      end
84      else if(rx_rst) begin
85          rx_counter <= 9;
86          rx_busy <= 0 ;
87          rx_done <= 0 ;
88          rx_reg <= 0 ;
89      end
90      else begin
91          if (rx_en & BCLK & !rx_done) begin
92              rx_counter <= rx_counter - 1 ;
93              rx_reg[rx_counter] <= RX ;
94
95              if (rx_counter == 0) begin
96                  rx_busy <= 0 ;
97                  rx_done <= 1 ;
98                  rx_data <= rx_reg [8:1] ;
99                  rx_counter <= 9 ;
100             end
101         else
102             rx_busy <= 1 ;
103             // rx_done <= 0 ;
104
105         end
106     end
107 end
108
109 //PRESET that reset all the outputs and the rx_error signal
110 always @(posedge PCLK or negedge PRESETn) begin
111     if (~PRESETn) begin
112         rx_error <= 0 ;
113     end
114     else if (rx_done && (rx_reg[0] != 1'b1 || rx_reg[9] != 1'b0))
115         rx_error <= 1 ;
116     else
117         rx_error <= 0 ;
118 end
119
120 endmodule

```

# Baud Generator Module

## 1. Purpose

The Baud module generates the baud clock (BCLK) 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

- **BCLK:** 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:

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

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

## 7. Function in UART System

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

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

## 8. BAUD Rate

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

## TOP Module

```

1 module UART_APB_Top (PCLK,PRESETn,PSEL,PENABLE,PWRITE,PADDR,
2 | | | | | PWDATA,PRDATA,PREADY,RX,TX);
3
4 input PCLK ; // System clock
5 input PRESETn ; // Active-low reset
6
7 // APB interface
8 input PSEL,PENABLE,PWRITE ;
9 input [31:0] PADDR ;
10 input [31:0] PWDATA ;
11 output [31:0] PRDATA ;
12 output PREADY ;
13
14 // UART pins
15 input RX ; // UART receive pin
16 output TX ; // UART transmit pin
17
18 // Internal wires
19 wire rx_en, rx_rst, tx_en, tx_rst;
20 wire [7:0] tx_data;
21 wire [7:0] rx_data;
22 wire tx_busy, tx_done, rx_busy, rx_done, rx_error;
23
24 // Baud clock
25 wire BCLK;
26
27 // Instantiate Baud generator
28 Baud baud_gen (
29     .clk(PCLK),
30     .rst(PRESETn),
31     .BCLK(BCLK)
32 );

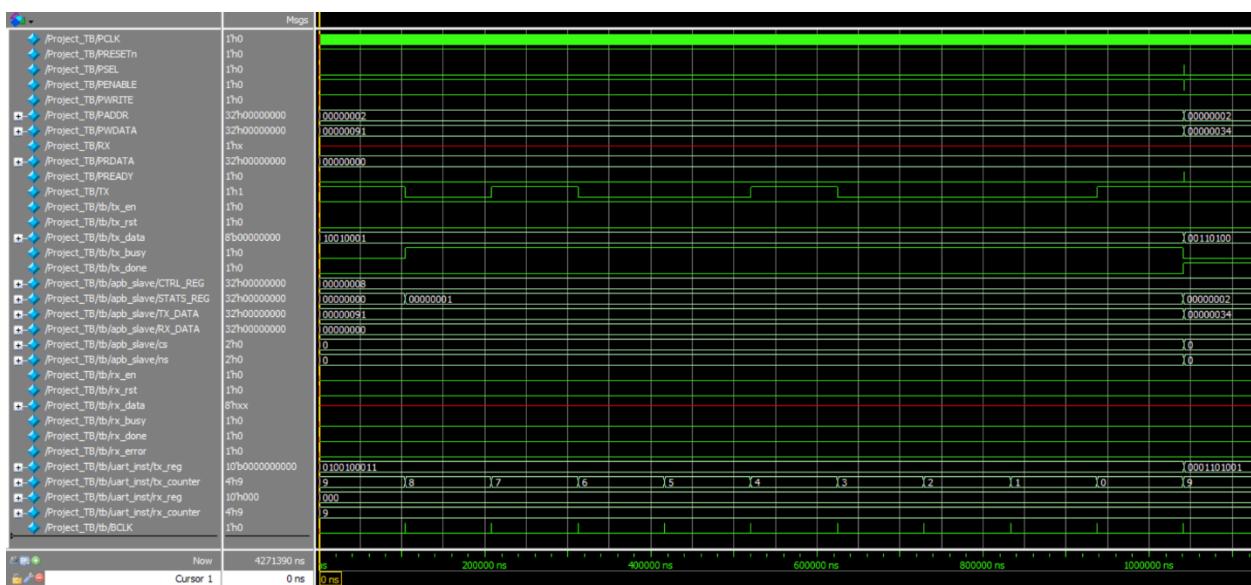
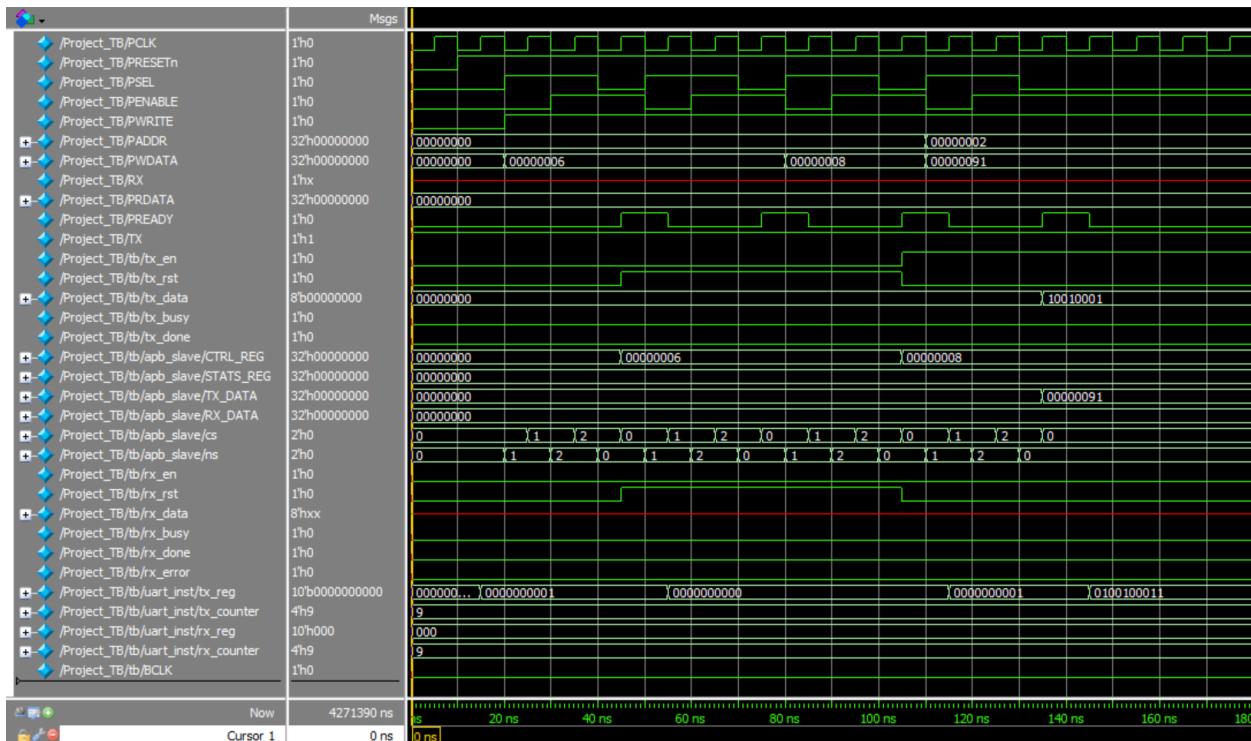
```

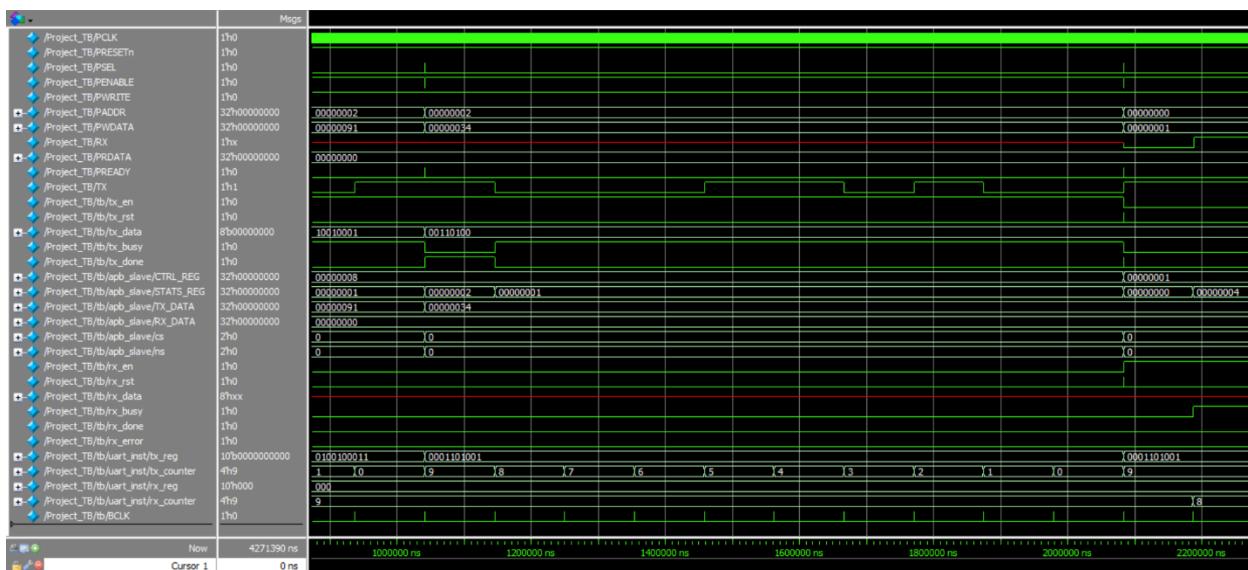
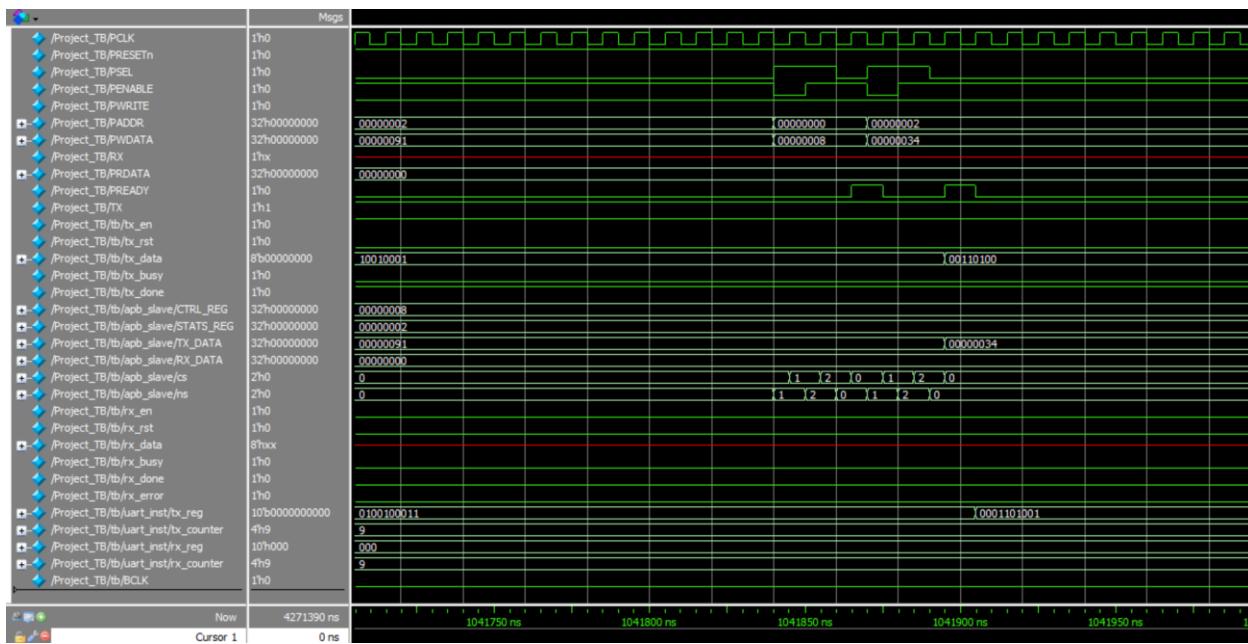
```

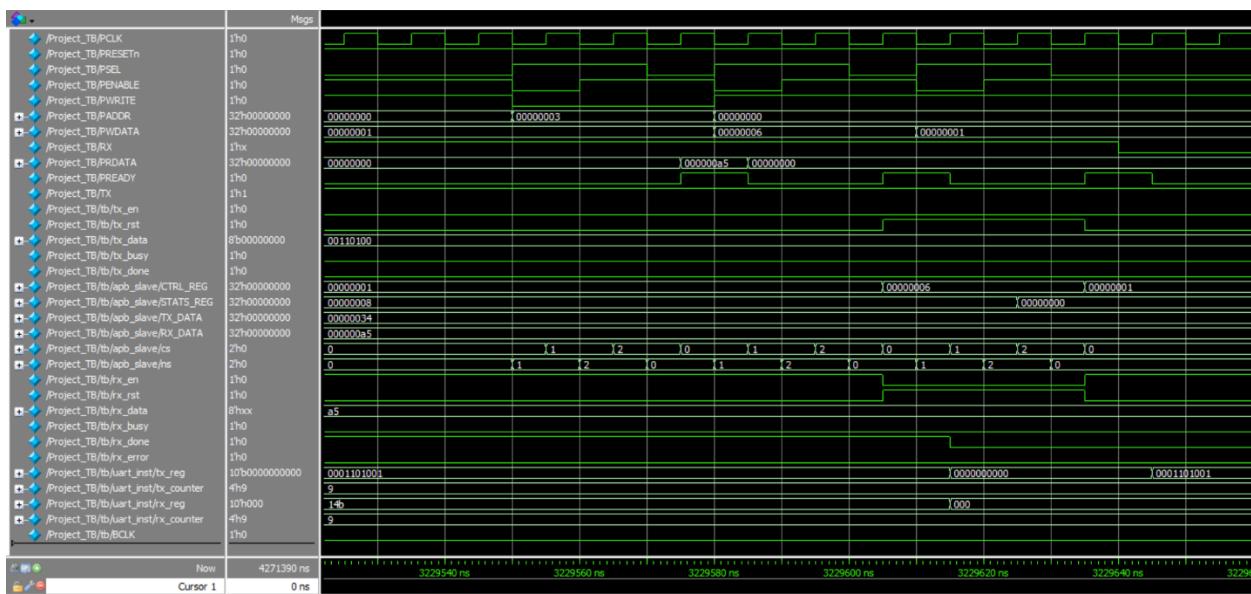
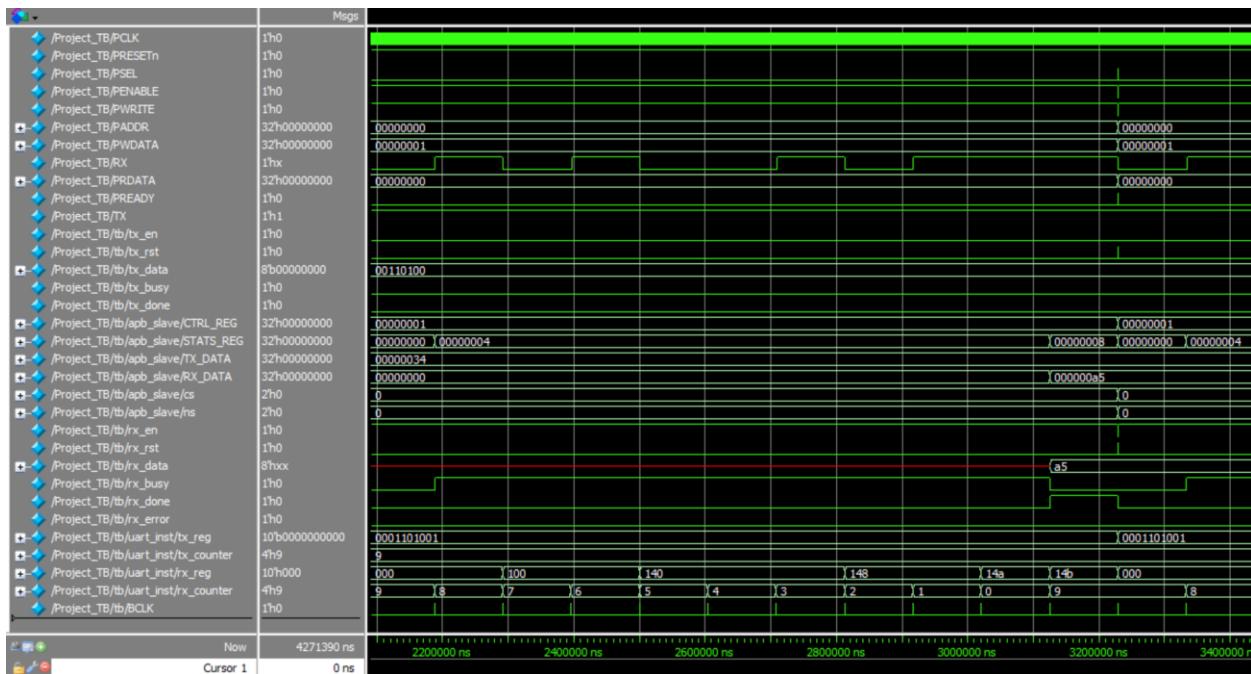
34     // Instantiate UART
35     UART_module uart_inst (
36         .PRESETn(PRESETn),
37         .PCLK(PCLK),
38         .rx_en(rx_en),
39         .rx_rst(rx_rst),
40         .tx_en(tx_en),
41         .tx_rst(tx_rst),
42         .tx_data(tx_data),
43         .tx_busy(tx_busy),
44         .tx_done(tx_done),
45         .rx_busy(rx_busy),
46         .rx_done(rx_done),
47         .rx_error(rx_error),
48         .rx_data(rx_data),
49         .BCLK(BCLK),
50         .RX(RX),
51         .TX(TX)
52     );
53
54     // Instantiate APB Slave
55     APB_Slave apb_slave (
56         .PCLK(PCLK),
57         .PRESETn(PRESETn),
58         .PSEL(PSEL),
59         .PENABLE(PENABLE),
60         .PWRITE(PWRITE),
61         .PADDR(PADDR),
62         .PWDATA(PWDATA),
63         .PRDATA(PRDATA),
64         .PREADY(PREADY),
65
66         // UART -> APB
67         .tx_busy(tx_busy),
68         .tx_done(tx_done),
69         .rx_busy(rx_busy),
70         .rx_done(rx_done),
71         .rx_error(rx_error),
72         .rx_data(rx_data),
73
74         // APB -> UART
75         .rx_en(rx_en),
76         .rx_rst(rx_rst),
77         .tx_en(tx_en),
78         .tx_rst(tx_rst),
79         .tx_data(tx_data)
80     );
81
82 endmodule

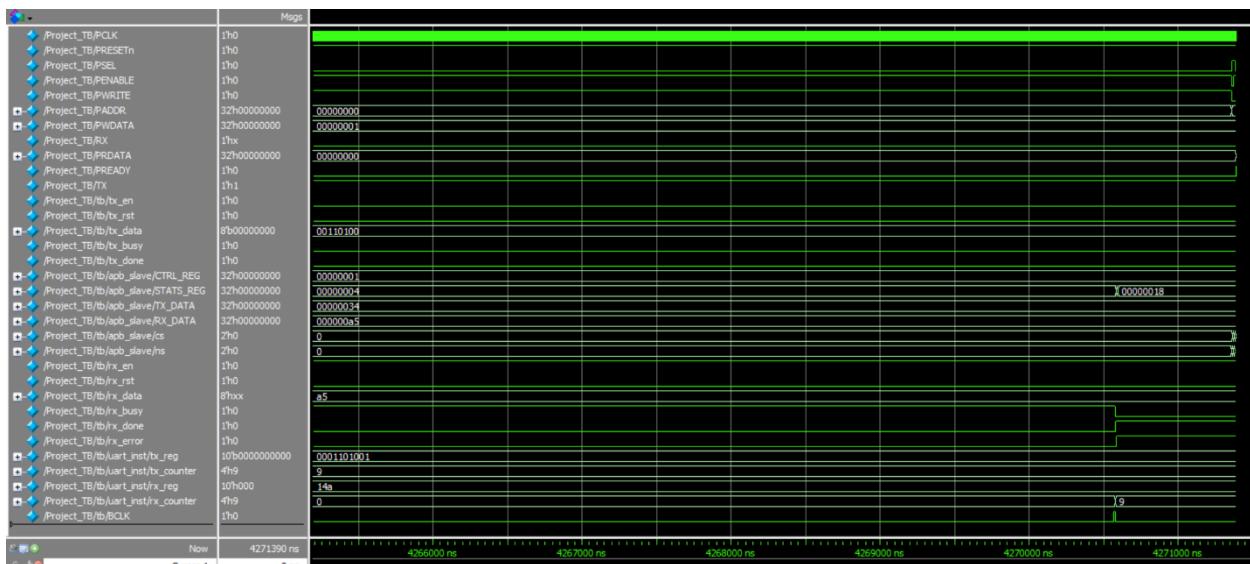
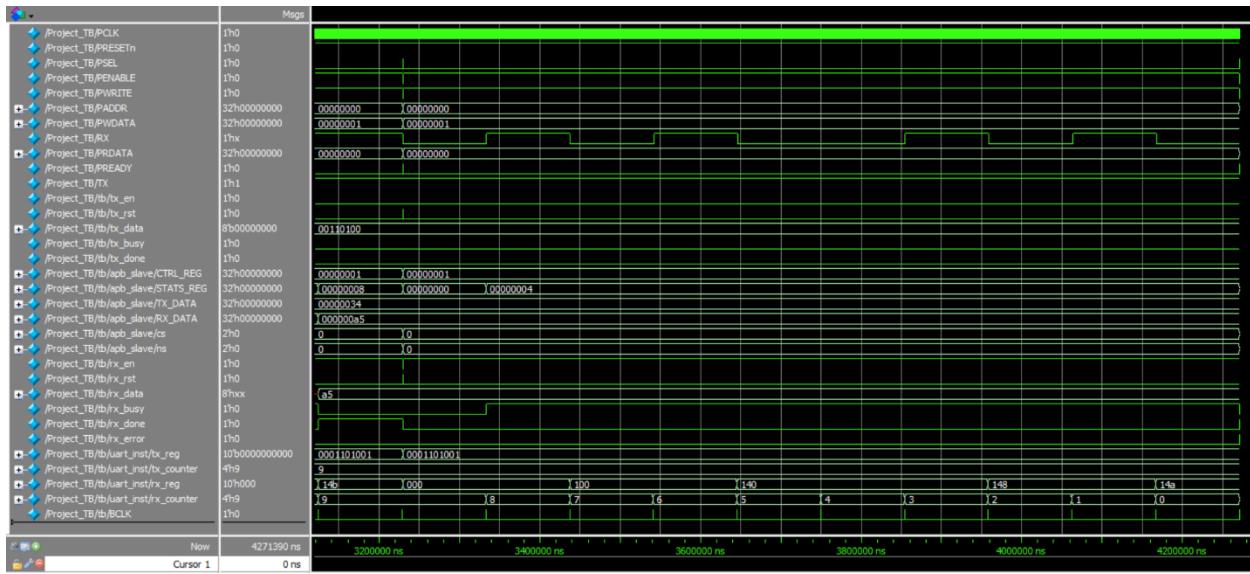
```

## Questa Sim Snippets (TB Code in GITHUB)









## QuestLint Snippets

The screenshot shows the QuestLogic IDE interface with the following components:

- Design View:** Displays the Verilog code for the `UART_APB_Top` module.
- Lint Summary:** A table showing the count of various lint issues found in the design.
- Lint Checks:** A detailed table of specific lint errors with columns for Severity, Status, Check, Alias, Message, Module, Category, State, Owner, and STARC Reference.

```

module UART_APB_Top (PCLK, PRESETn, PSEL, PENABLE, PWRITE, PA1,
                     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 tx_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 BCLK;
                     // Instantiate Baud generator
                     Baud baud_gen (
                         .clk(PCLK),
                         .rst(PRESETn),
                         .BCLK(BCLK)
                     );
                     // Instantiate UART
                     UART_module uart_inst (
                         .PRESETn(PRESETn),
                         .PCLK(PCLK),
                         .rx_en(rx_en),
                         .tx_en(tx_en)
                     );
    
```

Name	Count
Open(uninspected, ...)	5 (10)
Info	5 (10)

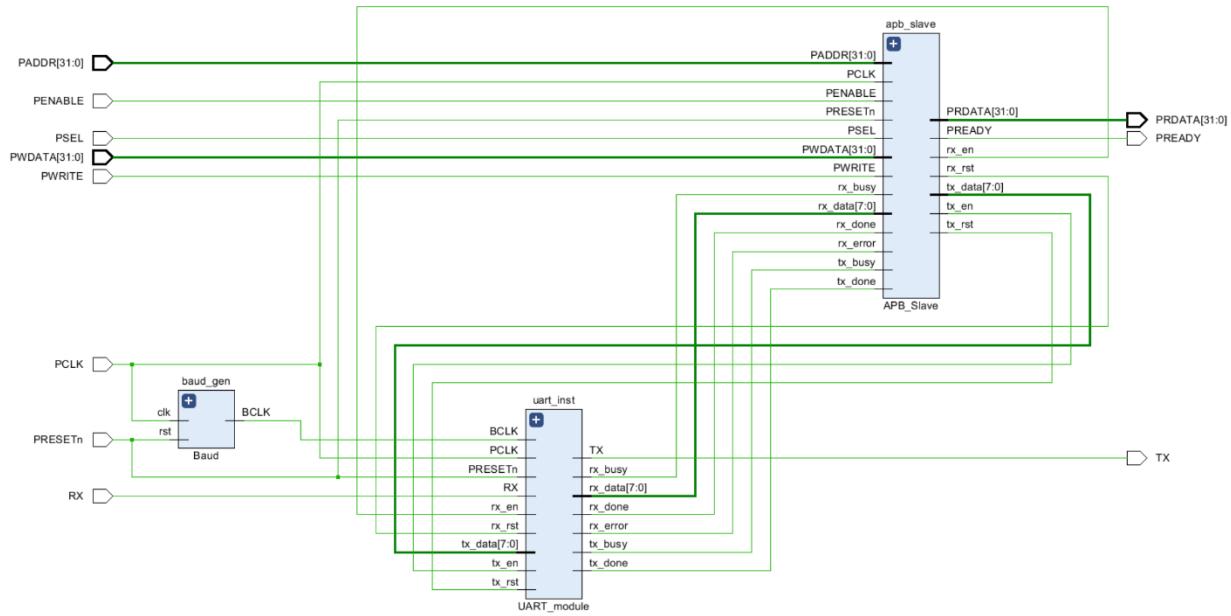
Severity	Status	Check	Alias	Message	Module	Category	State	Owner	STARC Reference
+	Open	assign_width_overflow		Width of assignment RHS is greater than width of LHS.	APB_Slave	Rtl Design ...	open	unass...	2.1.3.2, 2.10.3.3, 2.10.3.4, 2.10.4.3, 2.10.6.1, 2.10.6.2
?	Open	multi_ports_in_single_line		Multiple ports are declared in one line.	Mod... APB_Slave	Rtl Design ...	open	unass...	3.5.6.3
?	Open	multi_ports_in_single_line		Multiple ports are declared in one line.	Mod... Baud	Rtl Design ...	open	unass...	3.5.6.3
?	Open	multi_ports_in_single_line		Multiple ports are declared in one line.	Mod... UART_AP...	Rtl Design ...	open	unass...	3.5.6.3
?	Open	multi_ports_in_single_line		Multiple ports are declared in one line.	Mod... UART_mo...	Rtl Design ...	open	unass...	3.5.6.3

## DO File:

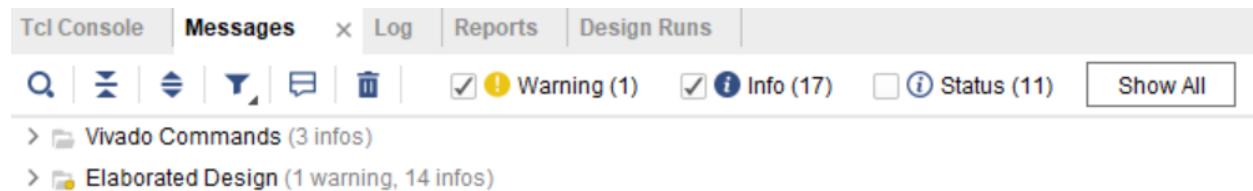
```
vlib work
vlog APB_Master.v
vlog Baud.v
vlog UART_module.v
vlog Top_Module.v
vlog Project_TB.v
vsim -voptargs=+acc work.Project_TB
add wave -position insertpoint \
sim:/Project_TB/PCLK \
sim:/Project_TB/PRESETn \
sim:/Project_TB/PSEL \
sim:/Project_TB/PENABLE \
sim:/Project_TB/PWRITE \
sim:/Project_TB/PADDR \
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/BCLK
run -all
#quit -sim
```

# VIVADO

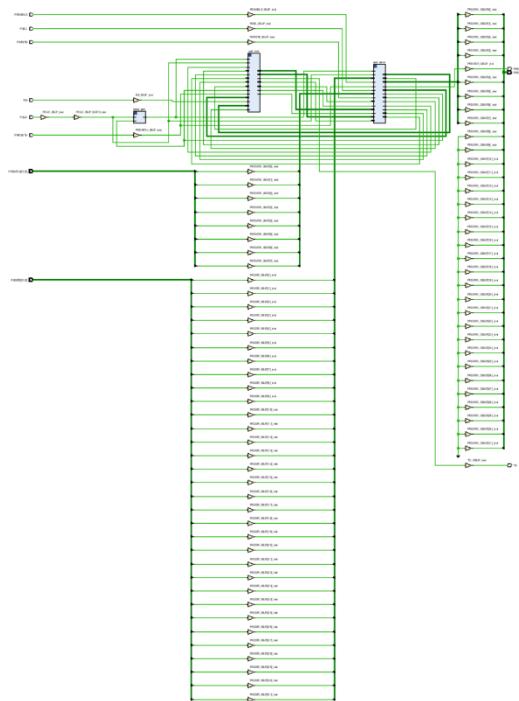
- Elaboration schematic:



- Elaboration Message:



- **Synthesis Schematic:**



- **Synthesis Timing report snippet:**

Tcl Console	Messages	Log	Reports	Design Runs	Timing	x		
Design Timing Summary								
General Information		Setup		Hold		Pulse Width		
Timer Settings		Worst Negative Slack (WNS): 6.211 ns		Worst Hold Slack (WHS): 0.139 ns		Worst Pulse Width Slack (WPWS): 4.500 ns		
Design Timing Summary		Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns		Total Pulse Width Negative Slack (TPWS): 0.000 ns		
Clock Summary (1)		Number of Failing Endpoints: 0		Number of Failing Endpoints: 0		Number of Failing Endpoints: 0		
> Check Timing (61)		Total Number of Endpoints: 141		Total Number of Endpoints: 141		Total Number of Endpoints: 99		
> Intra-Clock Paths		All user specified timing constraints are met.						
Inter-Clock Paths								
Other Path Groups								
Timing Summary - timing_1								

- **Synthesis Utilization report:**

Tcl Console	Messages	Log	Reports	Design Runs	Timing	Utilization	x
Hierarchy							
Hierarchy							
Name	1	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Bonded IOB (106)	BUFGCTRL (32)	
UART_APB_Top		85	100	1	80	1	
apb_slave (APB_Slave)		35	38	0	0	0	
baud_gen (Baud)		25	21	0	0	0	
uart_inst (UART_modu...)		25	41	1	0	0	
utilization_1							

- **Massage after Synthesis:**



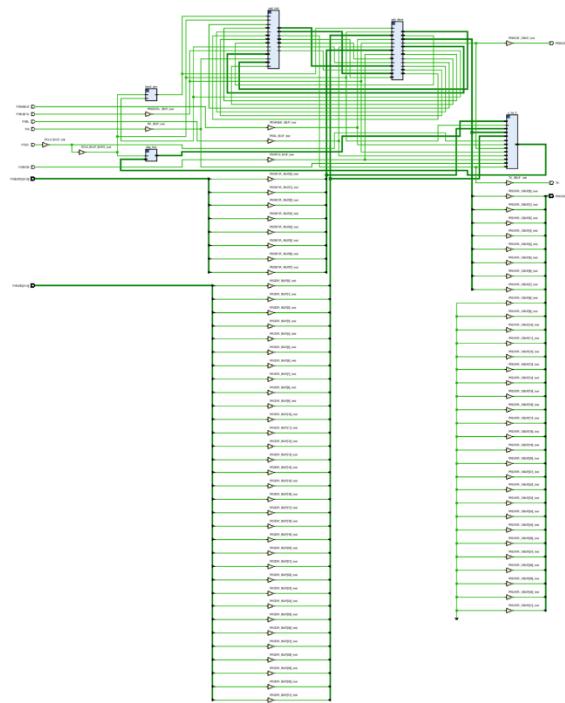
- **Implementation Timing report:**

Design Timing Summary																			
	Setup			Hold			Pulse Width												
	Worst Negative Slack (WNS):	0.2759 ns	Worst Hold Slack (WHS):	0.022 ns	Worst Pulse Width Slack (WPWS):	3.750 ns	Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWNS):	0.000 ns							
Clock Summary (2)		Number of Failing Endpoints: 0		Number of Failing Endpoints: 0		Number of Failing Endpoints: 0		Total Number of Endpoints: 4835		Total Number of Endpoints: 4819		Total Number of Endpoints: 2855							
> Check Timing (61)		Total Number of Endpoints: 4835		Total Number of Endpoints: 4819		Total Number of Endpoints: 2855													
> Intra-Clock Paths																			
> Inter-Clock Paths																			
All user specified timing constraints are met.																			
Timing Summary - impl_1 (saved)																			

- **Implementation Utilization report:**

Name	1	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)	BSCAN2 (4)
UART_APB_Top	1618	2533	7	799	1440	178	1036	2	80	2	1	
apb_slave (APP_Slave)	35	38	0	21	35	0	11	0	0	0	0	0
baud_gen (Baud)	25	21	0	14	25	0	21	0	0	0	0	0
> dbg_hub (dbg_hub)	476	727	0	234	452	24	315	0	0	1	1	
> u_ilia_0 (u_ilia_0)	1057	1706	6	534	903	154	664	2	0	0	0	0
uart_inst (UART_modu...)	25	41	1	11	25	0	18	0	0	0	0	0

- **Massage after Implementation:**



- **FPGA device snippet:**

