Indian Institute of Information Technology, Allahabad

Project B.Tech E.C.E 5th Semester

Secure UART Implementation on FPGA

Project Supervisor : Dr. Sunny Sharma

Group Members: IEC2014002 Ritwick Sundar
IEC2014090 Utkarsh Singh
IEC2014097 Kshitij Garg

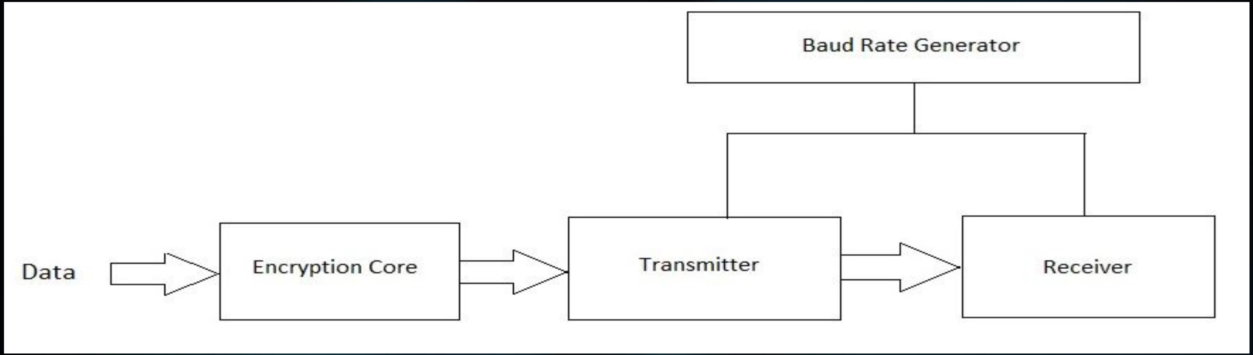
Abstract

- UART - Universal Asynchronous Transmitter Receiver is a computer hardware device for asynchronous communication in conjunction with communication standards like RS-232.
- UART works on a serial-to-parallel and parallel-to-serial data transfer protocol.
- Mostly used for short-distance, low-cost data exchange between computer and peripherals.

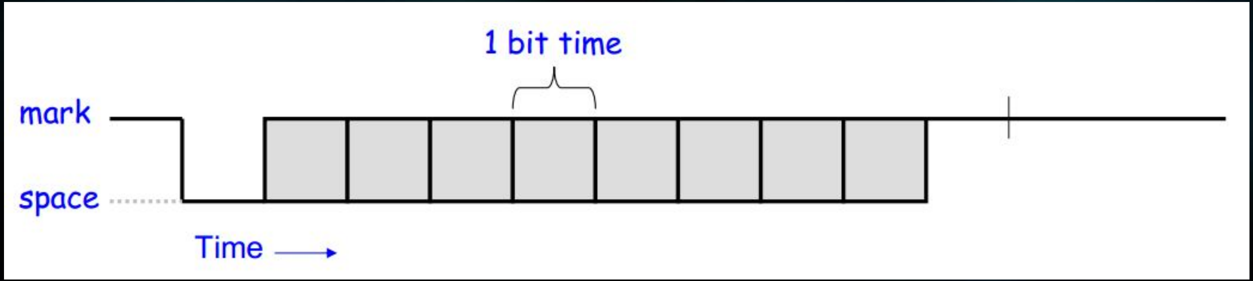
- Secure UART - Data transfer is secured using Secure Hashing Algorithm-1 (SHA-1) mainly for high speed integrity checks with end-to-end control in industrial uses.
- SHA-1 : It is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST.
- SHA-1 produces a 40 digit hex hash value known as a message digest.

Block Diagram

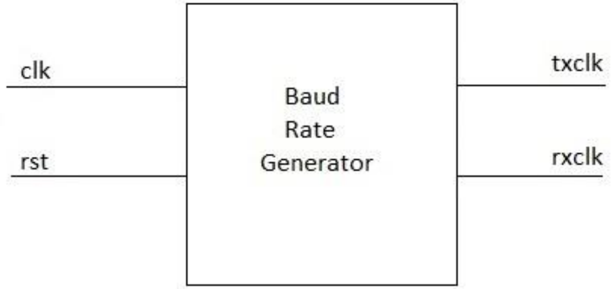
Full UART



Transmitted Bitstream



Functional Block Diagram of the Baud Rate Generator



Input Ports

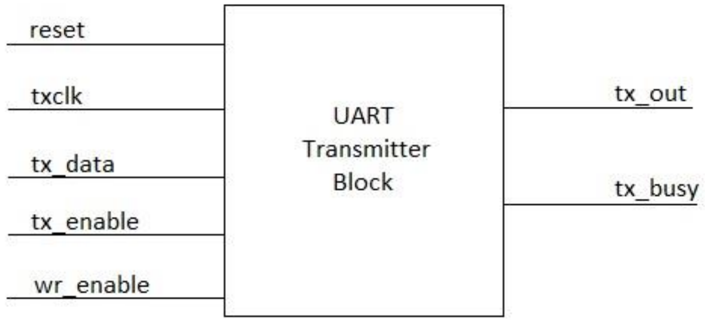
clk: Input system clock to the Baud Rate Generator
rst: Input reset that resets the Transmitter

Output Ports

txclk: Output of the Baud Rate Generator that acts as Input clock to the Transmitter. Transmission and Reception will occur when the system clock to both the Transmitter and Receiver are synchronised.

rxclk: Output of the Baud Rate Generator that acts as Input clock to the Receiver.

Functional Block Diagram of UART Transmitter



Input Ports:

reset:	Resets the clock
txclk:	Input clock to the Transmitter
tx_data:	8 bit data to be transmitter by the Transmitter
tx_enable:	Output of the Baud Rate Generator which tells whether transmitter should transmit data or not
wr_enable:	Write enable

Output Ports:

tx_out :	Output of the Transmitter that transmits the data stream to the Receiver.
tx_busy:	If the Transmitter is in the IDLE state, the tx_busy goes low and Transmitter is free to transmit to the Receiver. If not in IDLE state, tx_busy stays high.

Introduction

UART (Universal Asynchronous Receiver Transmitter) is an integrated circuit, which is used for transmitting and receiving data asynchronously via the serial port on the computer. It contains a parallel-to-serial converter for data transmitted from the computer and a serial-to-parallel converter for data coming in via the serial line. The UART also has a buffer for temporarily storing data from high-speed transmissions.

UART will usually provide additional circuits for signals that can be used to indicate the state of the transmission media and to regulate the flow of data in the event that the remote device is not prepared to accept more data.

The typical UART consists of three basic modules :

- Baud Rate Generator
- Transmitter
- Receiver

Secure UART

- The transmitted data from a UART is not secured by the UART module itself but by the software of the electronic device the UART is connected to.

- In this project we aim to integrate both the Encryption module and UART together for a full Hardware implementation and create a Secure UART.

- The messages can be encoded in different protocols like AES ,Tiny Encryption, etc based on specific requirements.

- For our project we aim to implement Secure Hashing Algorithm - 1 (SHA-1) with the UART module.

- Hence our transmitted messages will be a 40 digit hex code in the final implementation.

Advantages of Hardware Implementation

- The hardware based encryption uses a device’s on-board security to perform encryption and decryption.
- It is self-contained and does not require the help of any additional software or driver.
- Free from any possibility of contamination, malicious code infection, or vulnerability.
- Minimum user interaction and does not cause performance degradation.
- Hardware Implementation offers stronger resilience against brute force attacks.
- Faster than any software implementation.
- Reduces CPU overhead as the hardware itself is responsible for Encryption.

Progress So-Far

- Learnt Basics of Verilog HDL
- Completed Baud Rate Generator Module
- Completed Transmitter Module
- Simulated Transmitter to transmit an 8 bit code on Xilinx Vivado 2016.2
- Timing Analysis of the modules

Deliverables till End-Sem

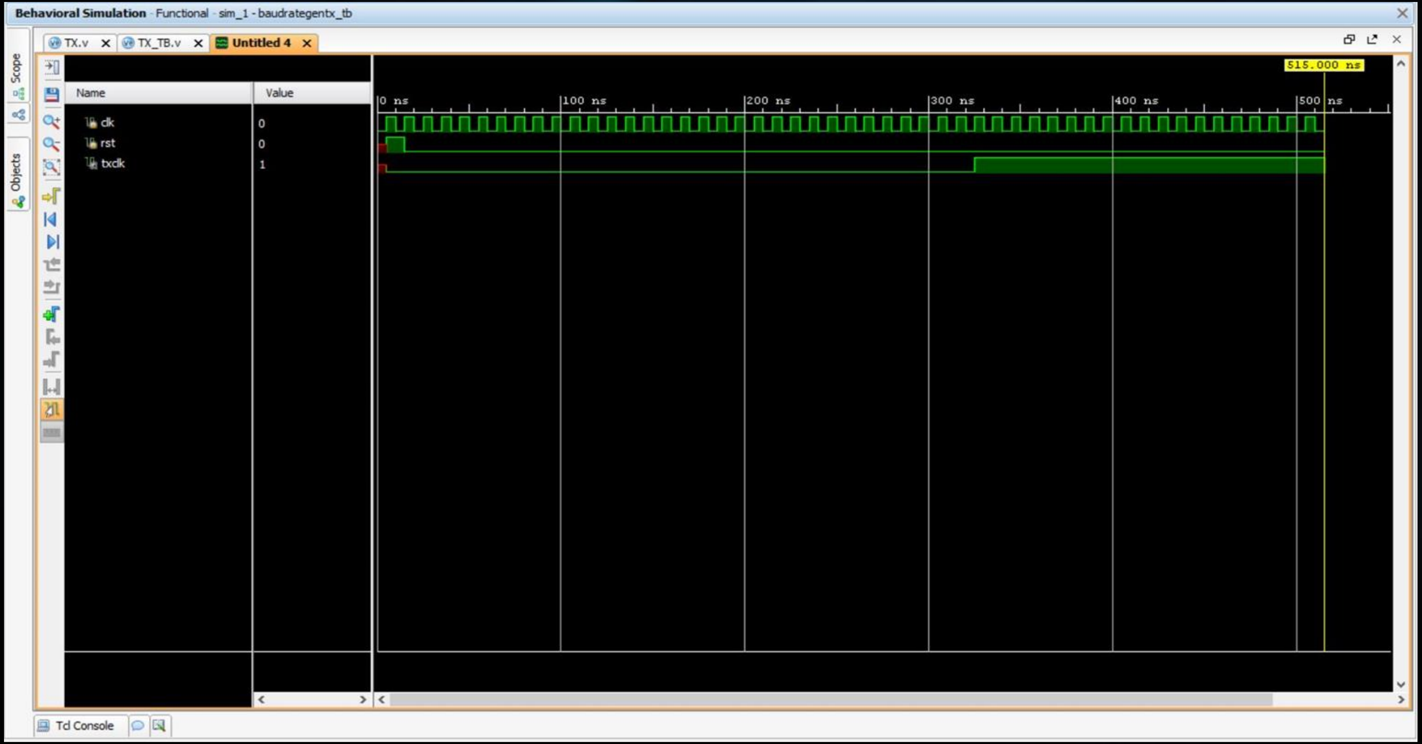
- Full UART Module (Baud Rate Generator, Transmitter, Receiver)
- SHA-1 Encryption Core
- Integrating SHA-1 Core with UART
- Full Timing and Performance Analysis
- FPGA Implementation

Baud Rate Generator

- The Baud Rate Generator contains 2 internal counters - one for Transmitter and one for Receiver clock.
- The Counters are used as Frequency Dividers.
- The Receiver clock frequency is normally 16 times the Transmitter clock frequency for oversampling.
- The Baud Rate Generator implemented uses Divide by 16 to create the 16:1 receiver to transmitter clock speed ratio.
- The Divide by 16 counter is a 4 bit counter. Once the counter reaches 15 , the transmitter clock goes high and counter is reset to 0.

Timing Diagram

- Timing Diagram of Transmitter clock (System Clock / 16) as generated by Baud Rate Generator.



Name	Value
clk	0
rst	0
txclk	1

Transmitter

The Transmitter takes in input from an 8 bit register and waits for wr_enable (CPU signal) to transmit.
By convention the default output is kept high to signify that line is active but idle.

Its function is to convert the parallel data to serial format and transmit it,

An even parity module has been created to send along the data for integrity check.

Inputs to the Transmitter include : Reset, Tx_Enable(Clock) ,Wr_enable, Tx_data and parity bit

Ouputs from the Transmitter include : Tx_Out , Tx_Busy flag.

Transmitter

- The Transmitter has been implemented as a Finite State Machine with 5 States :

1. IDLE - In Idle state the transmitter waits for the wr_enable (The CPU signal that tells UART it is sending data to be transmitted).

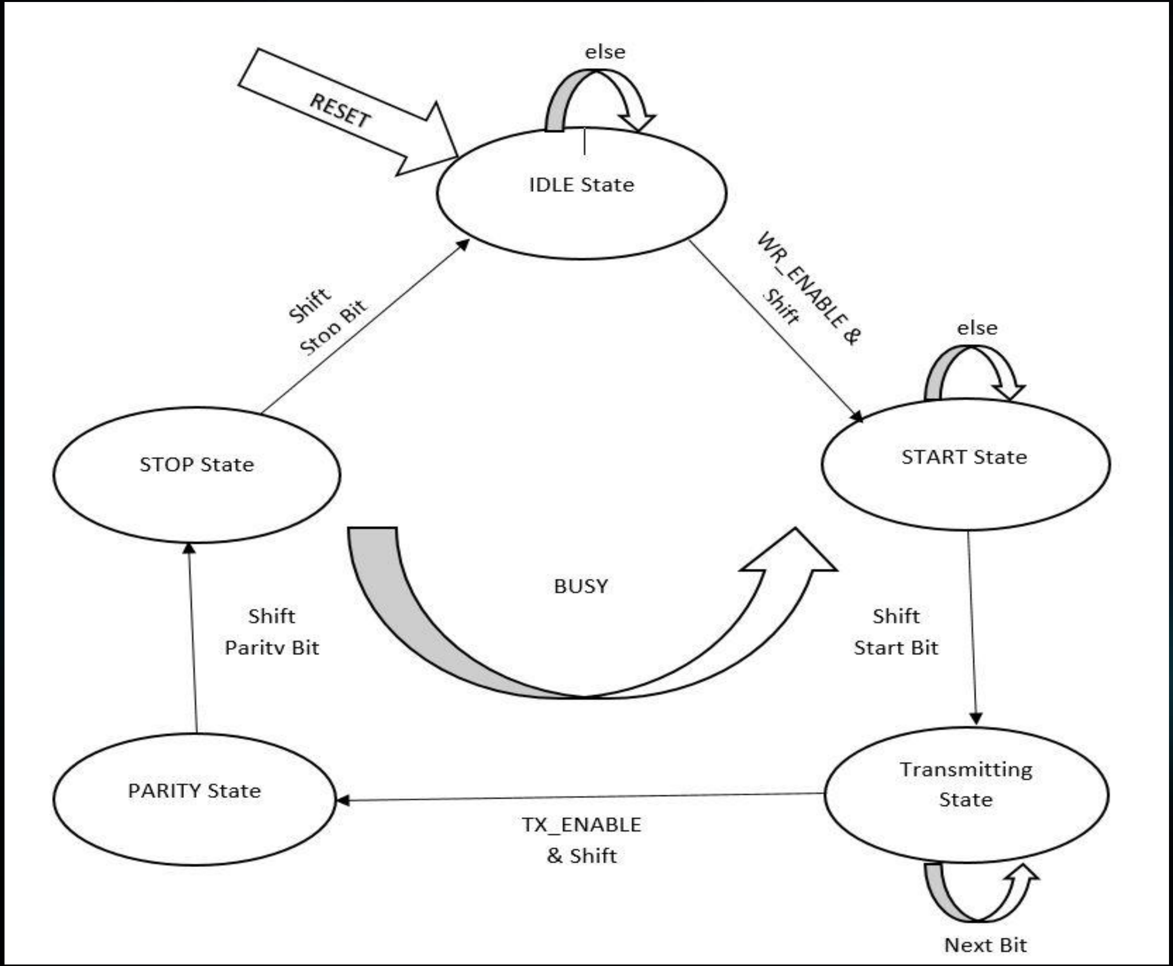
2. START- In Start state the transmitter makes the tx_out low to signal the start of data transmission to the receiver and sends out a 0 bit.

3. TX - The 8 bit data is now sent LSB first so that the data received by the receiver is in big-endian format (MSB-left LSB-right).

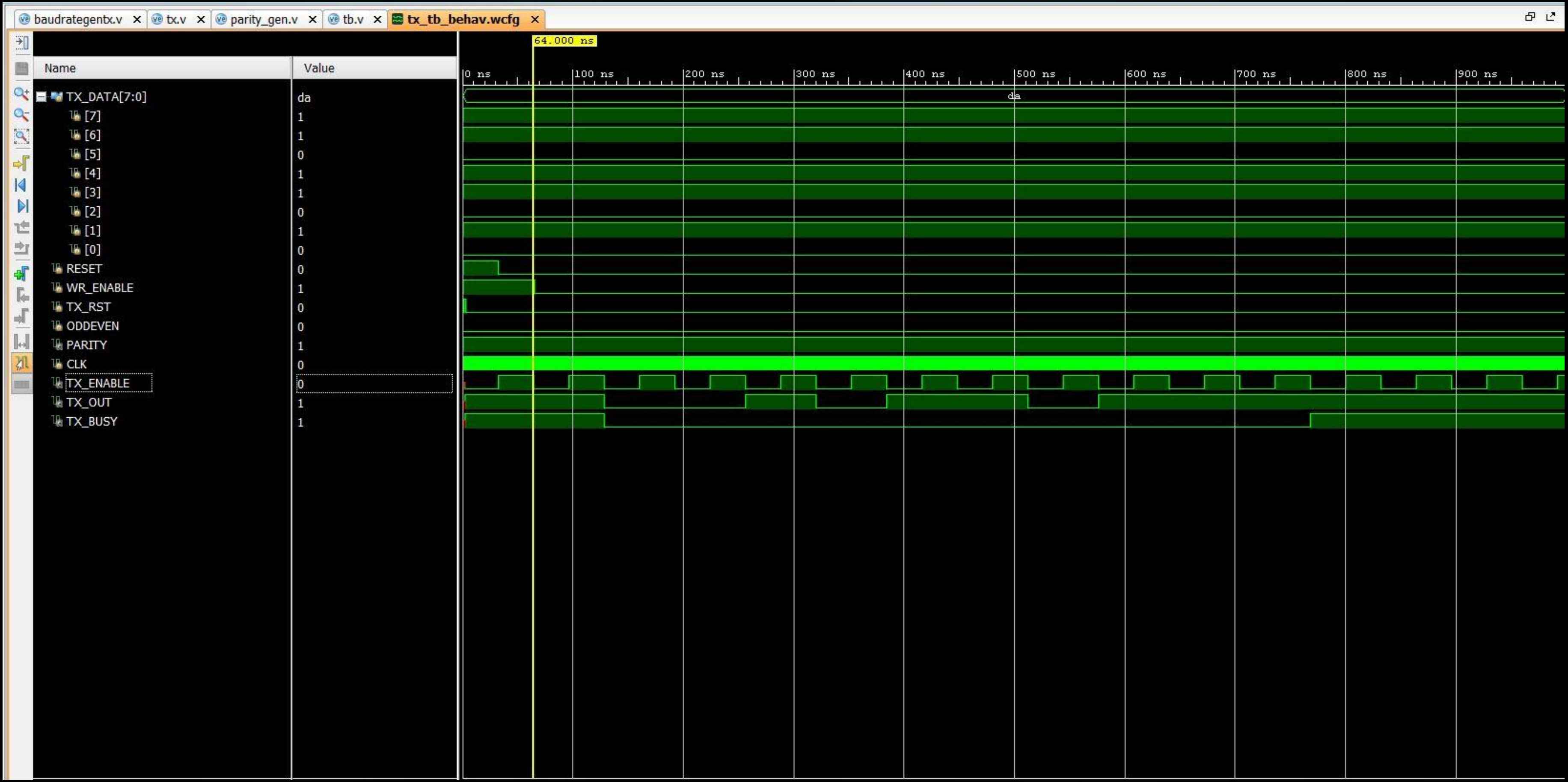
4. PARITY –The transmitter calculates the even parity of the transmitted code and appends it after the data transmission ; hence taking 1 bit for parity.

5. STOP - The ld_tx_data is made 0 and transmitter sends out 1 signalling completion of data transfer. The transmitter returns to IDLE state after STOP and waits for wr_enable.

Transmitter FSM

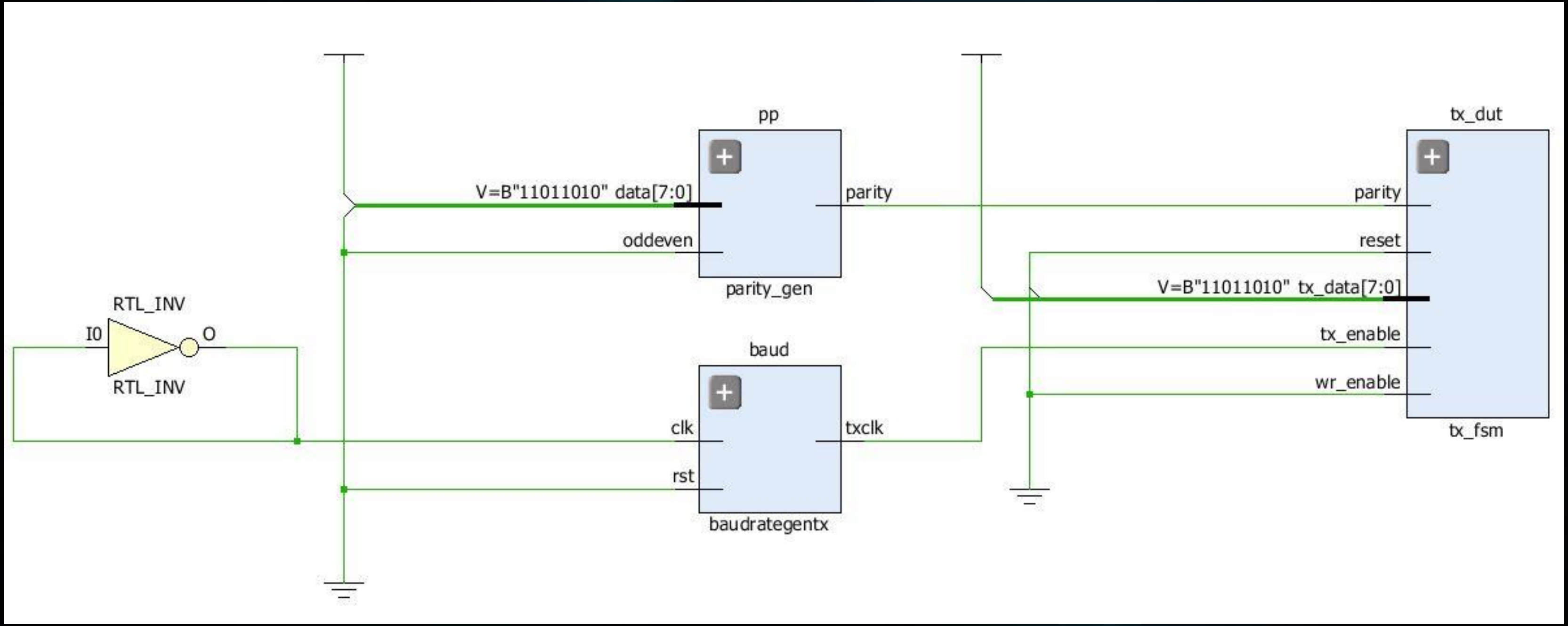


Overall Timing Diagram



Name	Value
TX_DATA[7:0]	da
[7]	1
[6]	1
[5]	0
[4]	1
[3]	1
[2]	0
[1]	1
[0]	0
RESET	0
WR_ENABLE	1
TX_RST	0
ODDEVEN	0
PARITY	1
CLK	0
TX_ENABLE	0
TX_OUT	1
TX_BUSY	1

RTL Synthesis



Code : // Baud Rate Generator

```
#(parameter width = 4, N = 8)

module baudrategentx(clk, rst, txclk);

  input clk, rst;
  output txclk;

reg [width - 1:0] r_reg, txclk_track;
wire [width - 1:0] r_nxt;

always @ (negedge clk or negedge rst) begin
  if (rst) begin
    r_reg <= 0;
    txclk_track <= 1'b0;
  end
  else if (r_nxt == N) begin
    r_reg <= 0;
    txclk_track <= ~txclk_track;
  end
  else
    r_reg <= r_nxt;
end
assign r_nxt = r_reg + 1;
assign txclk = txclk_track;
endmodule
```

Code : // Parity Module

```
`timescale 1ns / 1ps

module parity_gen (
data        ,
oddeven     ,
parity
);

    input [7:0] data;
    input oddeven;
    output parity;

    assign parity = (^data) ^ oddeven;

endmodule
```

Code : // Transmitter

```
module tx_fsm (
reset          ,
tx_data        ,
wr_enable      ,
tx_enable      ,
parity         ,
tx_out         ,
tx_busy
);
// Port declarations
input        reset          ;
input  [7:0] tx_data        ;
input        tx_enable      ;
input        wr_enable      ;
input        parity         ;
output       tx_out         ;
output       tx_busy        ;

// Internal Variables
reg [7:0]    tx_reg         ;
reg [3:0]    tx_cnt         ;
reg          tx_out         ;
reg                              tx_busy            ;
reg          ld_tx_data     ;
reg [SIZE-1:0]    state;

parameter SIZE = 5;
parameter IDLE = 3'b000, START = 3'b001, TX = 3'b010,  PARITY = 3'b011, STOP = 3'b100;
```

```
always @ (negedge tx_enable)
   if (reset)  begin
      tx_busy          <= 1;
      tx_reg       <= 0;
      tx_cnt       <= 0;
      ld_tx_data    <= 0;
      tx_out             <= 1;
      state <=  IDLE;

            end
   else

      case(state)
         IDLE : if (wr_enable) begin
                  tx_reg <= tx_data;
                  ld_tx_data <= 1;
                  state <= START;
                  tx_cnt <= 0;
                        end
         START : if (!tx_enable) begin
                   tx_busy <= 0;
                   $display("TX of data has been started");
                    if(tx_cnt == 0) begin
                        $display("Counter Value : %b",tx_cnt);
                        tx_out <= 0;
                                    end
                  state <= TX;
                  tx_cnt <= tx_cnt + 1;
                         end
         TX : if (!tx_enable) begin
               tx_out <= tx_reg[tx_cnt -1];
               $display("Counter Value: %b",tx_cnt);
               tx_cnt <= tx_cnt +1 ;
               if(tx_cnt == 8) begin
                  state <= PARITY;
                           end
                        end

PARITY: if(!tx_enable) begin
                                  tx_out <= parity;
                                  state <= STOP;
                                  $display("Parity Bit: %b",parity);
                                              end
        STOP :  if(!tx_enable) begin
                                                        ld_tx_data <=0;
              tx_out <= 1;
              tx_cnt <= 0;
              state <= #1 IDLE;
                                                        tx_busy        <= 1;
                                   $display("Counter Value: %b",tx_cnt);
                                   $display("TX of data has been completed");

                        end

        default : state <= #1 IDLE;
     endcase
endmodule
```

References

1. Basu, Debjani, Dipak K. Kole, and Hafizur Rahaman. "implementation of AES algorithm in UART module for secured data transfer." In Advances in Computing and Communications (ICACC), 2012 International Conference on, pp. 142-145. IEEE, 2012.

2. Fang, Yi-yuan, and Xue-jun Chen. "Design and simulation of UART serial communication module based on VHDL." In Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on, pp. 1-4. IEEE, 2011

3. Solanki, Yogendra Singh. "Performance Based Design and Implementation of a SHA-1 Hash Module on FPGA."

4. Verilog HDL : A Guide To Digital Design And Synthesis (IEEE 1364-2001 Compliant) – by Samir Palnitkar