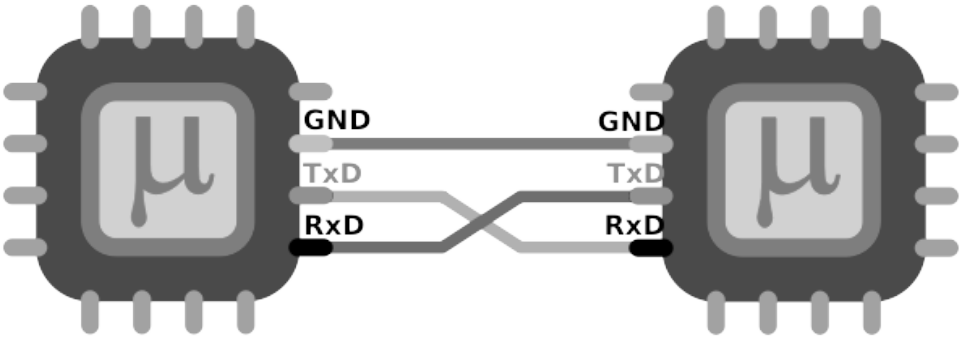
UART Tx Implementation

Chipions26 - Phase 1 Project



Team Members - G5

Name
AbdulRahman Mansour
Mohamed Atallah
Sahar Sallam
Muhammed Qasem

1 Introduction

Universal Asynchronous Receiver/Transmitter (UART) is a hardware communication protocol that facilitates asynchronous serial communication between devices. It is widely used in embedded systems, microcontrollers, and serial communication interfaces.

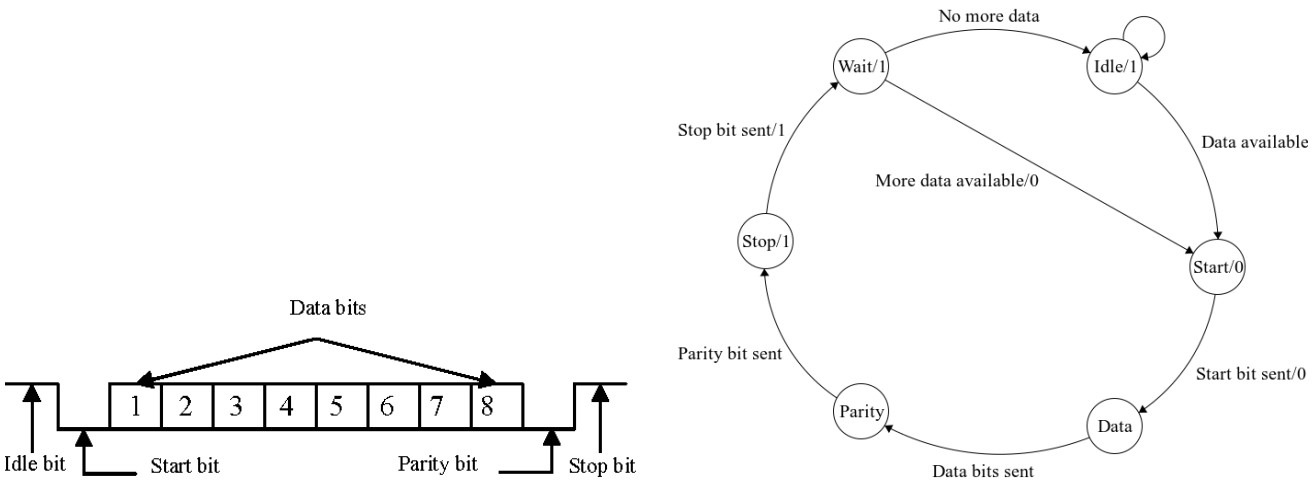
1.1 Key Features

1. **Asynchronous Communication:** UART does not require a shared clock signal between the transmitting and receiving devices, which simplifies the wiring.
2. **Full-Duplex Communication:** UART supports simultaneous two-way communication via separate transmit (TX) and receive (RX) lines.
3. **Simple Interface:** Typically requires only two data lines (TX and RX), in addition to ground (GND)

2 Operation

In the UART Protocol data is transmitted in frames consisting of the following states:

1. **Idle State:** The TX line is high when no data is being transmitted.
2. **Start Bit:** The TX line goes low to indicate the start of a transmission.
3. **Data Bits:** The transmitter sends the data bits sequentially.
4. **Parity Bit (if used):** The transmitter sends the parity bit.
5. **Stop Bits:** The TX line goes high for the duration of the stop bits.
6. **Return to Idle:** The TX line remains high until the next frame starts.



(a) UART Frame

(b) UART Tx FSM

Figure 1: UART Tx Operation

- **Note:** The LSB(Least Significant Bit) of the data is always transmitted first

3 Architecture

The UART transmitter circuit is suppose to serialize parallel data and transmit it over a single communication line. The architecture consists of several key components that work together to achieve this functionality. These components are: (Parity Generator, Frame Generator, Baud Rate Generator, Parallel-In-Serial-Out Shift Register, UART Transmitter[Top Module])

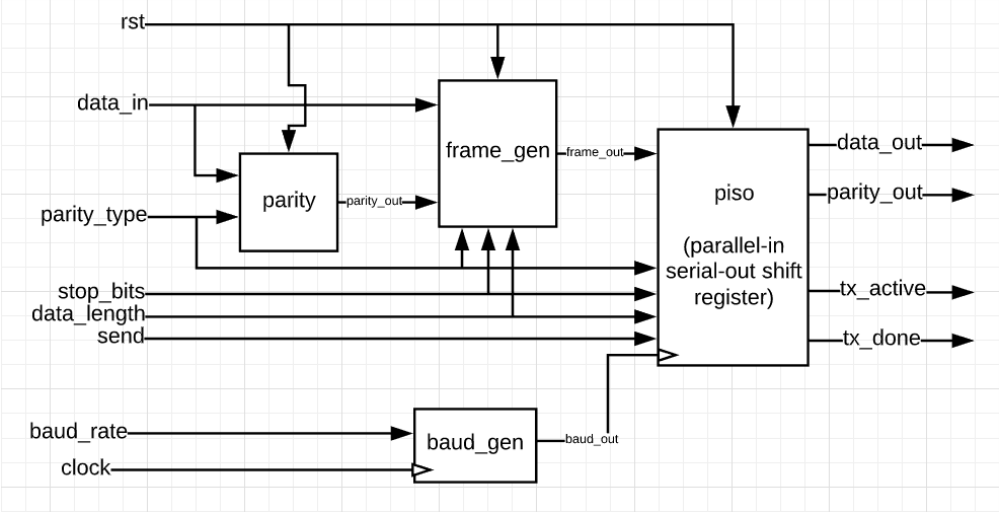


Figure 2: UART Tx Architecture

1. Parity Generator (`"parity"`):
   - Generates the parity bit based on the selected parity type (none, odd, even, or parallel odd).
   - Takes the input data and parity type as inputs.
   - Outputs the calculated parity bit.
2. Frame Generator (`"frame_gen"`):
   - Constructs the data frame to be transmitted, including start bit, data bits, parity bit, and stop bits.
   - Inputs include the data to be transmitted, parity bit, parity type, stop bits, and data length.
   - Outputs the constructed frame.
3. Baud Rate Generator (`"baud_gen"`):
   - Generates the baud rate clock signal for controlling the transmission speed.
   - Takes the system clock and selected baud rate as inputs.
   - Outputs the baud rate clock signal.
4. Parallel-In Serial-Out Shift Register (`piso`):
   - Serializes the parallel data frame and outputs it bit by bit based on the baud rate clock.
   - Inputs include the data frame, parity type, stop bits, data length, send signal, baud rate clock, and reset signal.
   - Outputs the serialized data, parallel parity output, transmission status, and completion signal.
5. UART Transmitter (`"uart_tx"`):
   - The top-level module that integrates all the submodules and controls the data transmission process.
   - Inputs include data, control signals (such as parity type, baud rate, etc.), and a clock signal.
   - Outputs the serialized data (`data_out`), parity output (`p_parity_out`), transmission status (`tx_active`), and completion signal (`tx_done`).

4 Implementation

This section includes a Verilog implementation of a UART Transmitter along with testbenches and the output waveforms.

4.1 Parity Generator

4.1.1 Verilog Code

```
module parity (
    input rst,
    input [7:0] data_in,
    input [1:0] parity_type,
    output reg parity_out
);

    wire parity;
    assign parity = ^data_in; // 1 if parity is odd, 0 if parity is even

    always @(negedge rst)
    begin
        if (~rst) begin
        parity_out = 1'b0;
        end
    end

    always@(*)
    begin
     begin
            case (parity_type)
                2'b00: // No parity
                            parity_out = 1'b0; // 0 NO ERROR , 1 ERROR
                2'b01: // Odd parity
                    begin
                        if (parity)
                            parity_out = 1'b0;
                        else
                            parity_out = 1'b1;
                    end
                2'b10: // Even parity
                    begin
                        if (parity)
                            parity_out = 1'b1;
                        else
                            parity_out = 1'b0;
                    end

                2'b11:    // Odd parity (parallel)
                    begin
                        if (parity)
                            parity_out = 1'b0;
                        else
                            parity_out = 1'b1;
                    end
                endcase
        end

    end

endmodule

```

4.1.2 Testbench

```
`timescale 1ns/1ns
module parity_tb;

reg       rst_tb;
reg [7:0] data_in_tb;
reg [1:0] parity_type;

wire      parity_out_tb;

wire parity_tb;
assign paraty_tb = ^ data_in_tb ;

parity dut(
    .data_in(data_in_tb),
    .rst(rst_tb),
    .parity_type(parity_type),
    .parity_out(parity_out_tb)
);

initial
 begin
    rst_tb= 1'b0;
    #10 rst_tb = 1'b1;

    $display(" TEST 1 00 test " );
    #10 parity_type = 2'b00;
    data_in_tb = 8'b00010111;
        if (parity_out_tb == 1'b0 ) $display("test 1 succeded");
        else $display("test 1 failed paraty out = %b ",parity_out_tb);

    $display(" TEST 2 01 test " );
    #10 parity_type = 2'b01;
    data_in_tb = 8'b00001111;
        if (parity_out_tb == 1'b0 && paraty_tb==1'b1 ) $display("test 2 succeded");
        else $display("test 2 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;

    $display(" TEST 3 01 test " );
    #10 parity_type = 2'b01;
    data_in_tb = 8'b00001101;

        if (parity_out_tb == 1'b1 && paraty_tb==1'b0 ) $display("test 3 succeded");
        else $display("test 3 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;

    $display(" TEST 4 10 test " );
    #10 parity_type = 2'b10;
    data_in_tb = 8'b00001111;
        if (parity_out_tb == 1'b1 && paraty_tb==1'b1 ) $display("test 4 succeded");
        else $display("test 4 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;

    $display(" TEST 5 10 test " );
    #10 parity_type = 2'b10;
    data_in_tb = 8'b00001101;
        if (parity_out_tb == 1'b0 && paraty_tb==1'b0 ) $display("test 5 succeded");
        else $display("test 5 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;

        $display(" TEST 6 11 test " );
    #10 parity_type = 2'b11;
    data_in_tb = 8'b00001111;
        if (parity_out_tb == 1'b0 && paraty_tb==1'b1 ) $display("test 6 succeded");
        else $display("test 6 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;

    $display(" TEST 7 11 test " );
    #10 parity_type = 2'b01;
    data_in_tb = 8'b00001101;
        if (parity_out_tb == 1'b1 && paraty_tb==1'b0 ) $display("test 7 succeded");
        else $display("test 7 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;$stop

 end
endmodule
```

4.1.3 Outputs

```
VSIM 38> run -all
#  TEST 1 00 test
# test 1 succeded
#  TEST 2 01 test
# test 2 failed paraty out = 0 and paraty = 0
#  TEST 3 01 test
# test 3 succeded
#  TEST 4 10 test
# test 4 failed paraty out = 0 and paraty = 1
#  TEST 5 10 test
# test 5 succeded
#  TEST 6 11 test
# test 6 failed paraty out = 1 and paraty = 1
#  TEST 7 11 test
# test 7 succeded
```

Figure 3: Parity Test Output

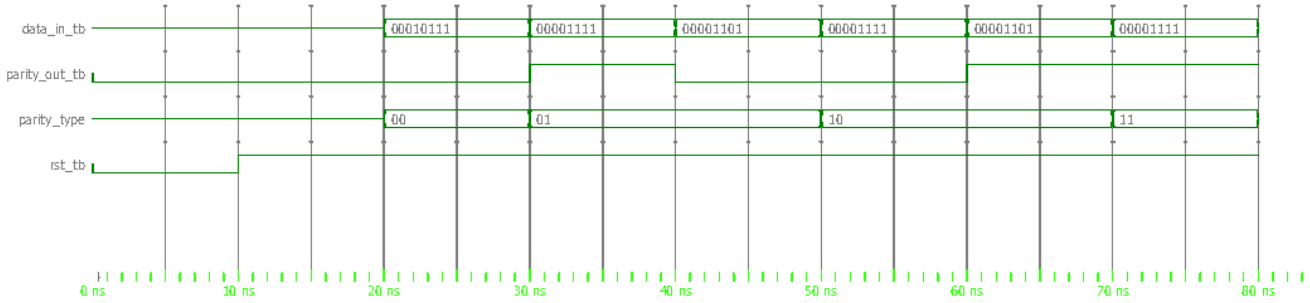


Figure 4: Parity Waveform Output

4.2 Frame Generator

4.2.1 Verilog Code

```
1  module frame_gen(
2      input rst,
3      input [7:0] data_in,
4      output reg parity_out ,
5      input [1:0] parity_type,
6      input stop_bits, data_length,
7      output reg [11:0] frame_out
8  );
9
10
11 wire parity;
12 assign parity = ^data_in;
13
14
15
16 reg [3:0] length  = 4'd7;
17
18 parameter idle = 1'b1;
19 reg  start = 1'b0;
20 reg [1:0] stop = 1'b1;
21
```

```
always @(negedge rst)
begin
    frame_out = 10'd0;
    parity_out = 1'b0;
end

always @(*) begin
    if (data_length) begin
        length = 4'd8;
        end
    else
        begin
        length = 4'd7;
        end
end

always @(*) begin
    if (stop_bits)
    stop = 2'b11;
else
    stop = 1'b1;
end

 always@(*)
    begin
     begin
        case (parity_type)
            2'b00: // No parity
                        parity_out = 1'b0; // 0 NO ERROR , 1 ERROR
            2'b01: // Odd parity
                begin
                    if (parity)
                        parity_out = 1'b0;
                    else
                        parity_out = 1'b1;
                end
            2'b10: // Even parity
                begin
                    if (parity)
                        parity_out = 1'b1;
                    else
                        parity_out = 1'b0;
                end

            2'b11:    // Odd parity (parallel)
                begin
                    if (parity)
                        parity_out = 1'b0;
                    else
                        parity_out = 1'b1;
                end
        endcase
        end
    end

always @(*)
begin
                if (parity_type == 1'b00 ||  parity_type == 1'b11)
           frame_out = {start, data_in[7:0], stop, idle};
                else frame_out = {start, data_in[7:0],parity_out,stop};
end

endmodule
```

4.2.2 Testbench

```
`timescale 1ns / 1ns

module frame_gen_tb;

  // Inputs
  reg rst;
  reg [7:0] data_in;
  reg [1:0] parity_type;
  reg stop_bits;
  reg data_length;

  // Outputs
  wire [11:0] frame_out;
  wire parity_out;

  // Instantiate the Unit Under Test (UUT)
  frame_gen uut (
    .rst(rst),
    .data_in(data_in),
    .parity_out(parity_out),
    .parity_type(parity_type),
    .stop_bits(stop_bits),
    .data_length(data_length),
    .frame_out(frame_out)
  );

  // Test sequence
  initial begin

        // Apply reset
        #10 rst = 0;
        #10 rst = 1;

    // Initialize Inputs
    parity_type = 2'b00; // No parity
    stop_bits = 0;       // 1 stop bit
    data_length = 1;     // 8 data bits
        data_in = 8'b10101010;


        //1
    $display("Test case 1: No parity, 1 stop bit, 8 data bits: %b", data_in);
        #5
    $display("frame_out = %b, parity_out = %b", frame_out, parity_out);

        #10;
    rst = 0; #5;
    rst = 1; #5;

        //2
        parity_type = 2'b01; // Odd parity
    stop_bits = 1;       // 2 stop bits
    data_length = 0;     // 7 data bits
        data_in = 8'b1100110; // New data to be transmitted
    $display("Test case 2: Odd parity, 2 stop bits, 7 data bits: %b", data_in);
        #5
    $display("frame_out = %b, parity_out = %b", frame_out, parity_out);
        #10;
    rst = 0; #5;
    rst = 1; #5;


        //3
        parity_type = 2'b10; // Even parity
    stop_bits = 0;       // 1 stop bit
    data_length = 1;     // 8 data bits
        #5 data_in = 8'b01101001; // New data to be transmitted
    $display("Test case 3: Even parity, 1 stop bit, 8 data bits: %b", data_in);
        #5
```

```
70      $display("frame_out = %b, parity_out = %b", frame_out, parity_out);
71          #10;
72      rst = 0; #5;
73      rst = 1; #5;
74
75
76          //4
77          parity_type = 2'b11; // Use p_parity_out
78      stop_bits = 1;        // 2 stop bits
79      data_length = 1;      // 8 data bits
80          #5 data_in = 8'b11110000; // New data to be transmitted
81      $display("Test case 4: Use p_parity_out, 2 stop bits, 8 data bits: %b", data_in);
82          #5
83      $display("frame_out = %b, parity_out = %b", frame_out, parity_out);
84
85          #10;
86      rst = 0; #5;
87      rst = 1; #5;
88
89      // End simulation
90      #50;
91      $stop;
92    end
93
94  endmodule
95
```

4.2.3 Outputs

```
VSIM 93> run -all
# Test case 1: No parity, 1 stop bit, 8 data bits: 10101010
# frame_out = 010101010011, parity_out = 0
# Test case 2: Odd parity, 2 stop bits, 7 data bits: 01100110
# frame_out = 001100110111, parity_out = 1
# Test case 3: Even parity, 1 stop bit, 8 data bits: 01101001
# frame_out = 001101001001, parity_out = 0
# Test case 4: Use p_parity_out, 2 stop bits, 8 data bits: 11110000
# frame_out = 011110000111, parity_out = 1
```

Figure 5: Frame Generator Test Output

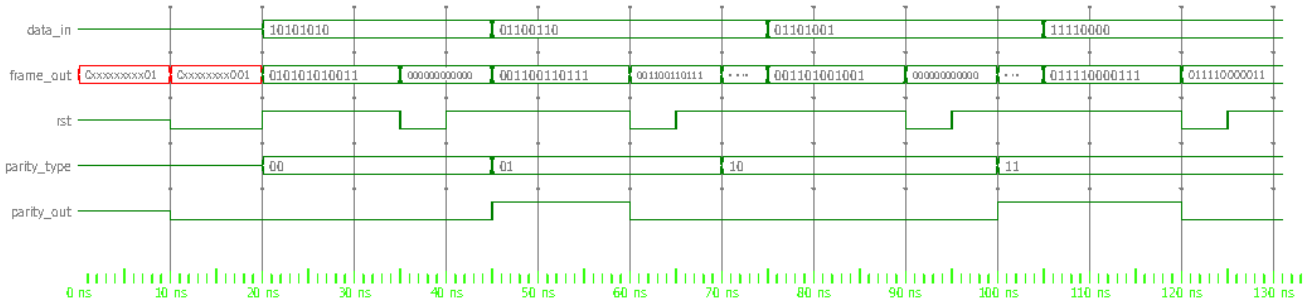


Figure 6: Parity Waveform Output

4.3 Baud Rate Generator

4.3.1 Verilog Code

```
module baud_gen(
  input [1:0] baud_rate,
  input rst, clock,
  output reg baud_out
);
reg [13:0] count;

always @(posedge clock or negedge rst) begin
    if (~rst) begin
        baud_out <= 1'b0;
        count <= 14'd0;
    end else begin
        case (baud_rate)
            2'b00: begin
                if (count < 14'd13003) begin
                    count <= count + 1'b1;
                    baud_out <= 1'b0;
                end else begin
                    count <= 14'd0;
                    baud_out <= 1'b1;
                end
            end
            2'b01: begin
                if (count < 10'd651) begin
                    count <= count + 1'b1;
                    baud_out <= 1'b0;
                end else begin
                    count <= 10'd0;
                    baud_out <= 1'b1;
                end
            end
            2'b10: begin
                if (count < 9'd326) begin
                    count <= count + 1'b1;
                    baud_out <= 1'b0;
                end else begin
                    count <= 9'd0;
                    baud_out <= 1'b1;
                end
            end
            2'b11: begin
                if (count < 8'd162) begin
                    count <= count + 1'b1;
                    baud_out <= 1'b0;
                end else begin
                    count <= 8'd0;
                    baud_out <= 1'b1;
                end
            end
        endcase
    end
end

endmodule

```

4.3.2 Testbench

```
`timescale 1ns/1ns

module baud_gen_tb ();

parameter clk = 20;

```

```
reg clk_tb;
reg rst_tb;
reg [1:0] buad_rate_tb;
wire baud_out_tb;

always #(clk/2) clk_tb = ~clk_tb;

initial begin
    initialize();
    reset();

    ////////////////// Test case 1 2400 baud ///////////////////////
    baud_gen_config(2'b00);
   // #(clk)
    chk_baud_out(2'b00, 1'd1);
    #(clk)

    ////////////////// Test case 2 4600 baud ///////////////////////
    baud_gen_config(2'b01);
    //#(clk)
    chk_baud_out(2'b01, 2'd2);
    #(clk)

    ////////////////// Test case 3 9600 baud ///////////////////////
    baud_gen_config(2'b10);
    //#(clk)
    chk_baud_out(2'b10, 2'd3);
    #(clk)

    ////////////////// Test case 4 19200 baud ///////////////////////
    baud_gen_config(2'b11);
    //#(clk)
    chk_baud_out(2'b11, 3'd4);
    #(3 * clk)
    $stop;
end

task initialize;
    begin
        clk_tb = 1'b0;
        rst_tb = 1'b1;
        buad_rate_tb = 2'b00;
    end
endtask

task reset;
    begin
        #(clk)
        rst_tb = 1'b0;
        #(clk)
        rst_tb = 1'b1;
        #(clk);
    end
endtask

task baud_gen_config;
    input [1:0] baud_rate;
    begin
        buad_rate_tb = baud_rate;
    end
endtask

task chk_baud_out;
    input [1:0] baud_rate;
    input [2:0] test_case;
    reg expected_out;
    reg chk;
    begin
        expected_out = 1'b1;
        case (baud_rate)
            2'b00: begin
```

```
                #(13003 * clk)
                @(negedge clk_tb)
                if (baud_out_tb == expected_out)
                    chk = 1'b1;
                else
                    chk = 1'b0;
            end
            2'b01: begin
                #(651 * clk)
                @(negedge clk_tb)
                if (baud_out_tb == expected_out)
                    chk = 1'b1;
                else
                    chk = 1'b0;
            end
            2'b10: begin
                #(326 * clk)
                @(negedge clk_tb)
                if (baud_out_tb == expected_out)
                    chk = 1'b1;
                else
                    chk = 1'b0;
            end
            2'b11: begin
                #(162 * clk)
                @(negedge clk_tb)
                if (baud_out_tb == expected_out)
                    chk = 1'b1;
                else
                    chk = 1'b0;
            end
        endcase
        if (chk == 1'b1)
            $display("test %d passed successfully", test_case);
        else
            $display("test %d failed", test_case);
    end
endtask

baud_gen DUT (
    .clock(clk_tb),
    .rst(rst_tb),
    .baud_rate(buad_rate_tb),
    .baud_out(baud_out_tb)
    );
endmodule

```

4.3.3 Outputs

```
VSIM 6> run -all
# test 1 passed successfully
# test 2 passed successfully
# test 3 passed successfully
# test 4 passed successfully
```

Figure 7: Baud Generator Test Output

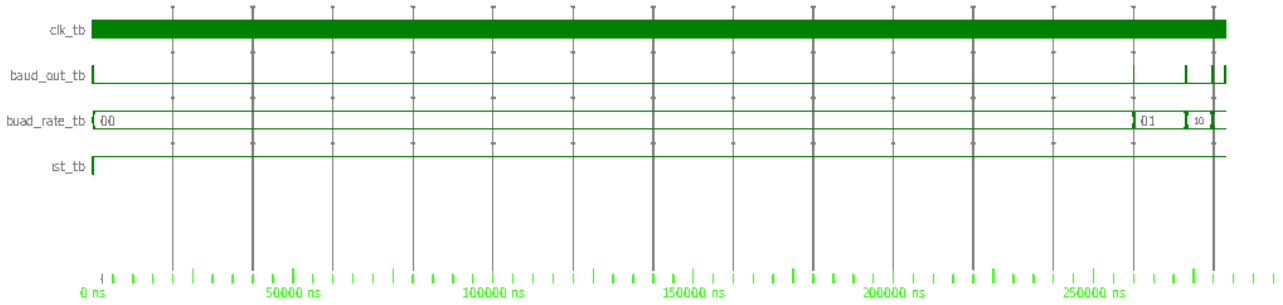


Figure 8: Baud Generator Waveform Output

4.4 Parallel-In Serial-Out Shift Register

4.4.1 Verilog Code

```
module piso(
    input [11:0] frame_out,
    input [1:0]  parity_type,
    input stop_bits, data_length, send, baud_out, rst,
    output reg  data_out, P_parity_out, tx_active, tx_done
);
reg [4:0] count;
reg [10:0] s_data;
wire count_max;

always @(posedge baud_out or negedge rst) begin
    if(~rst) begin
        s_data   <= 0;
    end
    else if (send) begin
        s_data   <= frame_out;
    end
end

always @(posedge baud_out)
begin
    if (count_max)
        begin
            data_out = 0;
            tx_active = 0;
        end
        else
            begin
                data_out = s_data [count];
                tx_active = 1;
            end
end

always@(posedge baud_out or negedge rst)
       begin
          if(!rst)
            count <= 4'b1000;
          else if (~send)
            begin
                count    <= 4'b0 ;
            end
          else if (!count_max)
            begin
                count    <= count + 4'b1 ;
            end
       end


```

```
   assign count_max = ( count == 4'b1011 );
   assign tx_done  = count_max;

always @(posedge baud_out or negedge rst) begin
    if (~rst)
    P_parity_out = 1'b0;
    else begin
    case (parity_type)                  // 0 no parity, 1 parity
        2'b00:
        P_parity_out = 1'b0;
        2'b01:
        P_parity_out = 1'b1;
        2'b10:
        P_parity_out = 1'b1;
        2'b11:
        P_parity_out = 1'b0;
    endcase
    end
end

endmodule

```

[pt]

4.4.2 Testbench

```
`timescale 1ns / 1ns

module piso_tb;

  // Inputs
  reg [11:0] frame_out;
  reg [1:0]  parity_type;
  reg stop_bits;
  reg data_length;
  reg send;
  reg baud_out;
  reg rst;

  // Outputs
  wire data_out;
  wire P_parity_out;
  wire tx_active;
  wire tx_done;

  // Instantiation
  piso dut (
    .frame_out(frame_out),
    .parity_type(parity_type),
    .stop_bits(stop_bits),
    .data_length(data_length),
    .send(send),
    .baud_out(baud_out),
    .rst(rst),
    .data_out(data_out),
    .P_parity_out(P_parity_out),
    .tx_active(tx_active),
    .tx_done(tx_done)
  );

  // Clock generation
  always #5 baud_out = ~baud_out;

  // Test sequence
  initial begin

        // Apply reset
```

```
    rst = 0;
    #10 rst = 1;

    // Initialize Inputs
    frame_out = 11'b10101010101; // Example frame
    parity_type = 2'b00; // No parity
    stop_bits = 0;       // 1 stop bit
    data_length = 1;     // 8 data bits
    send = 0;
    baud_out = 0;



    // Test case 1: No parity, 1 stop bit, 8 data bits
        $display("Test Case 1: %b", frame_out);
    send = 1;
    wait(tx_done);
    send = 0;
    #20;

    // Test case 2: Odd parity, 2 stop bits, 7 data bits
        rst = 0; #5;
    rst = 1; #5;
    parity_type = 2'b01; // Odd parity
    stop_bits = 1;       // 2 stop bits
    data_length = 0;     // 7 data bits
    frame_out = 11'b11001100110; // New frame
        $display("Test Case 2: %b", frame_out);
    #10;

    send = 1;
    wait(tx_done);
    send = 0;
    #20;

    // Test case 3: Even parity, 1 stop bit, 8 data bits
        rst = 0; #5;
    rst = 1; #5;
    parity_type = 2'b10; // Even parity
    stop_bits = 0;       // 1 stop bit
    data_length = 1;     // 8 data bits
    frame_out = 11'b01101011010; // New frame
        $display("Test Case 3: %b", frame_out);
    #10;
    send = 1;
    wait(tx_done);
    send = 0;
    #20;

    // Test case 4: Use p_parity_out, 2 stop bits, 8 data bits
        rst = 0; #5;
    rst = 1; #5;
    parity_type = 2'b11; // Use p_parity_out
    stop_bits = 1;       // 2 stop bits
    data_length = 1;     // 8 data bits
    frame_out = 11'b11110011110; // New frame
        $display("Test Case 4: %b", frame_out);
    #10;
    send = 1;
    wait(tx_done);
    send = 0;
    #20;

    // End simulation
    #50;
    $stop;
  end

  // Monitor the outputs
  initial begin
    $monitor("Time = %0t : data_out = %b, P_parity_out = %b, tx_active = %b, tx_done = %b", $time, data_out, P_parity_out, tx_activ
```

```
  end

endmodule

```

4.4.3 Outputs

```
VSIM 112> run -all
# Time = 0 : data_out = x, P_parity_out = 0, tx_active = x, tx_done = 0
# Test Case 1: 010101010101
# Time = 10 : data_out = 0, P_parity_out = 0, tx_active = 1, tx_done = 0
# Time = 30 : data_out = 1, P_parity_out = 0, tx_active = 1, tx_done = 1
# Time = 40 : data_out = 0, P_parity_out = 0, tx_active = 0, tx_done = 1
# Time = 110 : data_out = 0, P_parity_out = 0, tx_active = 0, tx_done = 0
# Time = 120 : data_out = 1, P_parity_out = 0, tx_active = 1, tx_done = 0
# Test Case 2: 011001100110
# Time = 140 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 160 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 180 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 200 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 220 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 240 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 260 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 270 : data_out = 0, P_parity_out = 0, tx_active = 1, tx_done = 0
# Test Case 3: 001101011010
# Time = 280 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 300 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 310 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 320 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 340 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 350 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 360 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 370 : data_out = 1, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 390 : data_out = 0, P_parity_out = 1, tx_active = 1, tx_done = 0
# Time = 410 : data_out = 0, P_parity_out = 0, tx_active = 1, tx_done = 0
# Test Case 4: 011110011110
# Time = 440 : data_out = 1, P_parity_out = 0, tx_active = 1, tx_done = 0
# Time = 480 : data_out = 0, P_parity_out = 0, tx_active = 1, tx_done = 0
# Time = 500 : data_out = 1, P_parity_out = 0, tx_active = 1, tx_done = 0
# Time = 540 : data_out = 0, P_parity_out = 0, tx_active = 1, tx_done = 0
```

Figure 9: PISO Test Output

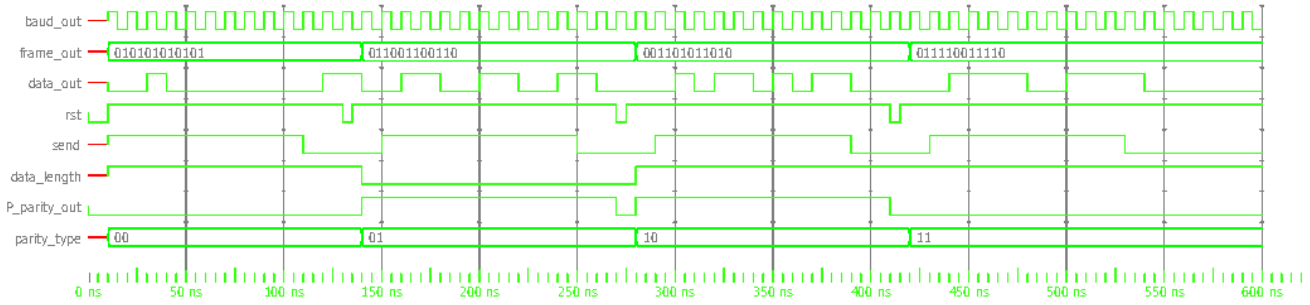


Figure 10: PISO Waveform Output

4.5 UART Tx (Top Module)

4.5.1 Verilog Code

```
module uart_tx(
        //DO NOT EDIT any part of this port declaration
        input                clock, rst, send,
        input [1:0]        buad_rate,
        input [7:0] data_in,
        input [1:0] parity_type,         //refer to the block comment above.
        input                stop_bits,                //low when using 1 stop bit, high when using two stop bits
        input                data_length,         //low when using 7 data bits, high when using 8.
```

```
9
10          output reg          data_out,                //Serial data_out
11          output reg          p_parity_out,          //parallel odd parity output, low when using the frame parity.
12          output reg          tx_active,               //high when Tx is transmitting, low when idle.
13          output reg          tx_done                 //high when transmission is done, low when not.
14  );
15
16          //You MAY EDIT these signals, or module instantiations.
17          reg parity_out, baud_out;
18          reg [10:0] frame_out;
19
20          //sub_modules
21          parity     parity_gen1 (rst, data_in, parity_type, parity_out);
22          frame_gen  frame_gen1  (rst, data_in, parity_out, parity_type, stop_bits, data_length, frame_out);
23          baud_gen   baud_gen1        (rst, clock, baud_rate, baud_out);
24          piso       shift_reg1        (rst, frame_out, parity_type, stop_bits, data_length, send, baud_out,
25                                       data_out, p_parity_out, tx_active, tx_done);
26
27
28  endmodule
29
```

4.6 Testbench

```
1   `timescale 1ns / 1ps
2
3   `timescale 1ns / 1ns
4
5   module uart_tx_tb;
6
7     // Inputs
8     reg clock;
9     reg rst;
10    reg send;
11    reg [1:0] baud_rate;
12    reg [7:0] data_in;
13    reg [1:0] parity_type;
14    reg stop_bits;
15    reg data_length;
16
17    // Outputs
18    wire data_out;
19    wire p_parity_out;
20    wire tx_active;
21    wire tx_done;
22
23    // Instantiation
24    uart_tx dut (
25      .clock(clock),
26      .rst(rst),
27      .send(send),
28      .baud_rate(baud_rate),
29      .data_in(data_in),
30      .parity_type(parity_type),
31      .stop_bits(stop_bits),
32      .data_length(data_length),
33      .data_out(data_out),
34      .p_parity_out(p_parity_out),
35      .tx_active(tx_active),
36      .tx_done(tx_done)
37    );
38
39    // Clock generation
40    initial begin
41      clock = 0;
42      forever #10 clock = ~clock; // 50MHz clock
43    end
44
45    // Test sequence
46    initial begin
```

```
    // Initialize Inputs
    rst = 1'b1;
    send = 1'b0;
    baud_rate = 2'b10;        // 9600 baud
    data_in = 8'b10101010;    // Data to be transmitted
    parity_type = 2'b00;      // No parity
    stop_bits = 0;            // 1 stop bit
    data_length = 1;          // 8 data bits

    // Reset the DUT
    #20;
    rst = 1'b0;
        #20
        rst = 1'b1;

    // Test case 1: No parity, 9600 baud, 8 data bits, 1 stop bit
    #100;
    send = 1'b1;
    #20;
    send = 1'b0;

    // Wait for transmission to complete
    wait(tx_done);
        $stop;

    // Test case 2: Even parity, 4800 baud, 7 data bits, 2 stop bits
    #100;
    parity_type = 2'b10;      // Even parity
    baud_rate = 2'b01;        // 4800 baud
    data_length = 1'b0;          // 7 data bits
    stop_bits = 1'b1;            // 2 stop bits
    data_in = 8'b1100110;     // New data to be transmitted

    #100;
    send = 1'b1;
    #20;
    send = 1'b0;

    // Wait for transmission to complete
    wait(tx_done);
        $stop;

    // Test case 3: Odd parity, 2400 baud, 8 data bits, 1 stop bit
    #100;
    parity_type = 2'b01;      // Odd parity
    baud_rate = 2'b00;        // 2400 baud
    data_length = 1'b1;          // 8 data bits
    stop_bits = 1'b0;            // 1 stop bit
    data_in = 8'b01101001;    // New data to be transmitted

    #100;
    send = 1'b1;
    #20;
    send = 1'b0;

    // Wait for transmission to complete
    wait(tx_done);
        $stop;


    // Test case 4: Use p_parity_out, 19.2K baud, 8 data bits, 2 stop bits
    #100;
    parity_type = 2'b11;      // Use p_parity_out
    baud_rate = 2'b11;        // 19.2K baud
    data_length = 1'b1;          // 8 data bits
    stop_bits = 1'b1;            // 2 stop bits
    data_in = 8'b11110000;    // New data to be transmitted

    #100;
    send = 1'b1;
    #20;
```

```
118     send = 1'b0;
119
120     // Wait for transmission to complete
121     wait(tx_done);
122
123     // End simulation
124     #100;
125     $stop;
126   end
127 endmodule
128
```

4.6.1 Outputs

Figure 11: Parity Test Output

Figure 12: Parity Waveform Output