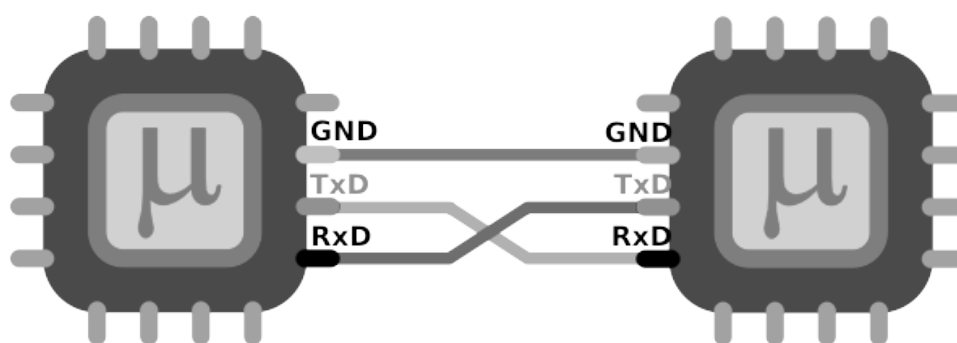


# UART Tx Implementation

## 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.

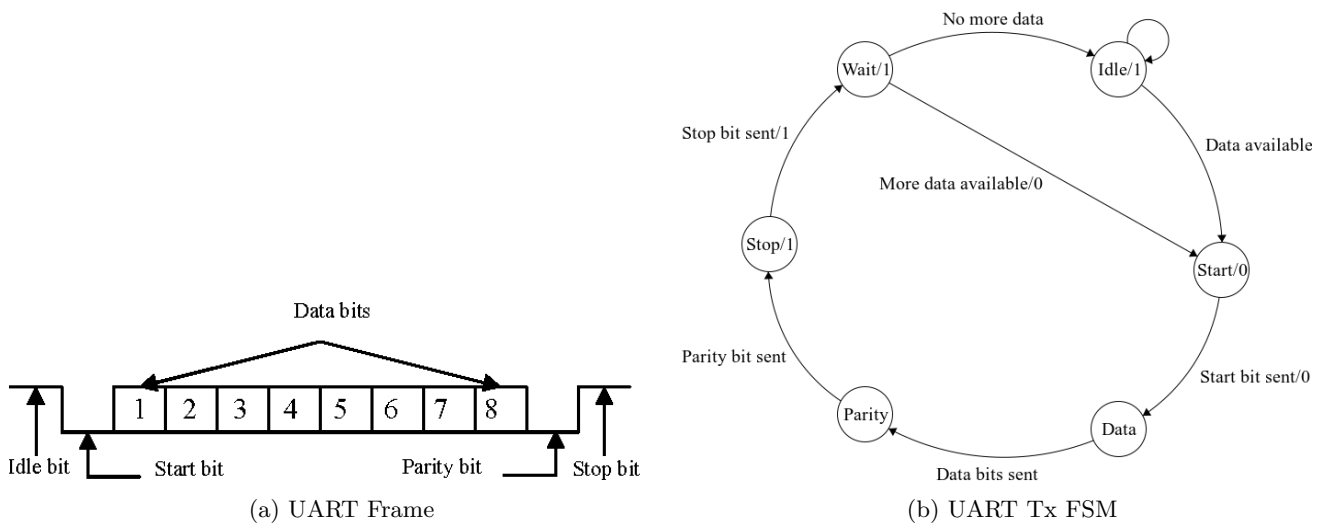


Figure 1: UART Tx Operation

- **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])

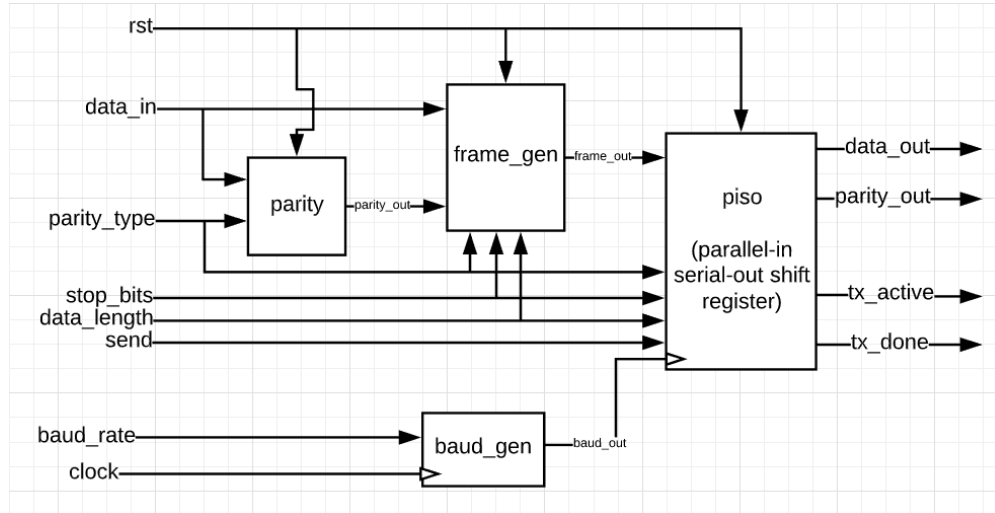


Figure 2: UART Tx Architecture

#### 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

```

1  module parity (
2      input rst,
3      input [7:0] data_in,
4      input [1:0] parity_type,
5      output reg parity_out
6  );
7
8      wire parity;
9      assign parity = ^data_in; // 1 if parity is odd, 0 if parity is even
10
11     always @(negedge rst)
12     begin
13         if (~rst) begin
14             parity_out = 1'b0;
15         end
16     end
17
18     always@(*)
19     begin
20         begin
21             case (parity_type)
22                 2'b00: // No parity
23                     parity_out = 1'b0; // 0 NO ERROR , 1 ERROR
24                 2'b01: // Odd parity
25                     begin
26                         if (parity)
27                             parity_out = 1'b0;
28                         else
29                             parity_out = 1'b1;
30                     end
31                 2'b10: // Even parity
32                     begin
33                         if (parity)
34                             parity_out = 1'b1;
35                         else
36                             parity_out = 1'b0;
37                     end
38                 2'b11: // Odd parity (parallel)
39                     begin
40                         if (parity)
41                             parity_out = 1'b0;
42                         else
43                             parity_out = 1'b1;
44                     end
45             endcase
46         end
47     end
48
49     end
50
51 endmodule
52

```

## 4.1.2 Testbench

```

1  `timescale 1ns/1ns
2  module parity_tb;
3
4  reg      rst_tb;
5  reg [7:0] data_in_tb;
6  reg [1:0] parity_type;
7
8  wire      parity_out_tb;
9
10 wire parity_tb;
11 assign paraty_tb = ^ data_in_tb ;
12
13 parity dut(
14     .data_in(data_in_tb),
15     .rst(rst_tb),
16     .parity_type(parity_type),
17     .parity_out(parity_out_tb)
18 );
19
20 initial
21 begin
22     rst_tb= 1'b0;
23     #10 rst_tb = 1'b1;
24
25     $display(" TEST 1 00 test " );
26     #10 parity_type = 2'b00;
27     data_in_tb = 8'b00010111;
28     if (parity_out_tb == 1'b0 ) $display("test 1 succeeded");
29     else $display("test 1 failed paraty out = %b ",parity_out_tb);
30
31     $display(" TEST 2 01 test " );
32     #10 parity_type = 2'b01;
33     data_in_tb = 8'b00001111;
34     if (parity_out_tb == 1'b0 && paraty_tb==1'b1 ) $display("test 2 succeeded");
35     else $display("test 2 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;
36
37     $display(" TEST 3 01 test " );
38     #10 parity_type = 2'b01;
39     data_in_tb = 8'b00001101;
40
41     if (parity_out_tb == 1'b1 && paraty_tb==1'b0 ) $display("test 3 succeeded");
42     else $display("test 3 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;
43
44     $display(" TEST 4 10 test " );
45     #10 parity_type = 2'b10;
46     data_in_tb = 8'b00001111;
47     if (parity_out_tb == 1'b1 && paraty_tb==1'b1 ) $display("test 4 succeeded");
48     else $display("test 4 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;
49
50     $display(" TEST 5 10 test " );
51     #10 parity_type = 2'b10;
52     data_in_tb = 8'b00001101;
53     if (parity_out_tb == 1'b0 && paraty_tb==1'b0 ) $display("test 5 succeeded");
54     else $display("test 5 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;
55
56     $display(" TEST 6 11 test " );
57     #10 parity_type = 2'b11;
58     data_in_tb = 8'b00001111;
59     if (parity_out_tb == 1'b0 && paraty_tb==1'b1 ) $display("test 6 succeeded");
60     else $display("test 6 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;
61
62     $display(" TEST 7 11 test " );
63     #10 parity_type = 2'b01;
64     data_in_tb = 8'b00001101;
65     if (parity_out_tb == 1'b1 && paraty_tb==1'b0 ) $display("test 7 succeeded");
66     else $display("test 7 failed paraty out = %b and paraty = ",parity_out_tb ,paraty_tb) ;$stop
67
68 end
69 endmodule

```

### 4.1.3 Outputs

```

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

```

Figure 3: Parity Test Output

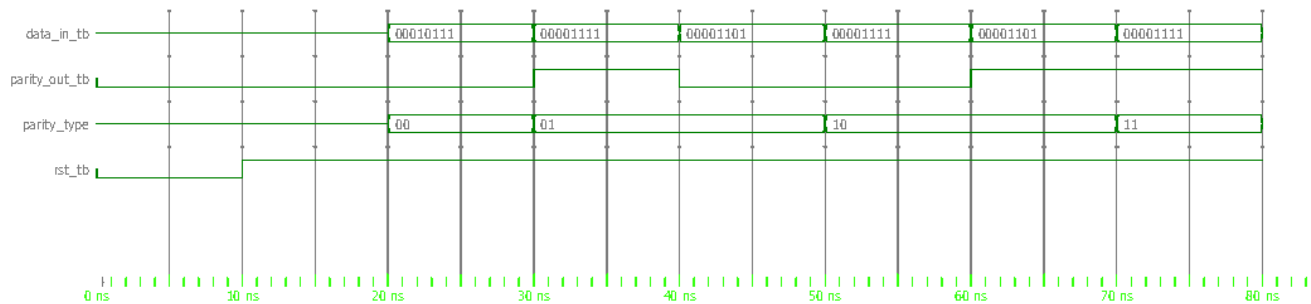


Figure 4: Parity Waveform Output

## 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

```

```

22 always @(negedge rst)
23 begin
24     frame_out = 10'd0;
25     parity_out = 1'b0;
26 end
27
28 always @(*) begin
29     if (data_length) begin
30         length = 4'd8;
31     end
32     else
33     begin
34         length = 4'd7;
35     end
36 end
37
38 always @(*) begin
39     if (stop_bits)
40         stop = 2'b11;
41     else
42         stop = 1'b1;
43 end
44
45 always@(*)
46 begin
47     begin
48         case (parity_type)
49             2'b00: // No parity
50                 parity_out = 1'b0; // 0 NO ERROR , 1 ERROR
51             2'b01: // Odd parity
52                 begin
53                     if (parity)
54                         parity_out = 1'b0;
55                     else
56                         parity_out = 1'b1;
57                 end
58             2'b10: // Even parity
59                 begin
60                     if (parity)
61                         parity_out = 1'b1;
62                     else
63                         parity_out = 1'b0;
64                 end
65             2'b11: // Odd parity (parallel)
66                 begin
67                     if (parity)
68                         parity_out = 1'b0;
69                     else
70                         parity_out = 1'b1;
71                 end
72             endcase
73         end
74     end
75 end
76
77 always @(*)
78 begin
79     if (parity_type == 1'b00 || parity_type == 1'b11)
80         frame_out = {start, data_in[7:0], stop, idle};
81     else frame_out = {start, data_in[7:0], parity_out, stop};
82 end
83
84 endmodule
85

```

## 4.2.2 Testbench

```

1  `timescale 1ns / 1ns
2
3  module frame_gen_tb;
4
5      // Inputs
6      reg rst;
7      reg [7:0] data_in;
8      reg [1:0] parity_type;
9      reg stop_bits;
10     reg data_length;
11
12     // Outputs
13     wire [11:0] frame_out;
14     wire parity_out;
15
16     // Instantiate the Unit Under Test (UUT)
17     frame_gen uut (
18         .rst(rst),
19         .data_in(data_in),
20         .parity_out(parity_out),
21         .parity_type(parity_type),
22         .stop_bits(stop_bits),
23         .data_length(data_length),
24         .frame_out(frame_out)
25     );
26
27     // Test sequence
28     initial begin
29
30         // Apply reset
31         #10 rst = 0;
32         #10 rst = 1;
33
34         // Initialize Inputs
35         parity_type = 2'b00; // No parity
36         stop_bits = 0;      // 1 stop bit
37         data_length = 1;    // 8 data bits
38         data_in = 8'b10101010;
39
40
41         //1
42         $display("Test case 1: No parity, 1 stop bit, 8 data bits: %b", data_in);
43         #5
44         $display("frame_out = %b, parity_out = %b", frame_out, parity_out);
45
46         #10;
47         rst = 0; #5;
48         rst = 1; #5;
49
50         //2
51         parity_type = 2'b01; // Odd parity
52         stop_bits = 1;      // 2 stop bits
53         data_length = 0;    // 7 data bits
54         data_in = 8'b1100110; // New data to be transmitted
55         $display("Test case 2: Odd parity, 2 stop bits, 7 data bits: %b", data_in);
56         #5
57         $display("frame_out = %b, parity_out = %b", frame_out, parity_out);
58         #10;
59         rst = 0; #5;
60         rst = 1; #5;
61
62         //3
63         parity_type = 2'b10; // Even parity
64         stop_bits = 0;      // 1 stop bit
65         data_length = 1;    // 8 data bits
66         #5 data_in = 8'b01101001; // New data to be transmitted
67         $display("Test case 3: Even parity, 1 stop bit, 8 data bits: %b", data_in);
68         #5
69

```



```

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

```

VSIM93> 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

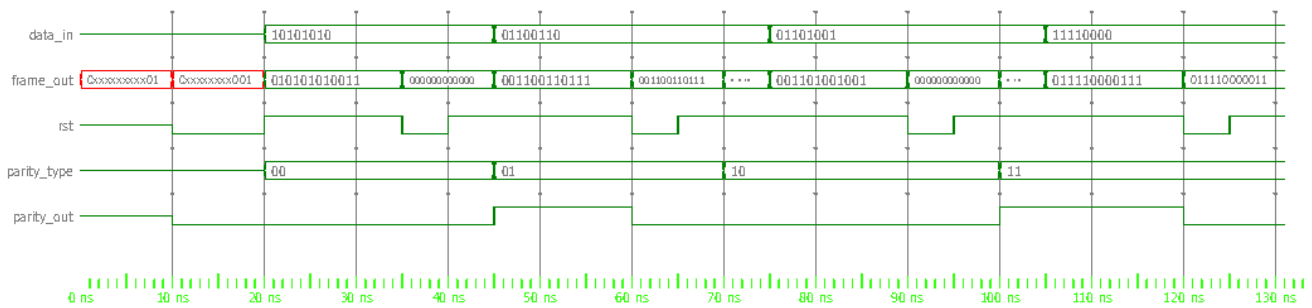


Figure 6: Parity Waveform Output

## 4.3 Baud Rate Generator

### 4.3.1 Verilog Code

```

1  module baud_gen(
2      input [1:0] baud_rate,
3      input rst, clock,
4      output reg baud_out
5  );
6      reg [13:0] count;
7
8      always @(posedge clock or negedge rst) begin
9          if (~rst) begin
10             baud_out <= 1'b0;
11             count <= 14'd0;
12         end else begin
13             case (baud_rate)
14                 2'b00: begin
15                     if (count < 14'd13003) begin
16                         count <= count + 1'b1;
17                         baud_out <= 1'b0;
18                     end else begin
19                         count <= 14'd0;
20                         baud_out <= 1'b1;
21                     end
22                 end
23                 2'b01: begin
24                     if (count < 10'd651) begin
25                         count <= count + 1'b1;
26                         baud_out <= 1'b0;
27                     end else begin
28                         count <= 10'd0;
29                         baud_out <= 1'b1;
30                     end
31                 end
32                 2'b10: begin
33                     if (count < 9'd326) begin
34                         count <= count + 1'b1;
35                         baud_out <= 1'b0;
36                     end else begin
37                         count <= 9'd0;
38                         baud_out <= 1'b1;
39                     end
40                 end
41                 2'b11: begin
42                     if (count < 8'd162) begin
43                         count <= count + 1'b1;
44                         baud_out <= 1'b0;
45                     end else begin
46                         count <= 8'd0;
47                         baud_out <= 1'b1;
48                     end
49                 end
50             endcase
51         end
52     end
53
54 endmodule
55

```

### 4.3.2 Testbench

```

1  `timescale 1ns/1ns
2
3  module baud_gen_tb ();
4
5      parameter clk = 20;
6

```

```

7  reg clk_tb;
8  reg rst_tb;
9  reg [1:0] buad_rate_tb;
10 wire baud_out_tb;
11
12 always #(clk/2) clk_tb = ~clk_tb;
13
14 initial begin
15     initialize();
16     reset();
17
18     ////////////////////////////////// Test case 1 2400 baud //////////////////////////////////
19     baud_gen_config(2'b00);
20     // #(clk)
21     chk_baud_out(2'b00, 1'd1);
22     #(clk)
23
24     ////////////////////////////////// Test case 2 4600 baud //////////////////////////////////
25     baud_gen_config(2'b01);
26     // #(clk)
27     chk_baud_out(2'b01, 2'd2);
28     #(clk)
29
30     ////////////////////////////////// Test case 3 9600 baud //////////////////////////////////
31     baud_gen_config(2'b10);
32     // #(clk)
33     chk_baud_out(2'b10, 2'd3);
34     #(clk)
35
36     ////////////////////////////////// Test case 4 19200 baud //////////////////////////////////
37     baud_gen_config(2'b11);
38     // #(clk)
39     chk_baud_out(2'b11, 3'd4);
40     #(3 * clk)
41     $stop;
42 end
43
44 task initialize;
45     begin
46         clk_tb = 1'b0;
47         rst_tb = 1'b1;
48         buad_rate_tb = 2'b00;
49     end
50 endtask
51
52 task reset;
53     begin
54         #(clk)
55         rst_tb = 1'b0;
56         #(clk)
57         rst_tb = 1'b1;
58         #(clk);
59     end
60 endtask
61
62 task baud_gen_config;
63     input [1:0] baud_rate;
64     begin
65         buad_rate_tb = baud_rate;
66     end
67 endtask
68
69 task chk_baud_out;
70     input [1:0] baud_rate;
71     input [2:0] test_case;
72     reg expected_out;
73     reg chk;
74     begin
75         expected_out = 1'b1;
76         case (baud_rate)
77             2'b00: begin

```

```

78         #(13003 * clk)
79         @(negedge clk_tb)
80         if (baud_out_tb == expected_out)
81             chk = 1'b1;
82         else
83             chk = 1'b0;
84     end
85     2'b01: begin
86         #(651 * clk)
87         @(negedge clk_tb)
88         if (baud_out_tb == expected_out)
89             chk = 1'b1;
90         else
91             chk = 1'b0;
92     end
93     2'b10: begin
94         #(326 * clk)
95         @(negedge clk_tb)
96         if (baud_out_tb == expected_out)
97             chk = 1'b1;
98         else
99             chk = 1'b0;
100    end
101    2'b11: begin
102        #(162 * clk)
103        @(negedge clk_tb)
104        if (baud_out_tb == expected_out)
105            chk = 1'b1;
106        else
107            chk = 1'b0;
108    end
109    endcase
110    if (chk == 1'b1)
111        $display("test %d passed successfully", test_case);
112    else
113        $display("test %d failed", test_case);
114    end
115 endtask
116
117 baud_gen DUT (
118     .clock(clk_tb),
119     .rst(rst_tb),
120     .baud_rate(baad_rate_tb),
121     .baud_out(baud_out_tb)
122 );
123 endmodule
124

```

### 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

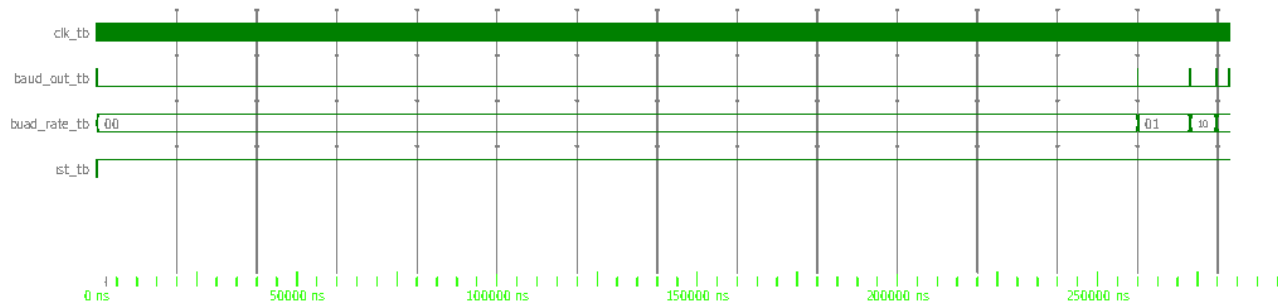


Figure 8: Baud Generator Waveform Output

## 4.4 Parallel-In Serial-Out Shift Register

### 4.4.1 Verilog Code

```

1  module piso(
2      input [11:0] frame_out,
3      input [1:0] parity_type,
4      input stop_bits, data_length, send, baud_out, rst,
5      output reg data_out, P_parity_out, tx_active, tx_done
6  );
7  reg [4:0] count;
8  reg [10:0] s_data;
9  wire count_max;
10
11  always @(posedge baud_out or negedge rst) begin
12      if (~rst) begin
13          s_data <= 0;
14      end
15      else if (send) begin
16          s_data <= frame_out;
17      end
18  end
19
20  always @(posedge baud_out)
21  begin
22      if (count_max)
23      begin
24          data_out = 0;
25          tx_active = 0;
26      end
27      else
28      begin
29          data_out = s_data [count];
30          tx_active = 1;
31      end
32  end
33
34  always@(posedge baud_out or negedge rst)
35  begin
36      if (!rst)
37          count <= 4'b1000;
38      else if (~send)
39      begin
40          count <= 4'b0 ;
41      end
42      else if (!count_max)
43      begin
44          count <= count + 4'b1 ;
45      end
46  end
47
48  end
    
```

```

49     assign count_max = ( count == 4'b1011 );
50     assign tx_done   = count_max;
51
52 always @(posedge baud_out or negedge rst) begin
53     if (~rst)
54         P_parity_out = 1'b0;
55     else begin
56         case (parity_type)                // 0 no parity, 1 parity
57             2'b00:
58                 P_parity_out = 1'b0;
59             2'b01:
60                 P_parity_out = 1'b1;
61             2'b10:
62                 P_parity_out = 1'b1;
63             2'b11:
64                 P_parity_out = 1'b0;
65         endcase
66     end
67 end
68
69 endmodule
70

```

[pt]

#### 4.4.2 Testbench

```

1  `timescale 1ns / 1ns
2
3  module piso_tb;
4
5      // Inputs
6      reg [11:0] frame_out;
7      reg [1:0]  parity_type;
8      reg stop_bits;
9      reg data_length;
10     reg send;
11     reg baud_out;
12     reg rst;
13
14     // Outputs
15     wire data_out;
16     wire P_parity_out;
17     wire tx_active;
18     wire tx_done;
19
20     // Instantiation
21     piso dut (
22         .frame_out(frame_out),
23         .parity_type(parity_type),
24         .stop_bits(stop_bits),
25         .data_length(data_length),
26         .send(send),
27         .baud_out(baud_out),
28         .rst(rst),
29         .data_out(data_out),
30         .P_parity_out(P_parity_out),
31         .tx_active(tx_active),
32         .tx_done(tx_done)
33     );
34
35     // Clock generation
36     always #5 baud_out = ~baud_out;
37
38     // Test sequence
39     initial begin
40
41         // Apply reset

```

```

42     rst = 0;
43     #10 rst = 1;
44
45     // Initialize Inputs
46     frame_out = 11'b10101010101; // Example frame
47     parity_type = 2'b00; // No parity
48     stop_bits = 0; // 1 stop bit
49     data_length = 1; // 8 data bits
50     send = 0;
51     baud_out = 0;
52
53
54
55     // Test case 1: No parity, 1 stop bit, 8 data bits
56     $display("Test Case 1: %b", frame_out);
57     send = 1;
58     wait(tx_done);
59     send = 0;
60     #20;
61
62     // Test case 2: Odd parity, 2 stop bits, 7 data bits
63     rst = 0; #5;
64     rst = 1; #5;
65     parity_type = 2'b01; // Odd parity
66     stop_bits = 1; // 2 stop bits
67     data_length = 0; // 7 data bits
68     frame_out = 11'b11001100110; // New frame
69     $display("Test Case 2: %b", frame_out);
70     #10;
71
72     send = 1;
73     wait(tx_done);
74     send = 0;
75     #20;
76
77     // Test case 3: Even parity, 1 stop bit, 8 data bits
78     rst = 0; #5;
79     rst = 1; #5;
80     parity_type = 2'b10; // Even parity
81     stop_bits = 0; // 1 stop bit
82     data_length = 1; // 8 data bits
83     frame_out = 11'b01101011010; // New frame
84     $display("Test Case 3: %b", frame_out);
85     #10;
86     send = 1;
87     wait(tx_done);
88     send = 0;
89     #20;
90
91     // Test case 4: Use p_parity_out, 2 stop bits, 8 data bits
92     rst = 0; #5;
93     rst = 1; #5;
94     parity_type = 2'b11; // Use p_parity_out
95     stop_bits = 1; // 2 stop bits
96     data_length = 1; // 8 data bits
97     frame_out = 11'b11110011110; // New frame
98     $display("Test Case 4: %b", frame_out);
99     #10;
100    send = 1;
101    wait(tx_done);
102    send = 0;
103    #20;
104
105    // End simulation
106    #50;
107    $stop;
108    end
109
110    // Monitor the outputs
111    initial begin
112        $monitor("Time = %0t : data_out = %b, P_parity_out = %b, tx_active = %b, tx_done = %b", $time, data_out, P_parity_out, tx_active);

```

```

113     end
114
115 endmodule
116

```

### 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

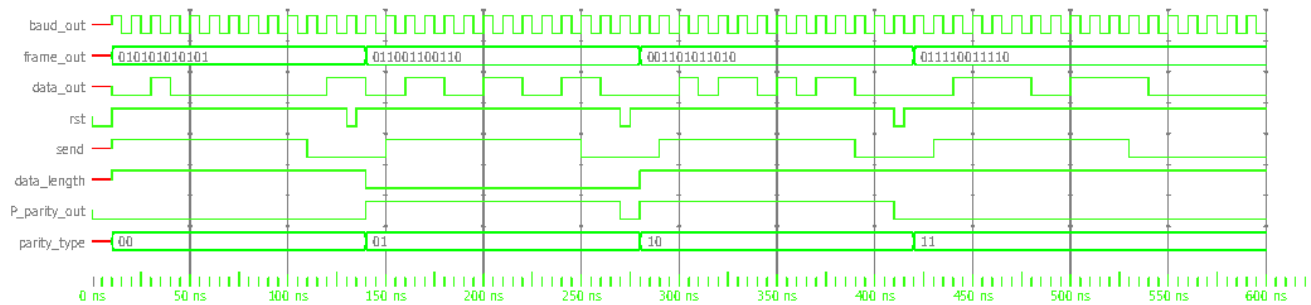


Figure 10: PISO Waveform Output

## 4.5 UART Tx (Top Module)

### 4.5.1 Verilog Code

```

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

```

```

47 // Initialize Inputs
48 rst = 1'b1;
49 send = 1'b0;
50 baud_rate = 2'b10; // 9600 baud
51 data_in = 8'b10101010; // Data to be transmitted
52 parity_type = 2'b00; // No parity
53 stop_bits = 0; // 1 stop bit
54 data_length = 1; // 8 data bits
55
56 // Reset the DUT
57 #20;
58 rst = 1'b0;
59 #20
60 rst = 1'b1;
61
62 // Test case 1: No parity, 9600 baud, 8 data bits, 1 stop bit
63 #100;
64 send = 1'b1;
65 #20;
66 send = 1'b0;
67
68 // Wait for transmission to complete
69 wait(tx_done);
70 $stop;
71
72 // Test case 2: Even parity, 4800 baud, 7 data bits, 2 stop bits
73 #100;
74 parity_type = 2'b10; // Even parity
75 baud_rate = 2'b01; // 4800 baud
76 data_length = 1'b0; // 7 data bits
77 stop_bits = 1'b1; // 2 stop bits
78 data_in = 8'b1100110; // New data to be transmitted
79
80 #100;
81 send = 1'b1;
82 #20;
83 send = 1'b0;
84
85 // Wait for transmission to complete
86 wait(tx_done);
87 $stop;
88
89 // Test case 3: Odd parity, 2400 baud, 8 data bits, 1 stop bit
90 #100;
91 parity_type = 2'b01; // Odd parity
92 baud_rate = 2'b00; // 2400 baud
93 data_length = 1'b1; // 8 data bits
94 stop_bits = 1'b0; // 1 stop bit
95 data_in = 8'b01101001; // New data to be transmitted
96
97 #100;
98 send = 1'b1;
99 #20;
100 send = 1'b0;
101
102 // Wait for transmission to complete
103 wait(tx_done);
104 $stop;
105
106
107 // Test case 4: Use p_parity_out, 19.2K baud, 8 data bits, 2 stop bits
108 #100;
109 parity_type = 2'b11; // Use p_parity_out
110 baud_rate = 2'b11; // 19.2K baud
111 data_length = 1'b1; // 8 data bits
112 stop_bits = 1'b1; // 2 stop bits
113 data_in = 8'b11110000; // New data to be transmitted
114
115 #100;
116 send = 1'b1;
117 #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