Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram

System on Programmable Chip Practice

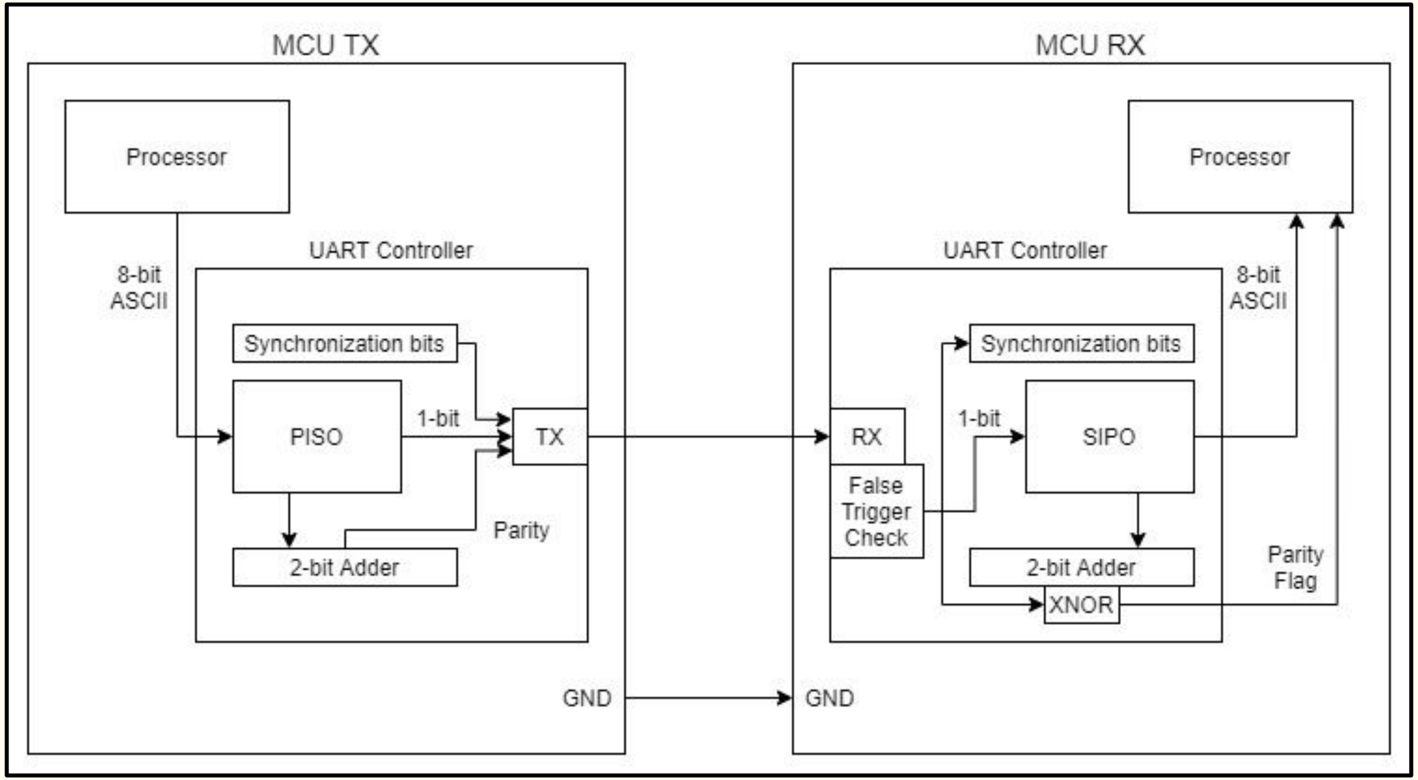
UART Communication Controller

Nehemiae G. Roy
EVD18I018

Outline

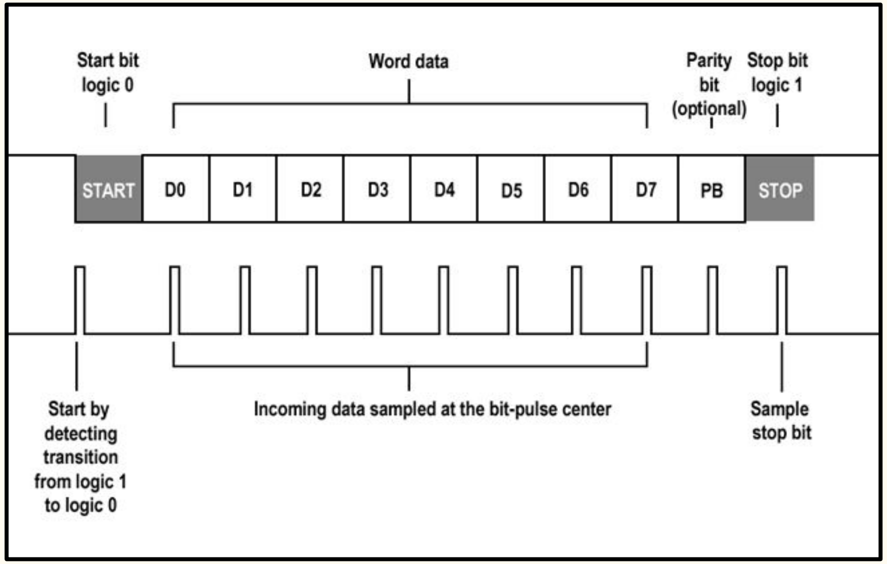
- Working Principle
- Requirement for UART
- Shift Registers
- Code
  - Transmitter Module
  - Receiver Module
  - Testbench
- Outputs
- Frequency Deviation Analysis
- Further Information
- References

Working Principle



Requirements for UART

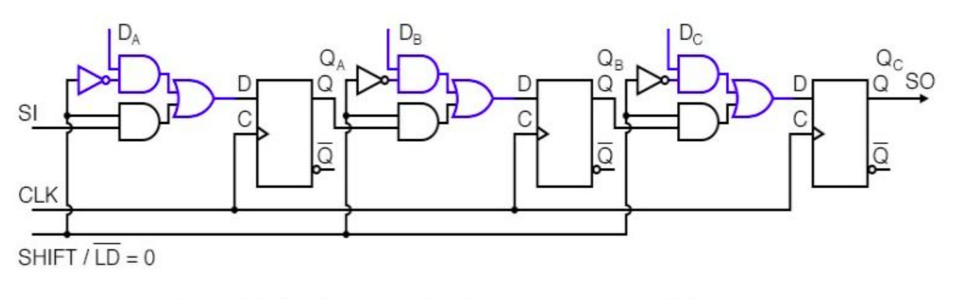
- Baud Rate
  - Baud rate generator
  - Typical speeds - 4200 bps, 9600 bps
  - Internal clock synchronization
  - Max. 10% deviation in baud rate
  - False Triggering
- Frame Structure
  - Synchronization bits
    - Idle state - HIGH
  - Data bits
  - Optional parity bit



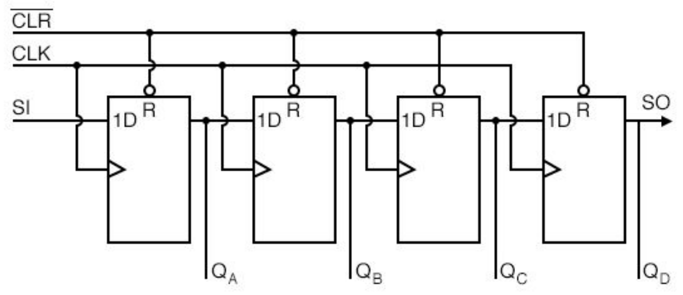
Frame Structure

Shift Registers

- Parallel-In Serial-Out Register
  - Load/Shift
  - LSB - Start Bit
  - 4 - bit word
  - Parity Bit
  - MSB - Stop Bit
- Serial-In Parallel-Out Register
  - Read parallel data after “Stages * Time Period”

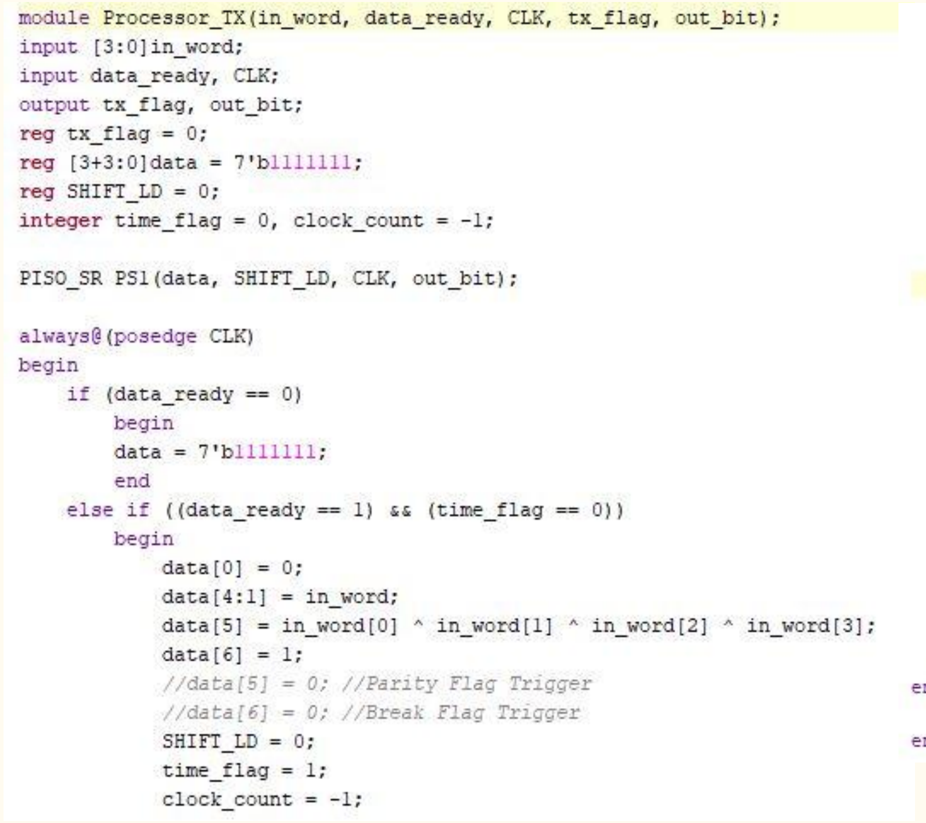


Parallel-in/ serial-out shift register showing parallel load path

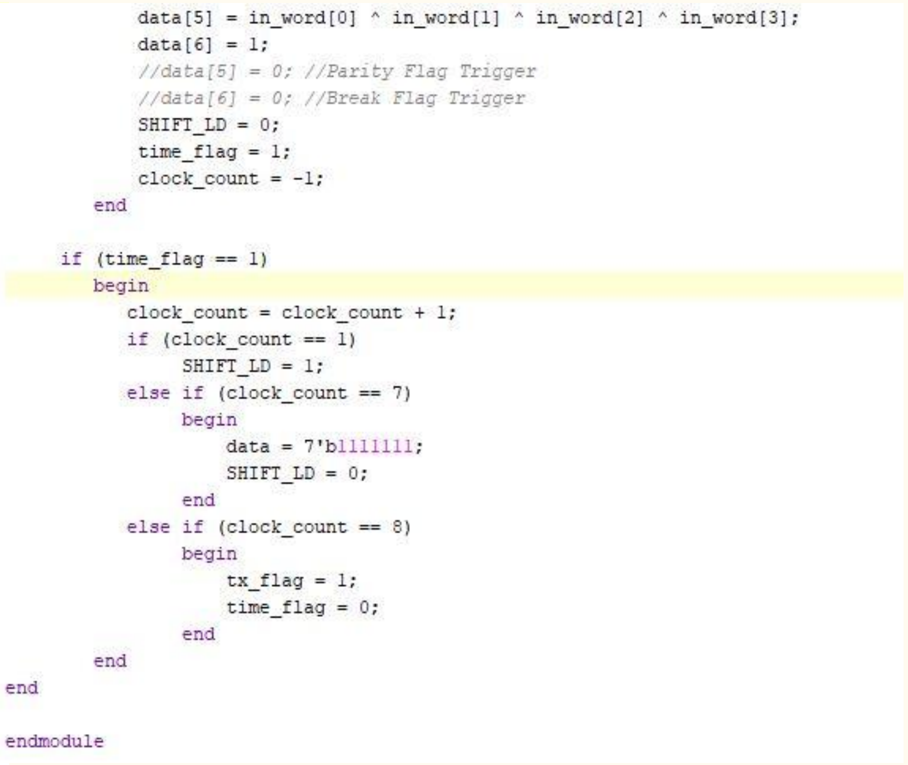


Serial-in/ Parallel-out shift register details

Transmitter Module

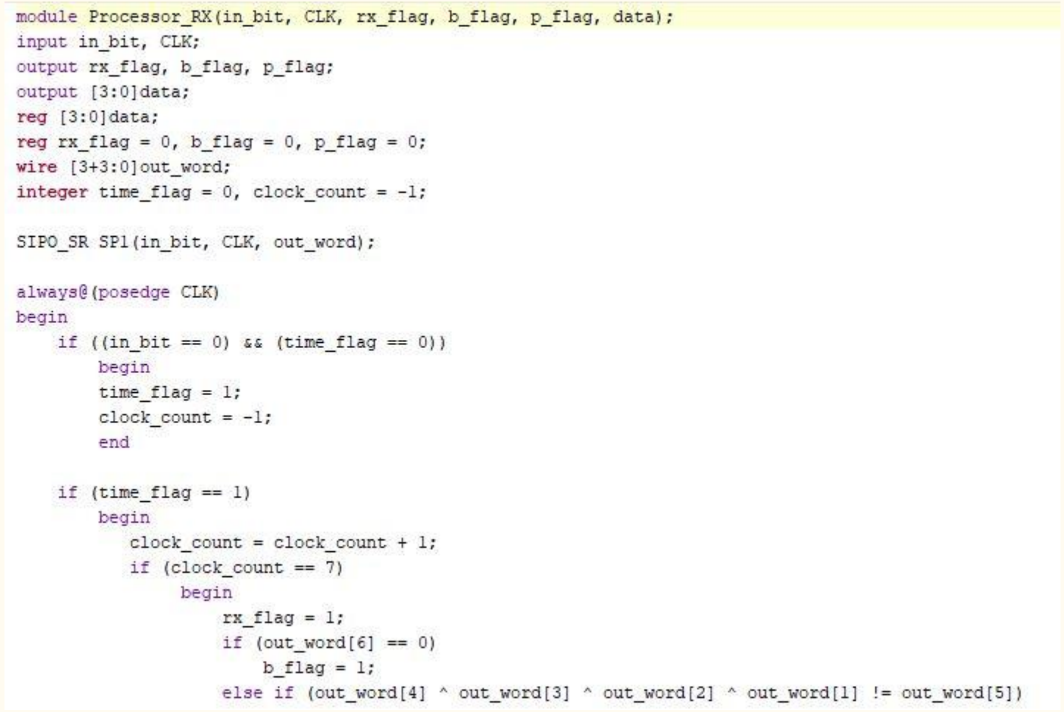


```
module Processor_TX(in_word, data_ready, CLK, tx_flag, out_bit);
input [3:0]in_word;
input data_ready, CLK;
output tx_flag, out_bit;
reg tx_flag = 0;
reg [3+3:0]data = 7'b1111111;
reg SHIFT_LD = 0;
integer time_flag = 0, clock_count = -1;

PISO_SR PS1(data, SHIFT_LD, CLK, out_bit);

always@(posedge CLK)
begin
    if (data_ready == 0)
        begin
        data = 7'b1111111;
        end
    else if ((data_ready == 1) && (time_flag == 0))
        begin
            data[0] = 0;
            data[4:1] = in_word;
            data[5] = in_word[0] ^ in_word[1] ^ in_word[2] ^ in_word[3];
            data[6] = 1;
            //data[5] = 0; //Parity Flag Trigger
            //data[6] = 0; //Break Flag Trigger
            SHIFT_LD = 0;
            time_flag = 1;
            clock_count = -1;
```



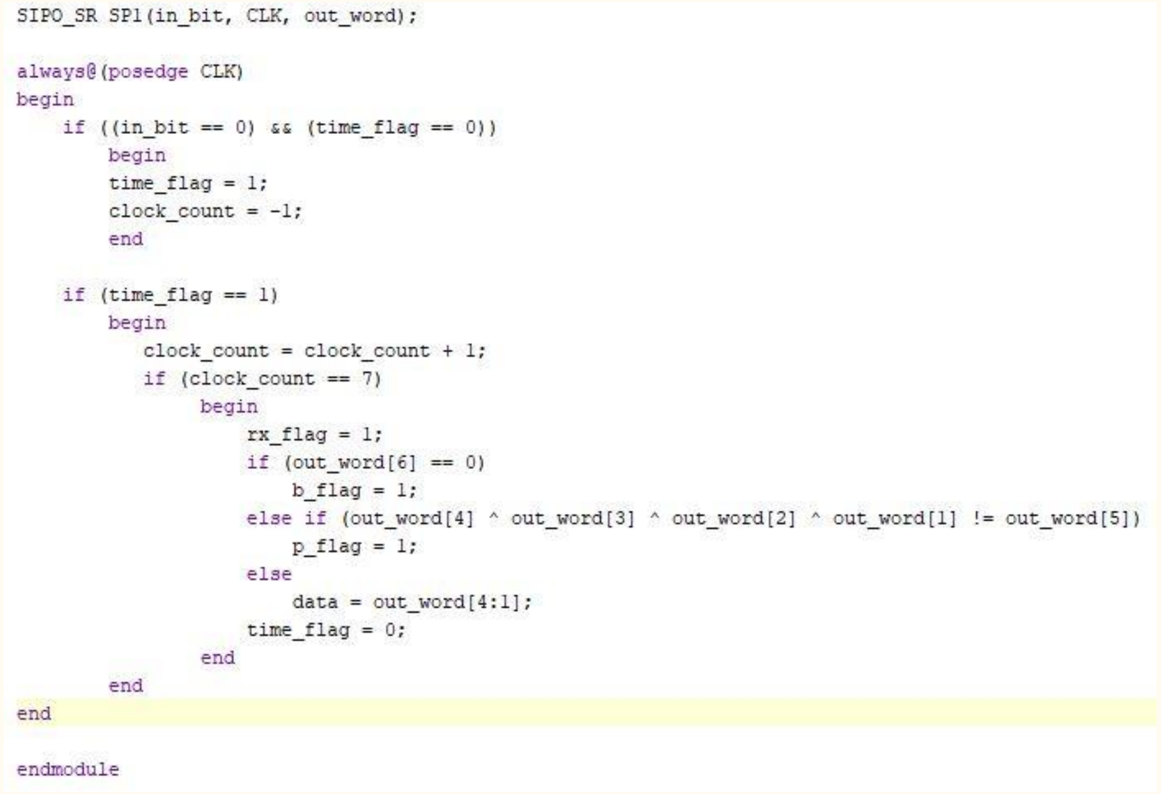
```
            data[5] = in_word[0] ^ in_word[1] ^ in_word[2] ^ in_word[3];
            data[6] = 1;
            //data[5] = 0; //Parity Flag Trigger
            //data[6] = 0; //Break Flag Trigger
            SHIFT_LD = 0;
            time_flag = 1;
            clock_count = -1;
        end

    if (time_flag == 1)
        begin
          clock_count = clock_count + 1;
          if (clock_count == 1)
              SHIFT_LD = 1;
          else if (clock_count == 7)
              begin
                  data = 7'b1111111;
                  SHIFT_LD = 0;
              end
          else if (clock_count == 8)
              begin
                  tx_flag = 1;
                  time_flag = 0;
              end
        end
end

endmodule
```

Receiver Module



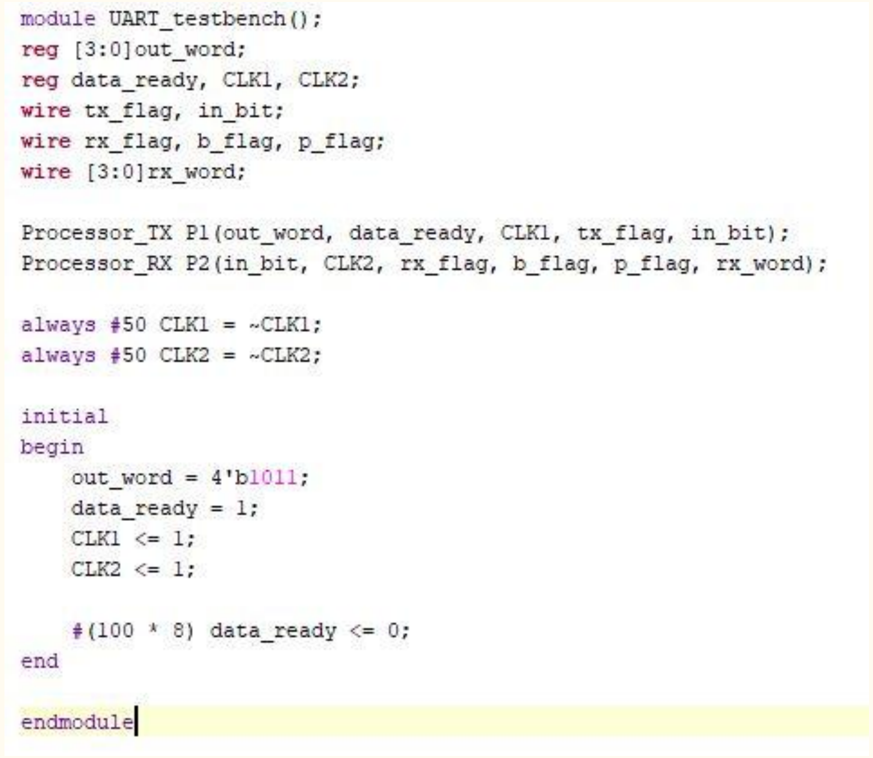
```
module Processor_RX(in_bit, CLK, rx_flag, b_flag, p_flag, data);
input in_bit, CLK;
output rx_flag, b_flag, p_flag;
output [3:0]data;
reg [3:0]data;
reg rx_flag = 0, b_flag = 0, p_flag = 0;
wire [3+3:0]out_word;
integer time_flag = 0, clock_count = -1;

SIPO_SR SP1(in_bit, CLK, out_word);

always@(posedge CLK)
begin
    if ((in_bit == 0) && (time_flag == 0))
        begin
        time_flag = 1;
        clock_count = -1;
        end

    if (time_flag == 1)
        begin
           clock_count = clock_count + 1;
           if (clock_count == 7)
                begin
                    rx_flag = 1;
                    if (out_word[6] == 0)
                        b_flag = 1;
                    else if (out_word[4] ^ out_word[3] ^ out_word[2] ^ out_word[1] != out_word[5])
```

Receiver Module



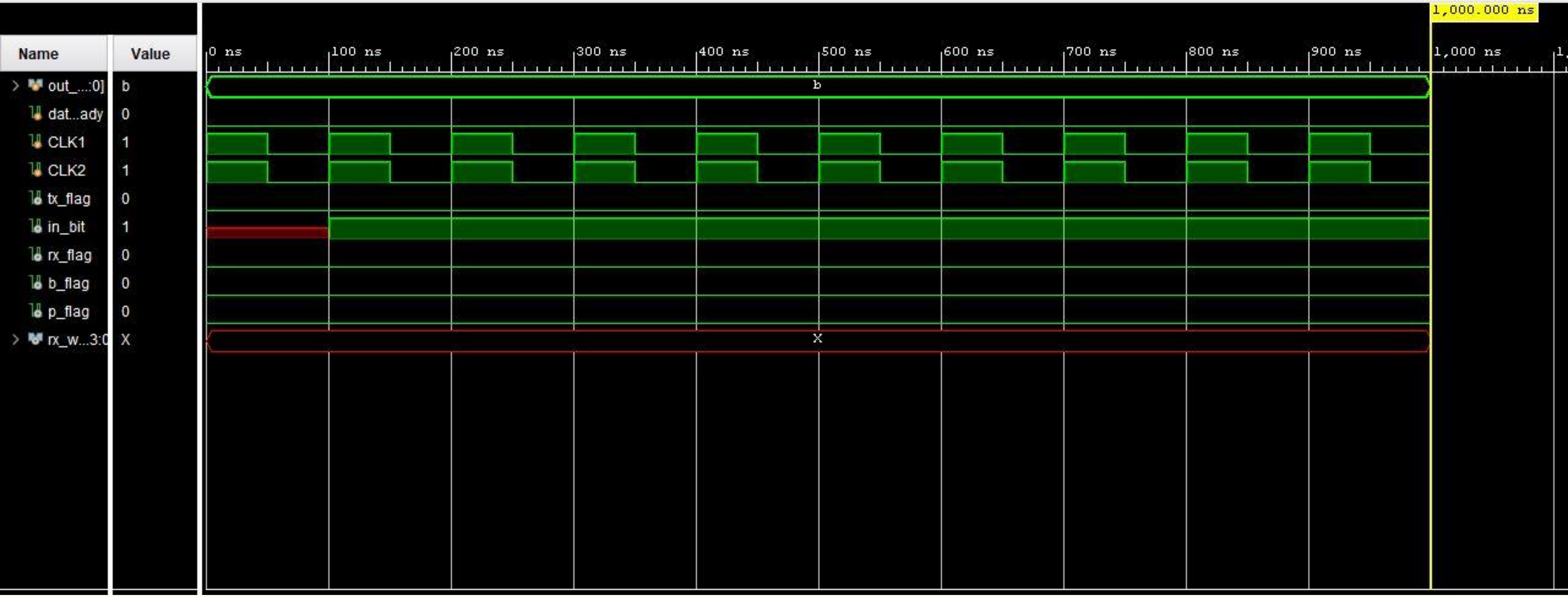
```
SIPO_SR SP1(in_bit, CLK, out_word);

always@(posedge CLK)
begin
    if ((in_bit == 0) && (time_flag == 0))
        begin
        time_flag = 1;
        clock_count = -1;
        end

    if (time_flag == 1)
        begin
           clock_count = clock_count + 1;
           if (clock_count == 7)
                begin
                    rx_flag = 1;
                    if (out_word[6] == 0)
                        b_flag = 1;
                    else if (out_word[4] ^ out_word[3] ^ out_word[2] ^ out_word[1] != out_word[5])
                        p_flag = 1;
                    else
                        data = out_word[4:1];
                    time_flag = 0;
                end
        end
end

endmodule
```

Testbench



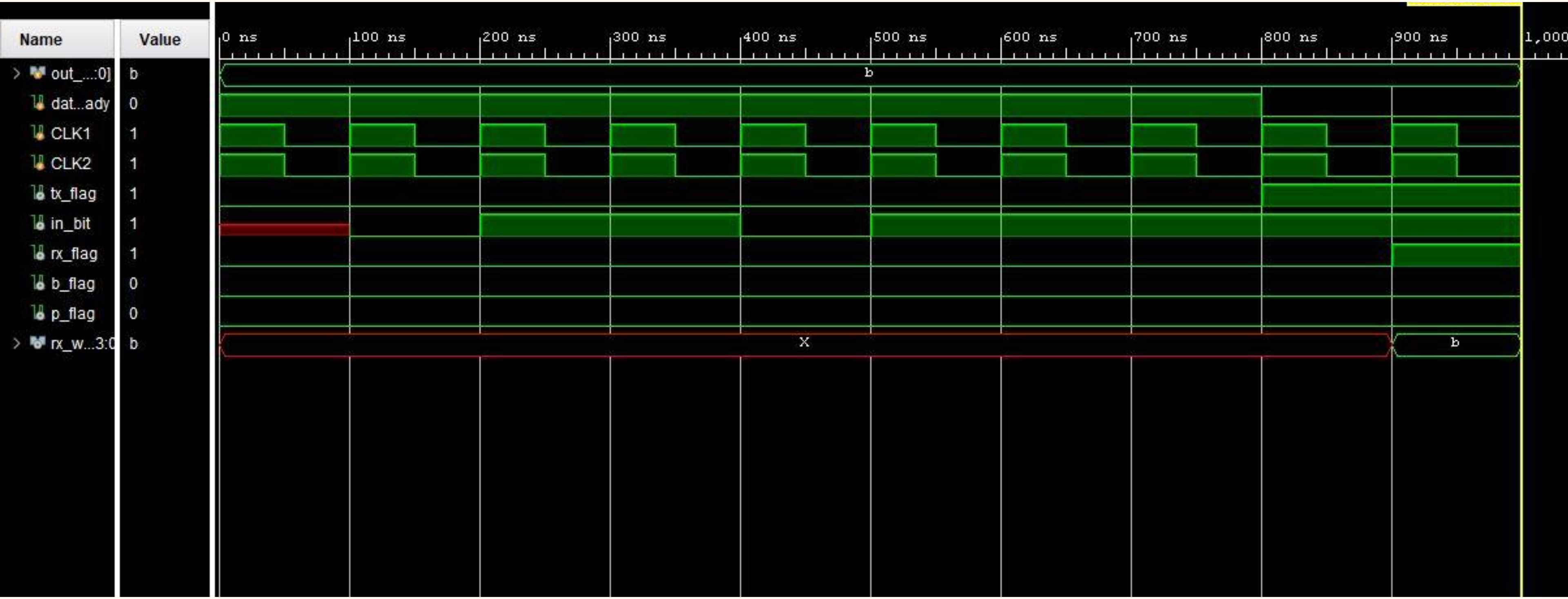
```
module UART_testbench();
reg [3:0]out_word;
reg data_ready, CLK1, CLK2;
wire tx_flag, in_bit;
wire rx_flag, b_flag, p_flag;
wire [3:0]rx_word;

Processor_TX P1(out_word, data_ready, CLK1, tx_flag, in_bit);
Processor_RX P2(in_bit, CLK2, rx_flag, b_flag, p_flag, rx_word);

always #50 CLK1 = ~CLK1;
always #50 CLK2 = ~CLK2;

initial
begin
    out_word = 4'b1011;
    data_ready = 1;
    CLK1 <= 1;
    CLK2 <= 1;

    #(100 * 8) data_ready <= 0;
end

endmodule
```

Idle State

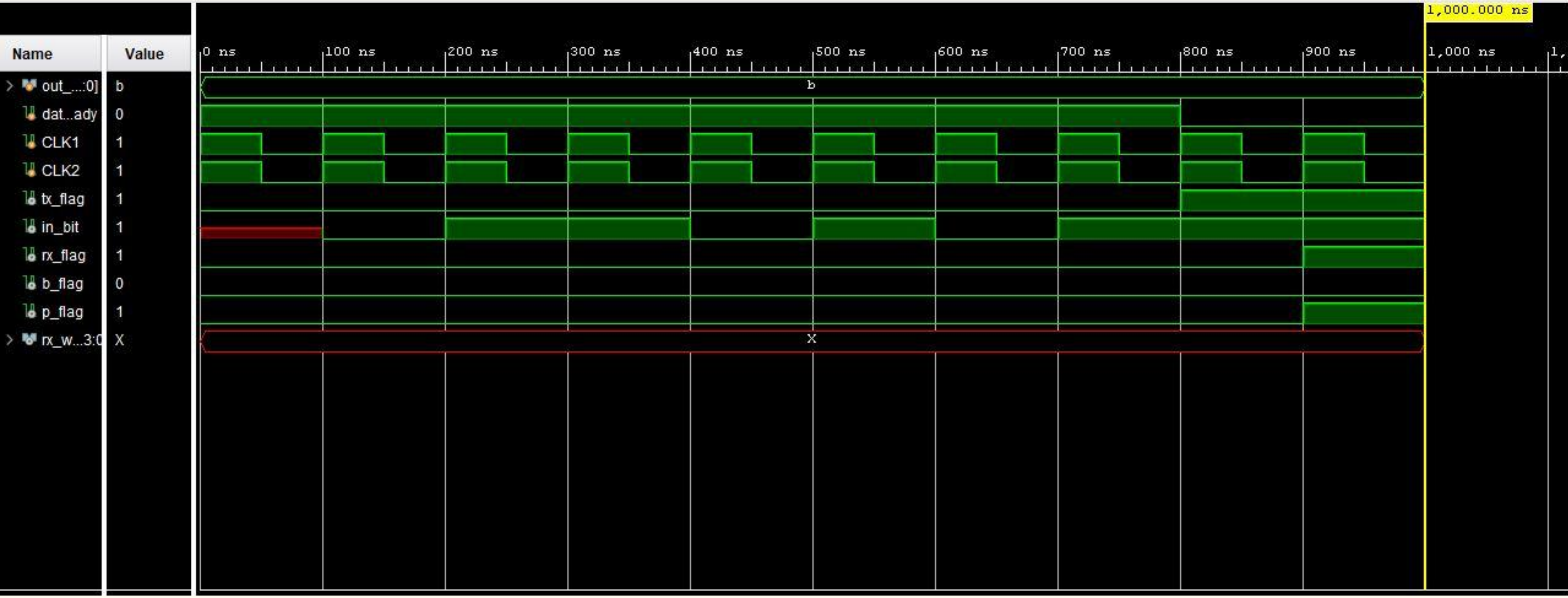


1,000.000 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
out_...:0]
b
dat...ady
0
CLK1
1
CLK2
1
tx_flag
0
in_bit
1
rx_flag
0
b_flag
0
p_flag
0
rx_w...3:0
X

Data Transmit

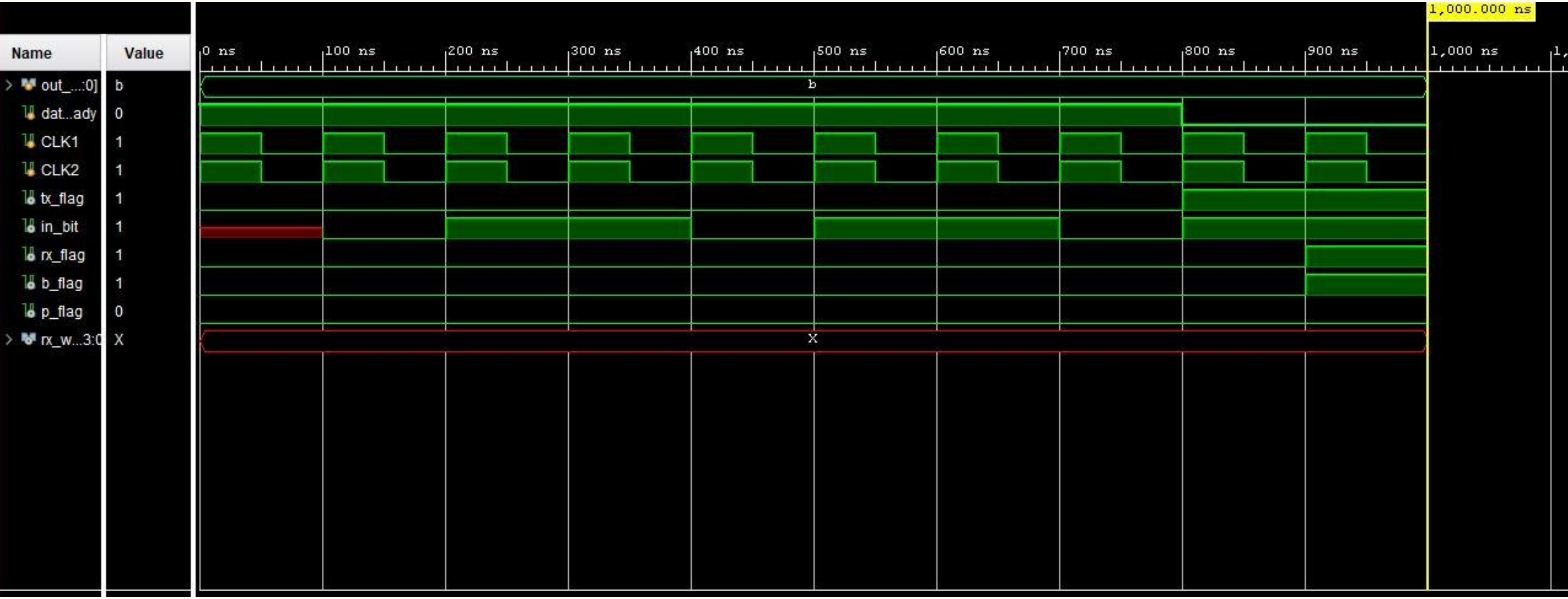


Parity Flag



Name
Value
out_...:0]
b
dat...ady
0
CLK1
1
CLK2
1
tx_flag
1
in_bit
1
rx_flag
1
b_flag
0
p_flag
1
rx_w...3:0
X
1,000.000 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
b
X

Break Flag



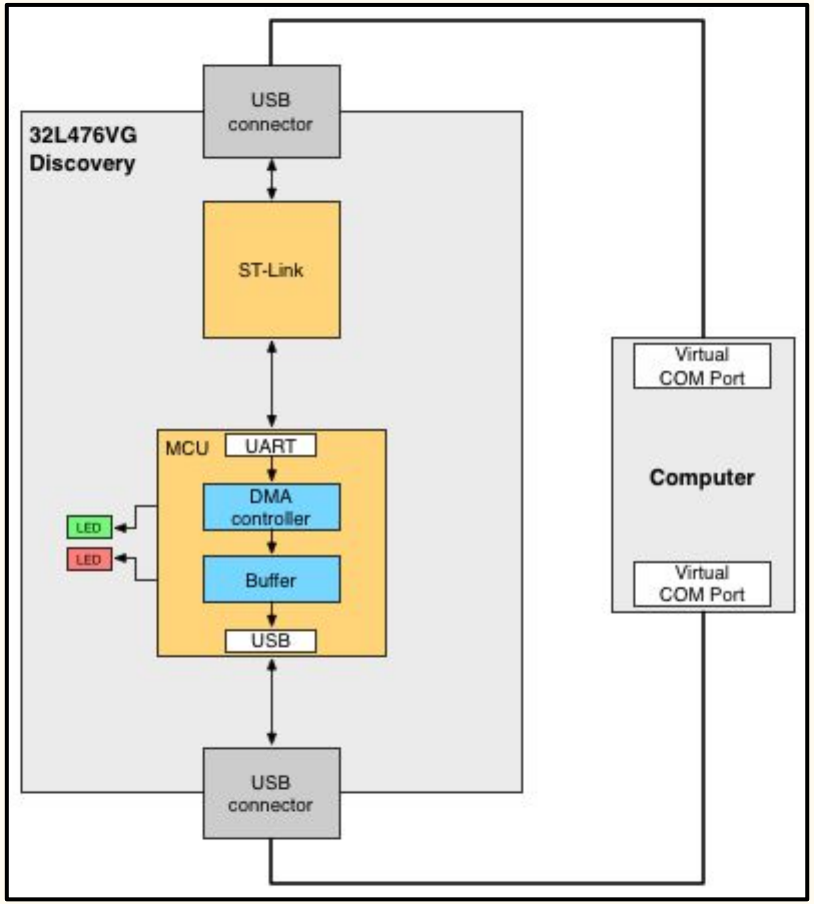
1,000.000 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
out_...:0] b
dat...ady 0
CLK1 1
CLK2 1
tx_flag 1
in_bit 1
rx_flag 1
b_flag 1
p_flag 0
rx_w...3:0 X
b
X

Frequency Deviation

Processor TX Time Period	Processor RX Time Period	Frequency Deviation	Status
100	100	0%	Correct
100	90	10%	Correct
100	88	12%	Correct
100	86	14%	Correct
100	84	16%	Correct
100	82	18%	Parity Error
100	80	20%	Incorrect Value

Further Information

- Inclusion of buffers
  - Overrun Error
- CPU Blocking
  - DMA
  - Peripheral-To-Memory
  - Double Buffering
- Applications
  - Serial Terminal
  - ASCII Characters
  - USB CDC (Virtual Com Port)



References

Websites:

1. “BASICS OF UART COMMUNICATION”, https://www.circuitbasics.com/basics-uart-communication/
2. “UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter”, https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html
3. “Universal Asynchronous Receiver-Transmitter”, https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

Videos:

1. “Understanding UART”, https://www.youtube.com/watch?v=sTHckUyxwp8&ab_channel=RohdeSchwarz
2. “SparkFun According to Pete 9-17-12: Serial Communication Demystified”, https://www.youtube.com/watch?v=JJZOTtwpAjA&ab_channel=SparkFunElectronics

References

Websites:

1. “The D-Type Flip-Flop”, https://www.electronics-tutorials.ws/sequential/seq_4.html
2. “Sequential Logic Circuits”, https://www.electronics-tutorials.ws/sequential/seq_1.html
3. “D Flip-Flop”, https://www.javatpoint.com/verilog-d-flip-flop#:~:text=D%20flip%20flop%20is%20an,the%20edge%20triggered%20always%20statements.
4. “Shift Registers: Serial-in, Parallel-out (SIPO) Conversion”, https://www.allaboutcircuits.com/textbook/digital/chpt-12/serial-in-parallel-out-shift-register/
5. “Shift Registers: Parallel-in, Serial-out (PISO) Conversion”, https://www.allaboutcircuits.com/textbook/digital/chpt-12/parallel-in-serial-out-shift-register/

Research Paper:

1. “A Review Paper on Design and Simulation of UART for Serial Communication”, Vibhu Chinmay, Shubham Sachdeva, IJIRT

THANK YOU