VHDL Report

Serial Data Receiver

الاسم	الرقم الجامعي
عمر ياسر كمال محمد حسين	**20011035**
عمر حسنى احمد	**20010987**
عمر احمد عبد العزيز عبد العزيز	**20010971**
عمر محمد أحمد محمد السيد	**20011013**
ياسر محمد عبد الجواد إبراهيم	**20012206**
مهند محمد علي محمد	**20012033**

Project 3: Serial Data Receiver:

The diagram of a serial data receiver is shown below. It contains a serial data input, din, and a parallel data output, data(6:0). A clock signal is also needed at the input. Two supervision signals are generated by the circuit: err (error) and data_valid. When no bits are received, the din is '0'. The input data train consists of ten bits. A new bit is received at the rising edge of the clock. The first bit is a start bit, which, when high, must cause the circuit to start receiving data. The next seven are the actual data bits. The ninth bit is a parity bit, whose status must be '0' if the number of ones in data is even or '1' otherwise. Finally, the tenth is a stop bit, which must be high if the transmission is correct. An error is detected when either the parity does not check, or the stop bit is not a '1'. When reception is finished and if no error has been detected, then the data stored in the internal registers is transferred to the output data(6:0) and the output data_valid output is '1' .

The steps of Design:

1. Make a simplify for design to easier its implementation
2. design for serial input parallel output register(SIPO)
3. define the i/p and o/p of design
4. check if we need signal or not
5. design the functions of problem
6. design the mode of operation

-Assumptions: when we start receiving data, we set parameters:

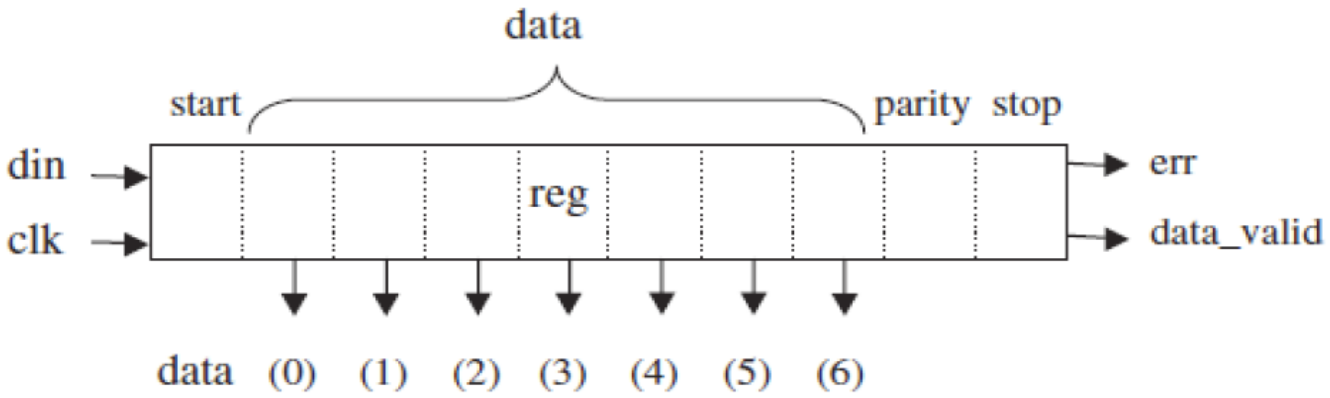
-data_valid→'0'

-err→'0'

-data(output) remains the same.

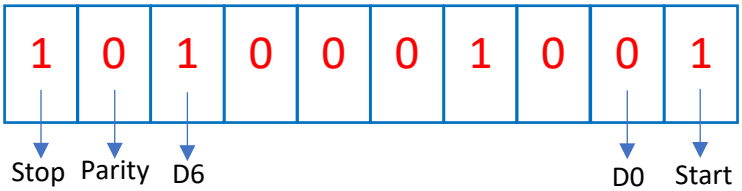
1.Make a simplify for design to easier its implementation

It is the original design of problem

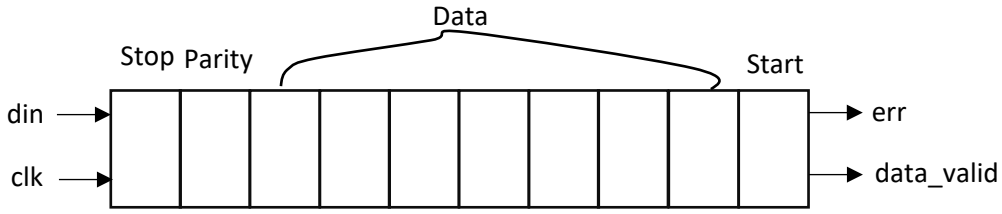


But this design has a complex step to understand in shift register that when we will send the bits like that 1010001001 these bits will store as the following diagram

so, we reverse this design to make it easy to understand when implementation

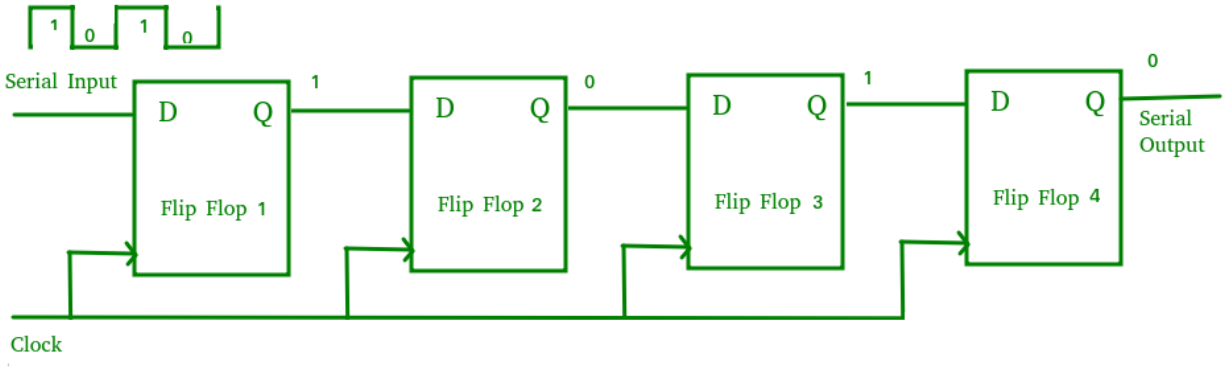


and check the first bit come that at the end of process store at start to make sure that the program run or not.



2.design for serial input parallel output register(SIPO)

to design this register we need to 10 flip-flop and connect like that

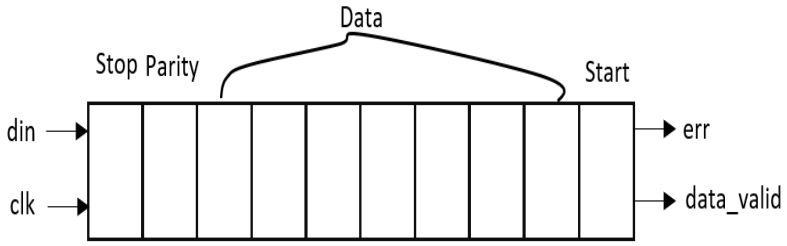


And we control to receive data or not by checking the first bit arriving to register

---

3.define the i/p and o/p of design

We have 2 inputs din and clk ,and have 2 outputs err and data_valid



4.check if we need signal or not

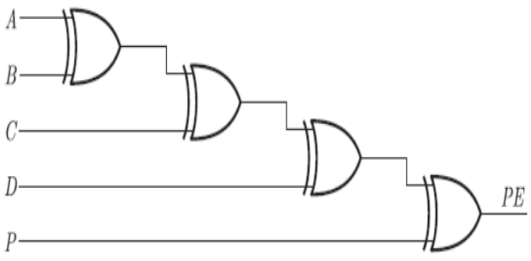
from the register diagram we find that we need signals between flipflop

---

5.design the functions of problem

We need to design a parity checker like the following diagram

Parity= A ⊕ B ⊕ C ⊕ D ⊕ ……..

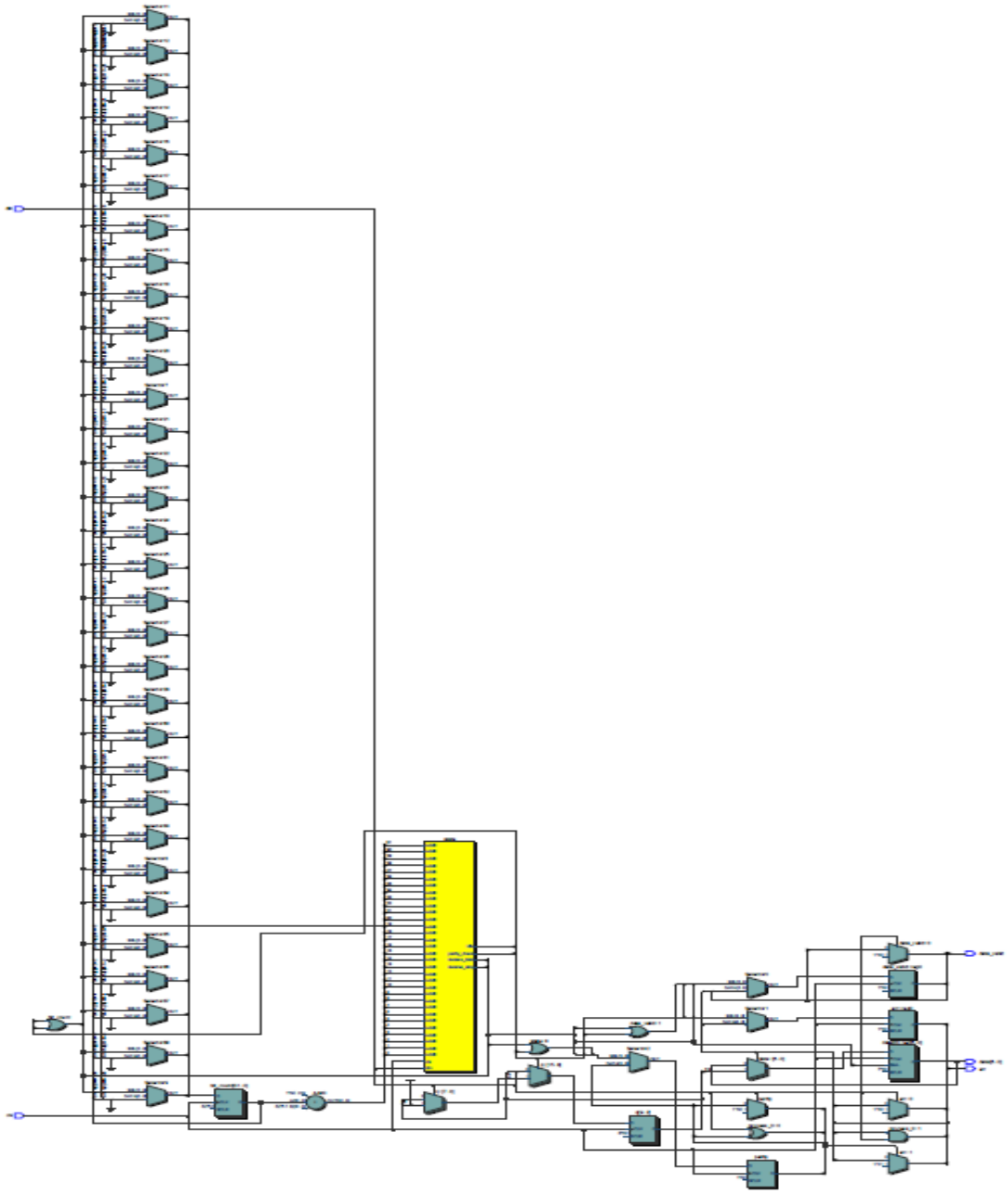


6.design the mode of operation

We make 3 mod of operation

1. idle: that is mean the receiver doesn't receive any data (start bit is 'ZERO ')
2. receive_data: that is mean the receiver starts to receive the data(7bits) (start bit is 'ONE')
3. parity_check: check if the parity(we have calculated) is equal to parity bit or not.
4. receive_stop: that means the receiver receives 9 bits, now it is receiving the last bit (stop bit) and when it receives last bit the state becomes idle again.

The schematic of the system



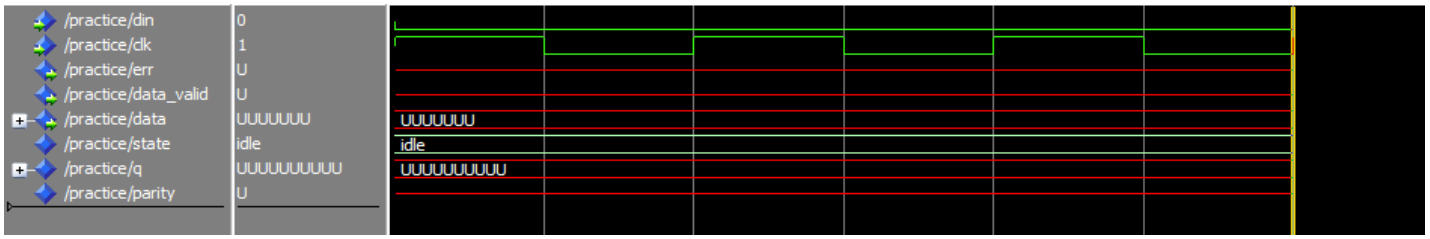
VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity practice is
  port(
    din, clk: in std_logic;
    data_valid: out std_logic;
    err:buffer std_logic;
    data: out std_logic_vector(6 downto 0)
  );
end practice;

architecture behavior of practice is
  type stateType is (idle, receive_data,parity_check ,receive_stop);
  signal state: stateType := idle;
  signal q: std_logic_vector(9 downto 0);
  signal parity: std_logic;

  begin
    process (clk)
    variable bit_count: integer := 0;
    begin
    if rising_edge(clk) then
      case state is
        when idle =>
          if din = '1' then
            data_valid <= '0';
            err <= '0';
            parity <= '0';
            q <= din & q(9 downto 1);
            state <= receive_data;
          end if;
        when receive_data =>
          parity <= parity xor din;
          q <= din & q(9 downto 1);
          bit_count := bit_count + 1;
          if bit_count = 7 then
            state <= parity_check;
          end if;
        when parity_check =>
          if parity /= din then
              err <= '1';
          end if;
          q <= din & q(9 downto 1);
          state <= receive_stop;
          when receive_stop =>
            if (err = '0') and (din = '1') then
                data <= q(8 downto 2);
                data_valid <= '1';
                err <= '0';
            else
                data_valid <= '0';
                err <= '1';
            end if;
            q <= din & q(9 downto 1);
            state <= idle;
            bit_count := 0;
      end case;
    end if;
  end process;

end behavior;
```

The simulation results

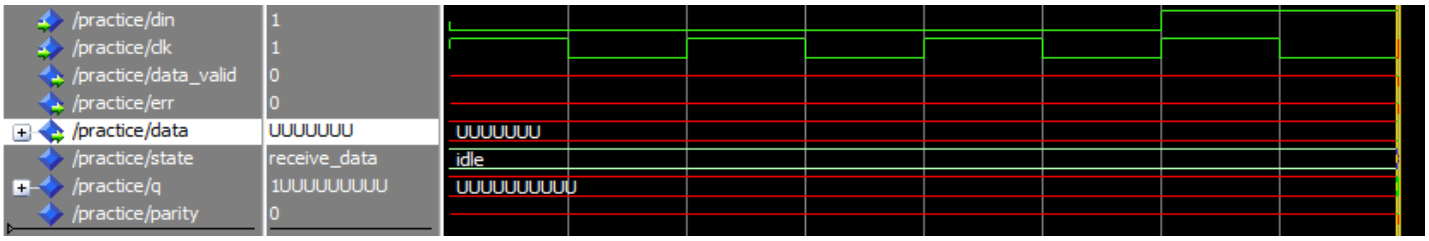
We will transmit “1010001001” but we will try to transmit ‘0’ at first to see if the receiver work or will wait to din to be ‘1’

-din→0



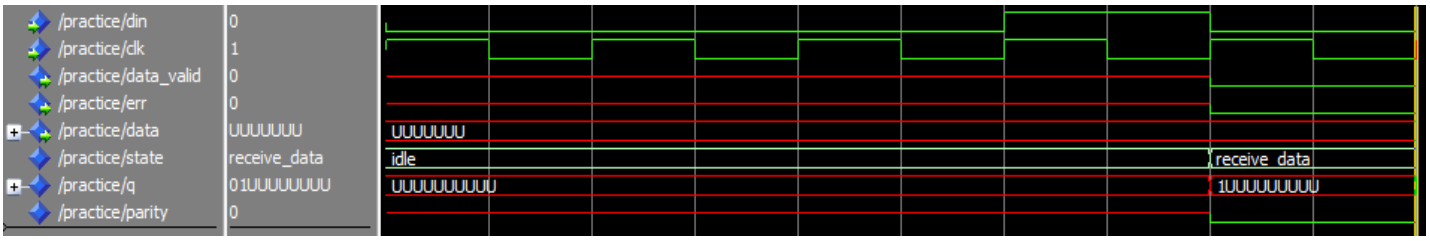
So, we will see that the receiver did not work, state still “idle” and all q values are still “UUUUUUUUUU”

-din→1 (start bit)



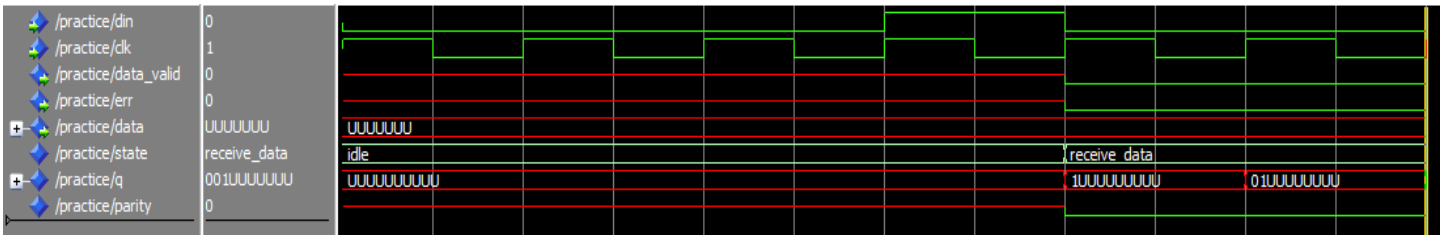
The receiver starts to work when din = 1, state → receive_data and q→ “1UUUUUUUUU”

-din→0 (D0)



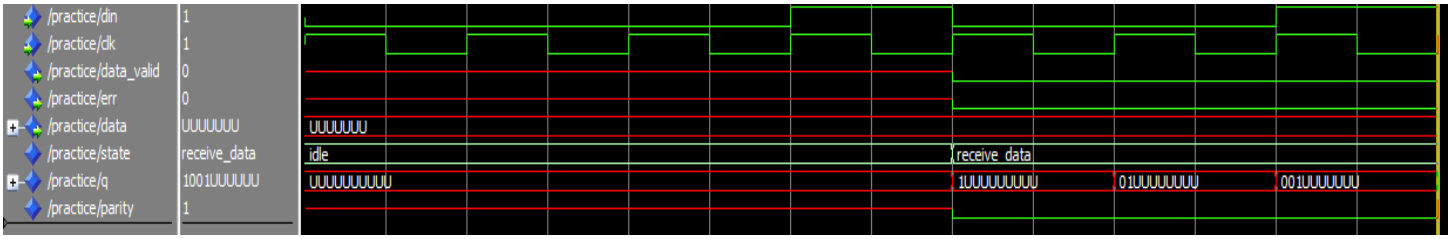
state still receive_data and q→ “01UUUUUUUU”

-din→0



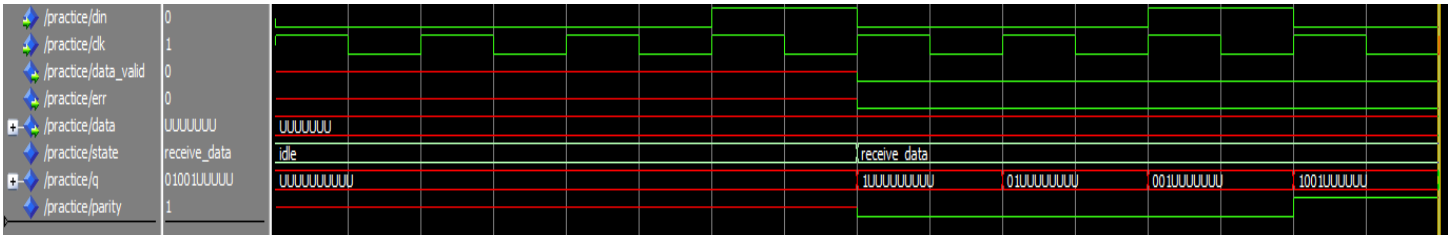
state still receive_data and q→ “001UUUUUUU”

-din→1



state still receive_data and q→ “1001UUUUUUU”

-din→0

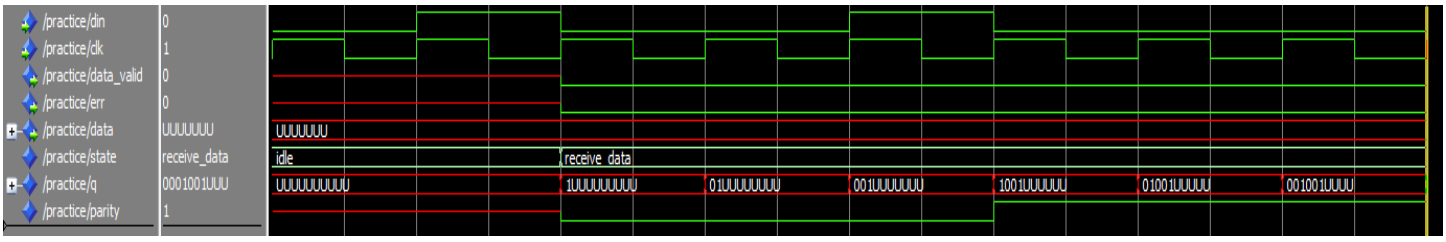


state still receive_data and q→ “01001UUUUUU”

-din→0

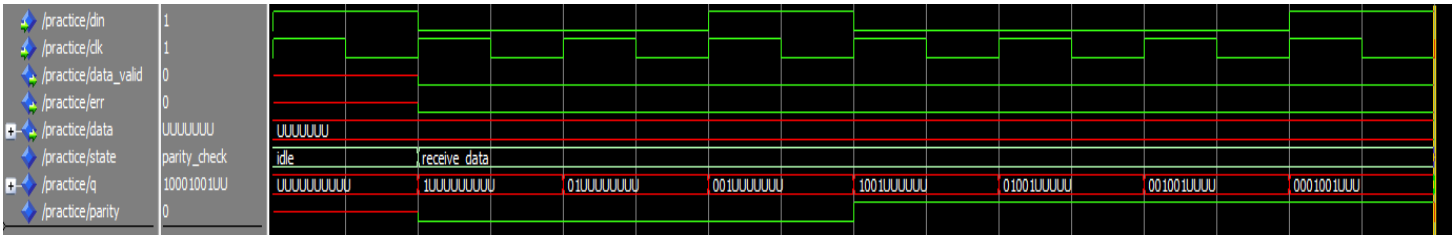
state still receive_data and q→ “001001UUUUU”

-din→0



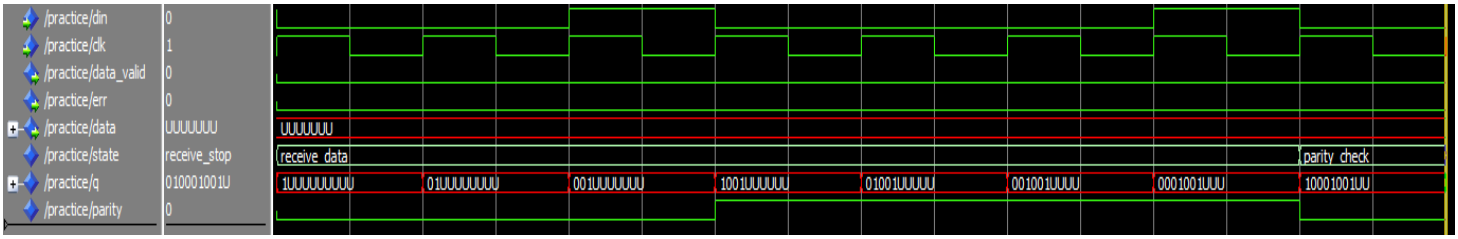
state still receive_data and q→ “0001001UUU”

-din→1 (D6)



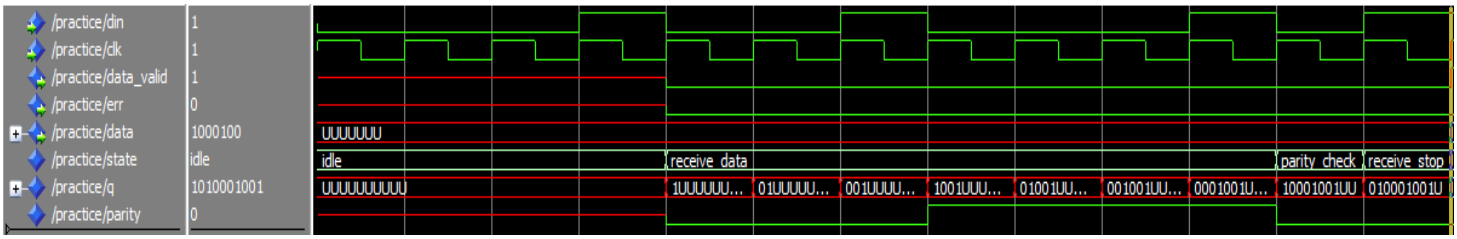
state → parity_check, parity will be calculated → ‘0’ and q→ “10001001UU”

-din→0 (parity bit)



state → receive_stop, and q→ “010001001U”

-din→1 (stop bit)



state → idle, parity will be calculated → ‘0’, q→ “1010001001”, data_valid→’1’ and err→’0’

err→’0’ and stop bit→1 so the data→”1000100”