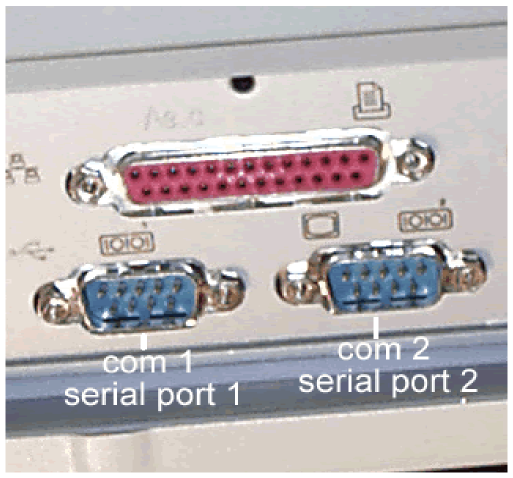
Serial Port

By

Emad Samuel

Serial Port

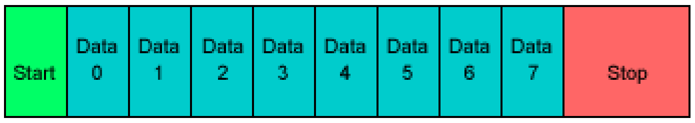


Serial port is a serial communication physical interface through which information transfers in or out one bit at a time.

Data transfer through serial ports connected the computer to devices such as terminals and various peripherals.

Some computers, such as the IBM PC, used an integrated circuit called a UART, that converted characters to (and from) asynchronous serial form, and automatically looked after the timing and framing of data.

A universal asynchronous receiver/transmitter (usually abbreviated UART) is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms.



Baudrate:

In embedded designs, it is necessary to choose a proper oscillator to get the correct baud rate with little or no error. Some examples of common crystal frequencies and baud rates with no errors are:

300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 115200 Bd

Data bits:

The number of data bits in each character can be 5 (for Baudot code), 6 (rarely used), 7 (for true ASCII), 8 (for any kind of data, as this matches the size of a byte), or 9 (rarely used). 8 data bits are almost universally used in newer applications. 5 or 7 bits generally only make sense with older equipment such as teleprinters.

Most serial communications designs send the data bits within each byte LSB (Least Significant Bit) first. This standard is also referred to as "little endian". Also, possible, but rarely used, is "big endian" or MSB (Most Significant Bit) first serial communications. (See Endianness for more about bit ordering.) The order of bits is not usually configurable, but data can be byte-swapped only before sending.

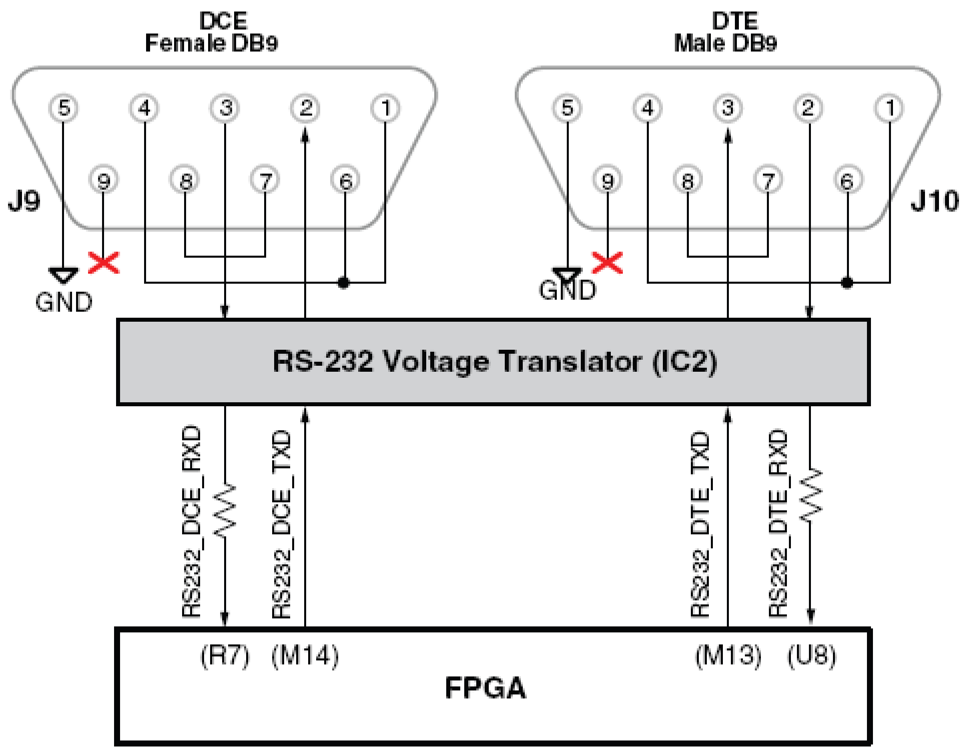
Parity:

Parity is a method of detecting errors in transmission. When parity is used with a serial port, an extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1's, then it must have been corrupted. However, an even number of errors can pass the parity check.

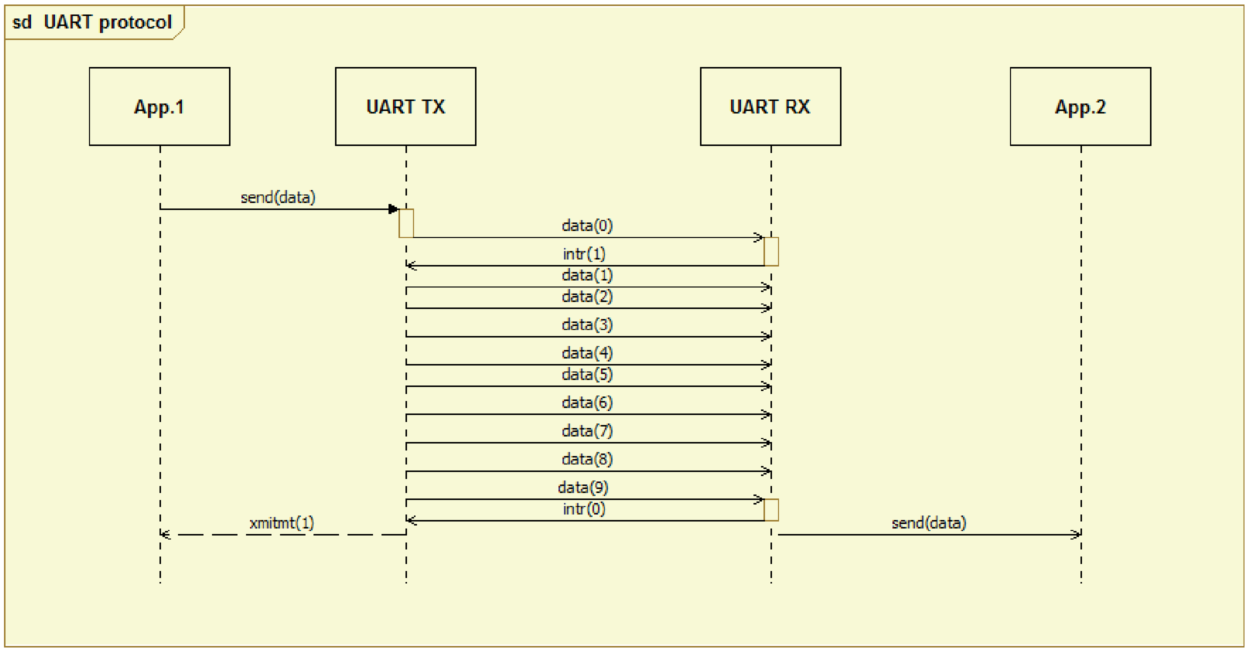
Stop bits

Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. If slow electromechanical teleprinters are used, one-and-one half or two stop bits are required.

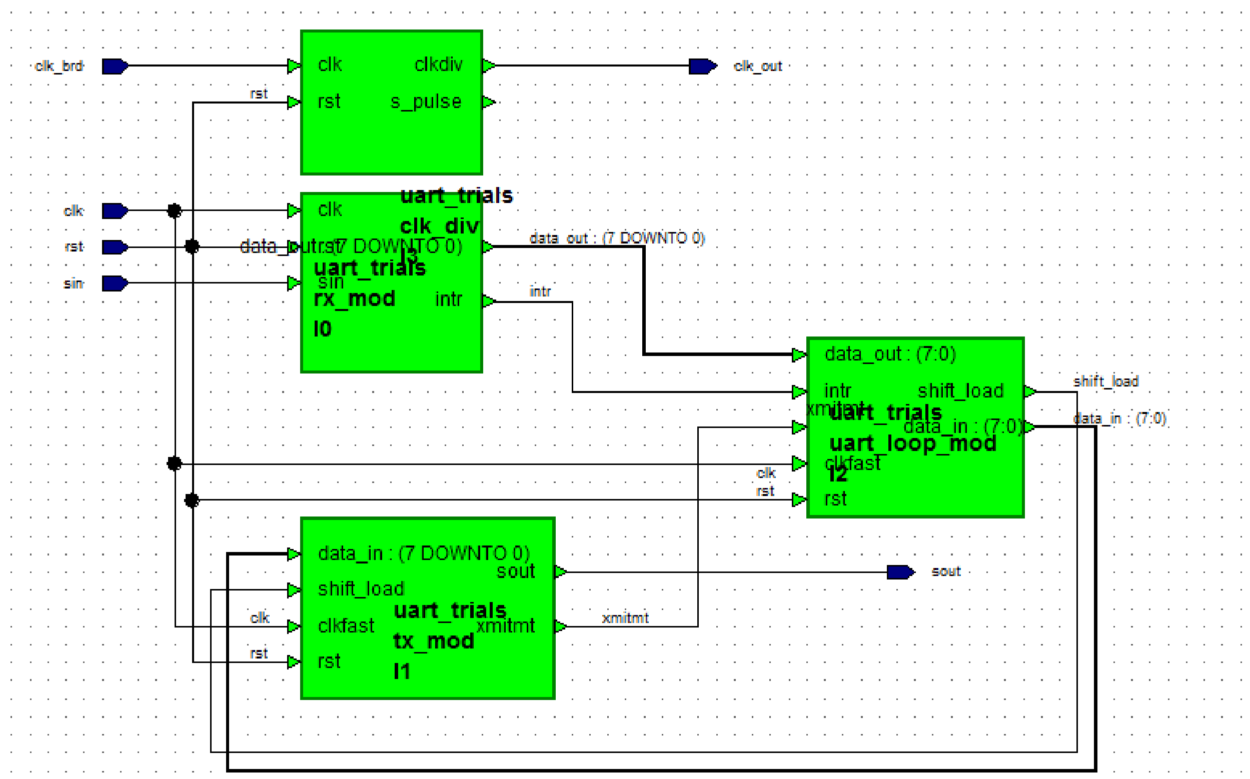
Structure of Serial port:



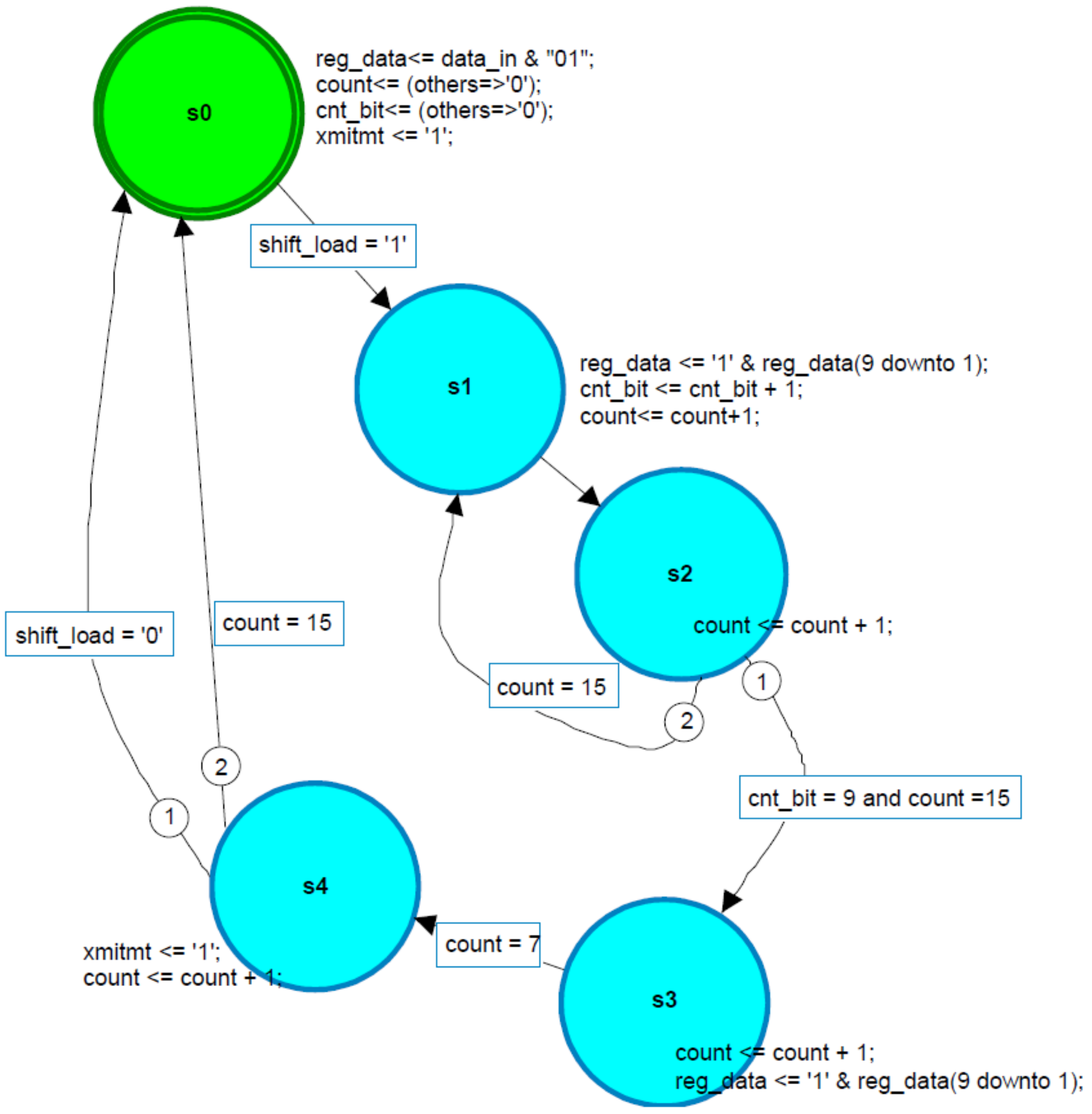
UART protocol:



VHDL Codes for serial port test circuit: RX data from PC=> loop => Tx the same data.

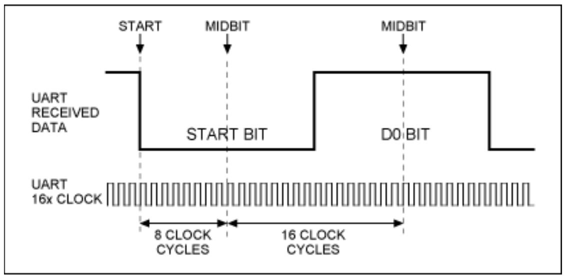


TX FSM:


s0
reg_data<= data_in & "01";
count<= (others=>'0');
cnt_bit<= (others=>'0');
xmitmt <= '1';
shift_load = '1'
s1
reg_data <= '1' & reg_data(9 downto 1);
cnt_bit <= cnt_bit + 1;
count<= count+1;
s2
count <= count + 1;
count = 15
1
2
cnt_bit = 9 and count =15
s3
count <= count + 1;
reg_data <= '1' & reg_data(9 downto 1);
count = 7
s4
xmitmt <= '1';
count <= count + 1;
shift_load = '0'
count = 15
1
2

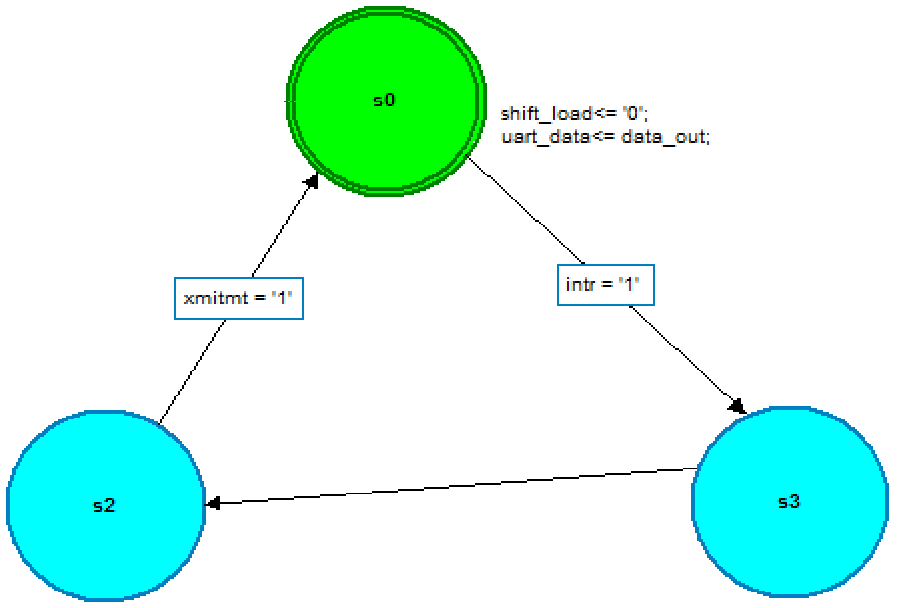
RX Code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ENTITY rx_mod IS
   PORT(
      clk      : IN     std_logic;
      rst      : IN     std_logic;
      sin      : IN     std_logic;
      data_out : OUT    std_logic_vector (7 DOWNTO 0);
      intr     : OUT    std_logic);

END rx_mod ;

-- hds interface_end
ARCHITECTURE rtl OF rx_mod IS
    signal rxreg: std_logic_vector(9 downto 0);
    signal count: unsigned (3 downto 0);
    signal rxmt: std_logic;
    signal rxin,start_flag: std_logic;
    begin
        process (clk, rst)
            begin
                if (rst = '0') then
                    count <= (others => '0');
                    rxmt <= '1';
                    rxreg <= (others => '1');
                    intr <= '0';
                    rxin <= '1';
                    start_flag<='0';
                elsif (rising_edge(clk)) then
                    rxin<=sin;
                    if (rxmt = '1' and rxin = '0') then
                        count <= (others => '0');
                        rxmt <= '0';
                        rxreg <= (others => '1');
                        start_flag<='0';
                    elsif (count = 7 and rxmt = '0' and rxin = '0' and start_flag='0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= (others => '0');
                        start_flag<='1';
                    elsif (count = 15 and rxmt = '0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= count + 1;
                    else
                        count <= count + 1;
                    end if;
                    if (rxmt = '0' and rxreg(9) = '1' and rxreg(0) = '0') then
                        intr <= '1';
                        rxmt <= '1';
                    else
                        intr <= '0';
                    end if;
                end if;
        end process;
        data_out <= rxreg(8 downto 1);
END rtl;
```



Loop:

data_in <= uart_data;



Clock divider:

In order to use the UART you need to know what baud rate you want to transmit at. The transmitter and receiver modules have a clock divider (see block diagram), which runs 16 times slower than the clock signal sent to it.

If for example, you want to transmit at 19.2 kbps and the FPGA board runs at 125 MHz then:

Baud rate x 16 = 19200 x 16 = 307200
Clock division ratio = 125000000 / 307200 = 406

***Extra read: Chapter 11, Book: VHDL Coding Styles and Methodologies by Ben Cohen**