UART

Universal Asynchronous Receiver and Transmitter

A serial communication protocol that sends parallel data through a serial line.

Typically used with RS-232 standard.

Your FPGA boards have an RS-232 port with a standard 9-pin connector.

The voltages of the FPGA and serial port are different, and therefore a level-converter circuit is also present on the board.

The board handles the RS-232 standard and therefore our focus is on the UART.

The UART includes both a transmitter and receiver.

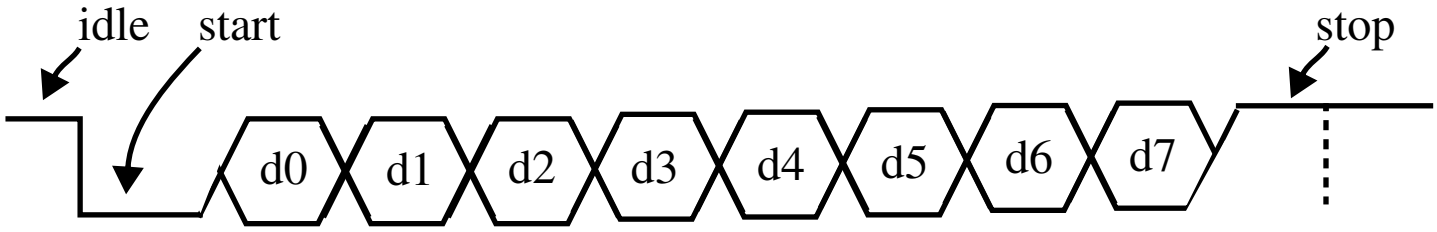
The transmitter is a special **shift** register that loads data in parallel and then shifts it out bit-by-bit.

The receiver shifts in data bit-by-bit and reassembles the data byte.

The data line is '1' when idle.

UART Spec

Transmission starts when a *start bit* (a '0') is sent, followed by a number of *data bits* (either 6, 7 or 8), an optional *partity bit* and *stop bits* (with 1, 1.5 or 2 '1's).



This is the transmission of 8 data bits and 1 stop bit.

Note that **no clk** signal is sent through the serial line.

This requires agreement on the transmission parameters by both the transmitter and receiver in advance.

This information includes the **band rate** (number of bits per second), the number of data bits and stop bits, and whether parity is being used.

Common baud rates are 2400, 4800, 9600 and 19,200.

UART Receiving Subsystem

An *oversampling scheme* is commonly used to locate the middle position of the transmitted bits, i.e., where the actual sample is taken.

The most common oversampling rate is **16** times the baud rate.

Therefore, each serial bit is sampled 16 times but only one sample is saved as we will see.

The oversampling scheme using *N* data bits and *M* stop bits:

- Wait until the incoming signal becomes '0' (the start bit) and then start the sampling **tick** cnter.
- When the cnter reaches 7, the incoming signal reaches the middle position of the *start* bit. Clear the cnter and restart.
- When the cnter reaches 15, we are at the middle of the first *data* bit. Retrieve it and shift into a register. Restart the cnter.
- Repeat the above step *N-1* times to retrieve the remaining *data* bits.
- If optional *parity* bit is used, repeat this step once more.
- Repeat this step *M* more times to obtain the stop bits.

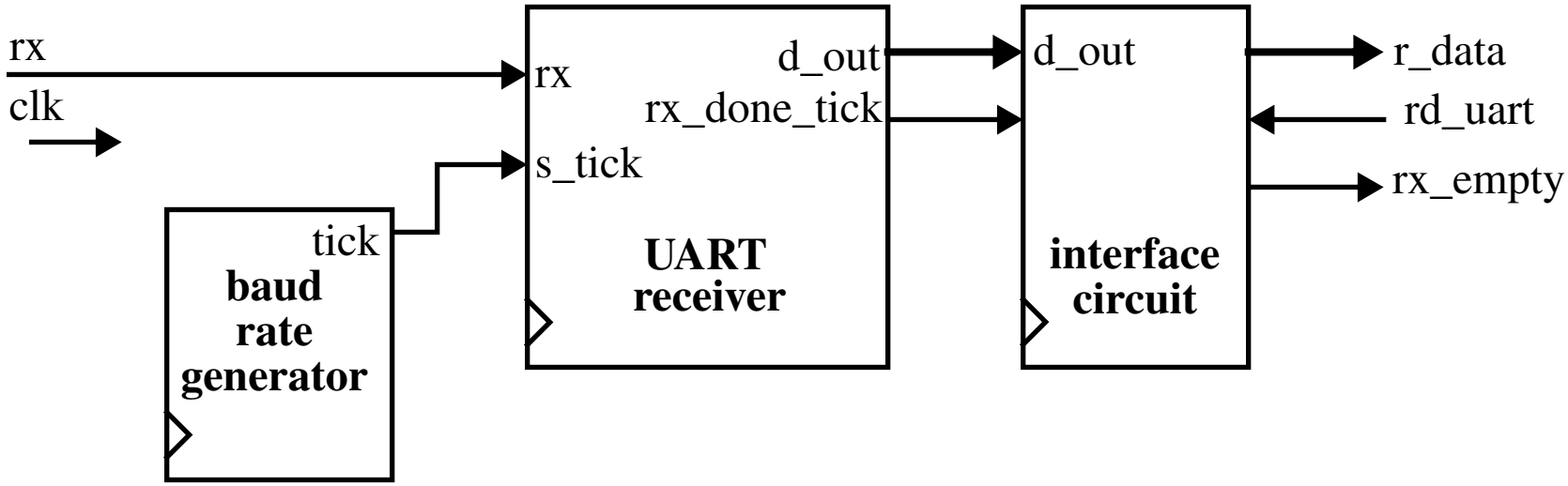
UART Receiving Subsystem

The oversampling scheme replaces the function of the clock.

Instead of using the rising edge to sample, the sampling ticks are used to estimate the center position of each bit.

Note that the system clock must be much faster than the baud rate for oversampling to be possible.

The receiver block diagram consists of three components



The interface circuit provides a buffer and status between the UART and the computer or FPGA.

UART Receiving Subsystem

The baud rate generator generates a sampling signal whose frequency is exactly **16** times the UART's designated baud rate.

To avoid creating a new clock domain, the output of the baud rate generator will serve to enable ticks within the UART rather than serve **as** the clk signal.

The whole system will use **one** clk as we will see.

For a 19,200 baud rate, the sampling rate has to be 307,200 (19,200*16) ticks per second.

With a system clk at 50 MHz, the baud rate generator need a mod-163 cnter (50 MHz/307,200).

Therefore, the *tick* output will assert for one clk cycle every 163 clk cycles of the system clk.

The following code from page 83 of the text can be used to implement a mod-163 cnter.

UART Receiving Subsystem: mod-163 cnter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity mod_m_cnter is
  generic(
      N: integer := 4;
      M: integer := 10;
  );
  port(
      clk, reset: in std_logic;
      max_tick: out std_logic;
      q: out std_logic_vector(N-1 downto 0);
  );
  end mod_m_cnter;

architecture arch of mod_m_cnter is
  signal r_reg: unsigned(N-1 downto 0);
  signal r_next: unsigned(N-1 downto 0);
```

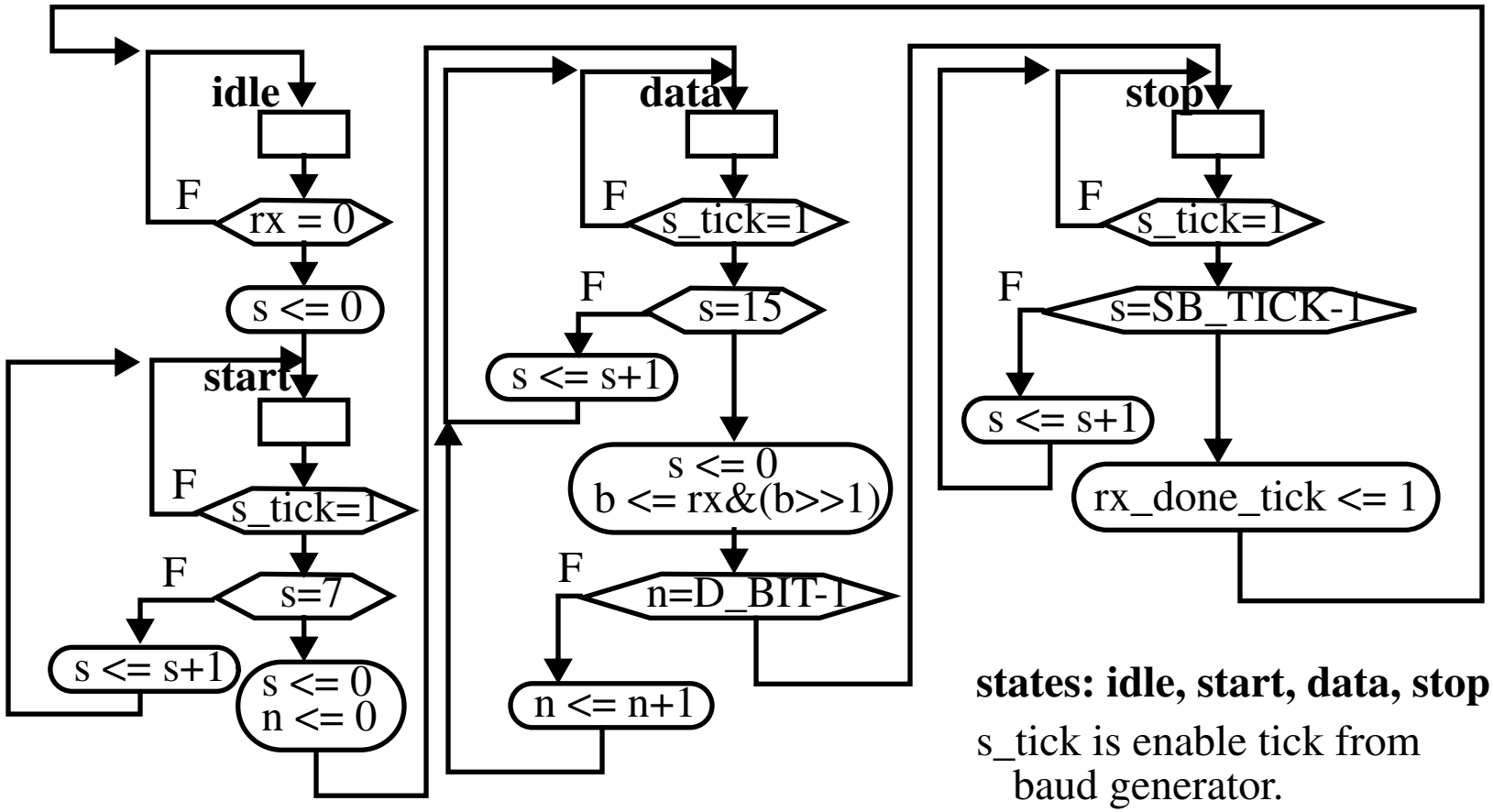
UART Receiving Subsystem: mod-163 cnter

```
    begin
    process(clk, reset)
        begin
        if (reset = '1') then
            r_reg <= (others => '0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
        end process;
    end arch;

  -- next state logic
    r_next <= (others => '0') when r_reg=(M-1) else
                  r_reg + 1;
  -- output logic
    q <= std_logic_vector(r_reg);
    max_tick <= '1' when r_reg=(M-1) else '0';
  end arch;
```

UART Receiving Subsystem: UART receiver

The ASMD chart for the receiver is shown below:



states: idle, start, data, stop

s_tick is enable tick from baud generator.

D_BIT indicates the number of data bits and SB_TICK indicates the number of ticks needed for the stop bits (16, 24 and 32 for 1, 1.5 and 2 stop bits).

UART Receiving Subsystem: UART receiver

We will assign D_BIT and SB_TICK to 8 and 16, respectively, in our design.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity uart_rx is
  generic(
      DBIT: integer := 8;
      SB_TICK: integer := 16;
  );
  port(
      clk, reset: in std_logic;
      rx: in std_logic;
      s_tick: in std_logic;
      rx_done_tick: out std_logic;
      dout: out std_logic_vctor(7 downto 0)
  );
  end uart_rx;
```

UART Receiving Subsystem: UART receiver

```
architecture arch of uart_rx is
  type state_type is (idle, start, data, stop);
  signal state_reg, state_next: state_type;
  signal s_reg, s_next: unsigned(3 downto 0);
  signal n_reg, n_next: unsigned(2 downto 0);
  signal b_reg, b_next: std_logic_vector(7 downto 0);
  begin
  process(clk, reset) -- FSMD state and data regs.
     begin
     if (reset = '1') then
        state_reg <= idle;
        s_reg <= (others => '0');
        n_reg <= (others => '0');
        b_reg <= (others => '0');
     elsif (clk'event and clk='1') then
        state_reg <= state_next;
        s_reg <= s_next;
        n_reg <= n_next;
```

UART Receiving Subsystem: UART receiver

```
            b_reg <= b_next;
        end if;
        end process;

    -- next state logic
      process (state_reg, s_reg, n_reg, b_reg, s_tick, rx)
        begin
        state_next <= state_reg;
        s_next <= s_reg;
        n_next <= n_reg;
        b_next <= b_reg;
        rx_done_tick <= '0';
        case state_reg is
          when idle =>
            if (rx = '0') then
              state_next <= start;
              s_next <= (others => '0');
            end if;
```

UART Receiving Subsystem: UART receiver

```
            when start =>
                if (s_tick = '1') then
                    if (s_reg = 7) then
                        state_next <= data;
                        s_next <= (others => '0');
                        n_next <= (others => '0');
                    else
                        s_next <= s_reg + 1;
                    end if;
                end if;
            when data =>
                if (s_tick = '1') then
                    if (s_reg = 15) then
                        s_next <= (others => '0');
                        b_next <= rx & b_reg(7 downto 1);
                        if (n_reg = (DBIT - 1)) then
                            state_next <= stop;
                        else
```

UART Receiving Subsystem: UART receiver

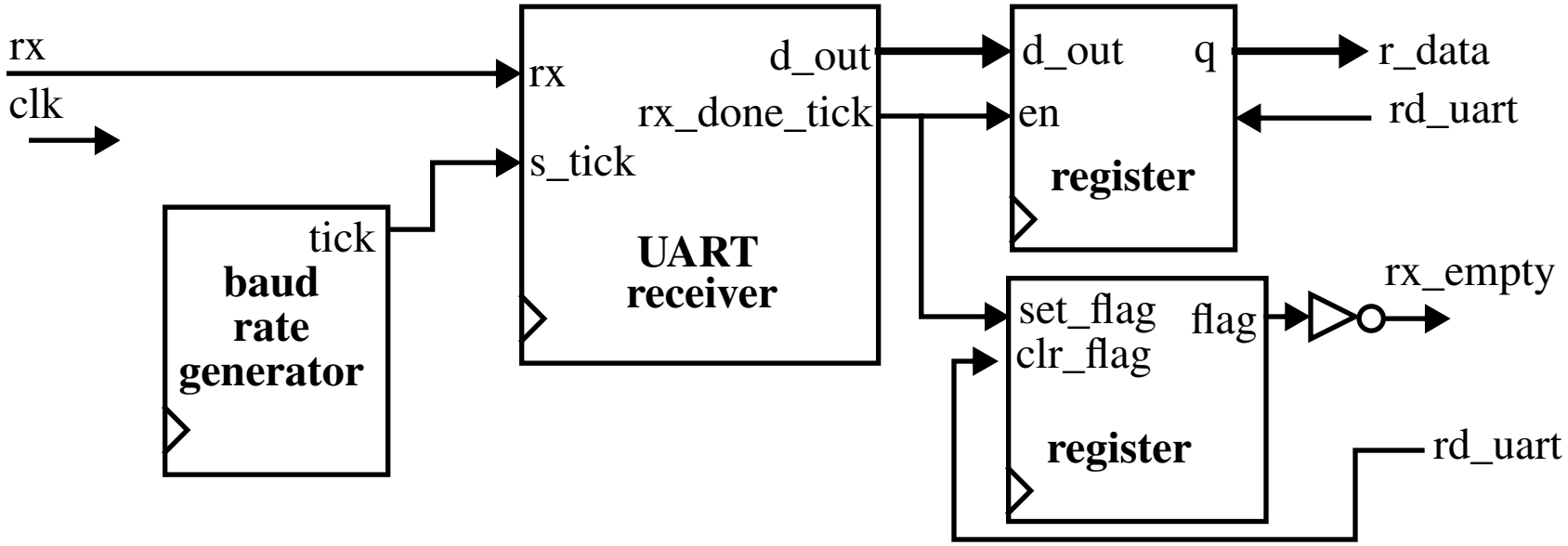
```
                        n_next <= n_reg + 1;
                    end if;
                else
                    s_next <= s_reg + 1;
                end if;
            end if;
        when stop =>
            if (s_tick = '1') then
                if (s_reg = (SB_TICK-1)) then
                    state_next <= idle;
                    rx_done_tick <= '1';
                else
                    s_next <= s_reg + 1;
                end if;
            end if;
        end case;
    end process;
  dout <= b_reg; end arch;
```

UART Receiving Subsystem: Interface Circuit

The receiver interface circuit has two functions:

- It provides a mechanism to signal the availability of a new word
- It provides buffer space between the receiver and main system.

Several architectures are possible, including one with a FIFO. Here is one with a flag **FF** (to indicate the reception of a data byte) and a one byte buffer.



Here, *rx_done_tick* is connected to *set_flag* while the system connects to *clr_flag*. The system checks *rx_empty* to determine when a data byte is available.

UART Receiving Subsystem: Interface Circuit

When *rx_done_tick* is asserted, the received byte is loaded to the buffer and the flag ***FF*** is asserted.

The receiver can continue to fetch the next byte, giving time for the system to retrieve the current byte.

```
library ieee;
use ieee.std_logic_1164.all;
entity flag_buff is
  generic(W: integer := 8);
  port(
      clk, reset: in std_logic;
      clr_flag, set_flag: in std_logic;
      din: in std_logic_vector(W-1 downto 0);
      dout: out std_logic_vector(W-1 downto 0);
      flag: out std_logic
      );
  end flag_buff;
```

UART Receiving Subsystem: Interface Circuit

```
architecture arch of flag_buff is
  signal buf_reg, buf_next: std_logic_vector(W-1 downto
0);
  signal flag_reg, flag_next: std_logic;
  begin
  process(clk, reset)
     begin
     if (reset = '1') then
        buf_reg <= (others =>'0');
        flag_reg <= '0';
     elsif (clk'event and clk='1') then
        buf_reg <= buf_next;
        flag_reg <= flag_next;
     end if;
  end process;

  -- next-state logic
```

UART Receiving Subsystem: Interface Circuit

```
    process (buf_reg, flag_reg, set_flag, clr_flag, din)
        begin
        buf_next <= buf_reg;
        flag_next <= flag_reg;
        if (set_flag = '1') then
            buf_next <= din;
            flag_next <= '1';
        elsif (clr_flag = '1') then
            flag_next <= '0';
        end if;
    end process;

  -- output logic
    dout <= buf_reg;
    flag <= flag_reg;
  end arch;
```

UART Transmitting Subsystem

The UART transmitting subsystem is similar to the receiving subsystem.

It consists of UART transmitter, baud rate generator and interface circuit.

Roles are reversed for the interface circuit, i.e., the **system** sets the flag FF or writes the buffer interface circuit while the UART transmitter clears FF or reads the buffer.

The transmitter is essentially a shift register that shifts out data bits.

Since no *oversampling* is involved, the frequency of the ticks are 16 times slower than that of the receiver.

However, instead of introducing another cnter, the transmitter usually shares the baud rate generator and uses an internal cnter to cnt through the 16 ticks.

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

UART Transmitting Subsystem

```
entity uart_tx is
  generic(
      DBIT: integer := 8;
      SB_TICK: integer := 16;
  );
  port(
      clk, reset: in std_logic;
      tx_start: in std_logic;
      s_tick: in std_logic;
      din: out std_logic_vctor(7 downto 0)
      tx_done_tick: out std_logic;
      tx: out std_logic
  );
  end uart_tx;
```

UART Transmitting Subsystem

```
architecture arch of uart_tx is
  type state_type is (idle, start, data, stop);
  signal state_reg, state_next: state_type;
  signal s_reg, s_next: unsigned(3 downto 0);
  signal n_reg, n_next: unsigned(2 downto 0);
  signal b_reg, b_next: std_logic_vector(7 downto 0);
  signal tx_reg, tx_next: std_logic;
  begin
  process(clk, reset) -- FSMD state and data regs.
     begin
     if (reset = '1') then
        state_reg <= idle;
        s_reg <= (others => '0');
        n_reg <= (others => '0');
        b_reg <= (others => '0');
        tx_reg <= '1';
```

UART Transmitting Subsystem

```
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
            s_reg <= s_next;
            n_reg <= n_next;
            b_reg <= b_next;
            tx_reg <= tx_next;
        end if;
        end process;

    -- next state logic
      process (state_reg, s_reg, n_reg, b_reg, s_tick,
    tx_reg, tx_start, din)
        begin
        state_next <= state_reg;
        s_next <= s_reg;
        n_next <= n_reg;
        b_next <= b_reg;
        tx_next <= tx_reg;
```

UART Transmitting Subsystem

```
tx_done_tick <= '0';
case state_reg is
   when idle =>
      tx_next <= '1';
      if (tx_start = '1') then
         state_next <= start;
         s_next <= (others => '0');
         b_next <= din;
      end if;
   when start =>
      tx_next <= '0';
      if (s_tick = '1') then
         if (s_reg = 15) then
            state_next <= data;
            s_next <= (others => '0');
            n_next <= (others => '0');
         else
            s_next <= s_reg + 1;
```

UART Transmitting Subsystem

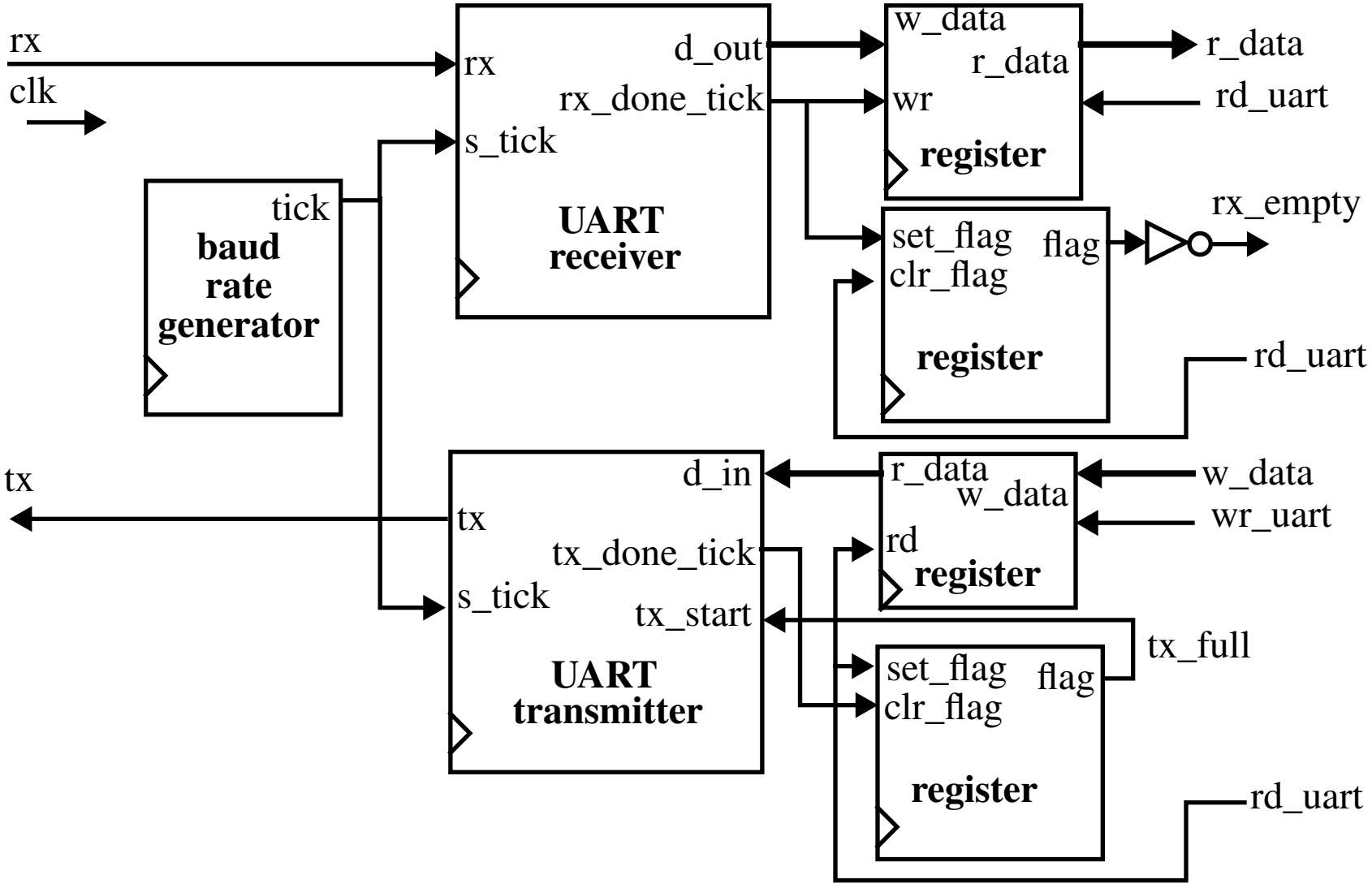
```
                    end if;
                end if;
            when data =>
                tx_next <= b_reg(0);
                if (s_tick = '1') then
                    if (s_reg = 15) then
                        s_next <= (others => '0');
                        b_next <= '0' & b_reg(7 downto 1);
                        if (n_reg = (DBIT - 1)) then
                            state_next <= stop;
                        else
                            n_next <= n_reg + 1;
                        end if;
                    else
                        s_next <= s_reg + 1;
                    end if;
                end if;
```

UART Transmitting Subsystem

```
                when stop =>
                    tx_next <= '1';
                    if (s_tick = '1') then
                        if (s_reg = (SB_TICK-1)) then
                            state_next <= idle;
                            rx_done_tick <= '1';
                        else
                            s_next <= s_reg + 1;
                        end if;
                    end if;
                end case;
            end process;
          tx <= tx_reg;
        end arch;
```

Entire UART System

Block diagram of whole system



Entire UART System

See text for the UART main module that instantiates the entities discussed above.

Note text uses a FIFO buffer so the diagram above is different.

The text also includes a **loop-back** circuit that instantiates the UART and connects the outputs of the receiver with the inputs of the transmitter on the FPGA.

It also adds '1' to the incoming data before looping it back to the computer.

Windows hyperterminal is discussed as a mechanism to communicate directly to the serial ports on your computer.