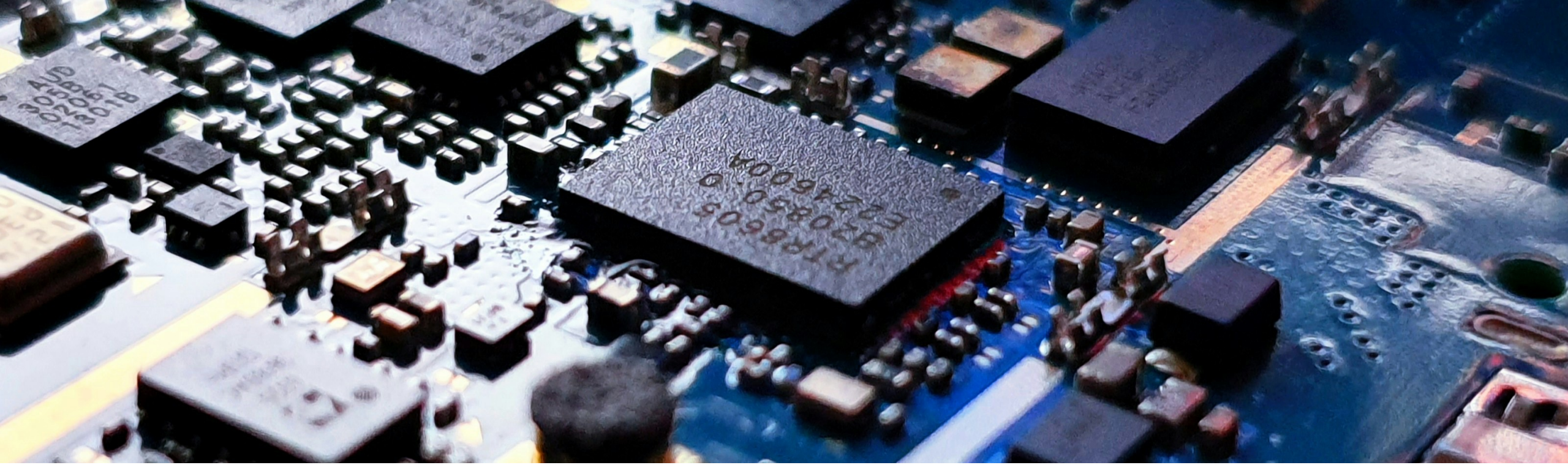


Laboratory 9

UART Design (Part 3)

Computer Architecture

A.Y. 2023/24

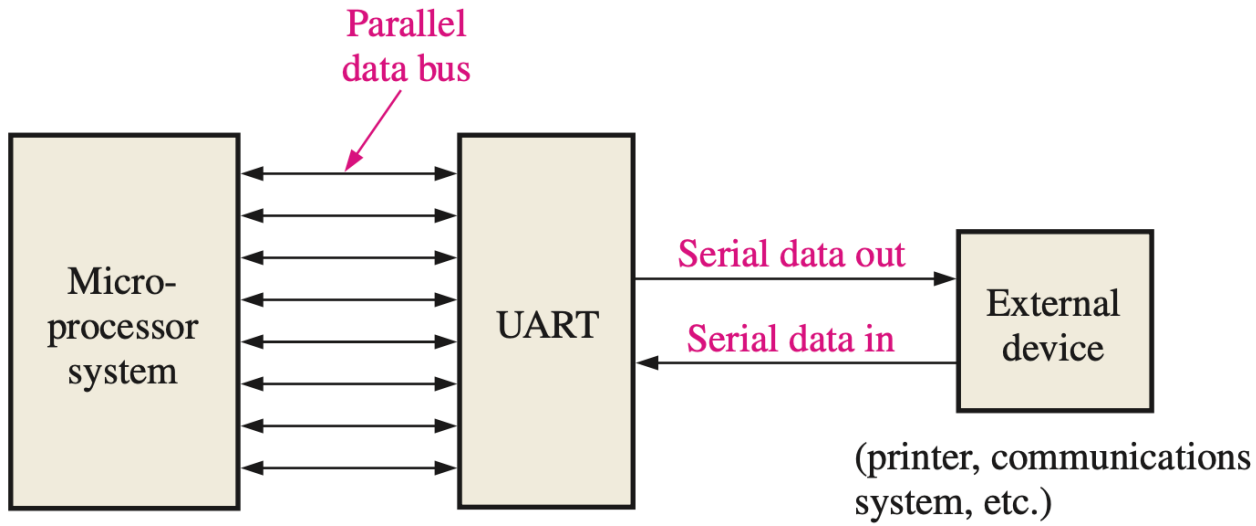
Laboratory goals

- Understand the structure and functions of a UART device
- Design and implement said functions in VHDL (over multiple labs)
- In this lab: implement the reception logic.

Recap

UART – Definition

A Universal Asynchronous Receiver Transmitter (UART) device is used to communicate between a microprocessor, or device that uses parallel data buses, and devices that communicate in serial data.



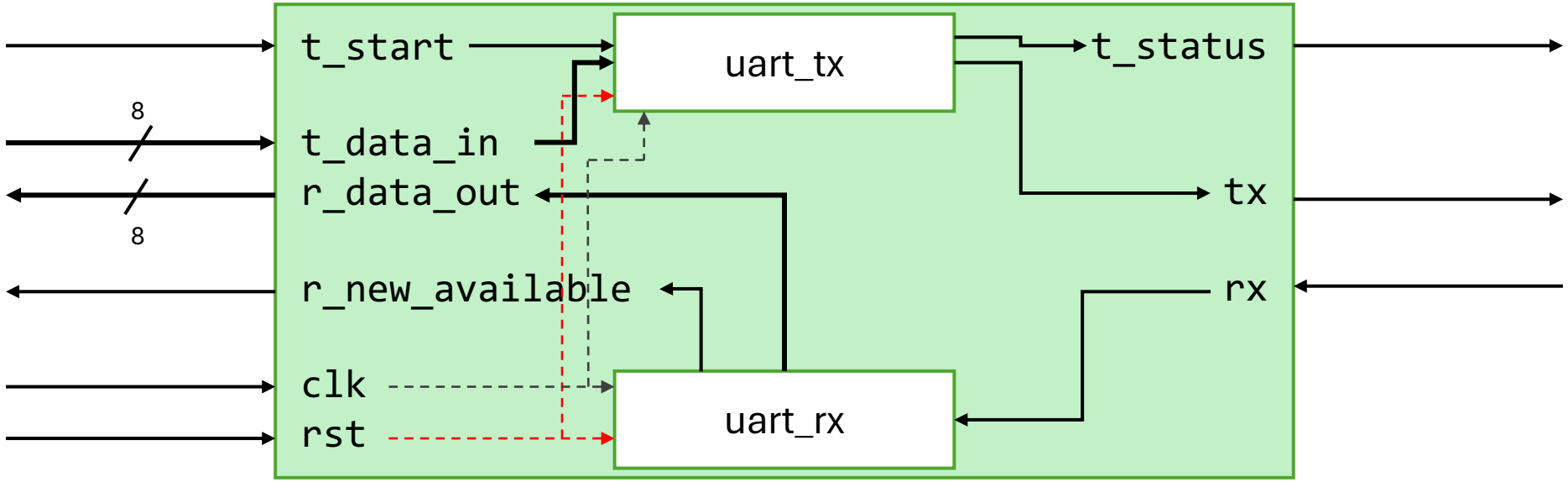
UART Data Transfer Protocol

To enable data transfer without synchronization, both the UART and the device at the other end must agree on:

- Transmission speed (i.e. bit rate or baud)
- Data format: send 8-bit characters, framed by a start bit and a stop bit.

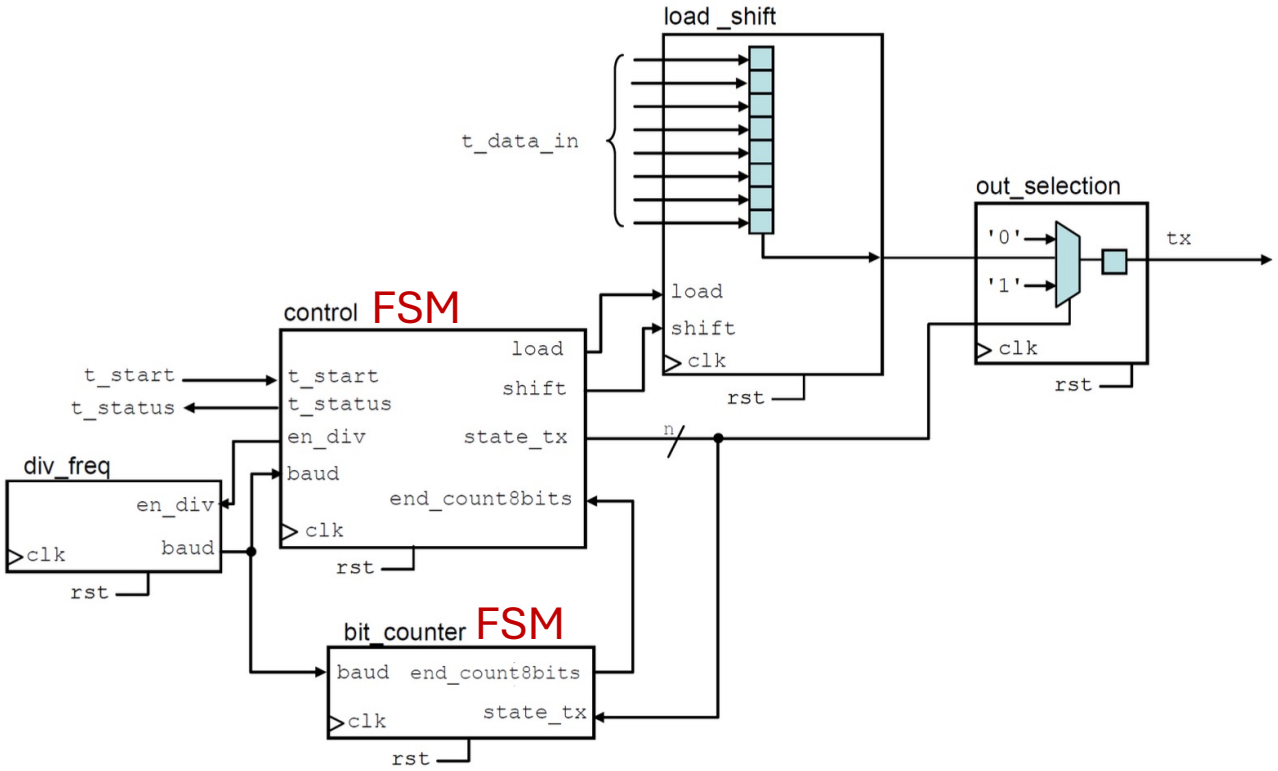
Design of a UART – Example

1. *Essential block diagram for your design.*



Design of a UART – Example

2. Finite State Machine diagram (TX).



Transmitter design

Exercise 1: Implement a shift register in VHDL and validate its functionality.

Exercise 2: Implement the frequency divider in VHDL and test its functionality.

Ex. 2b: If your design has other blocks, implement and test them in VHDL.

Exercise 3: Design the transmission control block FSM, define the number of states and the truth table.

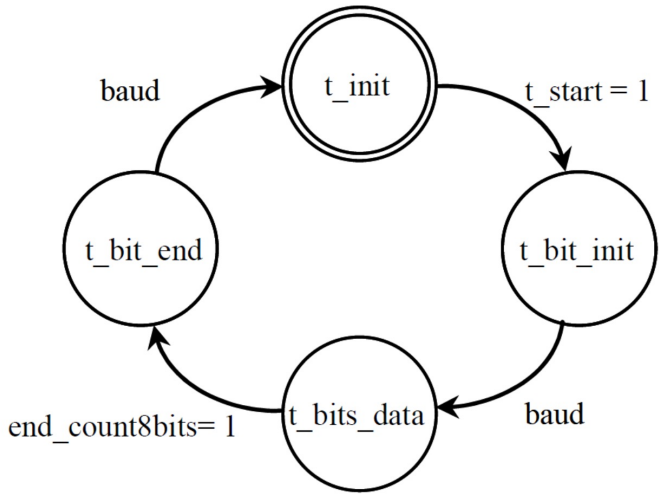
Exercise 4: Implement the transmission control block in VHDL.

Exercise 5: Join all the blocks and design a testbed to validate the transmission functionality.

Laboratory 8 Solution – Transmitter (part 2)

Transmitter design – Solution (3)

Exercise 3: Design the transmission control FSM, define the number of states and the truth table.



baud: one clock pulse.

Note: rst signals and outputs are omitted in the graph, but should be considered (in the truth table.)

pr_state	Inputs			nx_state	Outputs			
	t_start	baud	end_count8bits		load	shift	en_div	t_status
t_init	**0**	X	X	t_init	0	0	0	0
t_init	**1**	X	X	t_bit_init	1	0	0	0
t_bit_init	X	**0**	X	t_bit_init	0	0	1	1
t_bit_init	X	**1**	X	t_bits_data	0	baud	1	1
t_bits_data	X	X	**0**	t_bits_data	0	baud	1	1
t_bits_data	X	X	**1**	t_bit_end	0	0	1	1
t_bit_end	X	**0**	X	t_bit_end	0	0	1	1
t_bit_end	X	**1**	X	t_init	0	0	1	1

Transmitter design – Solution (4)

Exercise 4: Implement the transmission control block in VHDL.

control.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY control IS
   PORT( t_start, baud, end_count8bits, clk, rst: IN STD_LOGIC;
         state_tx: OUT STD_LOGIC_vector (1 downto 0);
         load, shift, t_status, en_div: OUT STD_LOGIC);
END control;
---------------------------------------
ARCHITECTURE fsm OF control IS
   TYPE state IS (t_init, t_bit_init, t_bits_data, t_bit_end);
   SIGNAL pr_state, nx_state: state;
   SIGNAL temp : STD_LOGIC_VECTOR(3 downto 0);
   SIGNAL tmp_state: STD_LOGIC_VECTOR (1 downto 0);
BEGIN
------------SEQUENTIAL SECTION-------------------
sequential:
   PROCESS (rst, clk)
   BEGIN
      IF (rst = '1') THEN
         pr_state <= t_init;
      ELSIF (clk'EVENT AND clk = '1') THEN
         load <= temp(3);
         shift <= temp(2);
         en_div <= temp(1);
         t_status <= temp(0);
         state_tx <= tmp_state;
         pr_state <= nx_state;

      END IF;
   END PROCESS sequential;
```

```
------------LOGICAL SECTION----------------------
combinatorial:
   PROCESS (t_start, baud, end_count8bits, pr_state)
   BEGIN
      CASE pr_state IS
         WHEN t_init =>
            temp <= "0000";
            state_tx <= "00";
            IF (t_start='1') THEN
               nx_state <= t_bit_init;
               temp <= "1000";
            ELSE
               nx_state <= t_init;
            END IF;
         WHEN t_bit_init =>
            temp <= "0011";
            state_tx <= "01";
            IF (baud='1') THEN
               nx_state <= t_bits_data;
               temp <= "0111";
            ELSE
               nx_state <= t_bit_init;
            END IF;
         WHEN t_bits_data =>
            temp <= ('0' , baud, OTHERS =>'1');
            state_tx <= "10";
            IF (end_count8bits = '1') THEN
               temp <= "0011";
               nx_state <= t_bit_end;
            ELSE
               nx_state <= t_bits_data;
            END IF;
         WHEN t_bit_end =>
            temp <= "0011";
            state_tx <= "11";
            IF (baud='1') THEN
               nx_state <= t_init;
            ELSE
               nx_state <= t_bit_end;
            END IF;
      END CASE;
   END PROCESS combinatorial;
END fsm;
```

Transmitter design – Solution (5)

Exercise 5: Join all the blocks and design a testbed to validate the transmission functionality.

uart_tx.vhd

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

ENTITY uart_tx IS
   PORT(
         t_data_in_ext: IN STD_LOGIC_VECTOR (7 downto 0);
         t_start_ext, clk_ext, rst_ext: IN STD_LOGIC;
         load_ext, shift_ext, baud_ext, t_status_ext, tx_ext, end_count8_ext, en_div_ext: OUT STD_LOGIC;
         state_tx_ext: OUT STD_LOGIC_VECTOR (1 downto 0));
END uart_tx;

ARCHITECTURE behavior OF uart_tx IS
   COMPONENT control
      PORT( t_start, baud, end_count8bits, clk, rst: IN STD_LOGIC;
         state_tx: OUT STD_LOGIC_vector (1 downto 0);
         load, shift, t_status, en_div: OUT STD_LOGIC);
   END COMPONENT;

   COMPONENT div_freq
      GENERIC (N: INTEGER := 10);
      PORT( clk, rst, en_div: IN STD_LOGIC;
            baud: OUT STD_LOGIC);
   END COMPONENT;

   COMPONENT load_shift
      PORT( t_data_in : IN STD_LOGIC_VECTOR (7 downto 0);
            clk, load, shift, rst : IN STD_LOGIC;
            t_data_out : OUT STD_LOGIC);
   END COMPONENT;

   COMPONENT bit_counter
      PORT( clk, baud, rst: IN STD_LOGIC;
            end_count8bits: OUT STD_LOGIC;
            state_tx: IN STD_LOGIC_VECTOR (1 downto 0));
   END COMPONENT;

   COMPONENT out_selection
      PORT(
            clk, rst , in_data: IN STD_LOGIC;
            state_tx: IN STD_LOGIC_VECTOR (1 downto 0);
            tx: OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL baud_s, en_div_s, load_s, shift_s, end_count_s, tx_s : STD_LOGIC;
   SIGNAL state_s: STD_LOGIC_VECTOR (1 downto 0);

   BEGIN
      COM1: control
               port map(t_start_ext, baud_s, end_count_s, clk_ext,
                        rst_ext, state_s, load_s, shift_s, t_status_ext,
                        en_div_s);
      COM2: div_freq
               generic map (N => 10)
               port map (clk_ext, rst_ext, en_div_s, baud_s);

      COM3: load_shift
               port map(t_data_in_ext, clk_ext, load_s, shift_s, rst_ext, tx_s);

      COM4: bit_counter
               port map(clk_ext, baud_s, rst_ext, end_count_s, state_s);

      COM5: out_selection
               port map(clk_ext, rst_ext, tx_s, state_s, tx_ext);

      baud_ext <= baud_s;
      load_ext <= load_s;
      shift_ext <= shift_s;
      end_count8_ext <= end_count_s;
      state_tx_ext <= state_s;
      en_div_ext <= en_div_s;
END behavior;
```

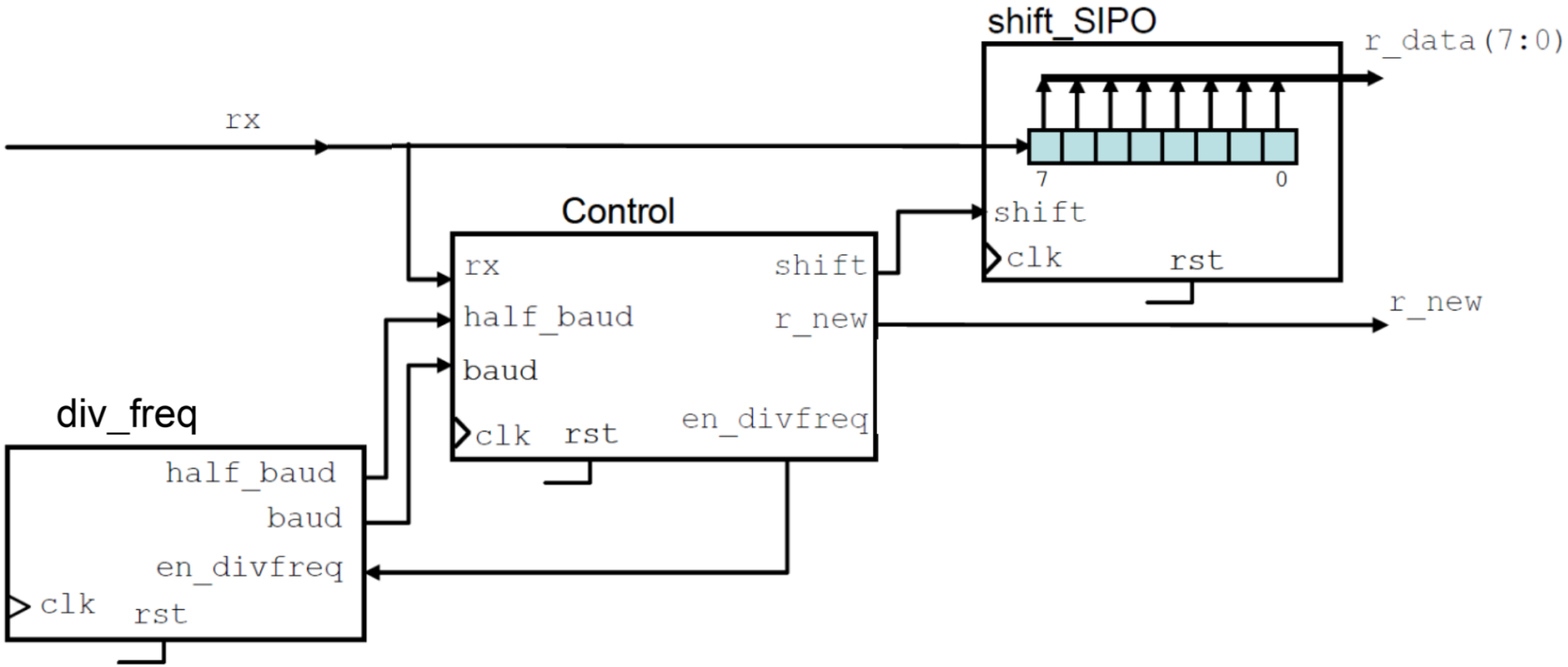
Receiver design exercise

Receiver behavior

All operations of the UART hardware are controlled by an internal clock signal, which typically runs 8 or 16 times the bit rate.

- The receiver tests the state of the incoming signal on each clock pulse, looking for the beginning of the start bit. If the apparent start bit lasts at least one-half of the bit time, it is valid and signals the start of a new character. If not, it is considered a spurious pulse and is ignored.
- After waiting one more bit time, the state of the line is sampled again, and the resulting level is clocked into a shift register. After the required number of bit periods for the character length have elapsed (usually 8), the contents of the shift register are made available to the receiving system. The UART will set a flag indicating new data is available.

Receiver block diagram



Receiver design

Exercise 1: Implement the shift SIPO register in VHDL and test its functionality.

Exercise 2: Implement the (receiver's) frequency divider in VHDL and test it.

Exercise 3: Design the reception control block FSM, define the number of states and the truth table.

Exercise 4: Implement the reception control block in VHDL.

Exercise 5: Join all the blocks and design a testbed to validate the receiver's functionality.

References

- Tokheim, R. Digital Electronics: Principles And Applications
- Floyd,Thomas L. Digital Fundamentals, 2016. Pearson
- Mano, Morris. Digital Desing.
- Intel QUARTUS Help Manuals
- Tocci, Ronald. Digital Systems Principles and Applications.