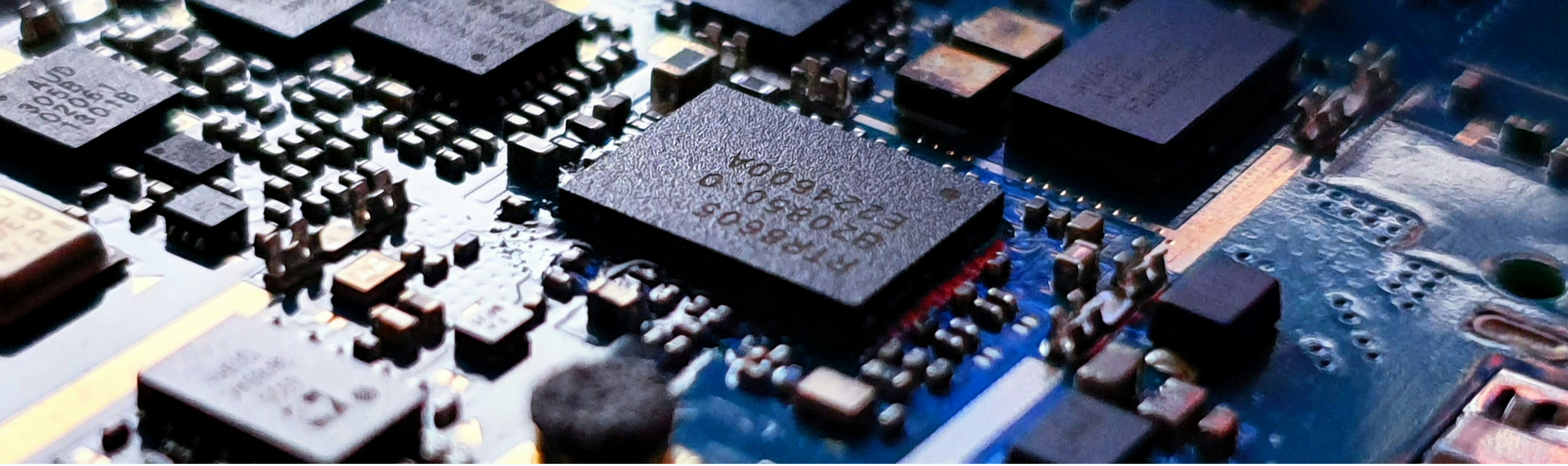


Laboratory 10

UART Design (Final)

Computer Architecture

A.Y. 2023/24

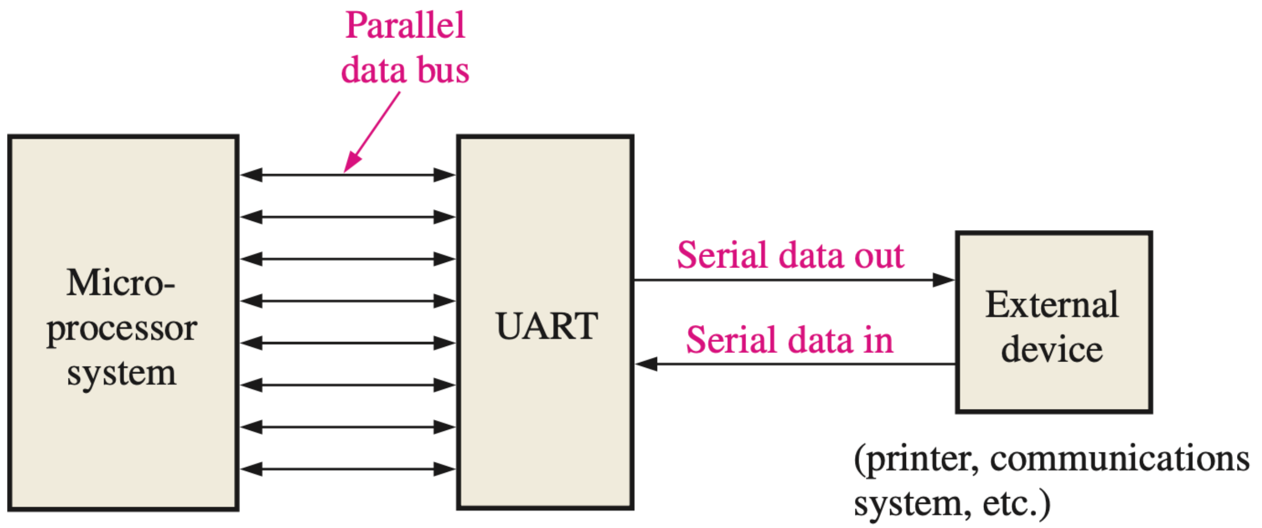
Laboratory goals

- Understand the structure and functions of a UART device
- Design and implement said functions in VHDL (over multiple labs)
- In this lab: Merge all components into a final design.

Recap

UART – Definition

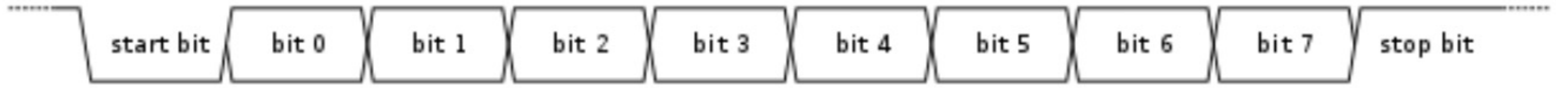
A Universal Asynchronous Receiver Transmitter (UART) device is used to communicate between a microprocessor, or device that uses parallel data buses, and devices that communicate in serial data.



UART Data Transfer Protocol

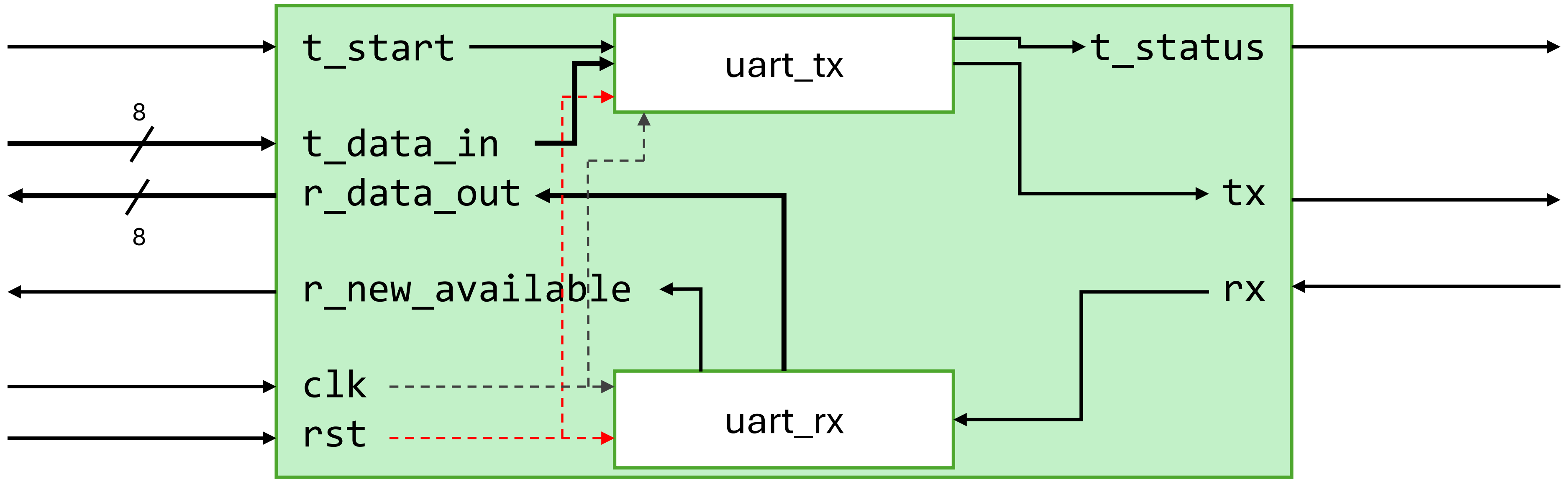
To enable data transfer without synchronization, both the UART and the device at the other end must agree on:

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



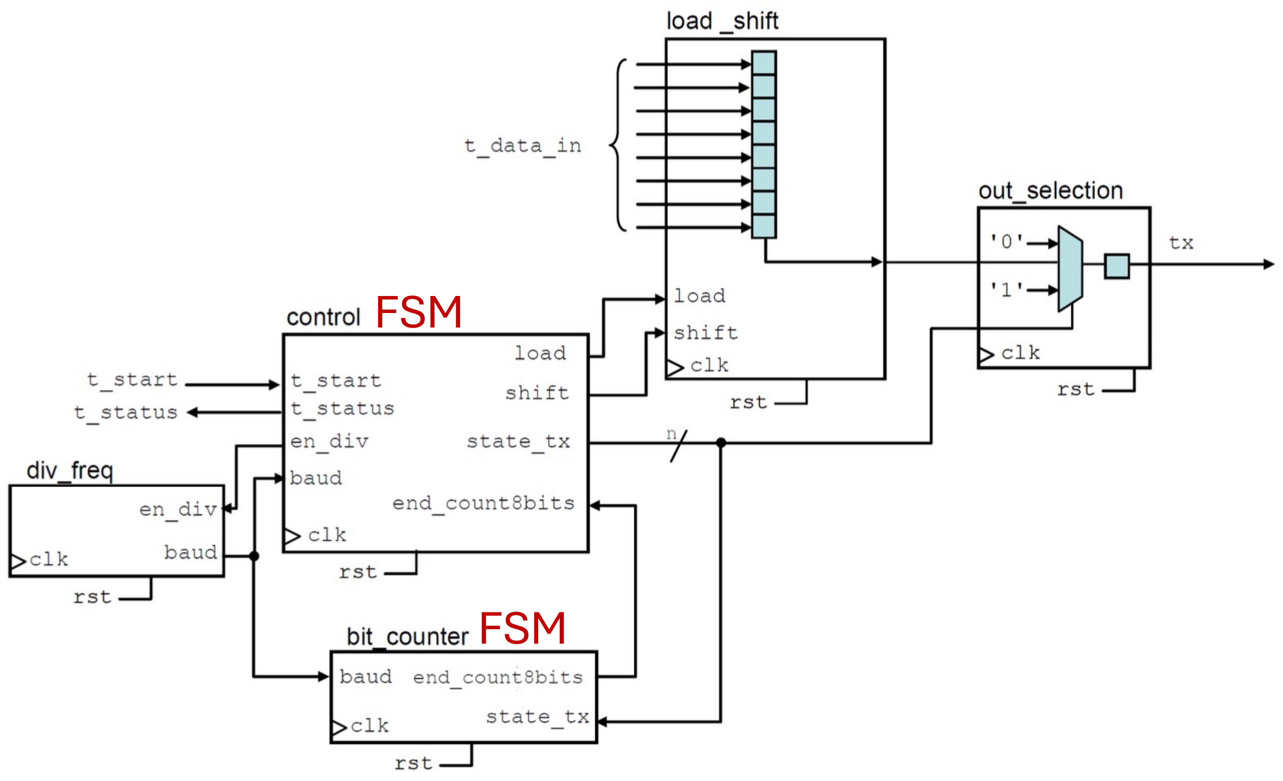
Design of a UART – Example

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.

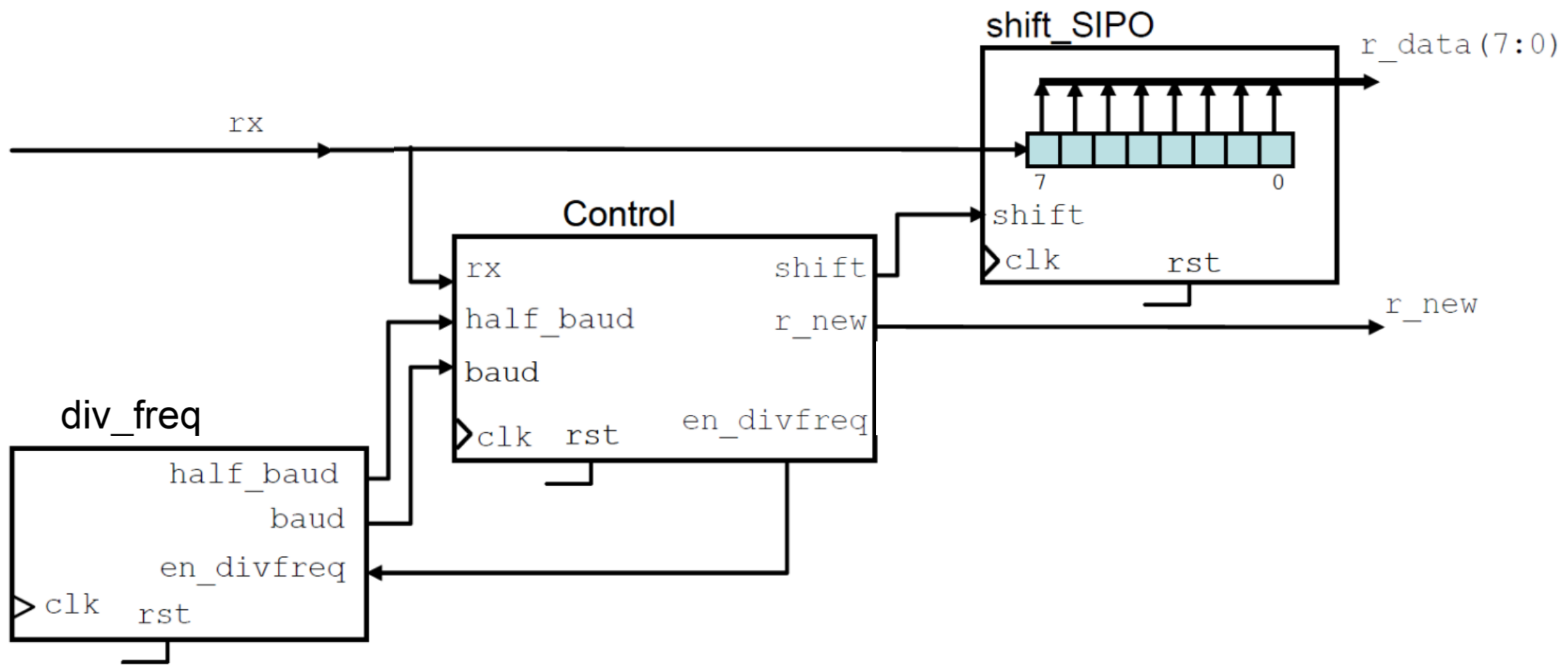
Exercise 5: Join all the blocks and design a testbed to validate the transmission functionality.

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.

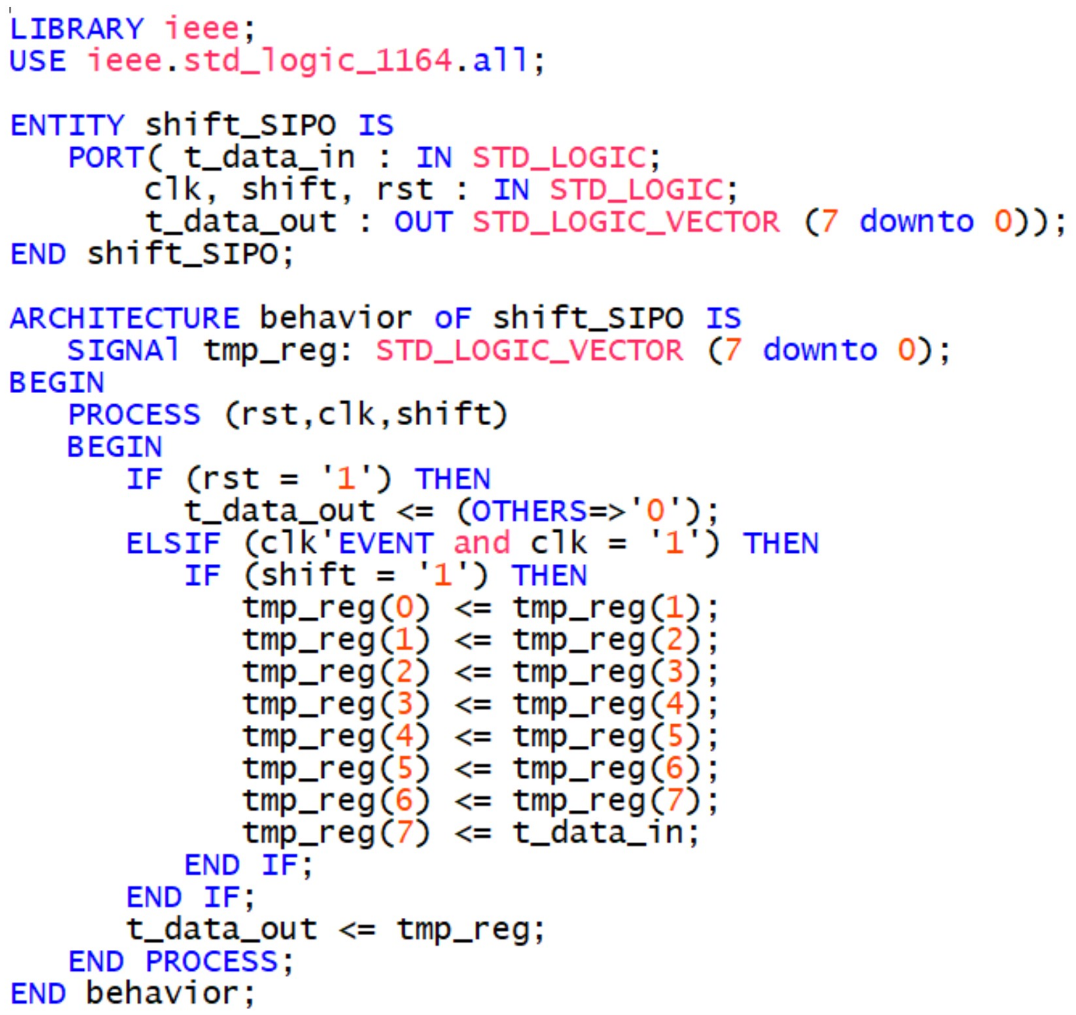
Exercise 5: Join all the blocks and design a testbed to validate the receiver's functionality.

Laboratory 9 Solution – Receiver

Receiver design solution

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

shift_SIPO.vhd

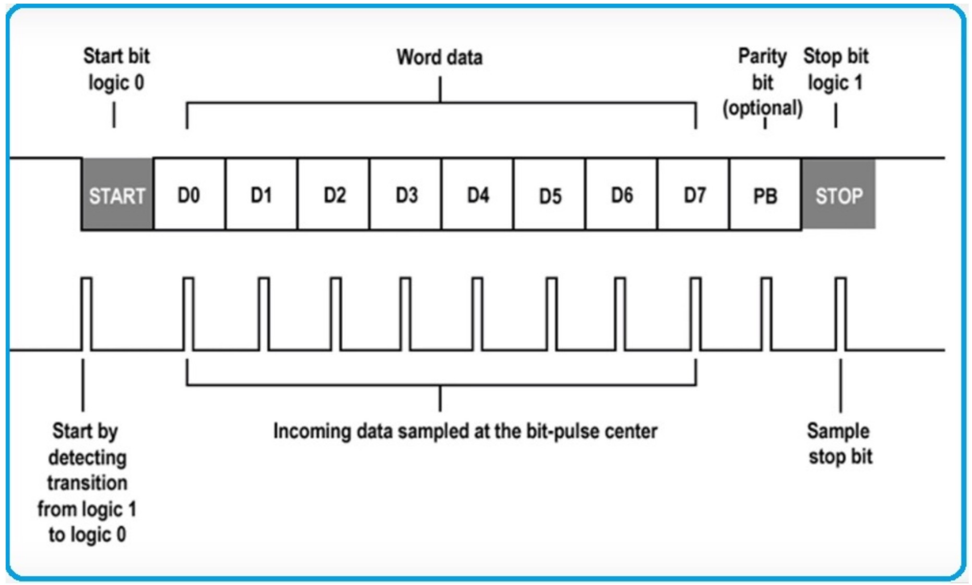


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift_SIPO IS
   PORT( t_data_in : IN STD_LOGIC;
         clk, shift, rst : IN STD_LOGIC;
         t_data_out : OUT STD_LOGIC_VECTOR (7 downto 0));
END shift_SIPO;

ARCHITECTURE behavior oF shift_SIPO IS
   SIGNAl tmp_reg: STD_LOGIC_VECTOR (7 downto 0);
BEGIN
   PROCESS (rst,clk,shift)
   BEGIN
      IF (rst = '1') THEN
         t_data_out <= (OTHERS=>'0');
      ELSIF (clk'EVENT and clk = '1') THEN
         IF (shift = '1') THEN
            tmp_reg(0) <= tmp_reg(1);
            tmp_reg(1) <= tmp_reg(2);
            tmp_reg(2) <= tmp_reg(3);
            tmp_reg(3) <= tmp_reg(4);
            tmp_reg(4) <= tmp_reg(5);
            tmp_reg(5) <= tmp_reg(6);
            tmp_reg(6) <= tmp_reg(7);
            tmp_reg(7) <= t_data_in;
         END IF;
      END IF;
      t_data_out <= tmp_reg;
   END PROCESS;
END behavior;
```

Receiver design solution

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

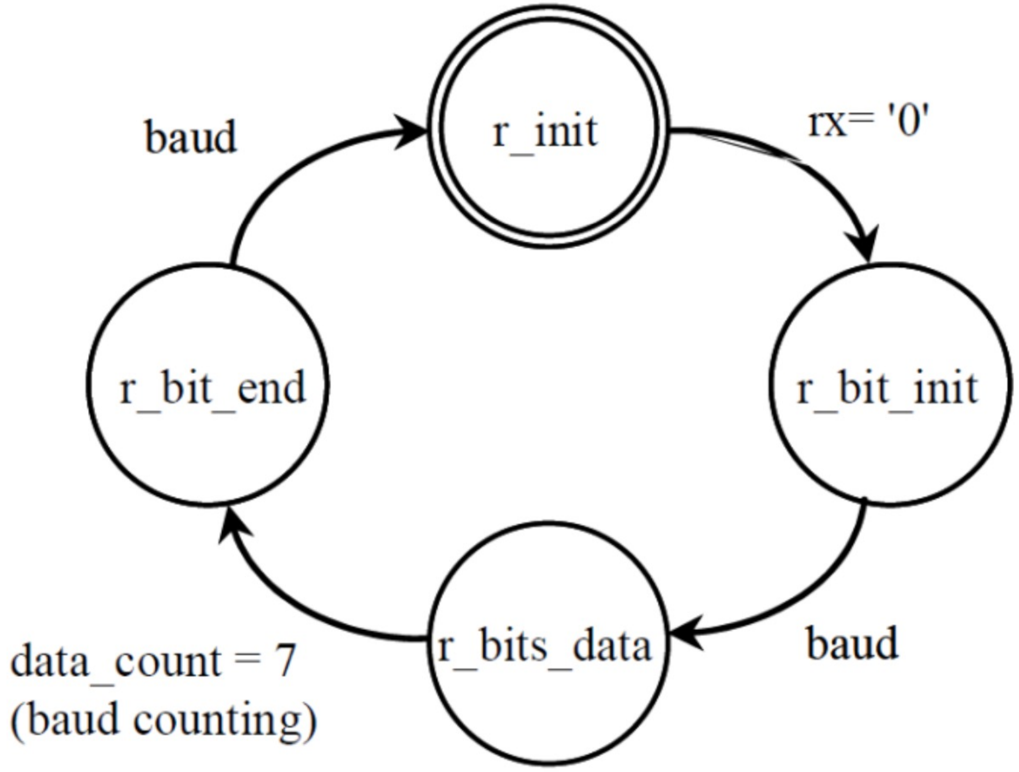
div_freq_rx.vhd



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

ENTITY div_freq_rx IS
   GENERIC (N: INTEGER := 10);
   PORT( clk, rst, en_div: IN STD_LOGIC;
         baud, half_baud: OUT STD_LOGIC);
END div_freq_rx;

ARCHITECTURE behavior OF div_freq_rx IS
   SIGNAL count_baud: INTEGER RANGE 0 TO N := 1;
BEGIN
   PROCESS (clk, rst)
   BEGIN
      IF rst = '1' THEN
         baud <= '0';
         half_baud <= '0';
         count_baud <=  1;
      ELSIF (clk'EVENT and clk = '1') THEN
         IF (en_div = '1') THEN
            IF count_baud = N THEN
               count_baud <= 1;
               baud <= '1';
            ELSIF count_baud = N/2 THEN
               half_baud <= '1';
               count_baud <= count_baud + 1;
            ELSE
               count_baud <= count_baud + 1;
               baud <= '0';
               half_baud <= '0';
            END IF;
         ELSE
            baud <= '0';
            half_baud <= '0';
            count_baud <= 1;
         END IF;
      END IF;
   END PROCESS;
END behavior;
```

Receiver design solution

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



- The baud signal is a pulse (one clock cycle)
- The rst transitions are omitted to keep the diagram legible but its functionality must be considered
- Outputs signals are omitted to keep the diagram legible. They are included in the truth table.

Receiver design solution

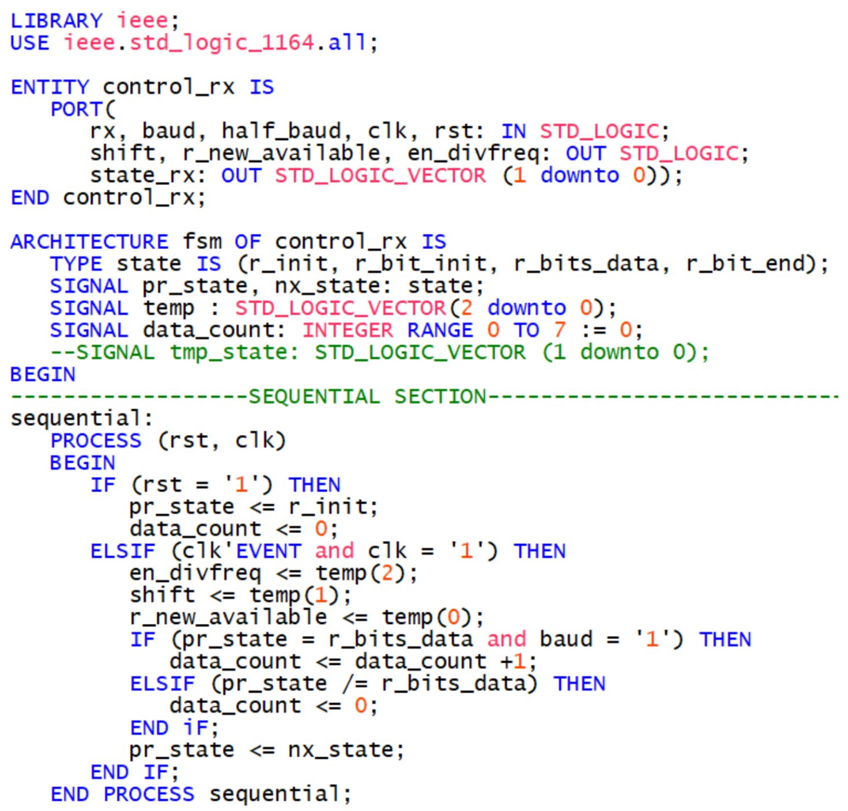
Exercise 3: Design the reception control block FSM, define the number of states and the truth table.

pr_state	Inputs				nx_state	Outputs		
	rx	baud	half_baud	data_count		en_div	shift	r_new
r_init	1	x	x	x	r_init	0	0	0
r_init	0	x	x	x	r_bit_init	0	0	0
r_bit_init	x	0	x	x	r_bit_init	1	0	0
r_bit_init	x	1	x	x	r_bits_data	1	0	0
r_bits_data	x	x	x	0	r_bits_data	1	half_baud	0
r_bits_data	x	x	x	1	r_bit_end	1	half_baud	0
r_bit_end	1	1	x	x	r_bit_init	1	0	1
r_bit_end	0	1	x	x	r_bit_init	1	0	0
r_bit_end	x	0	x	x	r_bit_end	1	0	0

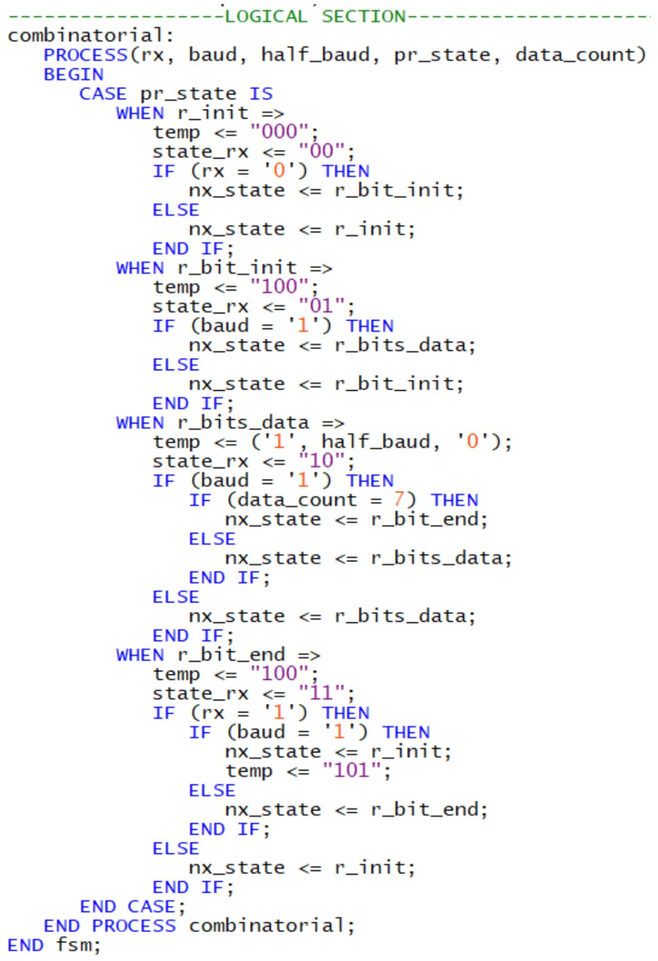
Receiver design solution

Exercise 4: Implement the reception control block in VHDL.

control_rx.vhd



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

ENTITY control_rx IS
   PORT(
      rx, baud, half_baud, clk, rst: IN STD_LOGIC;
      shift, r_new_available, en_divfreq: OUT STD_LOGIC;
      state_rx: OUT STD_LOGIC_VECTOR (1 downto 0));
END control_rx;

ARCHITECTURE fsm OF control_rx IS
   TYPE state IS (r_init, r_bit_init, r_bits_data, r_bit_end);
   SIGNAL pr_state, nx_state: state;
   SIGNAL temp : STD_LOGIC_VECTOR(2 downto 0);
   SIGNAL data_count: INTEGER RANGE 0 TO 7 := 0;
   --SIGNAL tmp_state: STD_LOGIC_VECTOR (1 downto 0);
BEGIN
------------------SEQUENTIAL SECTION--------------------------
sequential:
   PROCESS (rst, clk)
   BEGIN
      IF (rst = '1') THEN
         pr_state <= r_init;
         data_count <= 0;
      ELSIF (clk'EVENT and clk = '1') THEN
         en_divfreq <= temp(2);
         shift <= temp(1);
         r_new_available <= temp(0);
         IF (pr_state = r_bits_data and baud = '1') THEN
            data_count <= data_count +1;
         ELSIF (pr_state /= r_bits_data) THEN
            data_count <= 0;
         END iF;
         pr_state <= nx_state;
      END IF;
   END PROCESS sequential;
```

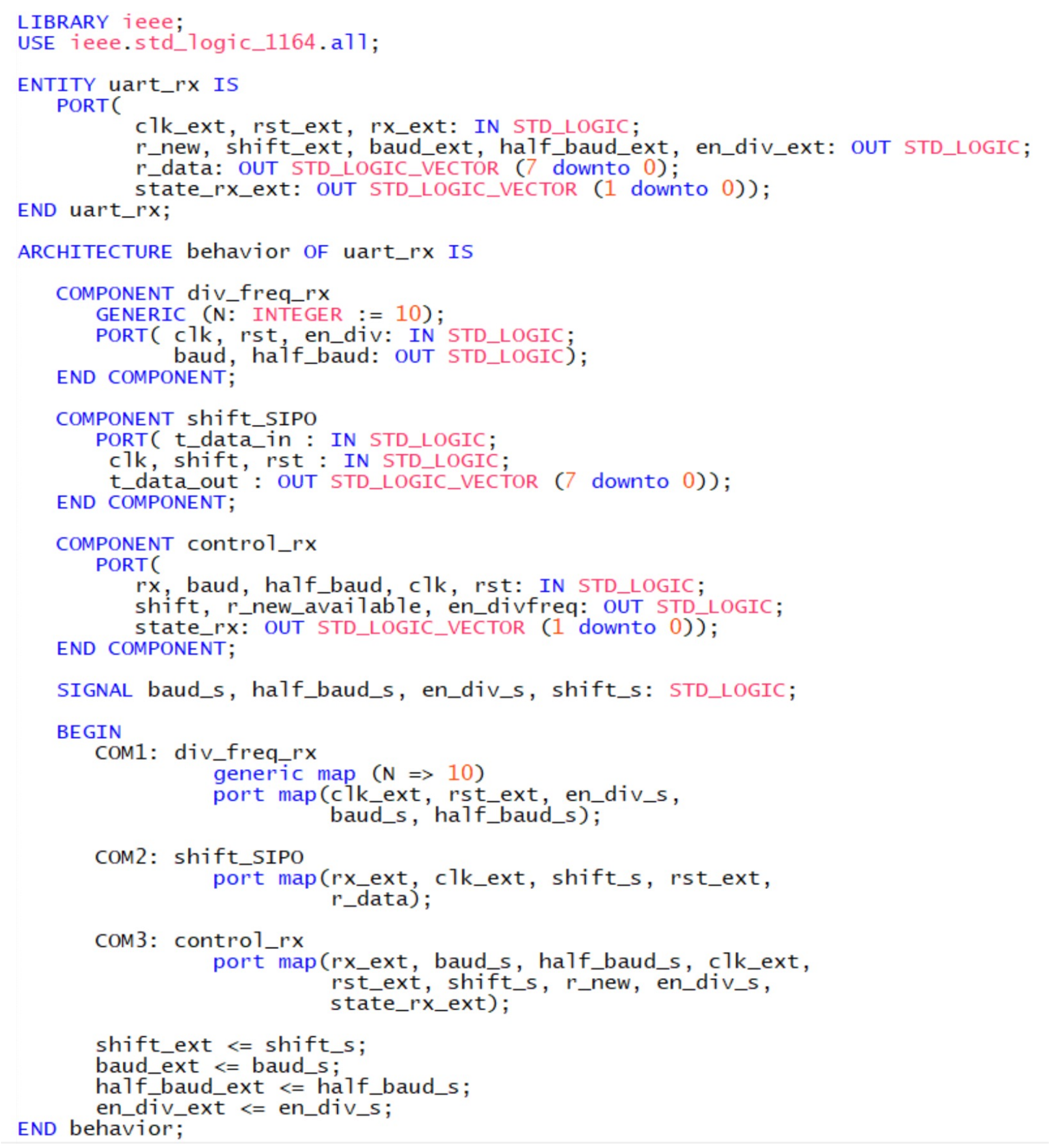


```
------------------LOGICAL SECTION-------------------
combinatorial:
   PROCESS(rx, baud, half_baud, pr_state, data_count)
   BEGIN
      CASE pr_state IS
         WHEN r_init =>
            temp <= "000";
            state_rx <= "00";
            IF (rx = '0') THEN
               nx_state <= r_bit_init;
            ELSE
               nx_state <= r_init;
            END IF;
         WHEN r_bit_init =>
            temp <= "100";
            state_rx <= "01";
            IF (baud = '1') THEN
               nx_state <= r_bits_data;
            ELSE
               nx_state <= r_bit_init;
            END IF;
         WHEN r_bits_data =>
            temp <= ('1', half_baud, '0');
            state_rx <= "10";
            IF (baud = '1') THEN
               IF (data_count = 7) THEN
                  nx_state <= r_bit_end;
               ELSE
                  nx_state <= r_bits_data;
               END IF;
            ELSE
               nx_state <= r_bits_data;
            END IF;
         WHEN r_bit_end =>
            temp <= "100";
            state_rx <= "11";
            IF (rx = '1') THEN
               IF (baud = '1') THEN
                  nx_state <= r_init;
                  temp <= "101";
               ELSE
                  nx_state <= r_bit_end;
               END IF;
            ELSE
               nx_state <= r_init;
            END IF;
      END CASE;
   END PROCESS combinatorial;
END fsm;
```

Receiver design solution

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

uart_rx.vhd



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

ENTITY uart_rx IS
   PORT(
         clk_ext, rst_ext, rx_ext: IN STD_LOGIC;
         r_new, shift_ext, baud_ext, half_baud_ext, en_div_ext: OUT STD_LOGIC;
         r_data: OUT STD_LOGIC_VECTOR (7 downto 0);
         state_rx_ext: OUT STD_LOGIC_VECTOR (1 downto 0));
END uart_rx;

ARCHITECTURE behavior OF uart_rx IS

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

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

   COMPONENT control_rx
      PORT(
         rx, baud, half_baud, clk, rst: IN STD_LOGIC;
         shift, r_new_available, en_divfreq: OUT STD_LOGIC;
         state_rx: OUT STD_LOGIC_VECTOR (1 downto 0));
   END COMPONENT;

   SIGNAL baud_s, half_baud_s, en_div_s, shift_s: STD_LOGIC;

   BEGIN
      COM1: div_freq_rx
               generic map (N => 10)
               port map(clk_ext, rst_ext, en_div_s,
                        baud_s, half_baud_s);

      COM2: shift_SIPO
               port map(rx_ext, clk_ext, shift_s, rst_ext,
                        r_data);

      COM3: control_rx
               port map(rx_ext, baud_s, half_baud_s, clk_ext,
                        rst_ext, shift_s, r_new, en_div_s,
                        state_rx_ext);

      shift_ext <= shift_s;
      baud_ext <= baud_s;
      half_baud_ext <= half_baud_s;
      en_div_ext <= en_div_s;
END behavior;
```

Exercise – Complete UART control

Complete UART control

Exercise 1: Merge the transmission and reception blocks in one single VHDL design file.

Exercise 2: Test the final UART circuit by performing a functional simulation.

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.