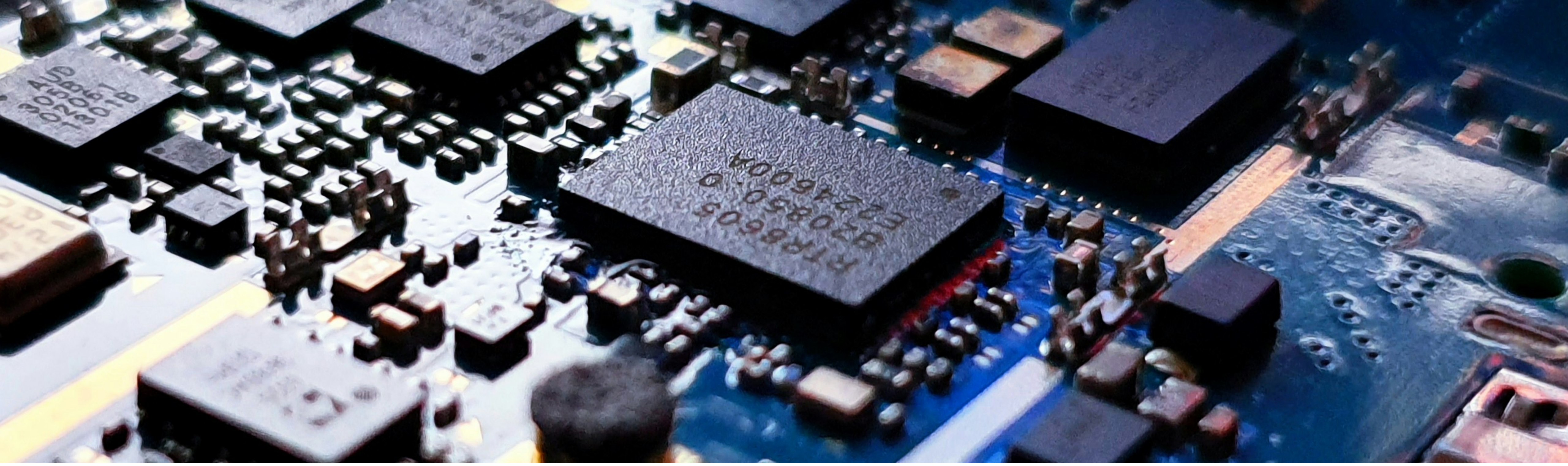


Laboratory 8

UART Design (Part 2)

Computer Architecture

A.Y. 2023/24

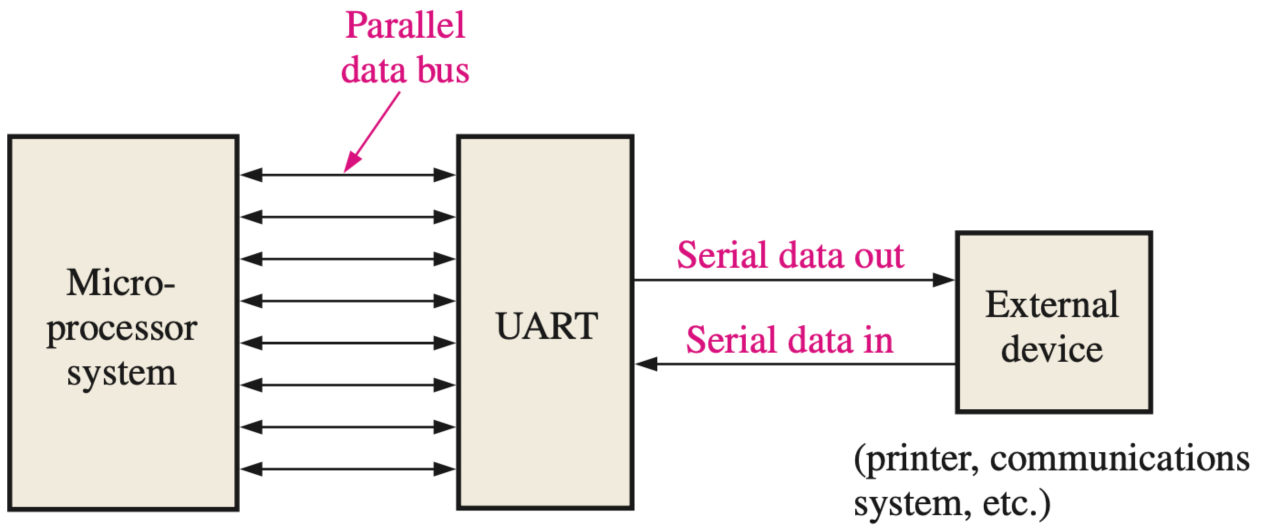
Laboratory goals

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

Recap

UART – Definition

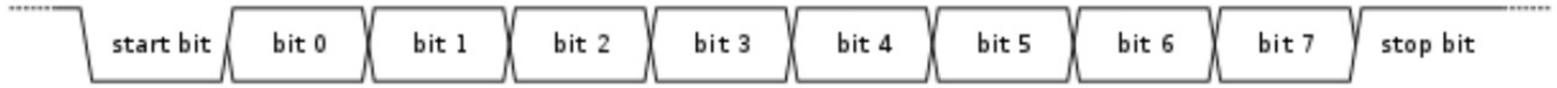
A Universal Asynchronous Receiver Transmitter (UART) device is used to communicate between a microprocessor, or device that uses parallel data buses, and devices that communicate in serial data.



UART Data Transfer Protocol

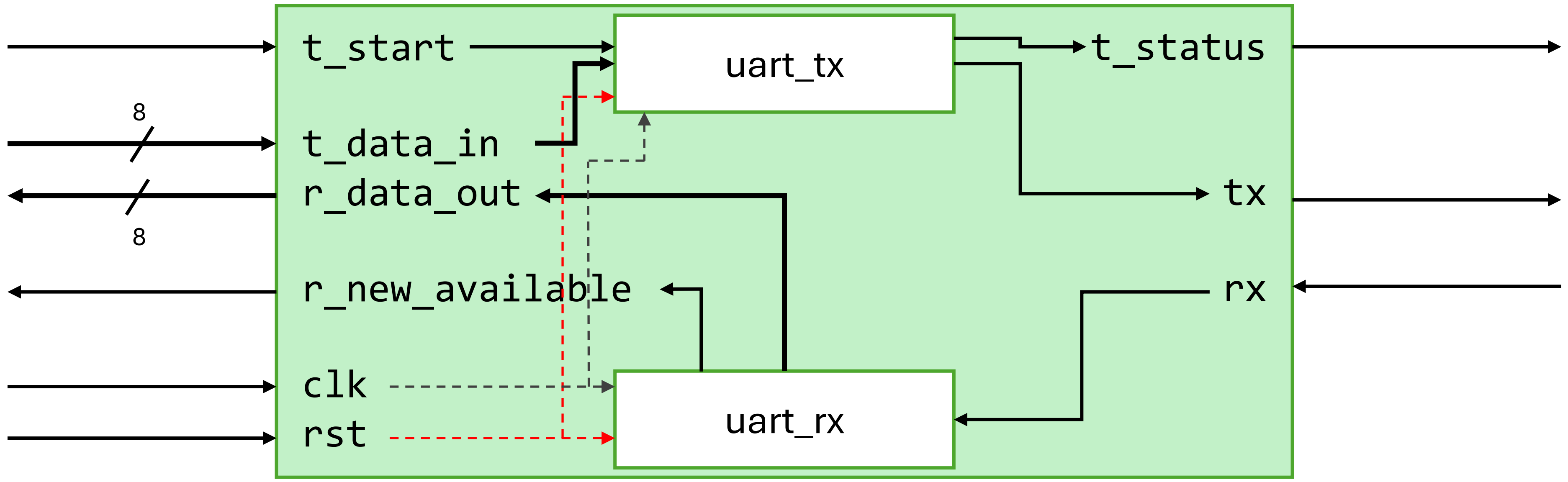
To enable data transfer without synchronization, both the UART and the device at the other end must agree on:

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



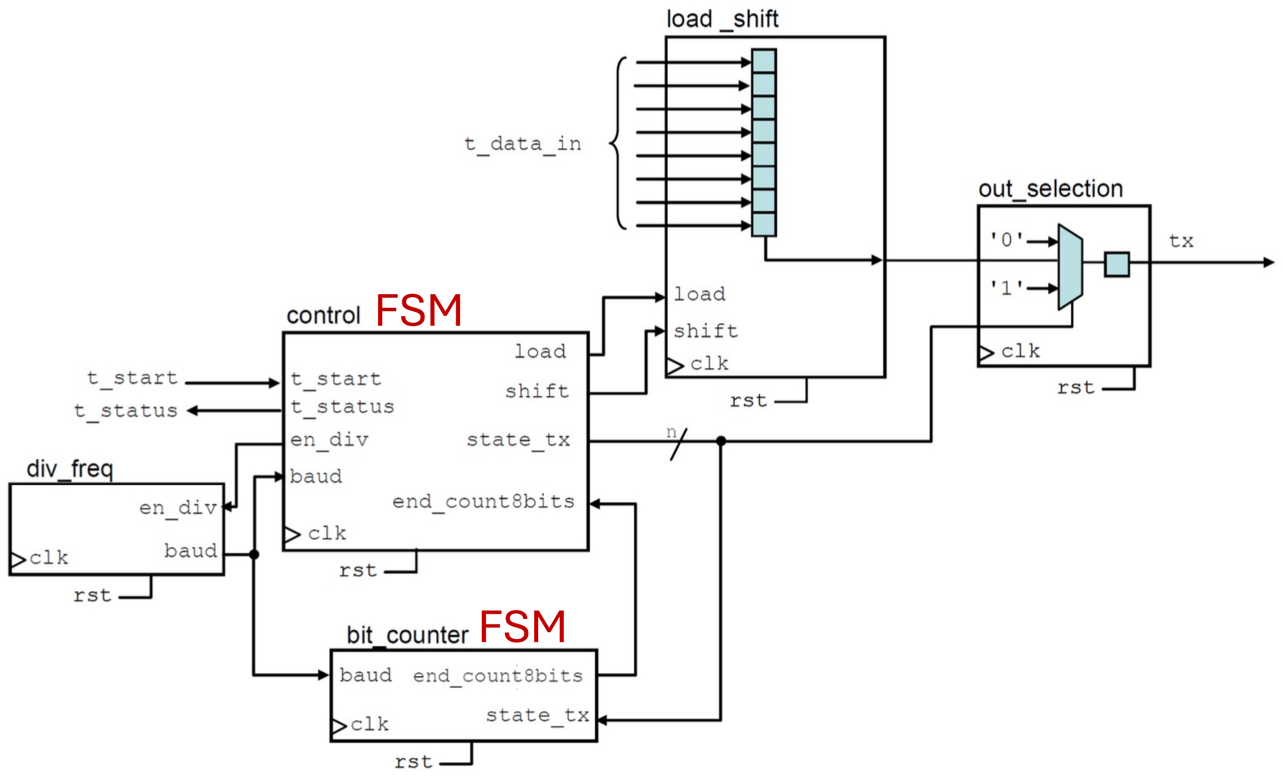
Design of a UART – Example

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.

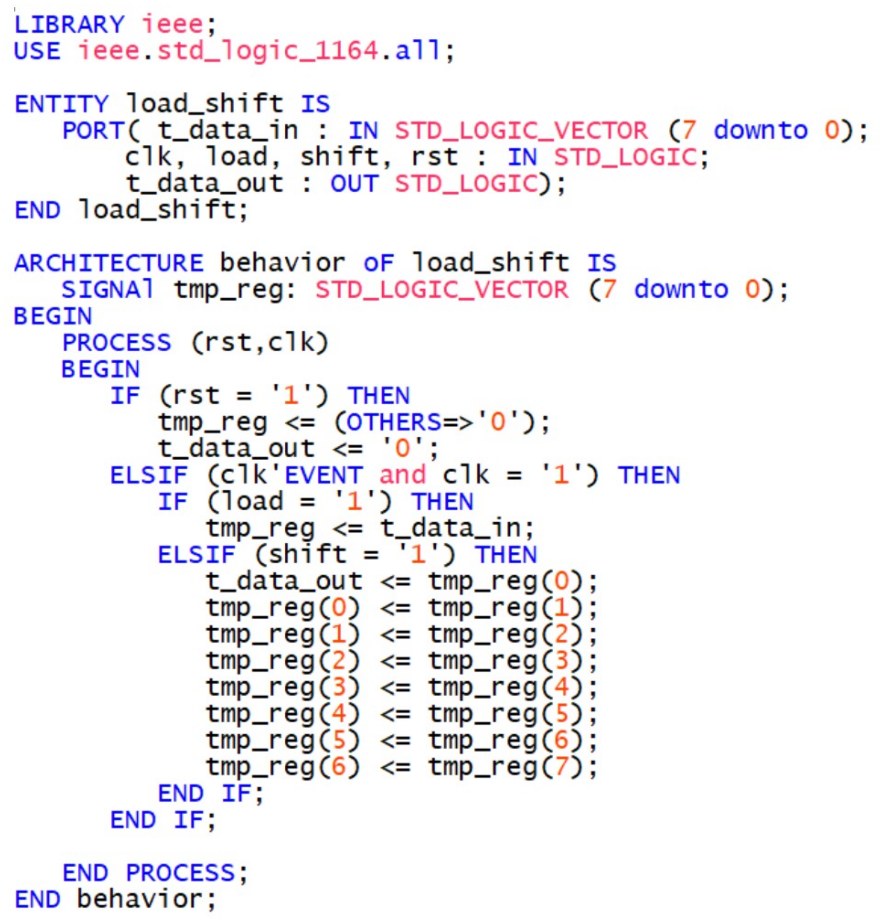
Exercise 4: Implement the transmission control block in VHDL.

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

Laboratory 7 Solution – Transmitter (part 1)

Transmitter design – Solution (1)

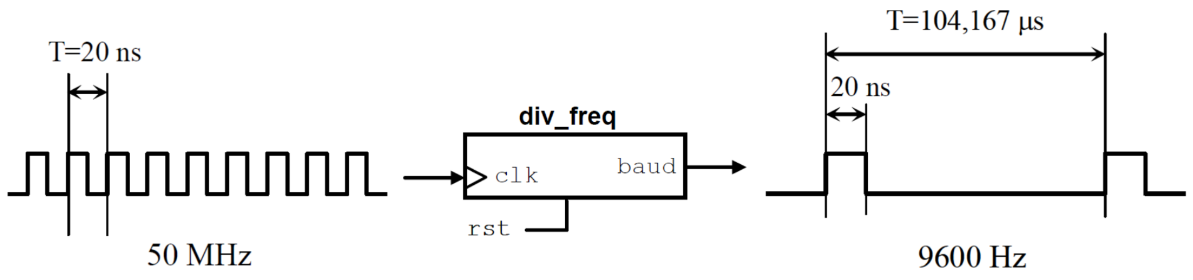
Exercise 1: Implement a shift register in VHDL and validate its functionality.

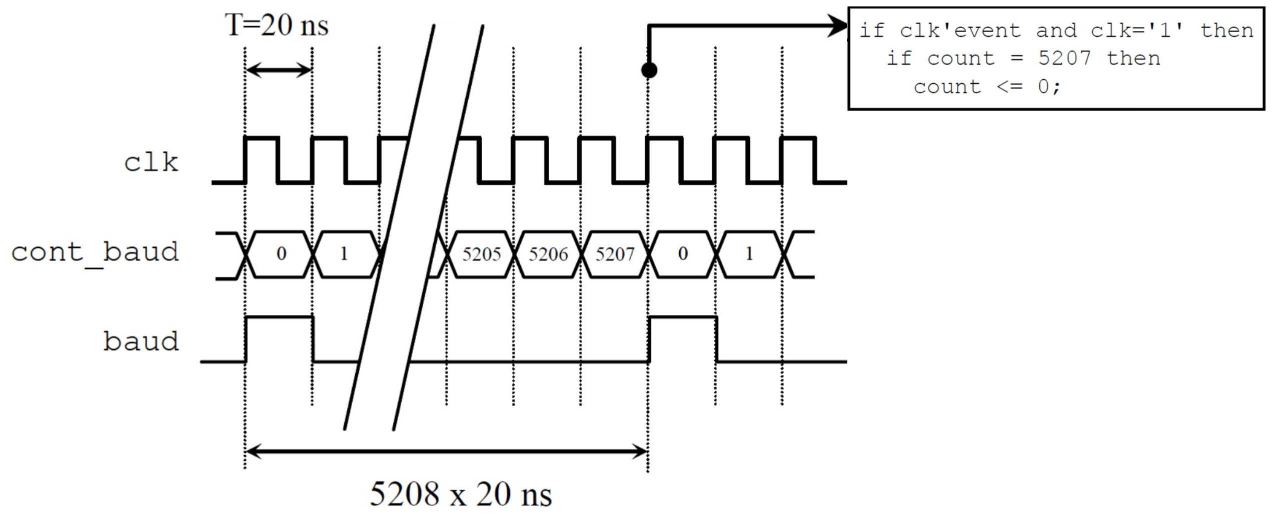


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY load_shift IS
   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 load_shift;

ARCHITECTURE behavior oF load_shift IS
   SIGNAl tmp_reg: STD_LOGIC_VECTOR (7 downto 0);
BEGIN
   PROCESS (rst,clk)
   BEGIN
      IF (rst = '1') THEN
         tmp_reg <= (OTHERS=>'0');
         t_data_out <= '0';
      ELSIF (clk'EVENT and clk = '1') THEN
         IF (load = '1') THEN
            tmp_reg <= t_data_in;
         ELSIF (shift = '1') THEN
            t_data_out <= tmp_reg(0);
            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);
         END IF;
      END IF;

   END PROCESS;
END behavior;
```

Transmitter design – Solution (2)

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

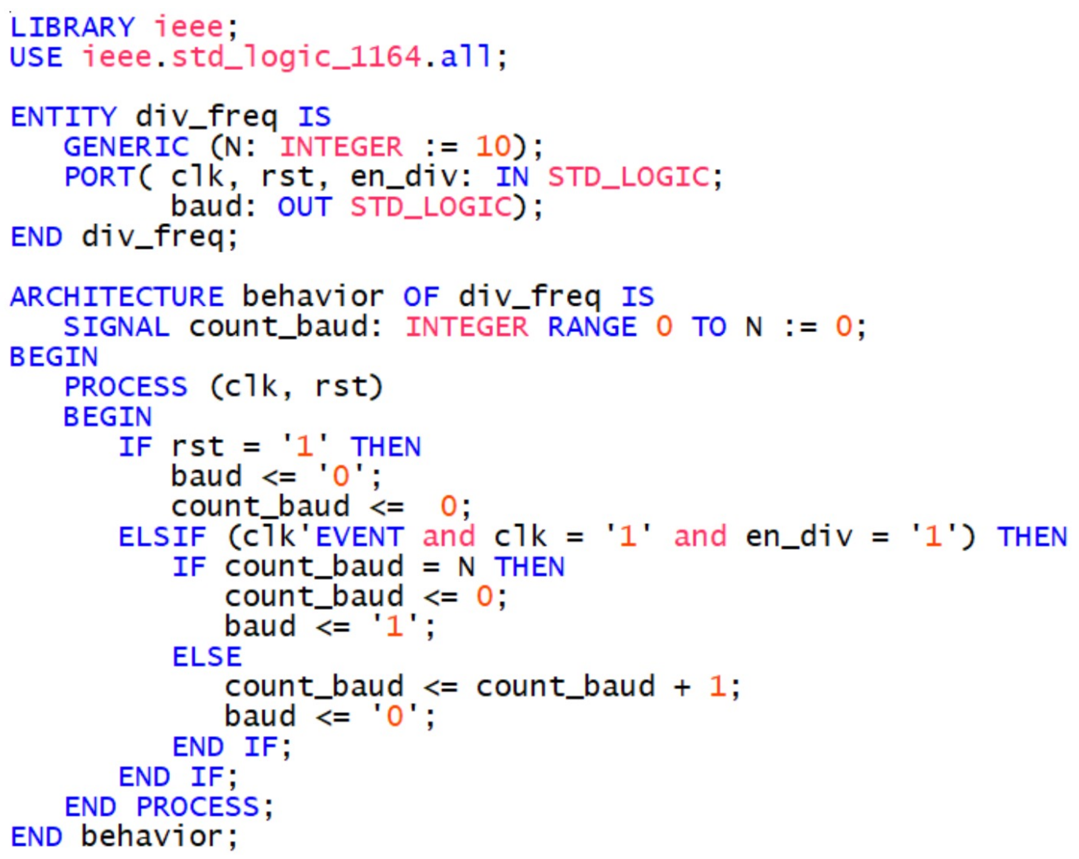




```
if clk'event and clk='1' then
  if count = 5207 then
    count <= 0;
```

Transmitter design – Solution (2)

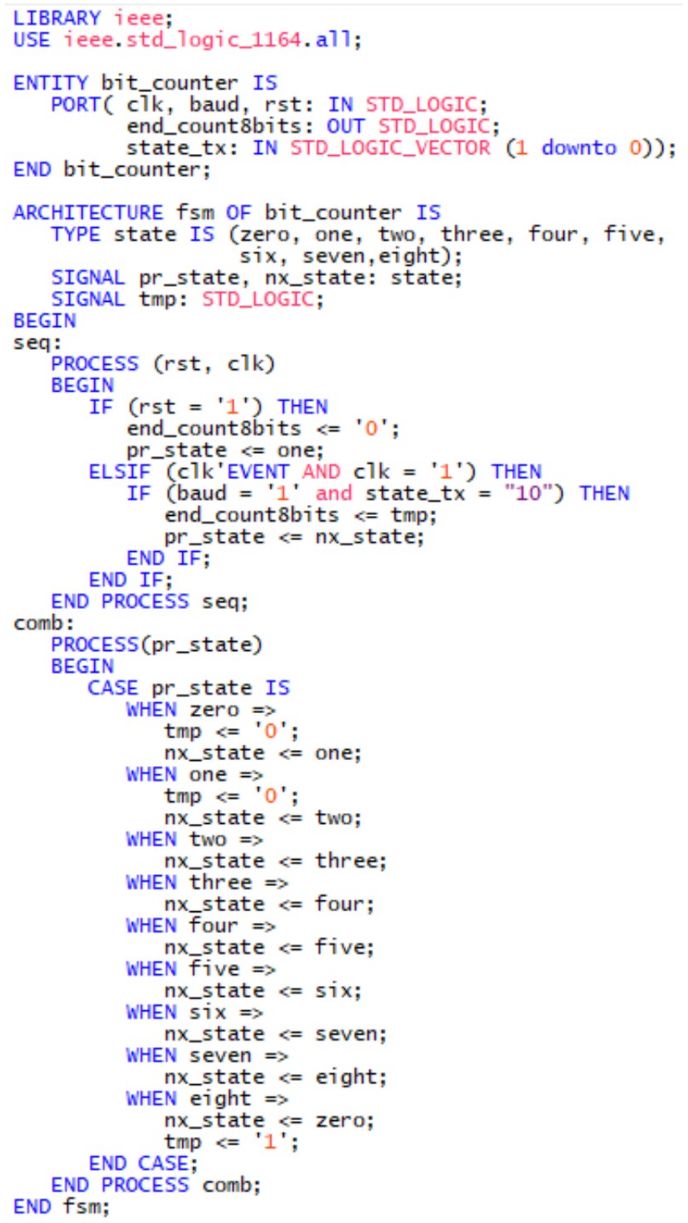
Exercise 2: Implement the frequency divider in VHDL and test its functionality.



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

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

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

Transmitter design – Solution (2b)

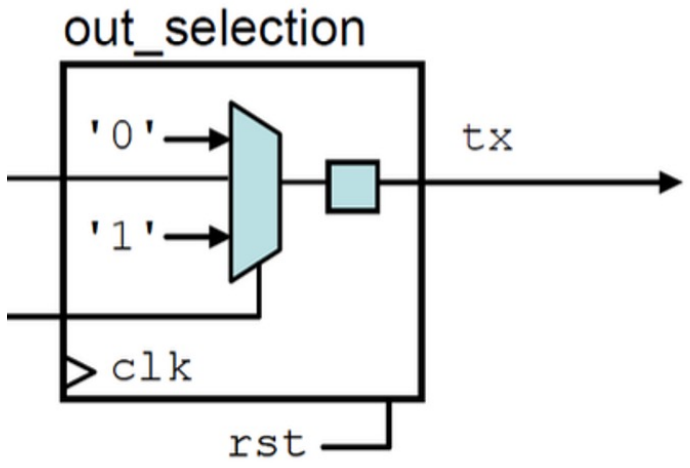
Ex. 2b: If your design has other blocks (**bit counter**), implement and test them in VHDL.



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

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

ARCHITECTURE fsm OF bit_counter IS
   TYPE state IS (zero, one, two, three, four, five,
                  six, seven,eight);
   SIGNAL pr_state, nx_state: state;
   SIGNAL tmp: STD_LOGIC;
BEGIN
seq:
   PROCESS (rst, clk)
   BEGIN
      IF (rst = '1') THEN
         end_count8bits <= '0';
         pr_state <= one;
      ELSIF (clk'EVENT AND clk = '1') THEN
         IF (baud = '1' and state_tx = "10") THEN
            end_count8bits <= tmp;
            pr_state <= nx_state;
         END IF;
      END IF;
   END PROCESS seq;
comb:
   PROCESS(pr_state)
   BEGIN
      CASE pr_state IS
         WHEN zero =>
            tmp <= '0';
            nx_state <= one;
         WHEN one =>
            tmp <= '0';
            nx_state <= two;
         WHEN two =>
            nx_state <= three;
         WHEN three =>
            nx_state <= four;
         WHEN four =>
            nx_state <= five;
         WHEN five =>
            nx_state <= six;
         WHEN six =>
            nx_state <= seven;
         WHEN seven =>
            nx_state <= eight;
         WHEN eight =>
            nx_state <= zero;
            tmp <= '1';
      END CASE;
   END PROCESS comb;
END fsm;
```

Transmitter design – Solution (2b)

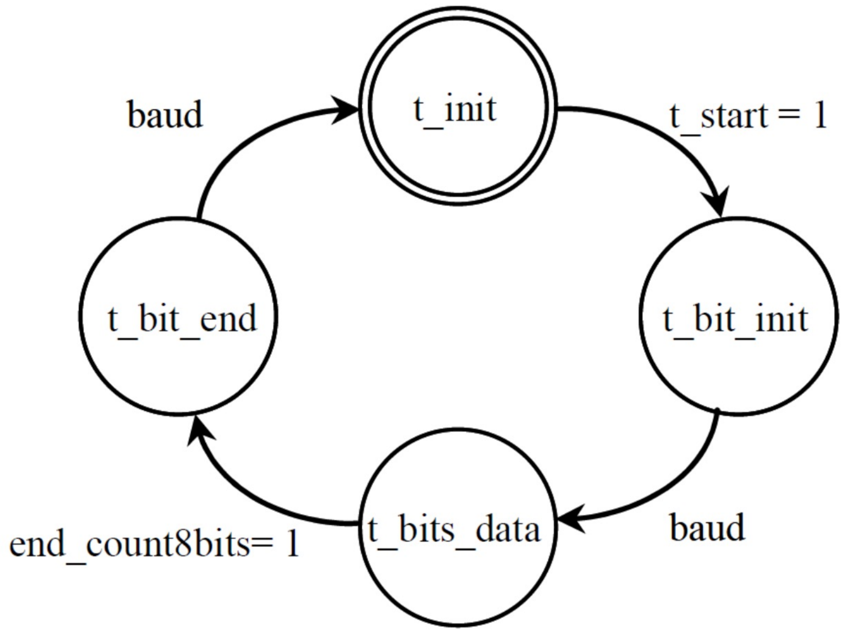
Ex. 2b: If your design has other blocks (**out_selection**), implement and test them in VHDL.



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

ENTITY out_selection IS
   PORT(
          clk, rst , in_data: IN STD_LOGIC;
          state_tx: IN STD_LOGIC_VECTOR (1 downto 0);
          tx: OUT STD_LOGIC);
END out_selection;

ARCHITECTURE behavior OF out_selection IS

BEGIN
   PROCESS(clk, rst)
   BEGIN
      IF (rst = '1') THEN
         tx <= '1';
      ELSIF (clk'EVENT and clk='1') THEN
         CASE state_tx IS
            WHEN "00" =>
               tx <= '1';
            WHEN "01" =>
               tx <= '0';
            WHEN "10" =>
               tx <= in_data;
            WHEN "11" =>
               tx <= '1';
         END CASE;
      END IF;
   END PROCESS;
END behavior;
```

Transmitter design – Solution (3)

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

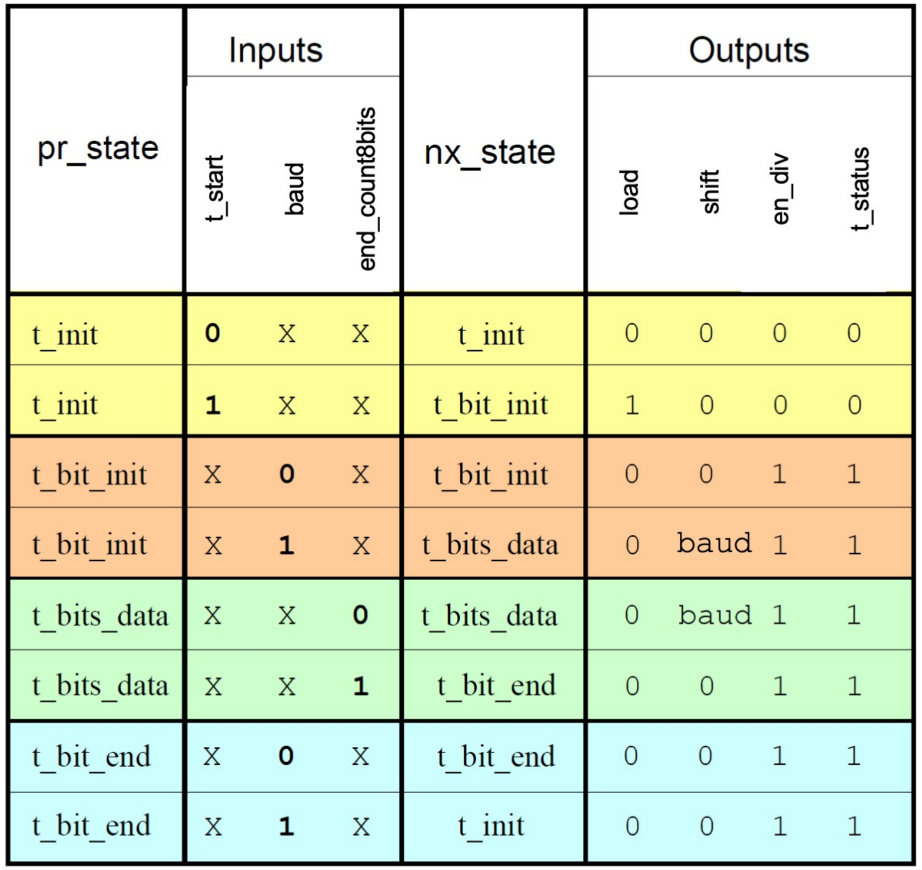


baud: one clock pulse.

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

Transmitter design – Solution (3)

Exercise 3: Design the transmission control block and the bit counter FSMs, define the number of states and their truth tables.



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

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.