Exercise 4:

In this exercise you will learn how to implement a finite state machine (FSM) in VHDL. As examples, the two most commonly used types of FSMs were selected: the Moore and the Mealy state machines (it is assumed that the differences in the behavior of both types are already known). There a several ways to code a state machine in VHDL. Most examples found on the internet or in the literature use a 2-process model, some even use 3 processes. The most common approach at our institute is a 1-process model, therefore this model is explained in this exercise.

a) Mealy Machines

In the following code, you will find a simple example of how to implement a Mealy FSM with 1 process. The first thing you must do is declare a new type for your states and a signal of this type.

```
type fsm_states is (STATE_ONE, STATE_TWO);
signal state_fsm : fsm_states := STATE_ONE;
```

Now, you have a signal state_fsm with the possible values STATE_ONE or STATE_TWO. In the next step, you have to calculate your state transitions. First, you must read out the actual state of your signal *state_fsm* with a case statement. Then, you can assign a new state to this signal depending on the conditions of your FSM.

```
state_transitions : process (clk)
begin
      if rising_edge(clk) then
            if reset = '1' then
                  state_fsm <= STATE_ONE;
            else
                  case state_fsm is
                        when STATE_ONE =>
                              if condition then
                                    state_fsm <= STATE_TWO;
                              end if;
                        when STATE_TWO =>
                              if condition then
                                    state_fsm <= STATE_ONE;
                              end if;
                  end case;
            end if;
      end if;
end process state_transitions;
```

After you have calculated your state_transitions, you can assign your outputs with concurrent statements outside the process. This concurrent statement describes a simple multiplexer. Remember that the outputs of a Mealy machine depend on the state and on the inputs.

```
o_x <= '0' when ( )
          else '1';
```

Your task is to implement a Mealy machine from the given state diagram on the next page.

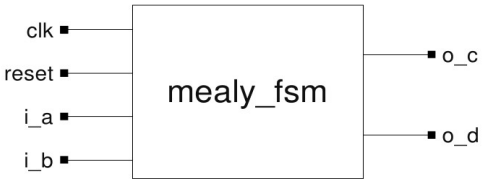

Figure 1: mealy_fsm

Port declaration:

- *clk* : in std_logic
- *reset* : in std_logic
- *i_a* : in std_logic
- *i_b* : in std_logic
- *o_c* : out std_logic
- *o_d* : out std_logic

Hint: There is only one testbench, which tests both state machines. If you compare the state machines with each other in the waveform of your simulation, they should always be in the same state – with a different behavior at the outputs!

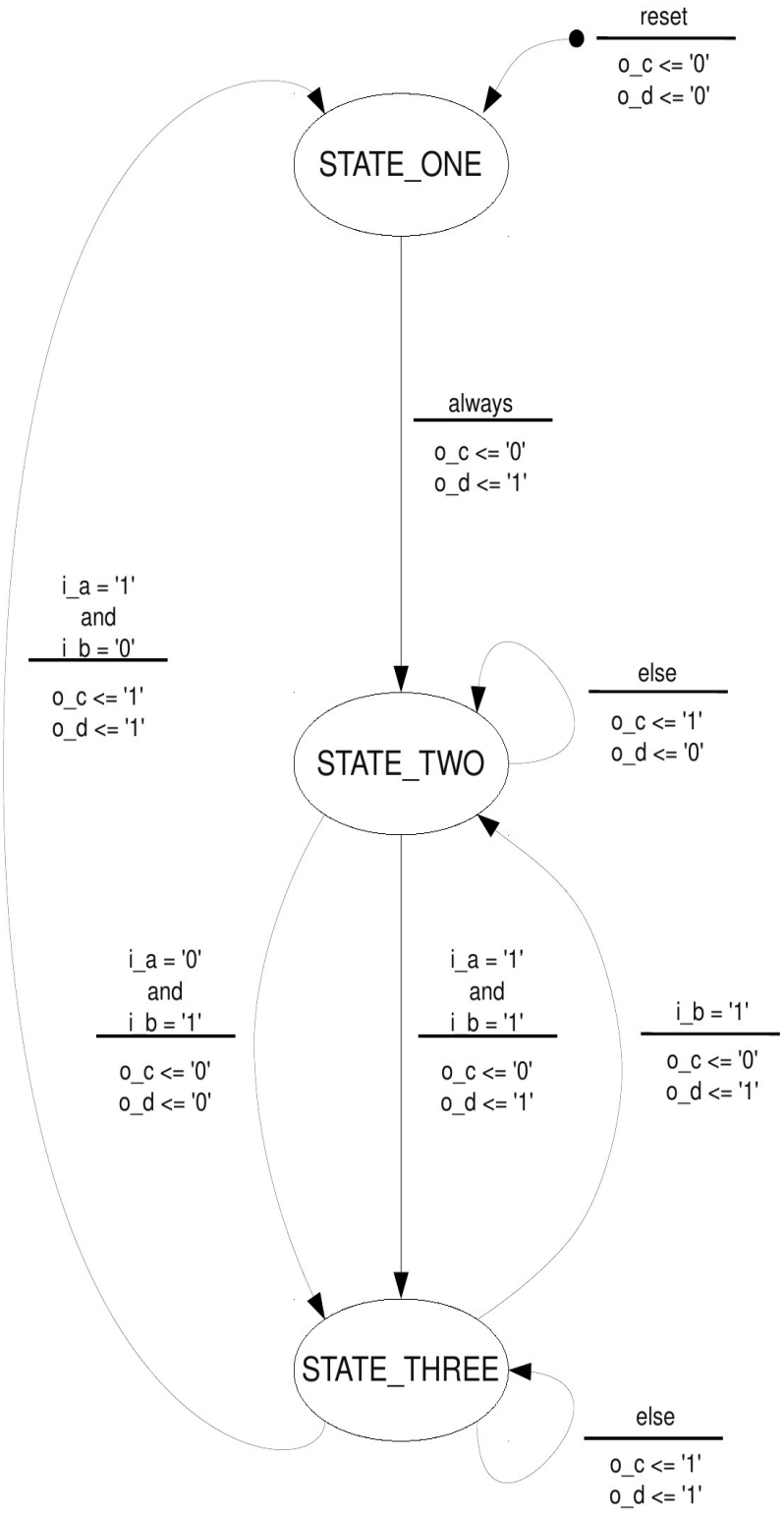

Figure 2: State chart of Mealy machine

b) Moore Machines

Now, you will implement a Moore machine of a given state diagram. This time, no example is given. Considering the differences in the design of Moore state machines compared to the Mealy design, it should be easy to change the given design from Mealy to Moore.

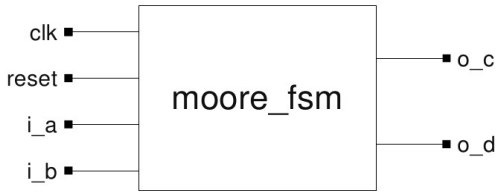

Figure 3: moore_fsm

Port declaration:

- *clk* : in std_logic
- *reset* : in std_logic
- *i_a* : in std_logic
- *i_b* : in std_logic
- *o_c* : out std_logic
- *o_d* : out std_logic

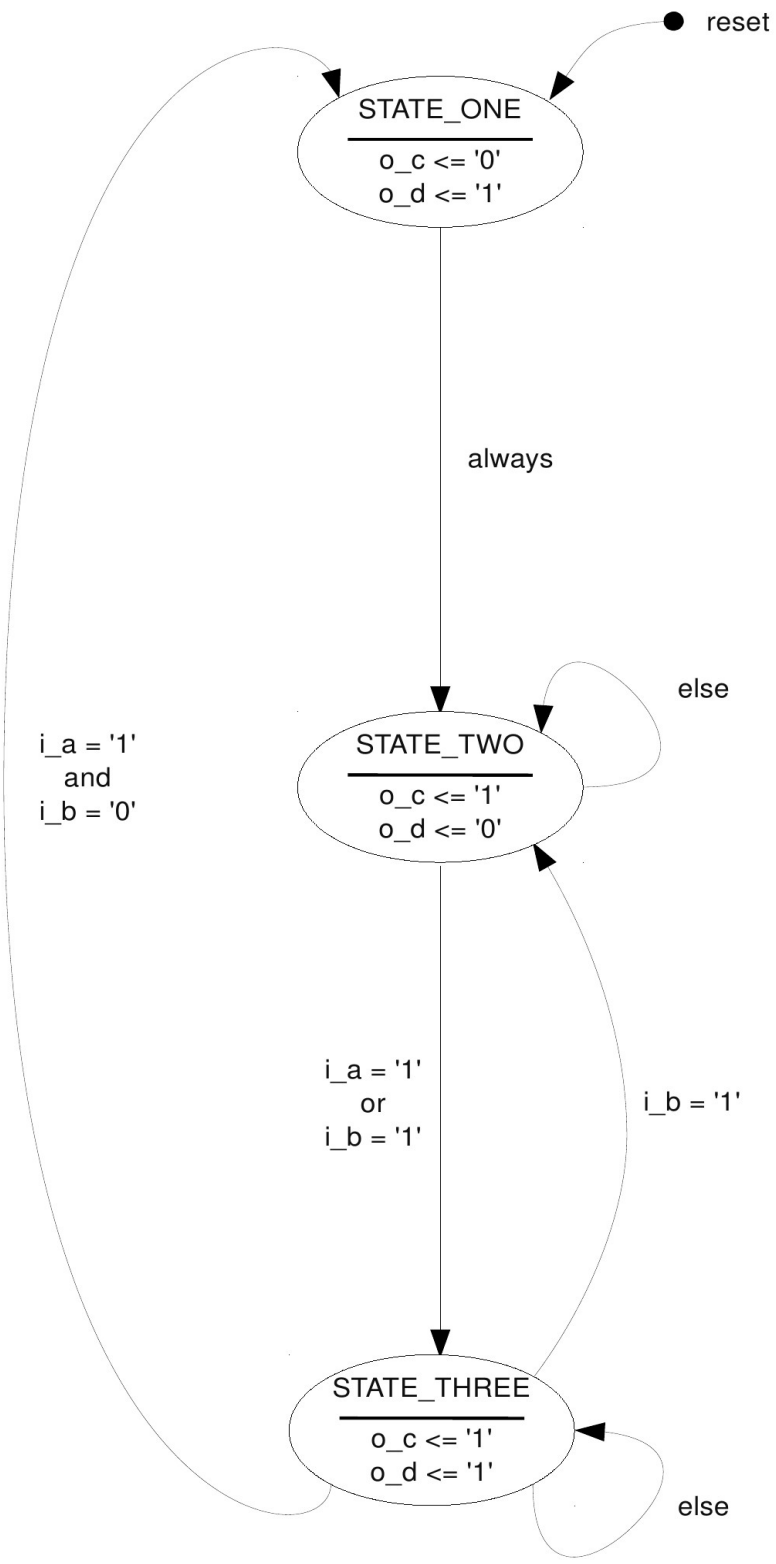

Figure 4: State chart of Moore machine

c) Testbenches (*game_states*)

As you have seen in the previous tasks, testing through simulation is an important step in the design process. At design time, the best choice is to test each module separately. This allows the designer to avoid problems and time consuming debugging after all modules have been combined into a larger design, because it is less difficult to find an error in the smaller, independent and less complex sub-modules.

Hardware description languages (HDLs) allow simulation at various levels, ranging from behavioral simulation, where only the logic behavior is simulated, up to a post-route simulation, where the HDL model is combined with technology libraries of the target architecture (in our case the FPGA) to get information on delays and rise and fall times of gates or signal lines. In this lab, we will only use a behavioural simulation of our VHDL models.
Unlike in the earlier lessons, where the test benches were provided, it is your task to write a test bench for a given VHDL file.

The unit under test will be the module *game_states*, a sub-module from the block *game_engine*. It is responsible for controlling the game flow, counting points and changing the serving player. It consists of two Mealy state machines. The state diagrams for both are provided for clarity.

The first state machine has four states:

- *SERVE* : In this state, the ball is bound to the serving player's paddle. Paddle movement is allowed. The state is exited when the serve key is pressed.
- *PLAYING* : In this state, ball and paddle movement is allowed. It is exited when one player scores a point.
- *WAITING* : After a scored point, the FSM enters this state and remains there for a period of time. Ball and paddle movement is disabled, but the ball remains at its previous location to give the players a chance to see the scored point. When exiting this state, there are two possibilities. If no player has reached the point limit to win the game, the state machine returns to the *SERVE* state. Otherwise it changes to the *WON* state.
- *WON* : This state is only entered when one player reaches the score limit. In this state ball and paddle movement remain disabled and it can only be exited by a reset of the module *game_engine*.

The second FSM has only two states:

- *PLAYER_1* : Player 1 has the right to serve. The state is exited after five serves.
- *PLAYER_2* : Player 2 has the right to serve. The state is exited after five serves.

The first thing you should do is examine the state diagrams and the VHDL code to fully understand the function of the module.
Then you can begin writing the testbench. Like in the previous lessons, the stimulus data should be read from a text file with the file ending *.vec*. It would be a good idea to look at previous testbenches to learn how to build them and how read data from files.
The testbench should move the state machine through all its transitions at least once. For each transition, you must ensure that it only occurs when all transition conditions are true.

Hint: To keep the simulated time low, you should change the value of the constant *C_WAIT_TIME*.