lea.schoenberger [☺] tu-dortmund.de
benjamin.glaeser [☺] tu-dortmund.de
niklas.ueter [☺] tu-dortmund.de
mikail.yayla [☺] tu-dortmund.de

Exercise for the lecture
Embedded Systems
Wintersemester 17/18

Exercise Sheet 4

(10 Points)

Lab exercises for the period from Wednesday, 15th November 2017

The lab exercises take place at room OH16/U08. The exercise sheets will be solved during the exercise sessions.

For this exercise sheet, we use the virtual machine **ES-FPGA**.

4.1 Reaction Game with State Machine (5 Points)

In this exercise, you will implement a reaction game for two players according to the state machine illustrated in Figure 1. Since Visual State does not support VHDL as target language, you will have to do the implementation manually.

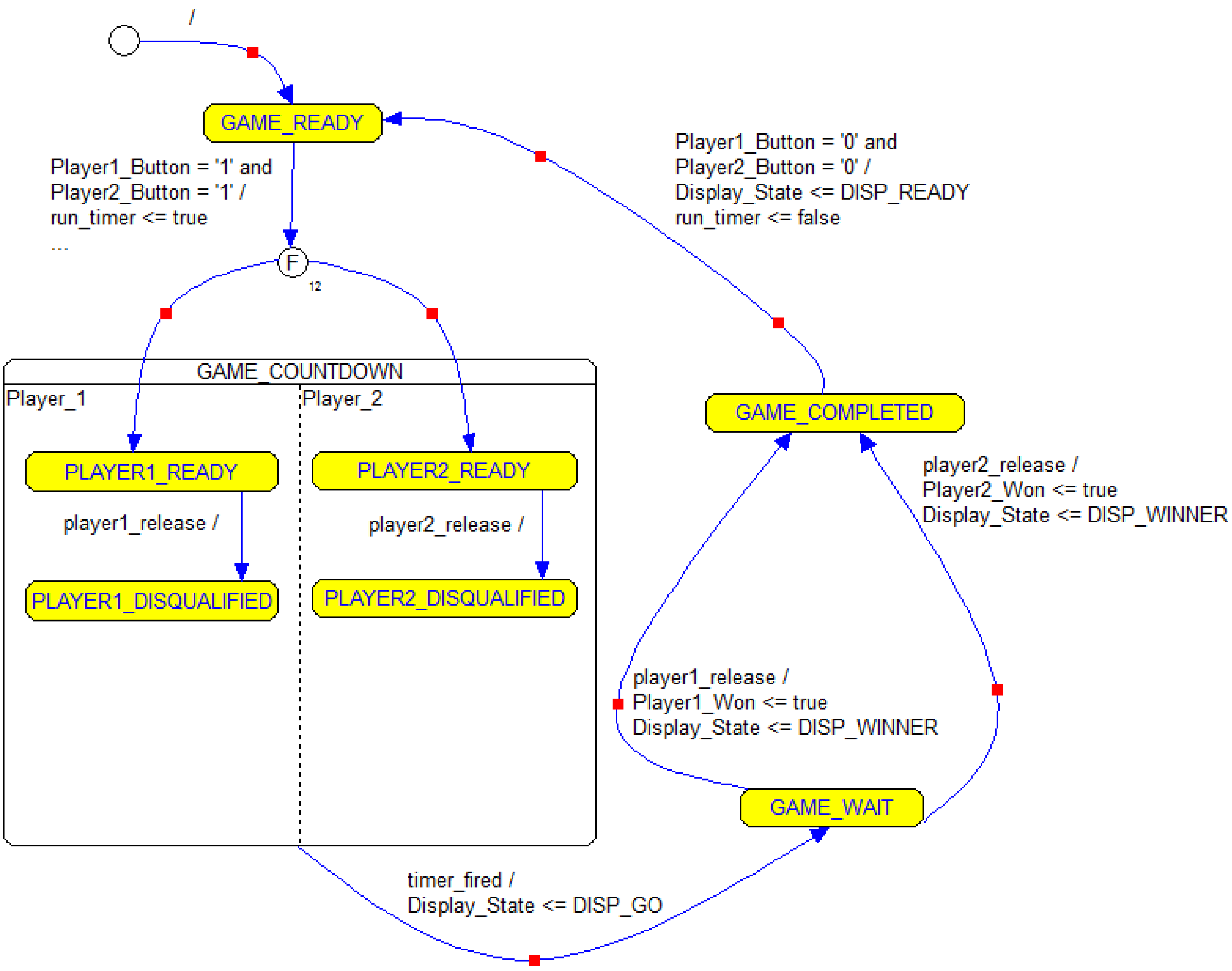

Figure 1: State machine of the game

Rules:

- There exist two players to each of which a button is assigned.
- A round of the game is started, when both players keep their buttons pressed.
- After a random time interval (3 – 10 seconds), "GO" is displayed and the players must release their buttons as quickly as possible.
- The first player who releases his or her button wins.
- If a player releases his or her button before "GO" is displayed, he or she is disqualified. This does not imply that the other player automatically wins, since he or she could also be disqualified.
- Another round can be started only after both players released their buttons.

Assignment: Implement the state machine describing the game in the provided file `GameLogic`. The transition from `GAME_READY` to `GAME_COUNTDOWN` has already been implemented by means of illustration. To implement the other state transitions, you have to request the triggers activating the respective state transitions in the branches of the `case` instruction using `if ... then ... end if;` and to then execute the respective actions and state transitions. If only a signal name as, e.g., `player1_release` is given in the diagram, this is a boolean signal which can be directly formulated as a condition of an `if`-instruction.

A VHDL example can be found on the last page of this exercise sheet.

4.2 Game Logic Improvement (5 Points)

Having implemented the given state chart, two problems become evident:

- The players can cheat - if one player releases his or her button during the waiting period and subsequently presses it again, he or she is disqualified but is still able to win the game.

  Make sure that a disqualified player is not accepted as the game's winner.

- If both players are disqualified, the game still waits until the countdown is finished and, furthermore, the game can be restarted no earlier than after one player presses his or her key again (if you fixed the previous issue, it can be started never again). If both players are disqualified, the game should be aborted.

  Implement a game abortion for the case that both players are disqualified.

General information: An overview about the exercise sessions as well as further information can be found on `https://ls12-www.cs.tu-dortmund.de/daes/lehre/lehrveranstaltungen/wintersemester-20172018/es-1718.html`. The exercise sheets will usually be published on the course website on Mondays and will be solved during the respective exercise sessions. The exercises are divided into two parts, in each of which at least 50% of the points must be achieved in order to receive the exam admission.

VHDL overview

```vhdl
-- the entity should have at least one input called "clock" and one output called "output"

architecture Behavioral of Example2 is
  type demo_enum is (ENUM_A, ENUM_B, ENUM_C); -- enumeration types are useful for state machines
  signal state: demo_enum := ENUM_A;          -- can be only ENUM_A, ENUM_B or ENUM_C

  signal bool_sig: boolean;           -- can be only "true" or "false"
  signal sl_sig  : std_logic := '0';  -- can be '0' or '1' (or 'U'ndefined or...)

  signal toggle     : boolean := false; -- some more example signals
  signal delay      : integer;
  signal another_one: std_logic;
begin

  process(Clock)  -- processes are blocks of instructions that are always executed
  begin           -- if the signals given in () change
    if rising_edge(Clock) then  -- do something only if the clock changes from 0->1
      -- everything here is executed once at each rising clock edge
      -- more precisely, in the order in which it is listed here
      -- BUT: The "visible" signal values change no earlier after the process is finished

      toggle <= not toggle; --  "toggle" changes between true and false at each clock pulse
      if toggle then  -- boolean values directly appear in if-statements
        sl_sig <= not sl_sig;      -- sl_sig changes between 0 and one at each second clock pulse
      else
        -- nested if-statements can also be used:
        if enum_sig = ENUM_B then  -- a test for equality is also a valid expression in an if-statement
                                   -- (for inequal use "/=" instead of "=" used here)
          bool_sig <= '0';
        else
          bool_sig <= '1';
        end if;  -- VHDL blocks nearly always end with "end <something>"
      end if;

      -- useful for state machines: case-when blocks (Java: switch/case)
      case state is
        when ENUM_A =>
          -- commands that are executed when state = ENUM_A
          if delay /= 0 then  -- (a nonsense example without detailed description)
            delay <= delay - 1;
          else
            state <= ENUM_B;
          end if;

        when ENUM_B => -- (exceptionally no "end" before the next "when")
          -- commands that are executed when state = ENUM_B
          output <= sl_sig;
          state  <= ENUM_C;

        when ENUM_C =>
          -- commands that are executed when state = ENUM_C
          delay <= 100;
          state <= ENUM_A;

      end case; -- the complete case block must have an "end"

      another_one <= '1'; -- simple assignments can also be used in processes

    end if;
  end process;    -- processes are finished with "end"
end Behavioral;
```