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

Übung zur Vorlesung
Eingebettete Systeme
Wintersemester 17/18

Übungsblatt 4

(10 Punkte)

Präsenzaufgaben zur Bearbeitung ab Mittwoch, 15. November 2017

Die praktischen Übungen finden im Raum OH16/U09 statt. Die Bearbeitung der Aufgaben erfolgt in den Übungen.

Für die Bearbeitung dieses Blatts wird die virtuelle Maschine **ES-FPGA** verwendet.

4.1 Ein Reaktionsspiel mit Zustandsautomat (5 Punkte)

In dieser Aufgabe implementieren Sie ein Reaktionsspiel für zwei Personen anhand des in Abbildung 1 gezeigten Zustandsautomaten. Da Visual State VHDL nicht als Zielsprache unterstützt, ist etwas Tipparbeit zur Umsetzung notwendig.

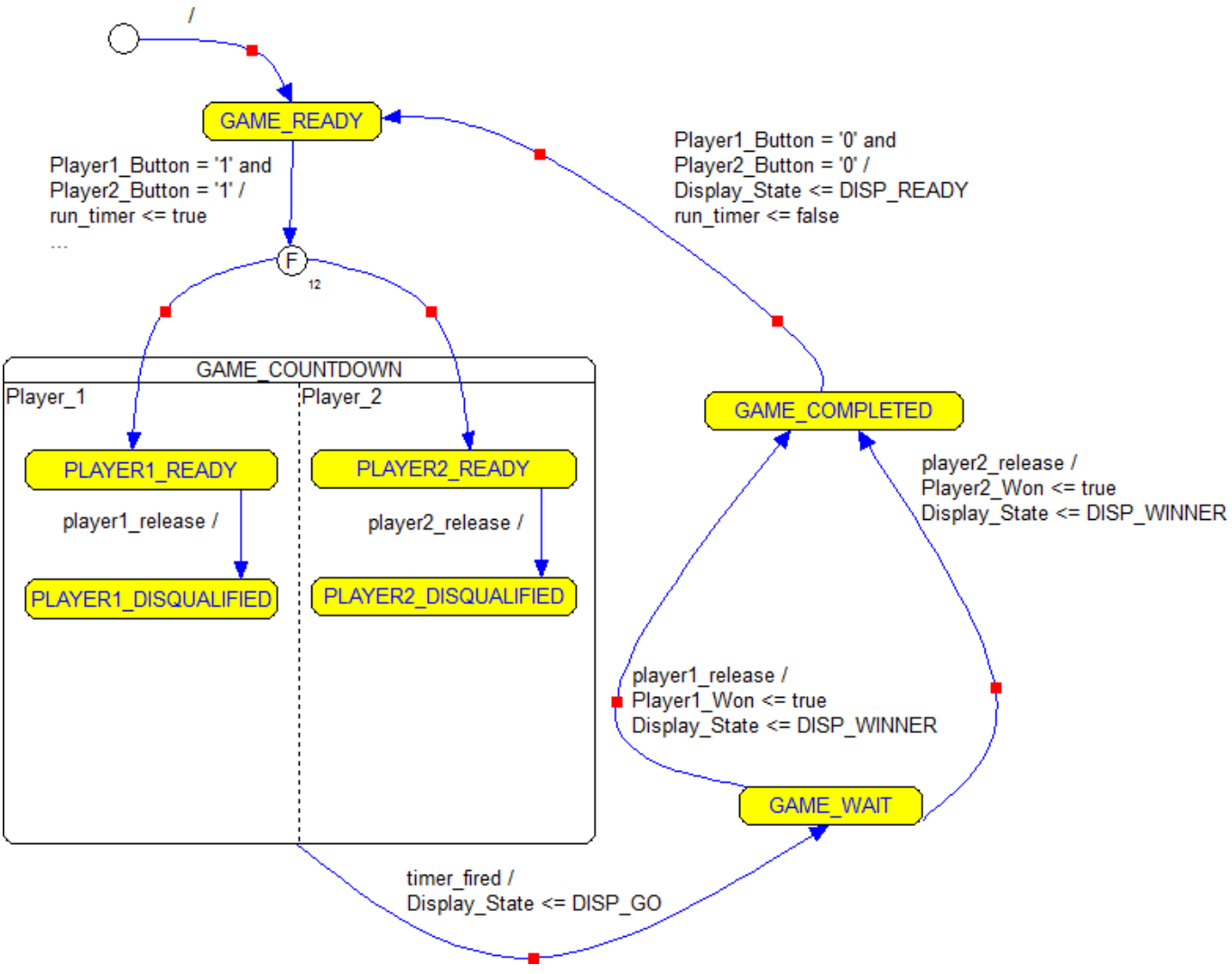

Abbildung 1: Zustandsdiagramm des Spiels

Regeln:

- Es gibt zwei Spieler. Jedem Spieler wird eine Taste zugewiesen.
- Eine Spielrunde wird gestartet, wenn beide Spieler ihre Taste gedrückt halten.
- Nach einer zufälligen Wartezeit (ca. 3-10 Sekunden) wird “GO” auf dem Display angezeigt und die Spieler müssen versuchen, schnellstmöglich ihre Taste loszulassen.
- Derjenige Spieler, der seine Taste zuerst loslässt, gewinnt.
- Sollte ein Spieler seine Taste loslassen, bevor “GO” angezeigt wurde, ist er disqualifiziert. Dies bedeutet keinen automatischen Sieg für den anderen Spieler, da er ebenfalls disqualifiziert werden könnte.
- Eine weitere Runde kann erst gestartet werden, wenn beide Spieler ihre Taste losgelassen haben.

Aufgabe: Implementieren Sie den Zustandsautomaten für das Spiel in der Datei `GameLogic` aus der Vorgabe. Der Übergang von `GAME_READY` zu `GAME_COUNTDOWN` wurde bereits in der Vorlage als Beispiel mitgeliefert. Zur Implementierung der weiteren Zustandsübergänge müssen Sie in den Zweigen der `case`-Anweisung mittels `if ... then ... end if;` die notwendigen Trigger für jeden Zustandsübergang abfragen und die dadurch ausgelösten Aktionen und den Wechsel des aktuellen Zustands ausführen. Wenn im Diagramm nur ein Signalname wie z.B. `player1_release` genannt ist, handelt es sich um ein Signal vom Typ Boolean, welches direkt als Bedingung in eine `if`-Anweisung geschrieben werden kann.

Ein VHDL-Codebeispiel zur Syntaxdemonstration findet sich auf der letzten Seite dieses Übungsblatts.

4.2 Verbesserungen der Spielelogik (5 Punkte)

Wenn Sie das vorgegebene Zustandsdigramm nachimplementieren, gibt es zwei Probleme:

- Die Spieler können mogeln - wenn ein Spieler während der Wartezeit seine Taste loslässt und wieder drückt, wird er als disqualifiziert markiert, kann aber trotzdem noch gewinnen.

  Sorgen Sie dafür, dass ein disqualifizierter Spieler nicht mehr als Gewinner akzeptiert wird.

- Wenn beide Spieler disqualifiziert wurden, wartet das Spiel trotzdem noch auf den Ablauf des Timers und eine neue Runde ist erst dann wieder startbar, wenn einer der Spieler seine Taste nochmals betätigt – oder evtl. gar nicht mehr startbar, wenn Sie den vorherigen Punkt bearbeitet haben. Da bei zwei disqualifitierten Spielern auch keiner mehr gewinnen kann, sollte an dieser Stelle besser das Spiel abgebrochen werden.

  Implementieren Sie einen Spielabbruch für den Fall, dass beide Spieler disqualifiziert wurden.

VHDL-Kurzreferenz

```vhdl
-- die Entity habe mindestens einen Eingang namens "Clock" und einen Ausgang "output"

architecture Behavioral of Beispiel2 is
  type demo_enum is (ENUM_A, ENUM_B, ENUM_C); -- Aufzählungstypen sind nett für Statemachines
  signal state: demo_enum := ENUM_A;          -- kann nur ENUM_A, ENUM_B oder ENUM_C annehmen

  signal bool_sig: boolean;           -- kann nur "true" oder "false" sein
  signal sl_sig  : std_logic := '0';  -- kann '0' oder '1' sein (oder 'U'ndefiniert oder...)

  signal toggle     : boolean := false; -- ein paar weitere Beispielsignale
  signal delay      : integer;
  signal another_one: std_logic;
begin

  process(Clock)  -- Prozesse sind Blöcke von Anweisungen, die immer ausgeführt
  begin           -- werden, wenn die in () angegebenen Signale sich ändern
    if rising_edge(Clock) then  -- nur bei den 0->1-Wechseln des Takts was machen
      -- alles hier drin wird bei jeder steigenden Taktflanke einmal ausgeführt
      -- und zwar in der Reihenfolge, in der es hier steht
      -- ABER: Die "sichtbaren" Werte von Signalen ändern sich erst, wenn
      --       der Prozess komplett durchgelaufen ist

      toggle <= not toggle; -- bei jedem Taktpuls wechselt "toggle" hiermit zwischen true und false
      if toggle then  -- boolean-Werte taugen direkt als Ausdruck im if
        sl_sig <= not sl_sig;      -- sl_sig wechselt bei jedem zweiten Taktpuls zwischen 0 und 1
      else
        -- verschachtelte ifs gehen natürlich auch:
        if enum_sig = ENUM_B then  -- ein Gleichheitstest ist auch ein gültiger Ausdruck im if
                                   -- (Ungleich wäre "/=" statt des "=" hier)
          bool_sig <= '0';
        else
          bool_sig <= '1';
        end if;  -- VHDL-Blöcke enden fast immer mit "end <irgendwas>"
      end if;

      -- Für Statemachines sehr praktisch: Case-When-Blöcke (Java: switch/case)
      case state is
        when ENUM_A =>
          -- Anweisungen, die ausgeführt werden, wenn state = ENUM_A ist
          if delay /= 0 then  -- (irgendwas sinnloses als Beispiel, ohne Detailbeschreibung)
            delay <= delay - 1;
          else
            state <= ENUM_B;
          end if;

        when ENUM_B => -- (ausnahmsweise mal kein "end" vor dem nächsten when)
          -- Anweisungen, die ausgeführt werden, wenn state = ENUM_B ist
          output <= sl_sig;
          state  <= ENUM_C;

        when ENUM_C =>
          -- Anweisungen, die ausgeführt werden, wenn state = ENUM_C ist
          delay <= 100;
          state <= ENUM_A;

      end case; -- aber der komplette case-Block bekommt natürlich wieder ein end

      another_one <= '1'; -- einfache Zuweisungen gehen im Prozess natürlich auch

    end if;
  end process;    -- und auch Prozesse werden mit "end" abgeschlossen
end Behavioral;
```