

Übungsblatt 4

(10 Punkte)

Präsenzaufgaben zur Woche ab Montag, 14. November 2016

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. Zum Glück unterstützt Visual State VHDL nicht als Zielsprache, daher ist etwas Tipparbeit zur Umsetzung notwendig.

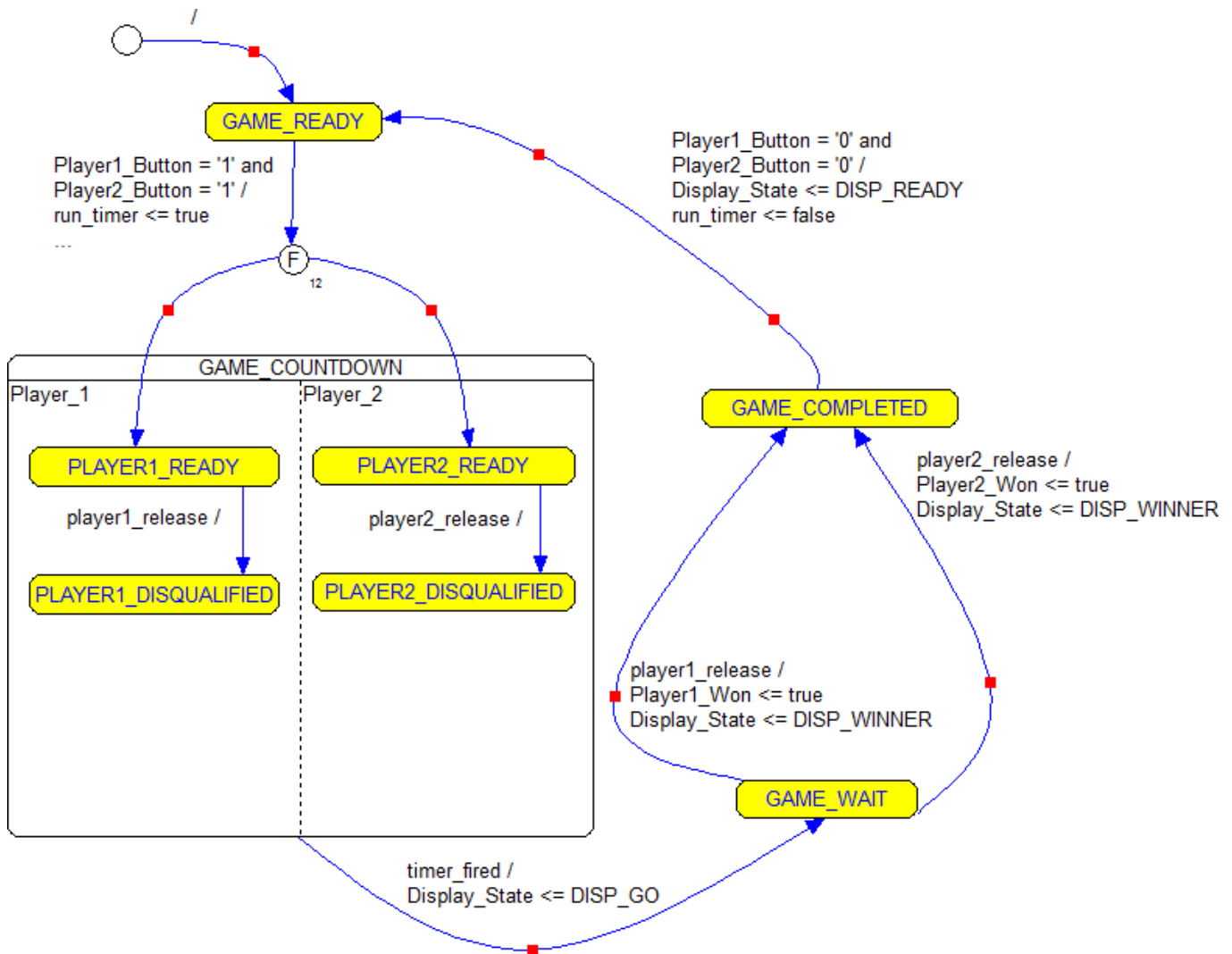


Abbildung 1: Zustandsdiagramm des Spiels

Regeln:

- Jedem Spieler ist eine Taste zugewiesen, die untere (*BTND*) für Spieler 1, die rechte (*BTNR*) für Spieler 2.
- 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, so schnell wie möglich ihre Taste loszulassen
- Der erste Spieler, der seine Taste als erster loslässt gewinnt.
- Sollte ein Spieler seine Taste loslassen, bevor "GO" angezeigt wurde, ist er disqualifiziert. Dies bedeutet keinen autoamtischen 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 hoffentlich instruktives VHDL-Codebeispiel zur Syntaxdemonstration findet sich auf der letzten Seite dieses Übungsblatts.

Tip: Die LED-Leiste über den Schaltern zeigt für einfachere Fehlersuche die Ausgänge des `GameLogic`-Moduls an. Von links nach rechts zeigen die ersten vier LEDs den aktuellen Wert von `Display_State` (`DISP_READY`, `DISP_COUNTDOWN`, `DISP_GO`, `DISP_WINNER`) und die letzten vier LEDs die Signale `Player1_Won`, `Player2_Won`, `Player1_Disq`, `Player2_Disq`.

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 disqualifizierten 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, die zweite

(wieder in Form eines intensiv kommentierten, aber unvollständigen Beispielcodes)

```
1  -- die Entity habe mindestens einen Eingang namens "Clock" und einen Ausgang "output"
2
3  architecture Behavioral of Beispiel2 is
4      type demo_enum is (ENUM_A, ENUM_B, ENUM_C); -- Aufzählungstypen sind nett für State machines
5      signal state: demo_enum := ENUM_A;          -- kann nur ENUM_A, ENUM_B oder ENUM_C annehmen
6
7      signal bool_sig: boolean;                  -- kann nur "true" oder "false" sein
8      signal sl_sig  : std_logic := '0';         -- kann '0' oder '1' sein (oder 'U' undefiniert oder...)
9
10     signal toggle   : boolean := false; -- ein paar weitere Beispielsignale
11     signal delay    : integer;
12     signal another_one: std_logic;
13 begin
14
15     process(Clock) -- Prozesse sind Blöcke von Anweisungen, die immer ausgeführt
16     begin        -- werden, wenn die in () angegebenen Signale sich ändern
17         if rising_edge(Clock) then -- nur bei den 0->1-Wechseln des Takts was machen
18             -- alles hier drin wird bei jeder steigenden Taktflanke einmal ausgeführt
19             -- und zwar in der Reihenfolge, in der es hier steht
20             -- ABER: Die "sichtbaren" Werte von Signalen ändern sich erst, wenn
21             -- der Prozess komplett durchgelaufen ist
22
23             toggle <= not toggle; -- bei jedem Taktpuls wechselt "toggle" hiermit zwischen true und false
24             if toggle then -- boolean-Werte taugen direkt als Ausdruck im if
25                 sl_sig <= not sl_sig; -- sl_sig wechselt bei jedem zweiten Taktpuls zwischen 0 und 1
26             else
27                 -- verschachtelte ifs gehen natürlich auch:
28                 if enum_sig = ENUM_B then -- ein Gleichheitstest ist auch ein gültiger Ausdruck im if
29                     -- (Ungleich wäre "/=" statt dem "=" hier)
30                     bool_sig <= '0';
31                 else
32                     bool_sig <= '1';
33                 end if; -- VHDL-Blöcke enden fast immer mit "end <irgendwas>"
34             end if;
35
36             -- Für State machines sehr praktisch: Case-When-Blöcke (Java: switch/case)
37             case state is
38                 when ENUM_A =>
39                     -- Anweisungen, die ausgeführt werden wenn state = ENUM_A ist
40                     if delay /= 0 then -- (irgendwas sinnloses als Beispiel, ohne Detailbeschreibung)
41                         delay <= delay - 1;
42                     else
43                         state <= ENUM_B;
44                     end if;
45
46                 when ENUM_B => -- (ausnahmsweise mal kein "end" vor dem nächsten when)
47                     -- Anweisungen, die ausgeführt werden wenn state = ENUM_B ist
48                     output <= sl_sig;
49                     state <= ENUM_C;
50
51                 when ENUM_C =>
52                     -- Anweisungen, die ausgeführt werden wenn state = ENUM_C ist
53                     delay <= 100;
54                     state <= ENUM_A;
55
56             end case; -- aber der komplette case-Block bekommt natürlich wieder ein end
57
58             another_one <= '1'; -- einfache Zuweisungen gehen im Prozess natürlich auch
59
60         end if;
61     end process; -- und auch Prozesse werden mit "end" abgeschlossen
62 end Behavioral;
```