Laborübung 4

Zustandsautomaten (Finite State Machines)

Für den Entwurf und die Beschreibung von digitalen Systemen bilden Zustandsautomaten (Finite State Maschines; FSMs) eine wesentliche Grundlage. Mit Zustandsautomaten werden zyklische Funktionsabläufe realisiert, sie steuern andere Logikschaltungen und in komplexen digitalen Systemen werden sie zur Synchronisation mehrerer Komponenten eingesetzt. Zustandsautomaten sind sequenziell arbeitende Logikschaltungen, die gesteuert durch ein periodisches Taktsignal eine Abfolge von Zuständen zyklisch durchlaufen.

aus: Reichardt, Schwarz, VHDL-Synthese, 4. Auflage

Teil 1

In diesem Teil soll ein Zustandsautomat implementiert werden, der zwei spezifische Sequenzen von Eingangssymbolen erkennen kann. Einerseits vier aufeinander folgende Nullen, andererseits vier Einsen. Als Eingang dient das Signal *w*, als Ausgang das Signal *z*. Immer wenn für vier aufeinander folgende Clock-Impulse (hier: steigende Flanken) *w=0* oder aber *w=1* war, dann soll *z* auf *1* sein, ansonsten auf *0*. Dies soll auch für sich überlappende Sequenzen gelten. Wenn also fünf Clock-Impulse lang *w=1* gilt, dann soll *z* nach dem vierten und dem fünften Impuls auf *1* stehen. Der geforderte Zusammenhang zwischen *w* und *z* ist noch einmal in Abb. 1 zu sehen.

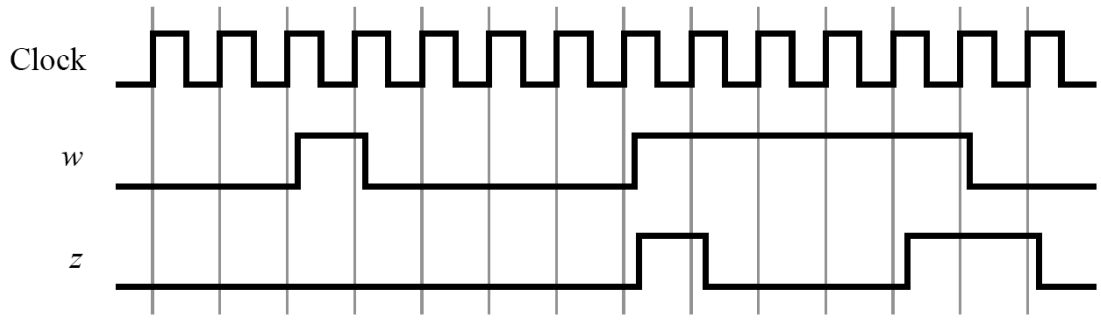

Abbildung 1: Timing für den Ausgang z

Der entsprechende Zustandsautomat (ein Moore-Automat) wird in Abb. 2 gezeigt.

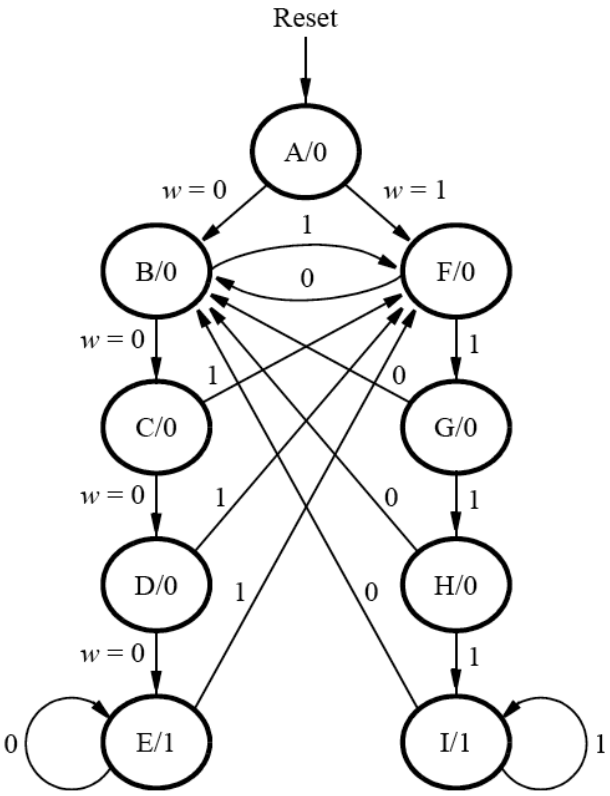

Abbildung 2: Zustandsautomat für die Sequenzerkennung

VHDL bietet eine Möglichkeit, einen Zustandsautomaten so zu spezifizieren, dass er vom Compiler und den Synthesewerkzeugen erkannt und entsprechend umgesetzt wird.
Innerhalb eines *PROCESS* wird dabei der aktuelle Zustand mittels *CASE* abgefragt und dann der jeweils nächste Zustand festgelegt. Dabei sind zwei verschiedene Signale (Vektoren) zu verwenden, von denen eines den aktuellen Zustand bereithält, während in das andere der gewünschte nächste Zustand geschrieben wird. In einem zweiten Prozess wird dann abhängig von einem Taktsignal der momentane Zustand aktualisiert. Listing 1 bietet ein entsprechendes Gerüst aus VHDL-Code.

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

ENTITY part1 IS
      PORT (
                  ... define inputs and outputs
            );
END part1;

ARCHITECTURE Behavior OF part1 IS

                  ... declare signals

TYPE State_type IS (A, B, C, D, E, F, G, H, I);
SIGNAL y_Q, Y_D : State_type;  -- y_Q is present state, y_D is next state

BEGIN

      ...

PROCESS (w, y_Q)  -- state table
BEGIN
      case y_Q IS
      WHEN A =>   IF (w = '0') THEN Y_D <= B;
                  ELSE  Y_D <= F;
                  END IF;

            ... other states

      END CASE;
END PROCESS;  -- state table

PROCESS (Clock)
BEGIN
            ...
END PROCESS;

            ... assignments for output z and the LEDs

END Behavior;
```

Listing 1: VHDL-Code für einen Zustandsautomaten

Die Codierung der Zustände in Binärwerte wird vom Synthesewerkzeug automatisch erledigt, der Code selbst enthält nur die Namen der Zustände.
In FPGAs wird häufig ein modifizierter „One-Hot-Code“ verwendet, bei dem je ein gesetztes Bit für einen Zustand steht. Abweichend davon ist der Initial-Zustand dabei als Null-Vektor codiert, um den in den Flipflops vorhandenen Clear-Eingang als Reset verwenden zu können. In Tabelle 1 wird ein solcher modifizierter One-Hot-Code gezeigt.

Name	Zustandscodierung
A	000000000
B	000000011
C	000000101
D	000001001
E	000010001
F	000100001
G	001000001
H	010000001
I	100000001

Tabelle 1: Modifizierte One-Hot-Codierung

Entwerfen und implementieren Sie nun einen Zustandsautomaten, der die oben erwähnten Sequenzen erkennt.

1. Schreiben Sie eine entsprechende VHDL-Datei. Nutzen Sie den Schalter *SW0* als synchronen active-low Reset für den Zustandsautomaten, *SW1* als Eingang *w* und den Taster *KEY0* (Achtung: active-low) als manuellen Clock-Eingang. Benutzen Sie die grüne LED *LEDG0* als Anzeige für den Ausgang *z* und die neun roten LEDs *LEDR8* bis *LEDR0* um den aktuellen Zustand auszugeben.

2. Untersuchen Sie die von Quartus II erzeugte Schaltung mit dem RTL-Viewer. Schauen Sie sich auch den erzeugten Zustandsautomaten an, und stellen Sie sicher, dass er dem Automaten in Abb. 2 entspricht. Beachten Sie ebenfalls die Codierung der Zustände.

3. Führen Sie eine funktionale Simulation der Schaltung durch.

4. Testen Sie die Schaltung auf dem DE2-Board. Stellen Sie sicher, dass der Automat die richtigen Zustandsübergänge benutzt (rote LEDs) und auf der grünen LED jeweils das korrekte Ergebnis angezeigt wird.

Teil 2

Anstatt der formalen Lösung oben soll nun die selbe Sequenzerkennung über Schieberegister durchgeführt werden. Schreiben Sie dafür einen VHDL-Code, der zwei 4-Bit-Schieberegister verwendet, eins für die vier Nullen und eins für die vier Einsen. Es steht Ihnen dabei frei, ob sie die Schieberegister selbst implementieren (in einem *PROCESS*), oder auf Alteras Megafunction-Library zurückgreifen; der Aufwand ist in beiden Fällen vergleichbar gering. Entwerfen Sie die entsprechende Schaltungslogik, um den Ausgang *z* anzusteuern. Die Schalter, Taster und LEDs sollen wie beim vorherigen Teil verwendet werden. Beobachten Sie das Verhalten der Schieberegister und des Ausgangs *z*.

Teil 3

Es soll nun eine Schaltung entworfen werden, die das Wort „HALLO“ über die acht 7-Segment-Anzeigen *HEX7* bis *HEX0* scrollen kann. Die Buchstaben sollen bei jeden Druck auf den Taster *KEY0* von rechts nach links durch die Anzeige laufen. Nachdem das Wort auf der linken Seite verschwindet, soll es auf der rechten wieder auftauchen (wie bekannt).

Verwenden Sie dazu acht 7-Bit-Register, die wie eine Schlange hintereinander geschaltet sind, so dass die Ausgänge des ersten Registers die Eingangssignale für das zweite Register liefern, usw. Eine solche Schaltung wird oft auch als „Pipeline“ bezeichnet.
Die Ausgänge jedes Registers sollen dabei direkt die Eingänge der jeweiligen 7-Segment-Anzeige treiben.

Entwerfen Sie einen Zustandsautomaten, der die Pipeline auf folgende zwei Weisen steuert:

1. Die ersten acht Clock-Impulse lang nach einem Reset fügt der Automat die jeweils richtigen Zeichen (H, A, L, L, O, , ,) über das erste Register in die Pipeline ein.
2. Nachdem der erste Schritt abgeschlossen ist, stellt der Zustandsautomat die Pipeline so um, dass eine Schleife entsteht, also das letzte Register den Eingang für das erste bildet, damit die Buchstaben endlos auf der Anzeige zirkulieren.

Schreiben Sie den VHDL-Code mit einem entsprechenden Zustandsautomaten. Implementieren sie das Register als eigene Komponente. Benutzen Sie den Taster *KEY0* als Takteingang und den Schalter *SW0* als synchronen active-low Reset.

Sie können sich die Arbeit erleichtern, indem Sie mittels des VHDL-Konstrukts *CONSTANT* die Codierung der Buchstaben als 7-Bit-Vektor Konstante definieren. Ein Decoder für die Segmentanzeigen ist nicht notwendig. Weiterhin können sowohl das „WITH/SELECT“ als auch das „WHEN/ELSE“ Konstrukt sinnvoll verwendet werden.

Teil 4

In diesem Teil soll nun die Schaltung aus dem vorherigen Teil so modifiziert werden, dass auf den Taster als manuelle Takteingabe verzichtet werden kann und statt dessen der auf dem Board vorhandene 50 MHz Takt (*CLOCK_50*) verwendet wird. Das Wort „HALLO“ soll dabei langsam über die Anzeige laufen. **Dennoch sollen der Zustandsautomat sowie die Pipeline-Register direkt die 50 MHz als Takteingang verwenden.** Überlegen Sie sich daher eine Lösung, wie Sie mittels eines zusätzlichen Signals die geforderte langsame Scrollgeschwindigkeit erreichen können.