Laborübung 2

Teil 1: Latches, Flipflops, Counter

A 1

Abbildung 1 zeigt den Schaltkreis eines gated D-Latches. In Listing 1 wird exemplarisch ein Stück VHDL-Code vorgestellt, der den abgebildeten Schaltkreis in Form von Logikausdrücken realisiert.

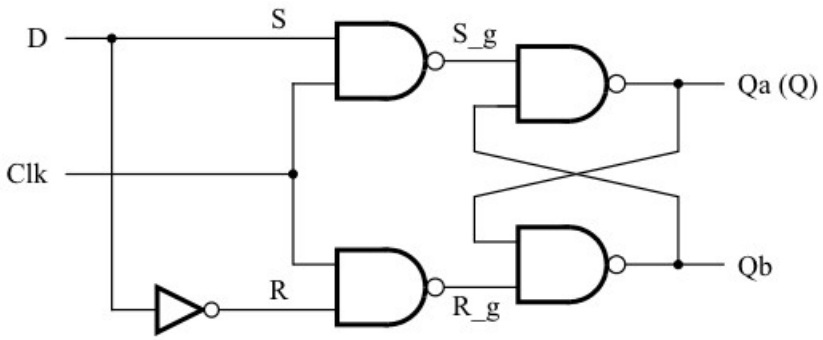

Abbildung 1: Schaltkreis eines Gated D-Latch

```
-- A gated D latch described the hard way
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part1 IS
     PORT (
            Clk, D : IN STD_LOGIC;
            Q      : OUT STD_LOGIC
           );
END part1;

ARCHITECTURE Structural OF part1 IS
     SIGNAL R, S, R_g, S_g, Qa, Qb : STD_LOGIC ;

BEGIN
     S <= D;
     R <= NOT D;
     R_g <= R AND Clk;
     S_g <= S AND Clk;
     Qa <= NOT (R_g OR Qb);
     Qb <= NOT (S_g OR Qa);
     Q <= Qa;
END Structural;
```

Listing 1: Gated D-Latch (kombinatorisch)

Die gleiche Funktionalität lässt sich mit Hilfe eines „PROCESS“ realisieren, der sensitiv auf die Signale „D“ und „Clk“ ist. Der entsprechende VHDL-Code ist in Listing 2 wiedergegeben.

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

ENTITY d_latch IS
     PORT (
            D, Clk : IN STD_LOGIC ;
            Q : OUT STD_LOGIC
           );
END d_latch ;
ARCHITECTURE Behavior OF d_latch IS
BEGIN
     PROCESS ( D, Clk )
     BEGIN
          IF Clk = '1' THEN
               Q <= D ;
          END IF ;
     END PROCESS ;
END Behavior ;
```

Listing 2: Gated D-Latch (Prozess)

Abbildung 2 zeigt nun einen Schaltkreis mit drei verschiedenen Speicherelementen. Neben einem gated D-Latch sind auch ein D-Flipflop mit positiver Taktflanke sowie eines mit negativer Taktflanke vorhanden. Aus Abb. 3 können die Signale an den Ausgängen abhängig von den Eingangssignalen „Clock“ und „D“ entnommen werden.

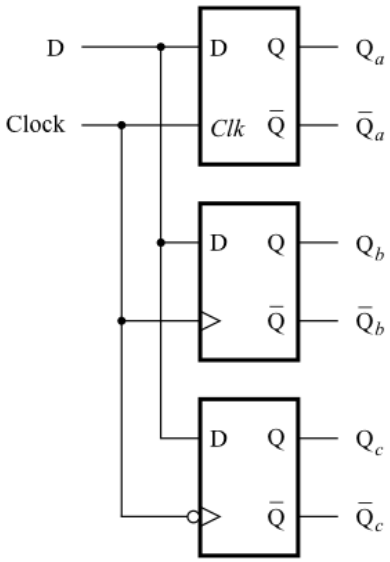

Abbildung 2: Schaltkreis

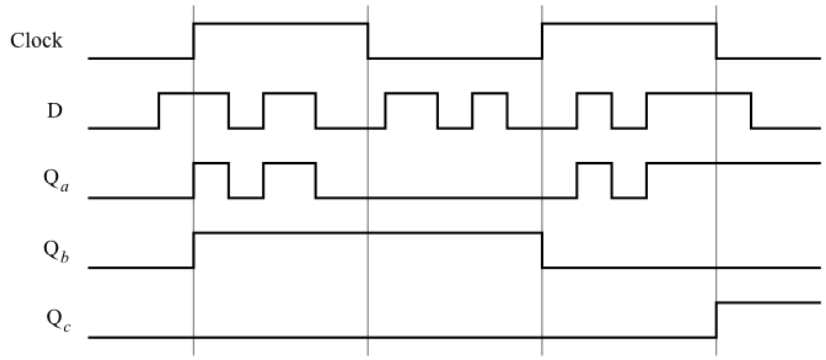

Abbildung 3: Timing-Diagramm

Implementieren Sie den abgebildeten Schaltkreis.

1. Machen Sie sich mit dem Konstrukt „`PROCESS`“ vertraut. Beachten Sie besonders das Schlüsselwort „`'event`“ im Zusammenhang mit (Takt-)Signalen.

2. Schreiben sie eine VHDL-Datei, welche die drei verschiedenen Speicherelemente instanziiert und implementieren Sie die Elemente als Komponenten. Verwenden Sie dazu wie in Listing 2 jeweils einen „`PROCESS`“.

3. Erstellen Sie eine Vector Waveform File (.vwf) um die Inputs und Outputs zu spezifizieren. Übernehmen sie die Inputs aus Abb. 3. Simulieren sie den Schaltkreis und überprüfen sie das unterschiedliche Verhalten der drei Speicherelemente.

A 2

Der Schaltkreis in Abb. 4 zeigt einen synchronen 4-Bit-Zähler, der aus vier T-Flipflops aufgebaut ist. Ist das Enable-Signal gesetzt, so erhöht er seinen Zählerstand bei jeder positiven Taktflanke. Mit dem Reset-Signal lässt sich der Zähler wieder auf Null zurücksetzen.

Ein T-Flipflop wechselt seinen Zustand („toggle“) bei jeder positiven Taktflanke, solange an „T“ ein High-Pegel anliegt, ansonsten wird der gespeicherte Zustand gehalten. Statt eines Dateneingangs besitzt es einen Clear-Eingang, mit dem der Speicher auf Null zurückgesetzt werden kann.

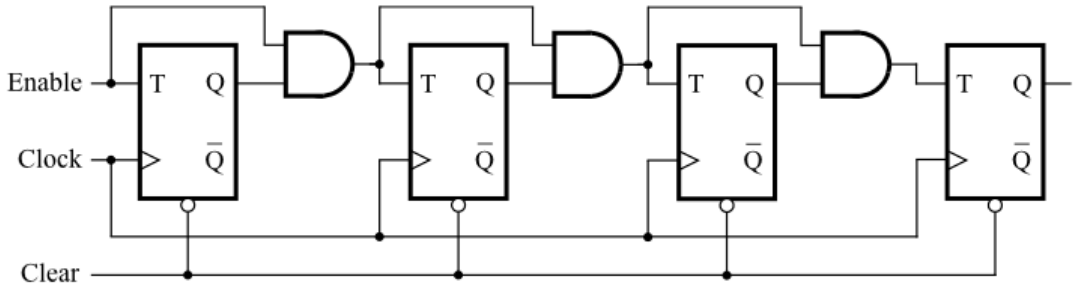

Abbildung 4: 4-Bit-Zähler

Implementieren Sie auf diese Weise einen 16-Bit-Zähler.

1. Erstellen sie ein T-Flipflop (als Komponente). Falls gewünscht, können Sie dafür auf ein bereits implementiertes Speicherelement zurückgreifen.

2. Schreiben sie eine VHDL-Datei, welche den 16-Bit-Zähler mit Hilfe der in Abb. 4 gezeigten Struktur umsetzt. Simulieren sie den Schaltkreis.

3. Erweitern Sie ihren Code so, dass der Taster KEY0 (Achtung: active-low) als Takteingang und die Schalter SW1 und SW0 als Enable und Reset dienen. Benutzen Sie die 7-Segment-Anzeigen HEX3-0, um hexadezimal den Zählerstand auszugeben.

A 3

Vereinfachen Sie ihren Code so, dass die Spezifikation des Zählers auf dem VHDL-Ausdruck

```
Q <= Q + 1;
```

basiert.
Um das Pluszeichen verwenden zu könnten, muss am Anfang der Datei zusätzlich über

```
USE ieee.std_logic_unsigned.all;
```

die vorzeichenlose Arithmetik aus der Bibliothek „ieee“ geladen werden.

Erstellen sie wieder einen 16-Bit-Zähler und überprüfen Sie seine Funktion.

A 4

Entwerfen und implementieren Sie einen Schaltkreis, welcher der Reihe nach die Ziffern 0 bis F auf der 7-Segment-Anzeige HEX0 ausgibt. Dabei soll jede Ziffer etwa eine Sekunde lang angezeigt werden. Benutzen Sie einen Zähler, um die Sekunden-Intervalle zu erzielen.
Der Zähler soll dabei vom auf dem Board vorhandenen 50 MHz Takt gespeist werden. Dieser ist an einen Pin mit der Bezeichnung „`CLOCK_50`“ angebunden, der wie einer der bekannten Schalter-Eingänge verwendet werden kann.

Teil 2: 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

A 5

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.

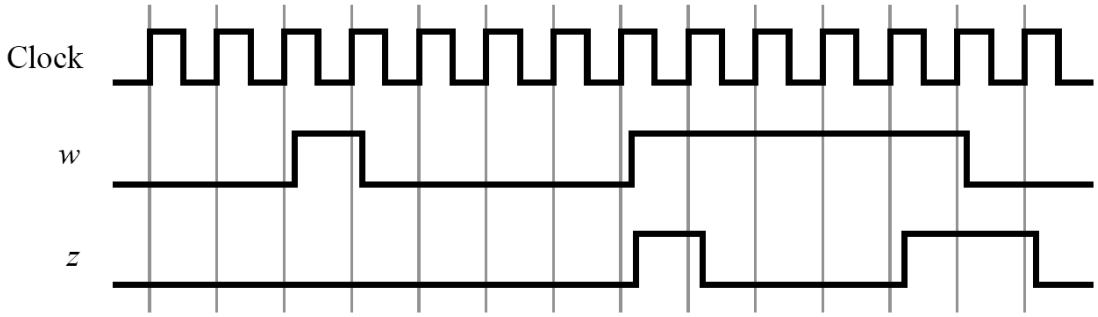

Abbildung 5: Timing für den Ausgang z

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

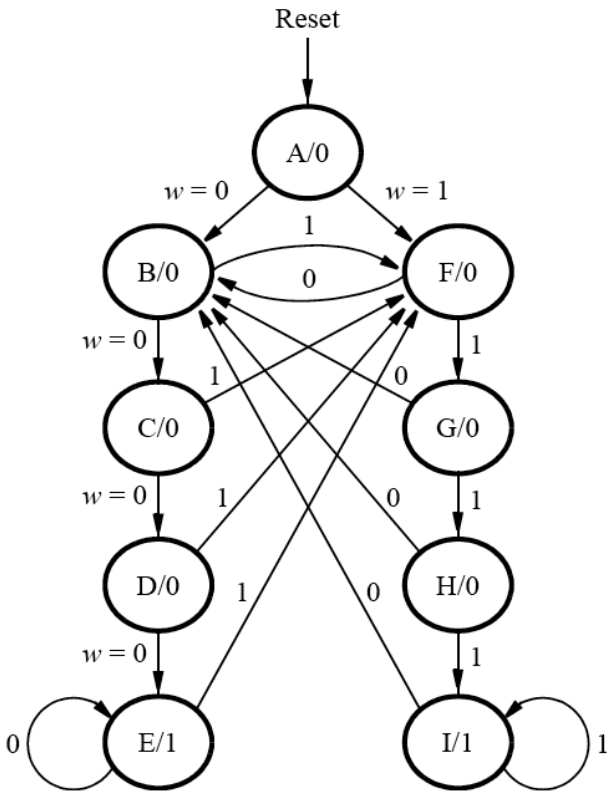

Abbildung 6: Zustandsautomat für die Sequenzerkennung

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 decoder IS
      PORT (
                  ... define inputs and outputs
             );
END decoder;

ARCHITECTURE Behavior OF decoder 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.

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.

A 6

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*.