# Übungen zu Architektur Eingebetteter Systeme

## Blatt 5

28./29.05.2009

# Teil 1: Grundlagen

## 1.1: VHDL

Bei der Erstellung Ihres Softcore-Prozessors mit Hilfe des SOPC Builder hatten Sie bereits erste Erfahrungen mit der Hardwarebeschreibungssprache VHDL gemacht. Weiterhin wurden Ihnen in der Vorlesung die grundlegenden Konzepte von VHDL vermittelt.

Es soll nochmals das Beispiel von Übungsblatt 1 aufgegriffen werden. Dort hatten Sie die Schalter mit den roten LEDS verbunden. Es soll nun die Struktur dieses Codes genauer betrachtet werden:

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY lichter IS
5      PORT (SW    : IN  STD_LOGIC_VECTOR(17 DOWNTO 0);
6            LEDR  : OUT STD_LOGIC_VECTOR(17 DOWNTO 0));
7  END lichter;
8
9  ARCHITECTURE Behavior OF lichter IS
10 BEGIN
11     LEDR <= SW;
12 END Behavior;
```

Die `ENTITY` definiert die externe Sichtweise einer Komponente. Es werden dort der Name, die Ein- und die Ausgänge dieser Komponente festgelegt. Wie diese Komponente realisiert ist wird dann unter der zugehörigen `ARCHITECTURE` beschreiben. Diese beschreibt das Verhalten. Es können verschiedenen Architekturen zu einer Entity deklariert werden. Im obigen Beispiel wird mit Hilfe der Architektur der Datenfluss beschrieben. Hierbei können auch kombinatorische logische Funktionen moduliert werden. Neu ist nun der `process` der innerhalb der Architektur das Verhalten beschreibt. Dessen Aufbau ist mit Programmiersprachen vergleichbar, die sequenziell arbeiten.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY lichter IS
5      PORT (SW    : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
6            LEDR  : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
7  END lichter;
8
9  ARCHITECTURE Prozess OF lichter IS
10     P1: PROCESS (SW(1), SW(0))
```

```
11      BEGIN
12          IF SW(1) = SW(0) THEN
13              LEDR(1) <= '1' after 1ns;
14          ELSE
15              LEDR(0) <= '1' after 1ns;
16          END IF;
17      END PROCESS P1;
18  END ARCHITEKTURE Prozess;
```

Weiterhin kann für die Architektur noch durch eine Strukturbeschreibung verwendet werden. Dieses sind Netzlisten aus Bibliothekselementen, welche über Elemente instandziirt und über Signale miteinander Verbunden werden. Dies würde für ein XOR das mit einem Inverter in Reihe geschaltet ist folgendermaßen aussehen:

```
1   ARCHITECTURE Structure OF lichter IS
2       SIGNAL I: bit;
3
4       -- Deklaration der Komponenten
5       COMPONENT xor IS
6           PORT (x,y: IN bit; z: OUT bit)
7       END COMPONENT xor;
8       COMPONENT inv IS
9           PORT (x: IN bit; z: OUT bit)
10      END COMPONENT inv;
11
12  -- Beschreibung der Netzliste
13  BEGIN
14      I0: xor PORT MAP (a,b,c);
15      I1: inv PORT MAP (c,d);
16  END ARCHITEKTURE Structure;
```

Mit diesen Beispielen haben Sie nun eine Grundstruktur und Übersicht für die Grundlegenden Komponenten eines VHDL Beschreibung kennen gelernt. Weitere Informationen finden Sie unter den auf der Homepage angegebenen Links sowie den meisten VHDL-Standardeinführungen.

# Teil 2: Aufgaben

### Aufgabe 1: Multiplexer 2-1

Wir wollen uns noch weiterhin etwas mit den Lichtern und Schaltern auf dem Board beschäftigen. Zum Anfang verwenden Sie einen der roten LEDs und drei der Schalter um einen einfachen Multiplexer aufzubauen. Ein Multiplexer hat die Eigenschaft, dass ein Eingang bestimmt, welcher der beiden weiteren Eingänge zum Ausgang durchgeschalten werden soll. Sie können dieses als eine VHDL Aussage in der Form `ausgang <= ...` ausdrücken.

Erweitern Sie Ihren VHDL-Code nun so, dass es Ihnen möglich ist einen 8-bit breiten, 2-zu-1 Multiplexer zu realisieren. Benutzen Sie $SW_{17}$ als Umschalter und die restlichen Schalter als Eingänge. Geben Sie die Eingaben auf den roten LEDs aus und das Ergebnis auf den grünen LEDs.

### Aufgabe 2: Multiplexer 5-1

Schreiben Sie einen VHDL Code, der einen 3-bit breiten, 5-zu-1 Multiplexer realisiert. Benutzen Sie $SW_{17-15}$ als Umschalter und die restlichen Schalter als Eingabeschalter. Geben Sie Ihre Eingabe wieder auf die roten LEDs aus und benutzen Sie für die Ausgabe die grünen LEDs ($LEDG_{2-0}$). Um den logischen Operationen besser folgen zu können, sollten Sie Zwischensignale verwenden.

### Aufgabe 3: 7 Segmentanzeige

Eine 7 Segmentanzeige besteht im wesentlichen aus 7 LEDs die einzeln angesteuert werden können. Um das Darstellen von Zahlen auf einer 7 Segmentanzeige zu vereinfachen wird oft ein 7-Segment Decoder vorgeschalten. Dieser soll in unserem Fall 4 Eingänge haben (4 Bit) und damit die Zahlen 0-9 darstellen können. Als Eingabe verwenden Sie 4 Schalter und als Ausgabe die 7 Segmentanzeige.

```
1   HEX0: OUT STD_LOGIC_VECTOR(0 TO 6)
```

Bedenken Sie bei der Erstellung, dass diese Entity später wieder verwendet werden soll, wenn der Counter erstellt wird. Weiterhin haben Sie die Möglichkeit, anstatt logischer Operationen einen Prozess zu verwenden. Dieser sieht folgendermaßen aus und erleichtert das schreiben des Codes:

```
1  PROCESS(demonstration)
2  CASE variable IS
3       WHEN "00" => ausgabe <= "0000";
4       WHEN "01" => ausgabe <= "1111";
5       WHEN OTHERS => ausgabe <= "1010";
6  END CASE;
```

### Aufgabe 4: Addierer

Beschreiben Sie zunächst einen Volladdierer mit den Eingängen `a,b,ci` und den Ausgängen `s,co`. Bevor sie nun mit Hilfe des Volladdierers einen Carry-Ripple-Addierer realisieren, sollten sie diesen auf Korrektheit testen, indem sie die Eingänge mit Schaltern verdrahten und die Ausgänge mit LEDs.

Mit der beschriebene Volladdierer Komponente können Sie nun recht einfach einen arry-Ripple-Addierer realisieren.Verwenden Sie hierzu $SW_{7-4}$ und $SW_{3-0}$ um die Eingänge A und B zu realisieren. $SW_8$ gibt Ihnen den In-Carry. Geben Sie wieder Ihre Eingaben auf den roten LEDs aus und verbinden Sie S, und Out-Carry mit den grünen LEDs. Die verwendung sollte ungefähr so aussehen:

```
1   bit0 : voll_addierer PORT MAP(A(0),B(0),C(0),S(0),C(1))
```

### Aufgabe 5: Latches und Flipflops

Hierbei soll auf das Altera Tutorial zurückgegriffen werden. Bearbeiten Sie in Tutorial drei *Latches, Flip-flops, and Registers* die Teile eins bis drei. Um eine neue vwf Datei

im ersten Teil zu erstellen müssen Sie unter `File` → `New..` gehen. Im zweiten und dritten Teil können Sie die Simulation gerne überspringen und das D-Latch direkt auf dem Board ausprobieren.

### Aufgabe 6: Counter

Erstellen Sie einen VHDL Code, der die Zahlen 0 bis 9 auf der 7-Segmentanzeige `HEX0` hochzählt. Jede Zahl soll dabei für etwas eine Sekunden dargestellt werden. Ihr Zähler erhält als Eingabe das 50-MHz Signal des DE2-Boards (`CLOCK_50`). Verwenden Sie bei der Ausgabe ihren Code von der 7-Segmentanzeige aus Aufgabe 3. Als Grundstruktur können Sie folgendes verwenden:

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_unsigned.all;
5
6  ENTITY zaehler IS
7      PORT (   CLOCK_50      : IN     STD_LOGIC;
8               HEX0          : OUT    STD_LOGIC_VECTOR(0 TO 6));
9  END zahler;
10
11 ARCHITECTURE Behavior OF zahler IS
12     COMPONENT ausgabe
13         PORT (   EIN       : IN     STD_LOGIC_VECTOR(3 DOWNTO 0);
14                  AUS       : OUT    STD_LOGIC_VECTOR(0 TO 6));
15     END COMPONENT;
16
17     SIGNAL EIN : STD_LOGIC_VECTOR(3 DOWNTO 0);
18     SIGNAL langsamer_zaheler : STD_LOGIC_VECTOR(24 DOWNTO 0);
19     SIGNAL zahlen_umschalter : STD_LOGIC_VECTOR(3 DOWNTO 0);
20
21     BEGIN
22         -- Hier beginnt er eigentliche 1Hz, 4-bit Zähler
23         -- Zuerst muss eine Sekunde ''gezählt'' werden
24
25         PROCESS (CLOCK_50)
26             BEGIN
27                 -- Wann soll welcher Zähler hochgezählt werden?
28         END PROCESS;
29
30         PROCESS (CLOCK_50)
31             BEGIN
32                 IF () THEN
33                     IF () THEN
34                         IF () THEN
35                             -- Anweisung an für zahlen_umschalter
36                         ELSE
37                             -- Anweisung an für zahlen_umschalter
38                         END IF;
39                     END IF;
40                 END IF;
41         END PROCESS;
42
```

```
43      EIN <= zahlen_umschalter;
44      -- Jetzt noch den Bitwert auf der 7-Seg. Anzeige ausgeben
45      7_Seg: ausgabe PORT MAP (EIN, HEX0);
46
47  END Behavior;
48
49  LIBRARY ieee;
50  USE ieee.std_logic_1164.all;
51
52  ENTITY ausgabe IS
53      --Ports definieren
54  END ausgabe;
55
56  ARCHITECTURE Behavior OF ausgabe IS
57      --Das verhalten der Ausgabe beschreiben
58  END Behavior;
```