# 2

# Einführung in VHDL

Wie bereits in der Einleitung erwähnt ist VHDL eine Hardwarebeschreibungssprache, die sich im Gegensatz zu Softwaresprachen dadurch auszeichnet, dass Abarbeitungen paralell ablaufen können. Dieses beutetet wiederum, dass eine sequentielle Beschreibung, wie sie in den meisten Software-Programmiersprachen stattfindet, nicht direkt in VHDL umgesetzt werden kann. Hierzu muss selbst eine Taktung eingeführt werden. Wie dieses im einzelnen aussieht, hängt von der Situation ab. Typische Konzepte sind die State-Machine oder die Benutzung der Clock in Zusammenhang mit einem Counter. Als Grundregel können Sie zuerst einmal davon ausgehen, dass alle innerhalb eines Prozesses beschriebenen Aktionen parallel ausgeführt werden und die Belegung der Signale sowie Ein- und Ausgänge erst am Ende des Prozesses stattfindet. Somit ist folgende Beschreibung in VHDL möglich:

```
1  ARCHITECTURE Tauschen OF Beispiel IS
2      SIGNAL a,b: STD_LOGIC;
3
4      BEGIN
5          PROCESS(a,b)
6              BEGIN
7                  a <= b;
8                  b <= a;
9          END PROCESS;
10 END Tauschen;
```

VHDL bietet die Möglichkeit sowohl eine strukturelle Beschreibung als auch eine Verhaltensbeschreibung eines Bausteins zu implementieren, um dessen Ein- und Ausgabeverhalten zu spezifizieren. Die strukturelle Beschreibung eines Bausteines gibt an, wie viele Instanzen der einzelnen Komponenten zur Realisierung dieses Bausteines benutzt werden, wie diese Komponenten untereinander verdrahtet sind und wie diese mit den Ein- und Ausgängen des Bausteines verdrahtet

sind. Die Verhaltensbeschreibung dagegen beschreibt einen Baustein nicht durch seine Komponenten sondern durch gleichzeitig nebeneinander laufende Prozesse, welche die Belegung der Signale des Bausteines funktional bestimmen. Prozesse beschreiben damit wie sich Signale in Abhängigkeit anderer Signale verhalten.

Bevor Sie im nächsten Kapitel beginnen sich mit dem DE2-Board und Quartus II zu beschäftigen, sollen im Folgenden einige Grundlegenden Konzepte von VHDL erklärt werden. Durch die weite Verbreitung von VHDL finden Sie auch ausreichend Literatur in der Bibliothek oder online.

Zu jeder (simulierbaren und synthetisierbaren) VHDL Beschreibung existiert eine sogenannte Top-Level-Entity. Das Hauptmerkmal dieser Komponente besteht darin, dass diese den Projektnamen trägt und dass alle nach externen geführten Signale aufgelistet sind. In unserem Fall sind dies die PINs des FPGA.

## Bibliotheken

Jeder Baustein beginnt mit dem Einbinden der Bibliotheken (Libraries). Folgende zwei Zeilen werden am Anfang von allen Ihren Bausteinen stehen:

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

Der Datentyp **std_logic** ist eine nach dem IEEE Standard 1164-1993 definiert 9-wertige Logik. Dabei können Signale die in Tabelle 2 angegebenen Werte annehmen:

Im Gegensatz zu dem Datentyp **bit**, der nur die Werte '0' und '1' annehmen kann, hat **std_logic** den Vorteil alle Zustände die auch auf der Zielplattform auftreten können zu beschreiben. Innerhalb der Übungen werden Sie jedoch fast ausschließlich mit starken logischen Nullen und Einsen ('0' bzw. '1') arbeiten. Jedoch sollten sie trotzdem alle Signale immer als **std_logic** Datentypen deklarieren, da innerhalb des Syntheseprozesses von Quartus II auch andere Zustände der Signale betrachtet werden können, beispielsweise wenn Ihr Signal noch nicht initialisiert wurde ('U') oder wenn der Zustand undefiniert ('X') ist.

Sobald sie elementare Rechenoperationen benötigen (wie beispielsweise Plus '+' oder Minus '-') müssen Sie zusätzlich aus der **ieee** Bibliothek folgendes laden:

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

| | |
|---|---|
| 'U' | *nicht initialisiert* |
| 'X' | *undefiniert* |
| '0' | *starke logische Null* |
| '1' | *starke logische Eins* |
| 'Z' | *Hochohmig* |
| 'W' | *schwach Unbekannt* |
| 'L' | *schwach logische Null* |
| 'H' | *schwach logische Eins* |
| '-' | *nicht beachten (don't care)* |

**Tabelle 2.1.** Signale der 9-wertigen standard logic nach IEEE Standard 1164-1993

## Schnittstellenspezifikation

Die Schnittstellen eines Bausteines werden über die Schnittstellenspezifikation (**entity** Konstrukt) beschrieben. Diese stellt die externe Sicht der Komponente dar. Den Namen des Bausteines können Sie selbst bestimmen (natürlich sollten keine reservierten Worte, Zahlen, Umlaute, Leerzeichen ... verwendet werden). Jedoch muss Ihr oberster Baustein den Namen des Projektes tragen. Beispielhaft sieht dieses folgendermaßen aus:

```
ENTITY baustein IS
  PORT (
       portname : INOUT   datentyp
       );
END baustein;
```

```
ENTITY most_used IS
    PORT (
         CLOCK_50     :   IN  STD_LOGIC;
         KEY          :   IN  STD_LOGIC_VECOTR(3 downto 0);
         SW           :   IN  STD_LOGIC_VECTOR(17 downto 0);
         LEDG         :   OUT STD_LOGIC_VECTOR(7 downto 0);
         LEDR         :   OUT STD_LOGIC_VECTOR(17 downto 0)
   );
END most_used;
```

Für die Bezeichnungen der externen Schnittstellen sollten Sie auf die im Handbuch angegebenen Bezeichnungen zurückgreifen. Dieses macht Ihnen die Arbeit leichter, da die jeweiligen Hardwarekomponenten auf dem Board auch mit den gleichen Bezeichnungen beschriftet sind. Die in diesen Übungen am meisten benutzten Ein- und Ausgaben sind im Beispiel *most_used* aufgelistet. Hierzu kommen noch die PINs welche Sie für den Softcoreprozessor benötigen werden. Diese sind dann im jeweiligen Tutorial angegeben.

Technisch gesehen sind die PINs des FPGA direkt über Leiterbahnen mit den jeweiligen elektronischen Bauteilen verbunden. Jedoch erscheint es sehr umständlich die PINs als Schnittstellen anzugeben. Daher ist es wesentlich einfacher, den PINs Namen und Vektoren zuzuordnen, so dass beispielsweise alle grünen LEDs einen gemeinsamen Vektor LEDG haben. Dieser ist nun 8-bit Breit und läuft damit von 7 bis 0.

Um Ihnen die Arbeit abzunehmen, gibt es von Altera schon die Datei, DE2pinassignments.csv welches die Zuordnung zwischen den FPGA PINs und den auf dem Board angegebenen Namen enthält. Im Tutorial des nächstes Kapitels werden Sie lernen, wie diese Datei eingebunden wird.

Zusätzlich soll noch darauf hingewiesen werden, dass alle nicht verwendeten PINs auf tri-state gesetzt werden sollen. Dieses verhindert, dass ungewollt Spannungspegel an Bauteile und PINs gelegt werden, die nicht verwendet werden.

Vor der Programmierung des FPGAs sollten alle offenen Pins noch auf Tri-state gesetzt werden. Unter `Assignments` → `Device` finden Sie einen Button `Device and Pin Options...`. Unter der Registerkarte `Unused Pins` können sie nun alle unbenutzten Pins auf `As input tri-stated` setzten.

## Beschreibung

Nachdem Sie Ihre Bibliotheken eingebunden haben und die Schnittstellen definiert haben, können Sie die Komponente durch sein Verhalten und/oder Struktur beschreiben. Der grundsätzliche Aufbau sieht folgendermaßen:

```
1 ARCHITECTURE behavior OF baustein IS
2     -- Hier koennen Signale definiert werden
3 BEGIN
4     --Code
5 END behavior;
```

Sie sollten sich selbständig über die Unterschiede zwischen einer Verhaltens- und Strukturbeschreibung informieren. Im folgenden finden Sie einige oft verwendete Routinen. Es soll noch auf folgendes

hingewiesen werden: Sie können Sich innerhalb der Komponente Hilfssignale definieren. Diese sollten auch für die Schnittstellensignale verwendet werden. Sollten sich die Signalnamen der Komponente ändern (beispielsweise weil sie andere LEDs benutzen oder reservierte Namen verwendet haben) müssen sie nur die Verdrahtung am Anfang Ihrer Komponente ändern und nicht alle Namen innerhalb der Komponente ersetzten. Dies könnte beispielhaft für die Clock so aussehen:

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4
5  ENTITY signaluse IS
6    PORT (
7        CLOCK_50     : IN     STD_LOGIC;
8    );
9  END singaluse;
10
11 ARCHITECTURE behavior OF signaluse IS
12
13   SIGNAL clk        : STD_LOGIC_VECTOR;
14
15   BEGIN
16        clk <= CLOCK_50;
17
18   PROCESS (clk)
19        -- ...
20   END PROCESS;
21
22 END behavior;
```

## Hilfreiche Routinen

- Bedingungen

```
1  ARCHITECTURE behavior OF Bedingung IS
2  SIGNAL a: STD_LOGIC_VECTOR (1 DOWNTO 0);
3  SIGNAL b: STD_LOGIC_VECTOR (2 DOWNTO 0);
4  BEGIN
5    PROCESS(a,b)
6    BEGIN
```

```
7         IF (a="01") THEN
8             b<="001";
9         ELSIF (a="11") THEN
10            b<="000";
11        ELSE
12            b<="111" ';
13        END IF;
14    END PROCESS;
15 END behavior;
```

- Fallunterscheidung

```
1  ARCHITECTURE behavior OF Fallunterscheidung IS
2  SIGNAL a,b: STD_LOGIC_VECTOR (2 DOWNTO 0);
3  BEGIN
4    PROCESS(a,b)
5    BEGIN
6        CASE a IS
7            WHEN "001" => b <= "000";
8            WHEN "011" | "110" => b <= "100";
9            WHEN OTHERS => b <= "111";
10       END CASE;
11   END PROCESS;
12 END behavior;
```

- Bedingung mit „WITH ... SELECT“

```
1  ARCHITECTURE structure OF Verzweigung IS
2  SIGNAL a             :STD_LOGIC_VECTOR(1 DOWNTO 0);
3  SIGNAL out,c,d,e       : STD_LOGIC;
4  BEGIN
5    WITH a SELECT b <=  c WHEN "10",
6                        d WHEN '01',
7                        e WHEN OTHERS;
8
9  END structure;
```

- Bedingung für Signale außerhalb eines Prozesses

```
1  ARCHITECTURE structure OF Signalbedingung IS
2  SIGNAL a,b,c,d: STD_LOGIC;
3  BEGIN
4    a<= NOT a WHEN b = '1' ELSE (c XOR d);
```

```
5 END structure;
```

- Clock-Signale

```
1  ARCHITECTURE behavior OF Clocksignal IS
2  SIGNAL clk: STD_LOGIC;
3  SIGNAL count: STD_LOGIC_VECTOR(4 DOWNTO 0);
4  BEGIN
5    clk <= clock_50;
6    PROCESS(clk)
7        IF (clk='1' AND clk'event) THEN -- wenn clk steigende Flanke
8            count <= count + 1;
9        ELSE
10           count <= count;
11       END IF;
12   END PROCESS;
13 END behavior;
```

- Variablen und Schleifen

```
1  ARCHITECTURE behavior OF Schleife IS
2  SIGNAL clk,reset: STD_LOGIC;
3  SIGNAL register: STD_LOGIC_VECTOR(6 DOWNTO 0);
4  BEGIN
5    clk <= clock_50;
6    reset <= NOT KEY(0);
7
8    PROCESS(clk)
9        VARIABLE i : INTEGER := 0;
10       IF reset ='1' THEN
11           FOR i IN 0 TO 5 LOOP
12               IF i = 0 OR i = 1 THEN
13                   register(i) <= NOT register(i+1);
14               ELSE
15                   register(i) <= register(i+1);
16               END IF;
17           END LOOP;
18       END IF;
19   END PROCESS;
20 END behavior;
```

# 3

# Das erste Projekt in Quartus II

Innerhalb dieser Übungsstunde wollen wir uns mit dem Entwurfswerkzeug Quartus II beschäftigen. Da der Hersteller Altera hierfür bereits Tutorials bereitstellt, soll auf diese zurückgegriffen werden. Die Tutorials finden Sie auf der Webseite von Altera `www.altera.com` unter `Support` → `Training` → `University Program` → `Materials` → `Tutorials`. Achten Sie darauf, dass Sie als Filter die „Quartus II Version“ 12.1 angeben und die Dokumente für die Hardwarebeschreibungssprache „VHDL“ verwenden.

**Achtung:** Bitte beachten Sie bei der Programmierung der Boards unbedingt, dass der kleine Schalter links des LCD-Displays auf **RUN** steht, dass Board mit Strom versorgt wird und **angeschaltet** ist. Bitte stecken Sie erst dann das USB-Kabel in die USB-BLASTER-Buchse. Nun können Sie das Board einschalten. Ist der Schalter nicht auf `run` gestellt und/oder das Board beim Programmieren ausgeschaltet wird das Board beschädigt!

### Aufgabe 1

Bearbeiten Sie das Tutorial *Quartus II Introduction*. Den Teil 8 über das „Simulating the Designed Circuit“ sowie den Teil 9.2 über das „active serial programming“ lassen Sie bitte aus. In Teil 7 des Tutorials können Sie direkt das „PIN-Assigment File“ verwenden. Damit Sie die PINs des FPGA nicht einzeln einem Signal zuweisen müssen, stellt Altera ein sogenanntes „PIN-Assigment File“ bereit. Dieses finden Sie auf der Altera-Seite unter `Support` → `Training` → `University Program` → `Boards` → `DE2-115` im Abschnitt `Materials` oder direkt `ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE2-115/DE2-115.qsf`. Es wird Ihnen empfohlen, die Datei an einer zentralen Stelle zu speichern, da Sie diese bei fast jedem Projekt benötigen. Vergessen Sie dabei nicht, wie in Kapitel 2 beschrieben, alle nicht benutzten PINs auf `tri-state input` zu setzen. Dies müssen Sie bei allen folgenden Projekten nun immer selbstständig machen.

**Aufgabe 2**

Ziel dieser Aufgabe ist es, Eingabe- und Ausgabe-Bausteine an einen FPGA anzubinden. Hierzu sollen Sie die 18 Eingabeschalter $SW_{17-0}$ mit den 18 roten LEDs $LEDR_{17-0}$ direkt verbunden werden. Dabei gibt Ihnen das Handbuch *DE2 User Manual* vor, dass beispielsweise der Schalter $SW_0$ mit dem PIN $N25$ und die $LEDR_0$ mit dem PIN $AE23$ verbunden ist. Es bietet sich an, die Namen aus dem *DE2 pin assignments.csv* File zu verwenden, um nun nicht alles PINS suchen zu müssen.

Nun können Sie eine Zuordnung in folgender Form vornehmen:

```
1 LEDR(17) <= SW(17);
2 LEDR(16) <= SW(16);
3  ...
4 LEDR(0) <= SW(0);
```

Dabei scheint es sinnvoll, beide Schnittellen als `Standard Logik Vektoren` zu implementieren. Damit kann die Zuordnung wie folgt erfolgen:

```
1 LEDR(17 downto 0) <= SW(17 downto 0);
```

Oder, da beide Vektoren die gleiche Größe haben:

```
1 LEDR <= SW;
```

Erstellen Sie nun ein Projekt und testen Sie dieses auf dem Board. Dabei sollen die die 18 Schalter so mit den 18 LEDs verbunden werden, dass jeweils ein Schalter eine LED ein- bzw. ausschaltet.

Erweitern Sie nun Ihren Code so, dass folgendes für alle Schalter funktioniert. Dabei sollten Sie die Vorteile der Vektoren nutzen, sodass am Ende nicht für jede LED eine eigene Zeile dasteht.

```
1 LEDR(0) <= SW(0) AND SW(9);
2 LEDR(1) <= SW(1) AND SW(10);
3   ...
```

# 4

# VHDL Grundlagen

In diesem Übungskapitel soll Ihnen der Umgang mit der Hardwarebeschreibung näher gebracht werden. Hierzu sollen Sie folgende Aufgaben in VHDL implementieren und auf dem DE2-Board testen.

### Aufgabe 1

Wir wollen uns noch etwas mit den Lichtern und Schaltern auf dem Board beschäftigen. Zum Anfang verwenden Sie eine 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 durchgeschaltet werden soll. Sie können dies als eine VHDL Aussage in der Form `ausgang <= ...` ausdrücken. Bauen Sie einen 2-1 Multiplexer. $SW_0$ und $SW_1$ dienen dabei als die 2 Dateneingänge und $SW_2$ als Umschalter. Die Ausgabe erfolgt über $LEDR_0$.

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 Schalter $SW_{0-15}$ als die 2 Blöcke mit jeweils 8 Eingängen. Geben Sie die Eingaben auf den roten LEDs $LEDR_{0-17}$ aus und das Ergebnis auf den grünen LEDs $LEDG_{7-0}$.

### Aufgabe 2

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

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. Hilfreich könnte die Verwendung des $CASE$ der $WITH...SELECT$ Statements sein.

### Aufgabe 4

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 Volladdiererkomponente können Sie nun recht einfach einen Carry-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))
```

# 6

# Counter in VHDL

Ein weiterer elementarer Teil von VHDL sind Counter. Diese werden sehr häufig benutzt und weisen oft eine ähnliche Struktur auf. Grundlegend dabei ist, dass das hochzählen dabei über eine clock funktioniert und meist alle Prozesse immer Taktsynchron arbeiten.

### Aufgabe 1

Erstellen Sie einen VHDL Code, der die Zahlen 0 bis 9 auf der 7-Segmentanzeige `HEX0` hochzählt. Jede Zahl soll dabei für etwa 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  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4
5  ENTITY counter IS
6   PORT (
7     CLOCK_50 : IN STD_LOGIC;
8     HEX0   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
9     );
10 END counter;
11
12 ARCHITECTURE behavior OF counter IS
13 COMPONENT htb -- Hex to binary decoder
14  PORT (
15    input : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
16    output : OUT  STD_LOGIC_VECTOR(6 DOWNTO 0)
```

```
  );
END COMPONENT;

SIGNAL clk   : STD_LOGIC_VECTOR;    -- Clock
SIGNAL count  : STD_LOGIC_VECTOR(24 DOWNTO 0); -- Counter
SIGNAL number : STD_LOGIC_VECTOR(3 DOWNTO 0);  -- Output number in hex
SIGNAL output : STD_LOGIC_VECTOR(6 DOWNTO 0);  -- Output for the hex display

BEGIN

 -- Connect the signals here
 -- ...
 Digit0: htb PORT MAP (number, output);


 PROCESS (clk)
 BEGIN
  IF () THEN -- Syncronize with the clock
   -- First of all an implementation for "counting" 1 second is needed
   -- Think about overflow
  END IF;
 END PROCESS;

 PROCESS (clk)
 BEGIN
  IF () THEN  -- Syncronize with the clock
   IF () THEN  -- Trigger the action to count
    IF () THEN  -- Is the number grater then 9?
     -- reset it
    ELSE
     -- increase it
    END IF;
   END IF;
  END IF;
 END PROCESS;
END behavior;
```

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

ENTITY htb IS
```

```
5  -- Define ports
6  END htb;
7
8  ARCHITECTURE behavior OF htb IS
9  BEGIN
10 -- Behavior of the hex to binary converter
11 -- You can copy it from your 7-segment display project
12 END behavior;
```

### Aufgabe 2

Erweitern Sie nun Ihren Code von Aufgabe 1 so, dass Ihr Zähler nicht nur bis 9 sondern bis 999 zählen kann. Passen sie die Taktung dabei so an, dass die Zählfrequenz ca. 100ms beträgt.