

VHDL

Bernhard Wintersperger

Elektronik und Digitaltechnik
HTBLuVA Bulme Graz-Gösting
5BHELI 2008/09

Inhaltsverzeichnis

1	Allgemeines und Geschichte	3
1.1	Geschichte	3
1.2	Analoge Systeme	3
2	Funktionsweise von VHDL	3
2.1	Abstraktionsebenen.....	4
2.1.1	Architektur	4
2.1.2	Funktion	4
2.1.3	Register-Transfer	4
2.1.4	Logik	5
2.1.5	Transistor und Physikalische Umsetzung	5
2.2	Fazit	5
3	Aufbau eines VHDL-Projektes	5
3.1	Package	6
3.2	Entity	6
3.3	Architecture	6
3.3.1	Strukturbeschreibung	6
3.3.2	Verhaltensbeschreibung	7
3.3.3	Prozesse	7
3.4	Configuration	7
3.5	Testbench	8
4	VHDL Beispiel	8
4.1	Aufgabenstellung.....	8
4.2	Package	9
4.3	Entity	9
4.4	Architecture	9
4.5	Testbench	10
4.6	Simulationsergebnis	12
5	VHDL-Hardware	13
5.1	CPLDs.....	13
5.2	FPGAs.....	13
6	Quellenverzeichnis	15

1 Allgemeines und Geschichte

VHDL ist eigentlich eine Abkürzung, steht für „*Very High Speed Integrated Circuit Hardware Description Language*“ und wird auch „*VHSIC Hardware Description Language*“ genannt. Es handelt sich dabei um eine weit verbreitete Hardwarebeschreibungssprache zur Beschreibung von digitalen Systemen. Eine solche Sprache ist zwar grundsätzlich mit einer Programmiersprache vergleichbar, bietet aber anders als diese, die Möglichkeit ein „Stück Hardware“ zu beschreiben, bevor es physisch hergestellt wird.

1.1 Geschichte

VHDL selbst wurde 1983 vom amerikanischen „Department of Defense“ initiiert und 1987 zum ersten Mal von der IEEE (Institute of Electrical and Electronics Engineers) standardisiert (IEEE 1076-1987). Die erste kommerzielle Version kam 1985 auf den Markt und wurde von IBM, Texas Instruments und Intermetrics entwickelt. Der aktuelle Standard stammt aus dem Jahr 1993 (IEEE 1976-1993) und enthält hauptsächlich Syntaxänderungen zum Originalstandard.

Der ursprüngliche Initiator von VHDL, das US-Verteidigungsministerium, das „Department of Defense“ – kurz DoD, ist bis heute auch einer der größten Anwender davon, in Nordamerika sogar der größte. Dieses Ministerium zeichnet auch für die heutige strikte Syntax der Sprache verantwortlich, da es die Einhaltung ebendieser zu einer Voraussetzung für die Erteilung von Aufträgen machte. 1988 legt das DoD sogar fest, dass alle digitalen Schaltungen in VHDL geschrieben sein müssen.

1.2 Analoge Systeme

Um nicht nur digitale Systeme beschreiben zu können, wurde als Erweiterung zu VHDL AHDL entwickelt. AHDL steht für „Altera Hardware Description Language“ und wurde von der Firma Altera entwickelt, um das Verhalten von elektronischen Bauelementen zu simulieren und vorauszuberechnen, ohne diese real aufbauen zu müssen.

Darüber hinaus gibt es auch VHDL-AMS. Mittels VHDL-AMS können analog-digitale Schaltungen simuliert werden. Es können beispielsweise Differentialgleichungen zur Beschreibung von Spulen oder Kondensatoren eingegeben werden. VHDL-AMS wurde 1999 unter dem Standard IEEE 1076.1-1999 standardisiert. VHDL-AMS soll außerdem in der Lage, sein den Rahmen der elektrischen Simulation zu verlassen und eine möglichst vollständige Systemsimulation bieten.

2 Funktionsweise von VHDL

Um mittels VHDL zu einer integrierten Schaltung zu kommen, arbeitet man nicht mehr mit den einzelnen Bauteilen der Schaltung, sondern beschreibt das gewünschte Verhalten der Schaltung. Genau hier liegt auch der große Vorteil von VHDL. Anstatt sich mühsam mit der Verbindung einzelner Bauteile befassen zu müssen, beschreibt man schlicht das gewünschte Verhalten und lässt sich die dazu passende Schaltung mittels Synthesetools synthetisieren. Ein weit verbreitetes Tool ist WebPack der Firma Xilinx.

Um nun zu einem digitalen System mithilfe von VHDL zu gelangen, gilt es ganz allgemeine Abstraktionsebenen zu durchlaufen.

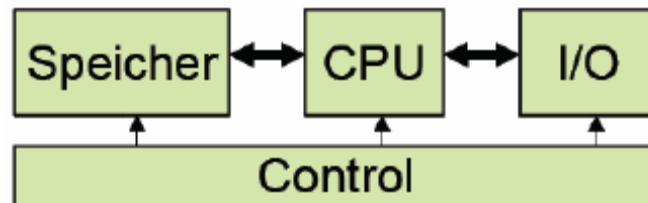
2.1 Abstraktionsebenen

Beim Entwurf von digitalen Systemen müssen generell verschiedene Abstraktionsebenen durchlaufen werden:

- Architektur
- Funktion
- Register-Transfer
- Logik
- Transistor
- Physikalische Umsetzung

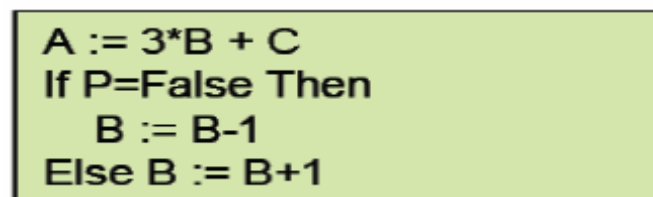
2.1.1 Architektur

Bei der Definition der Architektur geht es rein um die Hardware. Im einfachsten Fall geht es bei diesem Punkt darum, die Hardware für das zu entwickelnde System auszuwählen und im schwierigsten Fall geht es um die Entwicklung der Hardware.



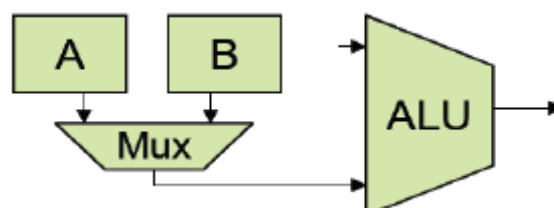
2.1.2 Funktion

Bei diesem Punkt geht es um die Definition der Funktion des Systems. Dies beinhaltet sämtliche Funktionen, Prozeduren, Prozesse, Zeitabläufe, etc. All diese Dinge werden durch bestimmte Algorithmen beispielsweise in Form von VHDL-Code festgelegt.



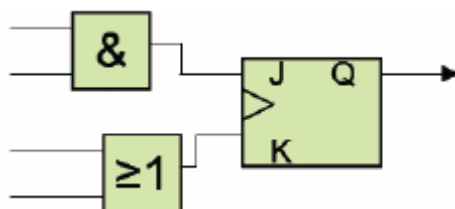
2.1.3 Register-Transfer

Beim Register-Transfer werden nun alle im vorigen Schritt getroffenen Entscheidungen bzw. beschriebenen Funktionen in Form von Operationen und Datenaustausch zwischen Registern beschrieben. Diese Funktion übernimmt bei VHDL bereits das Synthesetool.



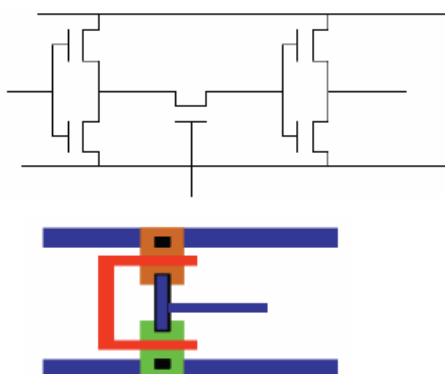
2.1.4 Logik

Bei dem Logik-Schritt wird das System durch Zusammenschaltung von Logik-Elementen und deren zeitlichen Verhalten beschrieben. Um typische VHDL-Hardware wie CPLDs oder FPGAs zu programmieren, ist dieser Schritt bereits ausreichend. Näheres dazu in Abschnitt 5.



2.1.5 Transistor und Physikalische Umsetzung

Dieser Schritt ist dann notwendig, wenn es darum geht, nicht nur beispielsweise einen FPGA Baustein zu beschreiben, bei dem die internen Schaltkreise klarerweise schon vorgegeben sind, sondern wenn man eine ASIC entwickeln möchte. ASIC steht für „application-specific integrated circuit“. Zu Deutsch: „Anwendungsspezifische integrierte Schaltung“. Wie der Name bereits sagt, handelt es sich dabei um eine spezifische integrierte Schaltung, die speziell für einen bestimmten Zweck gefertigt wird und dessen Funktion sich dadurch nicht mehr verändern lässt.



2.2 Fazit

Durch die verschiedenen Abstraktionsebenen wird noch einmal der große Vorteil von VHDL hervorgehoben. Durch VHDL ist es möglich, allein durch die Beschreibung der Funktion eines Systems am Schluss eine integrierte Schaltung mit genau dieser Funktionalität zu erhalten.

3 Aufbau eines VHDL-Projektes

Ein VHDL-Projekt besteht üblicherweise aus mehreren Teilen:

- Package
- Entity
- Architecture
- Configuration
- Testbench

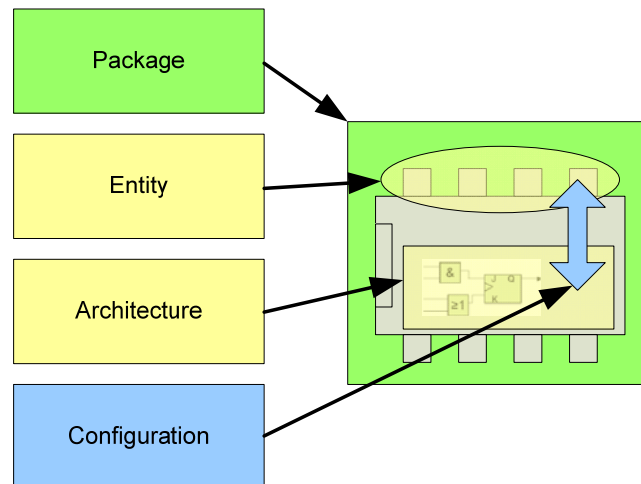


Abb. 3.1: Zusammenspiel von Package, Entity, Architektur und Configuration

3.1 Package

„Package“ ist die Bezeichnung für ganz allgemeine Dinge, die bei einem VHDL-Projekt festgelegt werden. Üblicherweise enthält das Package die verwendeten Bibliotheken. Diese sind am ehesten mit Header-Files aus C zu vergleichen. Mit diesen Bibliotheken wird beispielsweise festgelegt, dass der IEEE Standard verwendet wird, oder dass bestimmte Befehle für eine bestimmte Zielhardware vorhanden sind.

3.2 Entity

Die Entity ist im Wesentlichen eine Schnittstellenbeschreibung des Systems. Hier werden alle I/O-Ports, die am fertigen System benötigt werden, in Form von Signalen festgelegt.

Wichtig hierbei ist es, zwischen Signalen und Variablen zu unterscheiden. Variablen unterscheiden sich in VHDL nicht von Variablen in anderen Programmiersprachen. Signale hingegen gibt es nur in VHDL. Ein Signal stellt einen physischen Ein- bzw. Ausgang dar, kann aber nahezu gleich wie eine Variable angesprochen werden. Eine Entity wird auch gerne als „Blackbox“ bezeichnet. Man kennt ihre Anschlüsse, weiß aber von vorneherein nichts über das Verhalten.

3.3 Architecture

Die Architecture ist die eigentliche Funktionsbeschreibung. Hier ist der „Hauptcode“ zu finden, welcher für die Funktion des Systems zuständig ist.

In der Praxis gibt es zwei grundlegend unterschiedliche Architekturstile, die zum Einsatz kommen.

- Strukturbeschreibung (structural description)
- Verhaltensbeschreibung (behavioral description)

3.3.1 Strukturbeschreibung

Bei der Strukturbeschreibung geht es darum, verschiedene bereits definierte Komponenten miteinander zu verbinden. Eine Strukturbeschreibung hat daher eher die Form einer Netzliste.

Eine Komponente ist ein Baustein mit definierten Ein- und Ausgängen sowie einem definierten Verhalten. Eine Komponente kann somit aus jeder Entity mit Architektur erstellt werden.

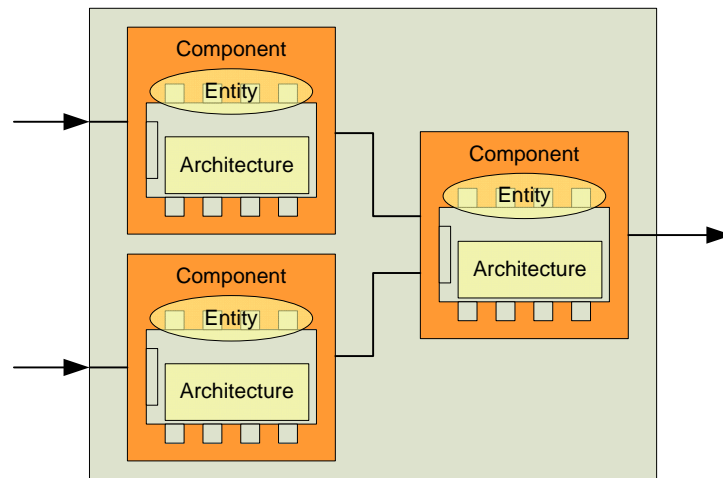


Abb. 3.3.1.1: Schematische Darstellung einer Strukturbeschreibung

3.3.2 Verhaltensbeschreibung

Bei der Verhaltensbeschreibung geht es nun nicht mehr darum einzelne, fertige Komponenten zu verbinden, sondern darum das Verhalten eines Bausteines durch Befehle festzulegen.

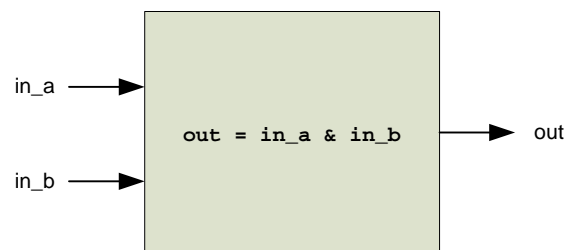


Abb. 3.3.2.1: Schematische Darstellung einer Verhaltensbeschreibung

3.3.3 Prozesse

Ein weiterer sehr wichtiger Punkt innerhalb der Architektur sind Prozesse. Sie dienen zur Beschreibung von Funktionsblöcken und werden parallel zu anderen Funktionsblöcken innerhalb der Architecture sequentiell abgearbeitet.

Bei der Erstellung eines Prozesses ist die Sensitivitätsliste ein sehr wichtiges Element. Durch sie wird festgelegt, bei welchen Ereignissen der Prozess ausgeführt wird.

3.4 Configuration

Mittels der Configuration wird eine Entity mit einer Architecture verknüpft. Dadurch ist es möglich, verschiedene Verhaltensweisen für eine Entity zu realisieren. Über die Configuration wird einfach die Architecture für eine Entity ausgetauscht und schon verhält sich der Baustein anders. Dies erhöht auch die Wiederverwendbarkeit von Code.

3.5 Testbench

Die Testbench dient zur Simulation des beschriebenen Systems. Da bei großen FPGAs die Synthese schon einmal mehrere Stunden dauern kann, ist es hilfreich, die Funktion des Bausteins zu simulieren.

In der Testbench können zeitlich genau festgelegte Eingangssignale (Input) an den Eingängen des Bausteins definiert werden. Dadurch und durch die Verhaltensbeschreibung lassen sich die Ausgangssignale (Output) simulieren.

Um den Baustein simulieren zu können, wird ein VHDL Modell erzeugt, ein „Device Under Test“ – kurz DUT.

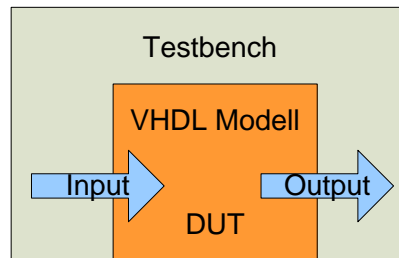


Abb. 3.5.1: Schematische Darstellung der Funktion einer Testbench

4 VHDL Beispiel

Am folgenden Beispiel werden nun die Begriffe: Package, Entity, Architecture, Verhaltensbeschreibung und Testbench anhand einer Aufgabenstellung erklärt.

4.1 Aufgabenstellung

Ein synchroner Zähler im Aiken-Code mit 3 Eingängen und 4 Ausgängen ist in VHDL zu realisieren. Es soll einen enable Eingang, einen enable Ausgang und einen Reset Eingang geben.

Eingänge	Ausgänge
reset_in	aiken_out
clock_in	enable_out
enable_in	

Zählertabelle	
Dez	Aiken
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111
Fehler	0101

4.2 Package

Als erstes wird das Package definiert. In diesem Fall wird die IEEE-Standard Bibliothek verwendet.

Die Texte mit den Bindestrichen zu Beginn sind Kommentare und haben keinen Einfluss auf die Funktion des Codes. Sie dienen lediglich zum besseren Verständnis des Codes.

```
-----
-----
-- Company: HTBLuVA Bulme Graz-Goesting
-- Engineer: Bernhard Wintersperger
--
-- Create Date:      10:04:07 10/05/2007
-- Design Name:
-- Module Name:      AikCount_source - Behavioral
-----
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

4.3 Entity

Hier werden nun die gesamten Ports für den Baustein definiert.

```
entity AikCount_source is
  Port ( reset_in : in  STD_LOGIC;
        clock_in  : in  STD_LOGIC;
        enable_in : in  STD_LOGIC;
        aiken_out  : out STD_LOGIC_VECTOR (3 downto 0);
        enable_out : out STD_LOGIC);
end AikCount_source;
```

4.4 Architecture

Der Code der Architecture zeigt nun gleich mehrere interessante Dinge. Zum einen ist gleich am Beginn ersichtlich, dass es sich bei dieser Architecture, um eine verhaltensbeschreibende Architecture handelt. Kennzeichnend hierfür ist das Signalwort `Behavioral`.

Bevor noch das eigentliche Verhalten definiert wird, wird eine Variable des Datentyps `signal` erstellt.

Unmittelbar danach beginnt die eigentliche Beschreibung der Funktion, welche mittels der Signalwörter `begin` und `end Behavioral;` definiert wird.

Als nächstes wird der Prozess `P1` erstellt. In der Sensitivitätsliste ist der Eingang `clock_in` zu finden. Das bedeutet, dass sobald ein Ereignis am Eingang `clock_in` auftritt, der Prozess ausgeführt wird. Im Prozess selbst wird die eigentliche Zählfunktion des Zählers beschrieben.

Am Schluss folgt noch eine Zuordnungstabelle, bei der jedem Zustand der Variable `code` ein entsprechender Zustand am Codeausgang `aiken_code` zugewiesen wird.

```

architecture Behavioral of AikCount_source is

    signal code : integer range 0 to 9; --variable um Dez
hochzuzählen

begin
    P1: process( clock_in )
    begin
        if( clock_in'event and clock_in = '1' ) then --wenn an
clock_in flanke auf 1 geht dann...
            if(reset_in = '1') then --wenn reset_in 1 ist
dann zähler rücksetzen
                code <= 0;
            else
                if(code = 9) then --wenn code auf
maximum ist dann rücksetzen
                    code <= 0;
                else
                    if (enable_in = '1') then --wenn reset_in
= 0 und code < 9 und enable_in = 1 dann code+1
                        code <= code + 1;
                    end if;
                end if;
            end if;
        end if;
    end process;

    enable_out <= '1' when code = 9 else '0'; --wenn code auf maximum
dann enable out = 1

    aiken_out <=
--zuweisungstabelle für den ausgang
"0000" WHEN code = 0 ELSE
"0001" WHEN code = 1 ELSE
"0010" WHEN code = 2 ELSE
"0011" WHEN code = 3 ELSE
"0100" WHEN code = 4 ELSE
"1011" WHEN code = 5 ELSE
"1100" WHEN code = 6 ELSE
"1101" WHEN code = 7 ELSE
"1110" WHEN code = 8 ELSE
"1111" WHEN code = 9 ELSE
"0101";

end Behavioral;

```

4.5 Testbench

Die Testbench ist grundsätzlich gleich aufgebaut wie die Beschreibung selbst. Es gibt jedoch einige Unterschiede. Die Entity der Testbench, darf beispielsweise niemals Ein- und/oder Ausgänge enthalten, da diese ausschließlich zur Beschreibung physischer Ports da sind.

In der Testbench wird anschließend die Komponente der Beschreibung des Zählers eingebunden und Variablen mit den Namen der Ports der Komponente erstellt. Danach werden die Variablen innerhalb der Architecture den jeweiligen Ports zugeordnet.

Als nächstes enthält die Testbench einen Prozess, welcher zur Generierung eines stabilen Taktsignals am Takteingang dient.

Parallel dazu werden am `enable_in` und am `reset_in` Eingang verschiedene Werte zu fix zugeordneten Zeiten angelegt und der Auswirkung auf die Ausgänge simuliert.

```

-----
-----
-- Company: HTBLuVA Bulme Graz-Goesting
-- Engineer: Bernhard Wintersperger
--
-- Create Date:    11:01:09 10/05/2007
-- Design Name:    AikCount_source
-- Module Name:    AikCount_TB.vhd
-- Project Name:   AikCount
--
-- VHDL Test Bench Created by ISE for module: AikCount_source
--
-- Notes:
-- This testbench has been automatically generated using types
std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design
in order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
-----
-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY AikCount_TB_vhd IS
END AikCount_TB_vhd;

ARCHITECTURE behavior OF AikCount_TB_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT AikCount_source
    PORT(
        reset_in  : IN std_logic;
        clock_in  : IN std_logic;
        enable_in : IN std_logic;
        aiken_out  : OUT std_logic_vector(3 downto 0);
        enable_out : OUT std_logic
    );
    END COMPONENT;

    --Inputs
    SIGNAL reset_in  : std_logic := '0';
    SIGNAL clock_in  : std_logic := '0';
    SIGNAL enable_in : std_logic := '0';

    --Outputs

```

```

SIGNAL aiken_out : std_logic_vector(3 downto 0);
SIGNAL enable_out : std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: AikCount_source PORT MAP( --zuordnung der Variablen zu
den Ports
        reset_in => reset_in,
        clock_in => clock_in,
        enable_in => enable_in,
        aiken_out => aiken_out,
        enable_out => enable_out
    );

    clk : PROCESS (clock_in) --prozess um clock zu generieren
(läuft immer parallel mit)
    BEGIN
        IF clock_in'EVENT AND clock_in = '1' then
            clock_in <= '0' AFTER 20 NS;
        ELSE
            clock_in <= '1' AFTER 20 NS;
        END IF;
    END PROCESS;

    tb : PROCESS
    BEGIN

        -- Wait 100 ns for global reset to finish
        wait for 100 ns;

        enable_in <= '1' after 5 ns, --setze enable auf 1
            '0' after 630 ns, --setze enable auf 0
            '1' after 700 ns; --setze enable auf 0
        reset_in <= '1' after 230 ns, --setze reset auf 1
            '0' after 250ns;--setze reset auf 0

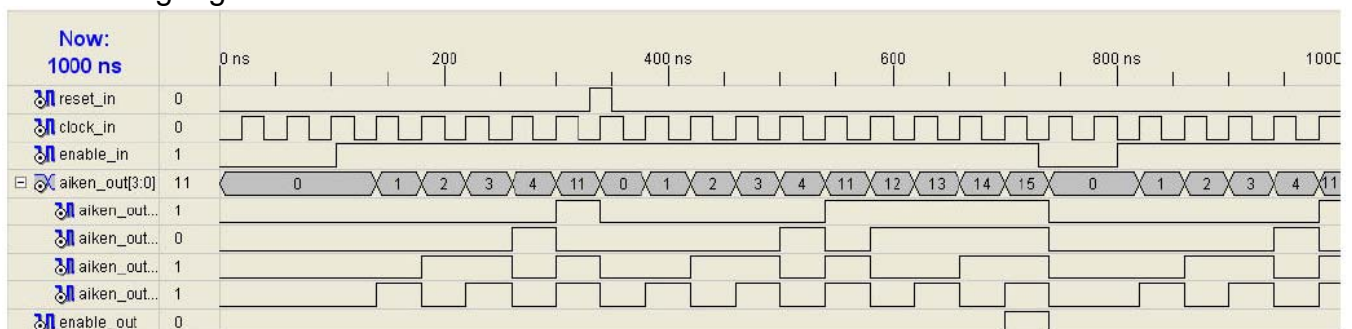
        wait; -- will wait forever
    END PROCESS;

END;

```

4.6 Simulationsergebnis

In der Simulation ist nun das Verhalten des Systems bei entsprechenden Signalen an den Eingängen ersichtlich.



5 VHDL-Hardware

Grundsätzlich kann mithilfe von VHDL das Verhalten von jedem beliebigen digitalen System beschrieben werden. In der Praxis wird es allerdings hauptsächlich für CPLDs und FPGAs eingesetzt.

Eine weitere oft verwendete Einsatzform ist, wie bereits in Abschnitt 2.1.5 beschrieben, zur Funktionsbeschreibung von ASICs.

5.1 CPLDs

CPLD steht für *Complex Programmable Logic Device* und ist die Bezeichnung für programmierbare Logikbausteine.

In einem CPLD sind im Wesentlichen viele logische Elemente (AND, OR) in Form einer Matrix zusammengeschaltet. Die Verbindungspunkte dieser Matrix lassen sich programmieren. Dadurch ist es möglich, bestimmte logische Kombinationen zu erhalten.

Zusätzlich zu der Logik-Matrix enthalten CPLDs an den Ein- bzw. Ausgängen schnelle Speicher wie Latches oder D-Flipflops. Die Zahl dieser Speicher ist allerdings sehr beschränkt.



Abb. 5.1.1: Ein CPLD der Firma Xilinx

CPLDs eignen sich durch ihre hohe Anzahl von Ein- und Ausgängen vor allem für die Lösung von komplexen, parallelen AND/OR-Logiken. Da die Anzahl der Register bzw. Speicher allerdings sehr begrenzt ist, sind FPGAs für Aufgaben wie Schieberegister oder digitale Zähler besser geeignet.

CPLDs werden unter anderem von folgenden Firmen produziert:

- Xilinx
- Altera
- Lattice
- Atmel

5.2 FPGAs

FPGA steht für *Field Programmable Gate Array* und ist ein programmierbarer integrierter Schaltkreis (IC – eng. integrated circuit). Durch spezifische Konfiguration der internen Strukturen können in FPGA verschiedenste Schaltungen abgebildet werden. Das reicht von einem einfachen Synchronzähler bis zu einem Mikroprozessorkern.

FPGAs werden vor allem dort eingesetzt, wo es auf schnelle Signalverarbeitung und nachträgliche Änderungsmöglichkeiten



Abb 5.2.1: Ein FPGA der Firma Altera

ankommt. Ein Beispiel sind Mobilfunk-Basisstationen (Sendemasten). Ein weiteres Argument für den Einsatz von FPGAs ist der niedrigere Preis im Vergleich zu ASICs, auch wenn diese dafür etwas schneller sind.

Intern besitzen FPGAs viele einzelne programmierbare Logikblöcke. Logische Operationen von den Typen AND, OR, NOR, NOT und NAND sind möglich. Neben dem Logikelement gehört zu einem Logikblock eines FPGAs auch immer ein Speicherelement, das beispielsweise als Latch oder Flipflop verwendet werden kann.

Die logischen Elemente selbst sind meistens über LUTs (Lookup-Tabellen) realisiert. Dies sind kleine Speicher, die für jeden möglichen Zustand der Eingänge den entsprechenden Ausgangszustand speichern.

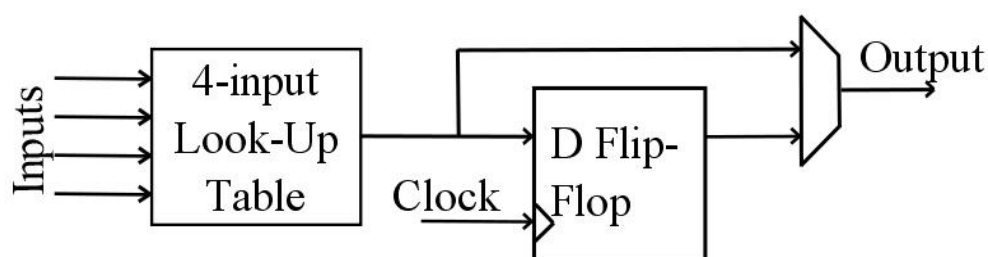


Abb. 5.2.2: Schematische Darstellung eines FPGA Logiblocks

Ähnlich wie bei den CPLDs sind die Logikblöcke der FPGAs mit einer Matrix verbunden, deren Verbindungspunkte konfigurierbare Schalter darstellen. Dadurch lassen sich verschiedenste Verbindungen zwischen den Logikblöcken herstellen.

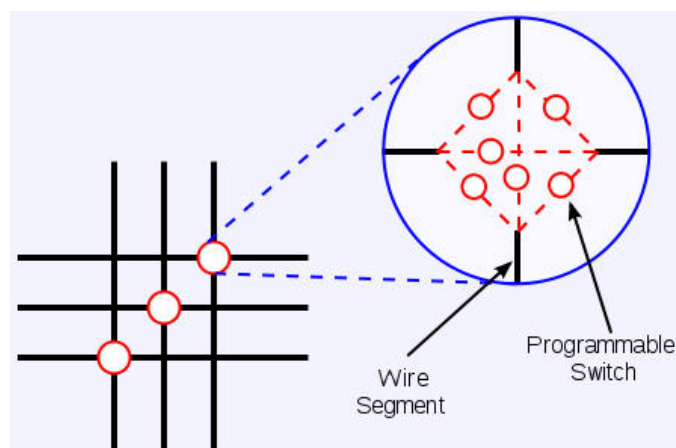


Abb. 5.2.3: Schematische Darstellung der Matrixverbindungen eines FPGAs

Der Unterschied zwischen CPLDs und FPGAs besteht hauptsächlich darin, dass FPGAs weitaus komplexer als CPLDs sind, nicht nur Signale von den Eingängen über AND/OR Verknüpfungen zu den Ausgängen verbinden können und weitaus mehr Flipflops aufweisen.

FPGAs werden, ebenso wie CPLDs, unter anderem von folgenden Firmen hergestellt:

- Xilinx
- Altera
- Lattice
- Atmel

6 Quellenverzeichnis

http://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language [26.04.2009]

<http://en.wikipedia.org/wiki/Vhdl> [26.04.2009]

<http://de.wikipedia.org/wiki/Hardwarebeschreibungssprache> [26.04.2009]

<http://de.wikipedia.org/wiki/IEEE> [26.04.2009]

<http://de.wikipedia.org/wiki/AHDL> [26.04.2009]

http://de.wikipedia.org/wiki/Anwendungsspezifische_Integrierte_Schaltung
[27.04.2009]

<http://de.wikipedia.org/wiki/CPLD> [27.04.2009]

<http://de.wikipedia.org/wiki/Fpga> [27.04.2009]

<http://en.wikipedia.org/wiki/CPLD> [27.04.2009]

Dipl. Ing. Franz Wolf – VHDL Einführung

Dipl. Ing. Franz Wolf – Einführung in VHDL

Windisch – Einführung in VHDL

Andreas Mäder - VHDL Kompakt, Universität Hamburg