Teil A Einführung

1 Entwurf elektronischer Systeme

1.1 Motivation

Die heutige Situation beim Entwurf elektronischer Systeme ist durch folgende Aspekte gekennzeichnet:

- Komplexität und Integrationsdichte nehmen ständig zu. Hauptgesichtspunkte dabei sind:
  - höhere Packungsdichten aufgrund geringerer Strukturgrößen beschleunigen das Vordringen von Produkten mit komplexem, elektronischem "Innenleben",
  - die Anforderungen an die Leistungsfähigkeit (Performance) der Elektronik (geringer Platzbedarf, hohe Taktrate, geringer Leistungsverbrauch, hohe Zuverlässigkeit) steigen.
- Wachsender Konkurrenzdruck und Anforderungen der Kunden bedingen immer kürzere Entwicklungszeiten. Eine Verzögerung der Markteinführung eines Produktes kann den Umsatz drastisch verringern und damit den Erfolg eines Unternehmens gefährden ("time to market"-Problematik).
- Die Komplexität, eine Wiederverwendung von Entwurfsdaten und die Wartung des fertigen Produktes erfordern eine vollständige, widerspruchsfreie und verständliche Dokumentation.

1.2 Entwurfssichten

Die Entwicklung elektronischer Systeme ist bei der heutigen Komplexität und den genannten Anforderungen nur durch eine strukturierte Vorgehensweise beherrschbar. Idealerweise wird man, ausgehend von einer Spezifikation auf Systemebene, die Schaltungsfunktion par-

titionieren ("Funktionale Dekomposition") und die grundsätzlichen Funktionen den einzelnen Modulen zuordnen. Schrittweise wird der Entwurf weiter strukturiert und zunehmend mit Details der Implementierung versehen, bis die für die Fertigung des elektronischen Systems notwendigen Daten vorliegen. Dies können Programmierdaten für Logikbausteine, Layouts für Leiterplatten oder Maskenbänder für die IC-Fertigung sein.

Man unterscheidet beim Entwurf elektronischer Systeme üblicherweise die drei Sichtweisen Verhalten, Struktur und Geometrie. Diese Einteilung wird durch das sog. Y-Diagramm verdeutlicht (Abb. A-1):

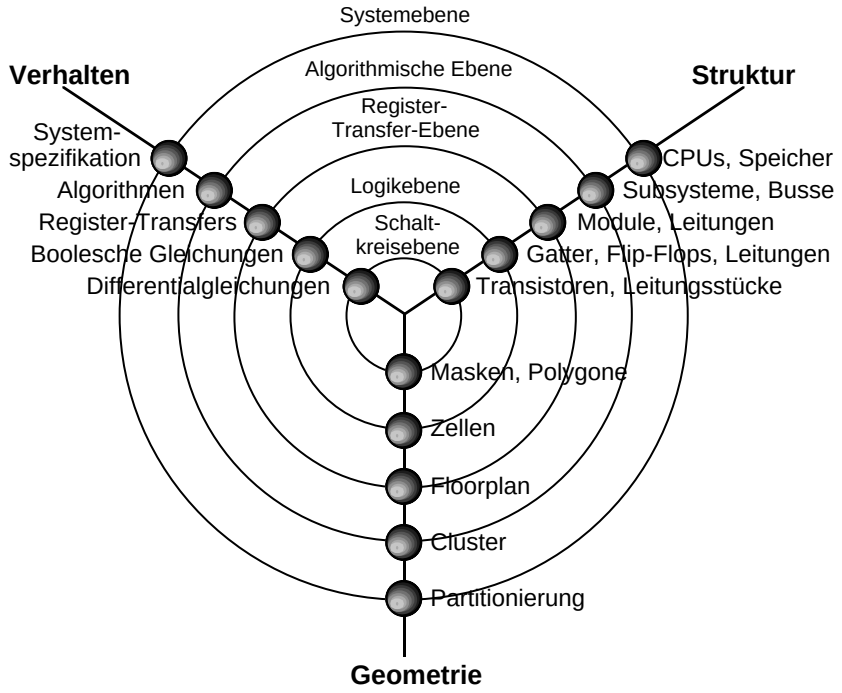

Abb. A-1: Y-Diagramm nach Gajski-Walker [WAL 85]

Gleichzeitig zu den drei Sichtweisen, die durch die Äste im Y-Diagramm repräsentiert werden, sind auch die verschiedenen Abstraktionsebenen durch Kreise mit unterschiedlichen Radien dargestellt. Ein großer Radius bedeutet hohe Abstraktion. Man kann nun vereinfacht den Entwurf elektronischer Systeme als eine Reihe von Transformatio-

nen (Wechsel der Sichtweise auf einem Abstraktionskreis) und Verfeinerungen (Wechsel der Abstraktionsebene innerhalb einer Sichtweise) im Y-Diagramm darstellen. Beginnend auf dem Verhaltensast in der Systemebene wird der Entwurf durch Verfeinerungs- und Syntheseschritte bis hin zum Layout auf dem geometrischen Ast durchgeführt. Ein "Place- and Route"-Werkzeug für Standardzellenentwürfe überführt beispielsweise eine strukturale Beschreibung in der Logikebene (Gatternetzliste) in eine geometrische Beschreibung in der Schaltkreisebene (IC-Layout).

Das reine Top-Down-Vorgehen (Entwicklung in Richtung Kreismittelpunkt) kann dabei nicht immer konsequent beibehalten werden. Verifikationsschritte zwischen den einzelnen Ebenen zeigen Fehler beim Entwurf auf. Gegebenenfalls muß man das jeweilige Entwurfsergebnis modifizieren, den Entwurfsschritt wiederholen oder sogar auf höherer Abstraktionsebene neu einsetzen. Man spricht deshalb auch vom "Jojo-Design".

1.3 Entwurfsebenen

1.3.1 Systemebene

Die Systemebene beschreibt die grundlegenden Charakteristika eines elektronischen Systems. In der Beschreibung werden typische Blöcke, wie Speicher, Prozessoren und Interface-Einheiten verwendet. Diese Module werden durch ihre Funktionalität (im Falle eines Prozessors z.B. durch dessen Befehlssatz), durch Protokolle oder durch stochastische Prozesse charakterisiert. Auf dieser Ebene dominieren meist noch die natürliche Sprache und Skizzen als Beschreibungsmittel.

Es werden auf Systemebene noch keinerlei Aussagen über Signale, Zeitverhalten und detailliertes funktionales Verhalten der einzelnen Blöcke getroffen. Vielmehr dient sie der Partitionierung der gesamten Schaltungsfunktion. In der geometrischen Sicht erfolgt auf dieser Ebene beispielsweise eine erste Unterteilung der Chipfläche.

1.3.2 Algorithmische Ebene

Auf dieser Ebene wird ein System oder eine Schaltung durch nebenläufige (d.h. parallel ablaufende) Algorithmen beschrieben. Typische Beschreibungselemente dieser Ebene sind Funktionen, Prozeduren, Prozesse und Kontrollstrukturen. Auf der Algorithmischen Ebene wird ein elektronisches System in der strukturalen Sicht durch allgemeine Blöcke beschrieben, die über Signale miteinander kommunizieren. In der Verhaltenssicht dagegen wird die Beschreibung des Verhaltens durch eine algorithmische Darstellung mit Variablen und Operatoren vorgenommen.

Es wird kein Bezug zur späteren Struktur der Realisierung (Hardwarepartitionierung) gegeben. Desgleichen werden auch keine zeitlichen Details durch Takt- oder Rücksetzsignale eingeführt.

1.3.3 Register-Transfer-Ebene

Bei Beschreibungen auf der Register-Transfer-Ebene ("Register Transfer Level", RTL) werden die Eigenschaften einer Schaltung durch Operationen (z.B. Addition) und durch den Transfer der verarbeiteten Daten zwischen Registern spezifiziert. Typischerweise werden in die Beschreibung Takt- und Rücksetzsignale integriert. Die einzelnen Operationen sind dann den Werten oder Flanken dieser Signale zugeordnet, so daß die zeitlichen Eigenschaften schon relativ genau definiert werden können.

In der strukturalen Sicht werden Elemente wie Register, Codierer, Multiplexer oder Addierer durch Signale miteinander verknüpft. In der Verhaltenssicht findet man vorwiegend Beschreibungen in Form von endlichen Automaten vor. Die Grobeinteilung der Chipfläche wird in der geometrischen Sicht zu einem sog. Floorplan verfeinert.

Eine mehr oder weniger automatisierte Umsetzung der Verhaltensbeschreibung in eine strukturale Beschreibung auf der Logikebene wird inzwischen durch kommerzielle Werkzeuge angeboten (d.h. Beschreibungen auf RT-Ebene sind meist synthetisierbar).

1.3.4 Logikebene

Auf der Logikebene werden die Eigenschaften eines elektronischen Systems durch logische Verknüpfungen und deren zeitliche Eigenschaften (i.a. Verzögerungszeiten) beschrieben. Der Verlauf der Ausgangssignale ergibt sich dabei durch die Anwendung dieser Verknüpfungen auf die Eingangssignale. Die Signalverläufe sind wertdiskret, d.h. die Signale nehmen nur bestimmte, vordefinierte Logikwerte (z.B. ´low´, ´high´, ´undefined´) an.

In der strukturalen Sichtweise wird der Elektronikentwurf durch eine Zusammenschaltung der Grundelemente (AND-, OR-, XOR-Gatter, Flip-Flops, etc.) dargestellt. Diese Grundelemente werden dabei von einer Bibliothek zur Verfügung gestellt. Innerhalb dieser Bibliothek sind die Eigenschaften der Grundelemente definiert. Diese bilden die Charakteristika der einzelnen Zellen der Zieltechnologie (z.B. 2 µm Standardzellen-Prozeß der Fa. XYZ, FPGA 1.0 der Fa. ABC) in vereinfachter Form nach. Zur Erstellung der strukturalen Beschreibung (Gatternetzliste) werden meistens graphische Eingabewerkzeuge ("Schematic Entry") verwendet.

Die Sichtweise "Verhalten" bedient sich dagegen vor allem einer Darstellung durch Boolesche Gleichungen (z.B. "y = (a AND b) XOR c") oder durch Funktionstabellen.

Der Übergang von der Verhaltenssichtweise auf die strukturale und technologiespezifische Sichtweise erfolgt entweder durch einen manuellen Entwurf oder ein Syntheseprogramm.

1.3.5 Schaltkreisebene

Auch auf der Schaltkreisebene besteht die strukturale Sichtweise aus einer Netzliste. Diesmal sind allerdings keine logischen Gatter kombiniert, sondern elektrische Bauelemente wie Transistoren, Kapazitäten und Widerstände. Einzelne Module werden nun nicht mehr durch eine logische Funktion mit einfachen Verzögerungen beschrieben, sondern durch ihren tatsächlichen Aufbau aus den Bauelementen.

In der geometrischen Sicht werden elektronische Systeme durch Polygonzüge dargestellt, die beispielsweise unterschiedliche Dotierungsschichten auf einem Halbleiter definieren. Die Verhaltenssicht verwendet vornehmlich Differentialgleichungen zur Modellierung des Sy-

stemverhaltens. Dementsprechend aufwendig und rechenzeitintensiv sind die Simulationsalgorithmen.

Die Signale auf Schaltkreisebene können im Gegensatz zur Logikebene prinzipiell beliebige Werte annehmen und weisen einen kontinuierlichen Verlauf über der Zeit auf, d.h. sie sind zeit- und wertkontinuierlich.

Abb. A-2 zeigt die typischen Ebenen beim Entwurf elektronischer Systeme im Überblick.

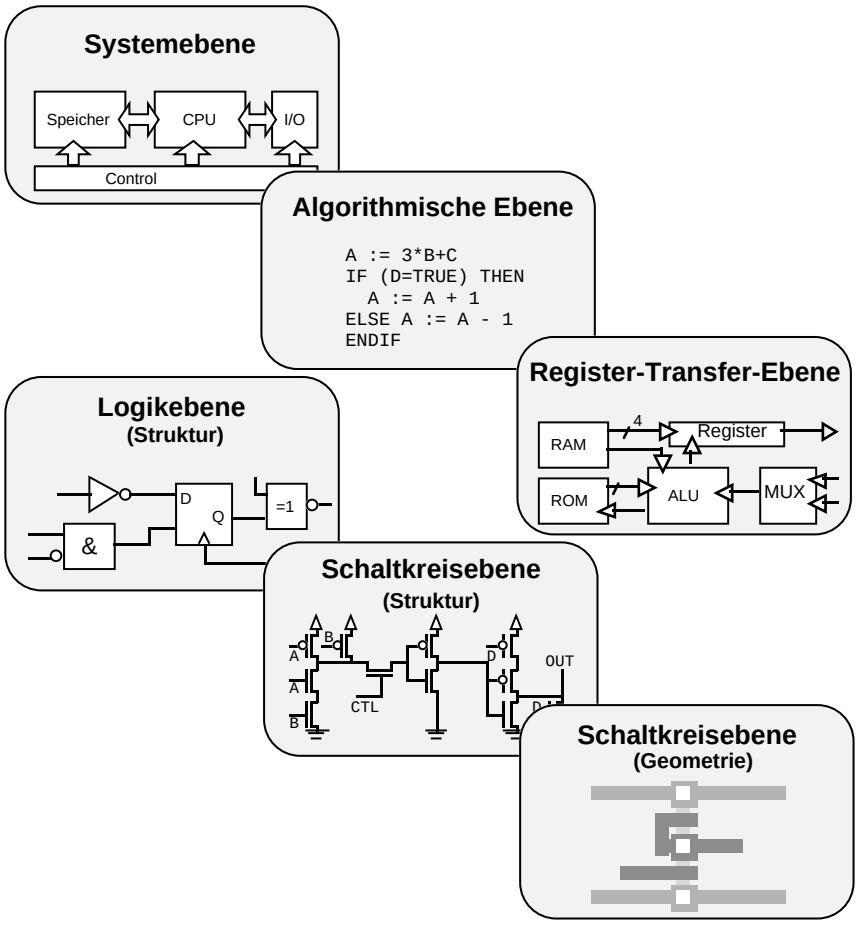

Abb. A-2: Ebenen beim Elektronik-Entwurf

Jede der vorgestellten Entwurfssebenen hat ihren Zweck. Während auf den oberen Ebenen hohe Komplexitäten gut beherrschbar sind, bieten die unteren Ebenen mehr Details und höhere Genauigkeit. So eignen sich die Systemebene und die Algorithmische Ebene für die Dokumentation und Simulation des Gesamtsystems, die Register-Transfer-Ebene für die Blocksimulation und synthesegerechte Modellierung und die Logikebene für Simulationen, mit denen beispielsweise die maximale Taktrate einer Schaltung bestimmt wird oder unerwünschte Impulse ("Spikes") detektiert werden. Auf jeder Ebene wird nur die benötigte Genauigkeit geboten; unwichtige Details sind nicht sichtbar (Abstraktionsprinzip).

Selbstverständlich wird nicht immer eine eindeutige Einordnung einer Beschreibung in eine bestimmten Ebene gelingen, da die Grenzen nicht immer klar definierbar sind. Außerdem ist es möglich, daß die Beschreibung der einzelnen Komponenten eines komplexen, elektronischen Systems auf unterschiedlichen Abstraktionsebenen stattfindet ("Multi-level-Beschreibung").

2 Motivation für eine normierte Hardwarebeschreibungs-sprache

Der Ablauf und die Randbedingungen beim Entwurf elektronischer Systeme haben sich in den letzten Jahren erheblich gewandelt. Dabei treten immer wieder die nachstehenden Probleme auf.

2.1 Komplexität

Die verbesserten Fertigungsmethoden der Halbleitertechnologie gestatten die Entwicklung hochintegrierter Bauelemente. Dies gilt sowohl für die anwendungsspezifischen Schaltkreise (z.B. Standardzellen-Schaltungen) als auch für die anwenderprogrammierbaren Bausteine (FPGAs, GALs, etc.) und die Standardbausteine (RAMs, Prozessoren, etc.). Selbst einfache Produkte, wie z.B. Haushaltsgeräte, enthalten heute umfangreiche elektronische Steuerungen.

Der Elektronikentwickler muß also den Entwurf immer komplexer werdender Systeme beherrschen. Daneben erwarten die Kunden immer kürzere Innovationszyklen bei den Produkten. Dies hat folgende Konsequenzen:

- Ein manueller Entwurf auf Logikebene ist durch graphische oder textuelle Eingabe von Gatternetzlisten nicht mehr beherrschbar, da dieser Vorgang sehr fehleranfällig ist und Simulationen des Gesamtsystems auf Logikebene zu hohe Rechenzeiten erfordern. Die Beschreibung des Systems muß deshalb auf einer abstrakteren Ebene vollzogen und durch geeignete Entwurfssoftware unterstützt werden. Die Überführung in die tieferliegenden Ebenen sollte weitgehend automatisiert sein.

- Entwicklungsaufgaben werden auf mehrere Personen oder Projektteams, die parallel arbeiten, aufgeteilt. Eine fehlerarme Kommunikation zwischen den Entwicklern ist dabei sicherzustellen.
- Entwurfsfehler müssen so früh wie möglich entdeckt werden, um die Kosten und den Zeitbedarf für den notwendigen Neuentwurf ("Redesign") gering zu halten. Um dies zu gewährleisten, sollten bereits Entwürfe der Systemebene oder der Algorithmischen Ebene simulierbar sein ("Simulierbare Spezifikation").
- Die Wiederverwendung von bestehenden Entwürfen kann die Entwicklungskosten und -zeiten verringern sowie das Entwicklungsrisiko eingrenzen. Neben einer geeigneten Dokumentation erfordert die Wiederverwendung Entwürfe mit einem breiten Anwendungsbereich. Details der Implementierung müssen deshalb vermieden werden, d.h. daß beispielsweise der RT-Entwurf eines Halbleiterschaltkreises nicht spezifisch auf eine Fertigungstechnologie ausgerichtet sein darf.

2.2 Datenaustausch

Während des Entwurfs eines elektronischen Systems entsteht eine Vielzahl von Informationen, die in der Summe seine Eigenschaften charakterisieren. Dazu zählen beispielsweise Beschreibungen der Systemfunktionalität, der dynamischen Eigenschaften oder der Einsatzbedingungen. Ein Austausch dieser Informationen findet dabei statt zwischen:

- Auftraggeber und -nehmer (Lasten- und Pflichtenheft),
- verschiedenen Entwicklern eines Projektteams,
- verschiedenen Entwurfsebenen,
- Entwurfsprogrammen verschiedener Hersteller,
- unterschiedlichen Rechnersystemen.

Aufwendige und fehlerträchtige Konvertierungen beim Austausch der Entwurfsdaten können nur vermieden werden, wenn die Beschreibungsmittel herstellerübergreifend normiert sind, keine Spezifika eines bestimmten Rechnersystems enthalten sind und mehrere Entwurfsebenen abgedeckt werden.

Eine Fixierung auf eine herstellerspezifische Beschreibungssprache würde außerdem ein hohes wirtschaftliches Risiko mit sich bringen: Zum einen kann der Hersteller aus verschiedenen Gründen aus dem Markt austreten (Konkurs, Aufkauf durch ein Konkurrenzunternehmen). Zum anderen zieht die Abhängigkeit von einer wenig verbreiteten Beschreibungssprache eine bedeutende Einschränkung bei der Auswahl der Entwurfswerkzeuge nach sich.

2.3 Dokumentation

Die Lebensdauer eines elektronischen Systems ist oft höher als die Verfügbarkeit der Systementwickler. Eine Wartung des Systems, die Wiederverwendung von Systemteilen oder eine technische Modifikation (z.B. der Wechsel auf eine modernere Halbleitertechnologie) machen deshalb eine klare, langlebige und einheitliche Dokumentation notwendig. Die Dokumentation sollte sowohl menschenlesbar als auch durch Rechnerwerkzeuge interpretierbar sein.

Außerdem nimmt mit der steigenden Komplexität der Bausteine auch der Umfang der Dokumentation zu. Die Konsistenz und Vollständigkeit der Beschreibung rücken damit in den Vordergrund.

Eine Verringerung oder Lösung der geschilderten Probleme beim Entwurf elektronischer Systeme ist durch den Einsatz einer Hardwarebeschreibungssprache möglich. Eine solche Sprache, die gleichzeitig für strukturale Beschreibungen und Verhaltensbeschreibungen auf mehreren Entwurfsebenen eingesetzt werden kann, erleichtert den Übergang von der Aufgabenstellung zur Implementierung. Durch die Möglichkeit der Simulation dieser Verhaltensbeschreibung auf hoher Abstraktionsebene kann der Entwurf frühzeitig überprüft werden. Die Synthese der verfeinerten Verhaltensbeschreibung verkürzt die weitere Implementierungszeit ganz erheblich. Ein weiterer wesentlicher Vorteil ist die menschenlesbare Form, die eine Art Selbstdokumentation darstellt. Wird eine solche Hardwarebeschreibungssprache normiert, kann sie auch als Austauschformat zwischen verschiedenen Werkzeugen, Designteams und zwischen Auftraggeber und -nehmer dienen.

Ein Beispiel für eine solche Hardwarebeschreibungssprache ist VHDL.

3 Geschichtliche Entwicklung von VHDL

Die Anfänge der Entwicklung von VHDL[1] reichen bis in die frühen achtziger Jahre zurück. Im Rahmen des VHSIC[2]-Programms wurde in den USA vom Verteidigungsministerium (Department of Defense, DoD) nach einer Sprache zur Dokumentation elektronischer Systeme gesucht. Dadurch sollten die enormen Kosten für Wartung und technische Nachbesserung bei militärischen Systemen, die etwa die Hälfte der Gesamtkosten ausmachten, reduziert werden. Besonderes Augenmerk wurde auf eine Möglichkeit zur sauberen und klaren Beschreibung von komplexen Schaltungen sowie auf die Austauschbarkeit von Modellen zwischen verschiedenen Entwicklungsgruppen gelegt.

Im Rahmen eines vom VHSIC-Programm finanzierten Workshops in Woods Hole, Massachusetts, wurden diese Ziele zuerst diskutiert und dann in eine Vorgabe umgesetzt. Nach mehreren Vorversuchen bekamen im Juli 1983 die Firmen Intermetrics, IBM und Texas Instruments den Auftrag, die neue Sprache mit Programmen zu ihrer Unterstützung zu entwickeln. Die Sprache sollte sich dabei an der existierenden Programmiersprache ADA anlehnen, da das DoD diese Sprache in weitem Umfang einsetzt. ADA-Programmierern werden daher etliche Übereinstimmungen und Ähnlichkeiten auffallen.

Im August 1985 konnte eine erste Version der Hardwarebeschreibungssprache VHDL (VHDL Version 7.2) vorgestellt werden, die dann im Februar 1986 zur Standardisierung an das IEEE[3] übergeben

---

1 VHDL = VHSIC Hardware Description Language

2 VHSIC = Very High Speed Integrated Circuit

3 IEEE = Institute of Electrical and Electronics Engineers

wurde. Mit dieser Aufgabe wurde die VHDL Analysis and Standardization Group (VASG) betraut.

Unter verstärkter Beteiligung der Software- und Hardwarehersteller aus der CAE[1]-Industrie konnte VHDL zunehmend Anhänger gewinnen und wurde schließlich im Dezember 1987 als IEEE 1076-1987 zum ersten und bislang einzigen IEEE-Standard für Hardwarebeschreibungssprachen. Dieser Standard definiert allerdings nur die Syntax und Semantik der Sprache selbst, nicht jedoch ihre Anwendung bzw. ein einheitliches Vorgehen bei der Anwendung. Mittlerweile ist VHDL auch als ANSI[2]-Standard definiert.

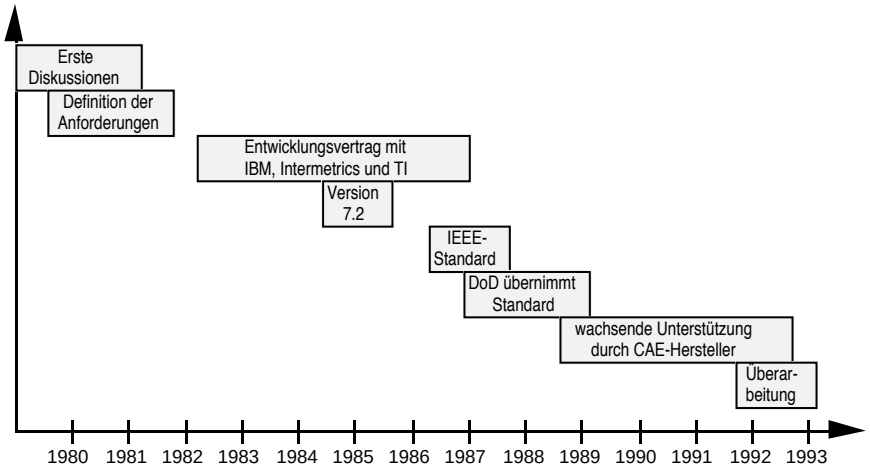

Abb. A-3: Geschichtliche Entwicklung von VHDL

Seit September 1988 müssen alle Elektronik-Zulieferer des DoD VHDL-Beschreibungen ihrer Komponenten und Subkomponenten bereitstellen. Ebenso sind VHDL-Modelle von Testumgebungen mitzuliefern.

---

1 CAE = Computer Aided Engineering

2 ANSI = American National Standards Institute

Nach den IEEE-Richtlinien muß ein Standard alle fünf Jahre überarbeitet werden, um nicht zu verfallen. Aus diesem Grund wurde im Zeitraum Anfang 1992 bis Mitte 1993 die Version IEEE 1076-1993 definiert. Die Dokumentation des neuen Standards, das sog. Language Reference Manual (LRM), wird Mitte 1994 durch das IEEE herausgegeben werden. Die neue Version enthält im wesentlichen "kosmetische" Änderungen gegenüber dem alten Standard, z.B. die Einführung eines XNOR-Operators. Im Teil B dieses Buches wird auf die Unterschiede zwischen dem alten und dem neuen Standard hingewiesen.

Seit Beginn der neunziger Jahre hat VHDL weltweit eine unerwartet große Zahl an Anhängern gefunden. Wurde die 1987er-Version noch durch das DoD finanziert, so ist der neue 1076-1993-Standard unter internationaler Beteiligung entstanden. Europa wirkt unter anderem über ESPRIT (ECIP2) daran mit. Auch Asien (vor allem Japan) hat den VHDL-Standard akzeptiert. Die heutige Konkurrenz von VHDL besteht vor allem in der Verilog HDL, die in den USA weit verbreitet ist, und UDL/I, welches in Japan eingesetzt wird.

Abb. A-4 zeigt eine Übersicht über die heute und in Zukunft verbreiteten Hardwarebeschreibungssprachen (Ergebnis einer Umfrage unter Ingenieuren in den USA [JAI 93]; Mehrfachnennungen waren möglich):

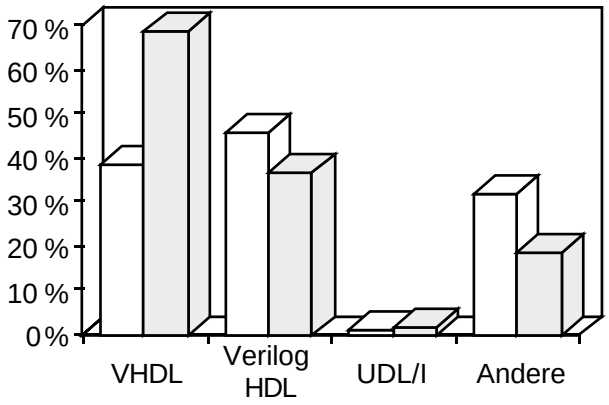

Abb. A-4: Anteile von HDLs heute (hell) und in Zukunft (dunkel) [JAI 93]

4 Aufbau einer VHDL-Beschreibung

Die Beschreibung eines VHDL-Modells besteht meist aus drei Teilen: einer Schnittstellenbeschreibung, einer oder mehreren Architekturen und einer oder mehreren Konfigurationen. VHDL-Beschreibungen können hierarchisch aufeinander aufbauen.

4.1 Schnittstellenbeschreibung (Entity)

In der Entity wird die Schnittstelle der zu modellierenden Komponente / des zu modellierenden Systems beschrieben, also die Ein- und Ausgänge sowie Konstanten, Unterprogramme und sonstige Vereinbarungen, die auch für alle dieser Entity zugeordneten Architekturen gelten sollen.

4.2 Architektur (Architecture)

Eine Architektur enthält die Beschreibung der Funktionalität eines Modells. Hierfür gibt es verschiedene Möglichkeiten. Das Modell kann aus einer Verhaltensbeschreibung bestehen oder strukturalen Charakter (Netzliste) haben. Beide Möglichkeiten können miteinander kombiniert werden. Für eine Entity können mehrere Architekturen definiert werden, d.h. es können für eine Komponentenschnittstelle mehrere Beschreibungen auf unterschiedlichen Abstraktionsebenen oder verschiedene Entwurfsalternativen existieren.

4.3 Konfiguration (Configuration)

In der Konfiguration wird festgelegt, welche der beschriebenen Architekturvarianten einer bestimmten Entity zugeordnet ist und welche Zuordnungen für die möglicherweise verwendeten Submodule in der Architektur gelten. Hier können auch hierarchisch den untergeordneten Entities bestimmte Architekturen zugeordnet werden; außerdem ist es möglich, Parameterwerte an hierarisch tieferliegende Komponenten zu übergeben.

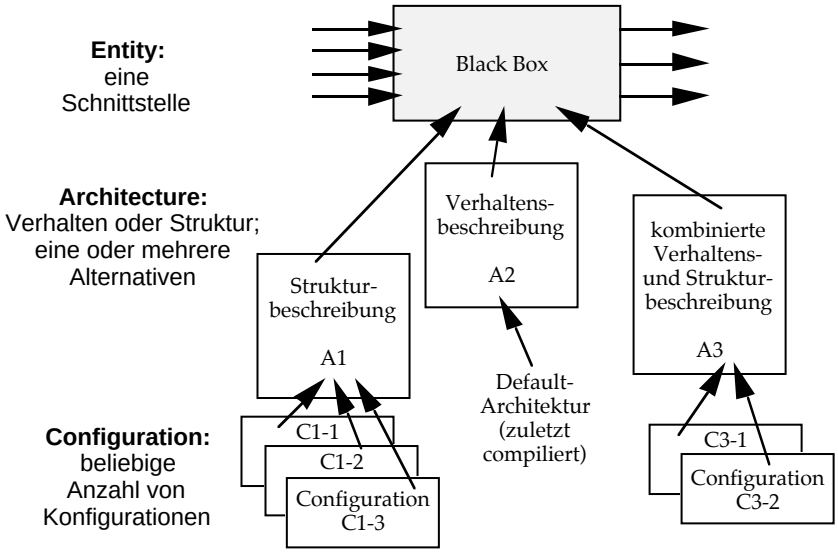

Abb. A-5: Entity, Architecture und Configuration

4.4 Package

Ein Package enthält Anweisungen wie Typ- oder Objektdeklarationen und die Beschreibung von Prozeduren und Funktionen, die in mehreren VHDL-Beschreibungen gebraucht werden. Zum Beispiel kann in einem Package der zu verwendende Logiktyp (zwei- oder mehrwertige Logik) mit allen korrespondierenden Operatoren definiert werden.

Vordefiniert sind bei VHDL die Packages `standard` (enthält den zweiwertigen Logiktyp "`bit`" sowie häufig zu verwendende Funktionen und Typen) und `textio`.

In Abb. A-6 sind die wichtigsten Bestandteile einer VHDL-Beschreibung und ihre Aufgaben im Überblick dargestellt.

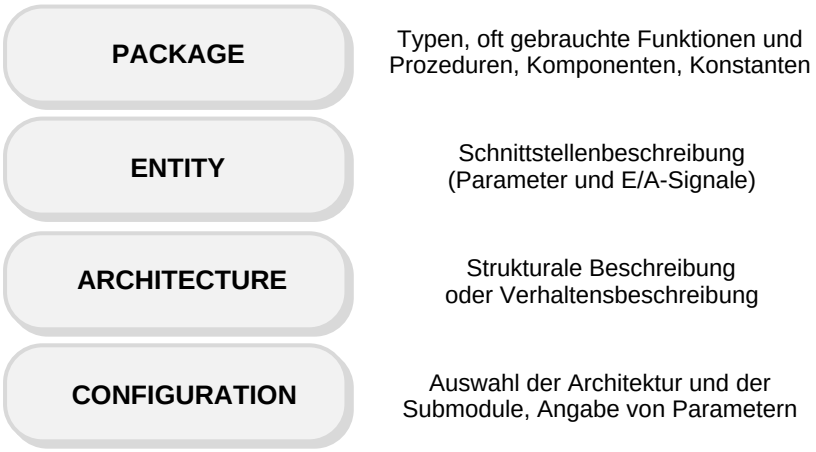

Abb. A-6: Grundbestandteile einer VHDL-Beschreibung

4.5 Beispiel eines VHDL-Modells

Anhand eines einfachen Beispieles soll der grundsätzliche Aufbau eines VHDL-Modells vorab gezeigt werden. Das gewählte AND2-Gatter ist in seinem Aufbau sehr einfach, so daß noch keine Kenntnisse der VHDL-Syntax erforderlich sind, um die Funktion des Modells zu verstehen.

Schnittstellenbeschreibung (Entity)

```
ENTITY and2 IS
   PORT (in1,in2: IN bit; and_out: OUT bit);
   -- definiere Pins als Signale vom Typ "bit"
END and2;
```

Architektur (Architecture)

```
ARCHITECTURE number_one OF and2 IS
BEGIN
   and_out <= in1 AND in2;         -- Verhaltensbeschreibung
END number_one;
```

Konfiguration (Configuration)

```
CONFIGURATION and2_config OF and2 IS
   FOR number_one                -- verknuepfe Architektur
   END FOR;                      -- "number_one" mit Entity
END and2_config;                 -- "and2"
```


5 Entwurfssichten in VHDL

Im vorgestellten Y-Diagramm werden drei Sichten unterschieden. Die Sprache VHDL ermöglicht eine Beschreibung in der strukturalen Sicht und in der Verhaltenssicht. Bei VHDL-Modellen wird deshalb prinzipiell zwischen:

- Verhaltensmodellierung ("behavioral modeling") und
- Strukturaler Modellierung ("structural modeling")

unterschieden. Die Modellierung der geometrischen Sicht eines elektronischen Systems, z.B. die Beschreibung der Layoutdaten, wird von VHDL nicht unterstützt.

5.1 Verhaltensmodellierung

Bei dieser Modellierungsart wird das Verhalten einer Komponente durch die Reaktion der Ausgangssignale auf Änderungen der Eingangssignale beschrieben. Die Komponente verzweigt nicht weiter in Unterkomponenten.

Am Beispiel eines Komparators werden die Vorteile der Verhaltensmodellierung deutlich. Der Komparator soll zwei Bit-Vektoren `a` und `b` miteinander vergleichen und eine logische `'1'` am Ausgang liefern, falls der Wert des Vektors `a` größer als der des Vektors `b` ist.

Das nachfolgende VHDL-Modell kann, gesteuert durch den Parameter `n`, beliebig breite Bit-Vektoren miteinander vergleichen. Eine dieses Verhalten realisierende Schaltung, die aus vielen Gattern aufgebaut ist, kann in der Verhaltenssicht mit VHDL durch wenige Zeilen Quellcode beschrieben werden:

```
ENTITY compare IS
   GENERIC (n  : positive := 4);
   PORT (a, b  : IN  bit_vector (n-1 DOWNTO 0);
         result: OUT bit);
END compare;
```

```
ARCHITECTURE behavioral_1 OF compare IS
BEGIN
   PROCESS (a, b)
   BEGIN
      IF a > b THEN
         result <= '1';
      ELSE
         result <= '0';
      END IF;
   END PROCESS;
END behavioral_1;
```

In der Verhaltenssichtweise werden zwei prinzipielle Beschreibungsmittel unterschieden:

- sequentielle Anweisungen ("sequential statements") und
- nebenläufige Anweisungen ("concurrent statements").

Zur genaueren Betrachtung dieser Zusammenhänge soll ein Halbaddierer dienen. Dessen Schnittstelle ist folgendermaßen aufgebaut:

```
ENTITY halfadder IS
   PORT (sum_a, sum_b: IN bit; sum, carry: OUT bit);
END halfadder;
```

In **sequentiellen** oder auch prozeduralen Beschreibungen werden Konstrukte wie Verzweigungen (`IF-ELSIF-ELSE`), Schleifen (`LOOP`) oder Unterprogrammaufrufe (`FUNCTION`, `PROCEDURE`) verwendet. Die einzelnen Anweisungen werden nacheinander (sequentiell) abgearbeitet. Diese Beschreibungen ähneln den Quelltexten höherer Programmiersprachen.

Eine entsprechende Architektur für den Halbaddierer lautet:

```
ARCHITECTURE behavioral_seq OF halfadder IS
BEGIN
   PROCESS (sum_a, sum_b)
   BEGIN
      IF (sum_a = '1' AND sum_b = '1') THEN
         sum <= '0'; carry <= '1';
      ELSE
         IF (sum_a = '1' OR sum_b = '1') THEN
            sum <= '1'; carry <= '0';
         ELSE
            sum <= '0'; carry <= '0';
         END IF;
      END IF;
   END PROCESS;
END behavioral_seq;
```

Im Gegensatz zu den gängigen Programmiersprachen mit ihren sequentiellen Konstrukten verfügt VHDL zusätzlich noch über **nebenläufige** Anweisungen, die es erlauben, parallel ablaufende Operationen zu beschreiben. Damit wird es möglich, die spezifischen Eigenschaften von Hardware (parallel arbeitende Funktionseinheiten) abzubilden.

Nachstehend ist die Architektur des Halbaddierers in dieser Beschreibungsvariante gezeigt. Die beiden Verknüpfungen `XOR` und `AND` können dabei gleichzeitig aktiv sein:

```
ARCHITECTURE behavioral_par OF halfadder IS
BEGIN
   sum   <= sum_a XOR sum_b;
   carry <= sum_a AND sum_b;
END behavioral_par;
```

5.2 Strukturale Modellierung

Bei der strukturalen Modellierung werden die Eigenschaften eines Modells durch seinen inneren Aufbau aus Unterkomponenten dargestellt. Die Eigenschaften der Unterkomponenten werden in unabhängigen VHDL-Modellen beschrieben. Diese stehen compiliert in Modellbibliotheken ("Libraries") zur Verfügung.

Für den Halbaddierer, der aus einem XOR2- und einem AND2-Gatter aufgebaut ist, ergibt sich beispielsweise folgende Beschreibung:

```
ARCHITECTURE structural OF halfadder IS
   COMPONENT xor2
      PORT (c1, c2: IN bit; c3: OUT bit);
   END COMPONENT;
   COMPONENT and2
      PORT (c4, c5: IN bit; c6: OUT bit);
   END COMPONENT;
BEGIN
   xor_instance: xor2 PORT MAP (sum_a, sum_b, sum);
   and_instance: and2 PORT MAP (sum_a, sum_b, carry);
END structural;
```

Eine eindeutige Einordnung eines VHDL-Modells in eine der beiden Modellierungsarten ist nicht immer möglich, da VHDL die Verwendung beider Beschreibungsmöglichkeiten innerhalb eines Modells gestattet.

6 Entwurfsebenen in VHDL

Die weiten Modellierungsmöglichkeiten von VHDL unterstützen Beschreibungen in verschiedenen Entwurfsebenen, ausgehend von der Systemebene bis hinab zur Logikebene. Folgende drei Beschreibungsebenen haben dabei die größte Bedeutung:

- Algorithmische Ebene,
- Register-Transfer-Ebene,
- Logikebene.

Daneben finden sich in der VHDL-Literatur Ansätze, die zeigen, daß auch eine Modellierung auf Schaltkreisebene bedingt möglich ist (z.B. [HAR 91]). Die Praxisrelevanz dieser Ansätze ist jedoch gering.

6.1 Algorithmische Ebene

Ein Beispiel für eine Beschreibung auf Algorithmischer Ebene zeigt einen Ausschnitt aus der Architektur eines Schnittstellenbausteins. Der Baustein soll immer dann, wenn er von einem Controller eine Aufforderung erhält, eine Adresse aus einem internen Register frühestens nach 10 ns auf den Bus legen.

Diese Beschreibung enthält keine Angaben über die spätere Schaltungsstruktur und keine Takt- oder Rücksetzsignale.

```
ARCHITECTURE algorithmic_level OF io_ctrl IS

BEGIN
   ...
   write_data_alg: PROCESS

   BEGIN
      WAIT UNTIL adr_request = '1';
      WAIT FOR 10 ns;
      bus_adr <= int_adr;
   END PROCESS write_data_alg;
   ...
END algorithmic_level;
```

6.2 Register-Transfer-Ebene

Um das obige Beispiel auf RT-Ebene darzustellen, wird ein Taktsignal (`clk`) und ggf. ein Rücksetzsignal hinzugefügt und die Operationen in Abhängigkeit von diesen Signalen beschrieben:

```
ARCHITECTURE register_transfer_level OF io_ctrl IS

BEGIN
   ...
   write_data_rtl : PROCESS (clk)
      VARIABLE tmp : boolean;
   BEGIN
      IF rising_edge(clk) THEN
         IF ((adr_request = '1') AND (tmp = false)) THEN
            tmp := true;
         ELSIF (tmp = true) THEN
            bus_adr <= int_adr;
            tmp := false;
         END IF;
      END IF;
   END PROCESS write_data_rtl;
   ...
END register_transfer_level;
```

Hier wird, wenn bei einer aktiven Taktflanke ein gesetztes `adr_req`-Signal entdeckt wird, zunächst die temporäre Variable `tmp` ge-

setzt, damit bei der nächsten aktiven Taktflanke (Wartezeit!) die Adresse auf den Bus geschrieben werden kann. Durch geeignete Wahl der Taktperiode ist sicherzustellen, daß die Wartezeit von mindestens 10 ns eingehalten wird.

Im Gegensatz zur Algorithmischen Ebene wird hier schon ein zeitliches Schema für den Ablauf der Operationen vorgegeben und implizit eine Schaltungsstruktur beschrieben.

Wie die beiden Beispielarchitekturen zeigen, werden Konstrukte der Verhaltensmodellierung sowohl auf Algorithmischer als auch auf Register-Transfer-Ebene verwendet.

6.3 Logikebene

Die Eigenschaften eines elektronischen Systems werden auf der Logikebene durch logische Verknüpfungen digitaler Signale und deren zeitliche Eigenschaften (i.a. durch Verzögerungszeiten der Verknüpfungen) beschrieben. Die Hardwarebeschreibungssprache VHDL besitzt dazu vordefinierte Operatoren (`AND`, `OR`, `XOR`, `NOT` etc.) für binäre Signale (`'0'`, `'1'`) und gestattet die Ergänzung weiterer, benutzerdefinierter Operatoren. Auch Konstrukte zur Modellierung zeitlicher Eigenschaften werden bereitgestellt. Nachstehend ist beispielhaft die Beschreibung der Halbaddierer-Architektur auf der Logikebene in der Verhaltenssichtweise abgebildet.

```
ARCHITECTURE logic_level OF halfadder IS
BEGIN

   sum   <= sum_a XOR sum_b AFTER 3 ns;
   carry <= sum_a AND sum_b AFTER 2 ns;

END logic_level;
```

Die Darstellung in strukturaler Sicht entspricht der obigen Architektur "`structural`". Die Verzögerungszeiten ergeben sich hier aus den internen Verzögerungszeiten der beiden Subkomponenten.

7 Design-Methodik mit VHDL

7.1 Entwurfsablauf

Abb. A-7 zeigt, wie der Entwurfsablauf unter Verwendung von VHDL aussehen könnte:

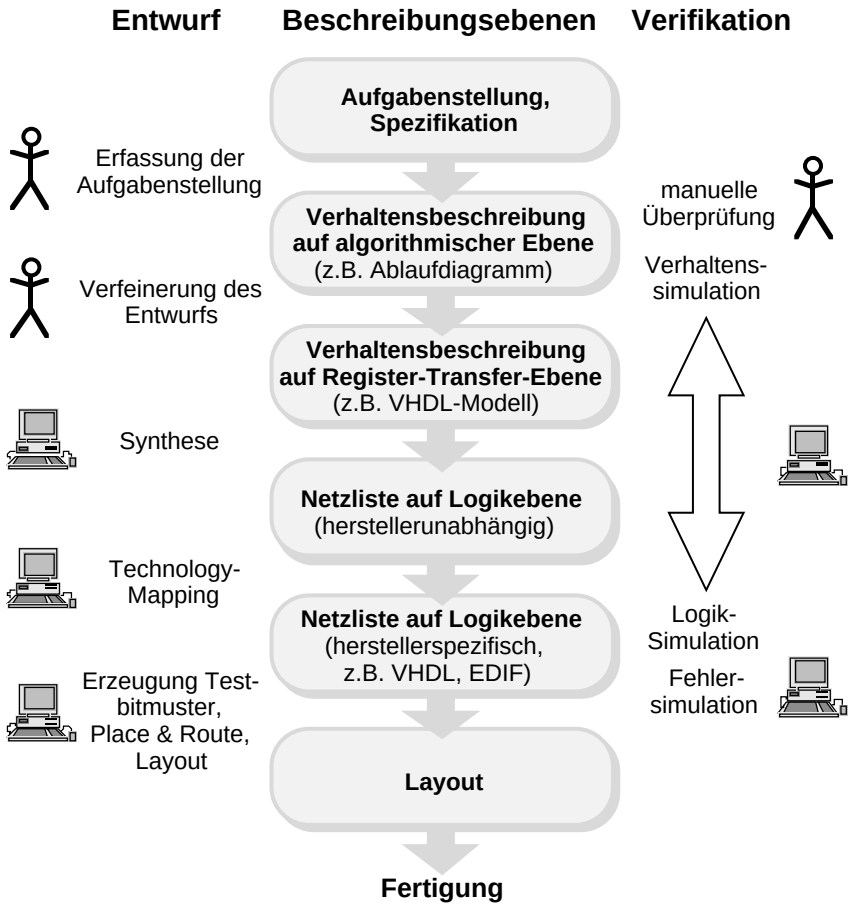

Abb. A-7: Entwurfsablauf mit VHDL

7.1.1 Erfassung der Spezifikation

Am Anfang steht die Erfassung der Spezifikation, die i.d.R. vom Auftraggeber in Form von Texten, Tabellen und Diagrammen an den Auftragnehmer übergeben wird. Diese Spezifikation enthält sowohl funktionale (Beschreibung der Schaltungsfunktion) als auch nichtfunktionale Angaben (Versorgungsspannung, Fertigungstechnologie, maximale Fläche und Verlustleistung, etc.).

Nach einem ggf. vorhandenen Zwischenschritt über manuell erstellte Graphiken (Ablaufpläne, Flußdiagramme, etc.) erfolgte **bisher** der Entwurf des elektronischen Systems von Hand bis zu einer Netzliste, die die Verdrahtung vorhandener Module aus einer technologiespezifischen Bibliothek definiert. Diese Netzliste wurde durch Schematic-Entry-Werkzeuge in den Rechner eingegeben und im weiteren mit entsprechenden Programmen bis zum Layout umgesetzt.

7.1.2 Beschreibung und Simulation auf System- oder Algorithmischer Ebene

Heute ist man jedoch in der Lage, auch in den frühen Phasen des Elektronik-Entwurfs gezielt mit Rechnerunterstützung zu arbeiten. Durch Verwendung einer Hardwarebeschreibungssprache wie VHDL kann nun, ausgehend von einer nichtformalen Spezifikation, auf der abstrakten Ebene von Algorithmen und Funktionsblöcken bereits ein erstes VHDL-Modell der Schaltung erstellt werden. Eine anschließende Simulation der Schaltung auf dieser Ebene zeigt die Korrektheit des Entwurfs oder veranlaßt schon sehr frühzeitig notwendige Korrekturen.

Für viele Einsatzwecke existieren auch Werkzeuge, die eine graphische Eingabe des Entwurfs ermöglichen. Die Erstellung und Simulation abstrakter Verhaltensmodelle wird also graphisch unterstützt. Häufig bieten solche Programme auch die Möglichkeit, VHDL-Code auf Register-Transfer-Ebene zu generieren. Stehen solche Werkzeuge nicht zur Verfügung, muß das VHDL-Modell auf Register-Transfer-Ebene manuell erstellt werden.

7.1.3 Beschreibung auf Register-Transfer-Ebene

Eine Beschreibung auf Algorithmischer Ebene kann von heute kommerziell verfügbaren Synthesewerkzeugen nicht in eine Gatternetzliste umgesetzt werden. Diese Programme erwarten eine Beschreibung auf Register-Transfer-Ebene. Die bereits erstellte abstrakte Beschreibung muß also manuell verfeinert werden. Dabei ist die Kenntnis des zu verwendenden Synthesewerkzeuges und des von ihm unterstützten VHDL-Sprachumfanges äußerst wichtig.

7.1.4 Logiksynthese und Technology Mapping

Bei der anschließenden Synthese der VHDL-RTL-Beschreibung wird weitgehend automatisch eine noch technologieunabhängige Beschreibung in der Logikebene erzeugt. Diese wird dann, nach Auswahl einer Zieltechnologie (z.B. Standardzellen-Prozeß 0.7 µm der Firma XYZ), in eine technologiespezifische Gatternetzliste umgesetzt. Der Begriff "Technology Mapping" bezeichnet diesen Vorgang. Er wird meist durch das Synthesewerkzeug durchgeführt. Anschließende Analysen können zeigen, ob die spezifizierten Eigenschaften, wie maximal zulässige Chipfläche, erreicht worden sind. Logiksimulationen lassen Aussagen über die Einhaltung von Zeitbedingungen zu.

7.1.5 Erzeugung der Testbitmuster

Nach der Erzeugung der Gatternetzliste müssen für den Produktionstest die Testbitmuster generiert werden. An dieser Stelle sind meist Modifikationen des Entwurfs notwendig, um eine gute Testbarkeit der Schaltung zu erreichen. Häufig wird der Entwickler auch bei dieser Tätigkeit durch Programme unterstützt, die das VHDL-Netzlistenmodell als Eingabe akzeptieren.

7.1.6 Layouterzeugung und Produktion

Ausgehend von der technologiespezifischen VHDL-Gatternetzliste werden im abschließenden Entwurfsschritt die für die Produktion des elektronischen Systems notwendigen Layout- oder Programmierdaten erstellt. Für diesen Entwurfsschritt existiert eine Vielzahl kommerzieller Werkzeuge.

Nach Erzeugung des Layouts können sehr genaue Aussagen über die Laufzeiten von Signalen gemacht werden, da jetzt die exakten geometrischen Daten der Verdrahtung bekannt sind. Diese "realen" Verzögerungszeiten werden in die Logikebene zurückgereicht ("Backannotation"), so daß in der Logiksimulation die vom Layout abhängigen Einflüsse berücksichtigt werden können.

7.2 VHDL-Software

Im Bereich des Entwurfs elektronischer Systeme gibt es unterstützende Werkzeuge für Spezifikationserfassung und Dokumentation, funktionale Simulation auf verschiedenen Abstraktionsebenen, Schaltplaneingabe, Fehlersimulation, Synthese, Layout oder Test.

Nicht alle Entwurfsschritte sind sinnvoll mit VHDL durchzuführen. Die Schwerpunkte beim Einsatz von VHDL liegen heute im Bereich der Simulation von Verhaltensmodellen der Algorithmischen oder der Register-Transfer-Ebene sowie der Synthese.

7.2.1 Texteditoren

Der erste Schritt bei der Arbeit mit VHDL besteht in der Regel darin, VHDL-Quellcode einzugeben. Dazu kann jeder beliebige Texteditor verwendet werden (emacs, vi, textedit, etc.). Manche Software-Hersteller bieten im Rahmen ihrer VHDL-Programme einen eigenen VHDL-Editor an. Solche speziellen Editoren können beispielsweise VHDL-Schlüsselwörter (durch Fettdruck o. ä.) besonders hervorheben oder sprachspezifisch auf Eingaben reagieren, z.B. durch automatisches Schließen von Blöcken mit "`END`"-Anweisungen.

7.2.2 Graphische Benutzerschnittstellen

Die Rechnerunterstützung in den frühen Entwurfsphasen gewinnt immer mehr an Bedeutung. Werkzeuge zur Erfassung eines Entwurfs auf hoher Abstraktionsebene finden zunehmende Verbreitung. Oft sind graphische Eingaben (Zeichnen von Automatengraphen oder Warteschlangenmodellen, etc.) möglich.

Eine frühzeitige Simulation des Entwurfs und die Ausgabe von (teilweise garantiert synthetisierbarem) VHDL-Code sind weitere Pluspunkte für diese Art von Programmen.

7.2.3 Simulatoren und Debugger

VHDL-Simulatoren dienen dazu, Schaltungen mit vorgegebenen Stimuli zu simulieren, d.h. die Reaktion aller internen Knoten und insbesondere der Ausgänge auf vorgegebene Änderungen der Eingänge zu ermitteln. Speziell zur Simulation von VHDL-Verhaltensmodellen gehört auch, daß der Zeitverlauf von VHDL-Objekten (Signalen und Variablen) dargestellt werden kann.

Gerade wegen des nebenläufigen Konzepts einer Hardwarebeschreibungssprache sind zusätzlich Funktionen notwendig, die von typischen Software-Debuggern bekannt sind. Dazu zählt beispielsweise das Setzen von sog. Breakpoints, welche die Simulation bei Eintreten einer bestimmten Bedingung (Erreichen einer vorgegebenen Simulationszeit, spezielle Werte von Objekten, etc.) anhalten.

Die meisten kommerziell angebotenen VHDL-Simulatoren können inzwischen jedes beliebige VHDL-Modell verarbeiten (100%-ige Codeabdeckung). Sieht man von wenigen Details ab, die vor allem auf Ungenauigkeiten der VHDL-Norm zurückzuführen sind, kann man von einer einheitlichen Interpretation eines VHDL-Modells durch verschiedene Simulatoren ausgehen.

Erwähnt werden sollte noch, daß auf Logikebene spezialisierte Simulatoren zur Zeit noch deutlich schneller als VHDL-Simulatoren sind. Die Hersteller der VHDL-Simulatoren arbeiten allerdings intensiv an einer Performance-Verbesserung durch neue Simulationstechniken (siehe Teil C) bzw. bieten eine Anbindung ihrer Werkzeuge an spezialisierte "Gate-Level-Simulatoren" an.

7.2.4 Syntheseprogramme

Syntheseprogramme können am meisten dazu beitragen, den Entwurf elektronischer Systeme produktiver zu gestalten. Sie setzen eine funktionale VHDL-Beschreibung (auf Register-Transfer-Ebene) in eine Netzliste auf Logikebene um. Zu diesem Zweck ist eine technologiespezifische Gatterbibliothek erforderlich. Sie enthält Informationen über Fläche, Laufzeiten, Verlustleistung usw. der einzelnen Gatter.

Ein wesentlicher Gesichtspunkt bei der Bewertung eines Syntheseprogrammes ist der Umfang von VHDL-Konstrukten, die synthetisiert werden können. Im Gegensatz zur Simulation kann bei der Synthese nicht der komplette Sprachumfang verarbeitet werden. Dies liegt v.a. daran, daß VHDL nicht primär auf Synthesezwecke hin ausgerichtet wurde, sondern auch viele Konstrukte enthält, die prinzipiell nicht in Hardware umgesetzt werden können.

Außerdem ist zu überprüfen, ob für eine gewünschte Zieltechnologie die werkzeugspezifische Gatterbibliothek verfügbar ist.

8 Bewertung von VHDL

Die vorhergehenden Kapitel haben gezeigt, daß der Einsatz einer normierten Hardwarebeschreibungssprache viele Vorteile für den Entwurf komplexer Elektronik bietet. Der erste Teil des Buches sollte jedoch nicht nur euphorisch VHDL loben, sondern auch einige kritische Worte enthalten. Im folgenden finden sich deshalb einige grundsätzliche Bemerkungen zu VHDL, die allerdings nicht immer objektiv möglich sind. Zum Beispiel kann die umfangreiche Syntax sowohl als Vorteil (viele Modellierungsmöglichkeiten), als auch als Nachteil (hoher Einarbeitungsaufwand) empfunden werden.

8.1 Vorteile von VHDL

8.1.1 Vielseitigkeit

Zweifellos als Vorteil kann es angesehen werden, daß VHDL eine Sprache für viele Zwecke ist.

VHDL ist sowohl für die Spezifikation und Simulation wie auch als Ein- und Ausgabesprache für die Synthese geeignet. Die menschenlesbare Form eignet sich gut zur Dokumentation. Beispielsweise bietet VHDL über benutzerdefinierte Attribute die Möglichkeit, entwurfsbegleitende Angaben, wie Vorgaben zur Fläche oder Laufzeit, zu dokumentieren.

Schließlich ist durch die firmenunabhängige Normierung der Sprache ein Datenaustausch zwischen verschiedenen Programmen, zwischen verschiedenen Entwurfsebenen, zwischen verschiedenen Projektteams und zwischen Entwickler und Hersteller möglich (siehe Abb. A-8).

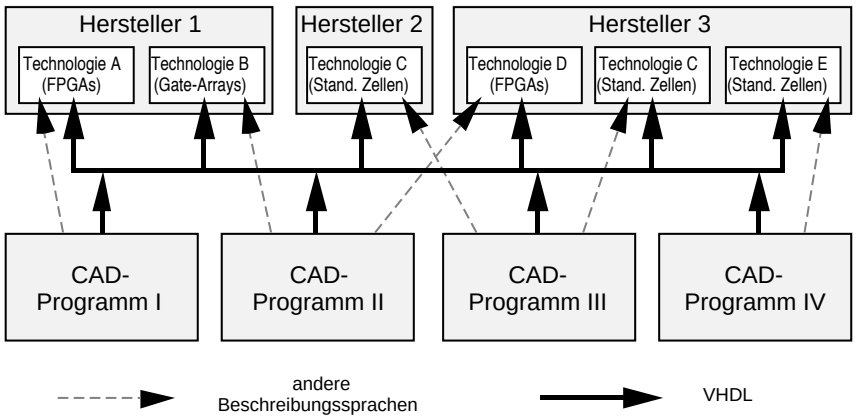

Abb. A-8: Datenaustausch zwischen CAD-Programmen und Halbleiterherstellern

8.1.2 Programmunabhängigkeit

Neben VHDL gibt es auch noch eine Reihe anderer Hardwarebeschreibungssprachen und Datenformate. Man unterscheidet dabei zwischen programmspezifischen und nicht-programmspezifischen Sprachen bzw. Formaten. Erstere sind an die Programme eines bestimmten Software-Herstellers gebunden und werden meist nicht von den Programmen anderer Hersteller unterstützt. Auch die heutige Hauptkonkurrenz von VHDL, die Verilog HDL, war lange Zeit herstellerspezifisch und hat sich erst in neuerer Zeit zu einer unabhängigen HDL entwickelt.

VHDL ist programmunabhängig. Es gibt mittlerweile eine Vielzahl von Software-Anbietern, die für viele Entwurfsschritte eine Lösung unter Einsatz von VHDL anbieten. Man kann somit unter mehreren Alternativen die für die geplante Anwendung optimale Software auswählen.

Außerdem wurde bei der Definition der Sprache VHDL sehr stark auf die Unabhängigkeit von einem bestimmten Rechnersystem geachtet. Die systemabhängigen Aspekte werden in Packages gekapselt, das VHDL-Modell selbst ist unabhängig vom eingesetzten Rechnersystem und damit (fast immer) portierbar.

8.1.3 Technologieunabhängigkeit

VHDL ist technologieunabhängig. Die Entscheidung für eine bestimmte Technologie (FPGA, Gate-Array, Standardzellen, etc.) muß erst zu einem relativ späten Zeitpunkt des Entwurfs getroffen werden. Ein späteres Umschwenken auf eine andere Technologie verursacht kein komplettes Redesign.

Auch die Offenheit der Sprache bzw. die umfangreichen Modellierungsmöglichkeiten tragen wesentlich zur Technologieunabhängigkeit bei. Durch die Freiheit zur Definition von:

- benutzereigenen Logiktypen mit entsprechenden Operatoren,
- Auflösungsfunktionen, die technologiespezifische Signalverknüpfungen (wired-or, wired-and, etc.) modellieren,
- neuen, bei strukturalen Modellen eingesetzten Komponentenbibliotheken

können die spezifischen Eigenschaften bestimmter Technologien mit VHDL abgebildet werden. Man sagt deshalb auch, VHDL arbeitet technologieunterstützend.

8.1.4 Modellierungsmöglichkeiten

VHDL stellt zahlreiche Konstrukte zur Beschreibung von Schaltungen und Systemen zur Verfügung. Mit diesen Konstrukten lassen sich Modelle auf verschiedenen Beschreibungsebenen erstellen: Algorithmische Ebene, Register-Transfer-Ebene und Logikebene. VHDL-Modelle können dabei Unterkomponenten verschiedener Entwurfssichten und -ebenen enthalten. Bei der strukturalen Modellierung kann eine mehrstufige Hierarchie implementiert werden.

Beschreibungen auf abstraktem Niveau haben mehrere Vorteile:

- sie sind kompakt und überschaubar, der Überblick über den gesamten Entwurf bleibt erhalten,
- sie ermöglichen kürzere Entwicklungszeiten,
- sie benötigen weniger Rechenzeit bei der Simulation,
- sie erlauben eine frühzeitige Verifikation.

Durch Kombination mit detaillierteren Beschreibungen ist eine ebenenübergreifende Simulation (sog. "Multi-Level-Simulation") möglich. Einzelne Projektteams können deshalb weitgehend unabhängig voneinander arbeiten, da eine Gesamtsimulation unterschiedlich weit verfeinerter Modelle durchführbar ist. Außerdem gestattet die Multi-Level-Simulation die Kontrolle eines Entwurfsschrittes, indem die Modelle vor und nach dem Entwurfsschritt durch gemeinsame Simulation miteinander verglichen werden können.

8.1.5 Unterstützung des Entwurfs komplexer Schaltungen

Die durch VHDL verstärkte Verbreitung von Synthesewerkzeugen erlaubt ein Umschwenken auf eine neue und produktivere Entwurfsmethodik mit strukturierter Top-Down-Vorgehensweise. Wesentliche Aspekte dabei sind:

- Da die Spezifikation eine simulierbare Beschreibung darstellt, kann der Entwurf frühzeitig überprüft werden,
- Verhaltensbeschreibungen in einer Hardwarebeschreibungssprache können synthetisiert werden,
- Zahlreiche Sprachkonstrukte zur Parametrisierung von Modellen erlauben ein unkompliziertes Variantendesign,
- Die Entwicklung wiederverwendbarer Modelle wird unterstützt,
- Es bestehen Umsetzungsmöglichkeiten auf verschiedene Technologien.

Alles zusammen führt zur Beherrschung von komplexeren Schaltungen und zu einer wesentlich verkürzten Entwicklungszeit.

8.1.6 Selbstdokumentation

VHDL ist eine menschenlesbare Sprache. Die Syntax ist sehr ausführlich gehalten und besitzt viele selbsterklärende Befehle. Dadurch entsteht eine Art Selbstdokumentation.

Bei der zusätzlichen Wahl von geeigneten Objektnamen enthält eine ohne weitere Kommentare versehene Beschreibung durchaus genü-

gend Informationen, um zu einem späteren Zeitpunkt noch ohne Probleme interpretiert werden zu können.

8.2 Nachteile von VHDL

8.2.1 Mehr als eine Sprache: Eine neue Methodik

Mit dem Einsatz von VHDL beim Entwurf ist weit mehr verbunden als bei der Einführung eines neuen Programmes oder eines neuen Formates. Der gesamte Entwurfsablauf hat sich vom manuellen Entwurf mit Schematic Entry auf Logikebene zur Schaltungsbeschreibung auf RT-Ebene mit anschließender Synthese gewandelt.

Dies hat mehrere Folgen:

- Es ist ein grundsätzlich neuer Entwurfsstil einzuführen, verbunden mit einem Umdenken bei den Hardware-Entwicklern. Erfahrungsgemäß haben aber gerade Entwickler, die in ihrer Ausbildung nicht mit modernen Programmiersprachen und der erforderlichen strukturierten Vorgehensweise vertraut gemacht wurden, dabei große Probleme: *"We don't know if to 'harden' a Software engineer or to 'soften' a Hardware engineer",* [BUR 92].
- Die erforderlichen Aus- und Weiterbildungsmaßnahmen verursachen Kosten und Ausfallzeiten.
- Die anfallenden Kosten für die Neuanschaffung einer ganzen Werkzeugpalette (Workstations, Speicher, Lizenzgebühren für Software, etc.) sind enorm.

8.2.2 Modellierung analoger Systeme

VHDL verfügt zwar über umfangreiche Beschreibungsmittel für digitale, elektronische Systeme, bietet aber auch in der neuen Norm (VHDL'93) keine Konstrukte zur Modellierung analoger elektronischer Systeme. Das gleiche gilt für Komponenten mit mechanischen, optischen, thermischen, akustischen oder hydraulischen Eigenschaften und Funktionen. Damit ist VHDL keine Sprache, die die vollständige Verhaltensmodellierung eines technischen Systems zuläßt.

Aktuelle Bestrebungen zielen allerdings auf die Definition einer erweiterten VHDL-Norm (IEEE 1076.1, AHDL), die die Modellierung analoger Schaltkreise mit wert- und zeitkontinuierlichen Signalen gestattet.

8.2.3 Komplexität

Die Komplexität der Sprache VHDL wird als Vorteil aufgrund der vielen Modellierungsmöglichkeiten angesehen, bringt aber auch einige Nachteile mit sich:

- VHDL erfordert einen hohen Einarbeitungsaufwand. Es ist mit einer weitaus längeren Einarbeitungszeit als bei anderen Sprachen zu rechnen. Insbesondere muß ein geeigneter Modellierungsstil eingeübt werden.
- Das Verhalten eines komplexen VHDL-Modells in der Simulation ist für den VHDL-Neuling kaum nachvollziehbar, da die zugehörigen Mechanismen (z.B. "preemption" von Ereignislisten) nicht von gängigen Programmiersprachen abgeleitet werden können.
- Die Semantik wurde in der ursprünglichen Version 1987 an vielen Stellen nicht eindeutig und klar genug festgelegt. In der Überarbeitung der Sprache wurden 1993 einige dieser Stellen beseitigt und die Interpretation damit vereinheitlicht.

Erschwerend zu diesen Problemen kommt, daß das Nachschlagewerk für VHDL, das "Language Reference Manual" (LRM), als eines der am schwersten zu lesenden Bücher der Welt einzustufen ist (*"The base type of a type is the type itself"*, [IEE 88]).

8.2.4 Synthese-Subsets

VHDL ist als Sprache in der Syntax und Simulationssemantik normiert, nicht jedoch für die Anwendung als Eingabeformat für Synthesewerkzeuge. Außerdem enthält VHDL Konstrukte, die sich prinzipiell nicht in eine Hardware-Realisierung umsetzen lassen. Darüber hinaus unterstützt jedes Synthesewerkzeug einen etwas anderen VHDL-Sprachumfang (VHDL-Subset) und erfordert einen spezifischen Modellierungsstil. Daraus resultieren folgende Probleme:

- VHDL-Modelle müssen auf ein spezielles Synthesewerkzeug zugeschnitten sein. Dies verhindert einen unkomplizierten Wechsel des Werkzeugs und erhöht die Abhängigkeit vom Werkzeughersteller.
- Der Elektronik-Entwickler muß die Anforderungen des gewählten Werkzeuges kennen und von Anfang an bei der Modellerstellung berücksichtigen.

8.2.5 Noch keine umfassende Unterstützung

Obwohl seit Ende der 80er Jahre die Unterstützung von VHDL durch Werkzeuge vieler Software-Hersteller enorm zugenommen hat, ist die heutige Situation noch nicht zufriedenstellend.

Ein Mangel besteht vor allem bei den Simulations- und Synthesebibliotheken für logische Gatter und Standardbausteine. Jede neue Technologie erzwingt eine komplette Neufassung der umfangreichen Daten, die oft erst zeitverzögert zur Verfügung gestellt werden. In diesem Punkt ist die schon länger existierende Hardwarebeschreibungssprache Verilog der neueren Sprache VHDL überlegen, da die Zahl der bestehenden Verilog-Bibliotheken noch weitaus größer ist.

8.2.6 Ausführlichkeit

Die Ausführlichkeit der Sprache VHDL kann auch als Nachteil empfunden werden. Der oft als "zu geschwätzig" empfundene Stil verursacht lange und umständliche Beschreibungen. Vor allem bei der manuellen Modellerstellung verhindert der Umfang des einzugebenden Textes ein schnelles Vorgehen.

