10 Hardwarebeschreibungssprache VHDL

10.1 Motivation

Aktuelle Situation beim Entwurf elektronischer Systeme

- Integrationsgrad und Integrationsdichte steigen kontinuierlich
- Konkurrenzdruck und Kundenanforderungen bedingen kurze Entwicklungszeiten

Gründe für vollständige, widerspruchsfreie und verständliche Dokumentation

- Komplexität, Modularisierung
- Wiederverwendung von Entwurfsdaten und
- Wartung des fertigen Produktes

Kompatibilität der Entwurfsdaten

- Beschreibungsmittel müssen herstellerübergreifend normiert, rechnerunabhängig sein und mehrere Entwurfsebenen abdecken - Fixierung auf herstellerspezifische Beschreibungssprache kann hohes wirtschaftliches Risiko bedeuten

10.2 Entwurfssichten

Entwurfsfluss (Top-down-Entwurf) Der Entwurf komplexer elektronischer Systeme kann nur durch eine strukturierte Vorgehensweise beherrschbar gestaltet werden. Systemspezifikation wird die Schaltungsfunktion partitioniert (funktionale Dekomposition) und die Funktionen einzelnen Modulen zugeordnet. Schrittweise wird der Entwurf feiner strukturiert und zunehmend mit Implementierungsdetails versehen, bis die für die Fertigung notwendigen Daten vorliegen. Dieses sind z. B. Programmierdaten für Logikbausteine, Layouts für PCB‘s oder Maskenbänder für IC-Fertigung.

Sichtweisen beim Entwurf elektronischer Systeme

- Verhalten,
- Struktur und
- Geometrie.

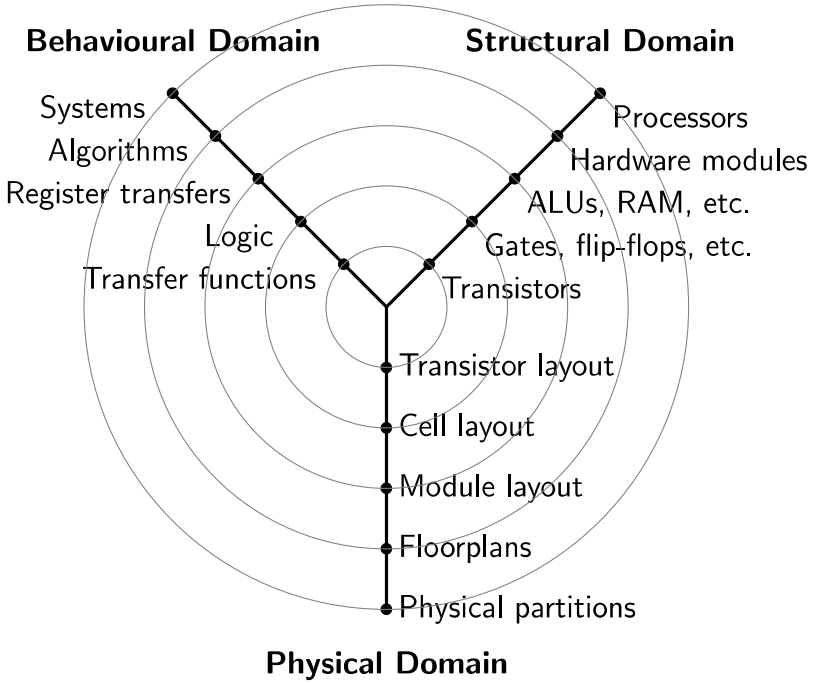


Abbildung 10.1: Gajski-Kuhn Y-Diagramm

Zu den drei Sichtweisen (Äste im Y-Diagramm) existieren verschiedene Abstraktionsebenen. Der Entwurf elektronischer Systeme kann als Reihe von Transformationen (Wechsel der Sichtweise) und Verfeinerungen (Wechsel der Abstraktionsebene innerhalb einer Sichtweise) im Y-Diagramm dargestellt werden.

10.3 Historische Entwicklung

1983 Start der VHDL-Entwicklung in den USA Arbeiten im Rahmen des VHSIC-Programms (Very High Speed Integrated Circuit). Ziel der Sprache: Austausch von Entwürfen zwischen Herstellern Anlehnung an Ada, da das Verteidigungsministerium diese Sprache in weitem Umfang einsetzt. 1987 Übernahme von VHDL als IEEE-Standard (IEEE 1076-1987). Dieser erste und bislang einzige IEEE-Standard für HDL definiert nur die Syntax und Semantik der Sprache, nicht jedoch ihre Anwendung bzw. einheitliches Vorgehen bei der Anwendung, insbesondere Synthese von Anfang 1992 bis Mitte 1993 Definition der Version IEEE 1076-1993 Dokumentation des neuen Standards (Language Reference

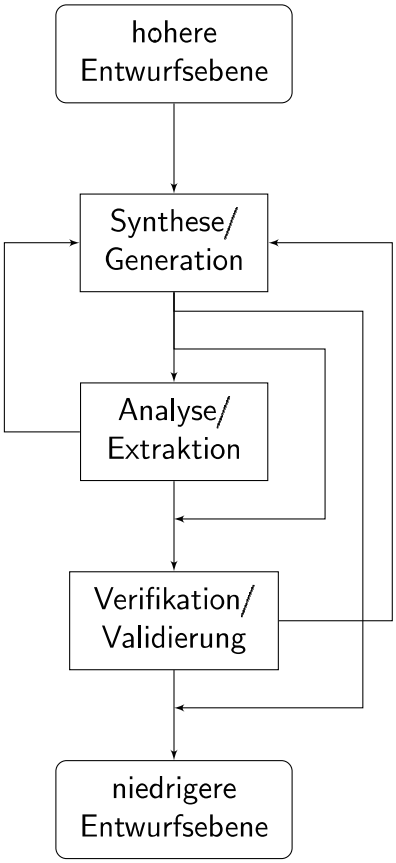


Abbildung 10.2: Prinzipieller Verlauf eines Entwurfsschrittes

Manual, LRM), wurde 1994 durch das IEEE herausgegeben neue Version enthält im wesentlichen kosmetische Änderungen (Systematisierung der Syntax), z.B. Einführung eines XNOR-Operators Seit Anfang 90er Jahre weltweiter Durchbruch für VHDL Heutige Konkurrenz von VHDL besteht vor allem in Verilog (USA) und UDL/I (Japan)

10.4 Aufbau einer VHDL-Beschreibung

10.4.1 Entity-Relationship-Modell

Datenmodelle zur Hardwarebeschreibung
Es gibt zwei unterschiedliche Datenmodelle. Zum einen das herkömmliche, strukturelle Datenmodell. In diesem gibt es hierarchische, netzartige und relationale Datenmodelle. Diese weisen je nach Anwendbarkeit als Hauptnachteil die ungenügende Modellierbarkeit komplexer Objekte auf. Semantischen Datenmodelle können zwar mächtige Datenstrukturen unterstützen. Allerdings lassen sich die Operationen oft nur unzulänglich definieren.

Allgemeines Grundkonzept des ER-Modells(semantisches Datenmodell
Man kann in diesem Modelltyp in zwei unterschiedliche Datentypen unterscheiden. Zum einen die Entitytypen (Objekttypen) mit denen man einzelne Objekte der realen Welt nachbildet. Diese sind im folgenden als Kästchen dargestellt. Zum anderen gibt es die Relationships, welcher in der Abbildung als Raute dargestellt ist. Diese beschreiben die Relation zwischen mehreren Entitäten.

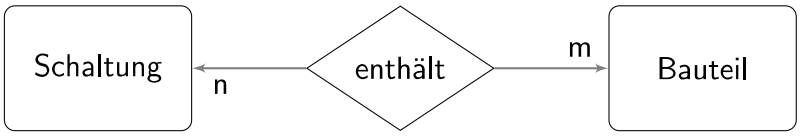


Abbildung 10.3: Einfaches ER-Diagramm.

Ein Beziehungstyp kann im ER-Modell über belieb vielen Entitäten definiert sein. Beim IC-Entwurf hat sich die Verwendung von Zellen als Grundstrukturmuster als zweckmäßig erwiesen. Bei der Beschreibung von Zellen unterschiedet man zwischen der Schnittstelle und dem Inhalt einer Zelle, die jeweils selbst wieder sehr komplexe Gebilde sein können.

10.4.2 Übersicht

Die wichtigsten Bestandteile eines VHDL-Modells ist die Instanz mit der Schnittstellenbeschreibung(entity), eine oder mehrere Architekturen(architecture), eine oder mehrere Konfigurationen(configuration), vordefinierte Funktionen, Prozeduren, Komponenten und Konstanten(package) und vordefinierte Bibliotheken(libraries).

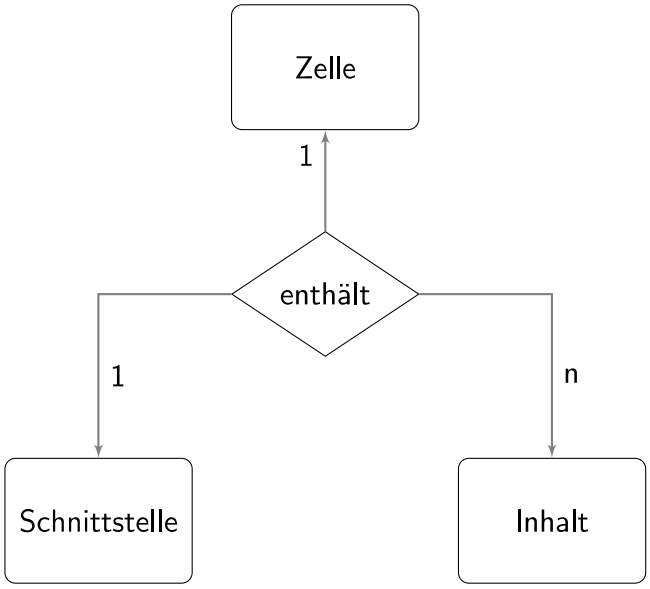


Abbildung 10.4: VHDL ER-Diagramm.

10.4.3 Schnittstellenbeschreibung

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

```
entity entity_name is
      [generic (generic_list)] [port (port_list);]
      entity_declarative_part
[begin
      passive_concurrent_statement]
end [entity] [entity_name] ;
```

Beispiel 92. *Schnittstellenbeschreibung (entity) eines OR-Gatters*

```
entity or2 is
      port (in1,in2: in bit; out1: out bit);
      -- definiere Pins als Signale vom Typ bit
end or2;
```

10.4.4 Architektur

Die Architektur - auch architecture genannt - ist eine Verhaltensbeschreibung oder kann einen strukturellen Charakter besitzen. Man kann mehrere dieser Funktionsbeschreibungen kombinieren. Somit kann man für eine Entity mehrere Architekturen deklarieren.

Dadurch können für eine Komponentenschnittstelle mehrere Beschreibungen auf unterschiedlichen Abstraktionsebenen oder verschiedene Entwurfsalternativen bestehen.

Syntax

```
architecture architecture_name of entity_name is
    architecture_declarative_part
begin
    all_concurrent_statements
end [architecture] [architecture_name];
```

Beispiel 93. *Architektur (architecture) eines OR-Gatters*

```
architecture or2_behavioral of or2 is
begin
    out1 <= in1 or in2; -- Verhaltensbeschreibung
end or2_behavioral;
```

10.4.5 Konfiguration

Eine Konfiguration(configuration) beschreibt welche Zuordnung für die möglicherweise verwendeten Submodule in der Architecture einer bestimmten Entity gelten sollen. Es können auch hierarchisch den untergeordneten Entities bestimmte Architekturen oder Parameterwerte zugeordnet werden.

Syntax

```
configuration configuration_name of entity_name is
    { use_clause. attribute_specification }
    { block-configuration }

    for component_specification
        [ use binding_indication ; ]
        [ block_configuration ]
    end for ;

end [configuration_name] ;
```

Beispiel 94. *Architektur (architecture) eines OR-Gatters*

```
configuration or2_config of or2 is
    for gate_level -- verknuepfe architecture gate_level
    end for;       -- mit entity or2
end or2_config;
```

10.4.6 Packages und Libraries

Packages fassen Typen oft gebrauchter Funktionen, Prozeduren, Komponenten oder Konstanten zusammen. Dazu zählen auch Anweisungen, wie Typ- oder Objektdeklarationen und Beschreibungen von Prozeduren und Funktionen, die in mehreren VHDL-Beschreibungen gebraucht werden. Es gibt vordefinierte Packages wie „standard“, welche den zweiwertigen Logiktyp „bit“ und häufig verwendete Funktionen und Typen beinhaltet.

Syntax

```
0 package name_of_the_package is
1       package_declarative_part
2 end [ package ] [ name_of_the_package ] ;
```

Bibliotheksobjekte werden in einer Library zusammengefasst. Die Bekanntgabe einer oder mehrerer dieser Bibiliotheken erfolgt durch das library-Tag. Dabei sind Bibliotheken wie std und work in VHDL allgemein bekannt.

Syntax

```
0 library library_name_1 {, library_name_i }; --Aufruf der Bibliothek
1
2 use library_name.all;
```

10.5 Entwurfssichten in VHDL

10.5.1 Verhaltensmodellierung

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

Beispiel 95. *Verhaltensmodellierung eines Komparators*
Dieser Liefert eine boolsche „1“ wenn a größer ist als b. Außerdem lassen sich mit dem n beliebig breite Vektoren vergleichen.

```
0 entity compare is
1       generic (n : positive := 4);
2       port (a, b : in bit_vector (n-1 downto 0); result: out bit);
3 end compare;
4
5 architecture behavioral_1 of compare is
6 begin
7       process (a, b)
8       begin
9             if a > b then -- liefert 1 falls a > b
10                  result <= '1';
```

```
11            else
12                  result <= '0';
13            end if;
14      end process;
15 end behavioral_1;
```

Die beiden prinzipiellen Beschreibungsmittel in der Verhaltenssicht sind sequentielle(sequential) oder nebenläufige Anweisungen (concurrent statements). Bei den sequentiellen bzw. prozeduralen Beschreibungen werden Konstrukte wie Verzweigungen(if-then-else), Schleifen(loop) oder Unterprogrammaufrufe(function, procedure) verwendet.

Beispiel 96. *Modell eines Halbaddierers mit Schnittstelle und Architektur*
Anm.: Die Ergebnisse werden erst beim Verlassen des Prozesses sichtbar.

```
0 entity halfadder is
1       port (a, b : in bit; sum, carry : out bit);
2 end halfadder;
3
4 architecture behavioral_seq of halfadder is
5 begin
6       process (a, b)
7       begin
8             if(a = '1' and b = '1') then sum <= '0'; carry <= '1';
9             else
10                  if (a = '1' or b = '1') then sum <= '1'; carry <= '0';
11                  else sum <= '0'; carry <= '0';
12                  end if;
13            end if;
14      end process;
15 end behavioral_seq;
```

Bei den nebenläufigen Anweisungen ist es erlaubt parallel ablaufende Operationen zu beschreiben. Hiermit kann man spezifischer Eigenschaften von Hardware bzw. parallel arbeitenden Funktionseinheiten beschreiben.

Beispiel 97. *Architektur des Halbaddierers mit nebenläufigen Anweisungen*
Anm.: Die beiden Verknüpfungen XOR und AND können dabei gleichzeitig aktiv sein.

```
0 architecture behavioral_par of halfadder is
1 begin
2       sum <= a xor b;
3       carry <= a and b;
4 end behavioral_par;
```

10.5.2 strukturelle Modellierung

Das Wesen der strukturellen Modellierung besteht aus der Darstellung des inneren Aufbaus aus Unterkomponenten. Die Eigenschaften derUnterkomponenten werden in unabhängigen VHDL-Modellen beschrieben. Diese stehen kompiliert in Modellbibliotheken zur Verfügung. Eine eindeutige Zuordnung eines VHDL-Modells in eine der beiden Modellierungsarten ist oft schwierig. Daher ist es in VHDL erlaubt beide Beschreibungsarten innerhalb eines Modells zu benutzen.

Beispiel 98. *Halbaddierer, der aus einem XOR2- und einem AND2-Gatter aufgebaut ist*

```
0  architecture structural of halfadder is
1       component xor2
2            port (c1, c2 : in bit; c3 : out bit);
3       end component;
4
5       component and2
6            port (c4, c5 : in bit; c6 : out bit);
7       end component;
8
9  begin
10      xor_instance : xor2 port map (a, b, sum);
11      and_instance : and2 port map (a, b, carry);
12 end structural;
```

10.6 Entwurfsebenen in VHDL

10.6.1 Algorithmusebene

Die Schaltung im Beispiel soll immer dann, wenn sie von einem Controller eine Aufforderung erhält, eine Adresse aus einem internen Register nach frühestens 10ns auf den Bus legen. Dabei enthält die Beschreibung keine Angaben über die spätere Schaltungsstruktur und Takt- oder Rücksetzsignale.

Beispiel 99. *Beschreibung eines Schnittstellenbausteins auf Algorithmusebene*

```
0  architecture algorithmic_level of io_ctrl is
1  begin
2       ...
3       write_data_alg: process
4
5       begin
6            wait until adr_request = '1';
```

```
7             wait for 10 ns;
8             bus_adr <= int_adr;
9       end process write_data_alg;
10 end algorithmic_level;
```

10.6.2 Register-Transfer-Ebene

Im Unterschied zur Algorithmusebene wird in der Register-Transfer-Ebene ein zeitliches Ablaufschema der Operation vorgegeben und implizit eine Schaltungsstruktur beschrieben. Im Beispiel wird ein Taktsignal (clk) hinzugefügt und die Operation in Abhängigkeit dieses Signals beschrieben.

Beispiel 100. *Beschreibung eines Schnittstellenbausteins auf Algorithmusebene*

```
0 architecture rt_level of io_ctrl is
1 begin
2       ...
3       write_data_rtl : process (clk)
4             variable tmp : boolean;
5
6       begin
7             if (clk'event) and (clk = '1') then
8                   if((adr_request = '1') and (tmp = false))then tmp := true;
9                   elsif (tmp = true) then bus_adr <= int_adr; tmp := false;
10                  end if;
11            end if;
12      end process write_data_rtl;
13 end rt_level;
```

Wird bei aktiver Taktflanke ein gesetztes adr-request-Signal entdeckt, wird die temporäre Variable tmp gesetzt, damit bei nächster aktiver Taktflanke (Wartezeit!) die Adresse auf den Bus geschrieben werden kann. Eine geeignete Wahl der Taktperiode stellt die Wartezeit von mindestens 10ns sicher.

10.6.3 Logik(Gatter)ebene

Elektronische Systeme werden durch logische Verknüpfungen digitaler Signale und deren zeitlichen Eigenschaften (Verzögerungszeiten der Verknüpfungen) beschrieben. VHDL besitzt dazu vordefinierte Operatoren wie AND, OR, XOR, NOT, etc. für binäre Signale ('0', '1') und gestattet die Ergänzung durch benutzerdefinierte Operatoren. Die Konstrukte zur Modellierung zeitlicher Eigenschaften werden bereitgestellt.

Beispiel 101. *Halbaddierer-Architektur auf Logikebene in Verhaltenssichtweise*

```
0 architecture logic_1eve1 of halfadder is
1 begin
2       sum <= a xor b after 3 ns;
3       carry <= a and b after 2 ns;
4 end logic_level;
```

Die strukturelle Darstellung entspricht vorherigem Beispiel der Architektur structural. Die Verzögerungszeiten ergeben sich hier aus den internen Verzögerungszeiten der Subkomponenten.

10.7 Logiktypen

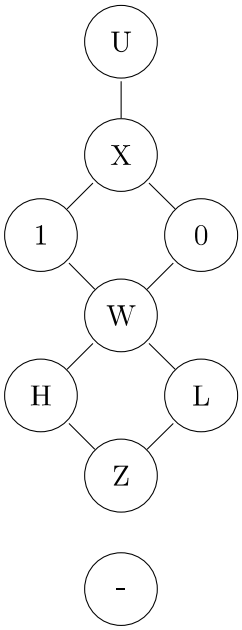
9-wertiges Logikwertesystem von VHDL `ieee.std_logic_1164` MVL9

Das Logikwertesystem ist nicht als Teil der Sprachdefinition definiert und kann an jeweilige Problemstellungen angepasst werden. Der Datenaustausch der VHDL-Modelle und der Einsatz von Synthesewerkzeugen erfordern aber eine Standardisierung. Für eine hardwarenahe Simulation wird häufig mit den 9 Zustand/Stärkewerten von MVL9 modelliert. Es beinhaltet eine Berücksichtigung der Signaltreiberstärke. Dies bedeutet, dass bei der Spannungsversorgung , Eingänge oder aktive Gatterausgänge stärkere Werte schwächere Stärken überschreiben. Dabei kann man folgende Festlegung der Signalwerte treffen. Es gibt zum einen die drei Signaltreiberstärken „stark“, „schwach“ und „hochohmig“. Dazu kommen die Logikpegel „0“, „1“ und „unbestimmt“. Darüber hinaus gibt es die zusätzlichen Werte „nicht initialisiert“ für die Simulation und ein „don’t-care“ für Syntheseanwendungen.

Die Ausgänge der Gatter können mit einem Bus verbunden werden. Die Auflösung der zusammengeschalteten Signale an einem Bus erfolgt nach folgender Grafik und von links nach rechts in aufsteigender Reihenfolge.

Wert	deutsche Beschreibung	englische Beschreibung
U	nicht Initialisiert	uninitialized
X	stark unbestimmt	strong unknown
0	starker „0“-Pegel	strong low
1	starker „1“-Pegel	string high
Z	Hochohmig	tri-state
W	schwach unbestimmt	weak unknown
L	schwacher „0“-Pegel	weak low
H	schwacher „1“-Pegel	weak high
-	don’t-care	

(a) Übersicht über das Logikwertesystem



(b) Hasse Diagramm

Abbildung 10.5: 9-wertige Logik

VHDL-Konstrukte zur Laufzeitmodellierung

Bei der Laufzeitmodellierung in VHDL muss man verschiedene Delays mit einbauen damit die Simulation so realitätsnah wird wie irgend möglich. Das erste dieser Delays ist das „Transport-Delay“, bei dem jeder Impuls unabhängig von seiner Dauer verzögert am Ausgang erscheint. Das „Inertial-Delay“ hingegen gibt nur Impulse weiter deren Dauer größer als die angegebene Verzögerungszeit des angesteuerten Gatters sind. Es findet hier sowohl eine Impulsunterdrückung als auch eine Zeitverzögerung von gleicher länge statt. Im Gegensatz dazu wird bei dem „Reject-Inertial-Delay“ der Zeitbereich für die Impulsunterdrückung und die Zeitverzögerung separat angegeben.

```
0 entity nand2 is
1       port(a,b: in std_logic; o: out std_logic);
2 end nand2;
3
4 architecture several_delays of nand2 is
5 begin
6       --einfache Verzoegerung
7       o <= transport a nand b after 2 ns;
8
9       --Verzoegerung mit Impulsmindestloenge
10      o <= [inertial] a nand b after 2 ns;
11
12      --Verzoegerung mit seperater Impulsmindestlaenge
13      o <= reject 1 ns inertial a nand b after 2 ns;
14 end several_delays;
```

10.8 Testumgebung

Die VHDL-Simulation erfolgt in drei Phasen. Die Elaboration ist der Aufbau der Schaltung aus den compilierten VHDL-Modellen. Die Initialisierung beinhaltet die Signale, Variablen und Konstanten die Anfangswerte erhalten. Diese Werte ergeben sich aus expliziter Angabe durch die Typ-Spezifikation. Jeder Prozess wird einmal gestartet. Die Ausführung ist die Simulation bis zum Ende der spezifizierten Simulationsdauer. Zur Stimuli-Generierung und zur Ergebnisauswertung wird das zu testende Modell in eine Testbench eingebunden. Das Modell wird instanziiert und mit den Ein- und Ausgangssignalen verdrahtet.