Systembeschreibung und Entwurf
VHDL und Verilog

Thomas Schumann und Bernhard Hoppe

M.Sc.-Studiengang Elektrotechnik

Hochschule Darmstadt

Fachbereich Elektrotechnik und Informationstechnik

09.08.2016

Inhaltsverzeichnis

Einleitung

In diesem Lehrbrief werden Hardwarebeschreibungsprachen (HDLs, *Hardware Description Languages*) behandelt. Das sind Werkzeuge für den Digitalentwurf, die sich mittlerweile in Richtung gemischt analog-digitaler Systeme weiterentwickelt haben. Der wichtigste Vertreter dieser Sprachen in Europa ist VHDL und die weltweit am weitesten verbreitete HDL ist Verilog. Ob nun VHDL oder Verilog besser geeignet ist, um Designaufgaben zu lösen, ist Ansichtssache. Beide Sprachen haben Stärken und Schwächen. Verilog lässt sich leichter erlernen, ist an der Sprache C orientiert und hat nicht den komplexen Aufbau von VHDL. VHDL ist hingegen systematischer und kann auch für abstraktere Modellierungsaufgaben eingesetzt werden, bei denen Verilog als mehr hardwareorientierte Sprache weniger gut geeignet ist. Es gibt aber keine Designaufgabe, die nur mit einer der beiden HDLs bearbeitet werden könnte. Da beim Hardwareentwurf mit FPGAs in der Praxis vielfach vorgefertigte HDL-Modelle verwendet werden (*IP-Cores*), die in Verilog oder auch in VHDL geschrieben sein können, ist es wichtig beide HDLs zu kennen. Die Verifikation von digitalen Designs kann mit Verilog wesentlich effizienter durchgeführt werden als mit VHDL. Die Verilog-Erweiterung SystemVerilog ist heute Industriestandard zur Schaltungsvalidierung. Interessanterweise sind zahlreiche VHDL-Konstrukte bei der Definition von SystemVerilog übernommen worden, wie etwa die Aufzählungsdatentypen oder Records, also Datenfelder, die sich aus verschiedenen Datentypen zusammensetzen.

In diesem Lehrbrief werden VHDL und Verilog für die digitale Modellierung mit Verhaltensbeschreibung und Verifikation vergleichend vorgestellt. Wir beginnen mit VHDL, erarbeiten die Grundkonzepte, die HDLs von gewöhnlichen Programmiersprachen wie C unterscheiden und die die Simulation von Modellen für elektronische Schaltungsstrukturen erst möglich machen. Dann entwickeln und verifizieren wir Modelle von einfachen logischen Grundschaltungen bis hin zu Zustandsautomaten. Im zweiten Teil folgt die Einführung in Verilog. Gemeinsamkeiten und Unterschiede werden herausgearbeitet und anschließen werden die gleichen Modelle in Verilog kodiert, simuliert und verifiziert, wie im VHDL-Teil.

Wir vermitteln also das Grundwissen über die Hardwarebeschreibungssprachen VHDL und Verilog und fassen die grundlegenden Verfahren der aktuellen Designmethodik für digitale (integrierte) Schaltungen zusammen.

HDLs werden also primär beim Entwurf digitaler elektronischer Systeme eingesetzt. Ihr Einsatz erfolgt zielgerichtet. Ein funktional korrekter digitaler Chip soll mit Hilfe von Sprache in möglichst kurzer Zeit entworfen werden. Der Schaltungsentwurf reduziert sich heute auf die Erstellung eines digitalen Modells auf Register-Transfer-Ebene (RTL-Modell). Ein solches Modell beschreibt die gewünschte Funktion mit einer Kette von Boolschen Schaltwerken, die ihre jeweiligen Ergebnisse in Registern speichern, damit sie vom nächsten Schaltwerk in der Kette weiterverarbeitet werden können.

Beim HDL-basierten Design steht die Verwendung der Sprache bei der praktischen Modellbildung mehr im Vordergrund als die rein syntaktischen Aspekte. Sowohl VHDL, wie auch Verilog, bieten vielfältigste Möglichkeiten, die aber weder zum Einstieg in das HDL-basierte Schaltungsdesign noch zur erfolgreichen Realisierung von digitalen Funktionen mit ASICs oder FPGAs sofort gebraucht werden. Die nicht angesprochenen Funktionalitäten können, wenn das grundlegende Verständnis durch dieses Teilmodul vermittelt worden ist, bei Bedarf mit den empfohlenen Fachbüchern in der Literaturliste oder aus den Language Reference Manuals der IEEE-Normen für die beiden Sprachen nachgelernt werden.

Ein funktionierendes HDL-Modell wird aber erst zur Schaltung, wenn es als Schaltungsbeschreibung in ein Computerwerkzeug zur Logiksynthese eingegeben wird. Ein Synthesetool analysiert das HDL-Modell, das das Schaltungsverhalten nur abstrakt beschreibt, setzt das Modell in eine synchrone boolesche Beschreibung um, optimiert die logische Struktur und bildet das Ergebnis auf die Komponenten einer ASIC-oder FPGA-Bibliothek (logische Gatter, Flip-Flops, etc.) in der Zieltechnologie ab. Das Resultat ist eine Implementierung der gewünschten Funktion als Schaltung aus Bibliothekszellen. Diese Schaltung kann als HDL-Netzliste oder in anderen Formaten ausgegeben werden und dient als Grundlage für den physikalischen Entwurf, bei dem aus der Schaltung eine Strukturdarstellung abgeleitet wird, die sich auf einen programmierbaren Logikbaustein übertragen lässt oder als ASIC-Layout an den Halbleiterhersteller gegeben werden kann.

Die synthesebedingten Rationalisierungseffekte bei der Schaltungsdefinition haben den Schwerpunkt der Designaktivitäten verschoben. Der Designer beschäftigt sich heute fast nur noch mit der Frage, was implementiert werden soll und überlässt die Umsetzung in eine technologienahe Gatterbeschreibung den Computer-Tools. Statt elegante und effiziente Schaltungsvarianten zu entwickeln, ist er primär damit befasst, ein synthesefähiges HDL-Modell zu erstellen und dessen Übereinstimmung auf Verhaltensebene und später auf Gatterniveau mit den Vorgaben der System-Spezifikation (Lastenheft) zu verifizieren. Verifikation bleibt auch der Schwerpunkt im gesamten weiteren Entwurfsablauf: Nach der Synthese und der Umsetzung in die Zieltechnologie folgen zusätzliche Verifikationsschritte, die dann auch Laufzeiten durch die eingesetzten Gatter berücksichtigen. Auch bei der Verifikation bieten HDL-Modelle Vorteile. Statt die Testsignale per Hand in den Simulator einzugeben und die Ergebnisse als Wellenzüge mit bloßem Auge zu analysieren, kann man so genannte *Testbench*-Modelle mit HDLs entwickeln, die diese Aufgaben automatisch erledigen und selbständig die Verifikationsaufgaben abarbeiten.

Dieser Lehrbrief ist wie folgt gegliedert: In den ersten Kapiteln wird die Modellbildung, Synthese und Verifikation von digitaler Elektronik mit Hilfe der Hardwarebeschreibungssprache VHDL behandelt. Die wichtigsten Konstrukte der Sprache zur Verhaltensbe-

schreibung werden vorgestellt und zur Modellierung von kombinatorischer Logik, Zustandsautomaten und speichernden Elemente verwendet. Dann folgt der Verilog-Teil, in dem die entsprechenden Sprachkonstrukte vergleichend mit den VHDL-Varianten vorgestellt werden. Anschließend diskutieren wir die Verilog-Modelle der gleichen digitalen Funktionsblöcke wie im VHDL-Teil.

Zur Verifikation der Modelle werden wiederum HDL-Modelle verwendet, die als Testbenches (Prüfstände) bezeichnet werden. In die Testbench wird das neu entwickelte Modell eingesetzt und dann mit Testmustern angesteuert, die von geeigneten HDL-Konstrukten erzeugt werden. Die sich ergebenden Ausgangssignale werden entweder direkt bewertet oder zur weiteren Analyse in Dateien abgespeichert.

Eine neue Hardwarebeschreibungssprache erlernt man wie jede Sprache nur durch fortgesetztes Üben. Digitale HDL-Modelle können mit dem Simulator ModelSim® der Firma Mentor Graphics simuliert werden. Mentor Graphics als führender EDA-Hersteller (EDA, Electronic Design Automation) bietet den Simulator in der Version PE Student Edition als Download im Internet an. Lizenzen (Educational Licenses) werden nach der Registrierung per Email erteilt. Der Umgang mit diesem Simulator wird in einem Anhang erklärt.

Lernziele

Das Ziel dieser Kurseinheit ist es, in den modernen digitalen Hardwareentwurf mit den Hardwarebeschreibungssprachen VHDL und Verilog einzuführen. Grundlegende Konstrukte der beiden Sprachen werden vor- und gegenübergestellt. Hardwarebeschreibungssprachen (HDLs) sind keine Sprachen zur Entwicklung von Software, sondern eine Hardwaremodellierungssprachen, die die Nebenläufigkeit der Informationsverarbeitung durch elektronische Schaltungen abbilden können: alle definierten Prozesse innerhalb eines Modells werden gleichzeitig ausgeführt, nur der Code innerhalb eines Prozesses wird sequentiell abgearbeitet. Simulationsmodelle von Schaltungen verhalten sich daher grundlegend anders als rein sequentieller Code, der z.B. in C geschrieben wurde.

Nach dem Studium dieser Kurseinheit sollten Sie

- die Grundstruktur und den elementaren Sprachumfang von VHDL und Verilog kennen;
- über Unterschiede, sowie Stärken und Schwächen der beiden Sprachen Bescheid wissen;
- wissen, wie die Entwicklung digitaler Schaltungen über HDL-Codierung und Logiksynthese sich gestaltet;
- in der Lage sein, digitale Funktionen zu definieren, zu modellieren und die Modelle mit Testbench-Konzepten verifizieren zu können;
- sich mit dem generischen Aufbau von HDL-Modellen für kombinatorische Logikblöcke, speichernde Elemente sowie Zustandsautomaten auskennen;
- Ihr Wissen über die beiden HDLs durch Lektüre einschlägiger Literatur und Normen problemlos erweitern können;
- mit dem HDL-Simulator ModelSim® umgehen können.

1 Spezifikation eines digitalen Systems mit HDLs

Die Entwicklung von digitalen Systemen wird durch folgende Trends bestimmt:

- starker Anstieg der Entwurfskomplexität
- reduzierte Entwicklungszeiten (time-to-market)

VHDL und Verilog sind Hardwarebeschreibungssprachen, die für zeitdiskrete digitale Modelle der Elektronik entwickelt wurden. Sie finden heute hauptsächlich bei den komplexesten Entwicklungen von digitalen Systemen, den integrierten Schaltungen, Anwendung. Der Siegeszug der Mikroelektronik ist hauptsächlich durch die exponentielle Reduktion der Kosten pro Funktion begründet, so dass heute nicht nur in allen elektronischen Consumer-Produkten Mikrochips zu finden sind, sondern auch in so genannten eingebetteten Systemen wie ABS im Automobil oder Steuerungselektronik in Werkzeugmaschinen und Robotern.

Im Rahmen dieses Lehrbriefs soll darauf verzichtet werden, jedes Konstrukt von VHDL oder Verilog zu erklären, vielmehr soll gezeigt werden, wie effizient sich mit einer Hardwarebeschreibungssprache ein komplexes digitales System modellieren, simulieren und schließlich zu einer integrierten Schaltungen synthetisieren lässt.

1.1 Von der Spezifikation zum Chip

Eine Hardwarebeschreibung in HDL wird durch CAE-Werkzeuge hinsichtlich der Schaltungssimulation sowie der Schaltungssynthese (für digitale Schaltungen) unterstützt. Die Schaltungssimulation dient zur Verifikation der Entwurfsidee. Unter Schaltungssynthese versteht man die automatische Generierung von Logikgatternetzlisten (Logik-Synthese) und die anschließende automatische Implementierung der Schaltung auf Hardwareplattformen wie ASICs oder FPGAs (Physikalische Synthese). HDLs bieten für digitale Schaltungen eine einheitliche Modellierung für Simulation und Synthese.

Eine HDL-Verhaltensbeschreibung modelliert das Eingangs-/Ausgangsverhalten von digitalen Hardwarekomponenten, im so genannten RTL (Register Transfer Level)-Format. Modelle auf RTL-Niveau sind synthesefähig. Ein RTL-Modell entsteht durch Zerlegung einer Schaltung in kombinatorische und speichernde Komponenten. RTL-Modelle für Grundschaltungen zu entwickeln wird Hauptgegenstand dieses Lehrbriefs sein. Zur Beschreibung der RTL-Modelle dürfen nur synthesefähige Konstrukte der HDL-Hardwarebeschreibungssprache verwendet werden. Kommerzielle Synthesewerkzeuge können dann diese Modellbeschreibung auf die Elemente einer ASIC-Gatterbibliothek bzw. auf eine FPGA-Architektur abbilden.

Die Platzierung und Verdrahtung der Gatter wird von einem weiteren EDA-Werkzeug ausgeführt (Physikalische Synthese), so dass schließlich beim ASIC-Entwurf die Maskeninformation für den Halbleiterhersteller bereitsteht bzw. die Funktion des FPGA-Chips programmiert werden kann. ASIC steht dabei für Application Specific Integrated Circuit und bedeutet, dass ein IC nach kundenspezifischen Wünschen gefertigt wird. Hingegen ist ein FPGA (Field Programmable Gate Array) ein Standard-IC, der erst durch Programmierung von Speicherzellen seine Funktion erhält.

1.2 Der VHDL/Verilog Standard

Eine HDL ist keine Sprache zur Entwicklung von Software, sondern eine Hardwaremodellierungssprache. Sie verfügt deshalb im Unterschied zu bekannten Programmiersprachen über das Konzept der Nebenläufigkeit: alle definierten Prozesse innerhalb eines Modells werden gleichzeitig ausgeführt, nur der Code innerhalb eines Prozesses wird sequentiell abgearbeitet.

VHDL wurde ursprünglich im Rahmen eines Projekts des amerikanischen Verteidigungsministeriums entwickelt und zwar zur Dokumentation von Struktur und Funktion digitaler Schaltungen. VHDL wurde 1987 durch IEEE als Norm 1076-1987 verabschiedet und 1993, weiterentwickelt, als IEEE 1076-1993 zum Standard [IEEE Standard 1076]. Die letzte syntaktische Erweiterung des Standards erfolgte 2008 unter der Bezeichnung IEEE Standard 1076-2008.

Verilog war die die erste HDL, die sich in der Elektronikindustrie durchgesetzt hat. VHDL kam später, war aber von Anfang an eine genormte (IEEE 1076), frei zugängliche Sprache. Verilog hingegen wurde von der Firma Gateway (heute Cadence) als proprietäres Werkzeug vermarktet und ist erst 1997 und 2001 in revidierter Form genormt (IEEE-1364) und frei verfügbar und wird von der Organisation Open Verilog International (OVI, auch bekannt als Accellera) verwaltet. 2001 gab es eine Erweiterung des Standards unter der Bezeichnung IEEE Standard 1364–2001 (Verilog 2001).

2 Modellierung kombinatorischer Logik mit VHDL

VHDL-Modelle kombinatorischer Logik werden in diesem Kapitel vorgestellt. Dabei geht es zunächst um den grundsätzlichen syntaktischen Aufbau eines VHDL-Modells. Es wird gezeigt, wie logische und arithmetische Operatoren sowie verschiedene Datentypen von Signalen zur Modellierung von Grundschaltungen kombinatorischer Logik eingesetzt werden können.

2.1 Aufbau eines VHDL Modells

Digitale Modelle beschreiben Systeme, in denen sich die Signale in diskreten Stufen zu diskreten Zeitpunkten ändern. *Signale* können innerhalb des VHDL Modells auf verschiedenen Hierarchieebenen definiert sein.

Die grammatischen Regeln für VHDL haben einen Namen (auf der linken Seite von ::= *kursiv*) und werden als Funktion anderer Regeln oder mit Marken (Endzeichen) ausgedrückt. Folgende Konventionen sind üblich:

`modul`	Schlüsselwörter Courier New
::=	Zuweisung einer Regel zu ihrem Namen
[xx]	Der Teil xx ist optional
x|y	wähle x oder y
<…>	Eingabe von Bezeichnern oder Ausdrücken erforderlich

VHDL Modelle setzten sich aus mindestens zwei Teilen zusammen: Einer `Entity` und einer `Architecture`. Die Entity stellt die Schnittstelle des Modells zur Außenwelt (z.B. Pins eines IC) dar. Zu jeder Entity muss es mindestens eine Architecture geben. Die Architecture beschreibt die Funktionalität oder das Verhalten der Entity. Signale können auf der höchsten Hierarchieebene innerhalb der Entity als Port-Deklaration auftreten, oder als lokale Signale in der Architecture definiert werden. Eine Entity-Deklaration hat die folgende Form:

Entity-Deklaration::=
`entity` <entity_name> `is`
`[port(` signal_name1: port_mode port_type;
signal_name2: port_mode port_type := initial_value);]
`end entity` <entity_name>;

port_mode::= `in`|`out`|`inout`

Signale innerhalb der Port-Deklaration besitzen einen *port_mode* zur Spezifizierung der Datenflussrichtung. Verwendung finden häufig die Modi: `in, out, inout`. Der Datentyp muss ebenfalls bei der Definition von Signalen angegeben werden. Selbsterklärend sind die Datentypen: `bit` und `bit_vector`.

Die Initialisierung eines Signals als Port erfolgt mit dem Zuweisungsoperator „:=“. Alle späteren Signal-Signal- sowie Signal-Port-Zuweisungen erfolgen mit dem Signalzuweisungsoperator „<=“.

Innerhalb der Architecture wird das Verhalten des Modells durch eine oder mehrere nebenläufige Anweisungen beschrieben. Eine Architecture-Deklaration hat die folgende Form:

Architecture Deklaration::=
`architecture` <architecture_name> `of` <entity_name> `is`
-- interne Objektdeklarationen
`begin`
-- nebenläufige Anweisungen
`end architecture` <architecture_name>;

Zunächst aber noch einige Anmerkungen zur VHDL-Syntax:

- Kommentare beginnen mit ´--´;
- Leerzeichen werden, wenn nicht von der Sprachdefinition explizit gefordert, im Quelltext ignoriert;
- Einschübe werden ebenfalls ignoriert, erhöhen aber die Lesbarkeit;
- Es wird **nicht** unterschieden zwischen Groß- und Kleinschreibung;
- Bezeichner (Namen von Signalen, Architectures, Entities) dürfen keine reservierten Wörter sein;
- Bezeichner beginnen immer mit einem Buchstaben. Die nachfolgenden Zeichen können Buchstaben, Ziffern oder aber auch Unterstriche sein;
- Es dürfen keine zwei oder mehr Unterstriche in einem Bezeichner aufeinanderfolgen;

Als erste Anwendung von VHDL wird das Modell eines Multiplexers besprochen. Ein 4-zu-1 Multiplexer wählt mit kombinatorischer Logik nach Maßgabe eines Steuersignals `S` ein Eingangssignal `E` aus vier Möglichkeiten aus und übergibt das gewählte Signal an einen Ausgangsport `Y`.

Der folgende Code ist die VHDL-Codierung des beschriebenen Verhaltens:

```
-- multiplexer.vhd
---------------------------
entity MUX4 is
   port(    S: in bit_vector(1 downto 0);
            E: in bit_vector(3 downto 0);
```

```
            Y: out bit);
end MUX4;

architecture Behaviour of MUX4 is
begin
   with S select
   Y <=     E(0) when "00",
            E(1) when "01",
            E(2) when "10",
            E(3) when "11";
end architecture Behaviour;
```

In der Entity Deklaration wird ein Eingangssignal E als 4bit-Bus deklariert. Die Bezeichnung (`x downto y`) bedeutet eine absteigende Wertigkeit der einzelnen Bits des `bit_vector` Signals. Das höchstwertigste Bit steht am weitesten links in der Bitfolge. Dies entspricht der üblichen Darstellung in der Digitaltechnik.

Die logische Beschreibung des Modellverhaltens erfolgt innerhalb der Architecture-Deklaration. Die Zuweisung eines Eingangssignals auf den Ausgang erfolgt durch eine *selektive Signalzuweisung*: Ein Selektor-Ausdruck muss auf alle möglichen Werte abgefragt werden. Werte können zusammengefasst werden durch den Ausdruck `others`. Die allgemeine Syntax lautet:

selektive Signalzuweisung::=
`with` <Selector> `select`
<Ausgangssignal> <= <logischer Ausdruck_1> `when`
<Selectorwert_1>,
<logischer Ausdruck_2> `when`
<Selectorwert_2>,
<logischer Ausdruck_N> `when`
`others`;

Kennzeichnend für die selektive Signalzuweisung ist, dass der aktuelle Wert des Selektorsignals festlegt, welche Anweisung ausgeführt wird. Soll auf unterschiedliche Bedingungen getestet werden (z.B. auf den Wert verschiedener Eingangssignale), so bietet sich die *bedingte Signalzuweisung* an. Die Bedingungen müssen sich dabei gegenseitig ausschließen. Die allgemeine Syntax lautet:

selektive Signalzuweisung::=
<Ausgangssignal> **<=** <logischer Ausdruck_1> `when`
<Bedingung_1> `else`
<logischer Ausdruck_2> `when`
<Bedingung_2> `else`
<logischer Ausdruck_N>;