Deel 1: 'Entity'- en 'Architecture'-beschrijvingen

De Kracht van VHDL

De fundamenten waarop de VHSIC Hardware Description Language (VHDL) berust zijn klein in aantal en simpel. Ten eerste bestaan digitale schakelingen uit componenten die met elkaar verbonden zijn door ingangen en uitgangen. Ten tweede verandert de toestand van een component als een ingangswaarde zich wijzigt en ten derde bestaat het gedrag uit parallelle samenwerkende processen. Tenslotte bestaan componenten mogelijkerwijs zelf uit een hiërarchie van componenten. VHDL is zowel een beschrijvings- als een modelleringstaal die hardware eigenlijk specificeert op systeemniveau. Dit eerste artikel van een serie van twee schetst de basiskenmerken van VHDL; het tweede zal nader ingaan op simulatie en synthese met behulp van deze taal.

Hans van Thiel, Finite Epistemics

De Very high speed integrated circuit Hardware Description Language (VHDL) is halverwege de jaren tachtig ontstaan op initiatief van het Amerikaanse ministerie van defensie. De functionaliteit van elektronica in wapensystemen was slecht gedocumenteerd en dit leidde tot problemen, met name als onderdelen die inmiddels al uit produktie waren genomen moesten worden vervangen. Die situatie was en is niet uitzonderlijk; de produktlevenscyclus van IC's is immers veel korter dan die van tanks of vliegtuigen. Door elektronicafabrikanten te verplichten een VHDL-beschrijving mee te leveren moest het gemakkelijker worden verouderde onderdelen te vervangen door gelijkwaardige, modernere varianten. VHDL is dan ook oorspronkelijk vooral bedoeld als modelleringstaal.

In 1987 werd het een IEEE-standaard (IEEE Std. 1076 -1987). Zulke normen worden in principe elke vijf jaar herzien en de nieuwe versie kwam uit in 1993. Voor dit jaar wordt VHDL 1076-2000 verwacht. De taal is vrij stabiel gebleven gedurende het afgelopen decennium maar het is niet de enige hardware beschrijvingstaal gebleven met een ruime toepassing. De grote concurrent is Verilog, dat in 1995 ook een IEEE-standaard is geworden (IEEE Std. 1364-1995). Ook voor deze norm zijn thans nieuwe voorstellen gepubliceerd waarover dit jaar een beslissing zal worden genomen.

Hoewel beide talen, zoals gezegd, concurrenten lijken te zijn is Verilog toch meer bedoeld voor realisatie van hardware op het laagste niveau en VHDL voor beschrijving en modellering op hoger niveau. Het laatste nieuws is dan ook dat VHDL International en Open Verilog International gezamenlijk een nieuwe standaard gaan ontwikkelen. Ook thans bestaan er overigens al diverse (officiële) bibliotheken en koppelingen van VHDL met andere verwante IEEE-normen (zie ref. 1 en 2).

Intussen wordt echter ook al gewerkt aan VHDL 200X dat, in tegenstelling tot de vorige versies, wel flinke veranderingen zal bevatten. Zo wordt er gedacht aan objectoriëntatie en vooral ook aan toepassing voor co-ontwerp van hardware en software. Omdat VHDL parallelle samenwerkende processen *(concurrency)* kan modelleren lijkt het een uitstekende kandidaat daarvoor.

Welke vernieuwingen de komende jaren inderdaad gerealiseerd zullen worden is op dit moment nog onzeker. Bovendien zal de beschikbaarheid van tools met eigen extensies en mogelijkheden misschien zelfs sterker bepalend zijn voor de praktijk dan de officiële standaarden. Men denke aan visueel ontwerpen met een tool als bijvoorbeeld Visual HDL van Innoveda.

Het huidige VHDL is een vrij uitgebreide taal die op het eerste gezicht veel weg heeft van een programmeertaal en dan met name ADA, dat ook een initiatief was van het Amerikaanse ministerie van defensie. VHDL is echter geen programmeertaal maar combineert een structurele specificatie van een digitaal systeem met een functionele specificatie. Een beschrijving van een schakeling kan, mits het geheel tot combinaties van bepaalde VHDL constructies is teruggebracht, dienen als basis voor een synthesetool. Omdat ook het gedrag van alle basisonderdelen wordt gespecificeerd, met inbegrip van hun interacties, is een VHDL-beschrijving ook simuleerbaar met behulp van een simulatietool.

Entity en Architecture

De eerste stap in de VHDL-specificatie van een digitale schakeling bestaat uit een entity-beschrijving. Deze bevat de naam van de schakeling en de namen van de in- en uitgangspoorten met hun typen zoals aangegeven in listing 1. Het register, de latch en de combinatorische schakeling hebben hier allemaal poorten van type *bit*, één enkele verbinding. VHDL kent echter zeer uitgebreide typeringsmogelijkheden en het is heel goed mogelijk om bijvoorbeeld een poort van type *word* of *integer* te definiëren. Verder hebben de eenheden uit listing 1 alleen maar ingangs- en uitgangspoorten, maar een *port* kan ook bi-directioneel zijn (modus *inout*).

Voor elke *entity* is vervolgens een *architecture*-beschrijving vereist om een compleet systeem te specificeren. Zo'n architectuur kan twee aspecten combineren, een structurele en een gedragsmatige, maar voorbeeld listing 2 (naar Ashenden) is alleen structureel. Het geeft aan hoe register *reg4* is opgebouwd uit vijf componenten, nl. vier *latches* en een *and* schakeling. Er is bovendien een interne verbinding gedeclareerd, een *signal* van het type *bit*. De vier *latch* componenten hebben uiteraard allemaal verschillende namen maar het zijn uitvoeringen van dezelfde *entity*. De namen tussen haakjes, *latchgedrag* en *andgedrag*, verwijzen naar architecturen van de betreffende onderdelen. Deze verwijzingen zijn optioneel omdat een systeem op verschillende manieren, dus door meer architecturen, kan worden gespecificeerd. Zo kan de ontwerper in simulaties gemakkelijk alternatieven vergelijken, met name of samenwerkende componenten daadwerkelijk voldoen aan een gegeven gedragsspecificatie van het geheel.

De port map geeft aan welke poorten verbonden zijn met welke interne signalen. De latch-klok lclk van alle componenten is gekoppeld aan het signaal int_clk maar de andere in- en uitgangen zijn geassocieerd met verschillende registerpoorten. Poorten zijn signalen en verzorgen samen met de lokaal gedeclareerde signalen alle verbindingen tussen componenten. De associaties in een port map , aangegeven met het teken => , moeten uiteraard van hetzelfde type zijn. Dikwijls worden in een VHDL beschrijving de poortnaam en het associatiesymbool weggelaten en dan bepaalt de volgorde van de signaallijst wat bij wat hoort. Voor de leesbaarheid is zijn de associaties hier allemaal met => uitgeschreven. De uitgangspoort y van component *and_gate* is geassocieerd met het signaal int_clk. De poort heeft *mode out* en int_clk is met alle andere componenten verbonden via een *in* poort. Er zijn dus geen conflicten en de beschrijving is in dat opzicht correct, zoals een VHDL-analysator zal kunnen vaststellen. Zo'n analysator zal ook kunnen controleren of er geen losse verbindingen zijn en, niet te vergeten, of met elkaar verbonden in- en uitgangen wel hetzelfde type bezitten.

Met VHDL *entity*- en *architecture*-beschrijvingen zoals hier geschetst is het mogelijk aan te geven hoe een systeem is opgebouwd uit sub-

systemen met hun verbindingen. Deze subsystemen kunnen ook weer op die manier gespecificeerd zijn en uiteindelijk kan men zo terechtkomen bij bepaalde bekende basiscomponenten. Behalve een dergelijke structurele ontleding kan een *architecture* echter ook gedragsbeschrijvingen bevatten.

Functionaliteit

Wat *reg4* doet (of moet doen) valt niet direct af te lezen uit listing 2 maar het gedrag van een systeem wordt bepaald door het gedrag van de subsystemen en in laatste instantie dus door de basiscomponenten. De ontwerper kan in VHDL gedrag specificeren voor subsystemen op elk niveau met een groot aantal constructies die ook in programmeertalen zoals Pascal of het al genoemde ADA voorkomen.
Zo is in listing 3 het gedrag aangegeven in de toekenning van a *and* b aan y.
Van een daadwerkelijke combinatorische schakeling (in hardware) wordt aangenomen dat die de operatie uitvoert. In een simulatie van de schakeling wordt van de simulatie-tool verwacht dat die de and operatie op de gewenste manier implementeert.
De VHDL-taal zelf heeft rekenkundige en logische bewerkingen, controleconstructies zoals if ... then (zie listing 4), lussen en zelfs variabelen van allerlei typen tot en met arrays en records. Het is dan ook in principe heel goed mogelijk om embedded software in VHDL te schrijven!
Het verschil met Pascal of ADA is echter dat een VHDL beschrijving niet wordt gecompileerd tot een executeerbaar programma maar òf eindigt in een hardwarebeschrijving (via een synthesetool) òf in een simulatie.
Een VHDL gedragsbeschrijving is gemodelleerd als een verzameling parallelle processen die met elkaar kunnen communiceren door middel van signalen.
Een proces in VHDL kan een een naam (label) hebben en bestaat uit bewerkingen die na elkaar worden uitgevoerd (sequentieel) met de toevoeging dat er, net als bij een programmeertaal, sprongen en lussen kunnen bestaan binnen een proces. Bovendien kent VHDL ook procedures en functies.
Een proces eindigt echter niet maar loopt alle bewerkingen af tot een *wait* is bereikt. Indien een of meer van de signalen waar het proces op wacht weer verandert dan begint het proces opnieuw. (Zie deel 2 van dit artikel.)
In alternatieve notatie kunnen de signalen waar het proces op reageert (als ze veranderen) tussen haken achter *process* worden geschreven, zoals in listing 4 is gedaan. Dergelijke signalen zijn in zekere zin de parameters van het proces, maar alleen in die zin dat verandering (bijvoorbeeld door een ander proces) het proces opnieuw opstart.

De verschillende processen binnen een architectuur worden verondersteld parallel te verlopen. Hun onderlinge volgorde is niet van belang voor het totale gedrag van de *architecture* waarin zij zijn gedefinieerd.
Listings 3 en 4 beschrijven elk maar één proces maar het gedrag van *reg4* uit listing 2 wordt bepaald door de functionele architecturen *latchgedrag* en *andgedrag* (uit listing 3 en 4). Die staan tussen de haken achter de betreffende *entity*-namen in listing 2.

```
entity reg4 is
    port(rd0, rd1, rd2, rd3, en, clk: in bit;
      rq0, rq1, rq2, rq3 : out bit);
end entity reg4;

entity d_latch is
    port(ld, lclk : in bit; lq : out bit);
end entity d_latch;

entity and2 is
    port(a, b : in bit; y : out bit);
end entity and2;
```

Listing 1 (naar ref. 1): Entity-beschrijvingen in VHDL'93 van een 4-bits register, een d-latch en een and-poort. De VHDL gereserveerde woorden zijn vet afgedrukt.

```
architecture struct of reg4 is
    signal int_clk : bit;
begin
    latch0 : entity work.d_latch(latchgedrag)
     port map (ld ⇒ rd0, lclk ⇒ int_clk, lq ⇒ rq0);

    latch1 : entity work.d_latch(latchgedrag)
     port map (ld ⇒ rd1, lclk ⇒ int_clk, lq ⇒ rq1);

    latch2 : entity work.d_latch(latchgedrag)
     port map (ld ⇒ rd2, lclk ⇒ int_clk, lq ⇒ rq2);

    latch3 : entity work.d_latch(latchgedrag)
     port map (ld ⇒ rd3, lclk ⇒ int_clk, lq ⇒ rq3);

    and_gate : entity work.and2(andgedrag)
     port map (a ⇒ en, b ⇒ clk, y ⇒ int_clk);

end architecture struct;
```

Listing 2 (naar ref. 1): De structurele architectuur van het 4-bits register uit listing 1, opgebouwd uit 'latch' en 'and' entity-componenten. Er is tevens een intern kloksignaal gedeclareerd dat de 'and' wordt van externe klok en register 'enable'. Het teken ⇒ staat voor associatie van een entity-poort met een signaal. De toevoeging 'work' is bestemd voor de simulator en verwijst naar de werkbibliotheek.

Dit alles betekent dus dat het gedrag van *reg4* is samengesteld uit vijf (*concurrent*) processen met hun verbindingssignalen. Omdat de structurele architectuur van een systeem op een dergelijke manier te reduceren is tot een functionele specificatie, volgt uit de componentsamenstelling ook het gedrag.

Het is echter de kracht van VHDL dat een systeem op verschillende manieren beschreven kan worden. Sommige beschrijvingen lenen zich bij uitstek voor documentatiedoeleinden, andere richten zich meer op het optimale syntheseresultaat, hoewel dit niet tegenstrijdig hoeft te zijn.
Listing 5 geeft als voorbeeld een gedragsbeschrijving op hoog niveau van reg4. Het is dus zeker niet het geval dat VHDL de ontwerper beperkt tot systeembeschrijvingen op het laagste niveau.
VHDL beschrijvingen kunnen zelfs parametriseerbaar worden gemaakt. In het voorgaande werd verondersteld dat *reg4* op 4 bits brede data

```
architecture andgedrag of and2 is
begin
    and2_behavior : process is
    begin
        y ⇐ a and b after 2 ns;
        wait on a, b;
    end process and2_behavior;
end architecture andgedrag;
```

Listing 3 (naar ref. 1): Het gedrag van de 'and' uit listing 1 in een 'process'. Het teken ⇐ staat voor toekenning van signalen en er wordt een propagatietijd gespecificeerd van 2 nanoseconde. VHDL kent een groot aantal logische en rekenkundige bewerkingen waaronder 'and'. Het proces wacht tot ingangen a of b veranderen en begint in dat geval opnieuw.

```
architecture latchgedrag of d_latch is
begin
    latch_behavior : process( lclk, ld) is
     begin
      if lclk = '1' then
        lq ⇐ ld after 2 ns;
      end if;
    end process latch_behavior;
end architecture latchgedrag;
```

Listing 4 (naar ref 1): Het gedrag van de d-latch entity uit listing 1. Als signaal lclk of ld verandert wordt het proces gestart, maar ingangssignaal ld wordt alleen doorgegeven naar uitgang lq als klok lckl hoog is. Ook hier is een propagatietijd aangegeven.

werkt, maar wat te doen indien er een *reg3* of een *reg16* is vereist? Listing 6 geeft een voorbeeld waarbij de breedte parametriseerbaar is geworden. In de entity staat nu de generic n, die wordt gebruikt om de breedte van *d* en *q* te bepalen. In één ontwerp kunnen zelfs instantiaties van *reg* worden opgenomen met verschillende breedtes van *d* en *q*. In listing 7 is tenslotte aangegeven hoe twee beschrijvingen met elkaar kunnen worden vergeleken, waarbij de gebruiker de stimuli aanbiedt. Verificatie door middel van simulatie is natuurlijk van groot belang bij het ontwerp van digitale systemen en vergelijkingen zoals in listing 7 vormen dan een krachtige ondersteuning.

```
architecture reggedrag of reg4 is
begin
    process(rd0, rd1, rd2, rd3, en, clk)
    begin
     if en='1' and clk='1' then
         rq0 <= rd0;
         rq1 <= rd1;
         rq2 <= rd2;
         rq3 <= rd3;
        end if;
    end process;
end architecture reggedrag;
```

Listing 5 (bron: Molenkamp): Het gedrag van reg4 uit listing 1 gespecificeerd in een 'process'. De ontwerper heeft op hoger niveau de functionaliteit van het register aangegeven, zonder rekening te houden met parallellisme en tijdvertragingen.

```
entity reg is
    generic (n : positive := 4);
    port(d : in bit_vector (n-1 downto 0);
         en, clk: in bit;
       q : out bit_vector (n-1 downto 0));
end entity reg;

architecture gedrag of reg is
begin
    process(d, en, clk)
    begin
     if en='1' and clk='1' then
         q <= d;
        end if;
    end process;
end architecture gedrag;
```

Listing 6 (bron: Molenkamp): Parametriseerbare beschrijving van een register met een willekeurige breedte. De functionaliteit zoals gespecificeerd in listing 5 is in principe hetzelfde voor 3, 4, 16 of 32 bits registers en het volstaat om de 'generic' variabele n aan te passen aan de gewenste afmeting.

Een simulatie is echter uit de aard der zaak niet hetzelfde als de werkelijkheid en daarop zal nader worden ingegaan in het tweede deel van dit artikel. Ook mogelijkheden en beperkingen van synthese vanuit VHDL komen daarin aan de orde.

Met dank aan ir. E.Molenkamp voor zijn zeer gewaardeerde correcties en aanvullingen bij de totstandkoming van dit artikel. Eventuele onjuistheden blijven uiteraard geheel voor rekening van de auteur. ■

Referenties

1] Ashenden, J.P. „The Designer's Guide to VHDL". Morgan Kaufmann Publishers (1996). ISBN 1-55860-270-4

2] Molenkamp, E. „VHDL, VHDL '87/93 en Voorbeelden" (1997). ISBN 90-802634-3-5 (te bestellen via: www.cs.utwente.nl/~molenkam)

Trainingen en Tools op het gebied van HDL

Transfer Nederland BV, Goorseweg 5, 7475 BB Markelo, tel.: (0547) 367367, e-mail: info@transfer.nl, web: www.transfer.nl

Websites

http://tech-www.informatik.uni-hamburg.de/vhdl
www.cs.utwente.nl/~molenkam
www.eda.org
www.vhdl.org
www.ovi.org
www.pato.nl

```
entity compare is
  port(d : in bit_vector (3 downto 0);
     en, clk: in bit;
     correct  : out boolean);
end compare;

architecture struct of compare is
  signal qs,qp : bit_vector(3 downto 0);
begin
  structuur : entity work.reg4(struct)
          port map (d(3),d(2),d(1),d(0),en,clk,
                qs(3),qs(2),qs(1),qs(0));

  param     : entity work.reg(gedrag)
          generic map (4)
          port map (d,en,clk,qp);
```

Listing 7 (bron: Molenkamp): Het vergelijken van de parametriseerbare beschrijving van reg uit listing 6 met de architectuur reg4(struct) uit listing 2. De boole-variabele 'correct' geeft aan of de beschrijvingen gelijkwaardig zijn.