Digitale Systemen (ET1 410)

Arjan van Genderen
Stephan Wong

Faculteit EWI
Technische Universiteit Delft
Cursus 2010-2011

VHDL overzicht

- Wat is VHDL? Waarvoor gebruiken we het?
- Deze college
- Sequentieel
- Verschil simulatie en synthese in VHDL
- Processen
- Etc..

VHDL?

- VHDL is een *hardware modeleringstaal* en niet een programmeertaal (zoals C, Java, etc.) → verschil??
- Wat modeleren we in VHDL?
  - Applicaties (denk aan software)
  - Systemen (denk aan computers)
  - Architecturen (denk aan processoren)
  - Hardware (denk aan componenten)
- Wat kunnen we doen met zo'n model?
  - Een ontwerp specificeren (omschrijven)
  - Een ontwerp simuleren (testen, nabootsen van realiteit)
  - Een ontwerp synthetiseren (implementeren)

Sequentieel vs. Concurrent

```
if (a = b)
  {<do something>;}
else
  {<do something else>;}
```

```
while (a = b){
  <task 1>;
  <task 2>;
}
```

```
for (int i = 0; i < 10; i++) {
  <task A>;
  <task B>;
}
```

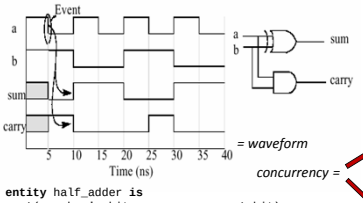
Wat is karakteristiek aan alle drie sequentiële codes?

Wat is de waarde van a?

```
int a=1, b=1;
if (b=0)
      a = 2;
else
      a = 3;
```

Sequentiële stappen

Sequentieel vs. Concurrent



= waveform

concurrency =

Wat gebeurt er als signaal 'a' verandert?

Dit zal tot gevolg hebben:
1. de ingangen van de 'exor'-gate en de 'and'-gate veranderen **op dezelfde moment.**
2. als beide gates dezelfde vertraging hebben, dan zal ook de uitgangen **op dezelfde moment** veranderen.

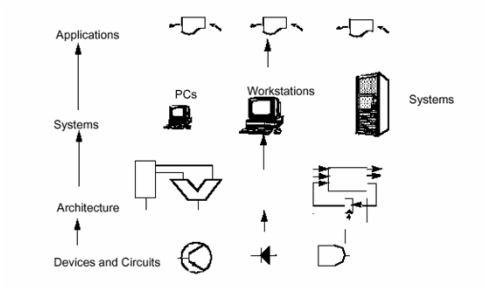
```
entity half_adder is
port( a, b: in bit; sum, carry: out bit);
end half_Adder;

architecture my_half_adder_arch of Half_Adder is
begin
carry <= a and b after 5 ns;
sum   <= a exor b after 5 ns;
end my_half_adder_arch;
```

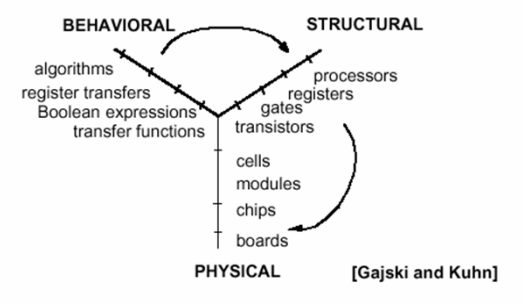

VHDL?

- VHDL is een *hardware modeleringstaal* en niet een programmeertaal (zoals C, Java, etc.)
- Wat modeleren we in VHDL?
  - Applicaties (denk aan **software**)
  - Systemen (denk aan **computers**)
  - Architecturen (denk aan **processoren**)
  - Hardware (denk aan **componenten**)
- Wat kunnen we doen met zo'n model?
  - Een ontwerp specificeren (omschrijven)
  - Een ontwerp simuleren (testen, nabootsen van realiteit)
  - Een ontwerp synthetiseren (implementeren)

Niveaus van abstractie



Verschillende systeembeschrijvingen



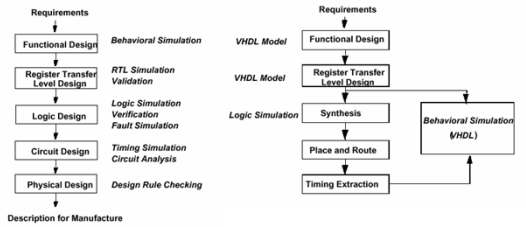
VHDL?

- VHDL is een *hardware modeleringstaal* en niet een programmeertaal (zoals C, Java, etc.)
- Wat modeleren we in VHDL?
  - Applicaties (denk aan software)
  - Systemen (denk aan computers)
  - Architecturen (denk aan processoren)
  - Hardware (denk aan componenten)
- Wat kunnen we doen met zo'n model?
  - Een ontwerp specificeren (omschrijven)
  - Een ontwerp simuleren (testen, nabootsen van realiteit)
  - Een ontwerp synthetiseren (implementeren)

Waarom 'beschrijven'/modeleren?

- Specificatie van ontwerp:
  - *Ondubbelzinnig definitie van functionaliteiten, componenten, en interfaces in een groot ontwerp*
- Simulatie van ontwerp:
  - *Verificatie van het systeem/deelsystemen/chip in termen van prestatie, functionaliteit, etc. voordat deze wordt geïmplementeerd*
- Synthese van ontwerp:
  - *Automatische generatie van een hardware ontwerp*

VHDL binnen een digitale systeemontwerpomgeving



- Automatische verfijning van ontwerpstappen
- Terugkoppeling voor nauwkeurigere simulatie
- Target technologieën: ASICs en FPGAs

VHDL als taal

V *Very High Speed Integrated Circuit*

H *Hardware*

D *Description*

L *Language*

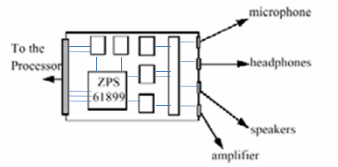
Waarom was zo'n hardware taal nodig?
- Interoperability: uitwisselen van ontwerpen
- Technologieonafhankelijk
- Hergebruik van component beschreven in VHDL

Geschiedenis van VHDL

- Ontworpen door IBM, Texas Instruments, Intermetrics als deel van een DoD gefinancierde VHSIC programma
- Gestandaardiseerd door de IEEE in 1987: IEEE 1076-1987
- Verbeterde versie gedefinieerd in 1993: IEEE 1076-1993
  - Software pakketten vragen altijd welke versie je wilt gebruiken
  - Subtiele syntax verschillen → syntax errors → DUS: let op!!!
- Additioneel gestandaardiseerde packages voor definities van data types en expressie van timing data:
  - IEEE 1164 (data typen)
  - IEEE 1076.3 (numeric)
  - IEEE 1076.4 (timing)

Wat heb ik nodig in VHDL?

Neem het volgende voorbeeld van een geluidskaart:

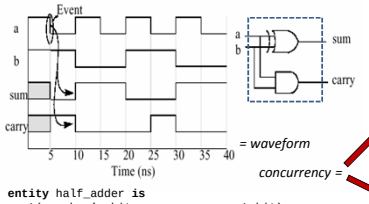


Hoe vertel ik een vriend in US over de telefoon deze op te bouwen als we dezelfde componenten tot onze beschikking hebben?

VHDL: Belangrijke vereisten

- Kijken naar de realiteit en nagaan wat ik nodig heb
- Bijv. hoe bouw ik een systeem in het algemeen?
  1. Specificeer ingangen en uitgangen → *entity* (& *ports*)
  2. Specificeer functie → *architecture*
     a. Functioneel (bijv. met 'a+b' of 'a and b') → *behavioral*
     b. Structureel (met componenten) → *structural*
  3. Specificeer signalen (→ *signals*) en welke waarden ze kunnen aannemen (→ *values*) en:
     a. Veranderingen in tijd → *events*
     b. Vertragingen → *propagation delays*
     c. concurrency
     d. Geordende serie van events in tijd→ *waveform*

Nogmaals de half-adder voorbeeld




```
entity half_adder is
port( a, b: in bit; sum, carry: out bit);
end half_Adder;

architecture my_half_adder_arch of Half_Adder is
begin
carry <= a and b after 5 ns;
sum   <= a exor b after 5 ns;
end my_half_adder_arch;
```

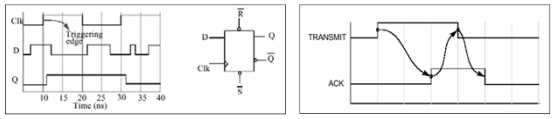
Wat gebeurt er als signaal 'a' verandert?

Dit zal tot gevolg hebben:
1. de ingangen van de 'exor'-gate en de 'and'-gate veranderen op dezelfde moment.
2. als beide gates dezelfde vertraging hebben, dan zal ook de uitgangen op dezelfde moment veranderen.

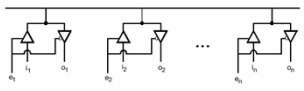
- Entity? Ports?
- Architecture?
- Signals?
- Values?
- Events?
- Propagation delays?
- Waveform?
- Behavioral or Structural?

VHDL: Nog meer vereisten

- Timing: *events* die gebeuren alleen op bepaalde tijdstippen, zoals na een klok-signaal (synchroon) of na een ander signaal (asynchroon) → *"wait for"* (behandeld in volgend college)



- Bussignalen → *shared signals* (behandeld in latere college)



Een enkele lijn die is aangesloten op meerdere "drivers"

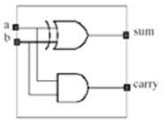
VHDL: Signal Values

- **`Bit`** is een standaard gedefinieerde signaaltype die de waarden '0' en '1' kan aannemen
- Een ander vaak gebruikte signaaltype is de IEEE 1164 value system: *(waarom? → denk aan simulatie)*

Value	Interpretation
U	Uninitialized
X	Forcing Unknown
0	Forcing 0
1	Forcing 1
Z	High Impedance
W	Weak Unknown
L	Weak 0
H	Weak 1
-	Don't Care

Basisconcepten van VHDL

- Design entity: *(wat willen we beschrijven)*
  - Interface → *hoe kunnen het ontwerp verbinden?*




```
entity half_adder is
port( a, b: in bit;
      sum, carry: out bit );
end half_Adder;
```

VHDL is case-insensitive!

Een vaak gemaakte syntax-fout!

Interface is een verzameling van *ports*:
- Ports gebruiken: *signals*
- Ports hebben een type, bv. `bit` (dit is dus een enkele bit)
- Ports hebben een mode: `in`, `out`, `inout` (bidirectioneel)

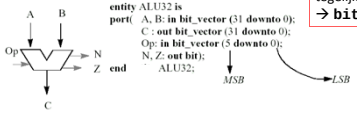
  - Functie → *wat doet het? (hier komen we later op terug)*

Het objecttype: *signal*

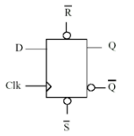
- VHDL ondersteunt 4 basisobjecten: *variables*, *constants*, *signals*, en *file types* (1993)
- *Variables* en *constants* kunnen alleen een enkele waarden aannemen waarbij *constants* niet meer veranderen na gedefinieerd te zijn
- *Signals* zijn speciaal (denk aan echte systemen):
  - Het heeft ook een **tijdscomponent** (anders dan *variables*) naast de waarden die het kan aannemen
  - Implementatie van een signaal is dus een serie van **tijd-waarde** paren!!
- *File types* zullen we in dit college niet behandelen

Meer *entity* voorbeelden

Wat als we meerdere bits tegelijkertijd willen specificeren?
→ `bit_vector`

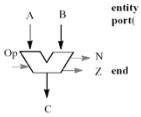



```
entity ALU32 is
port( A, B: in bit_vector (31 downto 0);
      C : out bit_vector (31 downto 0);
      Op: in bit_vector (5 downto 0);
      N, Z: out bit);
end    ALU32;
```



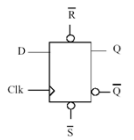

```
entity D_ff is
port(  D, Q, Clk, R, S: in bit;
       Q, Qbar : out bit);
end    D_ff;
```


Nog meer *entity* voorbeelden




```
entity ALU32 is
port( A, B: in std_ulogic_vector (31 downto 0);
      C : out std_ulogic_vector (31 downto 0);
      Op: in std_ulogic_vector (5 downto 0);
      N, Z: out std_logic);
end    ALU32;
```

Wat als de waarden '0' en '1' niet toereikend zijn?
→`std_ulogic`
→`std_ulogic_vector`

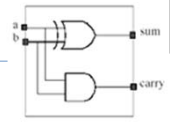



```
entity D_ff is
port(  D, Q, Clk, R, S: in std_ulogic;
       Q, Qbar : out std_ulogic);
end    D_ff;
```


Het gebruik van *std_ulogic*

Wie ziet het verschil?!!?

Libraries zijn repositories van packages!
Packages zijn repositories van functies, procedures, types, etc., (deze worden behandeld in latere college)
→*gelden alleen voor volgende entity en architecture!*




```
entity half_adder is
port( a, b: in bit;
      sum, carry: out bit);
end half_adder;

architecture my_half_adder_arch of
half_adder is
begin
carry <= a and b after 5 ns;
sum   <= a exor b after 5 ns;
end my_half_adder_arch;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity half_adder is
port( a, b: in std_ulogic;
      sum, carry: out std_ulogic);
end half_adder;

architecture my_half_adder_arch of
half_adder is
begin
carry <= a and b after 5 ns;
sum   <= a exor b after 5 ns;
end my_half_adder_arch;
```


De *architecture* construct

```
architecture my_half_adder_arch of half_adder is
-- declarative region -- zo schrijf je dus een comment!!
Begin
-- architecture body
carry <= a and b after 5 ns;
sum   <= a exor b after 5 ns;
end my_half_adder_arch;
```

- *declarative region:* instantieren van variables, signalen, components, etc.
- *architecture body:* werkelijke beschrijving van de functionaliteit van de *entity* en bevat vaak *simple signal assignment statements*, zoals bijv.:
  - `carry <= a and b after 5 ns;`
- Nogmaals: signalen zijn tijd-waarde paren!! Hoe kan ik dat aflezen?
- Wanneer wordt de assignment uitgevoerd? → als a of b veranderen
- Wanneer krijgt `carry` de nieuwe waarde? → 5ns nadat a of b verandert zijn
- Kan de 2 signal assigments regels omwisselen zonder gevolgen? → Ja

Meer *simple signal assignments*

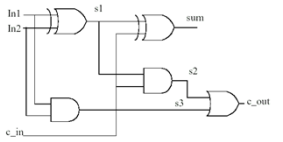
```
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
port (in1, in2, c_in: in std_ulogic;
      sum, c_out: out std_ulogic);
end          full_adder;

architecture dataflow of full_adder is
signal s1, s2, s3 : std_ulogic;
constant gate_delay: Time:= 5 ns;
begin
L1: s1 <= (In1 xor In2) after gate_delay;
L2: s2 <= (c_in and s1) after gate_delay;
L3: s3 <= (In1 and In2) after gate_delay;
L4: sum <= (s1 xor c_in) after gate_delay;
L5: c_out <= (s2 or s3) after gate_delay;
end          dataflow;
```

Definitie van tussensignalen

Definitie van een constant

Gebruik van labels



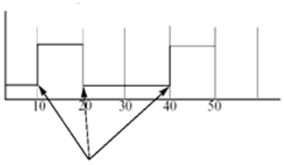
VRAAG: is het belangrijk op welke volgorde de *signal assignments* staan??

→ Nee, zie vorige slide!!

De *simple signal assignment statement*

- Signalen worden gebruikt om componenten binnen een ontwerp met elkaar te verbinden
- Een *signal assignment statement* wordt uitgevoerd wanneer een signaal in de RHS (right hand side) van de statement een nieuwe waarde krijgt toegewezen:
  - 1-op-1 relatie tussen *signal statement assignments* en signalen binnen een circuit
  - Volgorde van executie van de statements volgt de propagatie van *events* binnen een circuit
  - Volgorde in de tekst hoeft niet de echte volgorde te zijn

Waveform generatie



signal transitions for each waveform element

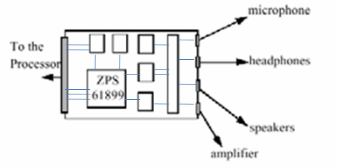
signal <= '0','1' after 10 ns,'0' after 20 ns,'1' after 40 ns;

- Meerdere waveform elementen (*events*) kunnen in een enkele *signal assignment statement* worden gespecificeerd

Structurele beschrijvingen

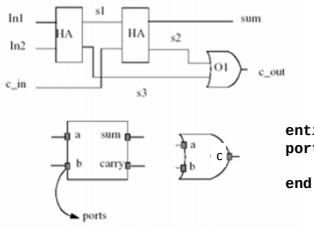
- Kijken naar de realiteit en nagaan wat ik nodig heb
- Bijv. hoe bouw ik een systeem in het algemeen?
  1. Specificeer ingangen en uitgangen → *entity* (& *ports*)
  2. Specificeer functie → *architecture*
     a. Functioneel (bijv. met 'a+b' of 'a and b') → *behavioral*
     b. Structureel (met componenten) → *structural*
  3. Specificeer signalen (→ *signals*) en welke waarden ze kunnen aannemen (→ *values*) en:
     a. Veranderingen in tijd → *events*
     b. Vertragingen → *propagation delays*
     c. *concurrency*
     d. Geordende serie van events in tijd→ *waveform*

Modellering van structuur



- Een structurele model beschrijft een digitale systeem als een verbindingsnetwerk van componenten
- Beschrijving van elk component moet beschikbaar zijn als *structural* of *behavioral*

Modellering van structuur




```
entity or_2 is
port (a, b: in std_ulogic;
      c: out std_ulogic);
end or_2;
```

- Definieer alle componenten die worden gebruikt
- Beschrijf de verbindingen tussen de componenten

Modellering van structuur

```
architecture structural of full_adder is
component half_adder is -- the declaration
port (a, b: in std_logic;
      sum, carry: out std_logic);
end component  ;

component or_2 is
 port(a, b : in std_logic;
      c : out std_logic);
end component  ;

signal s1, s2, s3 : std_logic;
begin
H1 : half_adder port map (a => In1, b => In2, sum=>s1, carry=>s3);
H2:half_adder port map (a => s1, b => c_in, sum =>sum,
            carry => s2);
O1 : or_2 port map (a => s2, b => s3, c => c_out);
end structural;
```

```
entity full_adder is
port (In1,In2: in std_logic;
      c_in: in std_logic;
      sum,c_out: out std_logic);
end full_adder;
```

unique name of the components
component type
interconnection of the component ports
component instantiation statement

Conditional Signal Assignment

Wat is een multiplexer??

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
port (    In0, In1, In2, In3 : in std_logic_vector (7 downto 0);
          Sel: in std_logic_vector(1 downto 0);
          Z : out std_logic_vector (7 downto 0));
end       mux4;

architecture behavioral of mux4 is
begin
Z <= In0 after 5 ns when Sel = "00" else
     In1 after 5 ns when Sel = "01" else
     In2 after 5 ns when Sel = "10" else
     In3 after 5 ns when Sel = "11" else
     "00000000" after 5 ns;
end               behavioral;
```

note type!

De eerste conditie die waar is wordt uitgevoerd!

Evaluation order is important!

Waarom deze laatste conditie??

Selected Signal Assignment

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
port (    In0, In1, In2, In3 : in std_logic_vector (7 downto 0);
          Sel: in std_logic_vector(1 downto 0);
          Z : out std_logic_vector (7 downto 0));
end       mux4;

architecture behavioral-2 of mux4 is
begin
with Sel select
Z <= (In0 after 5 ns) when "00",
     (In1 after 5 ns) when "01",
     (In2 after 5 ns) when "10",
     (In3 after 5 ns) when "11"
     (In3 after 5 ns) when others;
end ;             behavioral;
```

All options must be covered and only one must be true!

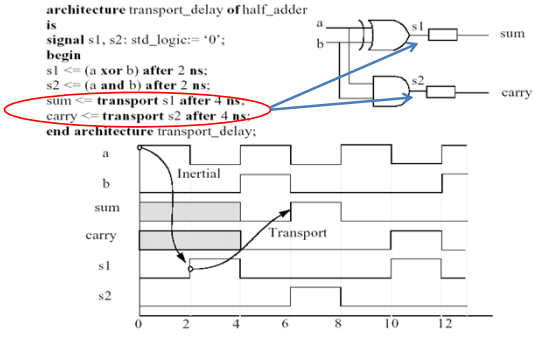
Waarom deze laatste conditie??

Delay Models

- Inertial delay
  - Standaard model
  - Geschikt voor modeleren van vertragingen van gates
- Transport delay
  - Modeleert (kleine) vertragingen door "draden" (wires)
  - Alle ingangen worden doorgegeven aan de uitgang
- Delta delay
  - Wat als er geen propagatie vertraging is?
  - Oneindig kleine vertraging die automatisch wordt toegevoegd door de simulator om correcte volgorde van events te waarborgen

Transport Delay Model

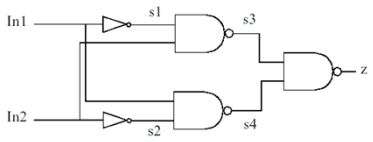
```
architecture transport_delay of half_adder
is
signal s1, s2: std_logic:= '0';
begin
s1 <= (a xor b) after 2 ns;
s2 <= (a and b) after 2 ns;
sum <= transport s1 after 4 ns;
carry <= transport s2 after 4 ns;
end architecture transport_delay;
```



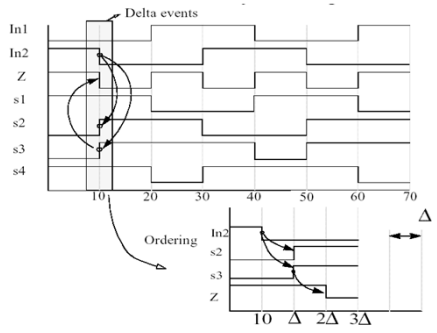
Delta Delay: een voorbeeld

```
library IEEE;
use IEEE.std_logic_1164.all;
entity combinational is
port (In1, In2: in std_logic;
    z : out std_logic);
end         combinational;
```

```
architecture behavior of combinational
signal s1, s2, s3, s4: std_logic:= '0';
begin
s1 <= not In1;
s2 <= not In2;
s3 <= not (s1 and In2);
s4 <= not (s2 and In1);
z <= not (s3 and s4);
end                 behavior;
```



Delta Delays: een voorbeeld



Discrete Event Simulator??

Configurations??