Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Digitale Systeem Engineering 1

Week 1 – VHDL basics, datatypes, signal assignment

Jesse op den Brouw

DIGSE1/2020-2021

DE HAAGSE HOGESCHOOL

Wat is VHDL

- VHDL = VHSIC Hardware Description Language
- VHSIC = Very High Speed Integrated Circuit

- VHDL is een modelleringstaal voor hardware.

- VHDL is een modelleringstaal voor *concurrent* systemen.

- Oorspronkelijk: taal voor beschrijving, simulatie en documentatie van digitale systemen.

- Nu: taal voor beschrijving, simulatie, documentatie en synthese van digitale systemen.

concurrent = gelijktijdig, parallel

DE HAAGSE HOGESCHOOL

Wat is VHDL

- Waarom beschrijven/modelleren?

- Specificatie van het ontwerp:
  - Eenduidige definitie van functionaliteiten, componenten en interfaces in een groot ontwerp.

- Simulatie van het ontwerp:
  - Verificatie van het systeem/deelsystemen/chip in termen van prestatie, functionaliteit etc. voordat deze wordt geimplementeerd.

- Synthese van het ontwerp:
  - Automatisch genereren van een hardware ontwerp.

Bron: Stephan Wong

Wat is VHDL

- Eerste ontwikkelingen rond 1970 (Ada).
- Eerste standaardisatie in 1987 door IEEE (IEEE 1076).
- Later aangepast in 1993 (VHDL-93), in 2001 (VHDL-2001), in 2008 (VHDL-2008).
- Specifiek voor synthese: IEEE 1076-6.

IEEE: (“Eye-triple-E”), Institute of Electrical and Electronics Engineers.

Wat is VHDL

- VHDL is platform-onafhankelijk[1].
- VHDL ondersteunt hiërarchisch ontwerpen, wat belangrijk is voor grote systemen.
- VHDL is *module-based met generieke parameters.*
  - mogelijkheid tot instantiëring maar niet tot overerving.
  - parameters worden verwerkt bij instantiëring.

1) Natuurlijk is dat wel betrekkelijk. Voor ontwerpen die niet “on the bleeding edge” van de technologie liggen is dit waar. Het gaat niet meer op voor ontwerpen die gebruik maken van fabrikant-specifieke componenten.

Wat is VHDL

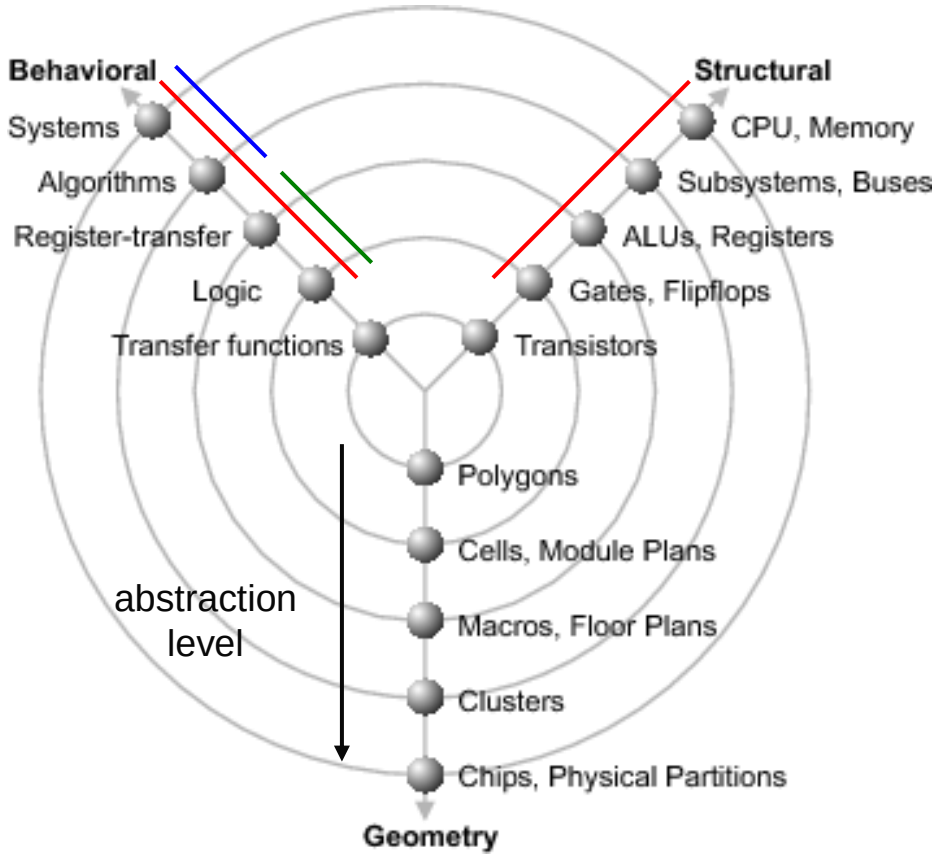
- VHDL is ontwikkeld door ontwerpers met jarenlange ervaring.
- Met VHDL is het “eenvoudig” om grote, ingewikkelde digitale systemen te ontwikkelen.
- Wij gebruiken en beschrijven VHDL-93. Veel producenten voldoen inmiddels aan de VHDL-2008 standaard.
- VHDL is nauw gekoppeld aan simulatie.

Wat is VHDL niet

- VHDL is *geen* programmeertaal. Het is een beschrijvingstaal.
  - Dit levert problemen op voor softwareschrijvers. Ze moeten anders leren denken: concurrent i.p.v. sequentieel.
- VHDL is geen Verilog, SystemC.
- Verilog is een HDL die veel in Amerika gebruikt wordt.
- SystemC is verwant aan C.

Ontwerp- en abstractieniveau's

- Ontwerpen kunnen op drie manieren bescheven worden.
- Behavioral – het gedrag van (een deel van) het ontwerp wordt beschreven.
- Structural – het ontwerp wordt beschreven in termen van deelontwerpen.
- Geometry – Het ontwerp wordt beschreven in termen van fysieke plaatsing van structuren op een IC.



kan in VHDL

high-level synthese

synthese

Gajski-Kuhn Y-chart

DE HAAGSE HOGESCHOOL

Ontwerp- en abstractieniveau’s

- Wij zullen ons voornamelijk bezig houden met de volgende niveau’s:

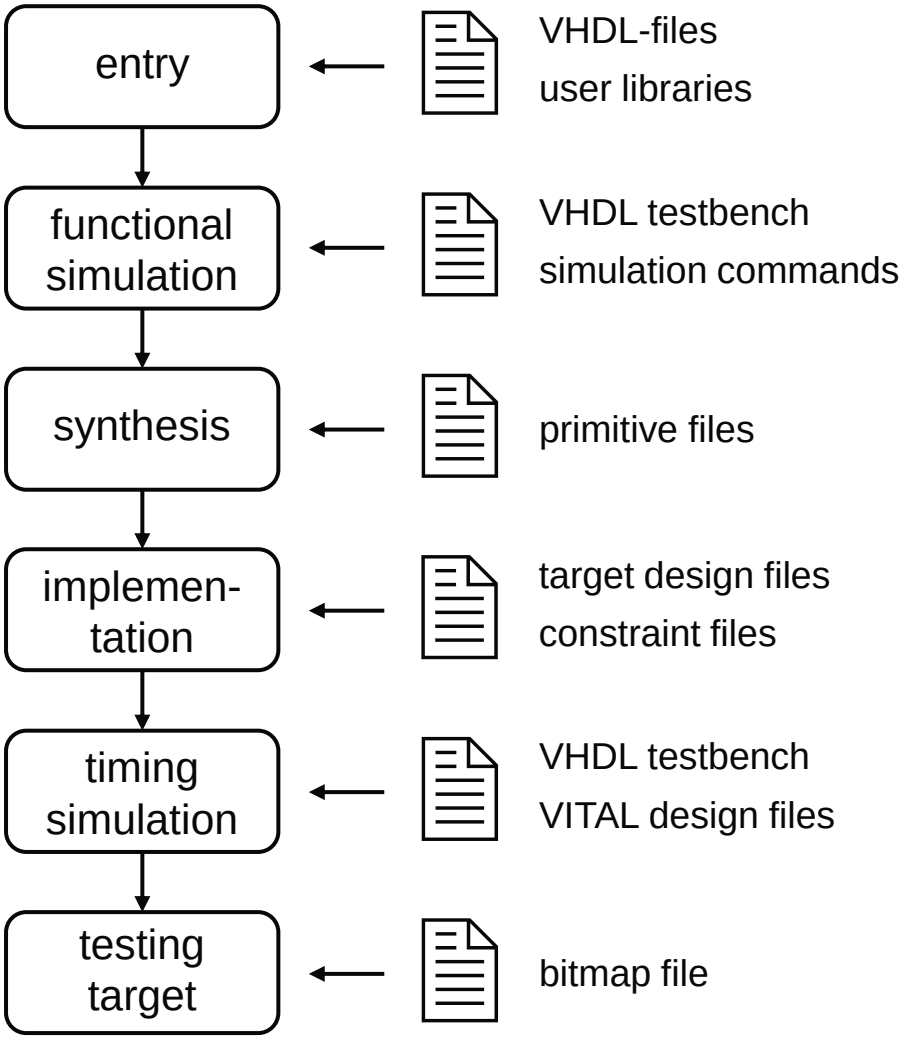
- **Register Transfer (RT)**

  Hierbij wordt het systeem verdeeld in afzonderlijke blokken die eenvoudig beschreven kunnen worden zoals: toestandmachines, registers, tellers, optellers, rom-tabellen. De functionaliteit van het systeem is niet meer in één oogopslag te zien. Simulatie en synthese zijn mogelijk met VHDL.

- **Logic**

  Hierbij wordt het systeem volledig beschreven in logische functies, met AND, OR en NOT. De functionaliteit van het systeem is niet meer te zien. Simulatie en synthese zijn mogelijk met VHDL.

VHDL FPGA Design Flow



- Hiernaast is de design flow weergegeven van een FPGA.

Simulatie en synthese

- Nadat een digitaal ontwerp beschreven is in VHDL dient het *gesimuleerd* te worden.
- Simulatie wordt gebruikt voor *verificatie* van het ontwerp; doet het ontwerp wat het moet doen?
- Simulatoren worden door verschillende fabrikanten gemaakt.
  - Wij gebruiken Modelsim
- Simulatie is een model van de werkelijkheid en kan dus afwijken van die werkelijkheid.

Simulatie en synthese

- Is gebleken dat de simulatie goed is verlopen, dan kan het ontwerp *gesynthetiseerd* worden.

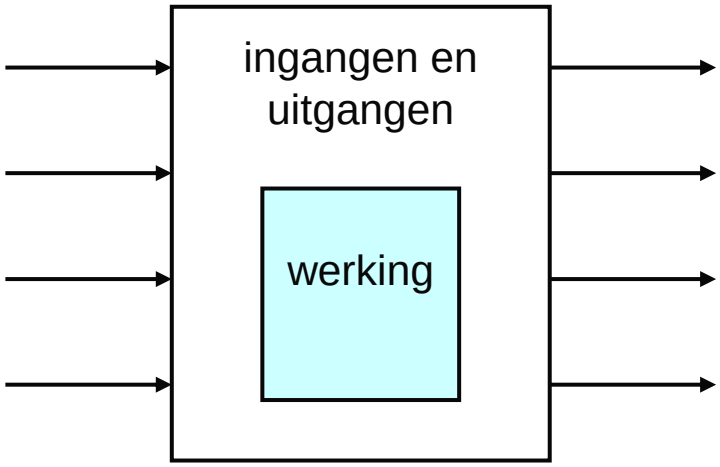
- Synthese is het automatisch vertalen van VHDL code naar *structuren* (poorten, flipflops, adders, muxen) op een chip. Dit kan een configureerbaar IC (FPGA, CPLD) zijn of een ASIC.

- Synthese is een lastig proces. De synthesesoftware is nog volop in ontwikkeling.

- Het is mogelijk VHDL-code te schrijven die wel simuleerbaar is, maar niet synthetiseerbaar.

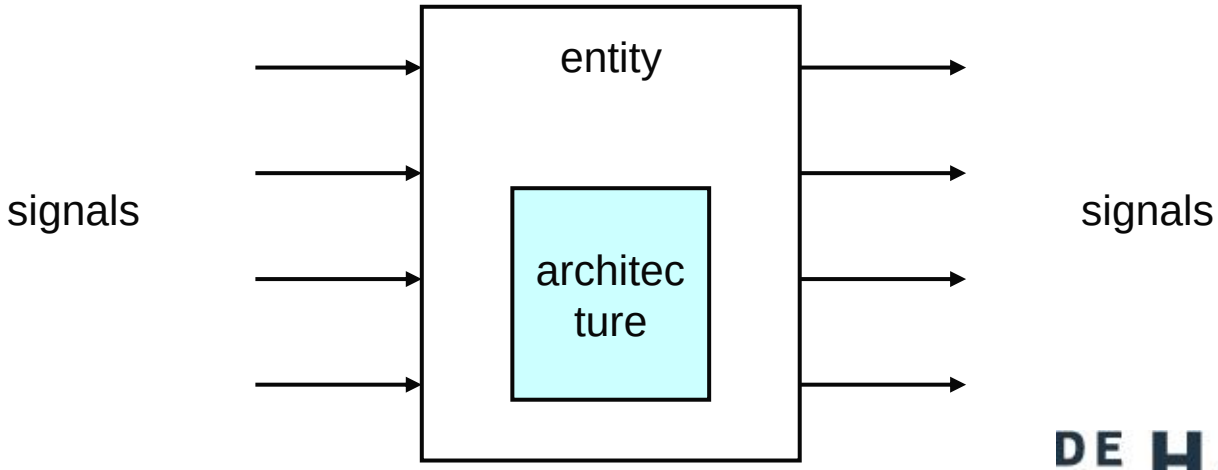
Design unit

- Een VHDL-ontwerp is opgebouwd uit *design units*.
- Een design unit kan worden voorgesteld als een black box.
- Deze black box heeft ingangen, uitgangen en een *werking.*



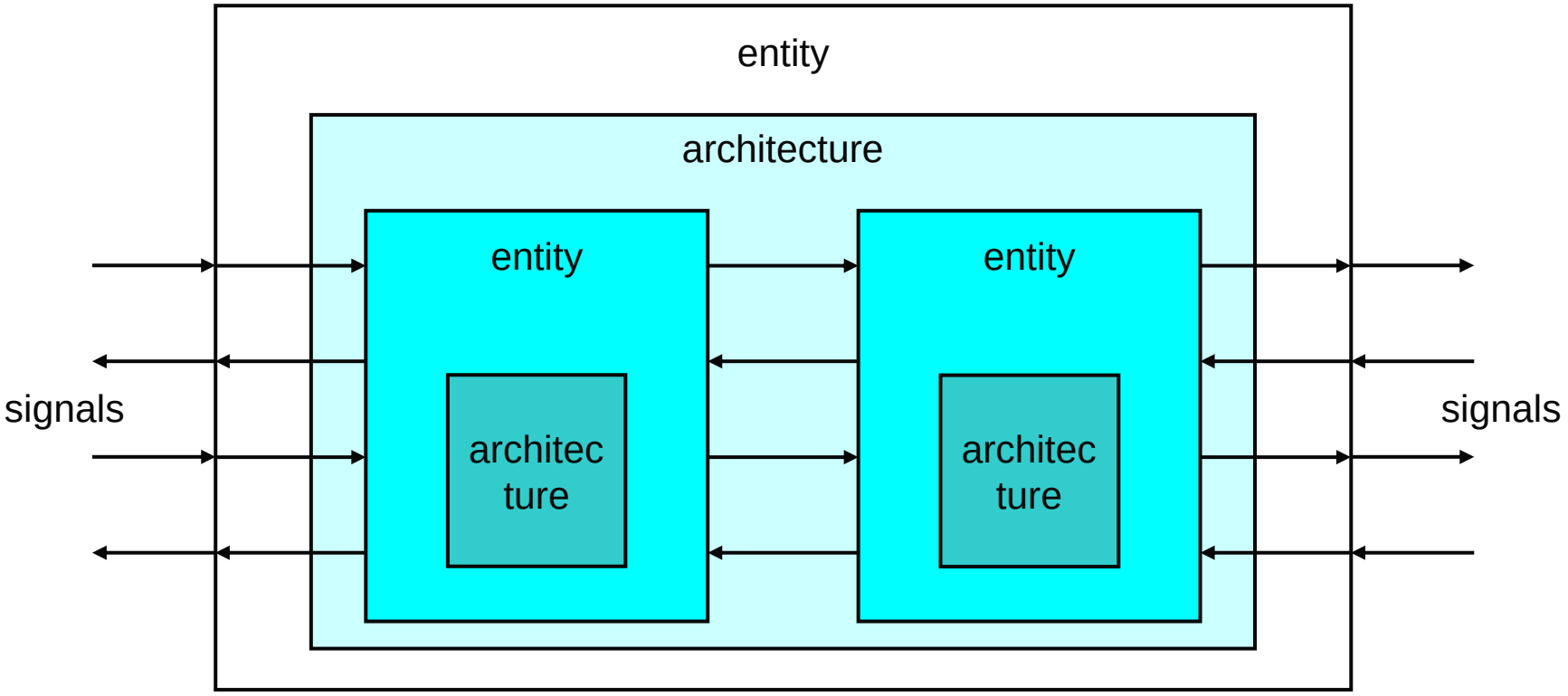
Design unit

- Een design unit bestaat uit een *entity* en een *architecture*.
- Merk op dat voor eenvoudige ontwerpen de gehele VHDL-beschrijving uit één design unit bestaat.
- Design units wisselen informatie uit via *signals*.



Design unit

- Design units kunnen gekoppeld en ingebed worden in een hogere hierarchische laag (*structural VHDL*).



Entity

- Een entity-declaratie beschrijft de koppeling (*interface*) tussen de design unit en de omgeving waarin het wordt gebruikt.

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

- De signalen a en b zijn ingangen (`in`) van de design unit, sum en carry zijn uitgangen (`out`). Alle signalen zijn van het type `bit`.

Architecture

- Een architecture beschrijft de *werking* van de design unit.

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

- Te zien is dat er twee *concurrent signal assignment statements*. De volgorde van de statements is niet van belang. Naast de logische functie is ook een vertraging opgegeven.

- Deze vorm wordt *dataflow VHDL* genoemd.

Signals

- Signals worden gebruikt om verbindingen (*wires*) te modelleren.
- Signals hebben een waarde- en een tijdcomponent (simulatie).
- Van signals is een timingdiagram te maken.
- Waarden aan signals worden toegekend met de *signal assignment operator* <=

```
r <= a;  -- geen after, zie slides week 2
s <= b after 5 ns;
t <= c and d after 10 ns;
```

- Opmerking: de namen die bij de port-beschrijving worden opgegeven zijn per definitie signals.

Datatypes

- Net als in een “gewone” programmeertaal heeft VHDL een aantal datatypes. De meest belangrijke zijn:

`boolean`	kan de waarde TRUE of FALSE hebben
`integer`	gehele getallen tussen -2147483647* en 2147483647 (32 bits, 2’s complement)
`bit`	kan de waarde ‘0’ of ‘1’ hebben
`character`	kan een karakter bevatten

- Deze kunnen allemaal gesynthetiseerd worden.

* Dit is het gegarandeerde bereik. Let op de symmetrie in de positieve en negatieve getallen. De meeste systemen kunnen ook – 2147483648 weergeven.

Datatypes

- Daarnaast zijn er nog twee andere:

`time`	kan een tijdwaarde bevatten (denk aan simulatie)
`real`	kan een kommagetal (floating point) bevatten

- Time kan niet gesynthetiseerd worden.
- Reals* kunnen gesynthetiseerd worden (VHDL-2008), maar leveren veel hardware op. Probeer reals te vermijden.

* Er is een nieuw type, `float`, dat gesynthetiseerd kan worden.

bit

- Het type bit is het standaard datatype voor binaire signalen.
- Een `bit` kan '0' of '1' zijn.
- Voorbeelden

```
signal t : bit;

t <= '0';  -- let op de quotes
t <= '1' after 5 ns;
```

std_ulogic

- Het type `bit` voldoet niet in alle gevallen (simulatie en synthese).
  - Wat als een waarde onbekend is?
  - Hoe kan een signal op don't care gezet worden?
  - Hoe worden pull-up/pull-down signalen beschreven?
  - Wat als een uitgang als tri-state gedefinieerd moet worden?

- Hiervoor is een apart datatype ontwikkeld: `std_ulogic`

- Het is gedefinieerd door IEEE (IEEE 1164-1993) en het moet in een VHDL-ontwerp geladen worden uit een bibliotheek.

std_ulogic

- Het datatype heeft ***9*** (negen) verschillende waarden (let op: hoofdletters):

```
type std_ulogic is
    ( '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
 );
```

std_ulogic

- Een voorbeeld van een two-input NAND (logisch gedrag).

```
library ieee;
use ieee.std_logic_1164.all;
entity nand_2 is
    port (a, b:   in   std_ulogic;
          c:      out  std_ulogic);
end entity nand_2;

architecture logic of nand_2 is
signal int: std_ulogic;   -- intern signaal
begin
    int <= a and b;
    c   <= not int;
end architecture logic;
```

std_logic

- Als een `std_ulogic` signal wordt aangestuurd door meerdere *drivers*, resulteert dat in een simulatiefout.
- Om toch te kunnen simuleren is het zogenaamde *resolved* type `std_logic` ontwikkeld.
- Resolved wil zeggen dat als een signaal door twee of meer drivers wordt aangestuurd, een *resolution functie* wordt aangeroepen om zo het conflict op te lossen en de nieuwe waarde van het signaal te bepalen.
- `std_logic` is het aanbevolen type.

std_logic resolution table

```
-------------------------------------------------------------------
-- resolution function
-------------------------------------------------------------------
CONSTANT resolution_table : stdlogic_table := (
--      ---------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -      |   |
--      ---------------------------------------------------
        ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
        ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
        ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
        ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
        ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
        ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
        ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
    );
```

- In totaal 81 verschillende mogelijkheden!

std_logic

- Een voorbeeld van een three-input NOR (logisch gedrag).

```
library ieee;
use ieee.std_logic_1164.all;
entity nor_3 is
    port (a, b, c:   in   std_logic;
          d:         out  std_logic);
end entity nor_3;
architecture logic of nor_3 is
signal int: std_logic;
begin
    int <= a or b or c;
    d   <= not int;
end architecture logic;
```

Vector

- Het is mogelijk om een *array* te definieren.
- Een array is een manier om bij elkaar horende gegevens te groeperen.
- Een andere, in de computerindustrie gebruikte term is *bus*.
- Een naam die in VHDL wordt gebruikt is *vector*.

Vector

- Twee voorbeelden van vectors:

```
signal test: array (0 to 3) of bit;
signal PC: array (31 downto 0) of std_logic;
```

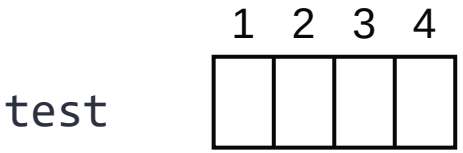
`to`	oplopende index, linker bit is laagste en rechter bit is hoogste,
`downto`	aflopende index, linker bit is hoogste en rechter bit is laagste.

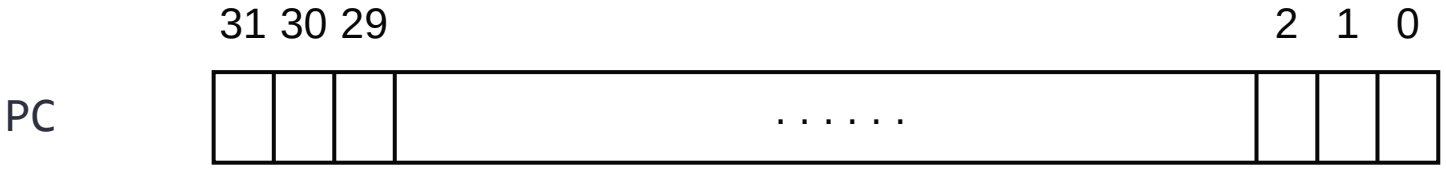
- Het gebruik van `downto` sluit heel goed aan bij de manier waarop ontwerpers denken. Het meest linker bit is het hoogste bit.

Vector

- Twee voorbeelden van vectors:

```
signal test: array (1 to 4) of bit;
signal PC: array (31 downto 0) of std_logic;
```





Vector

- Er is ook een type `bit_vector` en `std_logic_vector` zodat één en ander eenvoudiger geschreven kan worden:

```
signal test: bit_vector (1 to 4);
signal PC: std_logic_vector (31 downto 0);
```

- Toekennen gaat eenvoudig, middels een *bitstring*:

```
signal maximum: bit_vector (3 downto 0);
...
        maximum <= "1011";
```

DE HAAGSE HOGESCHOOL

Vector

- Toekennen en gebruik van één enkele index kan ook:

```
test(3) <= '1';          -- index 3 of test is '1'
test(2) <= test(1);      -- copy index 1 to index 2
```

- Een deel van de vector kan als één data-object gebruikt worden. Dit wordt een *vector slice* genoemd:

```
PC(31 downto 24) <= "00000000";
PC(15 downto 12) <= PC(3 downto 0);
```

- Het aantal indexen aan de linkerkant van het toekenningssymbool moet even groot zijn als aan de rechterkant.

DE HAAGSE HOGESCHOOL

Vector samenstelllen

- Het is mogelijk om een vector samen te stellen uit (delen van) andere vectoren en constanten.

```
signal PC, int_vec: std_logic_vector (31 downto 0);
signal int_no: std_logic_vector (2 downto 0);
signal shiftreg: std_logic_vector (7 downto 0);

int_vec  <= PC(31 downto 6) & int_no & "000";
```

- Schuiven is eenvoudig.

```
shiftreg <= shiftreg(6 downto 0) & '0'; -- shift left
shiftreg <= '0' & shiftreg(7 downto 1); -- shift right
```

DE HAAGSE HOGESCHOOL

Concurrent assignment

- Hardware is van nature concurrent.
- Dit is te beschrijven met toekenninsopdrachten (assignments).
- Er zijn drie mogelijkheden

  Simple assignment – reeds behandeld
  Conditional Signal Assignment
  Selected Signal Assignment

Conditional Signal Assignment

```
library ieee;
use ieee.std_logic_1164.all;

entity prior is
    port (a,b,c,d,en,data,reset: in std_logic;
          s : out std_logic);
end entity prior;

architecture behav of prior is
begin
    s <= a  when en = '1' else
         b  when data = '0' else
         c  when reset = '1' else
         d;
end architecture behav;
```

eerste conditie die waar is wordt uitgevoerd

volgorde is belangrijk

waarom deze laatste toekenning?

Conditional Signal Assignment

- Het beschrijven van een “groter dan”-schakeling gaat in VHDL heel eenvoudig.

```
library ieee;
use ieee.std_logic_1164.all;

entity agtb is
   port (a, b : in std_logic_vector (3 downto 0);
         s    : out std_logic);
end entity agtb;

architecture behav of agtb is
begin
   s <= '1' when a > b else '0';
end architecture behav;
```

Selected Signal Assignment

```
library ieee;
use ieee.std_logic_1164.all;

entity mux4 is
    port (in0,in1,in2,in3: in std_logic;
          sel: in std_logic_vector (1 downto 0);
          z : out std_logic);
end entity mux4;

architecture ssa of mux4 is
begin
    with sel select
    z <= in0  when "00",
         in1  when "01",
         in2  when "10",
         in3  when "11",
         'X'  when others;
end architecture ssa;
```

volgorde is niet belangrijk

waarom deze laatste toekenning?

Waarheidstabel

```
library ieee;
use ieee.std_logic_1164.all;

entity full_adder_ssa is
    port (a_b_cin: in std_logic_vector (2 downto 0);
          cout_s : out std_logic_vector (1 downto 0));
end entity full_adder_ssa;

architecture truth_table of full_adder_ssa is
begin
    with a_b_cin select
    cout_s <= "00" when "000",
              "01" when "001",
              "01" when "010",
              "10" when "011",
              "01" when "100",
              "10" when "101",
              "10" when "110",
              "11" when "111",
              "XX" when others;
end architecture truth_table;
```

DE HAAGSE HOGESCHOOL

Opgaven

- Geef de logische functies van:

```
s <= '1' when a = '0' else
     '1' when b = '1' else
     '1' when c = '0' else
     '0';

s <= a when data = '0' else
     b when data = '1' and en = '0' else
     c when en = '0' else
     d;
```

Opgaven

- Geef een VHDL-beschrijving van een EXOR d.m.v. een Conditional Signal Assignment en een Selected Signal Assignment.
- Geef de VHDL-statements om een 8-bits vector één plek naar rechts te schuiven.
- Geef de VHDL-statements om een 8-bits vector één plek naar links te roteren, waarbij het meest significante bit in het minst significante bit wordt geplaatst.
- Geef de volledige VHDL-beschrijving van een SR-latch met overheersende set.

Referenties

- VHDL Language Manual – http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4772740

- Y-chart – https://en.wikipedia.org/wiki/Gajski-Kuhn_chart
- Y-chart in \LaTeX – http://www.texample.net/tikz/examples/gajski-kuhn-y-chart/

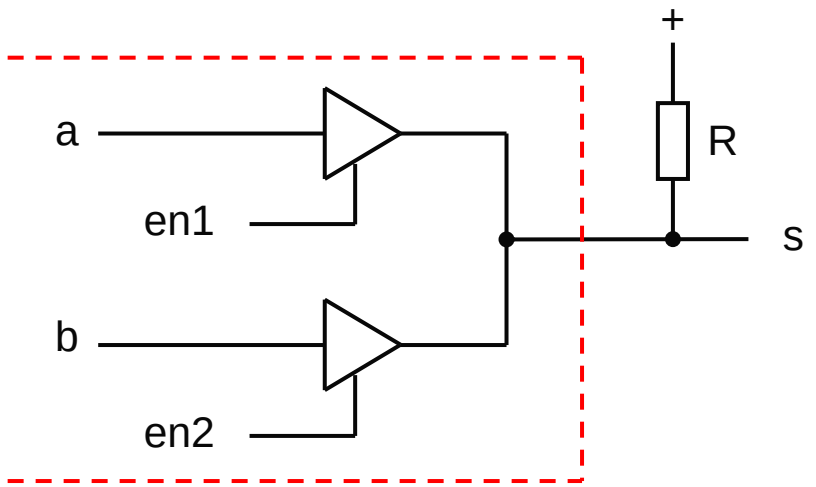
- Floating point bibliotheek – https://github.com/FPHDL/fphdl/blob/master/Float_ug.pdf

std_logic

- Het is mogelijk om een *tri-state buffer* te beschrijven:

```
signal a, b, en1, en2: std_logic;
begin
    s <= a when en1 = '1' else 'Z';
    s <= b when en2 = '1' else 'Z';
end;
```

Als en1= 1 óf en2 = 1 dan wordt signaal a óf b doorgegeven (buffer), als en1 = 0 én en2 = 0 wordt de uitgang in High-Z gezet en is fysiek van de bus ontkoppeld. De *pull-up* weerstand levert in dat geval een logische 1. en1 en en2 mogen niet tegelijk 1 zijn.

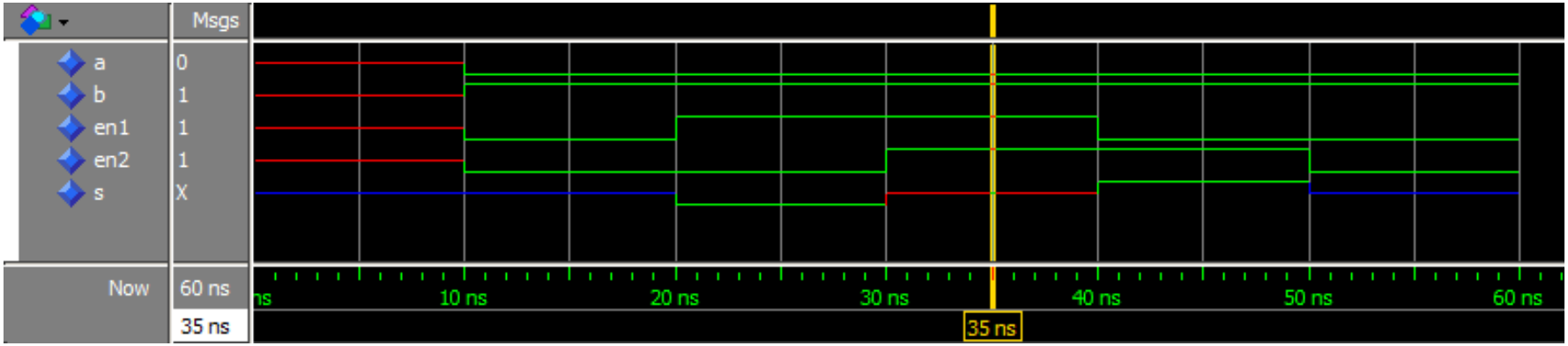


std_logic

- Met gebruik van `std_ulogic` levert de simulatie een fout op:

```
Error: D:/PROJECTS/QUARTUS/vhdl_tristate_example/vhdl_tristate_example.vhd(25):
Nonresolved signal 's' has multiple sources.
```

- Met gebruik van `std_logic` lukt de simulatie wel:



Academie voor Technology, Innovation & Society Delft

Academie voor ICT & Media

De Haagse Hogeschool, Delft
+31-15-2606311
J.E.J.opdenBrouw@hhs.nl
www.dehaagsehogeschool.nl

DE HAAGSE HOGESCHOOL