- Introduction to the VHDL Hardware description language

1. Introduction
2. Basic elements
3. Scalar data types
4. Composed data types
5. Basic constructs (system definition)
6. Data flow description level
7. Structural description level
8. Behavioral description level
9. Design organisation
1. Introduction

- VHSIC Project (DoD, USA)
- Initial goal: facilitate documentation
- Developed by TI, Intermetrics, IBM (1985)
- Strongly linked to data types (ADA)
- Basic description levels:
  - data flow, structural, behavioral
- Basic objects:
  - constants, variables, signals and files
2. Basic elements

- **Comments:** --
  -- That’s a comment

- **Identifiers (labels):**
  - alphabetic characters + digits + _
  - start: *always* with alphabetic character
  - *can not* end with _
  - *can not* contain two successive _

- **Characters:** ‘C’
- **Strings:** “OnlY a lINe”
- **Numbers (integer or real):** 23 45.2E-4 16#A4#E-7
- **Bit strings:** [B|O|X]”101”
- **Constants:**
  ```vhdl```
  ```constant n_bits_per_word: integer:=12;
  constant pi: real:=3.1415926535
  ```
- **Variables:**
  ```vhdl```
  ```variable counter: integer:=0;```
- **End of sentence:** ;
- **Assignments:** <= (signals) := (variables and constants)

- **Assert primitive:** Condition monitoring
  ```vhdl```
  ```assert vdd/=5 report “VDD not connected”
  severity warning;```
3. Scalar data types

- **Type declaration:**
  ```vhdl
  type mean_weight is 65 to 80;
  type logic_level is ('0', '1', 'Z', 'X');
  ```

- **Integer type:**
  ```vhdl
  type word_type is (4, 8, 16, 32, 64);
  variable opcode: word_type:=4;
  ```

- **Floating point type:**
  ```vhdl
  type probability is 0.0 to 1.0;
  ```

- **Physical type:**
  ```vhdl
  type resistance is range 0 to 1E9
  units
    ohm;
    kohm=1000 ohm;
    Mohm = 1000 kohm
  end units resistance;
  ```
- **Time type:** Default time resolution: $fs$
- **Character type:** 8-bits ISO character set
- **Boolean type:**
  ```vhdl```
  ```
  type boolean is (false, true);
  ```
- **Bit type:**
  ```vhdl```
  ```
  type bit is (‘0’, ‘1’);
  ```
- **Subtype declaration:**
  ```vhdl```
  ```
  subtype pointer is integer range 15 downto 0;
  ```
- **Conversion between types:** real(84) integer(36.9);
- **Operators:**
  ```vhdl```
  ```
  ** abs not
  * / mod rem
  + - &
  sll srl sla sra rol ror
  = /= < <= > >=
  and or nan nor xor xnor
  ```
  ```
  Precedence
  ```
4. Composed data types

- **Array:**

  ```vhdl
  type word is (0 to 15) of bit;
  variable register_A: word:=X"A42E";
  carry := register_A(4);
  
  type curve_points is (0 to 3) of integer;
  variable curve_1: curve_points:=
  (1=>0, 2=>5, others =>0);
  
  type RAM is (natural range <>) of bit_vector(7 downto 0);
  variable control_ram: ram(0 to 1024);
  
  type bidim_array is (0 to 3, 0 to 7) of bit;
  ```
➤ **Record:** Fields with **different types**

```vhdl
-- type instruction is record
  opcode: bit_vector(7 downto 0);
  source_reg: bit_vector(3 downto 0);
  destination_reg: bit_vector(3 downto 0);
end record instruction;

variable instruction_reg: instruction;
```

```vhdl
instruction_reg.opcode:=data_cpu(7 downto 0);
instruction_reg.source_reg:=data_cpu(11 downto 8);
instruction_reg.destination_reg:=data_cpu(15 downto 12);
```
### Attributes: Specify properties of an element

#### Scalar attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type of T</th>
<th>Type of result</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T’left</td>
<td>any scalar type or subtype</td>
<td>same as T</td>
<td>leftmost value in T</td>
</tr>
<tr>
<td>T’right</td>
<td>“</td>
<td>“</td>
<td>rightmost value in T</td>
</tr>
<tr>
<td>T’low</td>
<td>“</td>
<td>“</td>
<td>lowest value in T</td>
</tr>
<tr>
<td>T’high</td>
<td>“</td>
<td>“</td>
<td>highest value in T</td>
</tr>
<tr>
<td>T’ascending</td>
<td>“</td>
<td>boolean</td>
<td>true if T is an ascending range, false otherwise</td>
</tr>
<tr>
<td>T’image(x)</td>
<td>“</td>
<td>string</td>
<td>textual representation of the value x from type T</td>
</tr>
<tr>
<td>T'value(s)</td>
<td>“</td>
<td>base type of T</td>
<td>value in T represented by the string s</td>
</tr>
<tr>
<td>T’pos(x)</td>
<td>any discrete type or subtype</td>
<td>universal integer</td>
<td>position of x in T</td>
</tr>
<tr>
<td>T’val(x)</td>
<td>“</td>
<td>base type of T</td>
<td>value at position x in T</td>
</tr>
</tbody>
</table>
## Scalar attributes (cont.)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type of T</th>
<th>Type of result</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T’succ(x)</td>
<td>any discrete type or subtype</td>
<td>base type of T</td>
<td>value in T at position one greater than that of x</td>
</tr>
<tr>
<td>T’pred(x)</td>
<td></td>
<td>“</td>
<td>value in T at position one less than that of x</td>
</tr>
<tr>
<td>T’leftof(x)</td>
<td></td>
<td>“</td>
<td>value in T at position one to the left of x</td>
</tr>
<tr>
<td>T’rightof(x)</td>
<td>any discrete or physical type or subtype</td>
<td>“</td>
<td>value in T at position one to the right of x</td>
</tr>
<tr>
<td>T’base</td>
<td>any type or subtype</td>
<td>“</td>
<td>base type of type T, only allowed as a prefix of another attribute</td>
</tr>
</tbody>
</table>
## Array attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A’left(n)</td>
<td>left bound of index range of dimension n of A</td>
</tr>
<tr>
<td>A’right(n)</td>
<td>right bound of index range of dimension n of A</td>
</tr>
<tr>
<td>A’low(n)</td>
<td>lower bound of index range of dimension n of A</td>
</tr>
<tr>
<td>A’high(n)</td>
<td>upper bound of index range of dimension n of A</td>
</tr>
<tr>
<td>A’range(n)</td>
<td>index range of dimension n of A</td>
</tr>
<tr>
<td>A’reverse_range(n)</td>
<td>reverse of index range of dimension n of A</td>
</tr>
<tr>
<td>A’length(n)</td>
<td>length of index range of dimension n of A</td>
</tr>
<tr>
<td>A’ascending(n)</td>
<td>true if index range of dimension n of A is ascending, false otherwise</td>
</tr>
</tbody>
</table>
### Signal attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type of result</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>S’delayed(t)</td>
<td>base type of S</td>
<td>a signal that takes on the same value as S but is delayed by time T</td>
</tr>
<tr>
<td>S’stable(t)</td>
<td>boolean</td>
<td>a boolean signal that is true if there has been no event on S in the time interval t up to the current time, otherwise false</td>
</tr>
<tr>
<td>S’quiet(t)</td>
<td>boolean</td>
<td>a boolean signal that is true if there has been no transaction on S in the time interval t up to the current time, otherwise false</td>
</tr>
<tr>
<td>S’transaction</td>
<td>bit</td>
<td>implicit bit signal, which changes its value each time there is a transaction on S</td>
</tr>
<tr>
<td>S’event</td>
<td>boolean</td>
<td>true if there is a transaction on S in the current simulation cycle, otherwise false</td>
</tr>
<tr>
<td>S’active</td>
<td>boolean</td>
<td>true if there is a transaction on S in the current simulation cycle, otherwise false</td>
</tr>
<tr>
<td>S’last_event</td>
<td>time</td>
<td>time interval since the last event on S</td>
</tr>
<tr>
<td>S’last_active</td>
<td>time</td>
<td>time interval since the last transaction on S</td>
</tr>
<tr>
<td>S’last_value</td>
<td>base type of S</td>
<td>value of S before the last event</td>
</tr>
<tr>
<td>S’driving</td>
<td>boolean</td>
<td>true if the current process is producing a transaction on S</td>
</tr>
<tr>
<td>S’driving value</td>
<td>base type of S</td>
<td>value assigned to S in the current process</td>
</tr>
</tbody>
</table>

4. Composed data types
Example:

```vhdl
function increment (val:bit_vector) return bit_vector is
variable result: bit_vector(val'range);
variable carry: bit;
begin
    result(0):= not val(0);
    carry:= val(0);
    for i in 1 to val'high loop
        carry:= carry and val(i-1);
        result(i):= val(i) xor carry;
    end loop;
    return result;
end increment;
```
5. Basic constructs

- System = entity

**Entity body**
- Name
- Definition of I/O interface

**Architecture(s)**
- Declaration of subsystems (components)
- Declaration of signals
- Definition of functionality
Example:

```vhdl
entity adder_1bit is
  port(
    a: in bit;
    b: in bit;
    carry_in: in bit;
    result: out bit;
    carry_out: out bit
  );
end adder_1bit;

architecture dataflow of adder_1bit is
begin
  suma <= a xor b xor carry_entrada;
  carry_out <= a and b or ((a or b) or carry_in);
end dataflow;
```
• Architecture description levels:

- **Data flow (declarative):**
  - Assignments
  - Blocks

- **Structural (applicative semantics):**
  - Components
  - Generate primitive

- **Behavioral (procedural):**
  - Processes
  - Procedures and functions
6. Data flow description level

• Timing:

Response time (inertial delay)

Propagation time (transport delay)
• Inertial delay:

Low-pass filter behavior
• **Transport delay:**

\[
t_t = (R_o + R_w) \cdot \left( C_w + \sum_{j=1}^{n} C_j \right)
\]

**Infinite bandwidth behavior**
• Unconditional assignment:

\[ A\_signal \leftarrow \text{value1 after delay1, value2 after delay2, \ldots, valuen after delayn;} \]

• Delay types: inertial and transport

- transport: infinite frequency (propagation of current)

\[
\text{t_line: process (line_in)}
\begin{align*}
\text{begin} \\
\text{line_out} \leftarrow \text{transport line_in after 500 ps;}
\text{end process t_line;}
\end{align*}
\]

6. Data flow description level
asym_delay: process (a)
constant Tpd_01: time:=800 ps;
constant Tpd_10: time:=500 ps;
begin
  if a='1' then
    z<= transport a after Tpd_01;
  else
    z<= transport a after Tpd_10;
  end if;
end process asym_delay;
- inertial: default delay type

inverter: process (a)
begin
y <= (inertial) not a after 3 ns;
end process inverter;

S <= 1 after 5 ns, 5 after 10 ns, 10 after 15 ns;

S <= 1 after 5 ns;
S <= transport 5 after 10 ns;
S <= transport 10 after 15 ns;

6. Data flow description level
Conditional assignment:

\[
A\_signal \leftarrow \begin{array}{l}
\text{value1 when condition1} \\
\text{else value2 when condition2} \\
\vdots \\
\text{else default\_value};
\end{array}
\]

Selective assignment:

\[
\text{with expression select} \\
A\_signal \leftarrow \begin{array}{l}
\text{value1 when selection1,} \\
\text{value2 when selection2,} \\
\vdots \\
\text{default\_value when others;}
\end{array}
\]
• Signal resolution:

  ➢ Definition of *resolution functions* which determine the value of a net when it is connected to different drivers (active or not)

• Guarded signals: Definition of a value when drivers are disconnected (*null* assignment)

• **Bus**: Empty array (*pull-up*)
• **Register**: Last value (*dynamic memory*)

```
signal overflow: pullup_type bus;
signal load_1: bit register;
```
• **IEEE Standard 1164:** 9-value logic

  - ‘U’: Not initialised
  - ‘X’: Unknown (strong driver)
  - ‘0’: Low level (strong driver)
  - ‘1’: High level (strong driver)
  - ‘Z’: High impedance
  - ‘W’: Unknown (weak driver)
  - ‘L’: Low level (weak driver)
  - ‘H’: High level (weak driver)
  - ‘-’: Don’t care

• **Use of the 1164 data types:**

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

• **Basic types:**

  - `std_logic` ( = bit)
  - `std_logic_vector` ( = bit_vector)
• Blocks:
  - Functional subsystem
  - **Concurrent** execution
  - Interconnection through ports
  - Definition of guards

```vhdl
Rising_edge: block (clock'event and clock='1')
begin
  output <= guarded input;
  q <= d;
end block;
```

- **condition or guard**
- **unconditional assignment**
- **conditional assignment**
7. Structural description level

• Components:
  ➢ Subsystem defined in a library
  ➢ Component declaration:
    ```vhdl
cOMPONENT nand3
    GENERIC(Tpd: TIME:= 1ns);
    PORT(a,b,c: IN bit; f: OUT bit);
    END COMPONENT;
    ```
  ➢ Component instantiation:
    ```vhdl
    enable: nand3
    PORT MAP(en1, en2, int_req, interrupt);
    ```
### Example:

**entity** basic_gate is  
**port**(a,b,c: in std_logic; f: out std_logic);  
**end** basic_gate;

**architecture** structural of basic_gate is  
**component** nand2  
**port**(a,b: in std_logic; f: out std_logic);  
**end component**;

**signal** aux: std_logic;

**begin**

  n_1: nand2  
  **port map**(a=>a,b=>b,f=>aux);

  n_2: nand2  
  **port map**(a=>aux,b=>c,f=>f);

**end** structural;
• Definition of regular structures:

• Generate primitive:

adder: for i in 0 to k-1
  ls_bit: if i=0 generate
    ls_cell: half_adder port map(a(0), b(0), s(0), c_in(1));
  end generate ls_bit;
  middle_bit: if i > 0 and i < k-1 generate
    mid_cell: full_adder port map(a(i), b(i), c_in(i), s(i), c_in(i+1));
  end generate middle_bit;
  ms_bit: if i=k-1 generate
    ms_cell: full_adder port map(a(i), b(i), c_in(i), s(i), carry);
  end generate ms_bit;
end generate adder;
8. Behavioral description level

- Processes:
  - Sequential execution body
  - Activated by signal transitions
  - Concurrent execution

- Activation control:

**Sensitivity list**

```vhdl
process(reset, clock)
variable state: bit:= false;
begin
  if reset then
    state:=false;
  elsif clock=true then
    state:= not state;
  end if;
  q<= state after delay;
end process;
```

**Wait primitive**

```vhdl
process
variable state: bit:=false;
begin
  wait until (clock’event or reset’event);
  if reset then
    state:= false;
  elsif clock=true then
    state:= not state;
  end if;
  q<= state after delay;
end process;
```
• Primitives for sequential control:

➢ Wait:

```vhdl
wait [until condition;]
[on signal1, signal2, ...;]
[for temporal_expression;]
```

➢ If:

```vhdl
if [condition] then [actions] elsif [actions] ... else [actions] ... end if;
```

➢ Case:

```vhdl
case [expression] is
  when [selection] => [actions];
  ... 
end case;
```
- **Indefinite loop:**
  
  ```vhdl
  loop [actions] end loop;
  ```

- **While loop:**

  ```vhdl
  while [condition] loop
      [actions];
  end loop;
  ```

- **For loop:**

  ```vhdl
  for [identifier] in [range] loop
      [actions];
  end loop;
  ```

- **Loop control: next, exit**

  ```vhdl
  [when condition]
  ```
Example:

```vhdl
architecture behavior of counter is
begin
  increment: process variable value: integer:=0;
  begin
    output <= value;
    loop
      loop
        wait until clk='1' or reset='1';
        exit when reset='1';
        value:= (value+1) mod 16;
        output <= value;
      end loop;
      value:=0;
      output <= value;
      wait until reset='0';
    end loop;
  end process increment;
end behavior;
```
Procedures: May return a value

```vhdl
procedure mean is
variable partial: real := 0.0;
begin
    for index in matrix'range loop
        partial := partial + matrix(index);
    end loop;
    mean_value := partial / real(matrix'length);
end procedure mean;
```

Functions: Always return a value
9. Design organisation

- **Library**: Object (directory, file, ...) which contains already designed modules

- **Definition of a reference library**:
  
  ```vhdl
  library library_name;
  ```

- **Use of elements included in a library**:
  
  ```vhdl
  use library_name.defined_element;
  ```
• Configuration of objects (binding):

```vhdl
library basic_flip_flop;
use basic_flip_flop.ff_d_rising_edge;

configuration reg4_structural of reg4 is
  for structural
    for bit0: flipflop
      use entity ff_d_rising_edge(drive4);
    end for;
    for others: flipflop
      use entity ff_d_rising_edge(drive1);
    end for;
  end for;
end configuration reg4_structural;
```
• **Encapsulated declarations (packages)**

- declaration + body definition
- reference: `variable PC: data_types.address;`
- element use: `use data_types.all;`

• **Simulation:**

> **Test Bench:** VHDL model where stimuli for a given system are declared (and/or its outputs are verified)
• Organisation of the simulation process

**Analysis**
Syntax and semantics check

**Elaboration**
Hierarchical expansion of the system (signals and processes)

**Execution**
Initialisation + simulation cycle
• Simulation cycle

1) Time is advanced until the next time point where transactions on signals are to be performed
2) Transactions on signals are performed
3) The processes which are sensitive to the events produced are activated and executed

process ...
begin
...
s <= '1';
...
if s='1' then ...
...
end process;