VHDL

P. Marwedel
Informatik 12, U. Dortmund
Gliederung

Einführung
SystemC
  - Vorlesungen und Programmierung
FPGAs
  - Vorlesungen
  - VHDL-basierte Konfiguration von FPGAs mit dem XUP VII Pro Entwicklungssystem
Algorithmen
  - Mikroarchitektur-Synthese
  - Automatensynthese
  - Logiksynthese
  - Layoutsynthese

Zeitplan

- 3,5 Wochen
- 3,5 Wochen
- 6 Wochen
HDL = hardware description language
Textual HDLs replaced graphical HDLs in the 1980‘ies (better description of complex behavior).
In this course:
VHDL = VHSIC hardware description language
VHSIC = very high speed integrated circuit
1980: Definition started by DoD in 1980
1984: first version of the language defined, based on ADA (which in turn is based on PASCAL)
1987: revised version became IEEE standard 1076
1992: revised IEEE standard
more recently: VHDL-AMS: includes analog modeling
Entities and architectures

Each design unit is called an **entity**. Entities are comprised of **entity declarations** and one or several **architectures**.

Each architecture includes a model of the entity. By default, the most recently analyzed architecture is used. The use of another architecture can be requested in a **configuration**.
The full adder as an example
- Entity declaration -

Entity declaration:

```vhdl
entity full_adder is
  port(a, b, carry_in: in Bit;  -- input ports
       sum, carry_out: out Bit); -- output ports
end full_adder;
```
The full adder as an example
- Architectures -

Architecture = Architecture header + architectural bodies

```plaintext
architecture behavior of full_adder is
begin
  sum   <= (a xor b) xor carry_in after 10 Ns;
  carry_out <= (a and b) or (a and carry_in) or
               (b and carry_in) after 10 Ns;
end behavior;
```

Architectural bodies can be
- behavioral bodies or - structural bodies.
Bodies not referring to hardware components are called behavioral bodies.
architecture structure of full_adder is

  component half_adder
    port (in1,in2:in Bit; carry:out Bit; sum:out Bit);
  end component;
  component or_gate
    port (in1, in2:in Bit; o:out Bit);
  end component;

signal x, y, z: Bit;   -- local signals

begin
  -- port map section
  i1: half_adder port map (a, b, x, y);
  i2: half_adder port map (y, carry_in, z, sum);
  i3: or_gate port map (x, z, carry_out);
end structure;
Architectural bodies

**Syntax:**

```
architecture body_name of entity_name is
declarations  -- no variables allowed for pre-1992 VHDL
begin
  statements;
end body_name;
```

Let's look at declarations first!
Lexical Elements

1. **Identifiers**
   Legal characters: letters, digits, underscores,
   1st character must be a letter,
   No two adjacent underscores,
   Not case-sensitive.
   **Examples:**
   A_b, what_a_strange_identifier

2. **Comments**
   Comments start with 2 adjacent hyphens and continue
   until the end of the line
   **Example:**
   a := 4712;  - - this is no commercial advertisement
1. **Integer literals**
   sequences of digits, may include positive base 10 exponent, underscores to improve readability
   Examples:
   2, 22E3, 23_456_789
   2#1010101# - - base 2 integer literal

2. **Floating point literals**
   Examples:
   1.0
   33.6e-1

3. **Character sequence literals**
   sequences of characters enclosed in double quotes
   "ABC", "011"
1. **Bit vector literals**
   sequences of characters with constrained character set, underscores for readability, explicit base possible
   
   Examples:
   
   X"F07CB"  - - base 16
   B"1111_1111"  - - base 2
   "11111111"  - - equivalent to previous literal

2. **Physical literals**
   
   Examples:
   
   35 Ns  - - some time
   33.56 Ws  - - energy using floating point number
Objects, container types

3 Types of objects

1. Constants
   Single assignment during elaboration

2. Signals
   Modeling wires, values are assigned with some delay

3. Variables
   equivalent to variables in other programming languages
   Immediate assignment.
   Must be local to processes in pre-1992 VHDL.

Examples:

```vhdl
constant pi : Real := 3.14;
variable mem : some_name;
signal s,t,u : Bit := '0';  -- initial value
```
Scalar Types (1)

1. **Enumeration types**
   Ordered set of identifiers or characters
   Examples:
   
   type Bit is ('0','1');
   type Boolean is (False, True);

2. **Integer types**
   Integer numbers within implementation dependent range and subranges of these
   Examples:
   
   type byte_int is range 0 to 255;
   type bit_index is range 31 downto 0;
Scalar Types (2)

1. **Physical types**
   value = number + unit

   **Examples:**
   ```pascal
   type Time is range -2147483647 to 2147483647 units
       Fs; -- femtosecond
       Ps = 1000 Fs;
       Ns = 1000 Ps; ...
   end units;
   ```

2. **Floating point types**

   **Example:**
   ```pascal
   type probability is range 0.0 to 1.0;
   ```
Composite Types: Arrays

**Example:**

```pascal
type word is array (15 downto 0) of Bit;
```

Unconstrained arrays = arrays with templated size.

**Examples:**

```pascal
type String is array (Positive range <>) of Character;
type Bit_Vector is array (Natural range<> ) of Bit;
...
signal vector: Bit_Vector (0 to 15); - -fix size@variable decl.
```
Composite Types: Structures

Example:

```haskell
 type register_bank is record
   F0, F1: Real;
   R0, R1: Integer;
   A0, A1: Address;
end record;
```
**Aliasing**

Parts of objects can be associated with their own name.

**Example:**
Instruction register ir

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31..26</td>
<td>Opcode</td>
</tr>
<tr>
<td>25..21</td>
<td>register number</td>
</tr>
<tr>
<td>20..0</td>
<td>offset</td>
</tr>
</tbody>
</table>

**In VHDL:**

```vhdl
alias ir_opcode: Bit_Vector( 5 downto 0) is ir(31 downto 26);
alias ir_reg: Bit_Vector( 4 downto 0) is ir(25 downto 21);
alias ir_dist: Bit_Vector(20 downto 0) is ir(20 downto 0);
```
Functions

Syntax of functions and procedures is close to ADA

Examples:

```plaintext
function nat(a: in bit_vector)
    return integer is
    variable res: integer:=0;
    variable factor : integer :=1;
    begin
        for i in a'low to a'high loop
            if a(i)='1' then res:=res+factor;
        end if;
        factor:=factor*2;
    end loop;
    return res;
end nat;
```

```plaintext
function mcolor (z: in states)
    return color is
    begin
        statements;
        return value;
    end mcolor;
```

Resolved signals

Resolution functions can be invoked by proper subtype definitions.

**Example:**

```plaintext
let std_logic: subtype derived from type std_ulogic by calling resolution function resolved.
Can be achieved with:

```plaintext
subtype std_logic is resolved std_ulogic;
resolved called with array of element type std_ulogic.
Length of the array = #(signals) driving a line.
```
Component declarations provide the "signatures" for components to be instantiated in the body of an architecture.

**Example:**
```vhdl
component and_gate
  port (in1,in2: in Bit);
  result : out Bit);
end component;
```
Attributes are properties of some element

Syntax for using attributes:

`elementname'attributename` - - read "element-tick-attribute"

Declaration of attributes:

```plaintext
attribute Cost : Integer ;
```

Definition of attributes:

```plaintext
attribute Cost alu: entity is 22;
```

Elements that can have attributes:

- types and subtypes, procedures and functions
- signals, variables and constants;
- entities, architectures, configurations, packages, components, labels
Predefined Attributes

F'left(i): left index bound, \( i^{th} \) dimension of array \( F \)
F'right(i): right index bound, \( i^{th} \) dimension of array \( F \)
F'high(i): upper index bound, \( i^{th} \) dimension of array \( F \)
F'low(i): lower index bound, \( i^{th} \) dimension of array \( F \)
S'event: event at signal \( S \) in last cycle
S'stable: no event for signal \( S \) in last cycle

Application:

if (s'event and (s='1')) - - rising edge
if (not s'stable and (s='1')) - - rising edge

Let's look at statements next!
Component instantiations

Components can be instantiated in the body of an architecture.

**Example:**

Assume a signal declaration

```plaintext
signal a, b, c : Bit;
```

Then, in the body, we may have:

```plaintext
and1: and_gate(a, b, c);
```

Signal association may also be by name.

**Example:**

```plaintext
and1: and_gate(result => c, in1 => a, in2 => b);
```
VHDL processes

Processes model parallelism in hardware.

General syntax:

```
label:       --optional
process
  declarations --optional
begin
  statements   --optional
end process;
```

```
a <= b after 10 ns is equivalent to
process
begin
  a <= b after 10 ns
end;
```
Assignments

2 kinds of assignments:

- **Variable assignments**
  Syntax: `variable := expression;`

- **Signal assignments**
  Syntax:
  
  ```
  signal <= expression;
  signal <= expression after delay;
  signal <= transport expression after delay;
  signal <= reject time inertial expression after delay;
  ```

Possibly several assignments to 1 signal within 1 process.

For each signal there is one **driver** per process. Driver stores information about the **future** of signal, the so-called **projected waveform**.
## Predefined operators (1)

<table>
<thead>
<tr>
<th>Group</th>
<th>Symbol</th>
<th>Semantics</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>+</td>
<td>Addition</td>
<td>Integer, floating point, physical types</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Subtraction</td>
<td>Integer, floating point, physical types</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Multiplication</td>
<td>Integer, floating point, physical types</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>Division</td>
<td>Integer, floating point, physical types</td>
</tr>
<tr>
<td></td>
<td>mod</td>
<td>Modulo</td>
<td>Integer, floating point, one argument physical;</td>
</tr>
<tr>
<td></td>
<td>rem</td>
<td>Remainder</td>
<td>Integer, floating point, one argument physical;</td>
</tr>
<tr>
<td></td>
<td>**</td>
<td>Exponentiation</td>
<td>Integer, floating point, one argument physical;</td>
</tr>
<tr>
<td>arithmetic</td>
<td>+</td>
<td></td>
<td>Integer, floating point, one argument physical;</td>
</tr>
<tr>
<td>(unary)</td>
<td>-</td>
<td></td>
<td>Integer, floating point, one argument physical;</td>
</tr>
<tr>
<td></td>
<td>abs</td>
<td></td>
<td>Integer, floating point, one argument physical;</td>
</tr>
</tbody>
</table>

**No bitvectors, except if library is present**
## Predefined operators (2)

<table>
<thead>
<tr>
<th>group</th>
<th>Symbol</th>
<th>Semantics</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>logic (binary)</td>
<td>and</td>
<td></td>
<td>Bit, Boolean, 1-dimensional arrays of these</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>nand</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>nor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>logic (binary)</td>
<td>not</td>
<td>Complement</td>
<td></td>
</tr>
<tr>
<td>comparison</td>
<td>=</td>
<td>equal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/=</td>
<td>not equal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;, &lt;=</td>
<td>less (than)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;, &gt;=</td>
<td>greater (than)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>concatenation</td>
<td>1-dim. array</td>
</tr>
</tbody>
</table>
Wait-statements

Four possible kinds of **wait**-statements:

- **wait on signal list;**
  wait until signal changes;
  Example: **wait on** a;

- **wait until condition;**
  wait until condition is met;
  Example: **wait until** c='1';

- **wait for duration;**
  wait for specified amount of time;
  Example: **wait for** 10 ns;

- **wait;**
  suspend indefinitely
Sensivity lists are a shorthand for a single \texttt{wait on}-statement at the end of the process body:

\begin{verbatim}
process (x, y)
  begin
    prod <= x \texttt{and} y ;
  end process;
\end{verbatim}

is equivalent to

\begin{verbatim}
process
  begin
    prod <= x \texttt{and} y ;
    wait on x,y;
  end process;
\end{verbatim}

No local wait statements allowed!
IF-statements

If-Statements are always terminated with end if.
Nested if-statements use elsif

Example:
if a=3 then b:=z; d:=e
elsif a=5 then b:=z
else b:=e
end if;
Case-Statements are always terminated with `end case`. Selecting value is enclosed within `when` and `=>`. The default case is denoted by `others`.

Example:
```
case opcode is
  when 1 => result <= a + b;
  when 2 => result <= a - b;
  when others => result <= b;
end case;
```
Loops based on loop constructs with optional extensions. Labels are optional. next and exit used to change control.

Examples:

label: loop statements; exit when condition; next label when condition; statements; end loop label;

m: while condition loop statements; exit when condition; next label when condition; statements; end loop m;

m: for i in 0 to 10 loop statements; exit when condition; next label when condition; statements; end loop m;
Global view on model descriptions
Packages

Frequently used procedures and functions can be stored in **packages**, consisting of a declaration and a body:

```
package declaration
package body
```

**Syntax:**

```
package package_name is
declarations
end package_name;
package body package_name is
definitions
end package_name;
```
Declaration includes types, constants, signals, function, procedures and components (for variables see VHDL'92). Definition includes implementations of procedures/functions.

Example:
package math is
  function nat(a: in bit_vector) return integer;
  ...
end math;
package body math is
  implementation of nat;
  ...
end math;
Configurations

Configurations denote entities & architectures to be used. Default: most recently analyzed architectural body is used.

**Syntax:**
configuration cfg of module is
declarations;
configuration;
end cfg;

**Example:**
configuration example of test is
for structure
for comp1:generator
use entity bib.generator1;
end for;
for comp2:trafficlight
use entity bib.circuit(two);
end for;
end example;
Design units

A design unit is a segment of an VHDL description that can be independently analyzed (apart from references to other design units).

Two kinds of design units:
- Primary design units
  - Entity declaration
  - Package declaration
  - Configuration declaration
- Secondary design units
  - Architectural body
  - Package body
Libraries

Design units are stored in libraries.

Rules:

- Within a library, each primary design unit must have a unique name.
- There may be several secondary design units of the same name with one library.
- Primary and related secondary design units must be stored in the same library.
Visibility of design units

Design units can be made visible with **use** statements (applies to next design unit).

**Forms:**
- `use bibname.unitname;`
- `use unitname.object;`
- `use bibname.unitname.object;`
- `use bibname.unitname.all;`

Object **all** denotes all visible objects within a unit. Work denotes the library into which analyzed design units are stored.
Visibility of libraries

Libraries are made visible with `library` statements. Example:

```verbatim
library bib;
use bib.lights.all;
```

Implicitly, each design unit is preceded by

```verbatim
library work;
library std;
use std.standard.all;
```
Example

State machine with 2 states and one output

begin
  exec : process
  begin
    wait until CLOCK'event and CLOCK='1';
    case Chart is
      when S1 =>
        if nat(INP) = 0 then
          OUTP <= '0';
          Chart <= S1;
        elsif nat(INP) = 1 then
          OUTP <= '1';
          Chart <= S2;
        end if;
      when S2 =>
        OUTP <= not INP;
        Chart <= S2;
    end case;
  end process
end Arch_ENT2;

definition of a state machine with two states and one output:

library WORK;
use WORK.SYN.all; --some
use WORK.STC.all; --stuff
entity ENT2 is port
  (CLOCK: in bit;
   INP:    in bit;
   OUTP:   out bit);
end ENT2;
architecture Arch_ENT2 of ENT2 is
  type tpChart is (S1, S2);
signal Chart: tpChart;
begin
  exec : process
  begin
    wait until CLOCK'event and CLOCK='1';
    case Chart is
      when S1 =>
        if nat(INP) = 0 then
          OUTP <= '0';
          Chart <= S1;
        elsif nat(INP) = 1 then
          OUTP <= '1';
          Chart <= S2;
        end if;
      when S2 =>
        OUTP <= not INP;
        Chart <= S2;
    end case;
  end process
end Arch_ENT2;
Summary

VHDL:

- Entities and (behavioral/structural) architectures
- Declarations
- Processes
- Wait-statement
- Packages, libraries
- Configurations
- Example