What does VHDL stand for?

- VHSIC Hardware Description Language

- VHSIC: Very High Speed Integrated Circuits
HDLs

- **VHDL**
  USA Department of Defense
  IEEE Std 1076-1993
- **Verilog**
  IEEE Std 1364-1995
- **Super Verilog**
- **SystemC**
- ...

HDL applications

- High Level Modeling (Behavioral style)
- Design Entry (Structural & RTL styles)
- Simulation (Behavioral style)
  - validation by mean of a test bench
HDL vs. Schematic Entry

- The Design Description is independent from the IC Vendors Cell Libraries (in other words independent from physical implementation)
  - Enable portability
  - Foster reuse

- Higher Level of Abstraction (hiding details)
  - The design task become simpler
  - The design is less error prone
  - Productivity is increased

HDLs vs. Software Languages

Concurrent (parallel) Statements vs. Sequential Statements
HDL coding Styles

- **Register Transfer Level**
- **Structural**
- **Behavioral**

Be careful NOT everybody gives the same meaning to the term BEHAVIORAL!

---

**RTL**

- Only a small subset of the Language statements can be mapped in real “Silicon”.

![RTL Diagram]

1. HDL code
2. **SYNTHESIS**
3. HDL code translation
4. unoptimized generic boolean netlist
5. optimization & mapping
6. area and timing constraints
7. target technology
8. optimized gate level netlist
Structural

- Sub-Modules interconnection
- Primitive cells interconnection (net-list)
- The code describes a bunch of port mappings.

Behavioral

- Modeling a system (mimic functionality and performances)
- All language constructs can be used
Levels of Abstraction

- Behavioral
- RTL
- Structural

VHDL Design Organization

- **Entity**
  the “symbol” (input/output ports)
- **Architecture**
  one of the several possible implementation of the design
- **Configuration**
  binding between the symbol and one of the many possible implementation.
  Can be used to express hierarchy.
entity mux is
port (a: in std_logic;
b: in std_logic;
s: in std_logic;
f: out std_logic)
end mux;

architecture first_rtl of mux is
begin
  mux_p: process (a,b,s)
  begin
    f <= (a and s) or (b and not s);
  end process mux_p;
end first_rtl;
Architecture #2

architecture rtl of mux is
begin
mux_p: process (a,b,s)
begin
if (s='1') then
  f <= a;
else
  f <= b;
end if;
end process mux_p;
end rtl;

Configuration

configuration mux_c of mux is
for rtl
end for;
end mux_c;
Where did we get std_logic?

- Ohps !! We need to include some library before we can use this predefined data type
  
  ```
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
  ```

Predefined data types

- **bit**: ‘0’, ‘1’
- **boolean**: false, true
- **integer**: from negative $2^{31}-1$ to positive $2^{31}-1$
std_logic, and std_ulogic

- ‘1’, ’0’, ‘X’ → logic 1, logic 0, unknown
- ‘H’, ’L’, ’W’ → weak 1, weak 0, weak unknown
- ‘U’, ’Z’, ‘-’ → uninitialized, high impedance, don’t care

resolved or unresolved?

- VHDL Driver – it is one contributor to the final value of a signal
- Drivers are created by concurrent signal assignments
- Recommendation: use std_logic, but always check that you do not have any multiple drivers (you don’t want any wired OR inside an ASIC !!!)
Bad Multiple Drivers!!!

architecture bad of mux is
begin
  -- the two assignment works in parallel
  f <= a when s = '0' else '0';
  f <= b when s = '1' else '0';
end bad;

Better way of coding the mux

architecture better of mux is
begin
  f <= a when s = '0' else 
  f <= b when s = '1' else
     'X'; -- what should the synthesis tool do here ?
end better;
One more coding for the mux

architecture even_better of mux is
begin
  f <= a when s = '0' else
  f <= b when s = '1' else
  '-' ; -- there are tools that won't appreciate all this
    -- freedom (e.g. some formal verification tool)
end even_better;

Good way of coding the mux

architecture good of mux is
begin
  f <= a when s = '0' else
  f <= b; -- here all ambiguity are gone !!!
end good;
What is a process?

- A process statement is a concurrent statement, but all statements contained in it are sequential statement (statements that execute serially, one after the other).
- The use of processes makes your code more modular, more readable, and allows you to separate combinational logic from sequential logic.

The sensitivity list

- List of all signals that the process is sensitive to. Sensitive means that a change in the value of these signals will cause the process to be invoked.
The sensitivity list must be complete !!!

```
process (a)
variable a_or_b;
begin
  a_or_b := a or b;
  z <= a_or_b;
end process;
```

-- since \( b \) is not in the
-- sensitivity list, when
-- a change occurs on \( b \)
-- the process is not
-- invoked, so the value
-- of \( z \) is not updated
-- (still “remembering” the
-- old value of \( z \))

Incomplete sensitivity list effect

![Waveform diagram](image-url)

- \( a \)
- \( b \)
- \( z \) (VHDL)
- \( z \) (gate level)
What to put in sensitivity list?

- All signals you do a test on and all signals that are on the right side of an assignment.
- In other words all the signals you are “reading” in the value
- Don’t read and write a signal at the same time !!!

Object Types

- Constants
- Signals
- Variables
Constant

- It is just a name for a value.
  \[
  \text{reset}_c := '0'; \text{bus}_\text{width}_c := 32;
  \]
- a better documented design.
- it is easier to update the design.
- **But do not exaggerate !!!**
  (since you have to remember all these names you defined)

Signals

- It is a physical signal (you can think of it like a piece of wire)
- It is possible to define global signals (signals that can be shared among entities)
- But more often signals are just locally defined for a given architecture
- A signal assignment takes effect only after a certain delay (the smallest possible delay is called a “delta time”)

Variables

- It is used as a local storage mechanism, visible only inside a process
- All assignment to variables are scheduled immediately

Signals vs. Variables

- Signals assignments are scheduled after a certain delay $\delta$
- Variables assignments happen immediately, there is no delay
Delta Time

architecture rtl of logic is
signal a_or_b : std_logic;
begin
     a_or_b <= a or b;   -- a_or_b is scheduled @ t+Δ
     z <= a_or_b and c; -- z is scheduled @ t+2Δ
end rtl;

Bad example !!!

architecture bad of logic is
signal a_or_b : std_logic;
begin
     logic_p: process(a,b,c)
     begin
     a_or_b <= a or b;
     z <= a_or_b and c;
     end process;
end bad;

Do not “read” and “write” a signal at the same time !!!
How to fix the bad example

architecture good of logic is
variable a_or_b : std_logic;
begin
logic_p: process(a,b,c)
begin
    a_or_b := a or b;
z <= a_or_b and c;
end process;
end good;

Packages

- Packages offers a mechanism to globally define and share values, types, components, functions and procedures that are commonly used.

- package declaration and package body
Subprograms

- Procedures can return more than one value (they can have both input and output parameters)

- Functions return always just one value (can have only input parameters)
  Example: conversion functions, resolution functions, …

Attributes

- Info attached to VHDL objects
- Some predefined attributes:
  - ‘left → the leftmost value of a type
  - ‘right
  - ‘high → the greatest value of a type
  - ‘low
  - ‘length → the number of elements in an array
  - ‘event → a change on a signal or variable
  - ‘range → the range of the elements of an array object
Generic

- parameter that pass information to an entity

entity adder is
  generic (width : integer := 5);
  port (
    in_a : std_logic_vector(width-1 downto 0);
    in_b : std_logic_vector(width-1 downto 0);
    z      : std_logic_vector(width-1 downto 0);
    carry: std_logic)
  );
end entity adder;

Component (socket mechanism)

- Declare the name and interface of a "sub-unit", to be used in the current level of design hierarchy.

component adder
  generic (width : integer := 5)
  port (    in_a, in_b: in std_logic_vector;
            z : std_logic_vector; carry: std_logic);
end component;
Example: a 7 bit adder (with bugs 😊)

- adder-rtl.vhd
- big-adder-struct.vhd

adder-rtl.vhd

```vhdl
-- -- author: Claudio Talarico
-- file: adder-rtl.vhd
-- comment: example of how to use generics and components

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity adder is
generic (width : integer := 2);
port ( a : in  std_logic_vector(width-1 downto 0);
      b : in  std_logic_vector(width-1 downto 0);
      c : out std_logic; --carry
      z : out std_logic_vector(width-1 downto 0) )
end entity adder;

architecture rtl of adder is
begin
  adder_p: process (a,b)
    variable a_v, b_v : unsigned(a'range); -- use of attributes
    variable z_v : unsigned(z'length downto z'low);
  begin
    a_v := unsigned(a); -- type casting
    b_v := unsigned(b);
    z_v := a_v + b_v;
    z <= std_logic_vector(z_v(width-1 downto 0)); -- type casting
    c <= z_v(width);
  end process adder_p;
end rtl;
```

---

```
```
-- author: Claudio Talarico
-- file: big-adder-struct.vhd
-- comment: example of how to use generics and components

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity big_adder is
port ( a : in  std_logic_vector(6 downto 0);
      b : in  std_logic_vector(6 downto 0);
      c : out std_logic; --carryout
      z : out std_logic_vector(6 downto 0)
    );
end big_adder;

architecture struct of big_adder is
begin
inst_add_l: adder -- low adder
  generic map (width => 4) -- if there is more than one generic the separator is ,
  port map ( a => a_in(3 downto 0),
             b => b_in(3 downto 0),
             c => c_l,
             z => z_out(3 downto 0));

inst_add_h: adder -- high adder
  generic map (width => 3) -- if there is more than one generic the separator is ,
  port map ( a => a_in(6 downto 4),
             b => b_in(6 downto 4),
             c => c_h,
             z => z_out(4 downto 4));

-- dummy assignment
  c <= c_h;
end struct;
**ASSERT statement**

- The ASSERT checks a boolean expression and if the value is true does nothing, else will output a text string to std output. It can have different severity levels: NOTE, WARNING, ERROR, FAILURE

```plaintext
ASSERT false
REPORT "End of TestBench"
SEVERITY ERROR;
```

**COMPLEX TYPES:**

- **enumerated types**
  TYPE color is (red, blue, yellow, green)

- **ARRAY**
  TYPE dbus is ARRAY (31 downto 0) of std_logic
COMPLEX TYPES:

- RECORD

  TYPE instruction is RECORD
  opcode: integer;
  src: integer;
  dest: integer;
  END RECORD

COMPLEX TYPES:

- FILE

  TYPE ram_data_file_t IS FILE OF INTEGER;
  FILE ram_data_file : ram_data_file_t IS IN
  
  “/claudio/vhdl/tb/ram.txt”
More on FILEs

- use std.textio.all;
- READ, WRITE, READLINE, WRITELINE, ENDFILE, …

Advanced Topics

- VHDL supports overloading