VHDL

- VHDL=Very High Speed Integrated Circuit Hardware Description Language
- The development of VHDL was originally initiated by Ministry of Defense in USA in 1981
- VHDL is specified especially to design circuits at the behavioral and the gate level
- VHDL is
  - based on a public standard
  - design-system independent
  - technology independent

VHDL

- VHDL can model
  - –a component
  - –an ASIC or FPGA
  - –a PCB
  - –a system

VHDL model can be used as

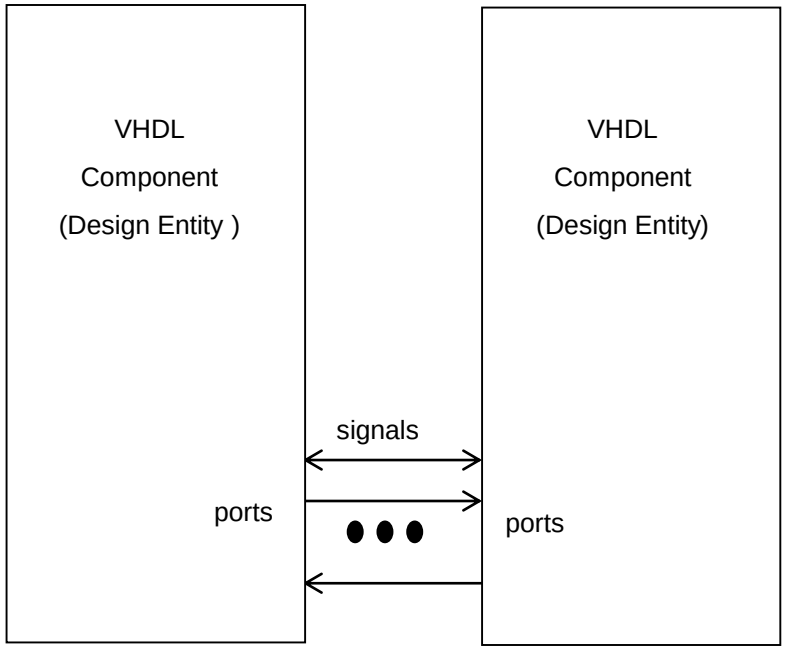
- –a document
- –a simulation model
- –a description in logic synthesis

Design Levels

- System level
- Behavioral (Algorithm) level
- RTL (Register Transfer Level)
- (Logic level)
- Gate level
- Circuit (transistor) level
- Physical level

VHDL Components

- *Design entity* describes a component
- Design entity includes *entity declaration* and *architectural body*
- Components are connected via *ports* by using *signals*



Design Entity Description

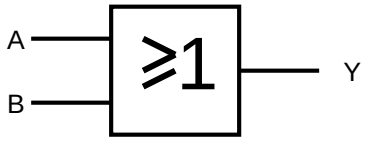
-- means comment in VHDL

entity starts the declaration of design unit

port is used to define interfaces to outside of component

```
--Design entity declaration
Entity logic_or_gate is
    port(A:in Bit; B:in Bit;Y:out Bit);
end  logic_or_gate;
```

```
entity identifier is
    generic interface_list;
    port interface-list;
    declarations;
begin
    statements;
end identifier;
```



Architecture Description

Architecture statement starts the behavioral description of a component

After **begin** are the statements that define the desired operation

architecture identifier **of** entity_mark **is**

begin

statements;

end identifier;

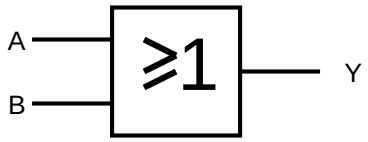
--Architecture of OR

Architecture func_or **of** logic_or_gate **is**

begin

Y<=A **or** B;

end func_or;



VHDL Data Types

- Integer
- Real
- Boolean
- Bit
- Severity Level
- Character

type integer **is range** -(2**3-1) **to** (2**31-1);

type real **is range**

-16#0.7FFF_FF8#E+32 **to** 16#0
.7FFF_FF8#E+32;

type probability **is range** 0.0 **to** 1.0;

type boolean **is** (false, true);

type bit **is** ('0','1');

type severity_level **is** (note, warning, error, failure);

type character **is** (NUL, SOH,STX,…

,…,'0','1','2',...

,…,'@','A','B',…

,…,'a','b',…DEL);

VHDL Data Types

- Type definition, general form
  type idenfier **is** type-def
- Examples

```
type basic_logic is ('0','1');
type 4_lev_logic is ('X','0','1','Z');
type resistance is range 1 to 10E9;
  --Time is predefined VHDL data type
type Time is range -(2**31-1) to (2**31-1)
units
    fs;
    ps=1000fs;
    ns=1000ps;
    us=1000ns;
    ms=1000us;
    sec=1000ms;
    min=60sec;
    hr=60min;
end units
```

- Composite types:record and array

```
type record-def is record
  record data;
end record;
```

```
type instruction is record
   OP: opcode;
    A:  address;
    D: data;
end record;
```

```
type word is array (15 downto 0) of bit;
type byte is array (7 downto 0) of bit;
-- example of unconstrained array
type vector is array(integer range <>) of real
```


Subtypes

- Subtype defines type, which is part of another type definition range
- Subtype definition, general form

subtype idenfier **is** type-de

subtype low_case **is** character **range** ’a’ **to** ’z’;

--part of the integer range
subtype byte_int **is** integer **range** 0 **to** 255;

IEEE Standard 1164

- IEEE 1164 standard standardises logic data types (std_ulogic)

'U'	Uninitialized
'X'	Unknown
'0' (driven)	Logic 0
'1' (driven)	Logic 1
'Z' impedance	High
'W'	Weak 1
'L' (read)	Logic 0
'H' (read)	Logic 1
'-'	Don't-care

```
library ieee;

use ieee.std_logic_1164.all;
```

Operators

- Logical operators

Operator	Description	Operand Types	Result Type
and	And	Any Bit or Boolean type	Same Type
or	Or	Any Bit or Boolean type	Same Type
nand	Not And	Any Bit or Boolean type	Same Type
nor	Not Or	Any Bit or Boolean type	Same Type
xor	Exclusive OR	Any Bit or Boolean type	Same Type
xnor	Exclusive NOR	Any Bit or Boolean type	Same Type

Operators

- Relational operators

Operator	Description	Operand Types	Result Type
=	Equality	Any type	Boolean
/=	Inequality	Any type	Boolean
<	Less than	Any scalar type or discrete array	Boolean
<=	Less than or equal	Any scalar type or discrete array	Boolean
>	Greater than	Any scalar type or discrete array	Boolean
>=	Greater than or equal	Any scalar type or discrete array	Boolean

Operators

- Some other operators

Operator	Description	Operand Types	Result Type
*	Multiplication	See manual	See manual
/	Division	See manual	See manual
mod	Modulus	Any integer type	Same type
rem	Remainder	Any integer type	Same type
+	Adding (also sign)		
-	Subtraction (also sign)		
**	Exponentiation	See manual	See manual

Process

- The process keyword defines a sequential process intended to model all or part of a design entity
- A process statement includes—in this order—an optional sensitivity list

```
process(Rst,Clk)
variable Qreg:
std_ulogic_vector(0 to 7);
 begin
    if Rst = '1' then -- Async reset
        Qreg := "00000000";
   elsif rising_edge(Clk) then
        if Load = '1' then
            Qreg := Data;
        else
             Qreg := Qreg(1 to 7) &
Qreg(0);
        end if;
   end if;
   Q <= Qreg;
end process;
```

Attributes

- Some objects can be attached additional information=attribute
- syntax is:

  name'attribute_identifier

- Object has only one value at the certain moment, but it can have several attributes
- There are available some predefined attributes
- VHDL allows also a designer define user-specific attributes

'left
'right
'high
'low
'length
'ascending
'pos
'val
'pred
'succ
'leftof
'rightof
'range

Some Signal Related Attributes

’active

’event

’last_value

’last_event

’last_active

```
--if any transaction, ’event returns TRUE
if data’event then

--look for clock edge
if Clk = ‘1’ and Clk’event then

--returns the value prior to the last event
Q <= ‘1’;
wait 10 ns;
Q <= ‘0’;
 wait 10 ns;
V := Q’last_value; -- V gets a value of ‘1

--returns the time elapsed after the
previous event
Time_elapsed:=data’last_event;

--’last_active returns time elapsed after
last  transaction
```


If statement

```
if condition then
    sequence of statements;
elsif
    sequence of statements;
Else
    sequency of statements;
end if;
```

```
if A > B then
    Compare <= GT;
elsif A < B then
    Compare <= LT;
else
    Compare <= EQ;
end if;
```

Case Statement

```
case expression is
    case_statement_1;
    case_statement_2;
                    …
end case;
```

```
case input is
    when '0'=> return '1';
    when '1'=> return '0';
    when 'Z'=> return 'Z';
end case;
```

Loop Statement

```
loop
    do something;
end loop;

for item in 0 to last loop
    do something;
end loop;

while condition loop
    do something;
end loop;
```

```
while (I < DBUS'length) loop
    ...
    I := I + 1;
end loop;

--or other example
for I in 0 to DBUS'length - 1 loop
    ...
end loop;
```


Interfaces

- Four possible structures to specify
  - entity declaration
  - local component declaration
  - block statement
  - subprogram definition (procedures and functions)
- Each entity declaration can have two interface list. One for ports and the second for general parameters (generics)

Interface objects in the same interface have three common things

- object class (**signal**, **constant**, **variable**)
- dataflow direction (**in**, **out**, **inout**, **buffer**)
- data type (,for example **bit**)

Subprograms

- Procedures and functions are the subprograms used in VHDL
- Procedures don't have return value like the functions do
- Procedures are used as independent statements but functions are a part of expression
- Function parameters must be **in**-type parameters

```
procedure dff (signal Clk,Rst,D: in
std_ulogic; signal Q: out std_ulogic) is
begin
    if Rst <= '1' then Q <= '0';
    elsif rising_edge(Clk) then Q <= D;
end if;
end dff;
```

Example Function

```
--Function finds the minimum value of two inputs
function min(X,Y:integer) return integer is
begin
    if X<Y then
        return X;
    else
        return Y;
    end if;
end min;
```

Local Subprogram

```
architecture my_architecture of my_design is
    begin
        my_process: process(...)
        function my_local_function(...) return bit is
            begin

                                . . .
            end my_local_function;
        begin
                                . . .
        end process my_process;
end my_architecture;
```


Global Subprogram

```
package my_package is
    function my_global_function(...) return bit;
end my_package;

package body my_package is
    function my_global_function(...) return bit is
    begin
        . . .
    end my_global_function;
end my_package;
```

```
. . .
use work.my_package.my_global_function;
entity my_design is
begin
. . .
end my_design;
```

Component Declaration and Use

- is a part of a package declaration or a part of architecture body
- Component declaration must be placed in the declaration section of the architecture body
- In the component instantiation must be a label
- **work** a special library (of VHDL) and that does not require a library statement and into which all design units are analyzed by default

```
architectuture parent_body of example is
   component and_1
      port (A,B: in bit; Y:out bit);
   end component;
   signal S1,S2,S3:bit;
begin
   test: and_1 port map (A=>S1, B=>S2,Y=>S3);
end parent_body;

--component for package
package pack is
component and_1
      port (A,B: in bit; Y:out bit);
end component;
end pack;

--package
use  work.pack;  --work VHDL specialities
architectuture parent_body of example is
   signal S1,S2,S3:bit;
begin
   test: pack.and_1 port map (A=>S1, B=>S2,Y=>S3);
end parent_body;
```

Library

- The library keyword identifies a library.
- The library statement is a context clause used to identify libraries from which design units can be referenced
- All design units include automatically STD and WORK libraries

--use library

libaray std_logic_1164;

-- all design units have this declaration (without user declaration)

library STD, WORK;

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
entity oor is
    port( A, B: in std_logic; Y: out std_logic);
end oor;

architecture functionality of oor is
begin
   process (A,B)
   begin
      Y<=A or B;
   end process;
end functionality;
```


STUDY MATERIAL

Second VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;  --take std logic in use

entity ls374 is      --entity with port definitions for ls374
   port (CLK, OC: in std_logic; D:in Std_logic_vector (0 to 7);
   Q: out std_logic_vector (0 to 7));
end ls374;

architecture functionality_of_ls374 of ls374 is  --architecture desription for ls374
begin
   process (CLK, OC)
      variable result: std_logic_vector (0 to 7) ;
   begin
      if (OC='1') then Q<="ZZZZZZZZ";
      else  Q<=result;
      end if;
      if (CLK='1' and CLK'event) then result:=D;
      end if;
   end process;
end functionality_of_ls374;
```


An Example of FPGA Development Tools

- QUARTUSII, Web Edition for Altera Products

http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html