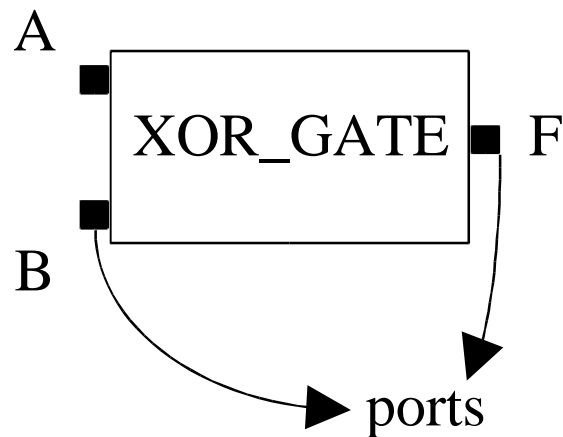


Getting Started with VHDL

- ❑ VHDL code is composed of a number of *entities*
- ❑ Entities describe the interface of the component
- ❑ Entities can be primitive objects or complex objects
- ❑ *Architectures* are associated with each entity
- ❑ Architectures describe:
 - ❑ Behavior
 - ❑ Structure

VHDL Entities



```
ENTITY xor_gate IS
  GENERIC (prop_delay:TIME := 10ns);
  PORT(
    a : IN std_logic;
    b : IN std_logic;
    f : OUT std_logic);
END xor_gate;
```

Entity Syntax

```
entity <name> is  
  [generic(  
    <gen_name> : <type*>[:=<init>];  
    .  
    .  
    . )]  
  port (  
    <port_name> : <dir**> <type>;  
    .  
    .  
    . );  
end <name>;
```

* <type> is any VHDL type

** <dir> is a direction of the port:
IN, OUT, or INOUT

VHDL Architectures

- ❑ Describe the implementation of entities
- ❑ Can be behavioral, data–flow, structural, or a combination

```
ARCHITECTURE behavior OF xor_gate IS
BEGIN
    new_input : PROCESS(a,b)
    BEGIN
        IF (a = b)
            f <= 0 AFTER prop_delay;
        ELSE
            f <= 1 AFTER prop_delay;
        END PROCESS;
    END behavior;
```

Data-flow Architecture

- ❑ Consists entirely of boolean equations
- ❑ References only port signals

```
ARCHITECTURE dataflow OF xor_gate IS
BEGIN
    f <= (a AND NOT b) OR (NOT a AND b)
        AFTER prop_delay;
END dataflow;
```

Structural Architecture

- ❑ Defines an entity using components
- ❑ Defines internal signals to connect components

```
ARCHITECTURE structure OF xor_gate IS
  COMPONENT nor_gate
    PORT (x, y: IN std_logic;
          z: OUT std_logic);
  END COMPONENT;
  COMPONENT and_gate
    PORT (x, y: IN std_logic;
          z: OUT std_logic);
  END COMPONENT;
  SIGNAL s1, s2: BIT;
BEGIN
  nor1: nor_gate PORT MAP (s1, s2, f);
  and1: and_gate PORT MAP (a, b, s1);
  nor2: nor_gate PORT MAP (a, b, s2);
END structure ;
```

Syntax of an Architecture

```
Architecture <name> of <entity> is  
    <declarations>  
begin  
    <statements>  
end <name> ;
```

Data Types

- ❑ Many data types in VHDL, but we will use a small subset:

STD_LOGIC

A single bit

STD_LOGIC_VECTOR

A bit vector, often used for a bus

INTEGER

Standard integer type that will often be used for generic parameters

Declarations – Signals

```
CONSTANT bus_width : INTEGER := 32;
CONSTANT rise_delay : TIME := 20ns;

VARIABLE data_val : STD_LOGIC_VECTOR (7 DOWNTO 0);
VARIABLE sum : INTEGER RANGE 0 TO 100;
VARIABLE done : BOOLEAN;

SIGNAL clock : STD_LOGIC;
SIGNAL addr_bus : STD_LOGIC_VECTOR (31 DOWNTO 0);
```

❑ Implicit declaration

```
FOR count IN 1 TO 10 LOOP -- declares count
    ❑ sum := sum + count;
END LOOP;
```

Declarations – Components

```
component fill_reg
port(
  clk      :in  std_logic;
  rst      :in  std_logic;
  write    :in  std_logic;
  read     :in  std_logic;
  data_in  :in  std_logic_vector(3 downto 0);
  FFout    :out std_logic;
  data_out :out std_logic_vector(31 downto 0));
end component;
```

Architecture Body

- ❑ Concurrent Signal Assignment statements (CSAs)

$a \leq b + c;$

- ❑ Component Instantiations

- ❑ Processes with behavioral descriptions

Component Instantiations

```
reg5 : fill_reg
  port map (  clk      => clk,
             rst      => rst,
             write    => write_comp,
             read     => readout,
             data_in  => dataout_comp,
             FFout   => FFout,
             data_out => datareg5_out);
```

```
reg5 : fill_reg
  port map (clk, rst, write_comp,
           readout, dataout_comp,
           FFout, datareg5_out);
```

Processes

❑ Process Statement

```
<label> PROCESS ( <sensitivity> )  
    < process declarations >  
BEGIN  
    < statements >  
END PROCESS <label> ;
```

- ❑ Process executes whenever an event occurs on a signal in the sensitivity list
- ❑ Multiple processes can execute concurrently

Processes

- ❑ Sensitivity list tell simulator when to evaluate process
- ❑ No sensitivity list is required, BUT if not, *wait* statements must be used in the process

Processes Example

```
process
begin
    wait for 15 ns;
    clkx2 <= not(clkx2);
end process;
```

```
process(clk, rst)
begin
    if rst = '1' then
        readout <= '0';
    elsif (clk'event and clk= '1') then
        if (Ffout = '1') then
            readout <= '1';
        else
            readout <= '0';
        end if;
    end if;
end process;
```

Architecture Example

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity count_example is
  port(
    clk      : in std_logic;
    rst      : in std_logic;
    enable   : in std_logic;
    load     : in std_logic;
    data_in  : in std_logic_vector(4 downto 0);
    data_out : out std_logic_vector(4 downto 0));
end count_example;
```

Architecture Example

architecture version1 of count_example is

component adder

```
generic ( WIDTH : INTEGER := 8 )
```

```
port (
```

```
  in1 :in std_logic_vector(WIDTH-1 downto 0);
```

```
  in2 :in std_logic_vector(WIDTH-1 downto 0);
```

```
  out :out std_logic_vector(WIDTH-1 downto 0)
```

```
);
```

```
signal CNT      :std_logic_vector(4 downto 0);
```

```
signal result   :std_logic_vector(4 downto 0);
```

```
CONSTANT ONE   :
```

```
  std_logic_vector(4 downto 0) := "00001";
```

Architecture Example

```
begin
data_out <= CNT;
  adder1 : adder
    generic map ( WIDTH => 5)
    port map (
      in1 => CNT,
      in2 => ONE,
      out => result
    );
process (clk, rst)
begin
if rst = '1' then
  CNT <= "00000";
elsif (clk'event and clk = '1') then
  if (load = '1') then
    CNT <= data_in;
  elsif (enable = '1') then
    CNT <= result;
  else
    CNT <= CNT;
  end if;
end if;
end process;
end version1;
```

Operators

□ Logical

and or nand nor xor xnor not

NOTE: nand and nor not associative

□ Relational

= /= < <= > >=

□ Shift

sll srl sla sra rol ror

□ Adding

+ - &

□ Multiplying

* / mod rem

a rem b = a - (a/b) * b

a mod b = a - b * n (for some n)

□ Other

abs **

Behavioral Statements

□ Wait Statement

```
WAIT ON <sensitivity> UNTIL <expression> FOR  
    <time>
```

Examples:

```
WAIT ON A, B, C;
```

```
WAIT UNTIL A > 10;
```

```
WAIT FOR 30ns;
```

```
WAIT ON A FOR 100ns;
```

```
WAIT UNTIL C < 4 FOR 50ms;
```

```
WAIT ON B UNTIL A > 4;
```

```
WAIT ON C UNTIL B > 5 FOR 30us;
```

```
WAIT; -- suspends forever
```

Control Flow Statements

□ LOOP Statement

```
<label> : FOR <identifier> IN <range> LOOP  
    <statements>  
END LOOP;
```

```
<label> : WHILE <expression> LOOP  
    <statements>  
END LOOP;
```

```
<label> : LOOP  
    <statements>  
END LOOP;
```

Control Flow Statements

□ IF Statement

```
IF <expression> THEN
    <statements>
ELSIF <expression>
    <statements>
ELSE
    <statements>
END IF;
```

□ CASE Statement

```
CASE <expression> IS
WHEN <choice> => <statements>
WHEN <choice> => <statements>
WHEN OTHERS => <statements>
END CASE;
```

Sub-Programs

❑ Functions

```
FUNCTION <designator> ( < args> ) RETURN  
    <return-type> IS  
    <declarations>  
BEGIN  
    <statements>  
END;
```

❑ Procedures

```
PROCEDURE <identifier> ( < args> ) IS  
    <declarations>  
BEGIN  
    <statements>  
END;
```

VHDL Literals

□ Numbers

```
14364  
14_364  
16#FF03#  
16.3  
13_634.34  
1.293E-23  
16#3A.B2#E3A
```

□ Characters

```
'A'  
'_'  
' ''  
"This is a string"
```

□ Bits

```
X"FF_AA"  
B"1101_1010"  
O"037"
```

Architectures and Entities

- ❑ At least one architecture must be available for each entity

```
ENTITY xor_gate IS
    GENERIC (prop_delay:TIME := 10ns);
    PORT(
        a : IN STD_LOGIC;
        b : IN STD_LOGIC;
        f : OUT STD_LOGIC);
END xor_gate;
ARCHITECTURE behavior OF xor_gate IS
BEGIN
    new_input : PROCESS(a,b)
    BEGIN
        IF (a = b)
            f <= 0 AFTER prop_delay;
        ELSE
            f <= 1 AFTER prop_delay;
        END PROCESS;
    END behavior;
```

Architectures and Entities

- ❑ More than one architecture may be available for each entity

```
ENTITY xor_gate IS
    .
    .
    .
END xor_gate;
ARCHITECTURE behavior OF xor_gate IS
BEGIN
    .
    .
    .
END behavior;
ARCHITECTURE structure OF xor_gate IS
BEGIN
    .
    .
    .
END structure;
```

Configuration

❑ Can appear in a structural architecture

```
ARCHITECTURE foo OF bar IS
  COMPONENT cmp1 PORT (a: IN std_logic);
  COMPONENT cmp2 PORT (b: IN std_logic);
  FOR ALL: cmp1
    USE ENTITY c1(c1_arch1);
  FOR c2_1, c2_2: cmp2
    USE ENTITY c2(c2_arch1);
  FOR C2_3 : cmp2
    USE ENTITY c2(c2_arch2);
  FOR OTHERS : cmp2
    USE ENTITY c3(c3_arch1);
BEGIN
  c1_1 : cmp1 PORT MAP (a_1);
```

Configuration Declaration

❑ Used as a separate design unit

```
CONFIGURATION c1 OF an_entity IS
  FOR c1_architecture
    FOR ALL: cmp1
      USE ENTITY c1(c1_arch1);
    END FOR;
  FOR c2_1, c2_2: cmp2
    USE ENTITY c2(c_2arch1);
    FOR c2_arch1
      FOR ALL xor: USE ENTITY x(xa);
    END FOR;
  END FOR;
  FOR C2_3 : cmp2
    USE CONFIGURATION foobar;
  END FOR;
  FOR OTHERS : cmp2
    USE OPEN;
  END FOR;
END FOR;
END CONFIGURATION;
```

Configuration Defaults

- ❑ For each unbound component instance:
 - ❑ An entity that is visible and has the same name is bound
 - ❑ The most recently analyzed architecture is used
 - ❑ For each port or generic must correspond by name, type, and mode

Direct Instantiation

- ❑ Can bind an instance directly to an entity or configuration

```
ARCHITECTURE foo OF bar IS
SIGNAL a, b, c;
BEGIN
a1: ENTITY a_ent1(a_arch1)
    PORT MAP (a, b);
a2: ENTITY a_ent1(a_arch2)
    PORT MAP (b, c);
a3: ENTITY a_ent2(a_arch3)
    PORT MAP (c, a);
b1: CONFIGURATION b_conf
    PORT MAP (a, b, c);
END foo;
```

Packages

❑ Package Declaration

```
PACKAGE <identifier> IS  
<declarations>  
END;
```

- ❑ No function or procedure bodies

❑ Package Body Declaration

```
PACKAGE BODY <identifier> IS  
<declarations>  
END;
```

- ❑ Required for each package declaration
- ❑ Body for each function or procedure in package declaration or body declaration

❑ Use Clause

```
USE <library>.<package>.ALL  
USE <library>.<package>.<item>
```

Designs

- ❑ Consist of an entity and an architecture describing it
- ❑ Architecture is synthesizable
- ❑ May have sub-components

Testbenches

- ❑ Used to test a design
- ❑ Entity is empty
- ❑ Generates all external stimulus, including clock and reset

Testbench Example

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
Library WORK;
use WORK.all;

entity testbench is
end testbench;

architecture test of testbench is

signal clk          : std_logic:= '0';
signal rst          : std_logic:= '0';

signal data_in1:std_logic_vector(4 downto 0);
signal data_in2:std_logic_vector(4 downto 0);
signal data_out:std_logic_vector(4 downto 0);
```

Testbench Example

```
component adder
  port(
    in1 : in  std_logic_vector(4 downto 0);
    in2 : in  std_logic_vector(4 downto 0);
    out  : out std_logic_vector(4 downto 0));
end component;

signal a : std_logic_vector(4 downto 0);
signal b : std_logic_vector(4 downto 0);
signal c : std_logic_vector(4 downto 0);

begin

add1 : adder
  port map(
    in1 => a,
    in2 => b,
    out => c
  );
```

Testbench Example

```
-- *****
-- process for simulatimg the clock
process
begin
    clk <= not(clk);
    wait for 20 ns;
end process;

-- *****
-- This process does the RESET
process
begin
    rst <= '1';
    wait for 53 ns;
    rst <= '0';
    wait until(rst'event and rst = '1');
-- stops this process (this is an initial )
end process;
```

Testbench Example

```
process
  variable i : integer;
  variable j : integer;
begin
  for i in 1 to 16 loop
    for j in 1 to 16 loop
      a <= CONV_STD_LOGIC_VECTOR(i, 5);
      b <= CONV_STD_LOGIC_VECTOR(j, 5);
      wait until (clk'event and clk='1');
    end loop;
  end loop;
end process;
--*****

end test;

configuration c_testbench of testbench is
  for test
  end for;
end c_testbench;
```