

# VHDL

# VHDL

- Official Definition:
  - VHSIC Hardware Description Language
    - VHISC – Very High Speed Integrated Circuit

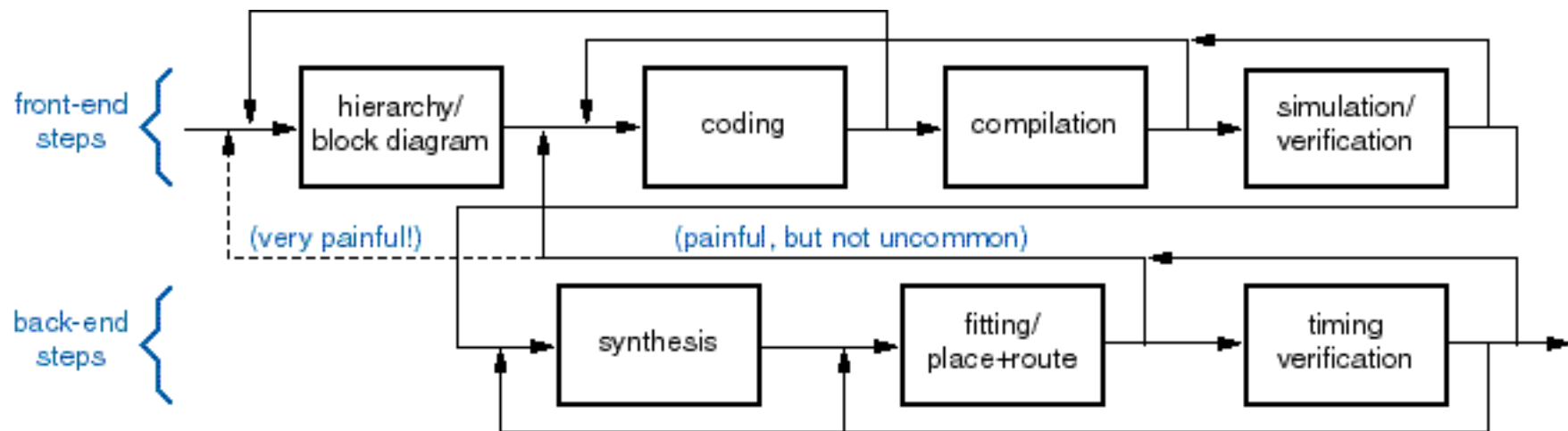
# VHDL

- Alternative (Student Generated) Definition
  - Very Hard Digital Logic language

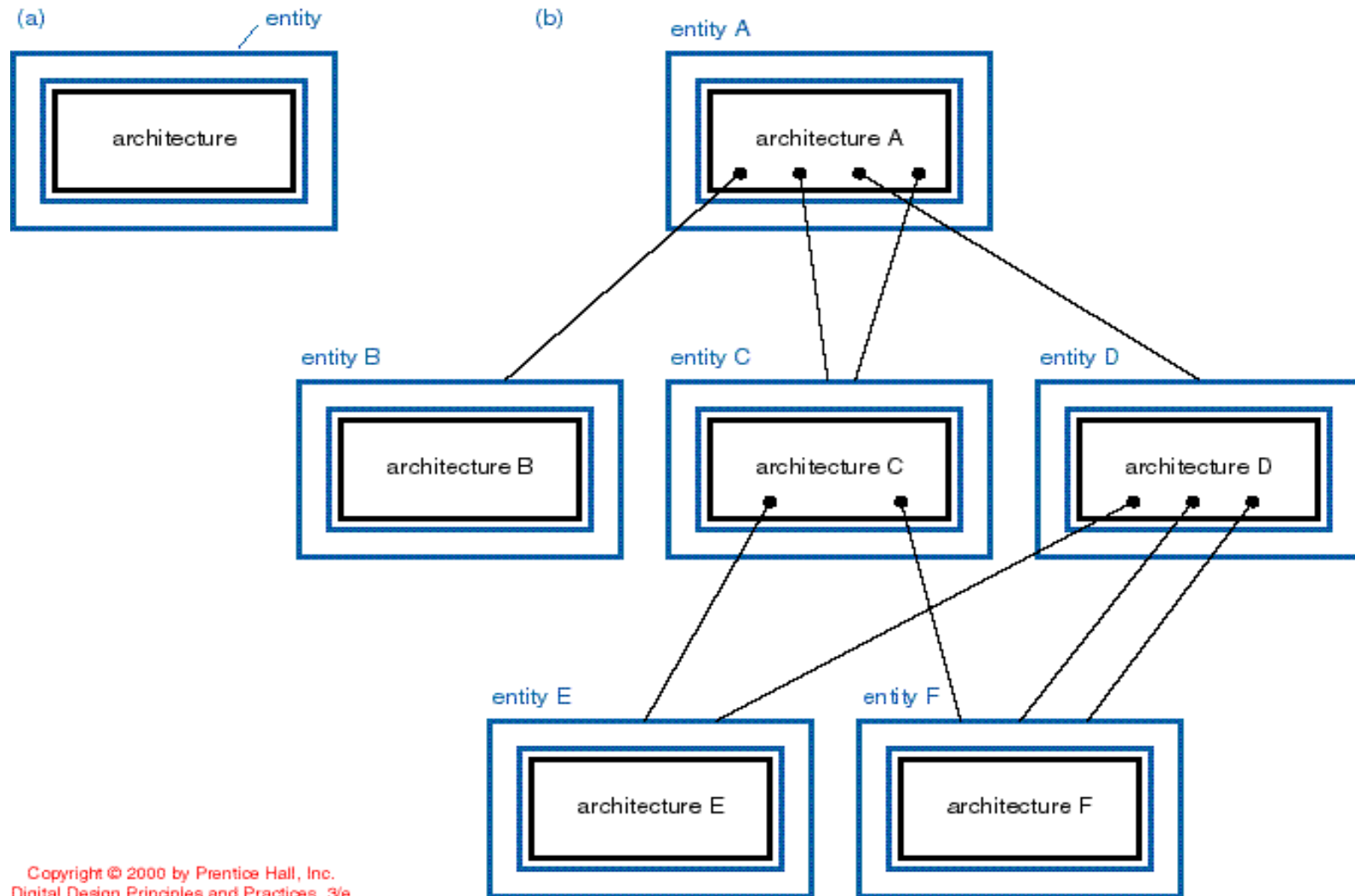
# VHDL Design Flow

- Front-end
  - Hierarchy/Block Diagram
  - Coding
  - Compilation
  - Simulation/Verification
- Back-end
  - Synthesis
  - Fitting/Place+Route
  - Timing Verification

# VHDL Design Flow

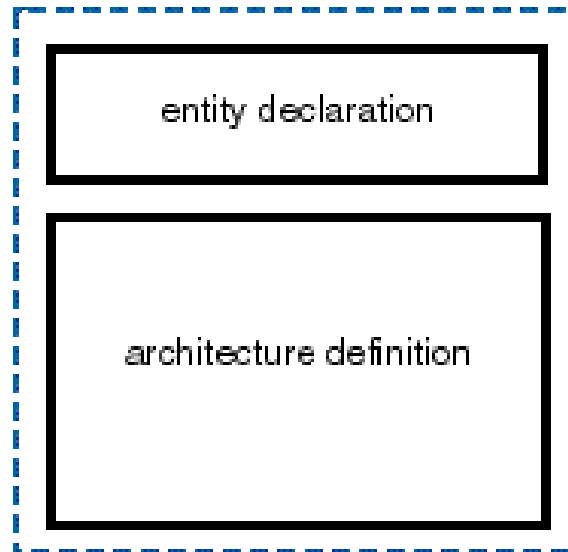


# VHDL Program Structure



# VHDL Entity/Architecture

text file (e.g., mydesign.vhd)



Copyright © 2000 by Prentice Hall, Inc.  
Digital Design Principles and Practices, 3/e

# Entity Syntax

```
entity entity-name is  
    port ( signal-names : mode signal-type ;  
          signal-names : mode signal-type ;  
          . . . .  
          signal-names : mode signal-type ) ;  
end entity-name ;
```

# Entity Port Declaration

- *entity-name* – user defined name for entity
- *signal-names* – list of one or more user defined names
- *mode* – one or more reserved words defining signal direction
  - ♦ *in* – signal is an input
  - ♦ *out* – signal is an output not readable within entity
  - ♦ *buffer* – signal is an output readable within entity
  - ♦ *inout* – signal available as input or output

# Architecture Syntax

```
architecture architecture-name of entity-name is  
    type declarations  
    signal declarations  
    constant declarations  
    function definitions  
    procedure definitions  
    component declarations  
begin  
    concurrent-statements  
    . . . .  
    concurrent-statements  
end architecture-name ;
```

# 2-Input AND Gate Example

```
entity And2 is
```

```
    port ( X, Y : in BIT;  
          F : out BIT );
```

```
end And2
```

```
architecture And2_arch of And2 is
```

```
begin
```

```
    F <= '1' when X='1' and Y='1' else '0';
```

```
end And2_arch
```

# Comparator Example

-- Eight-bit comparator

**entity** compare **is**

**port**( A, B: in bit\_vector(0 to 7);  
          EQ: out bit);

**end** compare;

**architecture** compare\_arch **of** compare **is**

**begin**

    EQ <= '1' when (A = B) else '0';

**end** compare\_arch;

# Architecture Declarations

- Declarations Examples:
  - signal *signal-names* : *signal-type* ;
  - variable *variable-names* : *variable-type* ;
- Predefined types:
  - bit
  - bit\_vector
  - boolean
  - character
  - integer
  - real
  - severity\_level
  - string
  - time

```
Library ieee;
use ieee.std_logic_1164.all;

library std;
use std.textio.all;

entity mux4 is
  port
    (i0, i1, i2, i3, a, b : in std_logic; -- inputs
     q : out std_logic); -- output
end mux4;

architecture mux4 of mux4 is
  signal sel : integer;
  begin
    with sel select
      q <= i0 after 10 ns when 0,
         i1 after 10 ns when 1,
         i2 after 10 ns when 2,
         i3 after 10 ns when 3,
         'X' after 10 ns when others;
    sel <= 0 when a = '0' and b = '0' else
          1 when a = '1' and b = '0' else
          2 when a = '0' and b = '1' else
          3 when a = '1' and b = '1' else
          4;
  end architecture mux4;
```

# Predefined Operators

## Integer Operators:

+	addition
-	subtraction
*	multiplication
/	division
mod	modulo division
rem	modulo remainder
abs	absolute value
**	exponentiation

## Boolean Operators:

and	AND
or	OR
nand	NAND
nor	NOR
xor	Exclusive OR
xnor	Exclusive NOR
not	complementation

# User Defined Enumerated Types

**type** *type-name* **is** (*value-list*) ;

**subtype** *subtype-name* **is** *type-name* **start** **to** *end* ;

**subtype** *subtype-name* **is** *type-name* **start** **downto** *end* ;

**constant** *constant-name* : *type-name* := *value* ;

# Classes of Data Types

- **Scalar**
  - Single numeric or enumerated types, e.g, integer, real
- **Composite**
  - Collection of values
    - Arrays – data of same type, e.g., bit-vector, string
    - Records – data of different types
- **Access**
  - Used to create data directions, similar to pointers in C
- **File**
  - Reference objects (typically disk files) that contain a sequence of values

# Object Types

- Signals – used to connect entities together to form models.
- Variables – used for local storage in process statements and subprograms.
- Constants – names assigned to specific values of a type. Values assigned once and do not change.
  - Syntax:  
`constant constant_name {, constant_name} : type_name [:= value] ;`
  - Example:  
`constant PI : real := 3.1414 ;`  
`constant PORT : string := “This is a string” ;`

# Object Type - Signal

- Signals – used to connect entities together to form models. Globally shared between entities and architectures.
  - Syntax:  
`signal signal_name : signal_type [:= initial_value]`
  - Examples:  
`signal vcc : std_logic := '1' ;`  
`signal ground : std_logic := '0' ;`

# Object Type – Variable

- Variable – used for local storage in processes, procedures and functions.
  - Syntax:  
`variable variable_name {, variable_name} : variable_type [:= value] ;`
  - Examples:  
`variable result : integer := 0 ;`  
`variable prod : integer := 1 ;`

# Concurrent Example

```
architecture arch1 of my_circuit is
    signal A, B, C: std_logic_vector(7 downto 0);
    constant Init: std_logic_vector(7 downto 0) := "01010101";
begin
    A <= B and C;
    B <= Init when Select = '1' else C;
    C <= A and B;
end arch1;
```

is exactly equivalent to:

```
architecture arch2 of my_circuit is
    signal A, B, C: std_logic_vector(7 downto 0);
    constant Init: std_logic_vector(7 downto 0) := "01010101";
begin
    C <= A and B;
    A <= B and C;
    B <= Init when Select = '1' else C;
end arch2;
```

# Sequential Statements

- Allow circuit operation to be described sequentially, i.e., order dependency
- Can be used for combinational as well as sequential circuits

# Process Structure

```
process_name: process (sensitivity_list)
```

```
    declarations
```

```
begin
```

```
    sequential_statements
```

```
end process;
```

# Process Example

```
process(Rst, Clk)
begin
  if Rst = '1' then
    Q <= "00000000";
  elsif Clk = '1' and Clk'event then
    if Load = '1' then
      Q <= Data_in;
    else
      Q <= Q(1 to 7) & Q(0);
    end if;
  end if;
end process;
```

# 4-to-1 Multiplexer Example

```
entity simple_mux is
  port (Sel: in bit_vector (0 to 1);
        A, B, C, D: in bit;
        Y: out bit);
end simple_mux;
```

```
architecture behavior of simple_mux is
begin
  process(Sel, A, B, C, D)
  begin
    if Sel = "00" then
      Y <= A;
    elsif Sel = "01" then
      Y <= B;
    elsif Sel = "10" then
      Y <= C;
    elsif Sel = "11" then
      Y <= D;
    end if;
  end process;
end simple_mux;
```

# VHDL References

- Digital Design, 3<sup>rd</sup> ed, J.F. Wakerly, pp. 264
- VHDL Programming by Example, 4<sup>th</sup> ed, D.L. Perry, 2002.
- Accolade's VHDL Language Guide
  - [<http://www.acc-eda.com/vhdlref/>]
- Xilinx's “PLD Quick Start Handbook”
  - [[http://www.xilinx.com/publications/products/cpld/logic\\_handbook.pdf](http://www.xilinx.com/publications/products/cpld/logic_handbook.pdf)]