Introduction of VHDL

FPGA Design process (1)

Design and implement a simple unit permitting to speed up encryption with RC5-similar cipher with fixed key set on 8031 microcontroller. Unlike in the experiment 5, this time your unit has to be able to perform an encryption algorithm by itself, executing 32 rounds.....

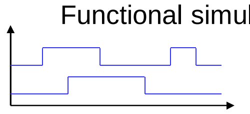
Specification / Pseudocode

On-paper hardware design
(Block diagram & ASM chart)

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

entity RC5_core is
        port(
             clock, reset, encr_decr: in std_logic;
             data_input: in std_logic_vector(31 downto 0);
             data_output: out std_logic_vector(31 downto 0);
             out_full: in std_logic;
             key_input: in std_logic_vector(31 downto 0);
             key_read: out std_logic;
           );
end AES_core;
```

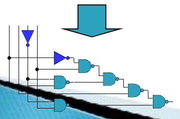
VHDL description (Your Source Files)

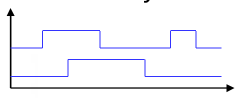
Functional simulation



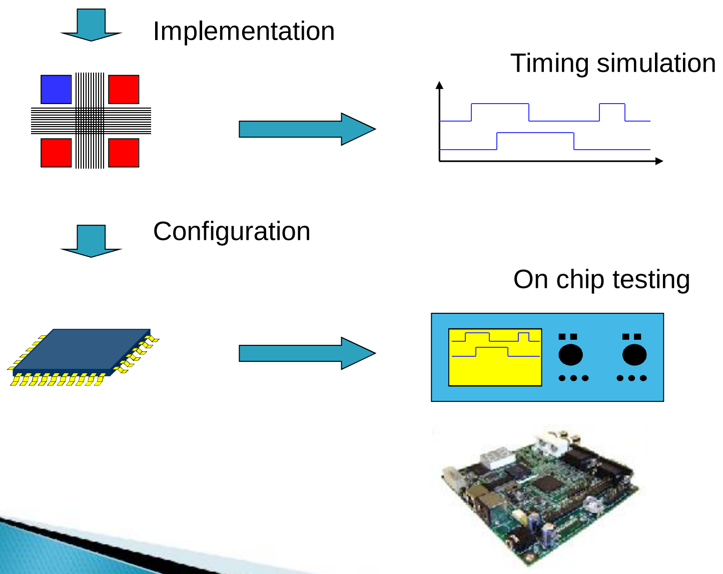
Synthesis

Post-synthesis simulation

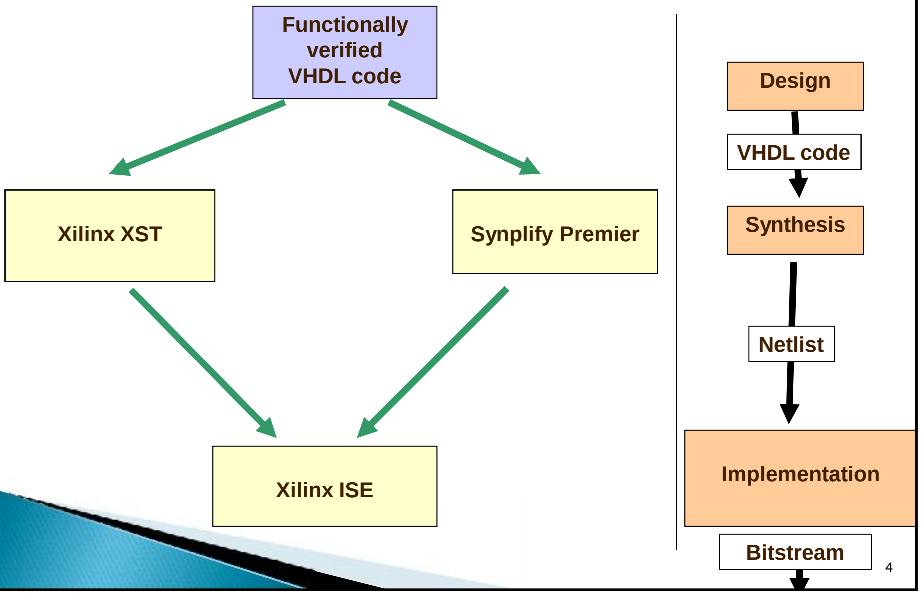




FPGA Design process (2)



Tools used in FPGA Design Flow



Synthesis Tools

XILINX

Synplicity

Xilinx XST

Synplify Premier

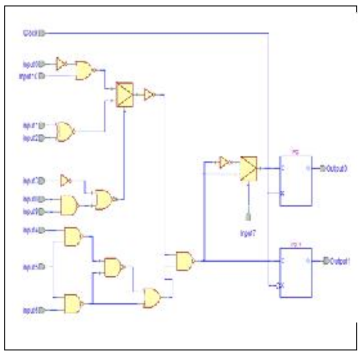
… and others

Logic Synthesis

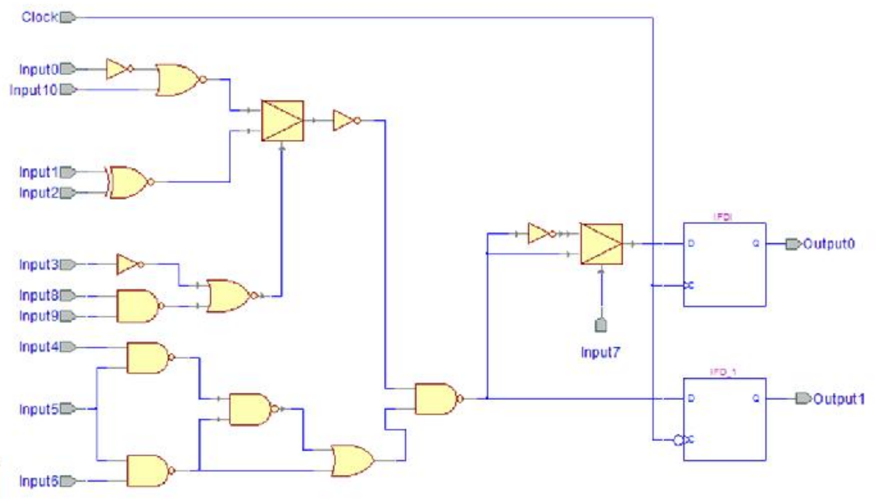
VHDL description

Circuit netlist

```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
            A1<=A when (NEG_A='0') else
                        not A;
            B1<=B when (NEG_B='0') else
                        not B;
            Y<=Y1 when (NEG_Y='0') else
                        not Y1;

            MUX_0<=A1 and B1;
            MUX_1<=A1 or B1;
            MUX_2<=A1 xor B1;
            MUX_3<=A1 xnor B1;

            with (L1 & L0) select
                        Y1<=MUX_0 when "00",
                                    MUX_1 when "01",
                                    MUX_2 when "10",
                                    MUX_3 when others;

end MLU_DATAFLOW;
```

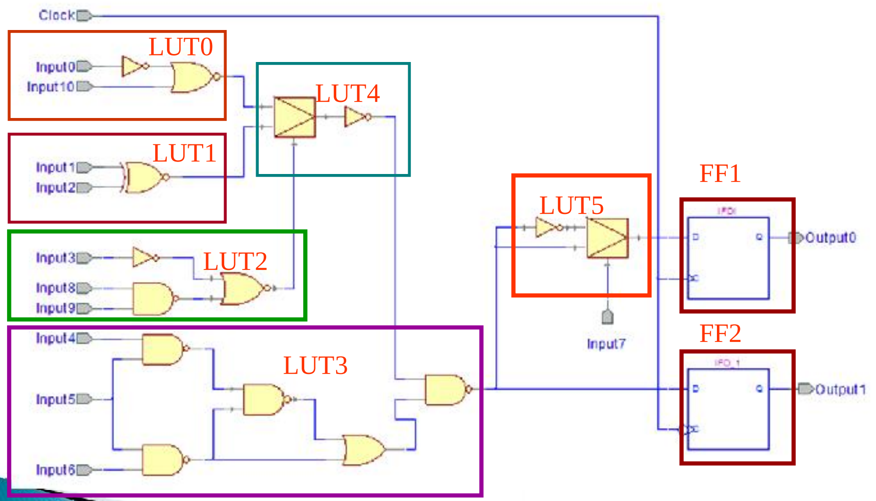


Circuit netlist (RTL view)



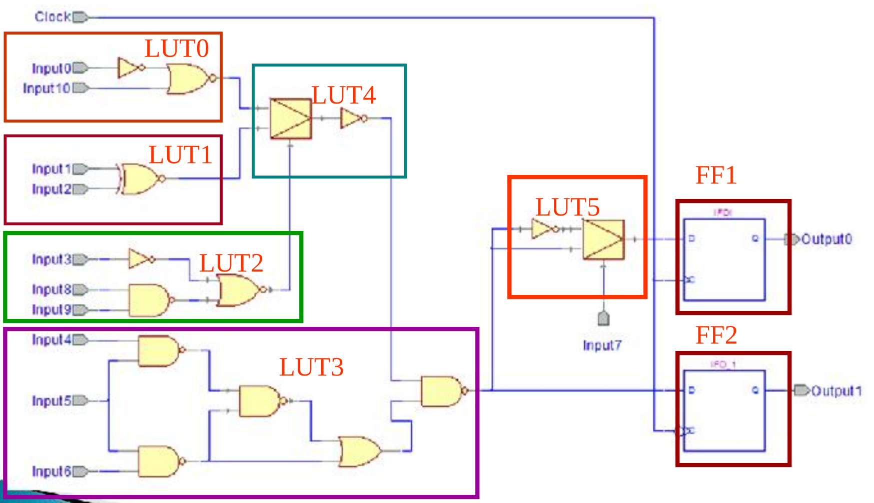
7

Mapping



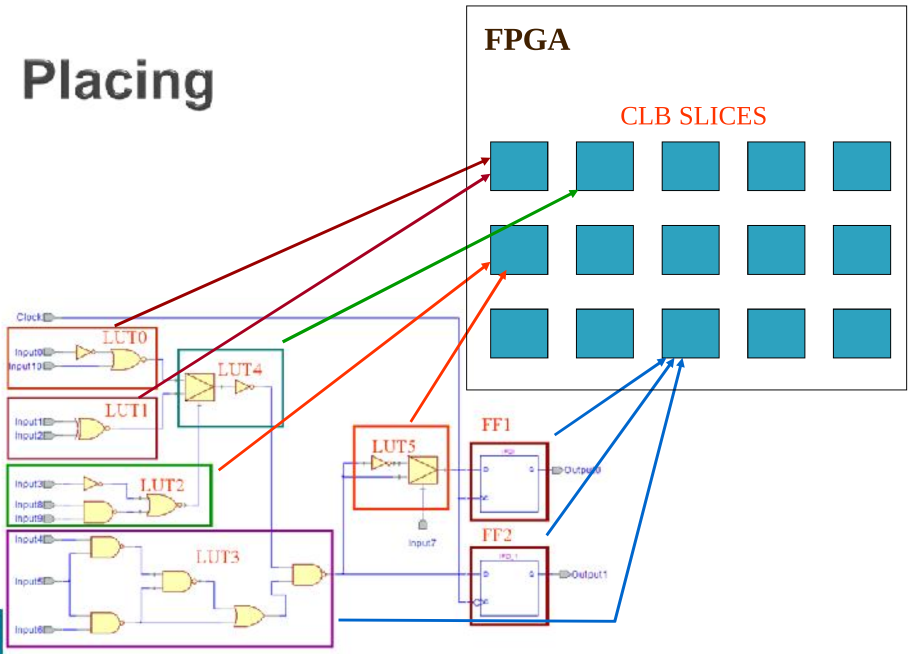
8

Mapping



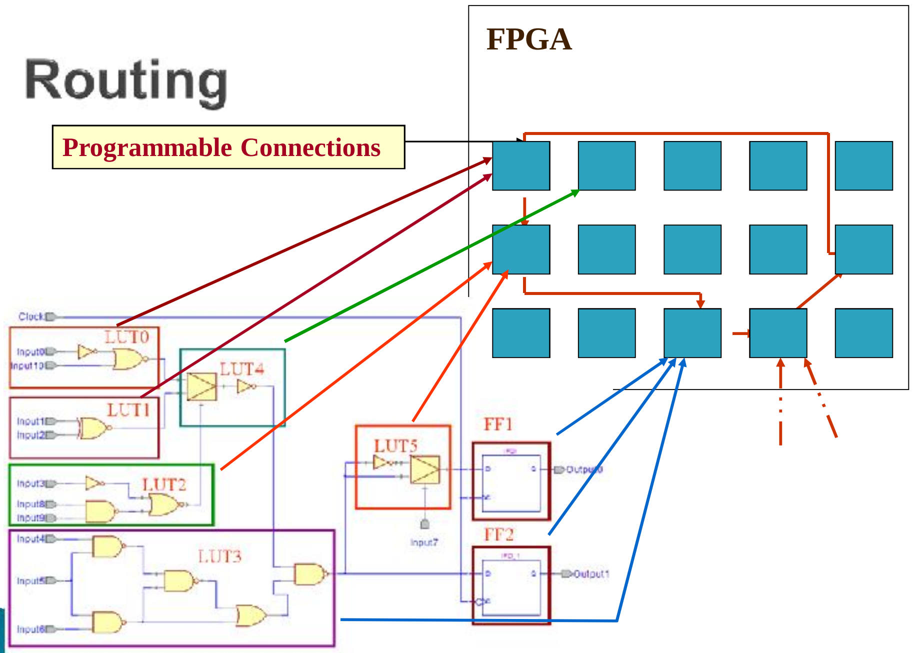
9

Placing



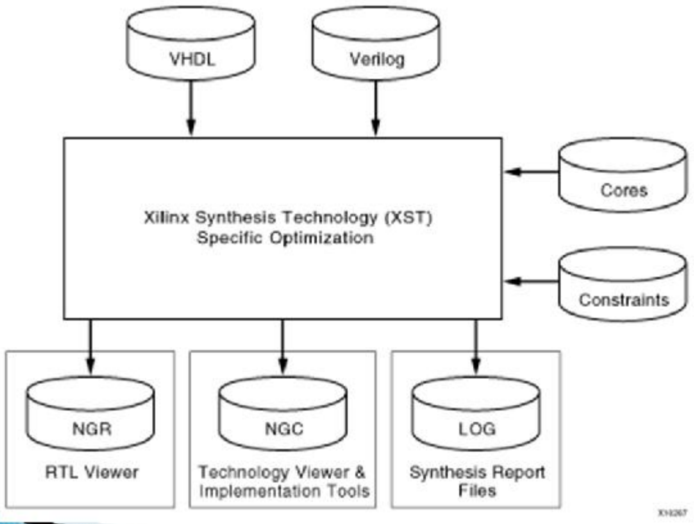
10

Routing



11

Xilinx XST Inputs/Outputs


VHDL
Verilog
Xilinx Synthesis Technology (XST) Specific Optimization
Cores
Constraints
NGR
RTL Viewer
NGC
Technology Viewer & Implementation Tools
LOG
Synthesis Report Files

12

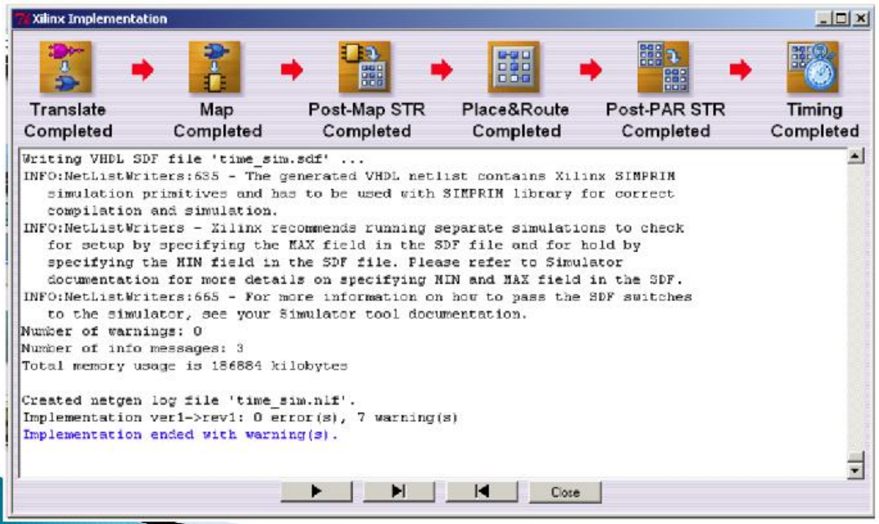
Implementation

- After synthesis the entire implementation process is performed by FPGA vendor tools

XILINX®

13

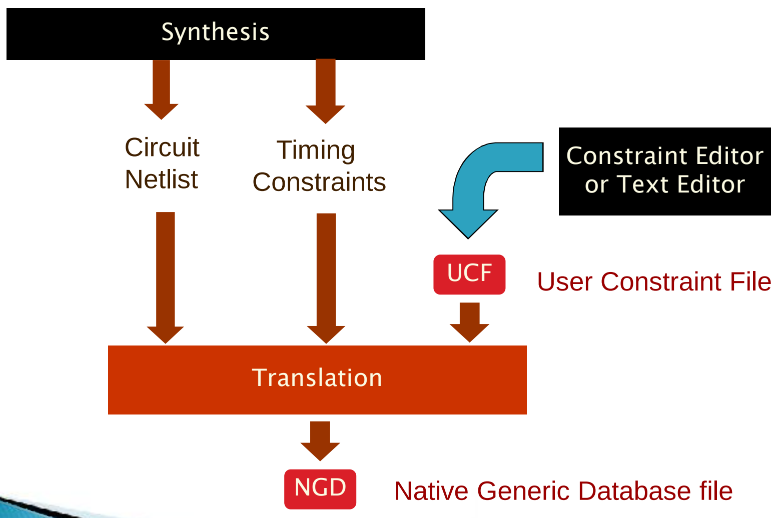
Implementation



```
Writing VHDL SDF file 'time_sim.sdf' ...
INFO:NetListWriters:635 - The generated VHDL netlist contains Xilinx SIMPRIM
   simulation primitives and has to be used with SIMPRIM library for correct
   compilation and simulation.
INFO:NetListWriters - Xilinx recommends running separate simulations to check
   for setup by specifying the MAX field in the SDF file and for hold by
   specifying the MIN field in the SDF file. Please refer to Simulator
   documentation for more details on specifying MIN and MAX field in the SDF.
INFO:NetListWriters:665 - For more information on how to pass the SDF switches
   to the simulator, see your Simulator tool documentation.
Number of warnings: 0
Number of info messages: 3
Total memory usage is 186884 kilobytes

Created netgen log file 'time_sim.nlf'.
Implementation ver1->rev1: 0 error(s), 7 warning(s)
Implementation ended with warning(s).
```

14

Translation



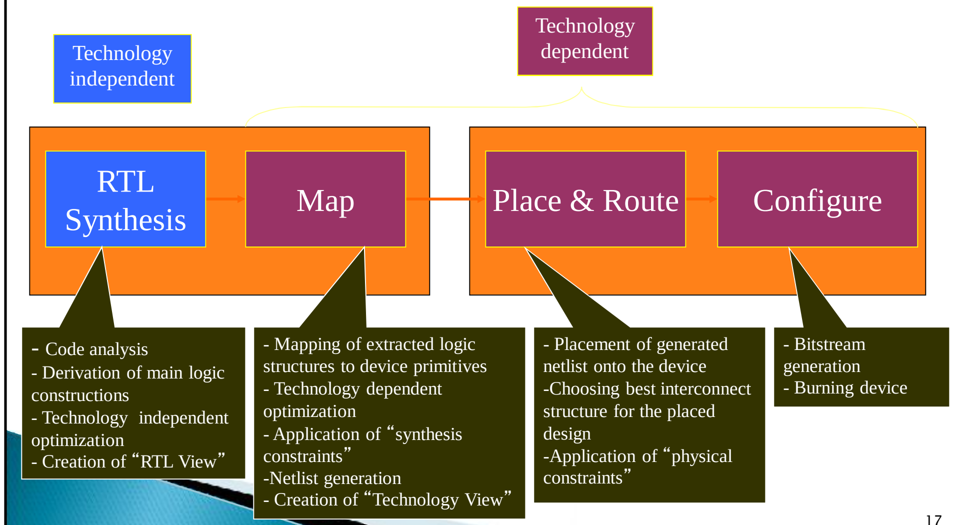
15

Configuration

- Once a design is implemented, you must create a file that the FPGA can understand
  - This file is called a bit stream: a BIT file (.bit extension)
- The BIT file can be downloaded directly to the FPGA, or can be converted into a PROM file which stores the programming information

16

Two main stages of the FPGA Design Flow



- Code analysis
- Derivation of main logic constructions
- Technology independent optimization
- Creation of “RTL View”

- Mapping of extracted logic structures to device primitives
- Technology dependent optimization
- Application of “synthesis constraints”
-Netlist generation
- Creation of “Technology View”

- Placement of generated netlist onto the device
-Choosing best interconnect structure for the placed design
-Application of “physical constraints”

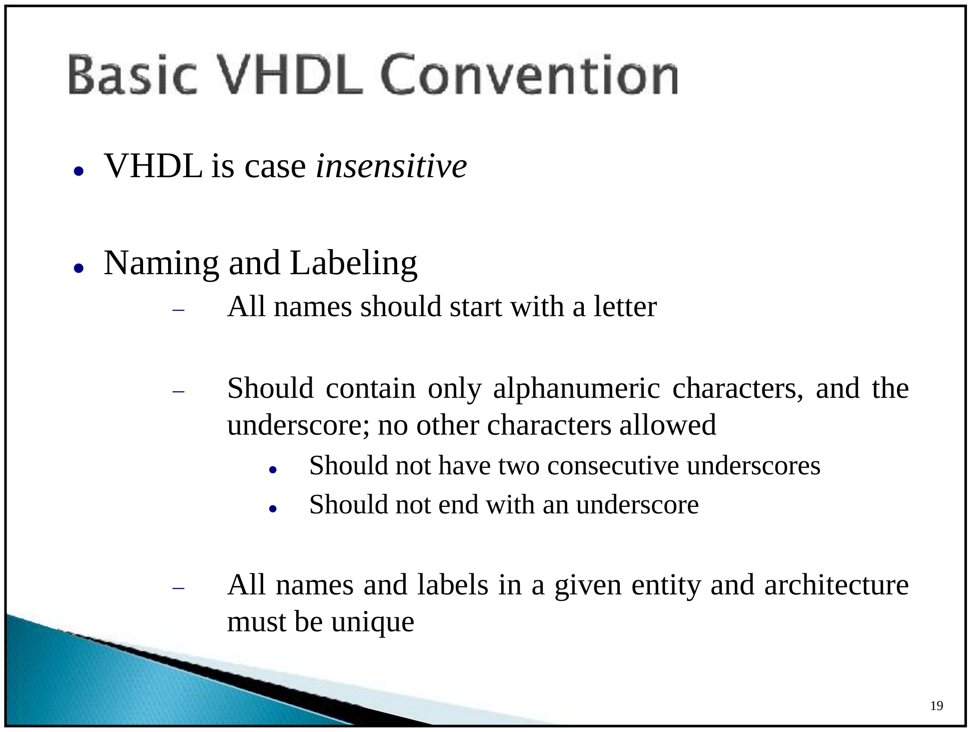
- Bitstream generation
- Burning device

17

Introduction to VHDL

- What is VHDL?
  - **V**ery High Speed Integrated Circuit (VHSIC)
  - **H**ardware
  - **D**escription
  - **L**anguage
- VHDL: a formal language for specifying the behavior and structure of a digital circuit.
- Verilog: another, equally popular, hardware description language (HDL).

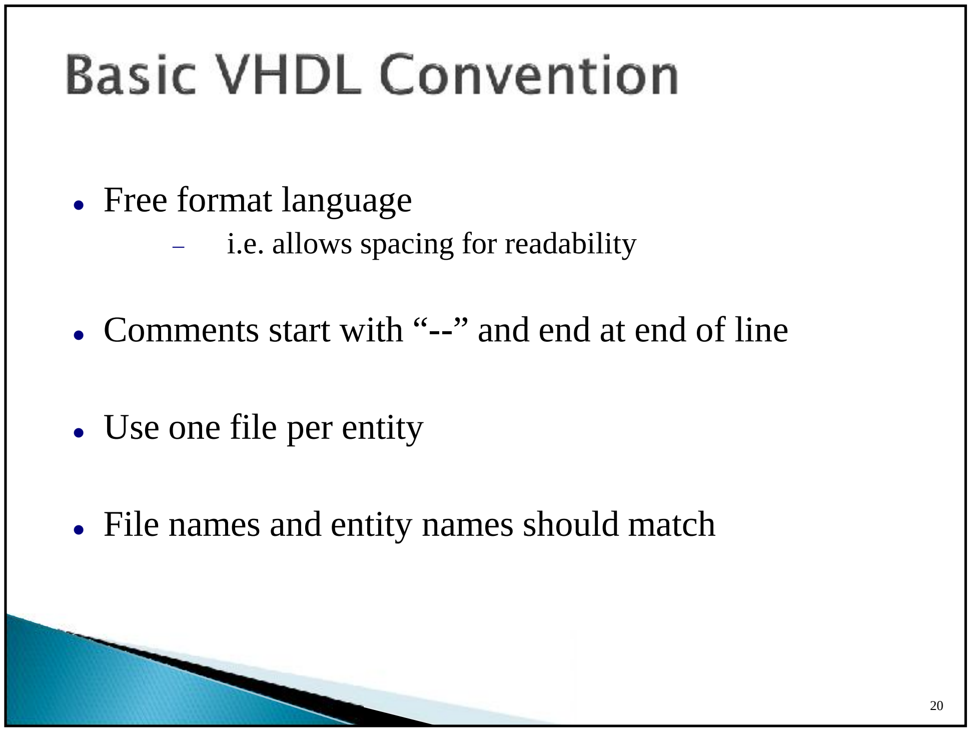
18



Basic VHDL Convention

- VHDL is case *insensitive*
- Naming and Labeling
  - All names should start with a letter
  - Should contain only alphanumeric characters, and the underscore; no other characters allowed
    - Should not have two consecutive underscores
    - Should not end with an underscore
  - All names and labels in a given entity and architecture must be unique

19



Basic VHDL Convention

- Free format language
  - i.e. allows spacing for readability
- Comments start with “--” and end at end of line
- Use one file per entity
- File names and entity names should match

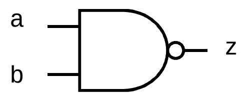
20

Logic Circuits in VHDL

- VHDL description includes two parts
  - Entity statement
  - Architecture statement
- Entity
  - Describes the interface (i.e. inputs and outputs)
- Architecture
  - Describes the circuit implementation

21

Example: NAND Gate

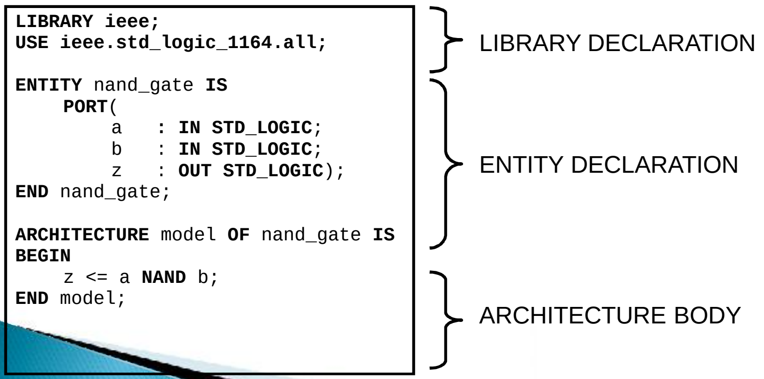


a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

22

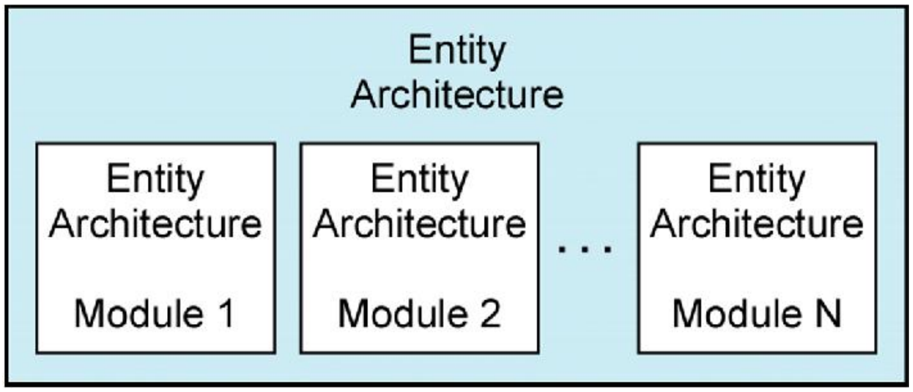
Example VHDL Code

- 3 sections to a piece of VHDL code
- File extension for a VHDL file is .vhd
- Name of the file should be the same as the entity name (nand_gate.vhd) [OpenCores Coding Guidelines]



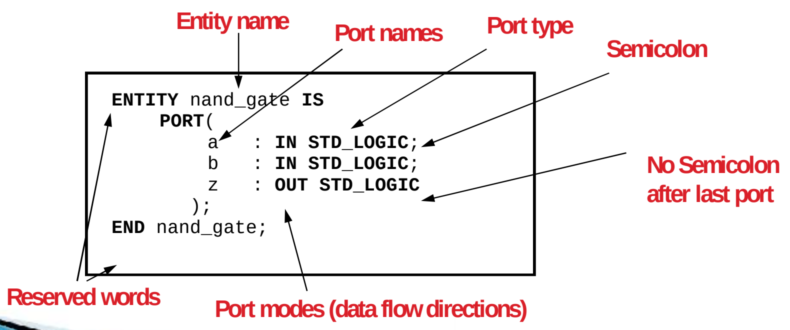
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
    z <= a NAND b;
END model;
```

VHDL Program Structure



Entity Declaration

- Entity Declaration describes the interface of the component, i.e. input and output ports.



```
ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC
       );
END nand_gate;
```

Entity name
Port names
Port type
Semicolon
No Semicolon after last port
Reserved words
Port modes (data flow directions)

25

Entity declaration – simplified syntax

ENTITY entity_name IS
   PORT (
      port_name : port_mode signal_type;
      port_name : port_mode signal_type;
      ………….
      port_name : port_mode signal_type);
END entity_name;

26

Ports: Mode

- IN
  - Driver outside entity; can be read
- OUT
  - Driver inside entity; cannot be read
- INOUT
  - Driver inside and outside entity; can be read
- BUFFER
  - Driver inside entity; can be read

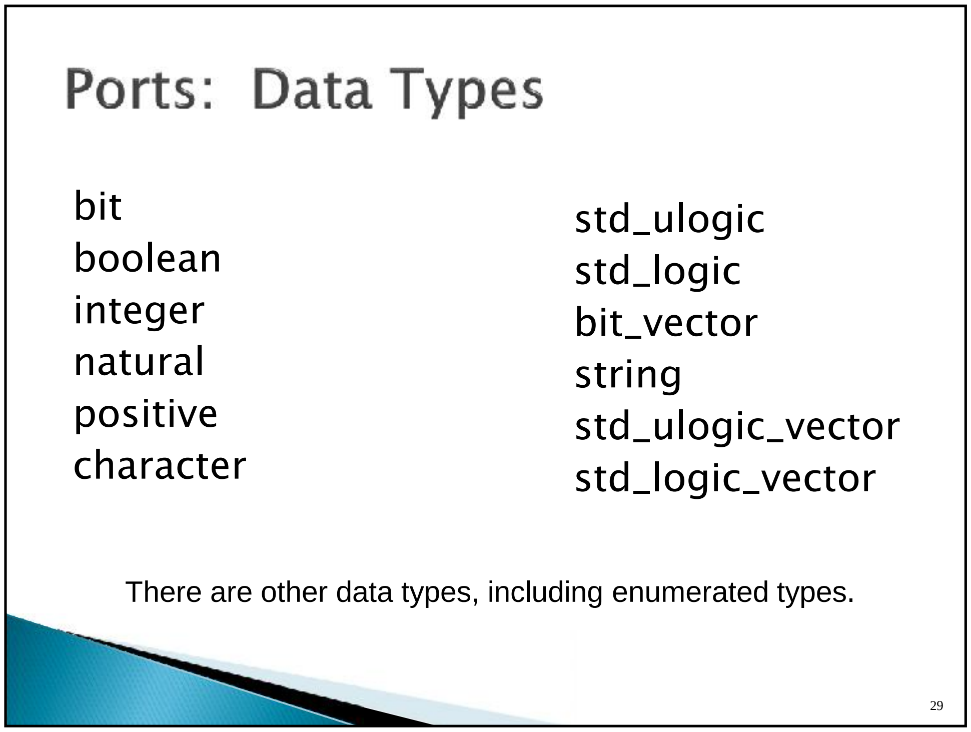
Port Modes – Summary

The *Port Mode* of the interface describes the direction in which data travels with respect to the c*omponent*

- **In**: Data comes into this port and can only be read within the entity. It can appear **only on the right side** of a signal or variable assignment.

- **Out**: The value of an output port can only be updated within the entity. **It cannot be read**. It can only appear **on the left side** of a signal assignment.

- **Inout**: The value of a bi-directional port can be read and updated within the entity model. It can appear on **both sides** of a signal assignment.

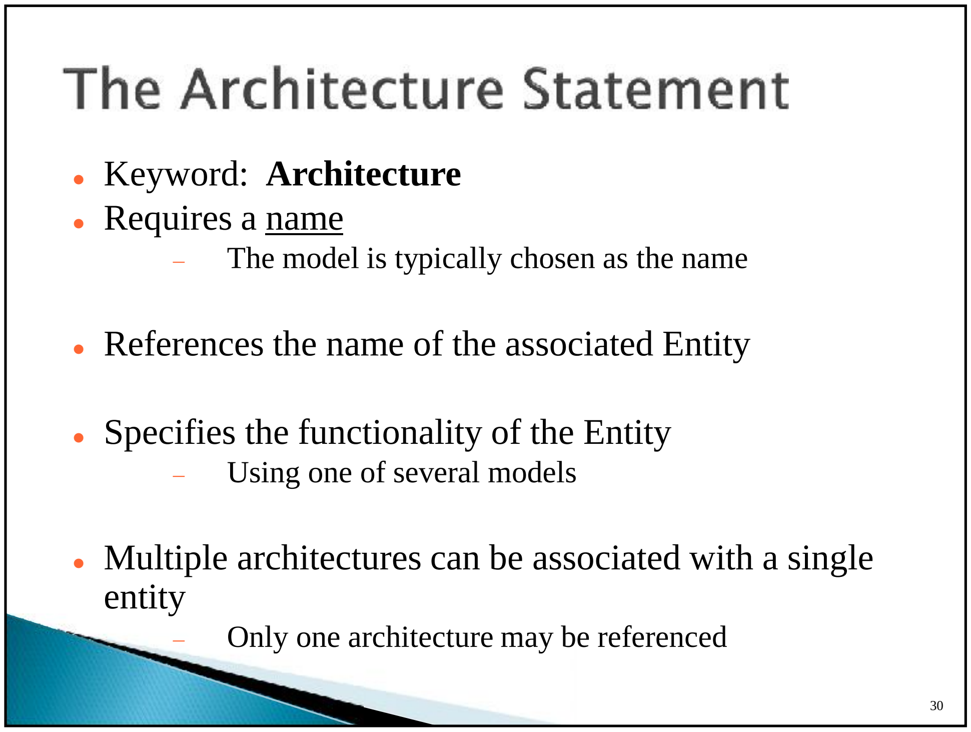


Ports: Data Types

bit	std_ulogic
boolean	std_logic
integer	bit_vector
natural	string
positive	std_ulogic_vector
character	std_logic_vector

There are other data types, including enumerated types.

29



The Architecture Statement

- Keyword: **Architecture**
- Requires a name
  - The model is typically chosen as the name
- References the name of the associated Entity
- Specifies the functionality of the Entity
  - Using one of several models
- Multiple architectures can be associated with a single entity
  - Only one architecture may be referenced

30

The Architecture Statement

Associated with each entity is one or more architecture declarations of the form

architecture architecture-name **of** entity-name **is**
  [declarations]
begin
  architecture body
end [**architecture**] [architecture-name];

In the declarations section, we can declare signals and components that are used within the architecture. The architecture body contains statements that describe the operation of the module.

31

Entity Declaration & Architecture

nand_gate.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```

32

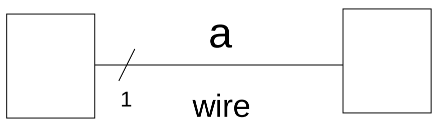
Signals

- Can be wires or buses (groups of wires)
  - Wire
    - SIGNAL a: STD_LOGIC;
  - Bus (with 8 wires)
    - SIGNAL b8: STD_LOGIC_VECTOR(7 DOWNTO 0);
  - Bus (with 16 wires)
    - SIGNAL b16: STD_LOGIC_VECTOR(15 DOWNTO 0);
- Used to interconnect entities and components

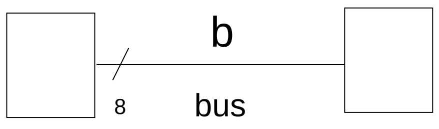
33

Signals

SIGNAL a : STD_LOGIC;



SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



34

Standard Logic Vectors

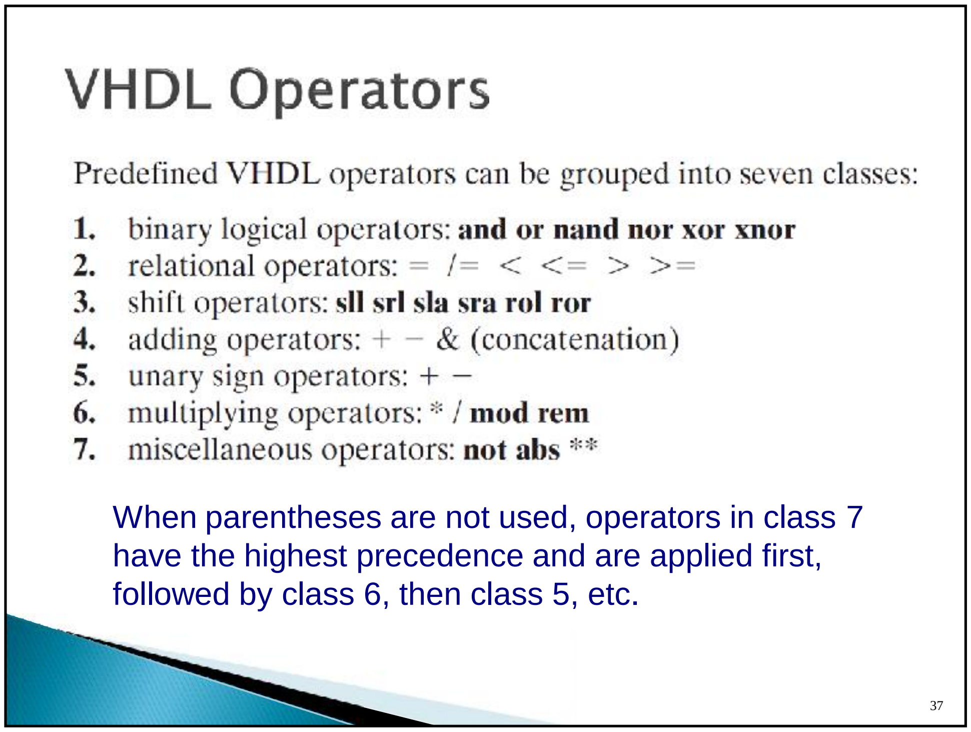
```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL e: STD_LOGIC_VECTOR(8 DOWNTO 0);
                                   ……….
a <= '1';
b <= "0000";            -- Binary base assumed by
default
c <= B"0000";           -- Binary base explicitly specified
d <= X"AF67";           -- Hexadecimal base
e <= O"723";            -- Octal base
```

35

Vectors and Concatenation

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

a <= "0000";
b <= "1111";
c <= a & b;               -- c = "00001111"

d <= '0' & "0001111";    -- d <= "00001111"

e <= '0' & '0' & '0' & '0' & '1' & '1' &
     '1' & '1';
                          -- e <= "00001111"
```

36



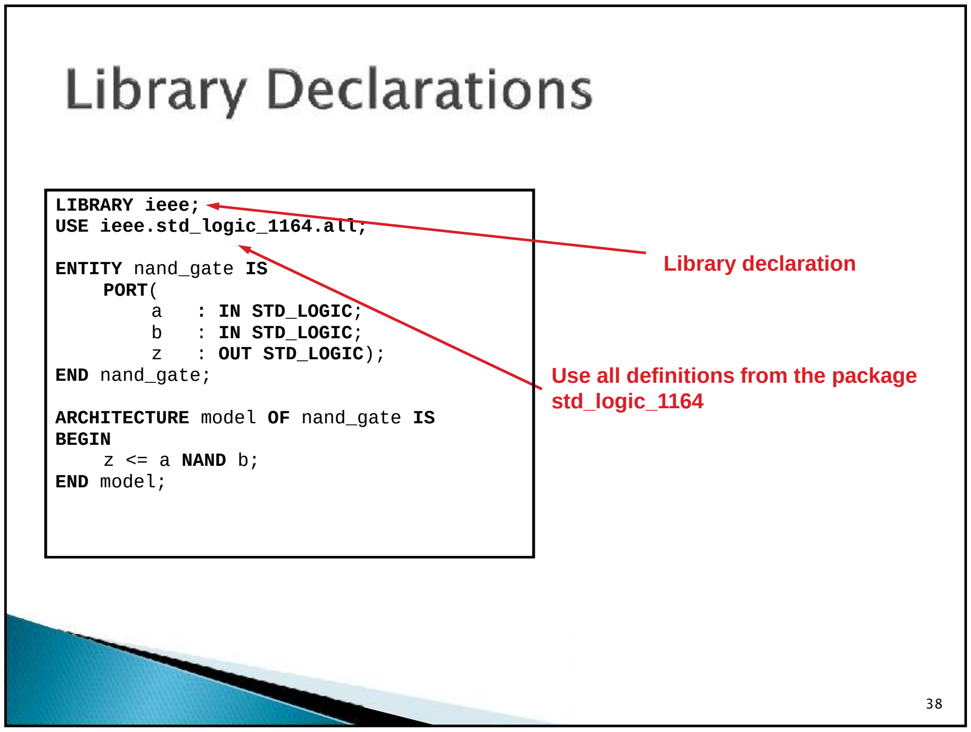
VHDL Operators

Predefined VHDL operators can be grouped into seven classes:

1. binary logical operators: **and or nand nor xor xnor**
2. relational operators: = /= < <= > >=
3. shift operators: **sll srl sla sra rol ror**
4. adding operators: + − & (concatenation)
5. unary sign operators: + −
6. multiplying operators: * / **mod rem**
7. miscellaneous operators: **not abs** **

When parentheses are not used, operators in class 7 have the highest precedence and are applied first, followed by class 6, then class 5, etc.

37



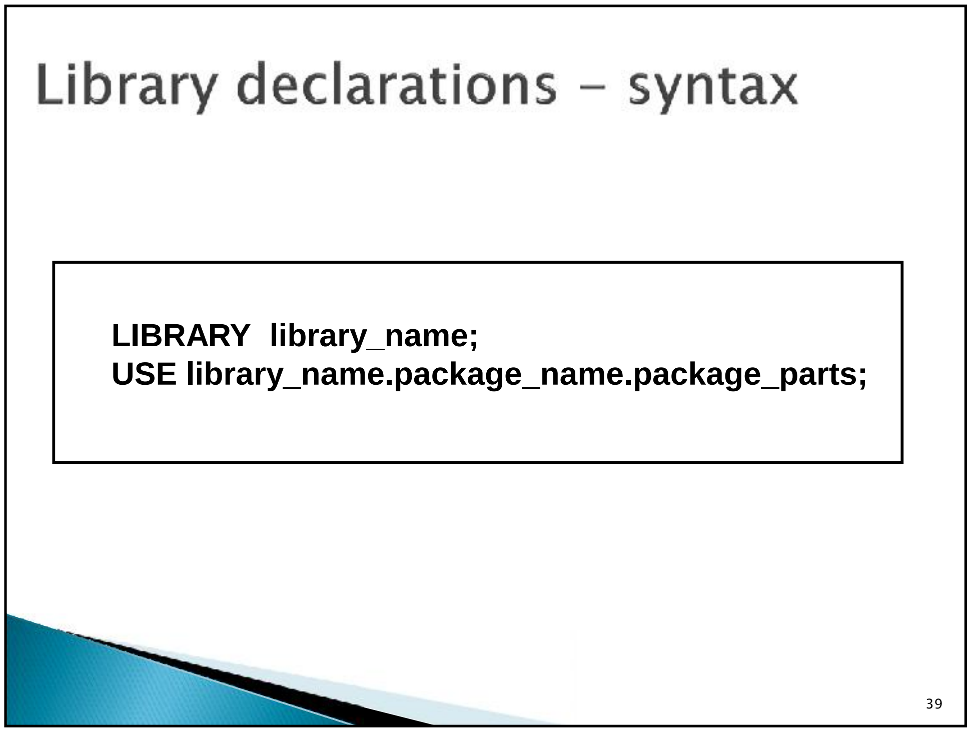
Library Declarations

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
    z <= a NAND b;
END model;
```

Library declaration

Use all definitions from the package std_logic_1164

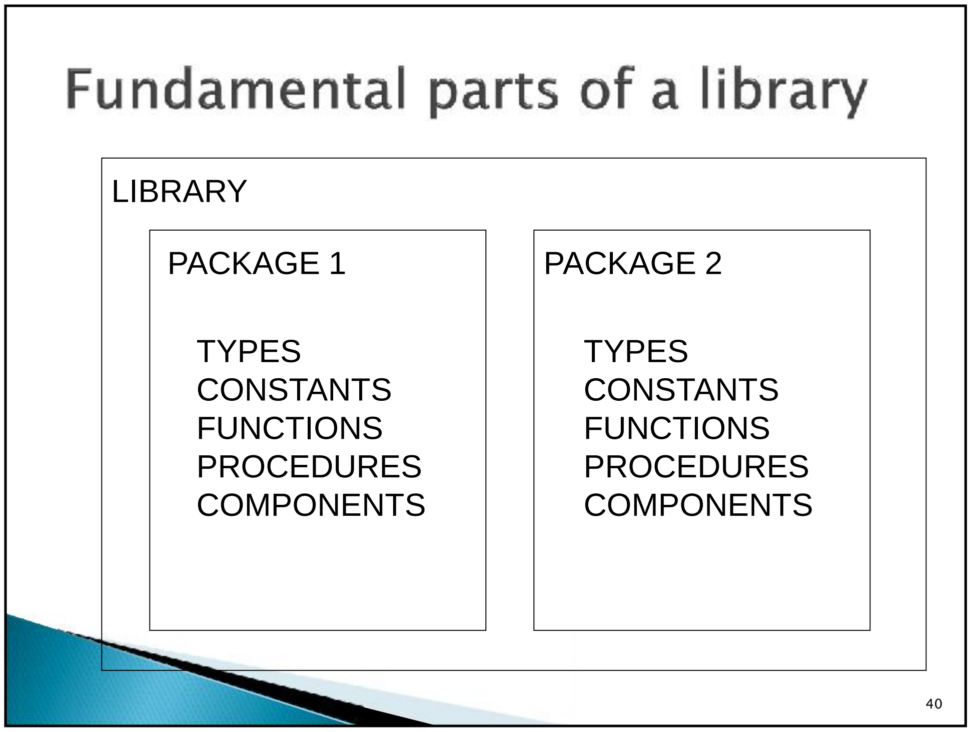
38



Library declarations – syntax

```
LIBRARY  library_name;
USE library_name.package_name.package_parts;
```

39



Fundamental parts of a library

LIBRARY

PACKAGE 1

TYPES
CONSTANTS
FUNCTIONS
PROCEDURES
COMPONENTS

PACKAGE 2

TYPES
CONSTANTS
FUNCTIONS
PROCEDURES
COMPONENTS

40

Libraries

- ieee
  Specifies multi-level logic system, including STD_LOGIC, and STD_LOGIC_VECTOR data types

Need to be explicitly declared

- std
  Specifies pre-defined data types (BIT, BOOLEAN, INTEGER, REAL, SIGNED, UNSIGNED, etc.), arithmetic operations, basic type conversion functions, basic text i/o functions, etc.
- work
  Holds current designs after compilation

Visible by default

41

STD_LOGIC

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```

What is **STD_LOGIC** you ask?

42

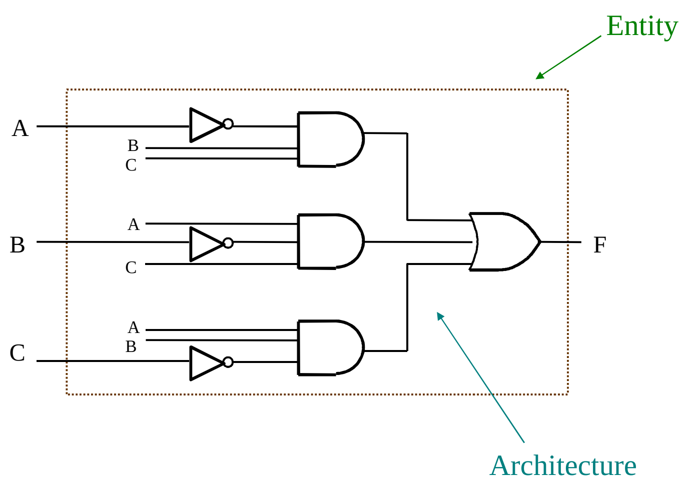
BIT versus STD_LOGIC

- BIT type can only have a value of ‘0’ or ‘1’
- STD_LOGIC can have nine values
  - ’U’,’X’,‘0’,’1’,’Z’,’W’,’L’,’H’,’-’
  - Useful mainly for simulation
  - ‘0’,’1’, and ‘Z’ are synthesizable (your codes should contain only these three values)

STD_LOGIC *type* demystified

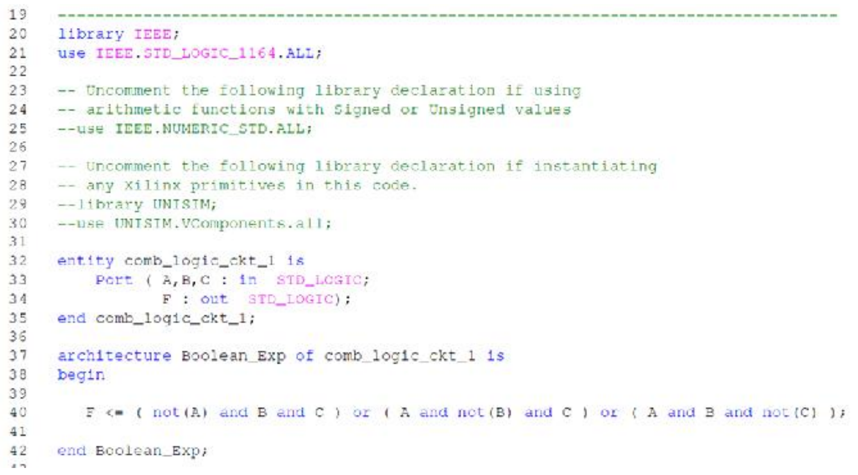
Value	Meaning
‘U’	Uninitialized
‘X’	Forcing (Strong driven) Unknown
‘0’	Forcing (Strong driven) 0
‘1’	Forcing (Strong driven) 1
‘Z’	High Impedance
‘W’	Weak (Weakly driven) Unknown
‘L’	Weak (Weakly driven) 0. Models a pull down.
‘H’	Weak (Weakly driven) 1. Models a pull up.
‘-’	Don’t Care

VHDL Example



45

VHDL Example

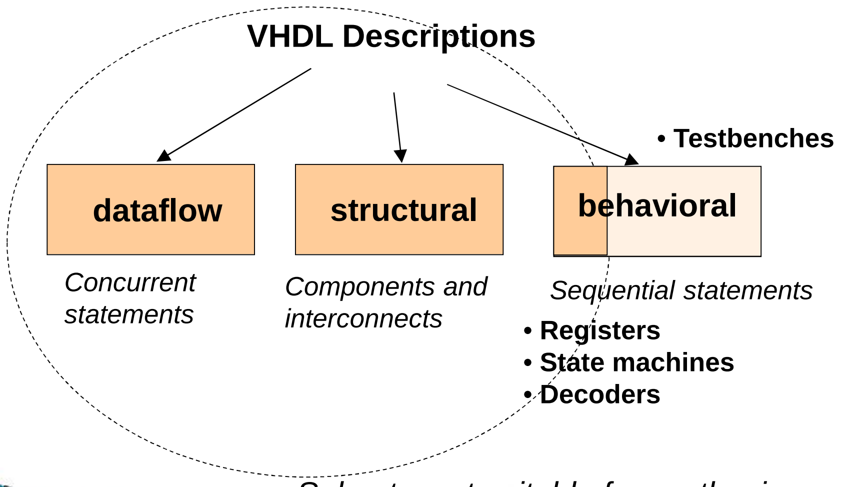


```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity comb_logic_ckt_1 is
    Port ( A,B,C : in  STD_LOGIC;
           F : out  STD_LOGIC);
end comb_logic_ckt_1;

architecture Boolean_Exp of comb_logic_ckt_1 is
begin

   F <= ( not(A) and B and C ) or ( A and not(B) and C ) or ( A and B and not(C) );

end Boolean_Exp;
```

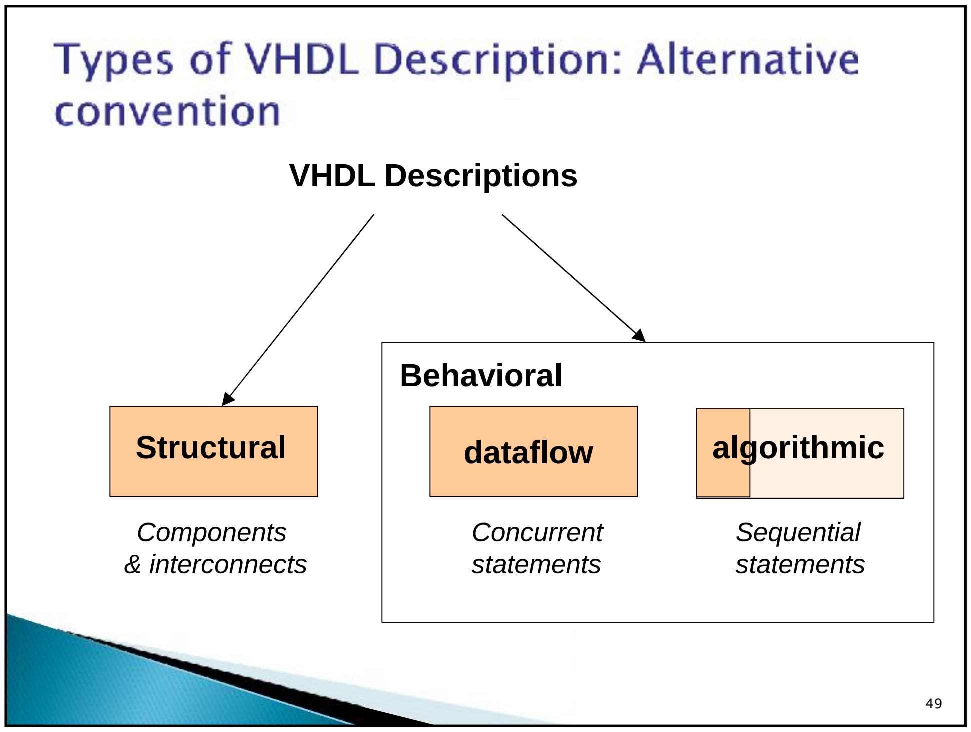
46

Types of VHDL Description (Modeling Styles)

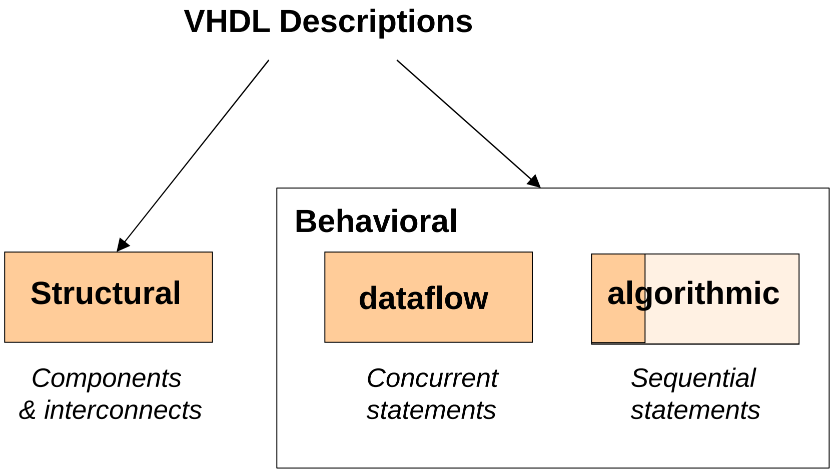
Types of VHDL Description: Convention used in this class

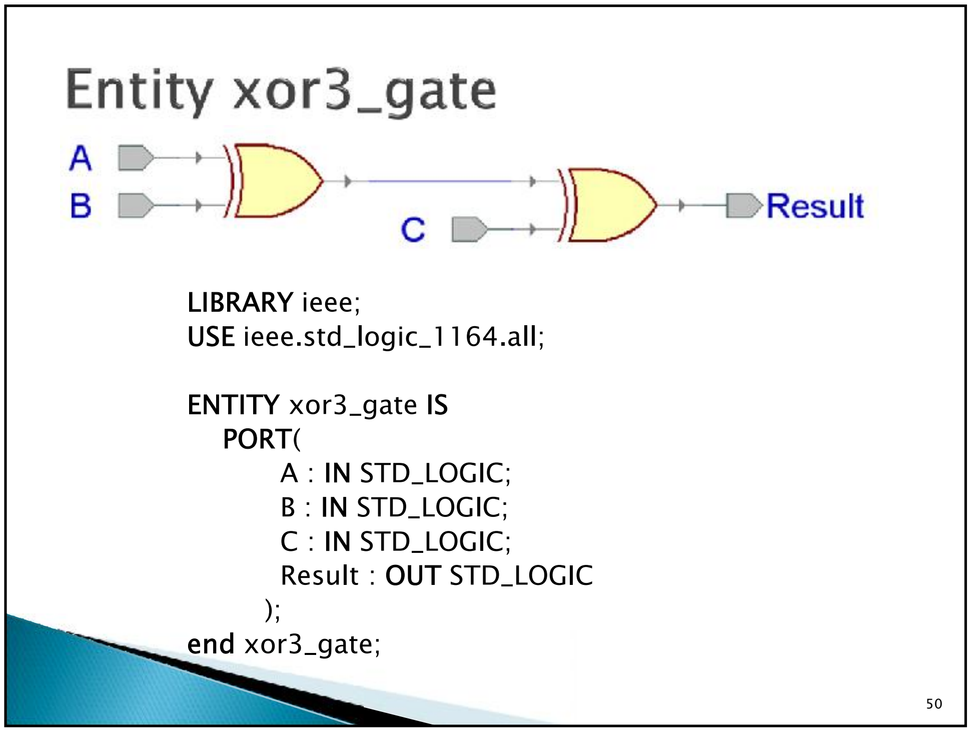


Subset most suitable for synthesis

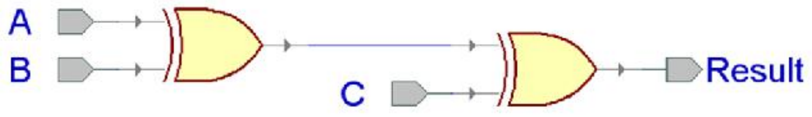


Types of VHDL Description: Alternative convention





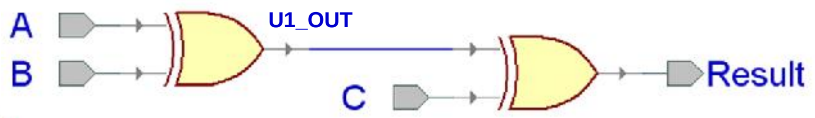
Entity xor3_gate



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor3_gate IS
    PORT(
            A : IN STD_LOGIC;
            B : IN STD_LOGIC;
            C : IN STD_LOGIC;
            Result : OUT STD_LOGIC
          );
end xor3_gate;
```

Dataflow Architecture (xor3_gate)

```
ARCHITECTURE dataflow OF xor3_gate IS
SIGNAL U1_OUT: STD_LOGIC;
BEGIN
        U1_OUT <= A XOR B;
        Result  <= U1_OUT XOR C;
END dataflow;
```



Dataflow Description

- Describes how data moves through the system and the various processing steps.
  - Dataflow uses series of concurrent statements to realize logic.
  - Dataflow is the most useful style to describe combinational logic
  - Dataflow code also called “concurrent” code
- Concurrent statements are evaluated at the same time; thus, the order of these statements doesn’t matter
  - This is not true for sequential/behavioral statements

This order…

```
U1_out <= A XOR B;
Result <= U1_out XOR C;
```

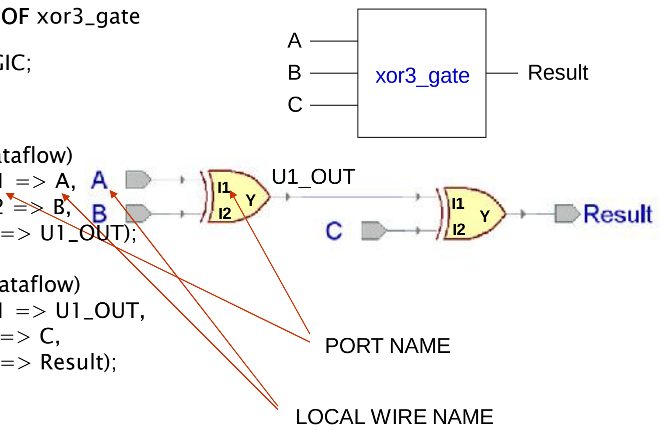
Is the same as this order…

```
Result <= U1_out XOR C;
U1_out <= A XOR B;
```

52

Structural Architecture in VHDL 93

```
ARCHITECTURE structural OF xor3_gate
   IS
SIGNAL U1_OUT: STD_LOGIC;

BEGIN
   U1: entity work.xor2(dataflow)
               PORT MAP (I1  => A,
                         I2  => B,
                         Y   => U1_OUT);

   U2:  entity work.xor2(dataflow)
               PORT MAP (I1  => U1_OUT,
                         I2  => C,
                         Y   => Result);

END structural;
```

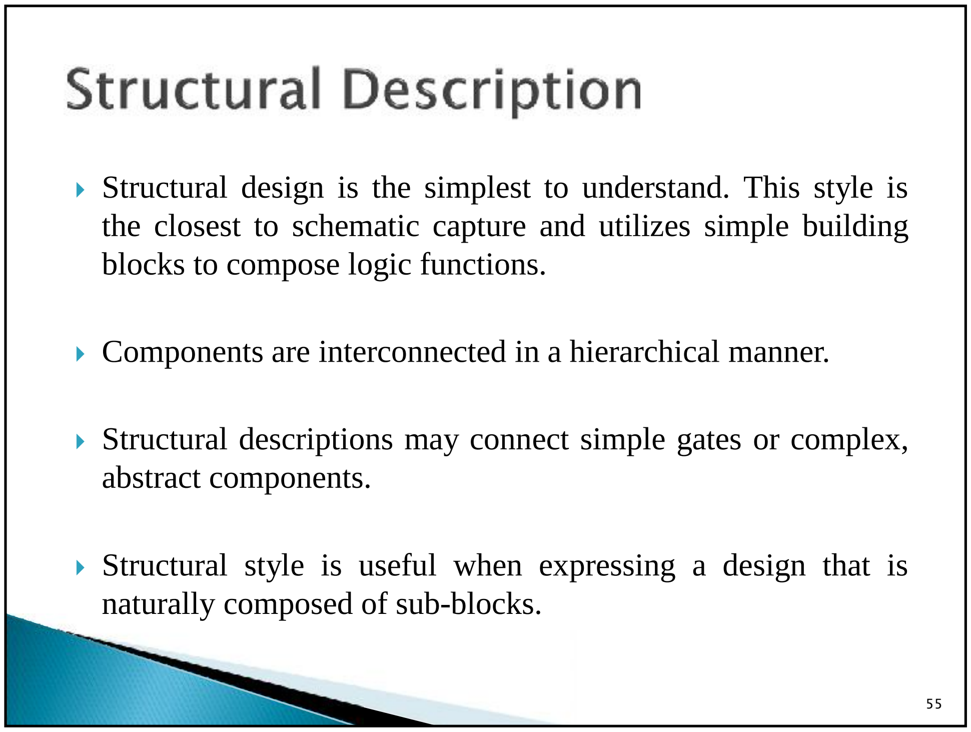


xor2

xor2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor2 IS
    PORT(
          I1   : IN STD_LOGIC;
          I2   : IN STD_LOGIC;
          Y    : OUT STD_LOGIC);
END xor2;

ARCHITECTURE dataflow OF xor2 IS
BEGIN
    Y <= I1 xor I2;
END dataflow;
```

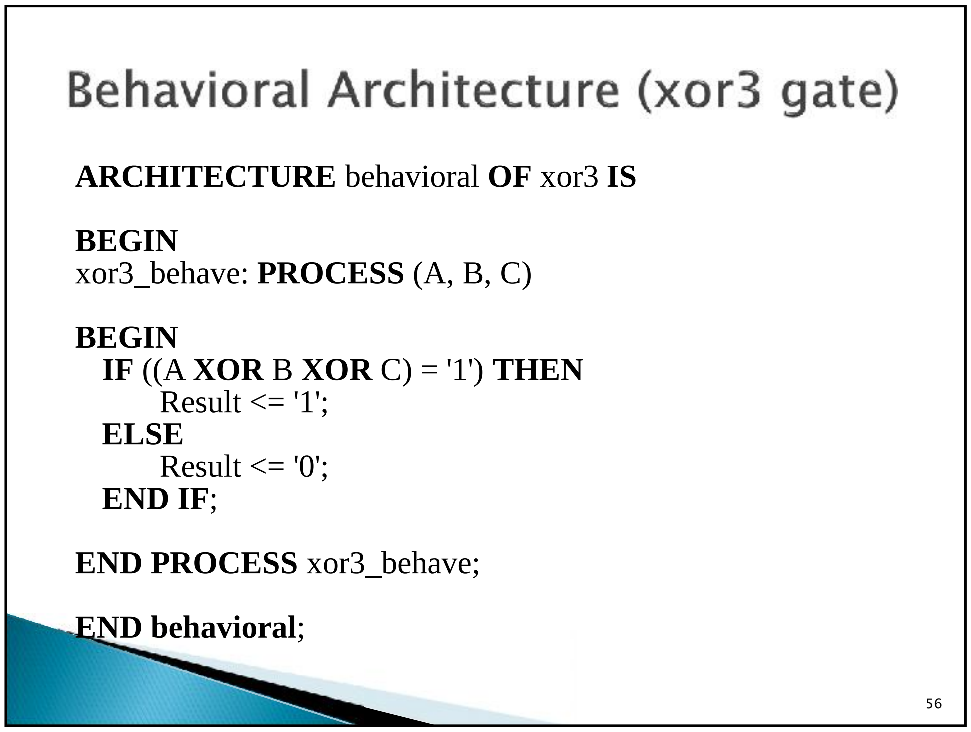
54



Structural Description

- Structural design is the simplest to understand. This style is the closest to schematic capture and utilizes simple building blocks to compose logic functions.
- Components are interconnected in a hierarchical manner.
- Structural descriptions may connect simple gates or complex, abstract components.
- Structural style is useful when expressing a design that is naturally composed of sub-blocks.

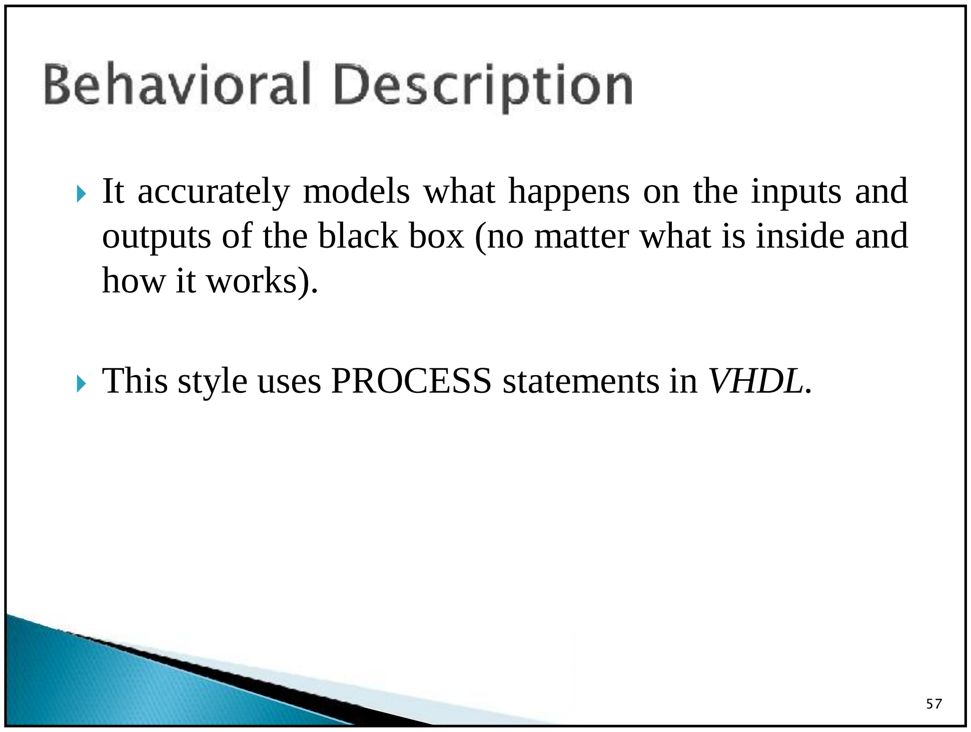
55



Behavioral Architecture (xor3 gate)

```
ARCHITECTURE behavioral OF xor3 IS

BEGIN
xor3_behave: PROCESS (A, B, C)

BEGIN
  IF ((A XOR B XOR C) = '1') THEN
       Result <= '1';
  ELSE
       Result <= '0';
  END IF;

END PROCESS xor3_behave;

END behavioral;
```

56



Behavioral Description

- It accurately models what happens on the inputs and outputs of the black box (no matter what is inside and how it works).
- This style uses PROCESS statements in *VHDL*.

57