CDA 4253 FPGA System Design
Introduction to VHDL

Hao Zheng
Dept of Comp Sci & Eng
USF

Reading

- P. Chu, ***FPGA Prototyping by VHDL Examples***
  - *Chapter 1, Gate-level combinational circuits*
- Xilinx XST User Guide
  - Xilinx specific language support
- Two purposes of using VHDL:
  - **Simulation**
  - **Synthesis** – focus of this course

Recommended reading

- Wikipedia – The Free On-line Encyclopedia

VHDL **-** ***http://en.wikipedia.org/wiki/VHDL***

Verilog **-** ***http://en.wikipedia.org/wiki/Verilog***

VHDL

- VHDL is a language for describing digital logic systems used by industry worldwide

VHDL is an acronym for **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage

- Now, there are extensions to describe analog designs.

Subsequent versions of VHDL

- IEEE-1076 1987
- IEEE-1076 1993 ← most commonly supported by CAD tools
- IEEE-1076 2000 (minor changes)
- IEEE-1076 2002 (minor changes)
- IEEE-1076 2008

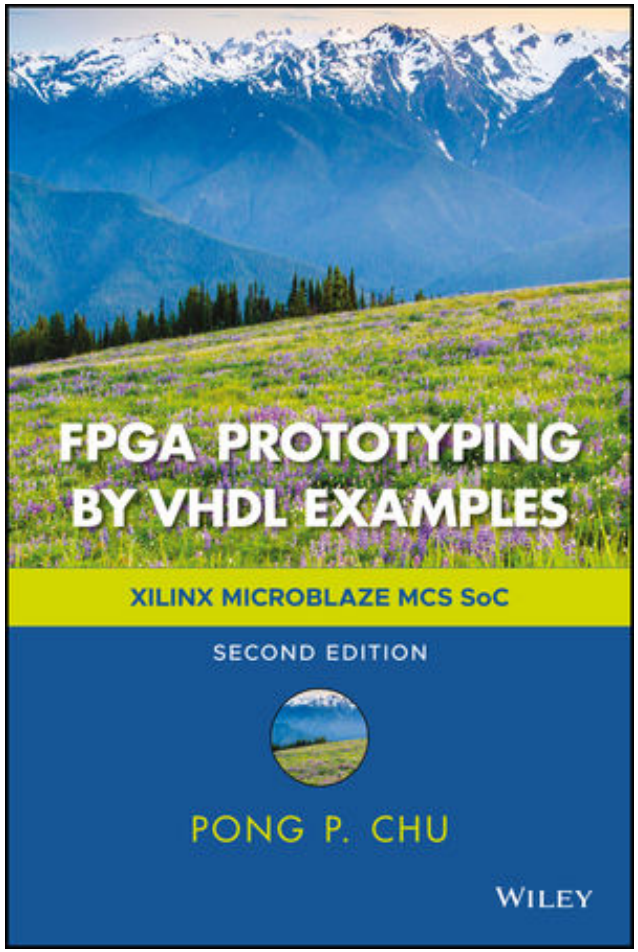
VHDL vs. Verilog

Government Developed	Commercially Developed
Ada based	C based
Strongly Type Cast	Mildly Type Cast
Case-insensitive	Case-sensitive
Difficult to learn	Easier to Learn
More Powerful	Less Powerful

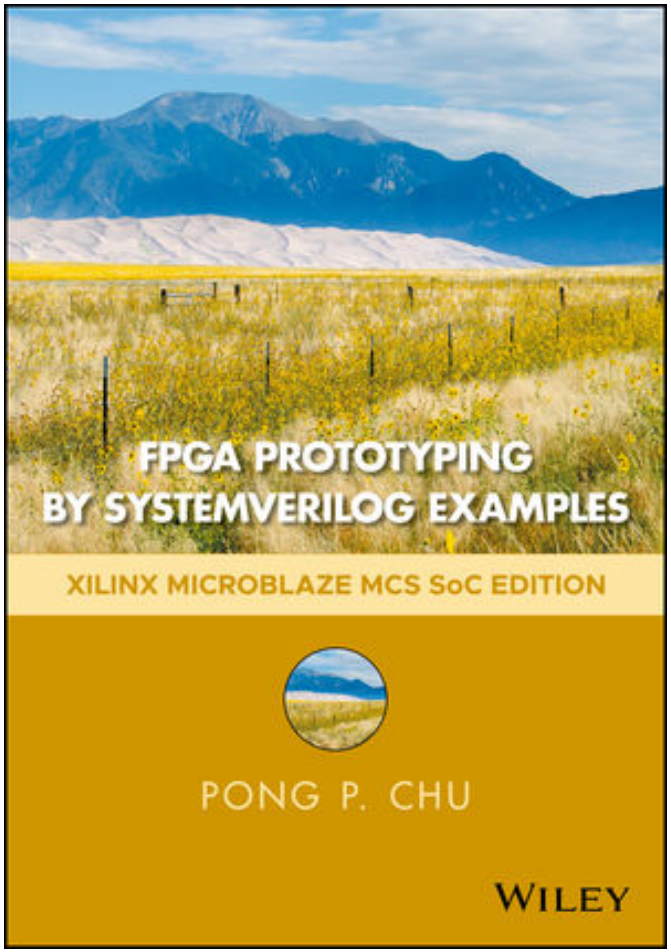
Features of VHDL/Verilog

- Technology/vendor independent
- Portable
- Reusable

Good VHDL/Verilog Books



coming soon



VHDL Fundamentals

Naming and Labeling (1)

- VHDL is case insensitive.

  Example:

  Names or labels

  `databus`

  `Databus`

  `DataBus`

  `DATABUS`

  are all equivalent

- Avoid inconsistent styles

Naming and Labeling (2)

General rules of thumb (according to VHDL-87)

1. All names should start with an alphabet character (a-z or A-Z)
2. Use only alphabet characters (a-z or A-Z) digits (0-9) and underscore (_)
3. Do not use any punctuation or reserved characters within a name (!, ?, ., &, +, -, etc.)
4. Do not use two or more consecutive underscore characters (_ _) within a name (e.g., Sel_ _A is invalid)
5. No forward slashes “/” in names.
6. All names and labels in a given entity and architecture must be unique.

Extended Identifiers

Allowed only in VHDL-93 and higher:

1. Enclosed in backslashes
2. May contain spaces and consecutive underscores
3. May contain punctuation and reserved characters within a name (!, ?, ., &, +, -, etc.)
4. VHDL keywords allowed
5. Case sensitive

Examples:

\rdy\ \My design\ \!a\
\RDY\ \my design\ \-a\

Should not be used to avoid confusioin!

Literals

- Numeric: 32, -16, 3.1415927
- Bits : ‘1’, ‘0’
- Strings: “Hello”
- Bit strings: B”1111_1111”, O”353”, X”AA55”
- Concatenation: “1111” & “0000” => “1111_0000”

Objects

- **Signal** – model real physical wires for communications
  - Or physical storage of information

- **Variable** – a programming construct to model temporary storage

- **Constant** – its value never changes after initialization

Comments

- Comments in VHDL are indicated with a "double dash", i.e., "--"
  - Comment indicator can be placed anywhere in the line
  - Any text that follows in the same line is treated as a comment
  - Carriage return terminates a comment
  - No method for commenting a block extending over a couple of lines
- Examples:

```
-- main subcircuit
Data_in <= Data_bus; -- reading data from the input FIFO
```

Comments

- Explain function of module to other designers
- Explanatory, not Just restatement of code
- Placed close to the code described
  - Put near executable code, not just in a header

Free Format

- VHDL is a “free format” language

  No formatting conventions, such as spacing or indentation imposed by VHDL compilers. Space, tabs, and carriage return treated the same way.

  Example:

```
if (a=b) then
```

or

```
if      (a=b)           then
```

or

```
if (a =
b) then
```

are all equivalent

Readability Standards & Coding Style

Adopt readability standards based on the textbook by Chu

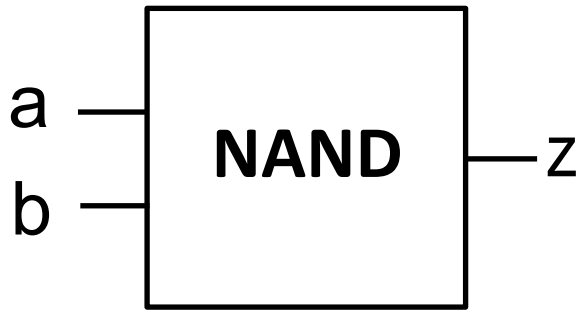
Use coding style recommended in OpenCores Coding Guidelines

Available at the course web page

Penalty may be enforced for not following these recommendations!!!

Describing Designs

Example: NAND Gate



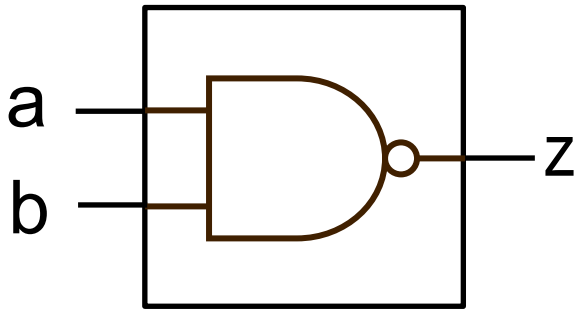
Design name

and

Interface

```
entity nand_gate is
    port(     a   : in STD_LOGIC;
              b   : in STD_LOGIC;
          z   : out STD_LOGIC);
end nand_gate;
```

Example: NAND Gate – Function



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

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

Example VHDL Code

- 3 sections of VHDL code to describe a design.
- 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*]

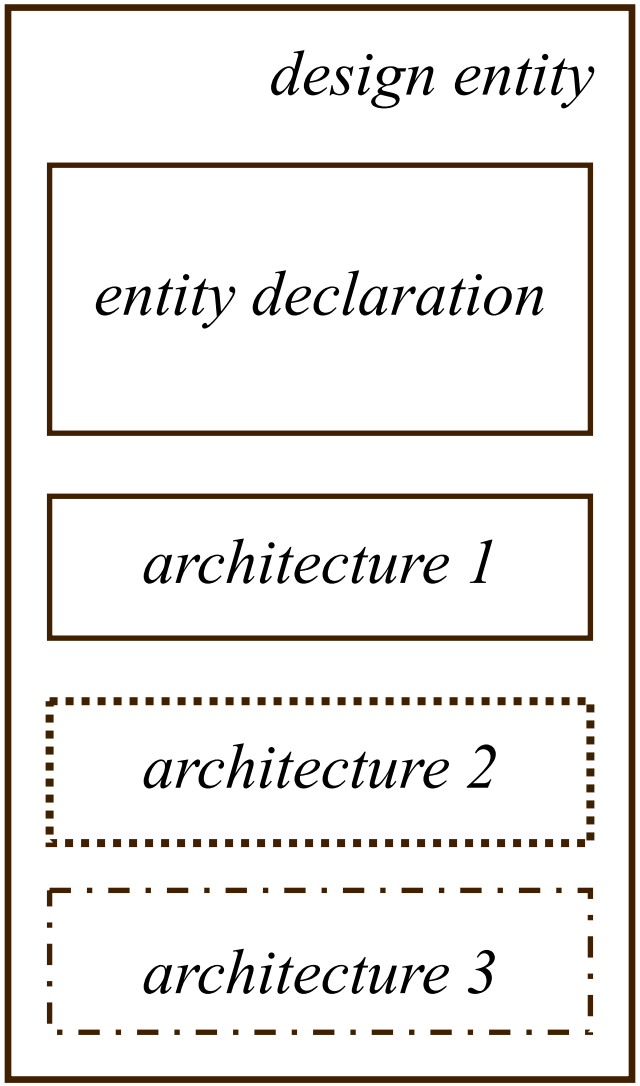
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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;
```

LIBRARY DECLARATION

ENTITY DECLARATION

ARCHITECTURE BODY

Design Entity

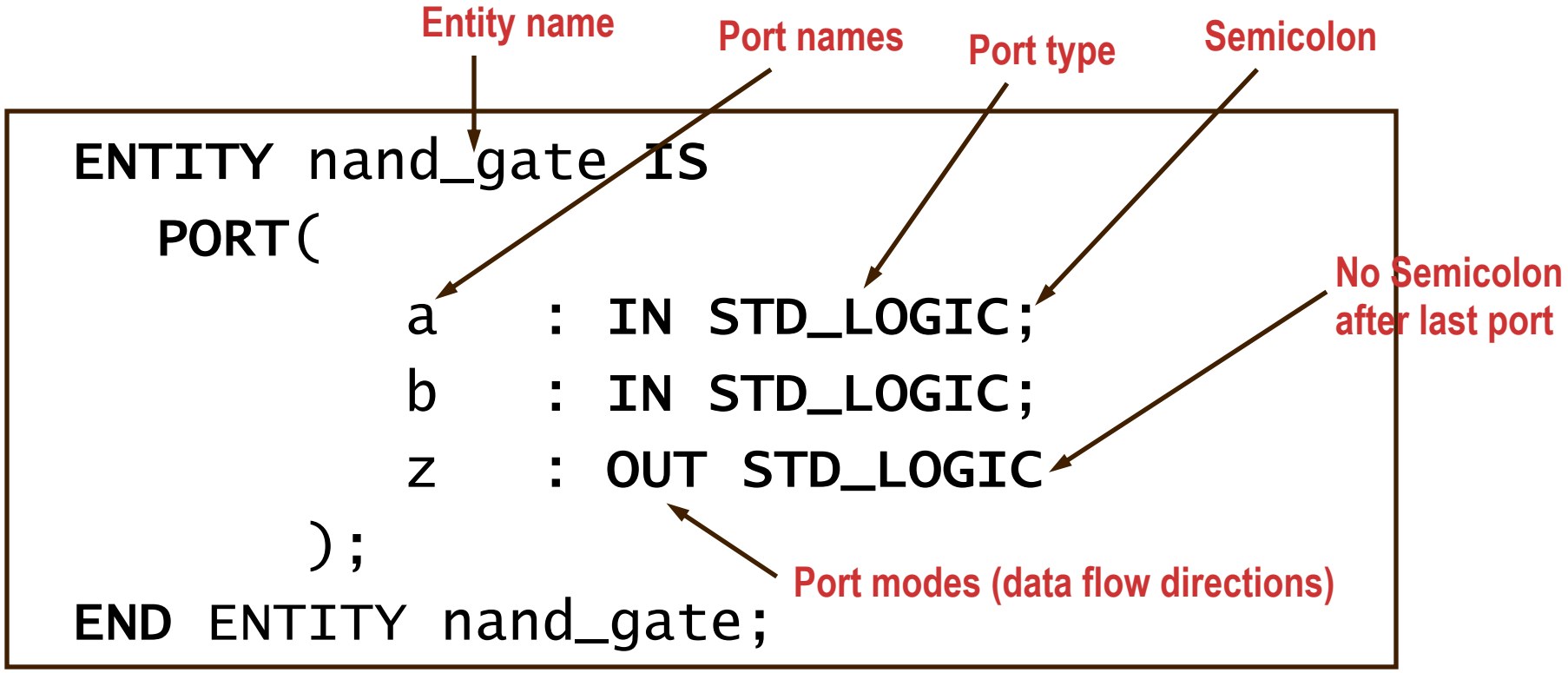


Design Entity - most basic building block of a design.

One *entity* can have many different *architectures.*

Entity Declaration

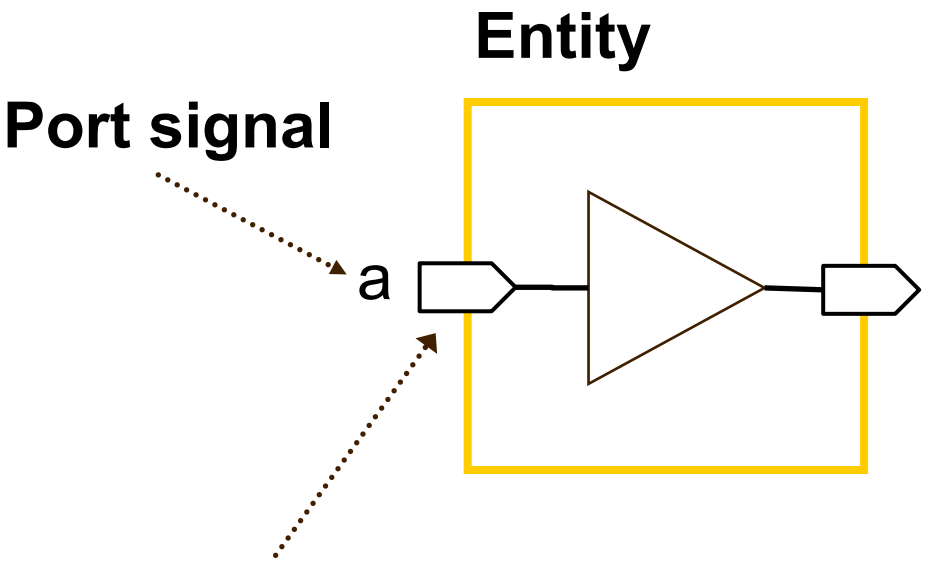
- Entity Declaration describes an interface of the component, i.e. input and output ports.



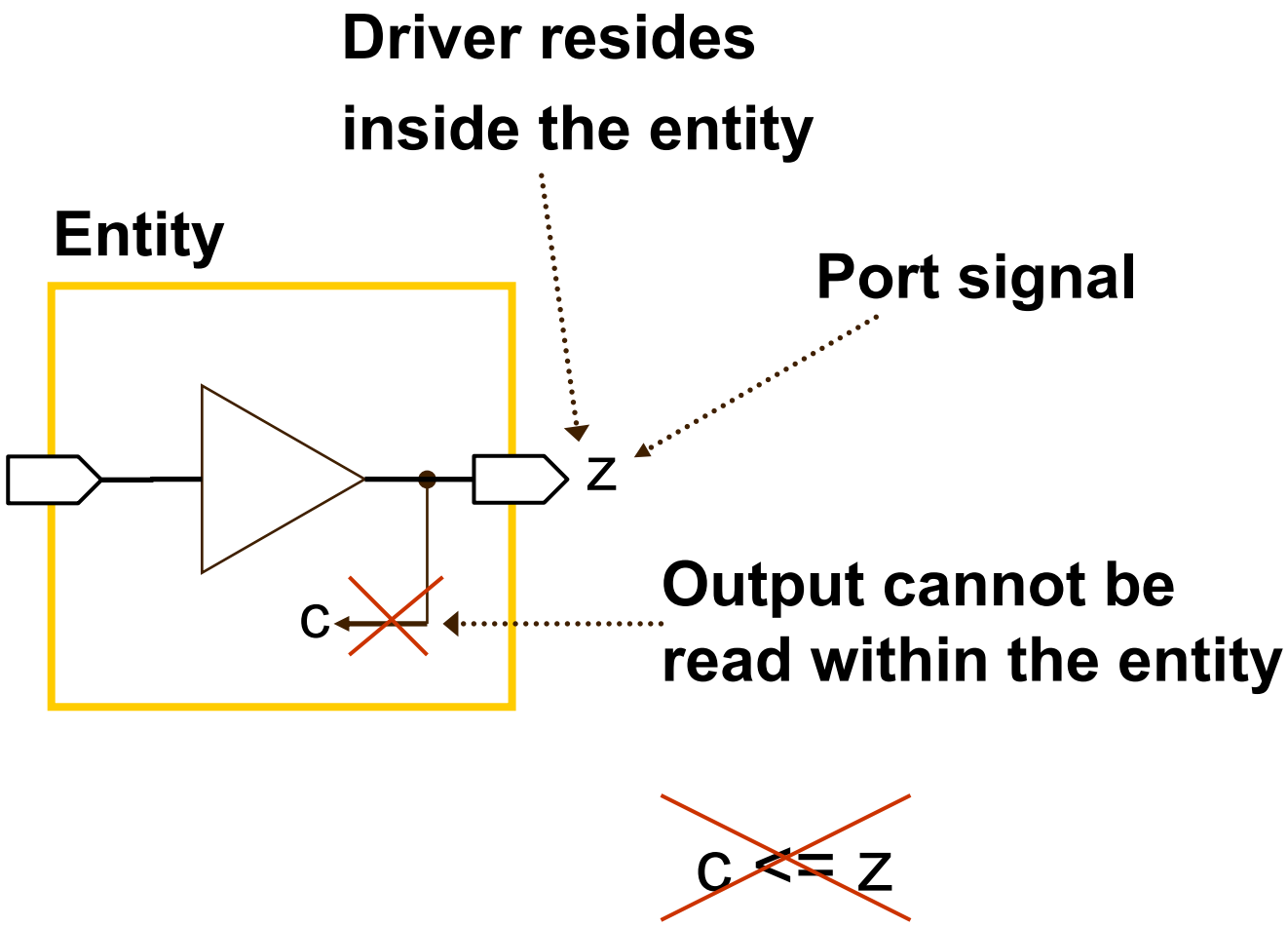
Entity Declaration – Simplified Syntax

```
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 entity_name;
```

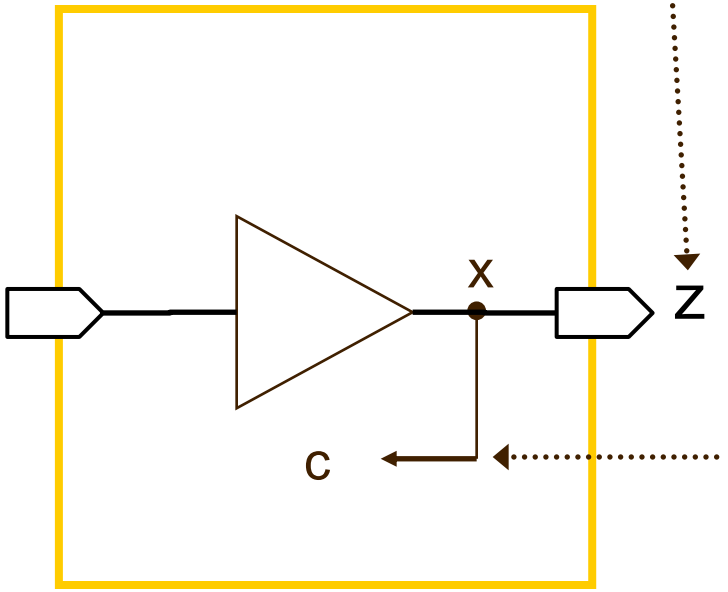
Port Mode – IN



Port Mode – OUT



Port Mode – OUT (with Extra Signal)



Driver resides inside the entity

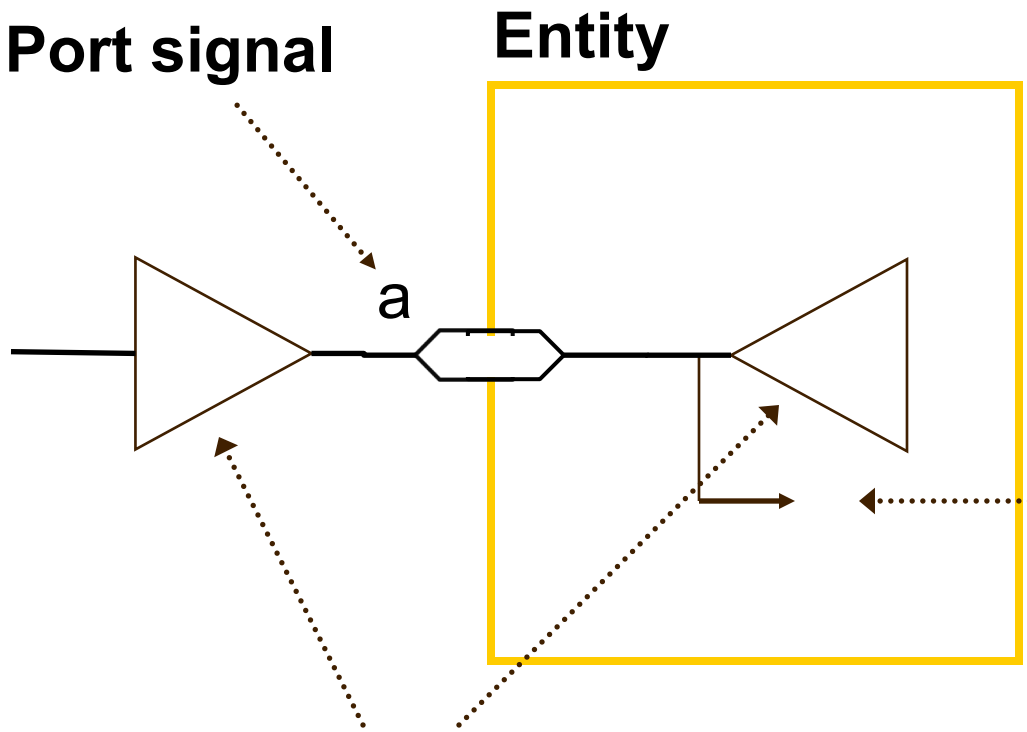
Port signal

Signal x can be read inside the entity

z <= x

c <= x

Port Mode – INOUT



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.

Architecture (Architecture body)

- Describes an implementation of a design entity
- Architecture example:

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

Logic operators:

NOT, AND, OR, NAND, NOR, XOR, XNOR

Architecture – Simplified Syntax

```
ARCHITECTURE architecture_name OF entity_name IS
    Declarations
BEGIN
    Concurrent statements
END [ARCHITECTURE] architecture_name;
```

Entity Declaration & Architecture

```
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 ENTITY nand_gate;

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

Tips & Hints

Place each entity in a different file.

The name of each file should be exactly the same as the name of an entity it contains.

These rules are not enforced by all tools
but are worth following in order to increase
readability and portability of your designs

Tips & Hints

Place the declaration of each port,
signal, constant, and variable
in a separate line
for better readability

These rules are not enforced by all tools
but are worth following in order to increase
readability and portability of your designs

Libraries

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 ENTITY nand_gate;

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

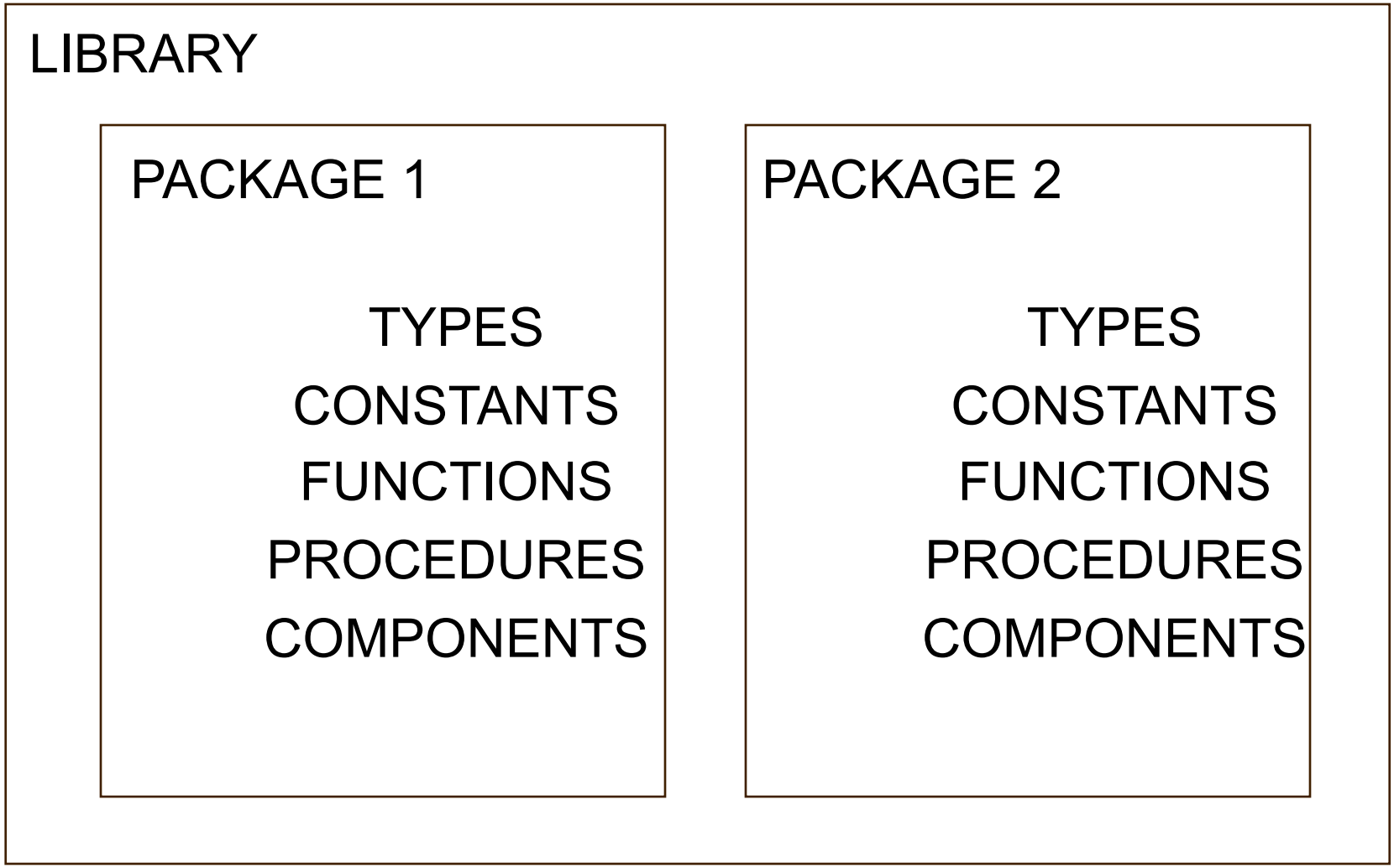
Library declaration

Use all definitions from the package std_logic_1164

Library Declarations – Syntax

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

Structure of a Library



Libraries

- **ieee** Specifies digital logic system, including `STD_LOGIC`, and `STD_LOGIC_VECTOR` types

Need to be explicitly declared

- **std** Specifies built-in data types (`BIT`, `BOOLEAN`, `INTEGER`, `REAL`, `SIGNED`, `UNSIGNED`, etc.), arithmetic operations, basic type conversion functions, basic text i/o functions, etc.

  Holds current designs after compilation

Visible by default

- **work** – current working directory

Operators in Standard VHDL

operator	description	data type of operand a	data type of operand b	data type of result
`a ** b`	exponentiation	`integer`	`integer`	`integer`
`abs` `a`	absolute value	`integer`		`integer`
`not` `a`	negation	`boolean`, `bit`, `bit_vector`		`boolean`, `bit`, `bit_vector`
`a * b`	multiplication	`integer`	`integer`	`integer`
`a / b`	division			
`a` **`mod`** `b`	modulo			
`a` **`rem`** `b`	remainder			
`+ a`	identity	`integer`		`integer`
`- a`	negation			
`a + b`	addition	`integer`	`integer`	`integer`
`a - b`	subtraction			
`a & b`	concatenation	1-D array, element	1-D array, element	1-D array

Standard VHDL – Data Types

- **integer**
  - Minimal range: -(2^31 – 1) to 2^31 – 1
- **boolean:** {true, false}
- **bit:** {‘1’, ‘0’}
- **bit_vector:** string of bits.
  - “0001_1111”

STD_LOGIC Demystified

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;
```

BIT versus STD_LOGIC

- VHDL standard **BIT** type
  - Can only model a value of ‘0’ or ‘1’

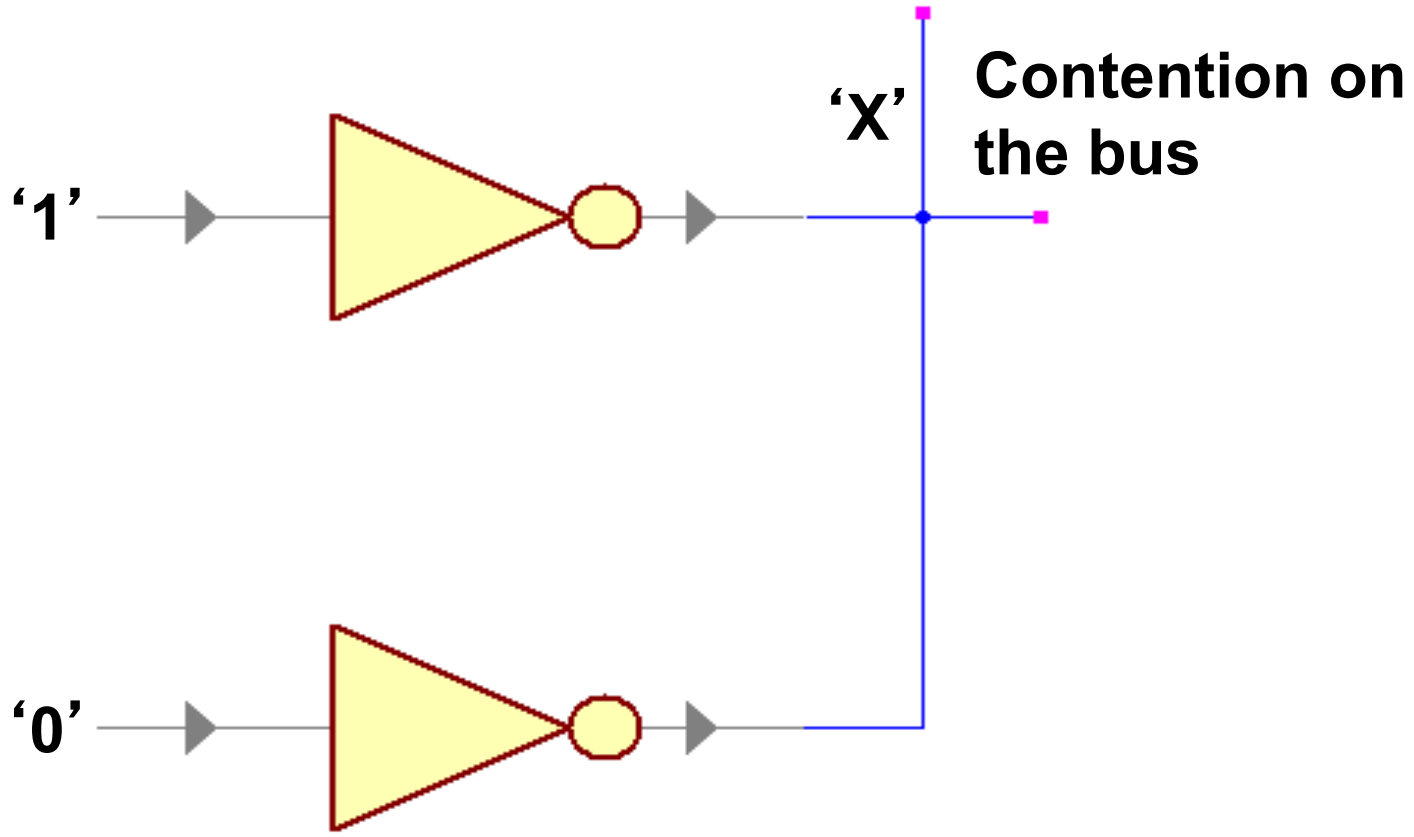
- **STD_LOGIC** can model nine values
  - ’U’, ’X’, ‘0’, ’1’, ’Z’, ’W’, ’L’, ’H’, ’-’
  - Useful mainly for simulation
  - ‘0’, ’1’, ‘X’ and ‘Z’ are *synthesizable*

    (**your codes should use only these four values**)

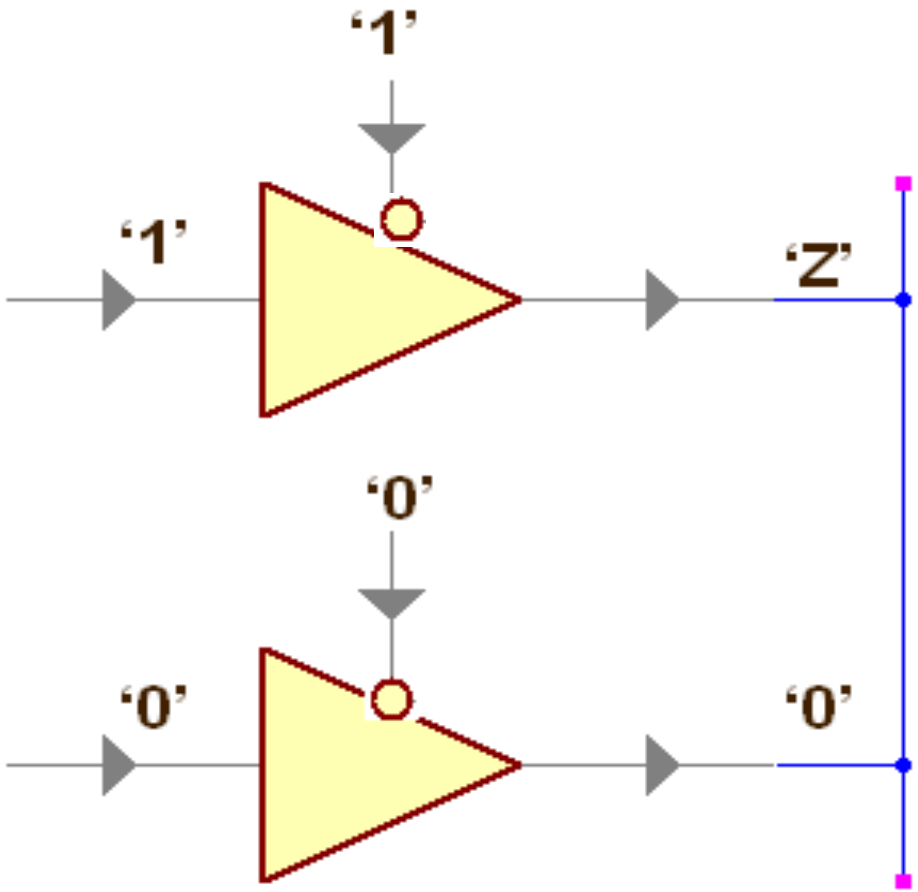
STD_LOGIC *type* demystified

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

More on STD_LOGIC Meanings (1)



More on STD_LOGIC Meanings (2)



Resolving Logic Levels

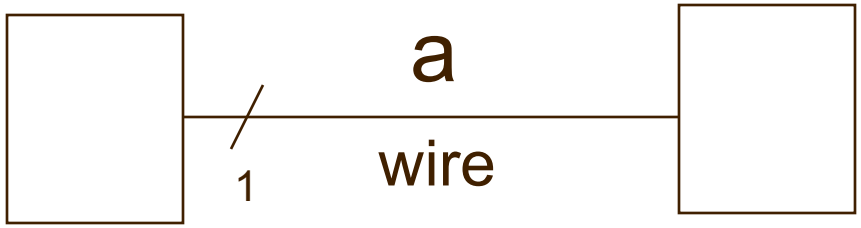
	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

STD_LOGIC Rules

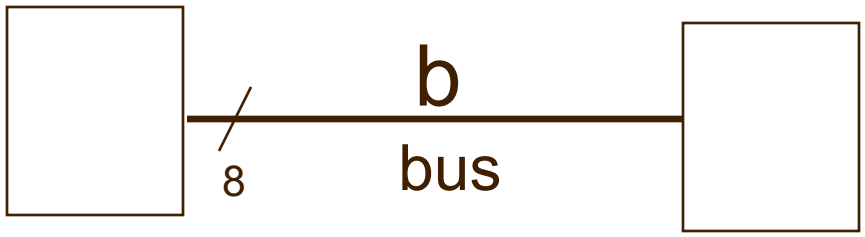
- In this course, use only **std_logic** or **std_logic_vector** for all entity input or output ports
- Do **NOT** use integer, unsigned, signed, bit for ports
  - You can use them inside of architectures if desired
  - You can use them in generics
- Instead use **std_logic_vector** and a conversion function inside of your architecture
  [Consistent with OpenCores Coding Guidelines]

Signals Modeling Wires and Buses

```
SIGNAL   a : STD_LOGIC;
```




```
SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);
```



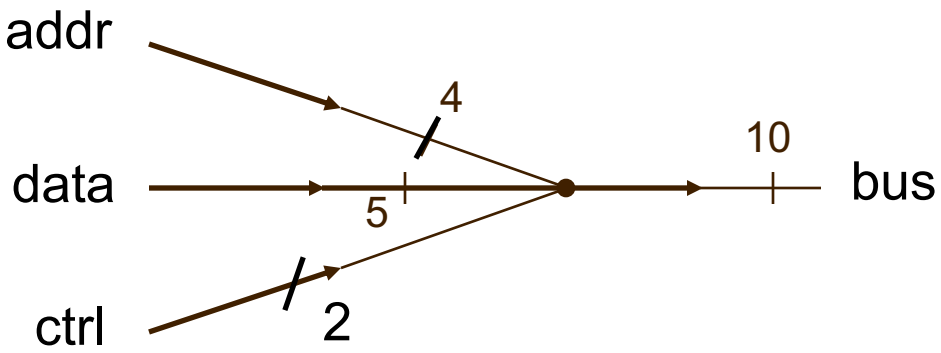
Standard Logic Vectors

```
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';          --assign a with logic ONE
b <= "0000";       --Binary base assumed by default
c <= B"0000";      --Binary base explicitly specified
d <= X"AF67";      -- Hexadecimal base
e <= O"723";       -- Octal base
```

Vectors and Concatenation

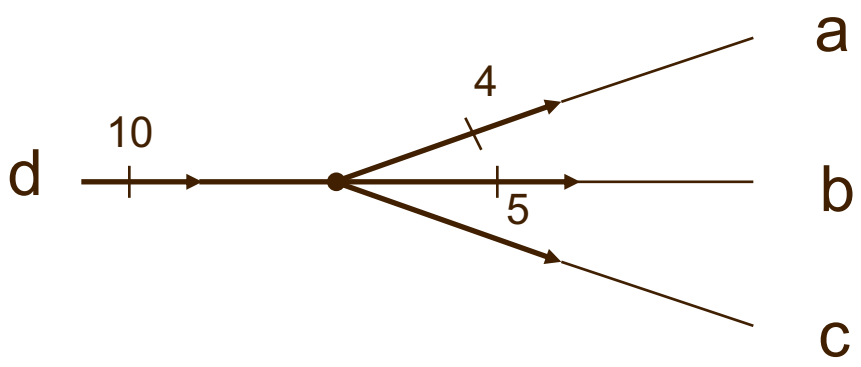
```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

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"
```

Merging Wires and Buses




```
SIGNAL addr : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL data : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL ctrl : STD_LOGIC_vector(1 downto 0);
SIGNAL bus  : STD_LOGIC_VECTOR(10 DOWNTO 0);

bus <= addr & data & ctrl;
```

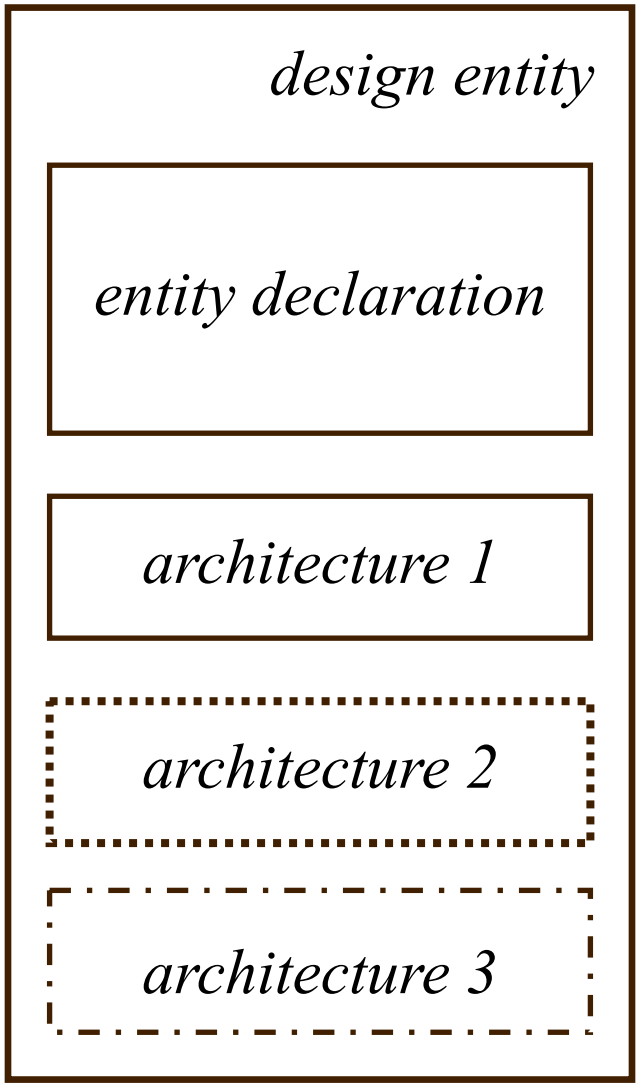
Splitting Buses




```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <= d(9 downto 6);
b <= d(5 downto 1);
c <= d(0);
```

Modeling Styles

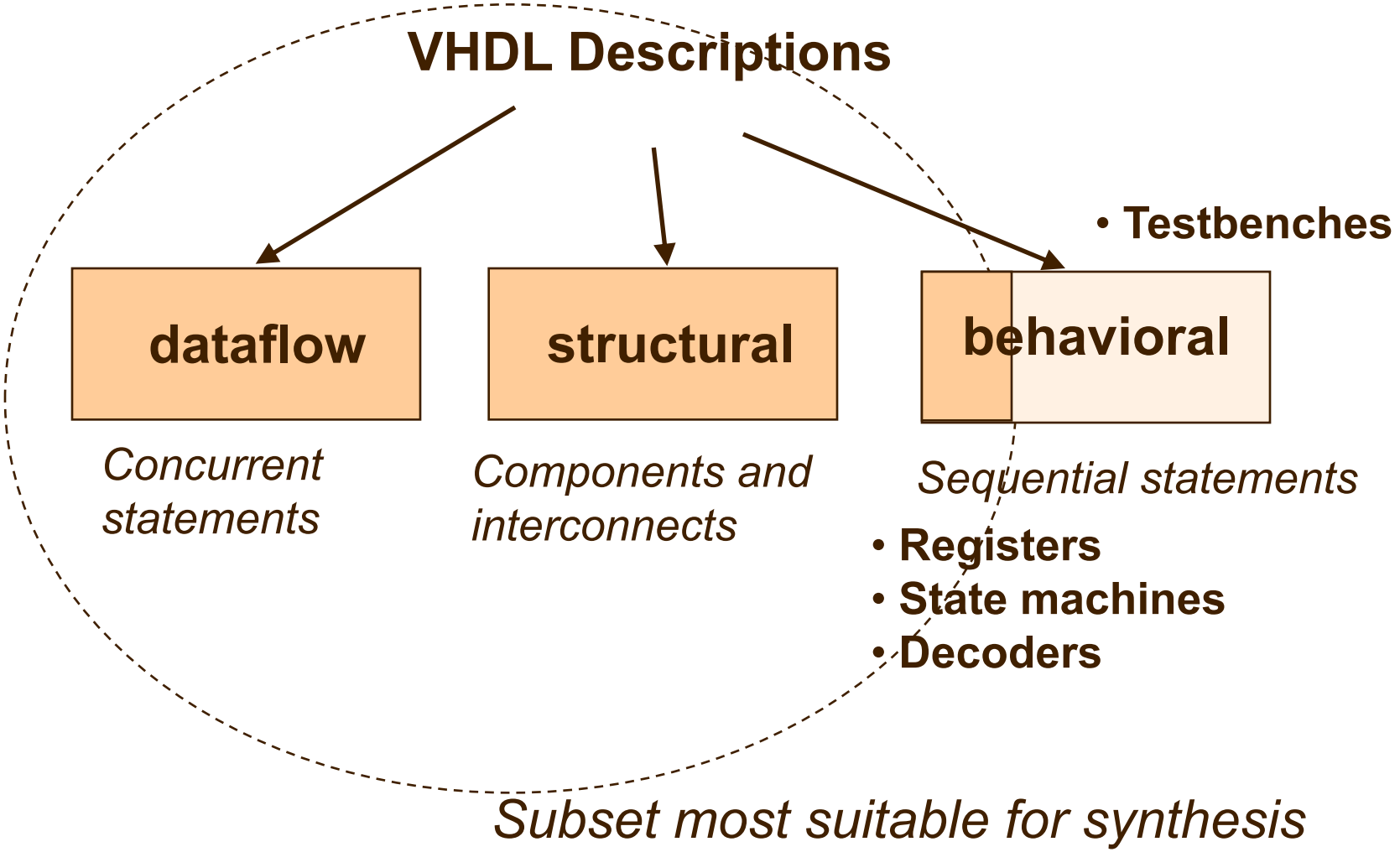
Design Entity



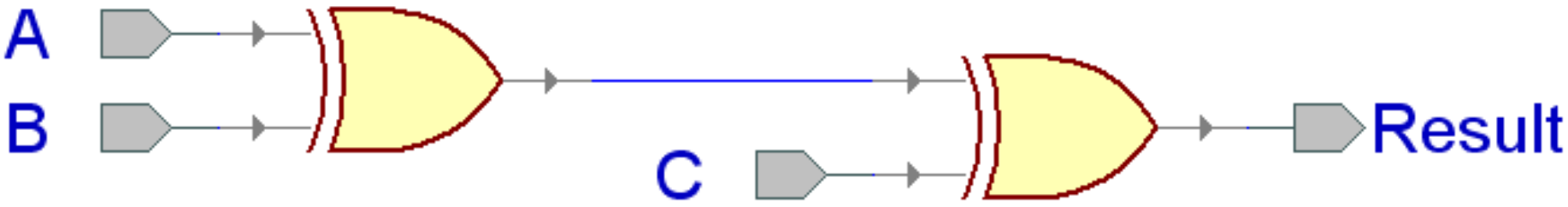
Design Entity - most basic building block of a design.

One *entity* can have many different *architectures.*

Types of VHDL Descriptions



xor3 Example



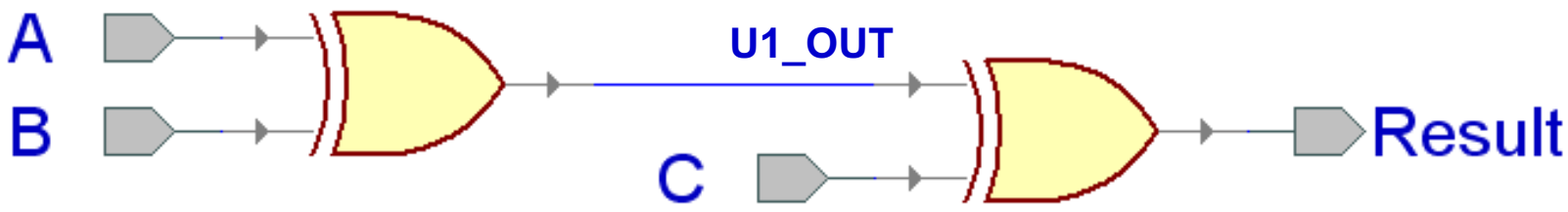
Entity xor3 Gate

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

ENTITY xor3 IS
    PORT(     A :          IN STD_LOGIC;
              B :          IN STD_LOGIC;
              C :          IN STD_LOGIC;
              Result :     OUT STD_LOGIC);
end ENTITY xor3;
```

Dataflow Modeling

Dataflow Architecture - xor3 Gate

```
ARCHITECTURE dataflow OF xor3 IS
    SIGNAL U1_OUT: STD_LOGIC;
BEGIN
    U1_OUT   <= A XOR B;
    Result   <= U1_OUT XOR C;
END ARCHITECTURE dataflow;
```



Dataflow Description

- Describes how data moves through the various processing steps of the system.
  - Uses series of concurrent statements to realize logic.
  - Most useful style when series of Boolean equations can represent a logic → used to implement simple combinational logic
  - Dataflow code also called *concurrent* code
- Concurrent statements are evaluated at the same time; thus, the order of these statements does NOT matter
  - This is not true for sequential/behavioral statements

Describe the same behavior

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

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

Event-Driven Semantics

- When a concurrent statement is evaluated?

when there is an event on a signal on the right hand side of an assignment.

- An event is a change of value on a signal.
- Consider the example in the previous slide.

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

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

Event-Driven Semantics

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

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

	0				
A	0				
B	0				
C	0				
U1_out	0				
Result	0				

Event-Driven Semantics

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

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

	0	**1**			
A	0	1			
B	0	0			
C	0	0			
U1_out	0	0			
Result	0	0			

Event-Driven Semantics

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

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

	0	**1**	**1 + *t***		
A	0	1	1		
B	0	0	0		
C	0	0	0		
U1_out	0	0	**1**		
Result	0	0	0		

Event-Driven Semantics

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

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

	0	**1**	**1 + *t***	**1 + *2t***	
A	0	1	1	1	
B	0	0	0	0	
C	0	0	0	0	
U1_out	0	0	1	1	
Result	0	0	0	**1**	

Event-Driven Semantics – Another Example

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

	0	**1**			
A	0	1			
B	0	0			
C	0	1			
U1_out	0	0			
Result	0	0			

Event-Driven Semantics – Another Example

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

	0	**1**	**1 + *t***		
A	0	1	1		
B	0	0	0		
C	0	1	1		
U1_out	0	0	**1**		
Result	0	0	**1**		

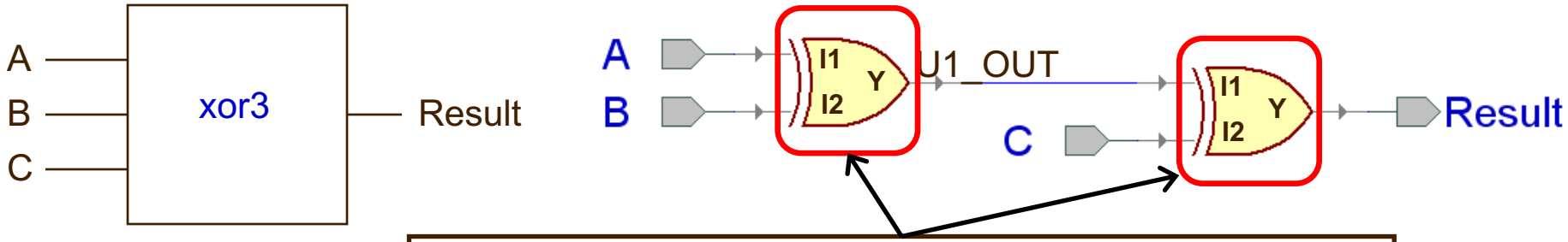
Event-Driven Semantics – Another Example

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

	0	**1**	**1 + *t***	**1 + *2t***	
A	0	1	1	1	
B	0	0	0	0	
C	0	1	1	1	
U1_out	0	0	1	1	
Result	0	0	1	**0**	

Structural Modeling

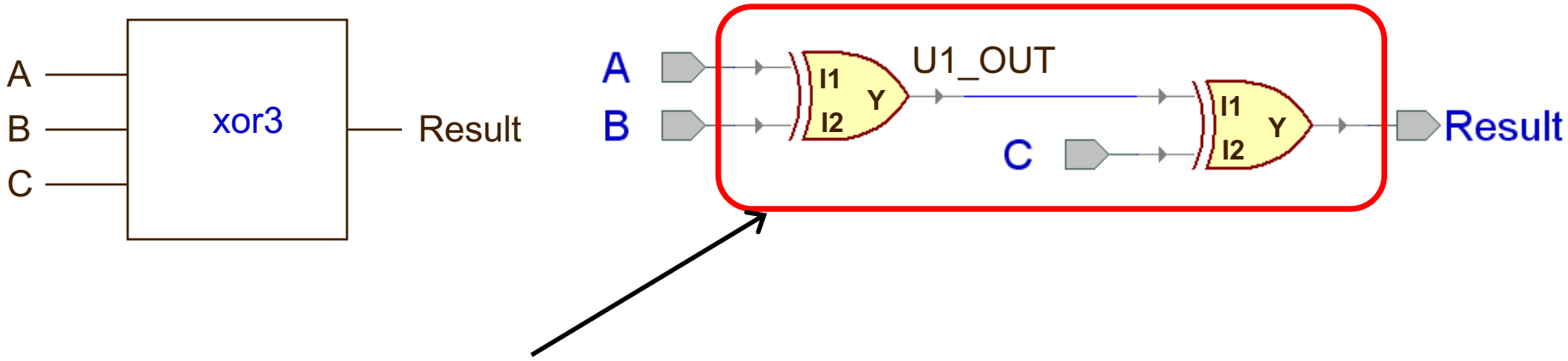
Structural Architecture – xor3 Gate



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 ENTITY xor2;

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

Structural Architecture in VHDL 93




```
ARCHITECTURE structural OF xor3 IS
   SIGNAL U1_OUT: STD_LOGIC;
BEGIN
   U1: entity work.xor2(dataflow) -- VHDL93 style
         PORT MAP (I1 => A, I2 => B, Y  => U1_OUT);

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

Structural Architecture in VHDL 93

General Syntax

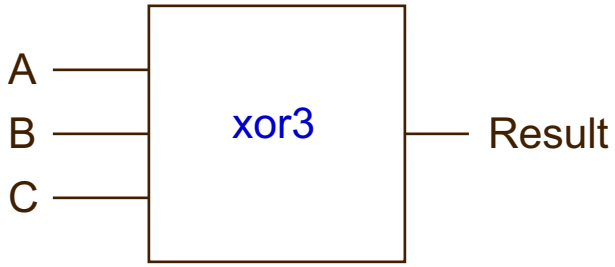
```
inst_label: entity lib_name.entity_name(arch_name)
        PORT MAP (
           port1 =>   actual_signal1,
           port2 =>   actual_signal2,
                 ...
           );
```

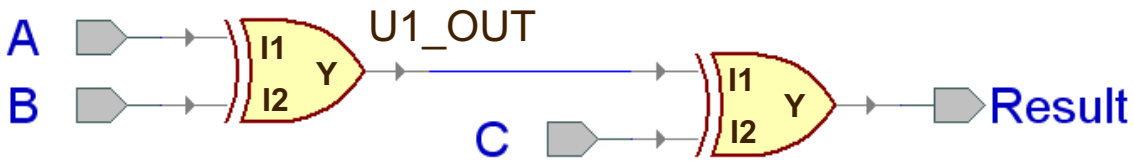
Actual signals can be

- ports of the entity where the component is instantiated
- signals declared in the architecture body

Structural Architecture in VHDL 87

```
ARCHITECTURE structural OF xor3 IS
   SIGNAL U1_OUT: STD_LOGIC;
   COMPONENT xor2
      PORT(
        I1 : IN STD_LOGIC;
        I2 : IN STD_LOGIC;
        Y : OUT STD_LOGIC);
   END COMPONENT;
```






```
BEGIN
   U1: xor2 PORT MAP (I1 => A, I2 => B, Y=> U1_OUT);

   U2: xor2 PORT MAP (I1 => U1_OUT,
                      I2 => C,
                      Y  => Result);
END structural;
```

Structural Description

- Allows divide-n-conquer for large designs.
- 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.

Behavioral Modeling

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 ARCHITECTURE behavioral;
```

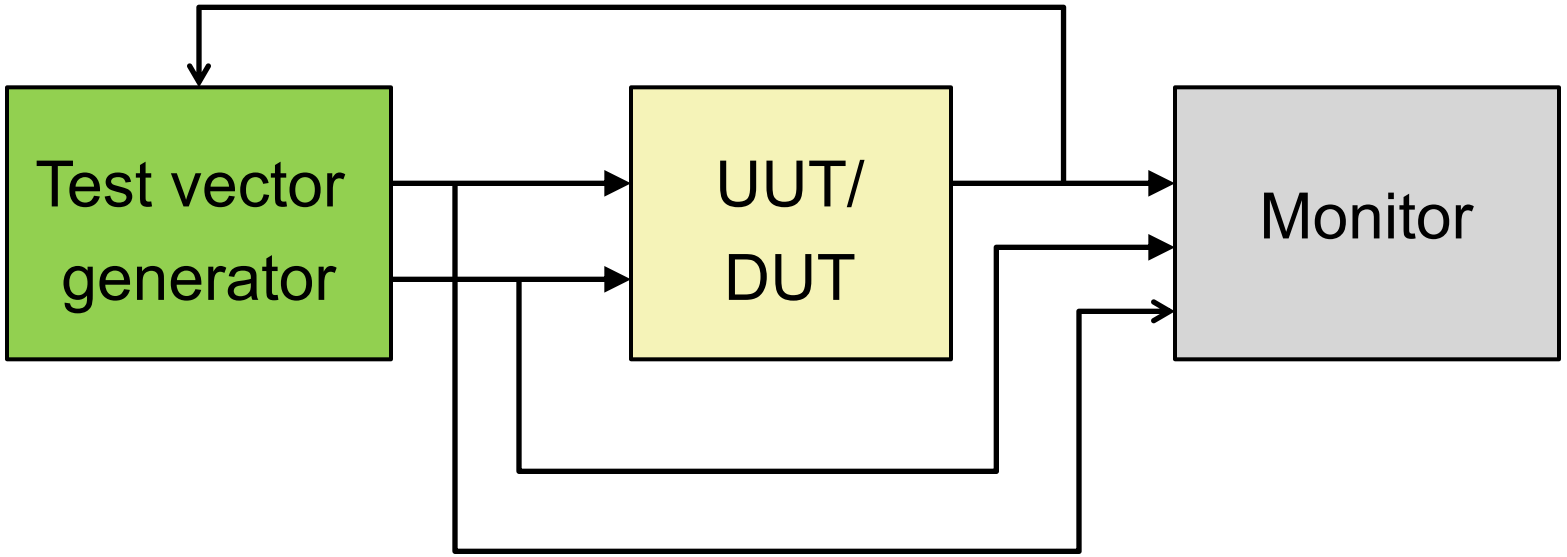
Behavioral Description

- It describes what happens on the inputs and outputs of the black box (no matter how a design is actually implemented).
  - Focus on functions mapping inputs to outputs
  - Similar to dataflow style,
  - More like sequential SW programming.
- This style uses process statements in VHDL.
  - A process itself is a concurrent statement.
  - A process consist of sequential statements.
- More will be covered later.

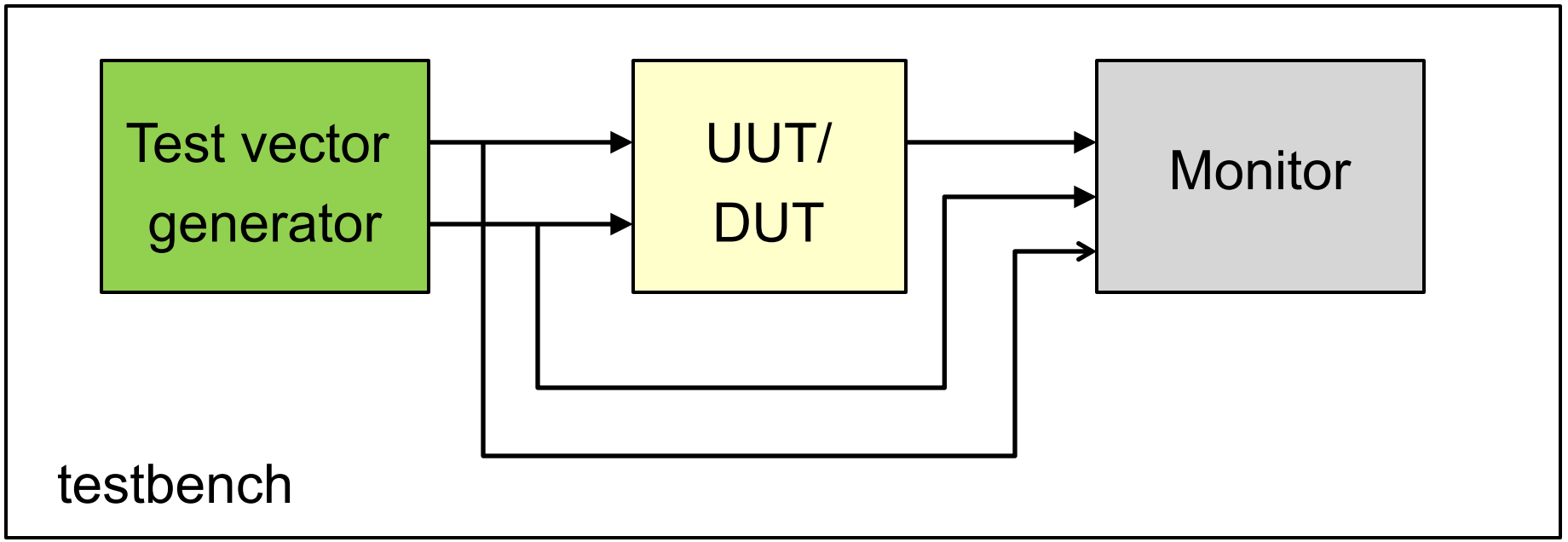
Verification and Test Bench

Design Testing and Testbenches

- After a design is done, it needs to be tested.
- During testing, design inputs are driven by various test vectors, and
- Outputs are monitored and checked.



Testbench in VHDL




```
library ieee;
use ieee.std_logic_1164.all;

ENTITY testbench IS
END testbench;
```

```
ARCHITECTURE tb_arch OF testbench IS
  -- signal declarations
  signal A, B, C, Result : std_logic;
BEGIN
  -- Instantiate the design under test.
  uut: entity work.xor3(structural)
          port map (A => A, B => B, ... );

  -- test vector generator
  test_gen: process
  begin
     -- generate sequence of vectors to
     -- drive design inputs A, B, and C.
  end process;
end tb_arch;
```

```
ARCHITECTURE tb_arch OF testbench IS
  -- signal declarations
  signal A, B, C, Result : std_logic;
BEGIN
  -- DUT instance
  ...
  -- test vector generator
  process
  begin
     A <= '1';   -- first test vector
     B <= '0';
     C <= '0';
     wait for 20ns; -- wait for circuit
                    -- to stablize
     ...
  end process;
end tb_arch;
```

Backup

Brief History of VHDL

Genesis of VHDL – State of art circa 1980

- Multiple design entry methods and hardware description languages in use
- No or limited portability of designs between CAD tools from different vendors
- **Objective**: shortening the time from a design concept to implementation from 18 months to 6 months

A Brief History of VHDL

- July 1983: a DoD contract for the development of VHDL awarded to
  - Intermetrics
  - IBM
  - Texas Instruments
- August 1985: VHDL Version 7.2 released
- December 1987:

  VHDL became IEEE Standard 1076-1987 and in 1988 an ANSI standard

Verilog

Verilog

- Essentially identical in function to VHDL
- Simpler and syntactically different
  - C-like
- Gateway Design Automation Co., 1985
- Gateway acquired by Cadence in 1990
- IEEE Standard 1364-1995 (Verilog-95)
- Early *de facto* standard for ASIC design
- Two subsequent versions
  - Verilog 2001 (major extensions) ← dominant version used in industry
  - Verilog 2005 (minor changes)
- Programming language interface to allow connection to non-Verilog code

Types of VHDL Descriptions: Alternative View