香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 01: Introduction to VHDL

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Outline

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL

Basic Structure of a VHDL Module

A VHDL file

Library Declaration

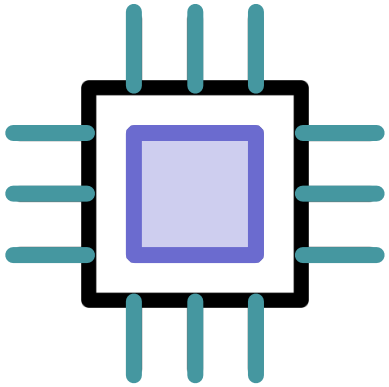
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

Entity Declaration

Define the signals to be seen outside externally (I/O pins)

Architecture Body

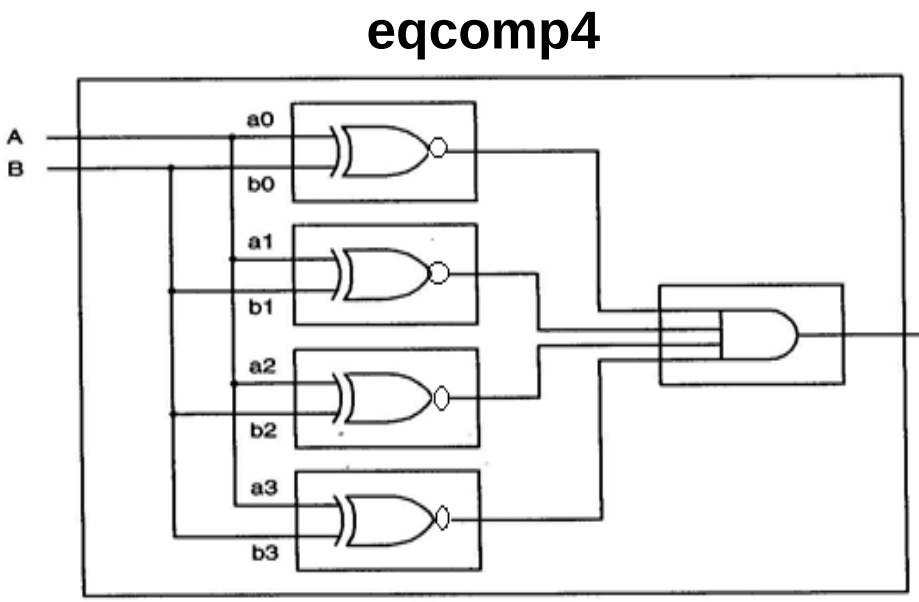
Define the internal operations of the entity (desired functions)



Example: 4-bit Comparator in VHDL (1/2)

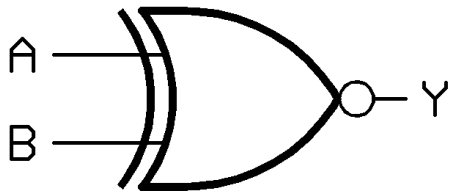
- **Schematic Circuit** of a 4-bit Comparator

A = [a3,a2,a1,a0]
B = [b3,b2,b1,b0]



*Recall: Exclusive NOR (XNOR)

– When A=B, Output Y = 0
– Otherwise, Output Y = 1



Truth Table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

VHDL for programmable logic, Skahill, Addison Wesley

Example: 4-bit Comparator in VHDL (2/2)

- Code of 4-bit Comparator in VHDL:

eqcomp4.vhd

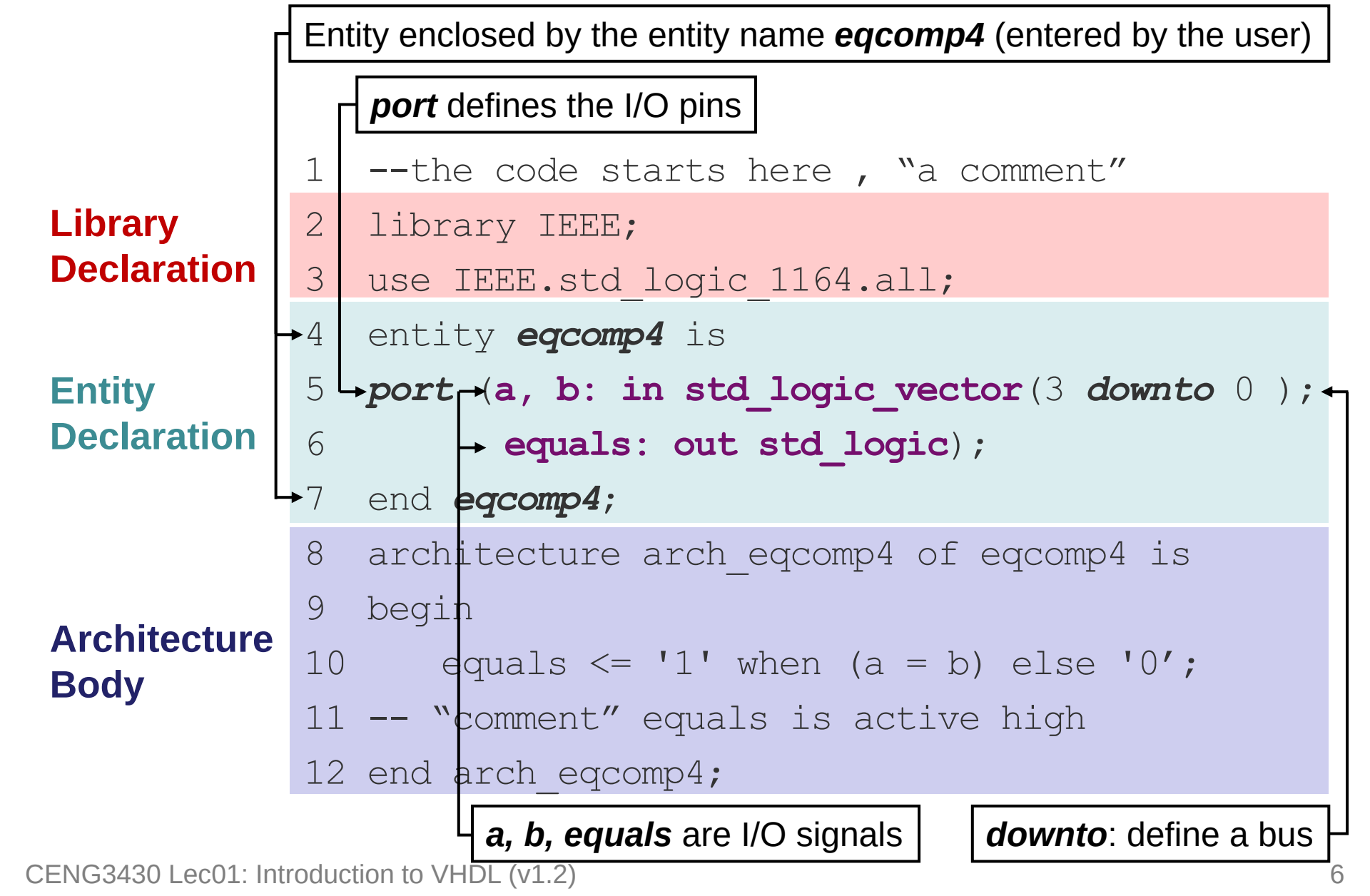
Library Declaration

Entity Declaration

Architecture Body

```
1  --the code starts here , "a comment"
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4  entity eqcomp4 is
5  port (a, b: in std_logic_vector(3 downto 0 );
6       equals: out std_logic);
7  end eqcomp4;
8  architecture arch_eqcomp4 of eqcomp4 is
9  begin
10     equals <= '1' when (a = b) else '0';
11 -- "comment" equals is active high
12 end arch_eqcomp4;
```

Entity Declaration



Concept of I/O Signals

- A I/O signal (or I/O pin) can
  - Carry logic information.
  - Be implemented as a wire in hardware.
  - Be “in”, “out”, “inout”, “buffer” (modes of I/O pin)

- There are many logic types of signals
  1) **bit**: can be logic 1 or 0 only
  2) **std_logic**: can be 1, 0, Z (high impedance), ..., etc
     - Standard logic (an IEEE standard)
  3) **std_logic_vector**: a group of wires (a bus)
     - a, b: in std_logic_vector(3 downto 0); in VHDL
     - a(0), a(1), a(2), a(3), b(0), b(1), b(2), b(3) are **std_logic** signals

Class Exercise 1.1

Student ID: _____ Date:
Name: _____ _____

eqcomp4.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity eqcomp4 is
4  port (a, b: in std_logic_vector(3 downto 0 );
5       equals: out std_logic);
6  end eqcomp4;
7  architecture arch_eqcomp4 of eqcomp4 is
8  begin
9     equals <= '1' when (a = b) else '0';
10 end arch_eqcomp4;
```

- How many input and output pins are there in the code?
  Answer: _____
- What are their names and their types?
  Answer: _____
- What is the difference between std_logic and std_logic_vector?
  Answer: _____

Class Exercise 1.2

Student ID: _____ Date:
Name: _____ _____

test1.vhd

```
1 entity test12 is
2 port (in1, in2: in std_logic;
3           out1: out std_logic);
4 end test12;
5 architecture test12arch of test12 is
6 begin
7     out1 <= in1 or in2;
8 end test12_arch;
```

- Give line numbers of (1) entity declaration and (2) arch. body.
  Answer: _____
- Find an error in the VHDL code.
  Answer: _____
- Draw the schematic chip and names the pins.
  Answer: _____
- Underline the words that are defined by users in the code.
  Answer: _____

Class Exercise 1.3

Student ID: _____ Date:
Name: _____ _____

- Rewrite the VHDL code in class exercise 1.2, with
  1) Entity name is test1x (not test1)
  2) Input names are in1x and in2x, resp. (not in1 and in2)
  3) Output name is out1x (not out1)
  4) Logic types of input and output are bit (not std_logic)
  5) Architecture name is test1x_arch (not test1_arch)

test1x.vhd

Modes of I/O Pins

- Modes of I/O pin should be explicitly specified in **port** of entity declaration:

  Example:

```
entity do_care is port(
   s: in std_logic_vector(1 downto 0);
   y: buffer std_logic);
end do_care;
```

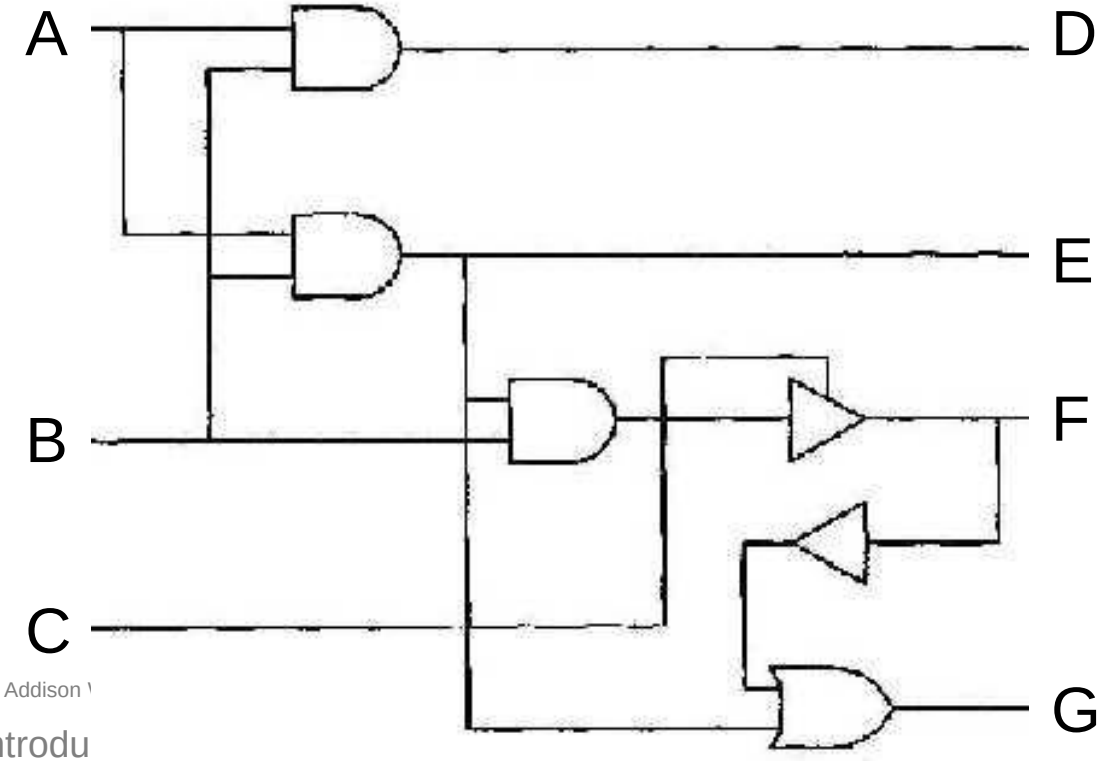
- There are 4 modes of I/O pins:
  1) **in**: Data flows **in** only
  2) **out**: Data flows **out** only (cannot be read back by the entity)
  3) **inout**: Data flows **bi-directionally** (i.e., in or out)
  4) **buffer**: Similar to **out** but it can be **read back** by the entity

Class Exercise 1.4

Student ID: _____ Date:
Name: _____ _____

- State the difference between out and buffer.
  Answer: _____

  _____
- Based on the following schematic, identify the modes of the IO pins.



Architecture Body *(More in Lec03)*

- Architecture Body: Defines the operation of the chip

  Example:

```
architecture arch_eqcomp4 of eqcomp4 is
begin
   equals <= '1' when (a = b) else '0';
   -- "comment" equals is active high
end arch_eqcomp4;
```

  How to read it?

  - **`arch_eqcomp4`**: the architecture name (entered by the user)
  - **`equals, a, b`**: I/O signal pins designed by the user in the entity declaration
  - **`begin`** … **`end`**: define the internal operation
    - equals <= '1' when (a = b) else '0';
  - "**`--`**": comment

Class Exercise 1.5

Student ID: _____ Date:
Name: _____ _____

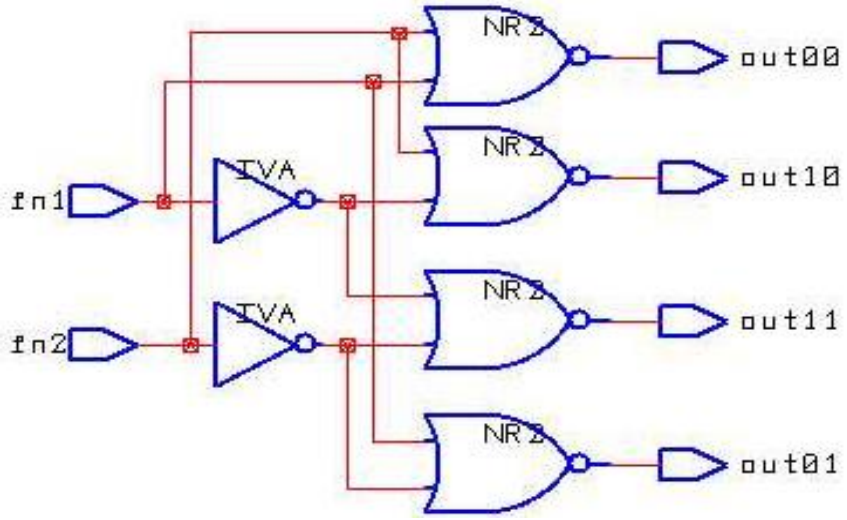
- Draw the schematic circuit for the following code.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity test15 is
4      port( in1: in std_logic_vector (2 downto 0);
5            out1: out std_logic_vector (3 downto 0));
6  end test15;
7  architecture test15_arch of test is
8  begin
9      out1(0)<= in1(1);
10     out1(1)<= in1(2);
11     out1(2)<= not(in1(0) and in1(1));
12     out1(3)<= '1';
13 end test15_arch ;
```

Class Exercise 1.6

Student ID: _____ Date:
Name: _____ _____

- Consider the circuit:
  - What is this circuit for?
    Answer: _____
  - Fill in the truth table.
  - Fill in the blanks of code.




```
1  entity test16 is
2  port (in1,in2: in std_logic;
3        out00,out01,out10,out11: out std_logic);
4  end test16;
5  architecture test16_arch of test16 is
6  begin
7    out00 <= not(____________________);
8    out10 <= not(____________________);
9    out11 <= not(____________________);
10   out01 <= not(____________________);
11 end test16_arch;
```

in1	in2	out 00	out 10	out 11	out 01
0	0				
1	0				
1	1				
0	1				

Outline

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL

Identifiers

- Identifiers: Used to represent and name an object
  - An object can be constant, signal or variable.

- Rules for naming data objects:
  1) Made up of alphabets, numbers, and underscores
  2) First character must be a letter
  3) Last character CANNOT be an underscore
  4) NOT case sensitive
     - Txclk, Txclk, TXCLK, TxClk are equivalent
  5) Two connected underscores are NOT allowed
  6) VHDL-reserved words may NOT be used

Class Exercise 1.7

Student ID: _____ Date:
Name: _____ _____

- Determine whether the following identifiers are legal or not. If not, please give your reasons.
  - tx_clk
  - _tx_clk
  - Three_State_Enable
  - 8B10B
  - sel7D
  - HIT_1124
  - large#number
  - link__bar
  - select
  - rx_clk_

Outline

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL

Objects: 3 Different Data Objects

- Data objects are assigned types and hold values of the specified types.
- Data objects belong to one of three classes:
  1) Constants (Globla): Hold unchangeable values
     - E.g., `constant width: INTEGER :=8;`
  2) Signals (Globla): Represent physical wires
     - E.g., `signal count: BIT := '1';`
  3) Variables (Local): Used only by programmers for internal representation (do not exist physically)
     - E.g., `variable flag: BOOLEAN := TRUE;`
- Data objects must be declared before being used.

Constant Objects (Global)

```
constant CONST_NAME: <type> := <value>;
```

Note: Constants must be declared with initialized values.

- Examples:
  - `constant CONST_NAME: STD_LOGIC := 'Z';`
  - `constant CONST_NAME: STD_LOGIC_VECTOR (3 downto 0) := "0-0-";`
    - `'-'` is don't care
- Constants can be declared in
  - Anywhere allowed for declaration.

Signal Objects (Global)

`signal SIG_NAME: <type> [: <value>];`

Note: Signals can be declared without initialized values.

- Examples:
  - `signal s1_bool : BOOLEAN;`
    - Declared without initialized value
  - `signal xsl_int1: INTEGER :=175;`
  - `signal su2_bit: BIT :='1';`

- Signals can be declared
  - Either in the "`port`" of the entity declaration,
  - Or before the "`begin`" of the architecture body.

Recall: Modes of I/O Pins

- If a signal is declared in **port**, it is used as I/O pins.
- Modes of I/O pin should be explicitly specified in **port** of entity declaration:

  Example:

```
entity do_care is port(
   s: in std_logic_vector(1 downto 0);
   y: buffer std_logic);
end do_care;
```

- There are 4 modes of I/O pins:
  1) **in**: Data flows **in** only
  2) **out**: Data flows **out** only (cannot be read back by the entity)
  3) **inout**: Data flows **bi-directionally** (i.e., in or out)
  4) **buffer**: Similar to **out** but it can be **read back** by the entity

Variable Objects (Local)

```
variable VAR_NAME: <type> [: <value>];
```

Note: Variables can be declared without initialized values.

- Examples:
  - `variable v1_bool : BOOLEAN:= TRUE;`
  - `variable val_int1: INTEGER:=135;`
  - `variable vv2_bit: BIT;`
    - Declared without initialized value

- Variables can only be declared/used in the `process` statement in the architecture body (*see Lec03*).

Signals and Variables Assignments

- Both signals and variables can be declared without initialized values.

- Their values can be assigned after declaration.
  - Syntax of signal assignment:

```
SIG_NAME <= <expression>;
```

  - Syntax of variable assignment:

```
VAR_NAME := <expression>;
```

Class Exercise 1.8

Student ID: _____ Date:
Name: _____ _____

- Consider the following segment of VHDL code:

```
entity test18 is port (
    CLK, ASYNC ,LOAD: in STD_LOGIC;
    DIN: in STD_LOGIC_VECTOR(3 downto 0);
    DOUT: out STD_LOGIC_VECTOR(3 downto 0) );
end test18;
```

- What are identifiers:
  Answer: _____
- What are input signals?
  Answer: _____
- What is the type of signal DIN?
  Answer: _____
- What is the mode of DOUT?
  Answer: _____

Class Exercise 1.9

Student ID: _____ Date:
Name: _____ _____

```
1  entity nandgate is
2      port (in1, in2: in STD_LOGIC;
3                 out1: out STD_LOGIC);
4  end nandgate;
5  architecture nandgate_arch of nandgate is
6  ____________________________________________
7  begin
8  connect1 <= in1 and in2;
9  out1<= not connect1;
10 end nandgate_arch;
```

- Declare a signal named “connect1” in Line 6.
- Can you assign an I/O mode to this signal? Why?
  Answer: _____
- Where can we declare signals?
  Answer: _____
- Draw the schematic circuit for the code.

Alias

- An alias is an alternate identifier for an existing object.
  - It is NOT a new object.
  - Referencing the alias is equivalent to the original one.
  - It is often used as a convenient method to identify a range of an array (signal bus) type.

- Example:
  - `signal sig_x: std_logic_vector(31 downto 0);`
  - `alias top_x: std_logic_vector (3 downto 0) is sig_x(31 downto 28);`

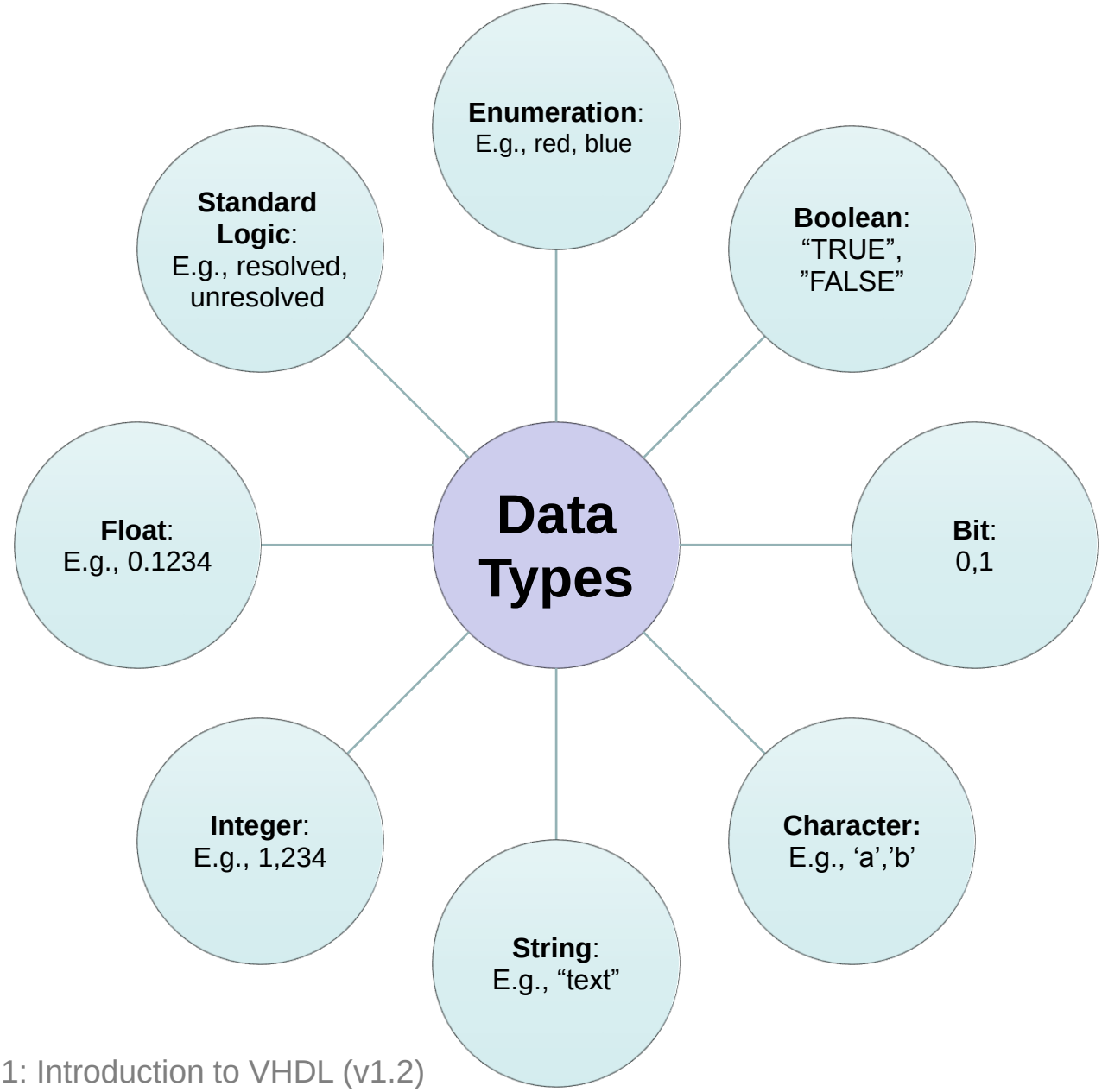
What we learnt so far

- Identifier
- Data Object
  - Constant
  - Signal
    - In Port: External I/O Pins
      - Modes of I/O Pins: In, Out, Inout, Buffer)
    - In Architecture Body: Internal Signals
  - Variable

Outline

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL

Data Types in VHDL (1/2)



Data Types in VHDL (2/2)

- VHDL is strongly-typed language.
  - Data objects of different base types CANNOT to assigned to each other without the use of type-conversion.
- A type has a set of values and a set of operations.
- Common types can be classified into two classes:
  - Scalar Types
    - Integer Type
    - Floating Type
    - Enumeration Type
    - Physical Type
  - Composite Types
    - Array Type
    - Record Type

Scalar: Integer Type

- An integer type can be defined with or without specifying a range.
  - If a range is not specified, VHDL allows integers to have a minimum rage of

$$-2{,}147{,}483{,}647\ to\ 2{,}147{,}483{,}647$$

$$-(2^{31}-1)\ to\ (231-1)$$

  - Or a range can be specified, e.g.,

```
variable a: integer range 0 to 255;
```

Scalar: Floating Type

- Floating point type values are used to approximate real numbers.

- The only predefined floating type is named REAL, which includes the range

$$-1.0E38\ to\ +1.0E38$$

- Floating point types are rarely used (or even not supported) in code to be synthesized.
  - Because of its huge demand of resources.

Scalar: Enumeration Type (1/2)

- How to introduce an abstract concept into a circuit?
- An enumeration type is defined by a list of values.
  - The list of values may be defined by users.
  - Example:

```
type colors is (RED, GREEN, BLUE);
signal my_color: colors;
```

- Enumeration types are often defined for state machines (*see Lec05*).
- There are two particularly useful enumeration types predefined by the IEEE 1076/1993 standards.
  - `type BOOLEAN is (FALSE, TRUE);`
  - `type BIT is ('0', '1');`

Scalar: Enumeration Type (2/2)

- An enumerated type is ordered.
  - The order in which the values are listed in the type declaration defines their relation.
  - The leftmost value is less than all other values.
  - Each values is greater than the one to the left, and less than the one to the right.
- Example:

```
type colors is (RED, GREEN, BLUE)
signal my_color: colors;
```

  - Then a comparison of my_color can be:

```
when my_color >= RED
```

Scalar: Physical Type

- Physical type values are used as measurement units.
  - They are used mainly in simulations (*see Lab01*).
- The only predefined physical type is TIME.
  - Its primary unit is `fs` (femtoseconds) as follows:

```
type time is range -2147483647 to 2147483647
    units
        fs;
        ps = 1000 fs;
        ns = 1000 ps;
        us = 1000 ns;
        ms = 1000 us;
        sec = 1000 ms;
        min = 60 sec;
        hr = 60 min;
    end units;
```

Composite: Array Type

- An object of an array type consists of multiple elements of the same type.
- The most commonly used array types are those predefined by the IEEE 1076 and 1164 standards:

```
type BIT_VECTOR is array (NATURAL range <>) of bit;
type STD_LOGIC_VECTOR is array (NATURAL range <>) of std_logic;
```

  - Their range are not specified (using `range <>`), and only bounded by `NATURAL` (positive integers).
- Example:

```
port (a: in std_logic_vector (3 downto 0);
      b: in std_logic_vector (0 to 3);
 equals: out std_logic);
```

  - `a`, `b` are both 4-bit vectors of std_logic.

Class Exercise 1.10

Student ID: _____ Date:
Name: _____ _____

- Given

```
a: std_logic_vector (3 downto 0);
```

- Create a 4-bit bus `c` using "`to`" instead of "`downto`":

- Draw the circuit for this assignment `c <= a`

Composite: Record Type

- An object of a record type consists of multiple elements of the different types.
  - Individual fields of a record can be used by element name.
- Example:

```
type iocell is record
    buffer_in: bit_vector(7 downto 0);
    bnable: bit;
    buffer_out: bit_vector(7 downto 0);
end record;
```

  - Then we can use the record as follows:

```
signal bus_a: iocell;
signal vec: bit_vector(7 downto 0);
bus_a.buffer_in <= vec;
```

Types and Subtypes

- A subtype is a type with a constraint.
  - Subtypes are mostly used to define objects based on existing base types with a constraint.
- Example:
  - Without subtype

```
signal my_byte: bit_vector(7 downto 0);
```

  - With subtype:

```
subtype byte is bit_vector(7 downto 0);
signal my_byte: byte;
```

- Subtypes are also used to resolve a base type.
  - A resolution function is defined by the IEEE 1164 standard.

```
subtype std_logic is resolved std_ulogic;
```

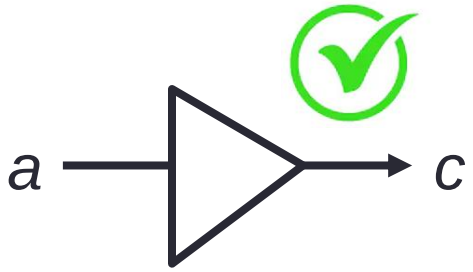
    - `Resolved` is the name of the resolution function.

Resolved Logic Concept

- Resolved Logic (Multi-value Signal): Multiple outputs can be connected together to drive a signal.
  - The resolution function is used to determine how multiple values from different sources (drivers) for a signal will be reduced to one value.

- Single-value Signal Example:

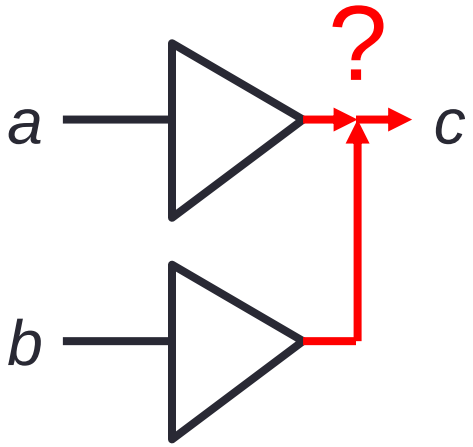
```
signal a, c: bit;
c <= a;
```



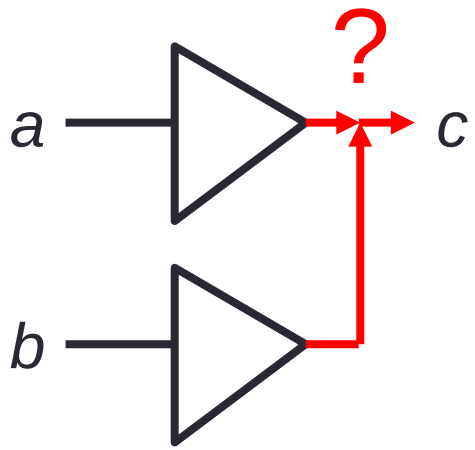
- Multi-value Signal Example:

```
signal a, b, c: bit;
c <= a;
c <= b;
```

← We need to "resolve" it!



std_logic vs. std_ulogic (1/2)



- `std_logic`: a type of resolved logic, that means a signal can be driven by 2 inputs

- `std_ulogic` ("**u**" means unresolved): a type of unresolved logic, that means a signal CANNOT be driven by 2 inputs

std_logic vs. std_ulogic (2/2)

- How to use it?

```
library IEEE;
use IEEE.std_logic_1164.all;
entity
```

```
architecture
```

IEEE 1164: 9-valued Logic Standard

- 'U': Uninitialized
- 'X': Forcing Unknown
- '0': Forcing 0
- '1': Forcing 1
- 'Z': High Impedance (Float)
- 'W': Weak Unknown
- 'L': Weak 0
- 'H': Weak 1
- '–': Don't care

VHDL Resolution Table

	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

- Rule: When 2 signals meet, the forcing signal dominates.

Class Exercise 1.11

Student ID: _____ Date:
Name: _____ _____

- Consider two std_logic signals S1 and S2 meet together, fill in the blanks in the following table.

	S1 = X	**S1 = 0**	**S1 = 1**	**S1 = Z**
S2 = X				
S2 = 0				
S2 = 1				
S2 = Z				

Outline

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL

Attributes (1/2)

- An attribute provides information about items such as entities, architecture, types, and signals.
  - There are several useful predefined value, signal, and range attributes.
- Example:

```
type count is integer range 0 to 127;
type states is (idle, decision, read, write);
type word is array(15 downto 0) of std_logic;
```

  - Then

```
count'left = 0          count'right = 127
states'left = idle      states'right = write
word'left = 15          word'right = 0
```

Attributes (2/2)

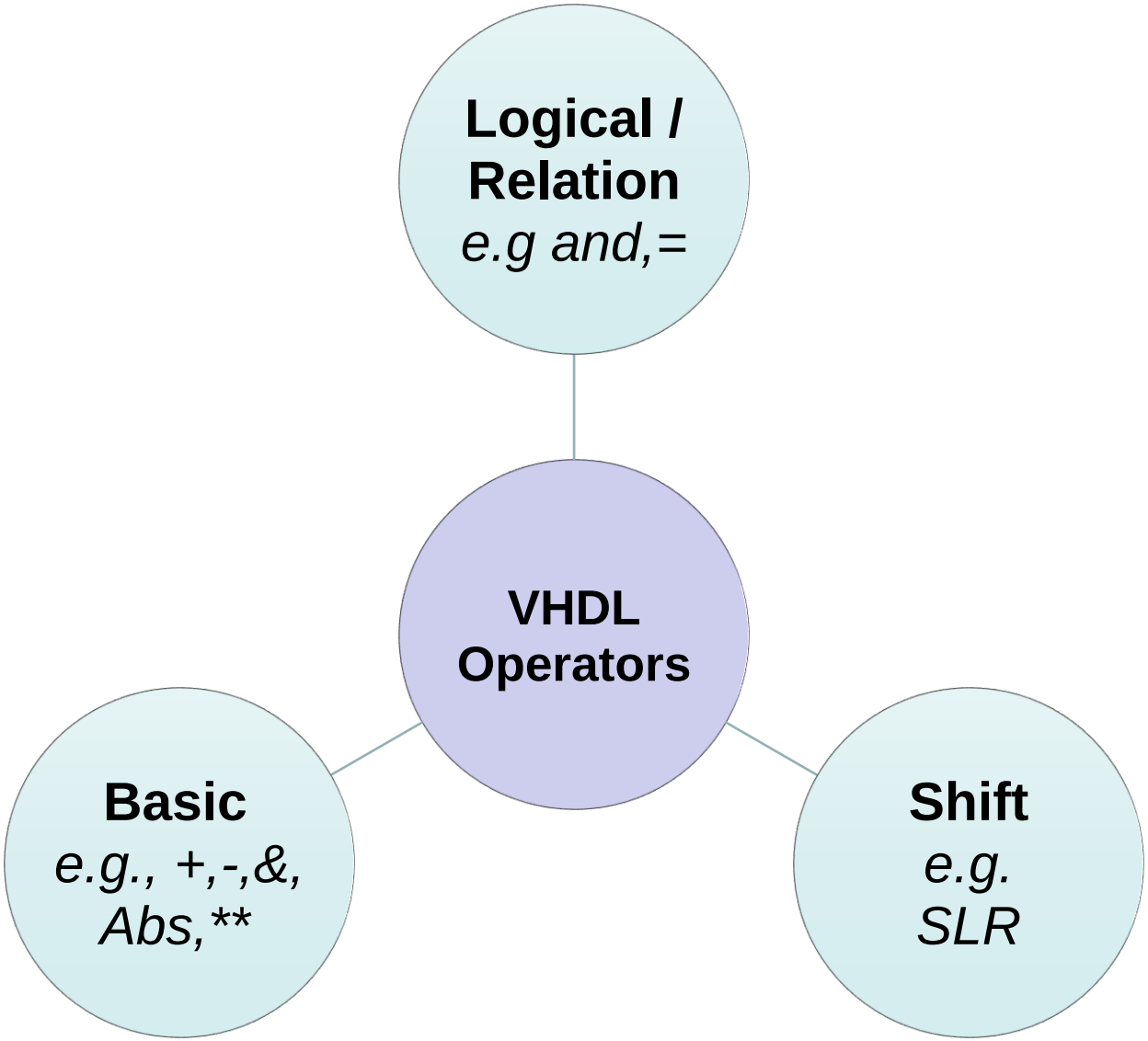
- Another important signal attribute is the **`'event`**.
  - This attribute yields a Boolean value of TRUE if an event has just occurred on the signal.
  - It is used primarily to determine if a **clock** has transitioned.
- Example (*more in Lec04*):

```
…
port(my_in, clock: in std_logic;
             my_out: out std_logic);
…
if clock = '1' and clock'event then
    my_out <= my_in;
```

Outline

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL

VHDL Operators



Basic Operators

`+`	arithmetic add, for integer, float.
`-`	arithmetic subtract, for integer, float.
`*`	multiplication
`/`	division
`rem`	remainder
`mod`	modulo ($A\ mod\ B = A\ -\ (B * N), N \in integer$)
`abs`	absolute value
`**`	exponentiation (e.g., 2**3 is 8)
`&`	concatenation (e.g., ‘0’ & ‘1’ → “01”)

Shift / Rotate Operators

- Logical Shift and Rotate
  - **`sll`** shift left logical, fill blank with 0
  - **`srl`** shift right logical, fill blank with 0
  - **`rol`** rotate left logical, circular operation
    - E.g. “10010101” rol 3 is “10101100”
  - **`ror`** rotate right logical, circular operation

- Arithmetic Shift
  - **`sla`** shift left arithmetic, fill blank with 0, same as `sll`
  - **`sra`** shift right arithmetic, fill blank with sign bit (MSB)

Class Exercise 1.12

Student ID: _____ Date:
Name: _____ _____

- Given signal A <= “1001 0101”, what are the values after applying the following shift or rotate operations?

- A `sll` 2 = _____
- A `srl` 3 = _____
- A `sla` 3 = _____
- A `sra` 2 = _____
- A `rol` 3 = _____
- A `ror` 5 = _____

Logical / Relation Operators

- Logical Operators: `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not` have their usual meanings.
  - E.g., `nand` is NOT associative
    - (A `nand` B) `nand` C ≠ A `nand` (B `nand` C)
    - A `nand` B `nand` C is illegal

- Relation Operators (result is Boolean)
  - = equal
  - /= not equal
  - < less than
  - <= less than or equal
  - > greater than
  - >= greater than or equal

Class Exercise 1.13

Student ID: _____ Date:
Name: _____ _____

- Fill in the blanks to show nand-gate is not associative:

A	B	C	A nand B	(A nand B) nand C	B nand C	A nand (B nand C)
0	0	0	1	1	1	1
0	0	1	1		1	1
0	1	0	1	1	1	1
0	1	1	1	0		1
1	0	0	1	1	1	0
1	0	1	1	0	1	
1	1	0	0	1	1	0
1	1	1	0	1		1

Summary

- Basic Structure of a VHDL Module
  - Library Declaration
  - Entity Declaration
  - Architecture Body
- Identifiers, Data Objects, and Data Types in VHDL
  - Identifier
  - Data Objects
    - Constant
    - Signal
    - Variable
  - Data Types
  - Attributes
- Operators in VHDL