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
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

Basic Structure of a VHDL Module

A VHDL file

1) Library Declaration

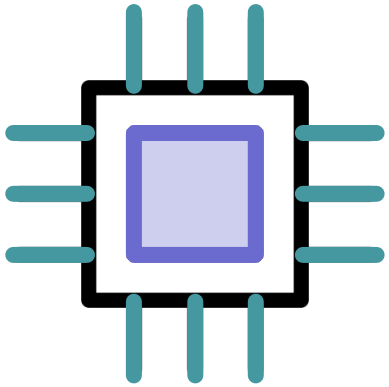
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

2) Entity Declaration

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

3) Architecture Body

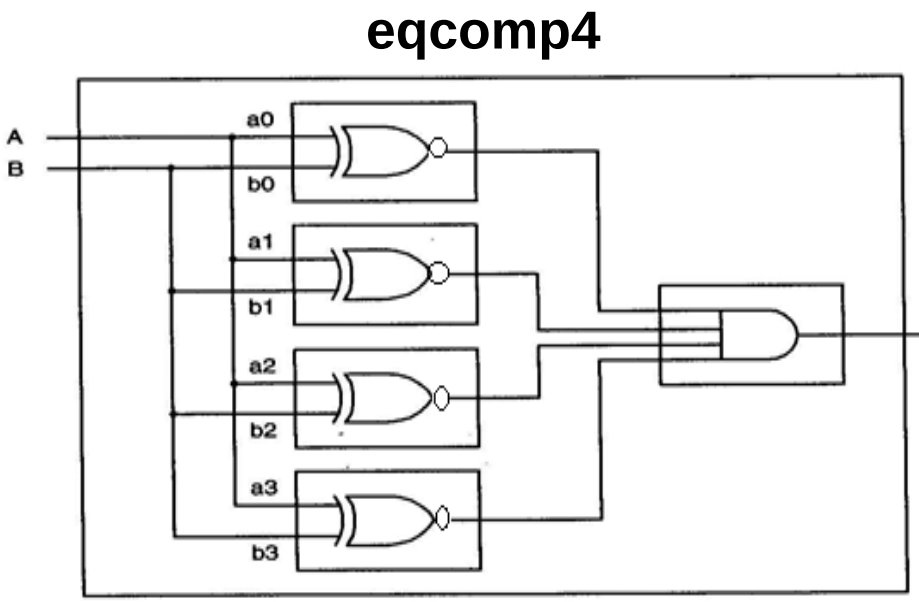
Define the internal signals and operations of the desired function



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

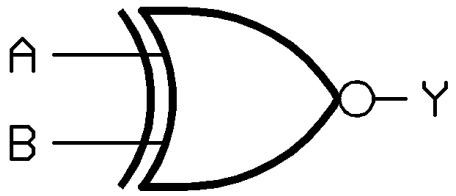
- **Schematic Circuit** of a 4-bit Comparator

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

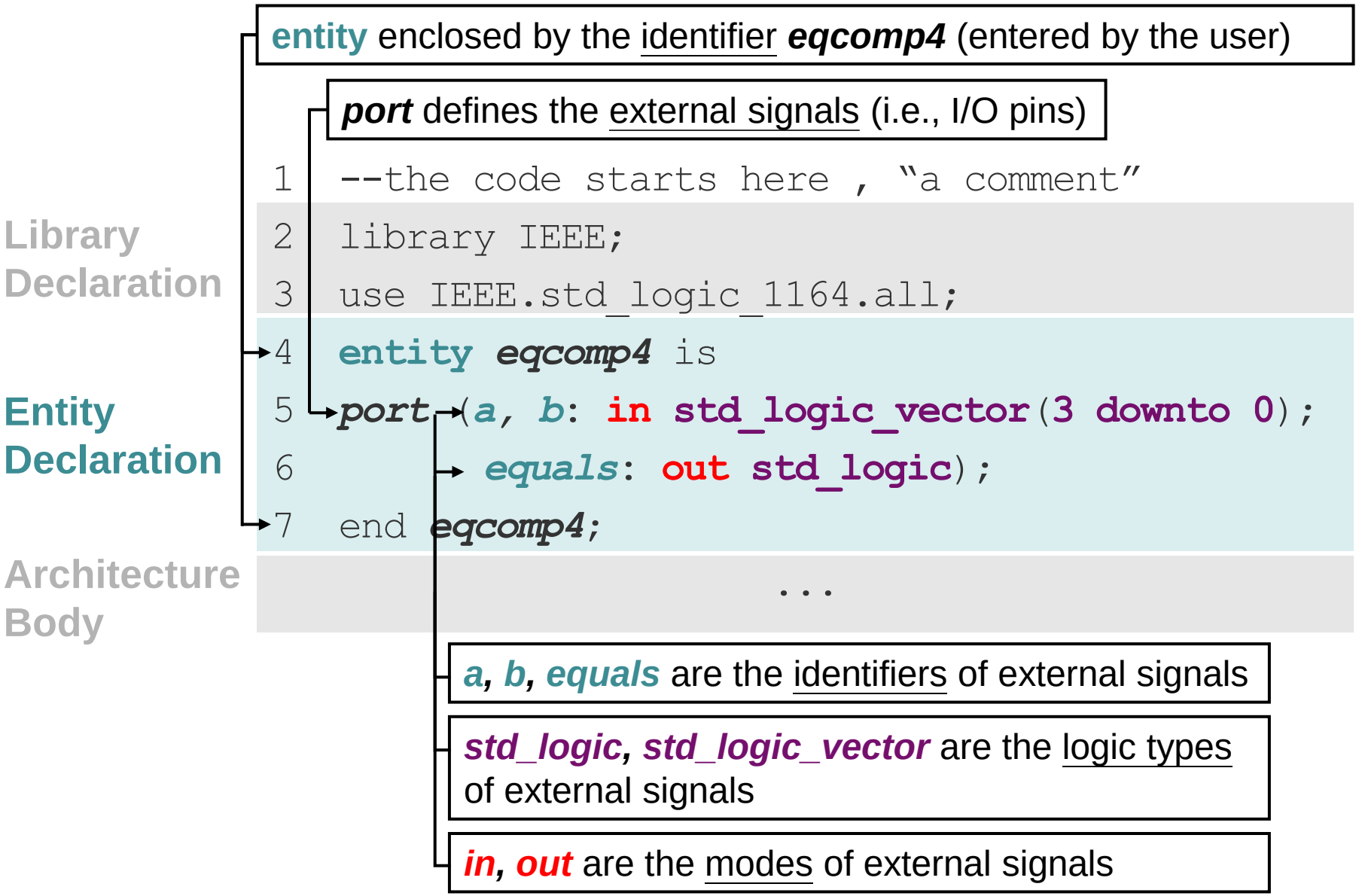
Architecture Body

```
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 line”
12 end arch_eqcomp4;
```

Outline

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

Entity Declaration



Identifiers

- Identifiers: Used to name any object in VHDL

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

VHDL Reserved Words

abs
access
after
alias
all
and
architecture
array
assert
attribute

begin
block
body
buffer
bus

case
component
configuration
constant

disconnect
downto

else
elsif
end
entity
exit

file
for
function

generate
generic
guarded

if
impure
in
inertial
inout
is

label
library
linkage
literal
loop

map
mod

nand
new
next
nor
not
null

of
on
open
or
others
out

package
port
postponed
procedure
process
pure

range
record
register
reject
rem
report
return
rol
ror

select
severity
shared
signal
sla
sll
sra
srl
subtype

then
to
transport
type

unaffected
units
until
use

variable

wait
when
while
with

xnor
xor

Class Exercise 1.1

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_

External Signals (I/O Pin)

- An external signal (or I/O pin) is a physical wire that can carry logic information.
- Many logic types are eligible for external signals, e.g.,
  - **bit**: can be logic '1' or '0' only
  - **std_logic**: can be logic 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', or '–'
    - 9-valued standard logic (IEEE standard 1164)
    - E.g., `equals: out std_logic;`
  - **std_logic_vector**: a group of wires (i.e., a signal bus)
    - E.g., `a, b: in std_logic_vector(3 downto 0);`
      - Each of `a(3), a(2), a(1), a(0)` is a **std_logic** signal.
- VHDL is strongly-typed language.
  - Signals of different base types CANNOT to assigned to each other without the use of type-conversion.

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

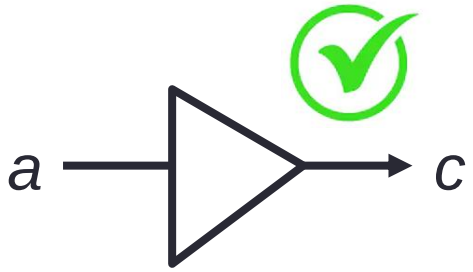
- Rule: When 2 signals meet, the forcing signal dominates.

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 Assignment:

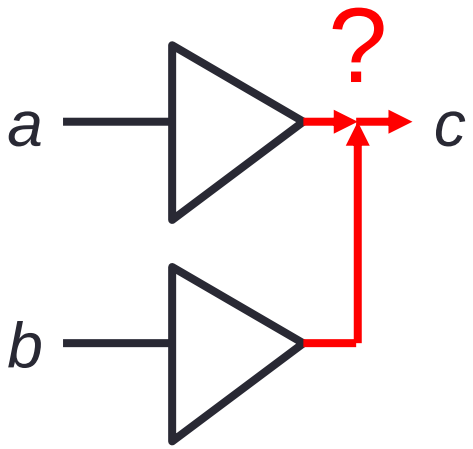
```
signal a, c: bit;
c <= a;
```



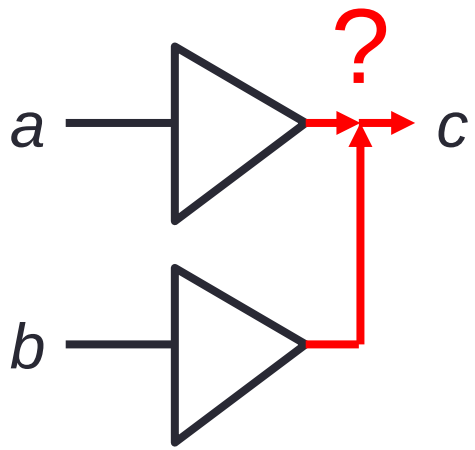
- Multi-value Signal Assignment:

```
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
```

Class Exercise 1.2

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: _____

Modes of I/O Pins

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

  Example:

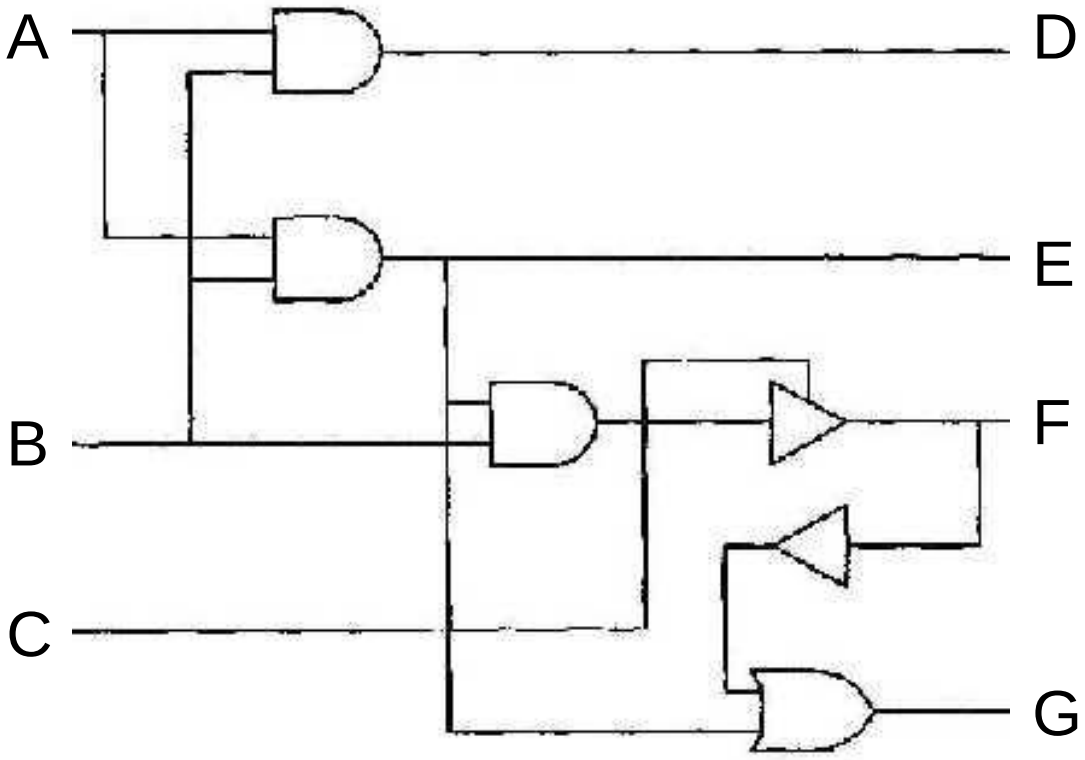
```
entity eqcomp4 is
  port (a, b: in std_logic_vector(3 downto 0);
        equals: out std_logic);
end eqcomp4;
```

- There are 4 available 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 1.3

Student ID: _____ Date:
Name: _____ _____

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



Reviews for Common Logic Gates

VHDL for programmable logic, Skahill, Addison Wesley

Outline

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

Architecture Body

- Architecture Body: Defines the internal of the chip

  Example: the architecture body of the entity ***eqcomp4***

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

  - **arch_eqcomp4**: the architecture identifier (entered by the user)
  - **equals**, **a**, **b**: I/O pins designed by the user in the entity declaration
  - **begin … end**: define the internal operation
  - `equals <= '1' when (a = b) else '0';`
    - <= here means "signal assignment" not "less than or equal".
    - `when-else` is a concurrent design construction.
  - `--`: comment on a line

Built-in Operators

- Logical Operators: `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not` have their usual meanings.
- Relation Operators (result is Boolean)

=	equal	<=	less than or equal
/=	not equal	>	greater than
<	less than	>=	greater than or equal

- 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

https://www.ics.uci.edu/~jmoorkan/vhdlref/Synario VHDL Manual.pdf

Outline

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

Keys to design complicated architecture body!

Internal Signal

- The entity declares the external signals.
- The architecture body can also declare signals that can be used internally.

```
architecture arch_eqcomp4 of eqcomp4 is
-- Internal signals shall be declared here!
begin
...
end arch_eqcomp4;
```

  - **Signal**: Represent physical wires
    - E.g., `signal s1: BIT := '1';`
  - **Constant**: Hold unchangeable values
    - E.g., `constant c1: BIT := '1';`

Signal Object

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

Note: Signals can be declared without initialized values.

- Examples:
  - `signal SIG_NAME: STD_LOGIC;`
    - Declared without initialized value
  - `signal SIG_NAME: STD_LOGIC := '1';`
- Signals can be declared
  - In the "`port`" of the entity declaration (as external signals);
  - Or in the architecture body (as internal signals).

Constant Object

```
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-";`
    - `'-'` means “don’t care”
- Constants can be declared in
  - Anywhere allowed for declaration.

Class Exercise 1.4

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 an internal signal named “connect1” in Line 6.
- Can you assign an I/O mode to this signal? Why?

_____

- Draw the schematic circuit for the code.

Outline

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

① Data Flow (Concurrent Statements)

- Data flow design method uses concurrent statements rather than ~~sequential statements~~.
  - Concurrent statements can be interchanged freely.
  - There's no "execution order" for concurrent statements.

```
 1 library IEEE; %Vivado2014.4 tested ok
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity eqb_comp4 is
 4 port (a, b: in std_logic_vector(3 downto 0);
 5       equals, bigger: out std_logic);
 6 end eqb_comp4;
 7 architecture dataflow4 of eqb_comp4 is
 8 begin
 9   equals <= '1' when (a = b) else '0'; --concurrent
10   bigger <= '1' when (a > b) else '0'; --concurrent
11 end dataflow4;
```

Lines 9 & 10 will be executed whenever signal a or b (or both) changes.

Class Exercise 1.5

Student ID: _____ Date:
Name: _____ _____

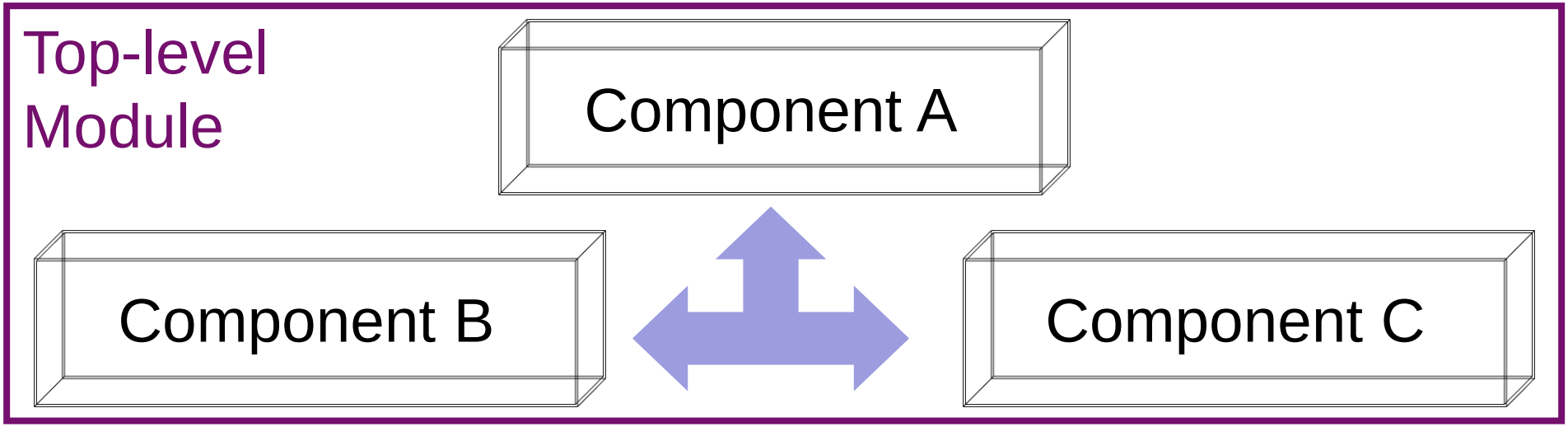
- Draw the schematic circuit of this code:

```
 1 library IEEE; --Vivado 14.4
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity abc is
 4   port (a,b,c: in std_logic;
 5               y: out std_logic);
 6 end abc;
 7 architecture abc_arch of abc is
 8 signal x : std_logic;
 9 begin
10   x <= a nor b;
11   y <= x and c;
12 end abc_arch;
```

Answer:

② Structural Design (use "port map")

- Structural Design: Like a circuit but describe it by text.

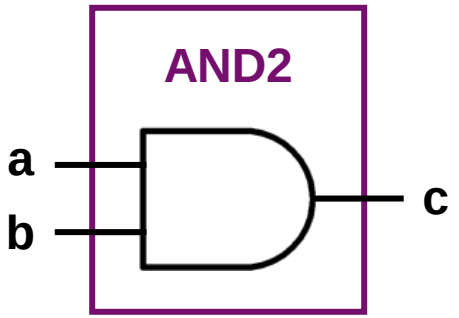


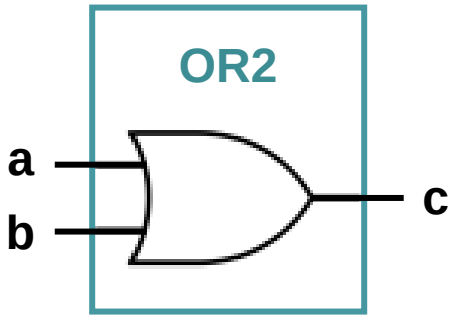
Connected by **`port map`** in the architecture body of the top-level design module

- **Design Steps**:
  - **Step 1**: Create **`entities`**
  - **Step 2**: Create **`components`** from **`entities`**
  - **Step 3**: Use "**`port map`**" to relate the components

Step 1: Create Entities

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity and2 is
 4 port (a,b: in STD_LOGIC;
 5         c: out STD_LOGIC );
 6 end and2;
 7 architecture and2_arch of and2 is
 8 begin
 9   c <= a and b;
10 end and2_arch;
11 --------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 entity or2 is
15 port (a,b: in STD_LOGIC;
16         c: out STD_LOGIC );
17 end or2;
18 architecture or2_arch of or2 is
19 begin
20   c <= a or b;
21 end or2_arch;
```



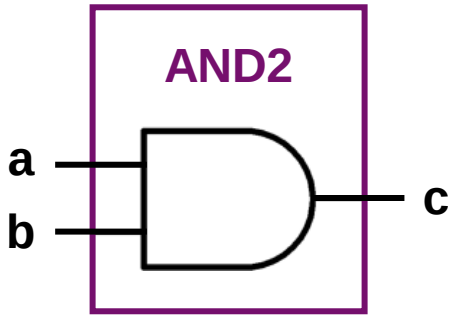


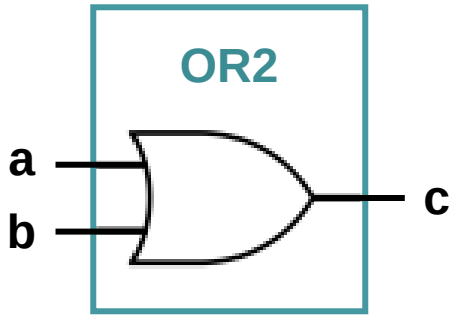
Step 2: Create Components

```
component and2 --create components--
   port (a,b: in std_logic; c: out std_logic);
end component;

component or2 --create components--
   port (a,b: in std_logic; c: out std_logic);
end component;

signal con1_signal: std_logic; --internal signal--
                               --   (optional)  --
```





Step 3: Connect Components

label1 & label 2 are line labels

```
begin
    label1: and2 port map (in1, in2, inter_sig);
    label2: or2 port map (inter_sig, in3, out1);
end test_arch;
```

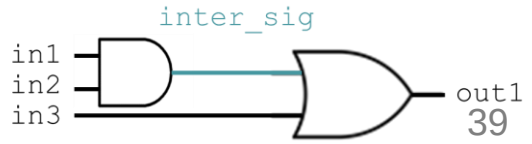
Lines can be interchanged for the same circuit design.

Put Together: A Running Example

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity and2 is                          Step 1
 4 port (a,b: in STD_LOGIC;
 5         c: out STD_LOGIC );
 6 end and2;
 7 architecture and2_arch of and2 is
 8 begin
 9   c <= a and b;
10 end and2_arch;
11 -------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 entity or2 is                           Step 1
15 port (a,b: in STD_LOGIC;
16         c: out STD_LOGIC );
17 end or2;
18 architecture or2_arch of or2 is
19 begin
20   c <= a or b;
21 end or2_arch;
```

Top-level Module

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 ---------------------------------------
 4 entity test is
 5 port ( in1: in STD_LOGIC; in2: in STD_LOGIC;
 6        in3: in STD_LOGIC;
 7       out1: out STD_LOGIC );
 8 end test;
 9 architecture test_arch of test is
10 component and2 --create component          Step 2
11   port (a,b: in std_logic; c: out std_logic);
12 end component ;
13 component or2 --create component
14   port (a,b: in std_logic; c: out std_logic);
15 end component ;
16 signal inter_sig: std_logic;
17 begin                                      Step 3
18   label1: and2 port map (in1, in2, inter_sig);
19   label2: or2 port map (inter_sig, in3, out1);
20 end test_arch;
```

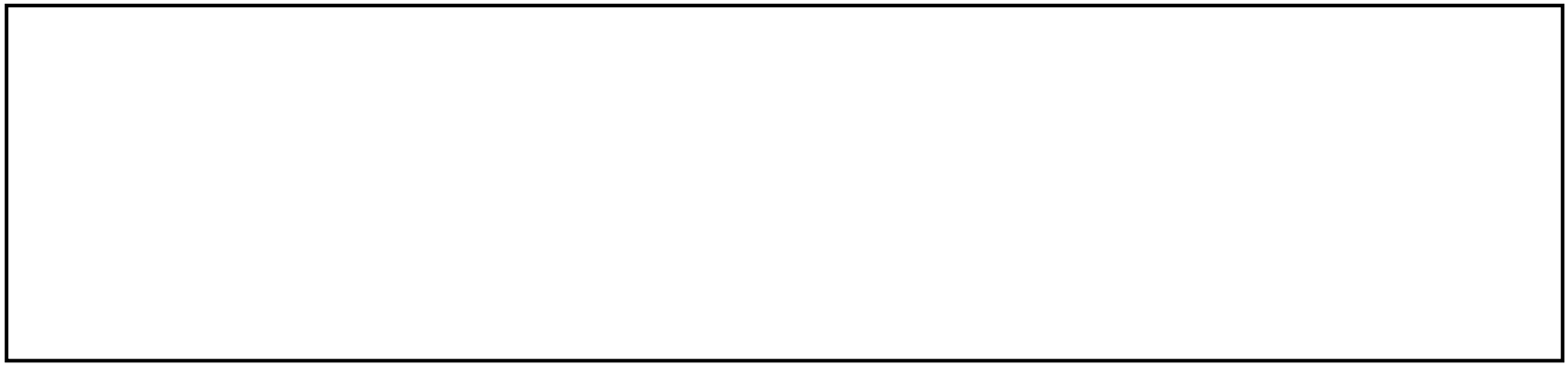


Class Exercise 1.6

Student ID: _____ Date:
Name: _____ _____

- Draw the schematic diagram for the statements:

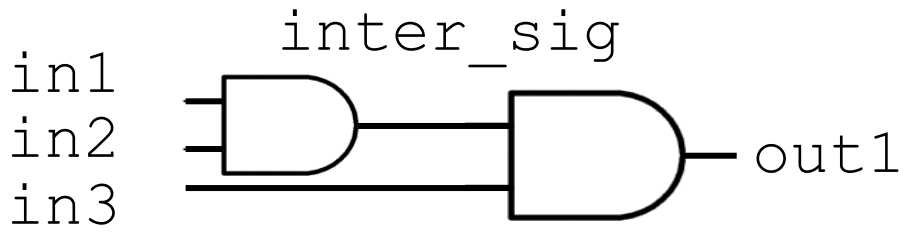
```
 i label_u0: and2 port map (a, c, x);
ii label_u1: or2 port map (b, x, y);
```



- When will Lines i and ii be executed?
- Answer:
  - Line i: _____
  - Line ii: _____

Another Running Example

```
entity test_andand2 is
port ( in1: in STD_LOGIC;
       in2: in STD_LOGIC;
       in3: in STD_LOGIC;
      out1: out STD_LOGIC
  );
end test_andand2;
architecture test_andand2_arch of test_andand2 is
component and2
  port (a, b: in std_logic; c: out std_logic);
end component ;
signal inter_sig: std_logic;
begin
    label1: and2 port map (in1, in2, inter_sig);
    label2: and2 port map (inter_sig, in3, out1);
end test_andand2_arch;
```



No need to create the component for the same entity for several times

But you can use the component multiple times

② Structural vs. ① Data Flow

② Structural

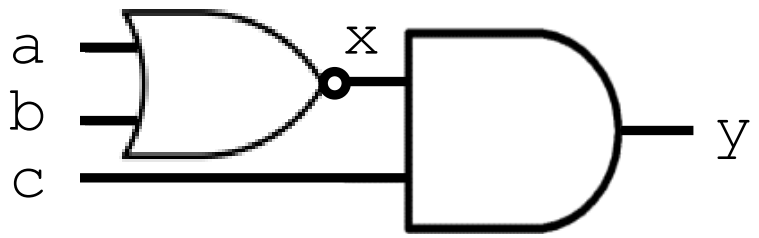
("port map")

```
architecture test_arch of test is
component and2
  port (a,b: in std_logic;
        c: out std_logic);
end component ;
component nor2
  port (a,b: in std_logic;
        c: out std_logic);
end component ;
signal x: std_logic;
begin
  label1: nor2 port map (a, b, x);
  label2: and2 port map (x, c, y);
end test_arch;
```

① Data Flow

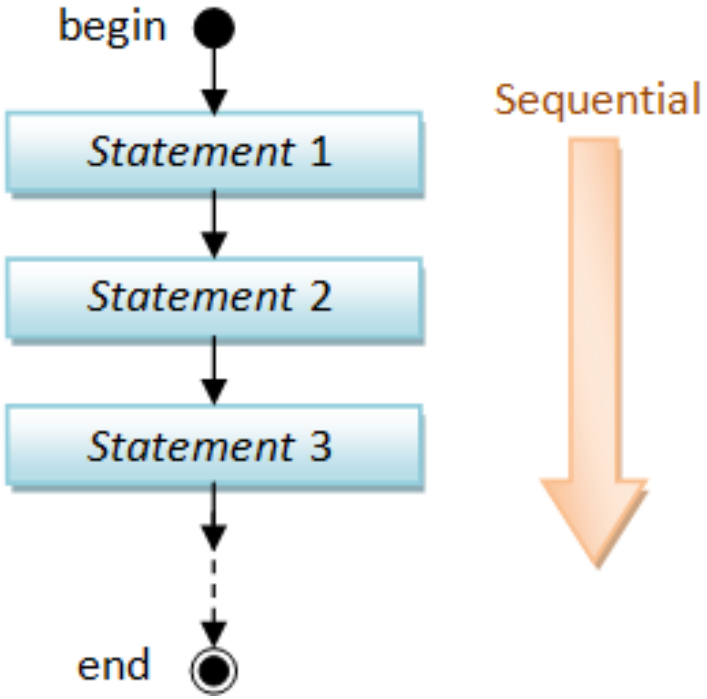
(concurrent statements)

```
architecture test_arch of test is
signal x : std_logic;
begin
  x <= a nor b;
  y <= x and c;
end test_arch;
```



③ Behavioral Design (use "process")

- Behavioral design is sequential
  - Just like a sequential program



- The keyword is "**`process`**":
  - The main character is "**`process (sensitivity list)`**".
  - A **`process`** is executed when one (or more) of the signals in the **`sensitivity list`** changes.
  - Statements inside a process are sequentially executed.

Behavioral Design Example

```
library IEEE; --vivado14.4
use IEEE.STD_LOGIC_1164.ALL;
entity eqcomp4 is
port (a, b: in std_logic_vector(3 downto 0);
    equals: out std_logic);
end eqcomp4;
architecture behavioral of eqcomp4 is
begin
  process(a, b)
  begin
    if a = b then
        equals <= '1';
    else
        equals <= '0';
    end if;
  end process;
end behavioral;
```

← **Behavioral Design**: Sequential in a "**process**"

Sequential Execution:
Statements inside a process are sequentially executed.

Another Example? See Lab01

Hardware

```
entity AND_Gate is
    port ( A: in STD_LOGIC;
           B: in STD_LOGIC;
           C : out STD_LOGIC);
end AND_Gate;

architecture AND_arch of
AND_Gate is
begin
    C <= A and B;
end AND_arch;
```

1) It is legal to have a process WITHOUT a **sensitivity list**.
2) Such process MUST have some kinds of **time-delay** or **wait** (see Lec03 for more examples).

Simulation

```
architecture Behavioral of AND_TEST is
component AND_Gate
    port(A, B: in STD_LOGIC;
             C: out STD_LOGIC);
end component;
signal ai, bi: STD_LOGIC;
signal ci: STD_LOGIC;
begin
    AND_Gate port map (A => ai, B => bi,
                       C => ci);
process
begin
    ai <= '0'; bi <= '0';
    wait for 100 ns;
    ai <= '1'; bi <= '0';
    wait for 100 ns;
    ai <= '0'; bi <= '1';
    wait for 100 ns;
    ai <= '1'; bi <= '1';
    wait;
end process;
end Behavioral;
```

Outline

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

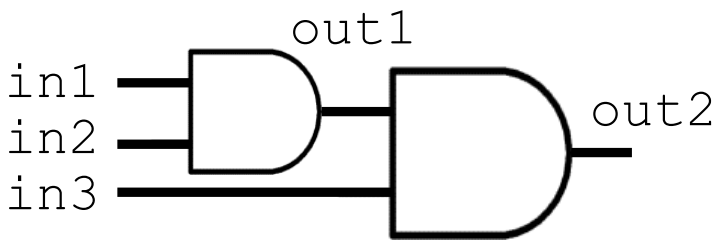
Concurrent vs. Sequential Statements

- **Concurrent Statement**
  1) Statements inside the architecture body can be executed concurrently, except statements enclosed by a **`process`**.
  2) Every statement will be executed once whenever any signal in the right-hand-side of statement changes.

- **Sequential Statement**
  1) Statements within a **`process`** are executed sequentially, and the result is obtained when the process is complete.
  2) **`process(sensitivity list)`**: Whenever any signals in the sensitivity list changes its state, the process executes once.
  3) A **`process`** can be treated as one concurrent statement in the architecture body.

Concurrent with Sequential

```
 1 library IEEE; --vivado14.4 ok
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity conc_ex is
 4 port (in1,in2,in3: in std_logic;
 5         out1,out2 : inout std_logic);
 6 end conc_ex;
 7 architecture for_ex_arch of conc_ex is
 8 begin
 9 process (in1, in2)
10 begin
11   out1 <= in1 and in2;
12 end process;
13 out2 <= out1 and in3; -- concurrent statement
14 end for_ex_arch;
```

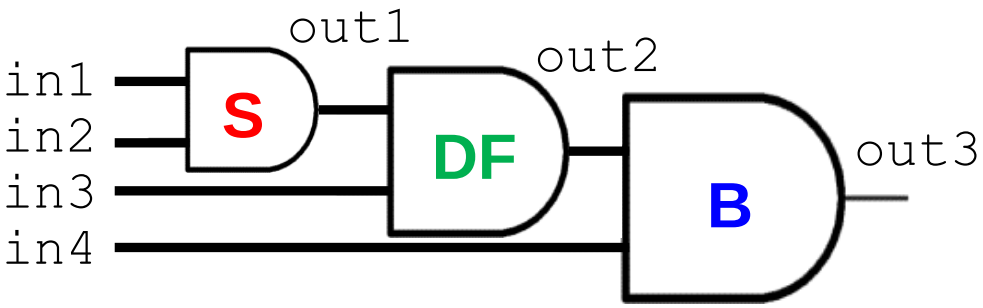


The process (9-12) and line 13 are concurrent and can be interchanged!

Class Exercise 1.7

Student ID: _____ Date:
Name: _____ _____

- Use **structural**, **data flow**, and **behavioral** designs to implement the following circuit in VHDL:



Outline

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("`port map`")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

Design Constructions

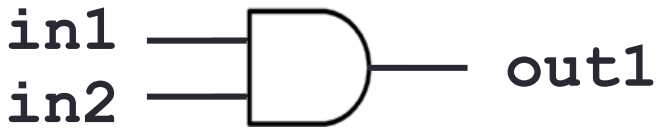
- **Concurrent**: Statements that can be stand-alone
  1) `when-else`
  2) `with-select-when`

Concurrent: **OUTSIDE** process

- **Sequential**: Statements inside the **`process`**
  1) `if-then-else`
  2) `case-when`
  3) `for-in-to-loop`

Sequential – **INSIDE** process

Concurrent 1) when-else




```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity when_ex is
 4 port (in1, in2 : in std_logic;
 5            out1 : out std_logic);
 6 end when_ex;
 7 architecture when_ex_arch of when_ex is
 8 begin
 9   out1 <= '1' when in1 = '1' and in2 = '1' else '0';
10 end when_ex_arch;
```

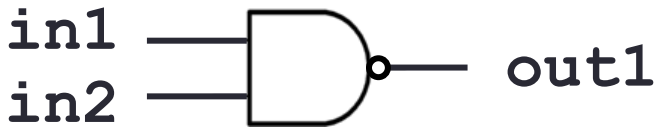
Condition based

*when **condition** is true then out1 <= '1'*
otherwise then out1 <= '0'

Class Exercise 1.8

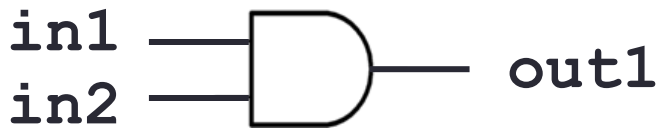
Student ID: _____ Date:
Name: _____ _____

- Fill in line 9 using **`when-else`**:




```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity when_ex is
 4 port (in1, in2 : in std_logic;
 5               out1 : out std_logic);
 6 end when_ex;
 7 architecture when_ex_arch of when_ex is
 8 begin
 9 ____________________________________________________________
10 end when_ex_arch;
```

Concurrent 2) with-select-when




```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity when_ex is
 4 port (in1, in2 : in std_logic;
 5           out1 : out std_logic);
 6 end when_ex;
 7 architecture when_ex_arch of when_ex is
 8 begin
 9   with in1 select
10     out1 <= in2 when '1',
11             '0' when others;
12 end when_ex_arch;
```

Signal based

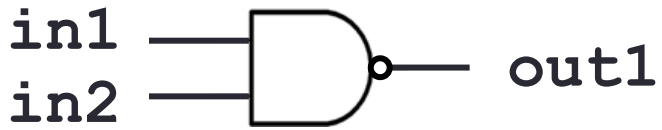
← *when **in1**='1' then out1 <= in2*

← *when **in1** = other cases then out1 <= '0'*

Class Exercise 1.9

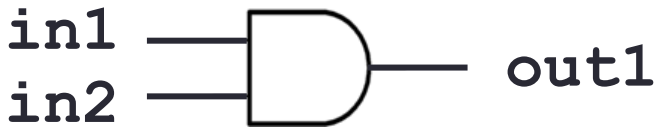
Student ID: _____ Date:
Name: _____ _____

- Fill in lines 10~11 using **`with-select-when`**:




```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity when_ex is
 4 port (in1, in2 : in std_logic;
 5               out1 : out std_logic);
 6 end when_ex;
 7 architecture when_ex_arch of when_ex is
 8 begin
 9   with in1 select
10 ____________________________________________________
11 ____________________________________________________
12 end when_ex_arch;
```

when-else VS. with-select-when



- Concurrent 1) `when-else`: **Condition** based

```
out1 <= '1' when in1 = '1' and in2 = '1' else '0';
```

*when **in1='1' and in2='1'** then out1 <= '1', otherwise out <= '0'*

- Concurrent 2) `with-select-when`: **Signal** based

```
with in1 select
  out1 <= in2 when '1',
          '0' when others;
```

← *when **in1**='1' then out1 <= in2*

← *when **in1** = other cases then out1 <= '0'*

Design Constructions

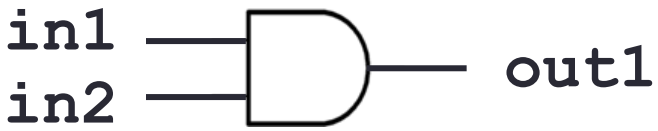
- **Concurrent**: Statements that can be stand-alone
  1) `when-else`
  2) `with-select-when`

Concurrent: **OUTSIDE** process

- **Sequential**: Statements inside the **`process`**
  1) `if-then-else`
  2) `case-when`
  3) `for-in-to-loop`

Sequential – **INSIDE** process

Sequential 1) if-then-else




```
entity if_ex is
    port(in1,in2: in std_logic;
             out1: out std_logic);
end if_ex;
architecture if_ex_arch of if_ex is
begin
    process(b)
    begin
        if in1 = '1' and in2 = '1' then
            out1 <= '1';
        else
            out1 <= '0';
        end if;
end process;
end if_ex_arch;
```

```
if (cond) then
    statement;
end if;
```

```
if (cond) then
    statement1;
else
    statement2;
end if;
```

```
if (cond1) then
    statement1;
elsif (cond2) then
    statement2;
elsif …
    …
else
    statementn;
end if;
```

Sequential 2) case-when

```
entity test_case is
    port ( in1, in2: in std_logic;
           out1,out2: out std_logic);
end test_case;
architecture case_arch of test_case is
signal b : std_logic_vector (1 downto 0);
begin
    process (b)
    begin
        case b is
        when "00"|"11" => out1 <= '0';
                          out2 <= '1';
        when others    => out1 <= '1';
                          out2 <= '0';
        end case;
    end process;
    b <= in1 & in2;
end case_arch;
```

"00"|"11" means case "00" or "11"

"=>" means "implies" not "bigger"

All cases must be present:
*Use **others** to complete all cases*

Class Exercise 1.10

Student ID: _____ Date:
Name: _____ _____

```
 1 entity test_case is
 2    port ( in1, in2: in std_logic;
 3           out1,out2: out std_logic);
 4 end test_case;
 5 architecture case_arch of test_case is
 6 signal b : std_logic_vector (1 downto 0);
 7 begin
 8    process (b)
 9    begin
10        case b is
11        when "00"|"11" => out1 <= '0';
12                          out2 <= '1';
13        when others    => out1 <= '1';
14                          out2 <= '0';
15        end case;
16    end process;
17    b <= in1 & in2;
18 end case_arch;
```

- List line numbers of concurrent statements:
  Answer:

  _____

  _____

- Fill in the truth table:

b(1)	b(0)	out1	out2
0	0		
0	1		
1	0		
1	1		

Concurrent vs. Sequential Constructions

Concurrent

when-else

```
b <= "1000" when a = "00" else
     "0100" when a = "01" else
     "0010" when a = "10" else
     "0001" when a = "11";
```

with-select-when

```
with a select
b <= "1000" when "00",
     "0100" when "01",
     "0010" when "10",
     "0001" when "11";
```

Sequential

if-then-else

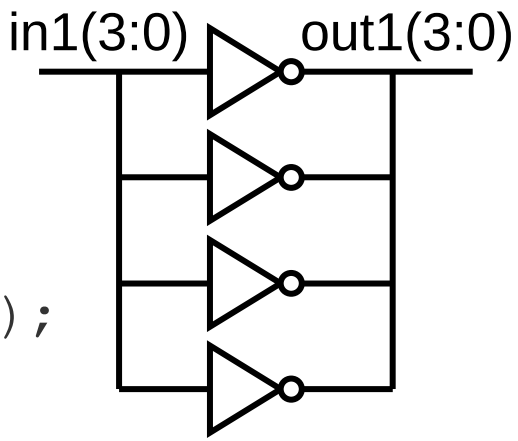
```
if    a = "00" then b <= "1000"
elsif a = "01" then b <= "1000"
elsif a = "10" then b <= "1000"
else                  b <= "1000"
end if;
```

case-when

```
case a is
    when "00" => b <= "1000";
    when "01" => b <= "0100";
    when "10" => b <= "0010";
    when others => b <= "0001";
end case;
```

Sequential 3) loop (1/2)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity for_ex is
port (in1: in std_logic_vector(3 downto 0);
      out1: out std_logic_vector(3 downto 0));
end for_ex;
architecture for_ex_arch of for_ex is
begin
```



for-loop

```
process (in1)
begin
  for i in 0 to 3 loop
    out1(i) <= not in1(i);
  end loop;
end process;
```

while-loop

```
process (in1)
variable i: integer := 0;
begin
    i := 0;
    while i < 4 loop
      out1(i) <= not in1(i);
      i := i + 1;
    end loop;
end process;
```

```
end for_ex_arch;
```

Sequential 3) loop (2/2)

- **for-loop**

```
for i in 0 to 3 loop
  out1(i) <= not in1(i);
end loop;
```

  - No need to declare the loop index (e.g., **i**).
    - It is implicitly declared within the loop.
    - It may not be modified within the loop (e.g., **i := i-1;**).
  - for-loop is generally supported for synthesis.

- **while-loop**

```
variable i: integer:=0;
…
while i < 4 loop
  out1(i) <= not in1(i);
  …
end loop;
```

  - The while loop repeats if the condition tested is true.
    - The condition is tested before each iteration.
  - while-loop is supported by some logic synthesis tools with restrictions.

https://www.ics.uci.edu/~jmoorkan/vhdlref/for_loop.html
https://www.ics.uci.edu/~jmoorkan/vhdlref/while.html

Variable Object

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

Note: Variables can be declared without initialized values.

- Examples:
  - `variable VAR_NAME: STD_LOGIC;`
    - Declared without initialized value
  - `variable VAR_NAME : STD_LOGIC := '1';`
- Variables can only be declared/used in the `process`.
- Variables are used only by programmers for internal representation (less direct relationship to hardware).

Signal vs. Variable Assignment

- Both signals and variables can be declared without initialized values.
  - `signal SIG_NAME: <type> [:= <value>];`
  - `variable VAR_NAME: <type> [:= <value>];`

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

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

  - Syntax of variable assignment:

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

Class Exercise 1.11

Student ID: _____ Date:
Name: _____ _____

- Rewrite *arch1* without using a process( )

```
architecture arch1 of ex1
is
begin
  process (in1)
  begin
    for i in 0 to 3 loop
      out1(i) <= not in1(i);
    end loop;
  end process;
end for_ex_arch;
```

```
architecture arch1 of ex1
is
begin
```

```
end for_ex_arch;
```

Summary

- Basic Structure of a VHDL Module
  1) Library Declaration
  2) Entity Declaration
     - External Signal (I/O Pins)
  3) Architecture Body
     - Internal Signal
     - Architectural Design Methods
       ① Data Flow Design (concurrent statements)
       ② Structural Design ("**`port map`**")
       ③ Behavioral Design (sequential statements)
     - Concurrent vs. Sequential Statements
     - Design Constructions

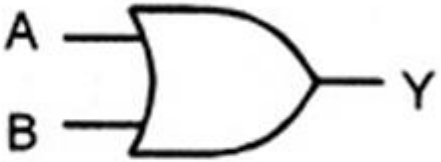
Review: Basic Gates in Logic Circuits

2-input AND gate

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

$Y = A \bullet B$

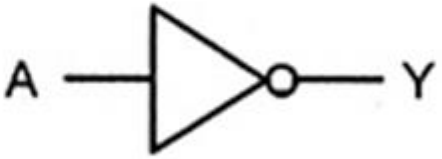
2-input OR gate



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

$Y = A + B$

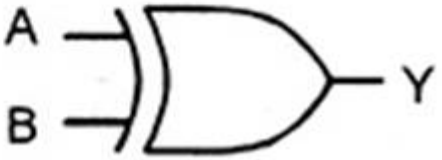
Inverter (NOT gate)



A	Y
0	1
1	0

$Y = \overline{A}$

2-input EX-OR gate



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

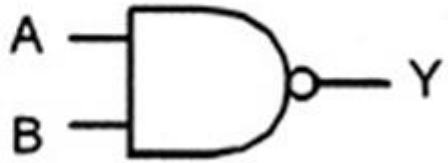
$Y = A \oplus B$

http://www.nutsvolts.com/magazine/article/understanding_digital_buffer_gate_and_ic_circuits_part_1

Review: NAND and NOR Gates

- In many technologies, implementation of NAND gates or NOR gates is easier than that of AND or OR gates.
  - NAND Gate:

2-input NAND gate

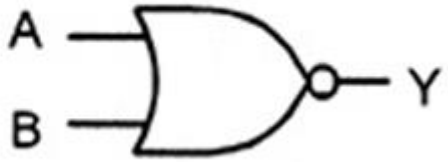


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

$Y = \overline{A \bullet B}$

  - NOR Gate:

2-input NOR gate



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

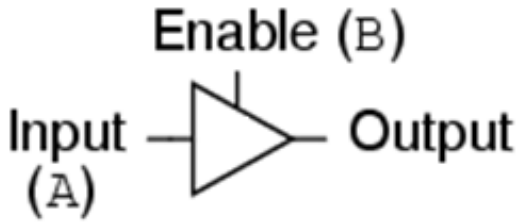
$Y = \overline{A + B}$

- Any logic function can be realized using only NAND gates or only NOR gates.

Review: Tristate Logic

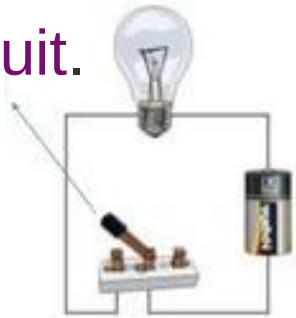
- The concept of tristate logic is also essential in digital system designs.
  - Directly connecting outputs of two gates together might not operate properly, and might cause damage to the circuit.
  - One ways is to use tristate buffers.
- Tristate buffers are gates with a high impedance state (High-Z or Z) in addition to high and low logic states.
  - High impedance state is equivalent to an open circuit.

Tristate buffer symbol



Truth table

A	B	Output
0	0	High-Z
0	1	0
1	0	High-Z
1	1	1

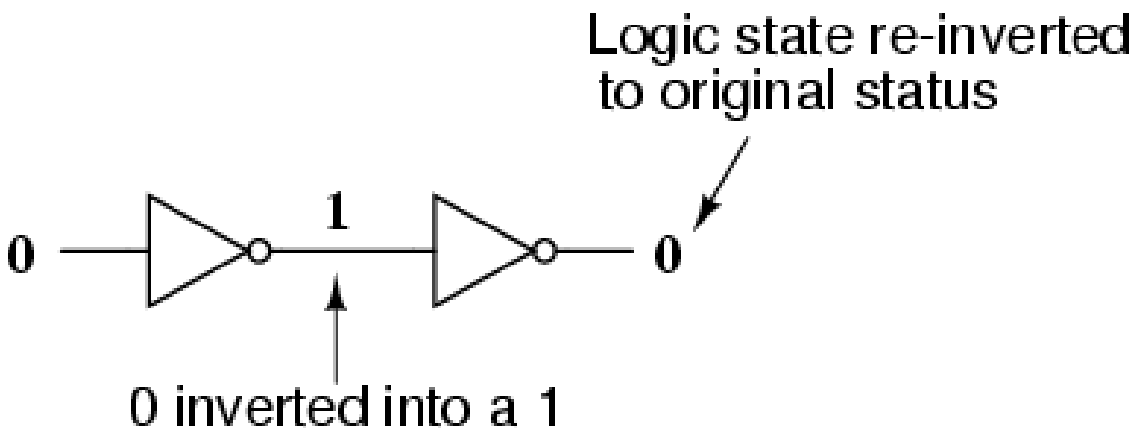


(analogy)

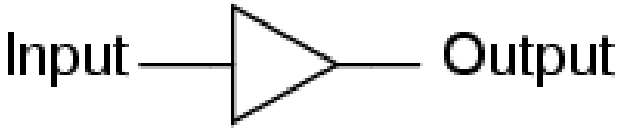
Review: Buffer Gate

- Double inversion would "cancel" each other out.
  - A weak signal may be amplified by means of two inverters.
- For this purpose, a special logic gate called a buffer gate is manufactured to perform the double inversion.
  - Its symbol is simply a triangle, with no inverting "bubble" on the output terminal:

Double inversion



"Buffer" gate



Input	Output
0	0
1	1

https://www.allaboutcircuits.com/textbook/digital/chpt-3/buffer-gate/