香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 03: Architectural Styles of VHDL

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Recall: What we have done in 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;
```

Simulation

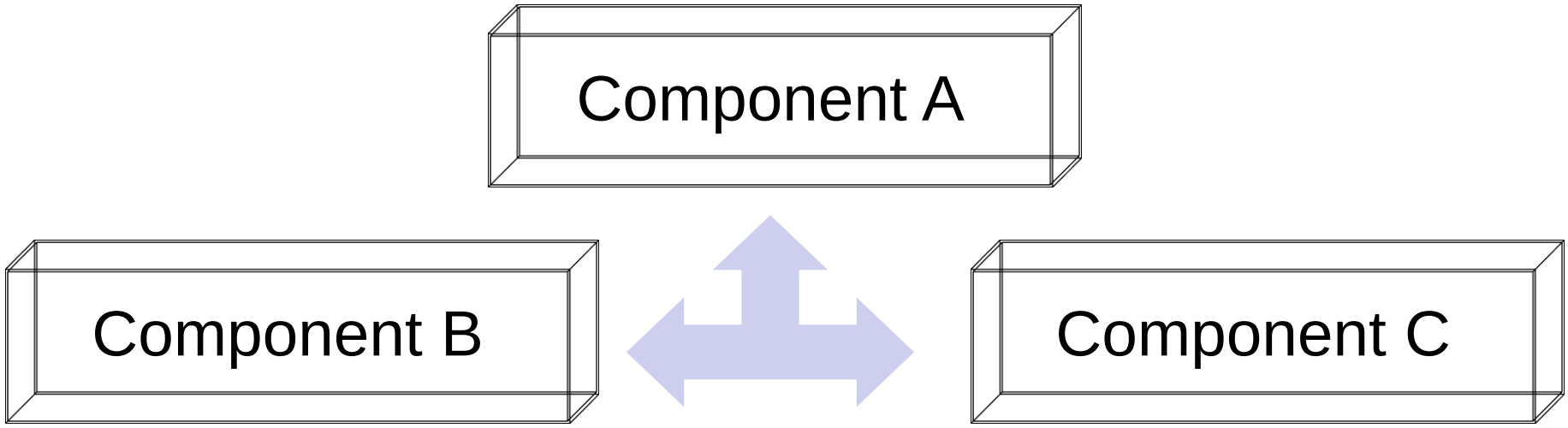
```
architecture Behavioral of AND_TEST is
component AND_Gate
    port(A, B: in STD_LOGIC;
             C: out STD_LOGIC);
end component;
signal ai, bi: STD_LOGIC;
signal ci: STD_LOGIC;
begin
    AND_Gate port map (A => ai, B => bi,
                       C => ci);
    process
    begin
        ai <=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

- Architectural Design Methods
  - Structural Design ("**`port map`**")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("**`process`**")
- Concurrent vs. Sequential Statements
- Revisit: Signal (**<=**) and Variable (**:=**) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

Structural Design: Use "port map"

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

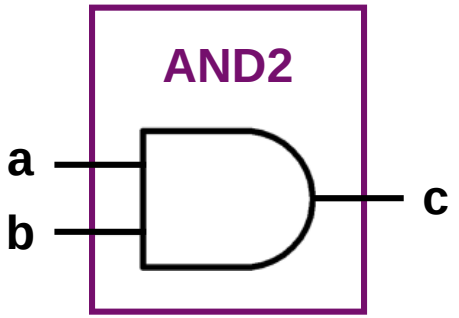


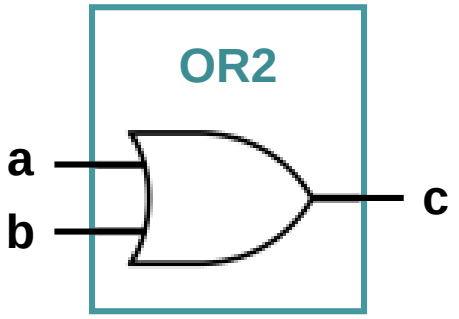
Connected by **`port map`** in architecture body

- 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;
```



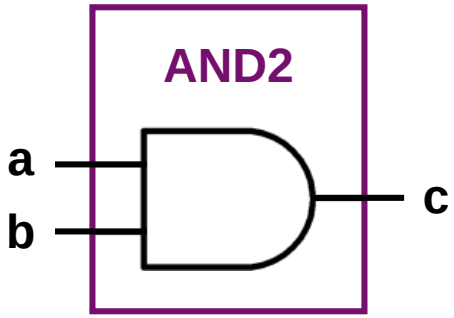


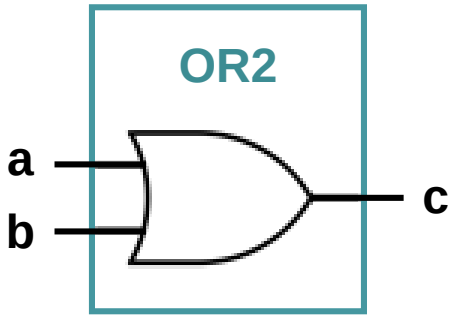
Step 2: Create Components

```
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;
```

```
 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 3.1

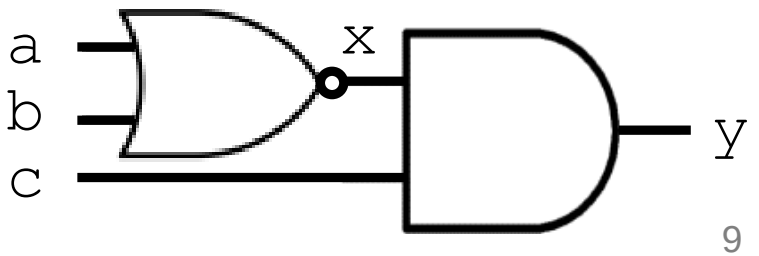
Student ID: _____ Date:
Name: _____ _____

- Draw the schematic diagram for the following lines:

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

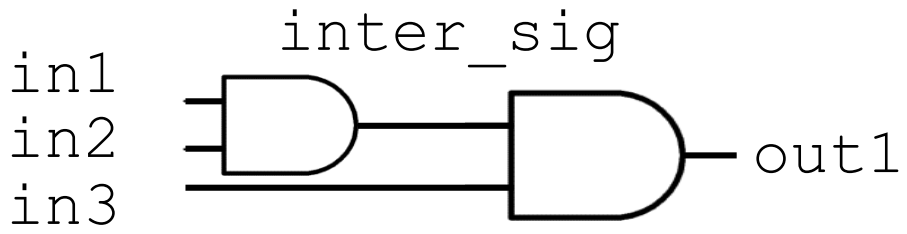
- When will lines i and ii be executed?
  Answer: _____
- Complete lines i and ii if the circuit is as follows:

```
 i label_u0: _____
ii label_u1: _____
```



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

Class Exercise 3.2

Student ID: _____ Date:
Name: _____ _____

- Draw the schematic diagram and fill in the truth table for the following the half-adder:

```
library IEEE; --Vivado 14.4 ok
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is -- another example
port (  x: in bit;    y: in bit;
      sum: out bit;  carry: out bit );
end half_adder;
architecture arch of half_adder is
component xor2
  port(a,b: in bit; c: out bit);
end component;
component and2
  port(a,b: in bit; c: out bit);
end component;
begin
   label1: xor2 port map (x, y, sum);
   label2: and2 port map (x, y, carry);
end arch;
```

input		output	
x	y	carry	sum

Outline

- Architectural Design Methods
  - Structural Design ("`port map`")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Revisit: Signal (`<=`) and Variable (`:=`) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

Data Flow: Use Concurrent Statements

- Data flow design method uses concurrent statements instead of “port map”.
  - 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 3.3

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:

Structural vs. Data Flow

Structural

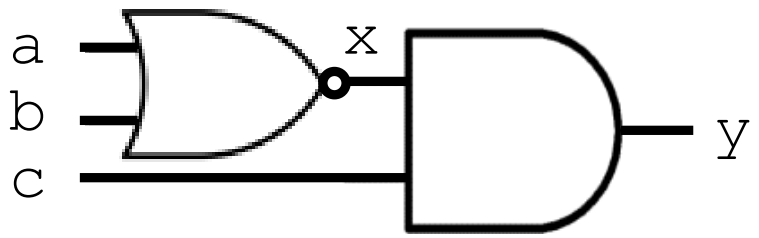
(port map)

```
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;
```

Data Flow

(concurrent statements)

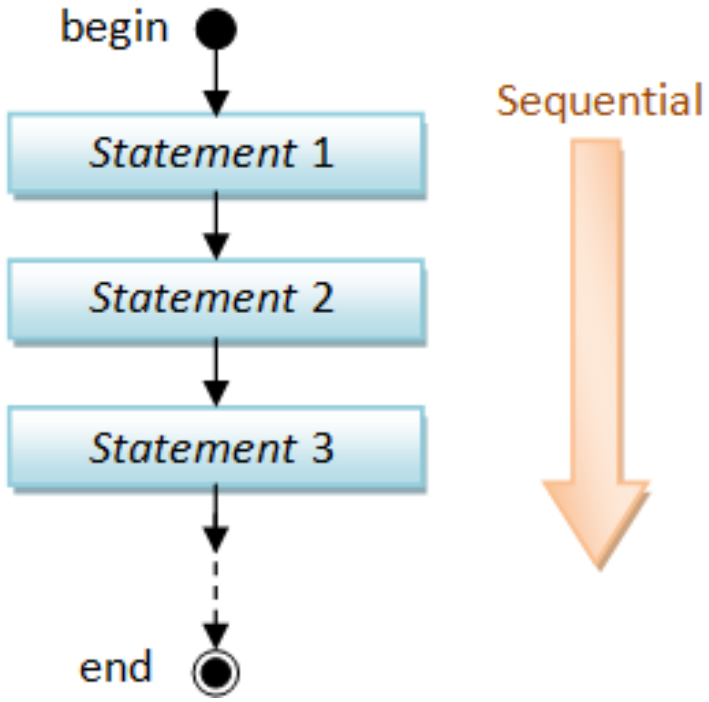
```
architecture test_arch of test is
signal x : std_logic;
begin
  x <= a nor b;
  y <= x and c;
end test_arch;
```



Outline

- Architectural Design Methods
  - Structural Design ("`port map`")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Revisit: Signal (`<=`) and Variable (`:=`) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

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(
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.

Recall: What we have done in 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 kind of **time-delay** or **wait** (Lec05).

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;
```

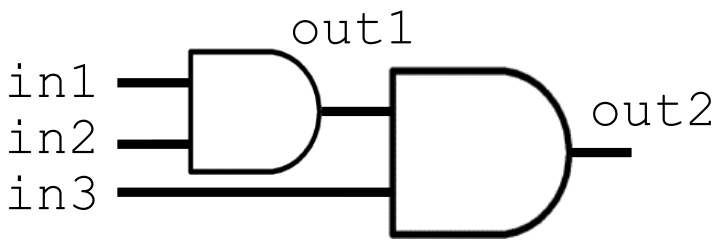
Outline

- Architectural Design Methods
  - Structural Design ("`port map`")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Revisit: Signal (`<=`) and Variable (`:=`) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

Concurrent vs. Sequential Statements

- **Concurrent Statement**
  - Statements inside the architecture body can be executed concurrently, except statements enclosed by a **`process`**.
  - Every statement will be executed once whenever any signal in the statement changes.
- **Sequential Statement**
  - Statements within a **`process`** are executed sequentially, and the result is obtained when the process is complete.
  - **`process(sensitivity list)`**: When one or more signals in the sensitivity list change state, the process executes once.
  - 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 3.4

Student ID: _____ Date:
Name: _____ _____

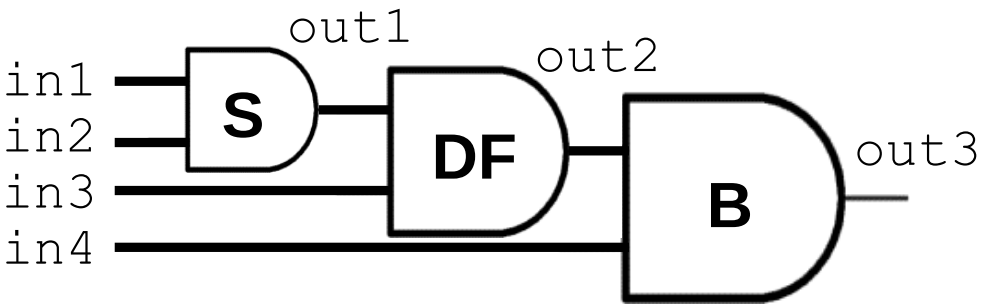
- State concurrent and sequential statements:

```
1 architecture for_ex_arch of for_ex is
2 begin
3    outx1 < = out1 and in3;
4    process (in1, in2)
5    begin
6       out1 <= in1 and in2;
7    end process;
8    outx2 < = out1 or in3;
9 end for_ex_arch;
```

Class Exercise 3.5

Student ID: _____ Date:
Name: _____ _____

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



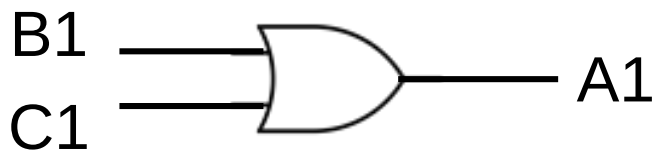
Outline

- Architectural Design Methods
  - Structural Design ("`port map`")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Revisit: Signal (<=) and Variable (:=) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

Signal Assignment (<=)

- **Signal Assignment (<=)**
  - Global to the **`entity`**
  - Concurrent execution
  - Do not be confused with the operator **<=** (equal or smaller)

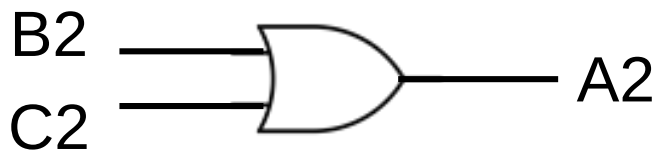
- For example: **`A1 <= B1 or C1`**
  - A1 must be declared outside a **`process`**.
  - A1 represents an internal wire or an input/output pin in port.



Variable Assignment (:=)

- **Variable Assignment (:=)**
  - Local to a **`process`**
  - Sequential execution
  - Constant/signal/variable initialization also uses "`:=`"

- For example: **`A2 := B2 or C2`**
  - A2 must be declared inside a **`process`**.
  - A2 must be a variable.



Outline

- Architectural Design Methods
  - Structural Design ("`port map`")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Revisit: Signal (`<=`) and Variable (`:=`) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

Design Constructions

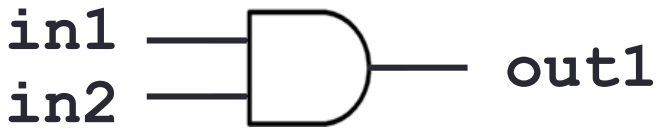
- **Concurrent**: Statements that can be stand-alone
  1) `when-else`
  2) `with-select-when`

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;
```

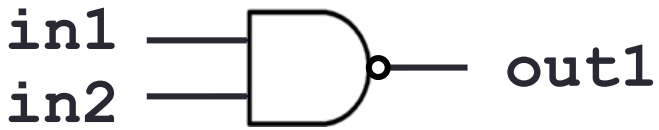
Condition based

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

Class Exercise 3.6

Student ID: _____ Date:
Name: _____ _____

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

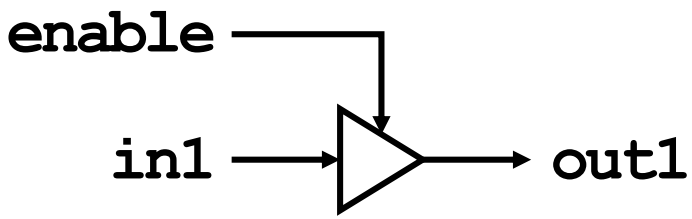



```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity when_ex is
 4 port (in1, in2 : in std_logic;
 5               out1 : out std_logic);
 6 end when_ex;
 7 architecture when_ex_arch of when_ex is
 8 begin
 9 ____________________________________________________________
10 end when_ex_arch;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

Class Exercise 3.7

Student ID: _____ Date:
Name: _____ _____

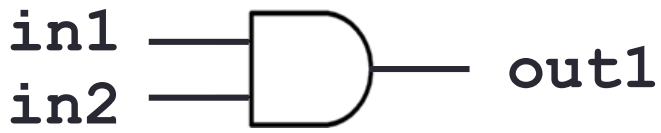
- Fill in the empty line to realize tri-state logic:



in1	enable	out1
0	0	Z
1	0	Z
0	1	0
1	1	1

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity tri_ex is
 4 port (in1, enable: in std_logic;
 5                ut1: out std_logic);
 6 end tri_ex;
 7 architecture tri_ex_arch of tri_ex is
 8 begin
 9 ____________________________________________________________
11 end tri_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

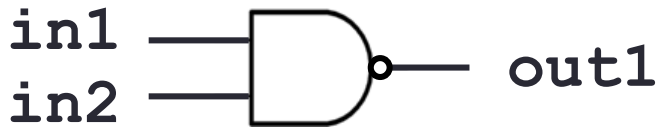
← *when **in1**='1' then out1 <= in2*

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

Class Exercise 3.8

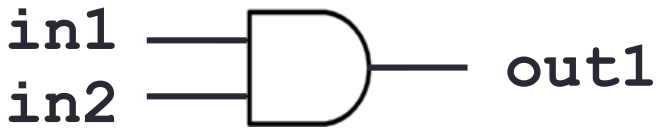
Student ID: _____ Date:
Name: _____ _____

- Fill in lines 9~11 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 ____________________________________________________
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) `when-else`: **Signal** based

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

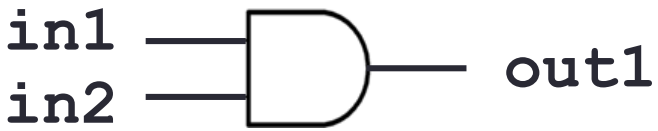
← *when **in1**='1' then out1 <= in2*

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

Outline

- Architectural Design Methods
  - Structural Design ("`port map`")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Revisit: Signal (`<=`) and Variable (`:=`) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

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 3.9

Student ID: _____ Date:
Name: _____ _____

```
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;
```

- 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

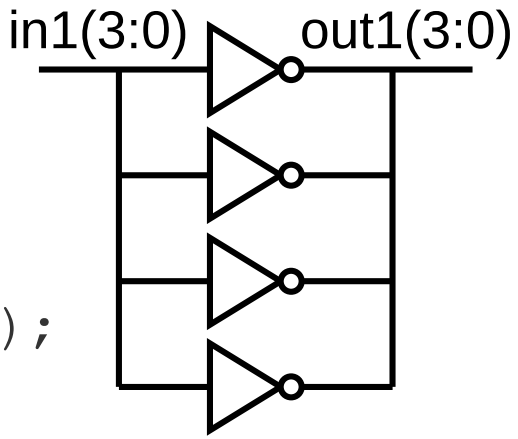
```
if    a = "00" then b <= "1000"
elsif a = "01" then b <= "1000"
elsif a = "10" then b <= "1000"
else                  b <= "1000"
end if;
```

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
    while i < 3 loop
      out1(i) <= not in1(i);
    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;
```

  - The loop parameter (e.g., `i`) does NOT need to be declared.
    - It is implicitly declared within the loop.
    - It may not be modified within the loop (e.g., `i := i-1;`).
  - `for-loop` is supported for synthesis.

- **`while-loop`**

```
variable i: integer:=0;
…
while i < 3 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 certain restrictions.

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

Class Exercise 3.10

Student ID: _____ Date:
Name: _____ _____

- Rewrite *arch1* without 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

- Architectural Design Methods
  - Structural Design ("**`port map`**")
  - Data Flow Design (concurrent statements)
  - Behavioral Design ("**`process`**")
- Concurrent vs. Sequential Statements
- Revisit: Signal (**<=**) and Variable (**:=**) Assignments
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`