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: Basic Structure of a VHDL File

A VHDL file

Library Declaration

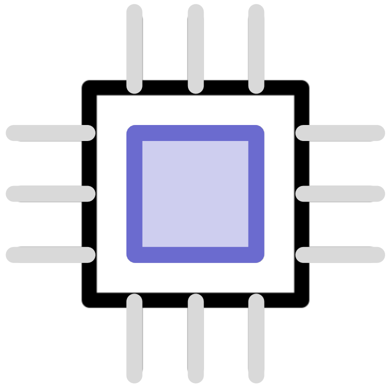
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

Entity Declaration

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

Architecture Body

Define the internal operations of the entity (desired functions)



Outline

- Architectural Design Methods
  - Data Flow Design (concurrent statements)
  - Structural Design (“**`port map`**”)
  - Behavioral Design (“**`process`**”)

- Concurrent vs. Sequential Statements

- 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 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 3.1

Student ID: ____________ Date:
Name: ____________________ ________

- Draw the schematic circuit of this code:

```vhdl
library IEEE; --Vivado 14.4
use IEEE.STD_LOGIC_1164.ALL;
entity abc is
  port (a,b,c: in std_logic;
            y: out std_logic);
end abc;
architecture abc_arch of abc is
signal x : std_logic;
begin
  x <= a nor b;
  y <= x and c;
end abc_arch;
```

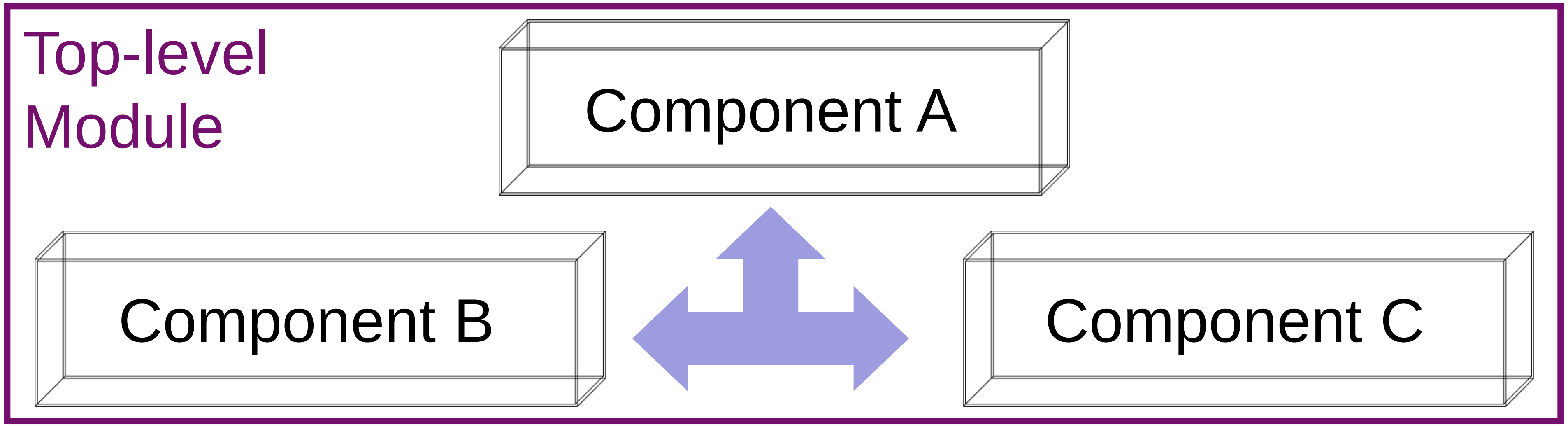
Answer:

Outline

- Architectural Design Methods
  - Data Flow Design (concurrent statements)
  - Structural Design (“**`port map`**”)
  - Behavioral Design (“**`process`**”)
- Concurrent vs. Sequential Statements
- 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.

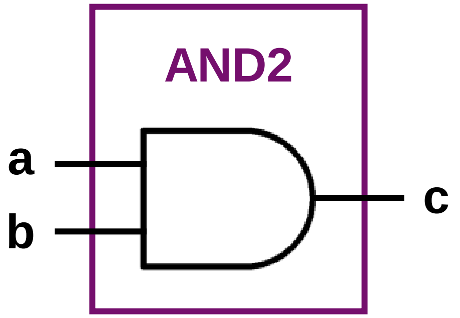


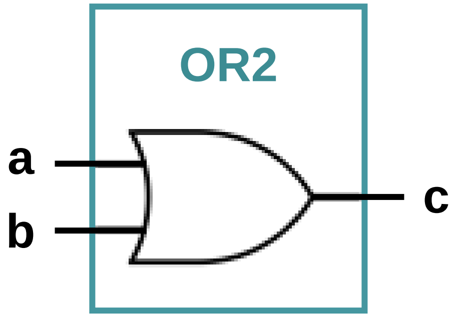
Connected by `port map` in the architecture body of the top-level design module

- **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

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and2 is
port (a,b: in STD_LOGIC;
        c: out STD_LOGIC );
end and2;
architecture and2_arch of and2 is
begin
  c <= a and b;
end and2_arch;
---------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity or2 is
port (a,b: in STD_LOGIC;
        c: out STD_LOGIC );
end or2;
architecture or2_arch of or2 is
begin
  c <= a or b;
end or2_arch;
```



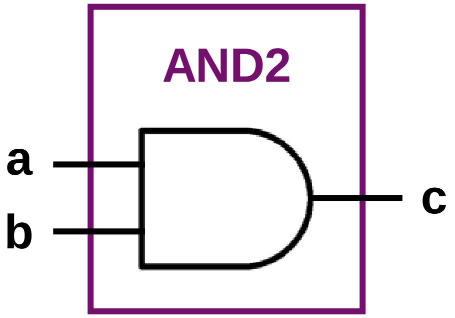


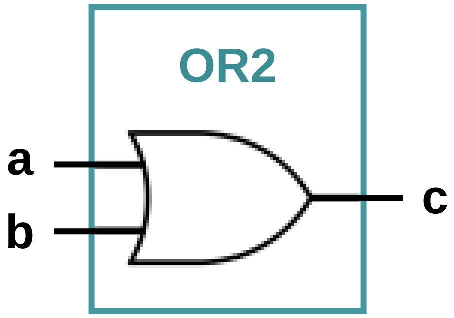
Step 2: Create Components

```
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)   --
```



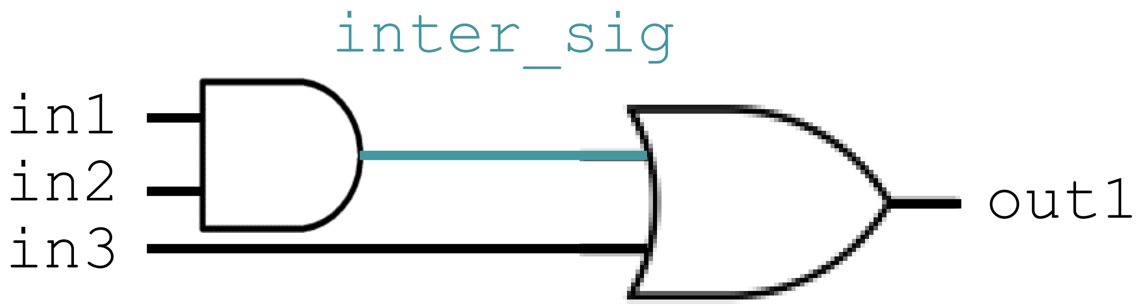


Step 3: Connect Components

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

Step 1

Step 1

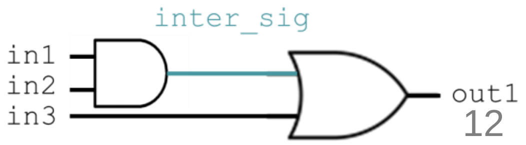
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and2 is
port (a,b: in STD_LOGIC;
        c: out STD_LOGIC );
end and2;
architecture and2_arch of and2 is
begin
  c <= a and b;
end and2_arch;
---------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity or2 is
port (a,b: in STD_LOGIC;
        c: out STD_LOGIC );
end or2;
architecture or2_arch of or2 is
begin
  c <= a or b;
end or2_arch;
```

Top-level Module

Step 2

Step 3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------
entity test is
port ( in1: in STD_LOGIC; in2: in STD_LOGIC;
       in3: in STD_LOGIC;
      out1: out STD_LOGIC );
end test;
architecture test_arch of test is
component and2 --create component
  port (a,b: in std_logic; c: out std_logic);
end component ;
component or2 --create component
  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: or2 port map (inter_sig, in3, out1);
end test_arch;
```



Class Exercise 3.2

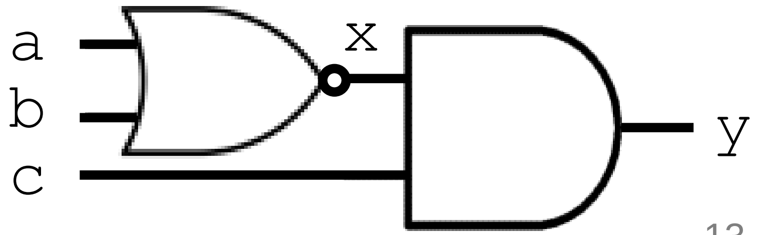
Student ID: ____________ Date:
Name: ________________ ________

- 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: ______________
```

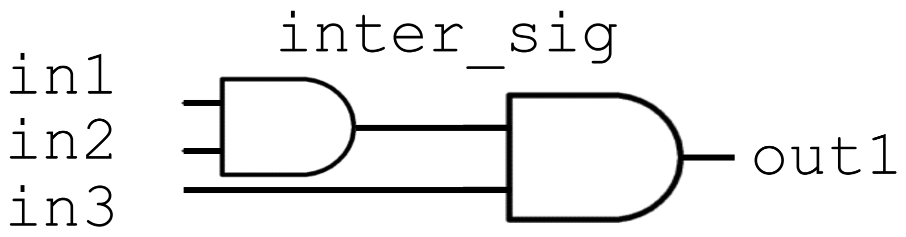


Another Running Example

```
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 3.3

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;
```

x
y
sum
carry

input		output	
x	**y**	**carry**	**sum**

Structural vs. Data Flow

Structural

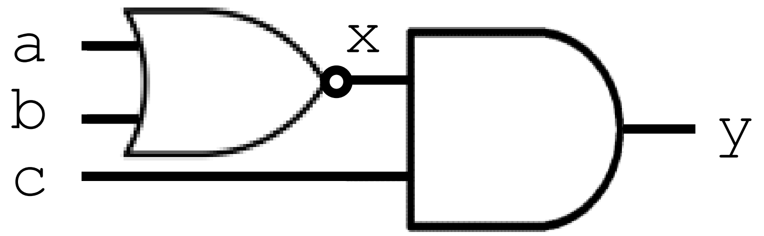
(port map)

```
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;
```

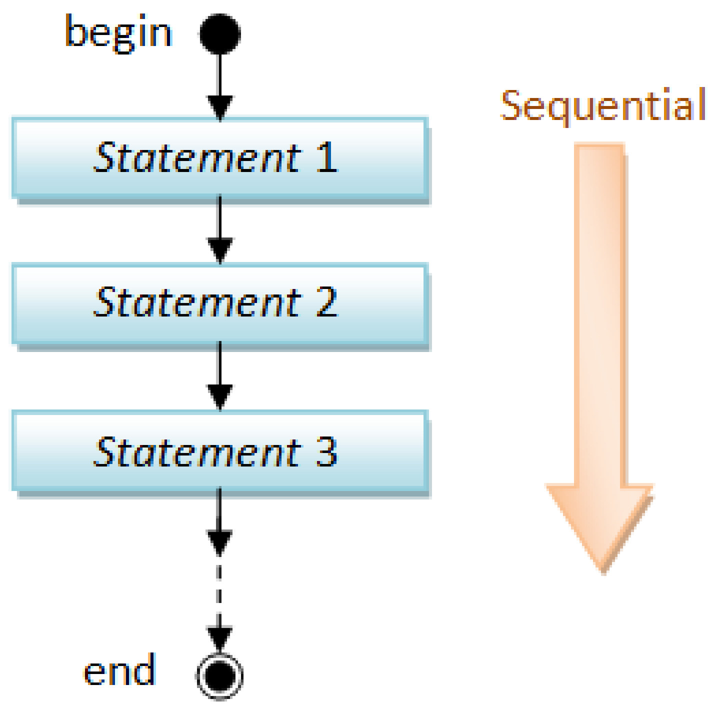


Outline

- Architectural Design Methods
  - Data Flow Design (concurrent statements)
  - Structural Design (“`port map`”)
  - Behavioral Design (“`process`”)
- Concurrent vs. Sequential Statements
- 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 (a, b: in std_logic_vector(3 downto 0);
    equals: out std_logic);
end eqcomp4;
architecture behavioral of eqcomp4 is
begin
  process(a, b)  ← Behavioral Design: Sequential in a "process"
  begin
    if a = b then
        equals <= '1';
    else
        equals <= '0';
    end if;
  end process;
end behavioral;
```

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 kinds of **time-delay** or **wait** (see Lec04).

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;
```

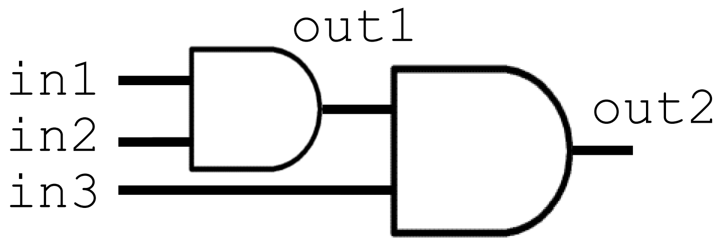
Outline

- Architectual Design Methods
  - Data Flow Design (concurrent statements)
  - Structural Design ("`port map`")
  - Behavioral Design ("`process`")
- Concurrent vs. Sequential Statements
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`

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 statement changes.
  ③
- **Sequential Statement**
  ① Statements within a **`process`** are executed sequentially, and the result is obtained when the process is complete.
  ② **`process(sensitivity list)`**: When one or more signals in the sensitivity list change 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;
```

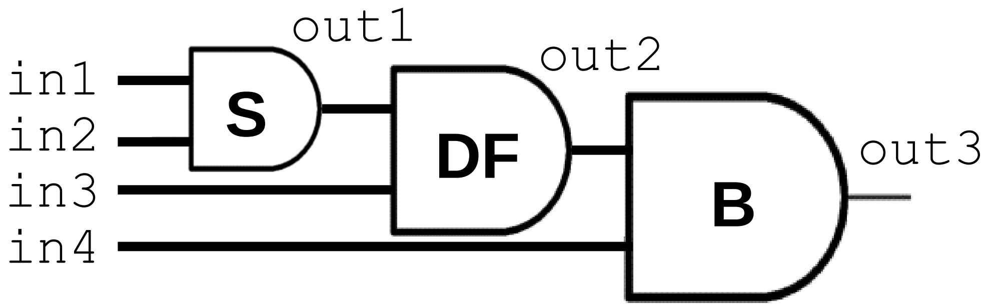
The process (9-12) and line 13 are concurrent and can be interchanged!

Class Exercise 3.4

Student ID: ___________ Date:

Name: _______________ ________

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



Outline

- Architectural Design Methods
  - Data Flow Design (concurrent statements)
  - Structural Design (“`port map`”)
  - Behavioral Design (“`process`”)
- Concurrent vs. Sequential Statements
- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`
  -

Design Constructions

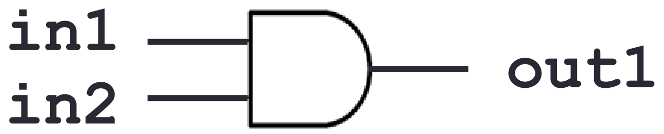
- **Concurrent**: Statements that can be stand-alone
  1) `when-else`
  2) `with-select-when`

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



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity when_ex is
port (in1, in2 : in std_logic;
          out1 : out std_logic);
end when_ex;
architecture when_ex_arch of when_ex is
begin
  out1 <= '1' when in1 = '1' and in2 = '1' else '0';
end when_ex_arch;
```

Condition based

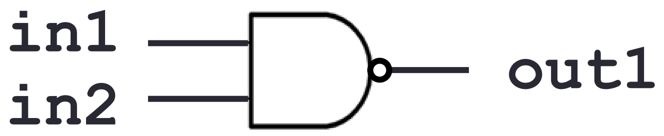
*when **condition** is true then out1 <= ‘1’*
otherwise then out1 <= ‘0’

Class Exercise 3.5

Student ID: ___________ Date:

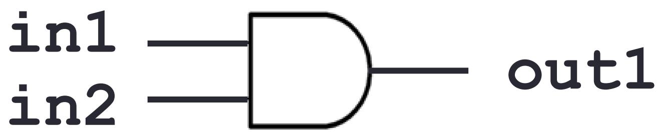
Name: _______________ _______

- 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



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity when_ex is
port (in1, in2 : in std_logic;
              out1 : out std_logic);
end when_ex;
architecture when_ex_arch of when_ex is
begin
  with in1 select
      out1 <= in2 when '1',
                  '0' when others;
end when_ex_arch;
```

Signal based

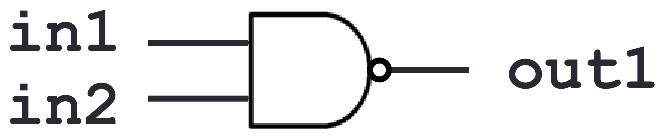
← *when **in1**='1' then out1 <= in2*

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

Class Exercise 3.6

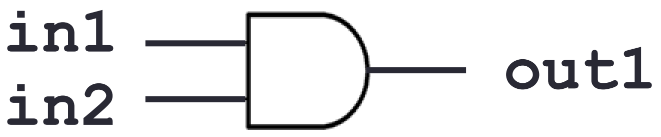
Student ID: ____________ Date:
Name: __________________ ________

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

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

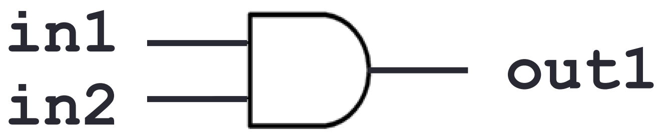
← *when in1='1' then out1 <= in2*

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

Outline

- Architectural Design Methods
  - Data Flow Design (concurrent statements)
  - Structural Design (“`port map`”)
  - Behavioral Design (“`process`”)
- Concurrent vs. Sequential Statements
- 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 3.7

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

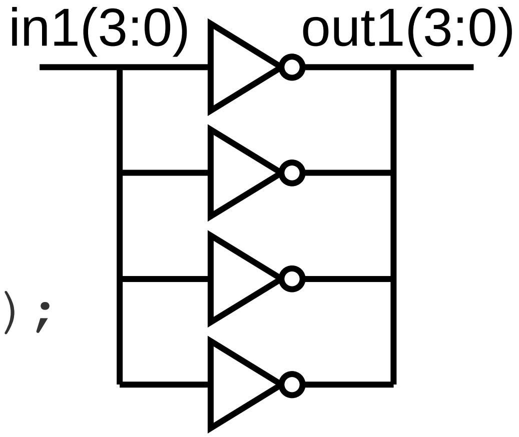
```
if    a = "00" then b <= "1000"
elsif a = "01" then b <= "1000"
elsif a = "10" then b <= "1000"
else                b <= "1000"
end if;
```

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;

end for_ex_arch;
```

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;
```

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 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

Recall: 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 (less direct relationship to the synthesized hardware)
     - E.g., **`variable`** `flag: BOOLEAN := TRUE;`
- Data objects must be declared before being used.

Class Exercise 3.8

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
  - Data Flow Design (concurrent statements)
  - Structural Design ("**`port map`**")
  - Behavioral Design ("**`process`**")

- Concurrent vs. Sequential Statements

- Design Constructions
  - Concurrent: `when-else`, `with-select-when`
  - Sequential: `if-then-else`, `case-when`, `loop`