EGEC281 – FALL 2024

LAB 2
COMPARATOR AND 7 SEGMENT DISPLAY

Designing with VHDL
Experiment 2A and 2B

Rakesh Mahto, Ph.D.
Office: E 314, California State University, Fullerton
Office Hour: Monday and Wednesday 2:00 - 3:30 pm
Or by appointment
Zoom Meeting ID: 894 4126 5483
Email: ramahto@fullerton.edu
Phone No: 657-278-7274

LEARNING OBJECTIVES

- Implement Comparator.
  - **Project 2A - Display output on LEDs.**
- Declaring Bus (vectors).
- Declaring Internal Signals ().
- Signal Assignments in VHDL (with/select, when/else, case).
- Processes (set of sequential statements).
- Implement Comparator.
  - **Project 2B - Display output on 7 Segment Display.**

PROJECT 2A

- 2 Inputs (Switches), 3 Outputs (LEDs)
- If both inputs are equal, first LED should be on.
- If first input is less than second input, second LED should be on.
- If first input is greater than second input, third LED should be on.

GETTING FAMILIAR WITH VHDL SYNTAX (TEST FOR EQUALITY – 2 INPUTS)

Inputs			Output
	A	B	F_E
1	0	0	1
2	0	1	0
3	1	0	0
4	1	1	1

F_E = true, B = A

The VHDL syntax is looking only for those conditions that result in a truth table entry of ‘1’ or true for the function.

Row #1 Row #4

F_E <= (not(A) and not(B)) or (A and B)

Assignment or logic statement
‘<=‘ NOT ‘=‘ by itself

Remember:

The complement of a false (‘0’) is true (‘1’)

GETTING FAMILIAR WITH VHDL SYNTAX (TEST FOR GREATER THAN – 2 INPUTS)

Inputs			Output
	A	B	F_G
1	0	0	0
2	0	1	1
3	1	0	0
4	1	1	0

F_G = true, when B > A

Row #2

F_G <= not(A) and B

Remember:

The complement of a false (‘0’) is true (‘1’)

GETTING FAMILIAR WITH VHDL SYNTAX (TEST FOR LESS THAN – 2 INPUTS)

Inputs			Output
	A	B	F_L
1	0	0	0
2	0	1	0
3	1	0	1
4	1	1	0

F_L = true, B < A

Row #3

F_L <= A and not(B)

Remember:

The complement of a false ('0') is true ('1')

LAB 2 PROJECT A - COMPARE

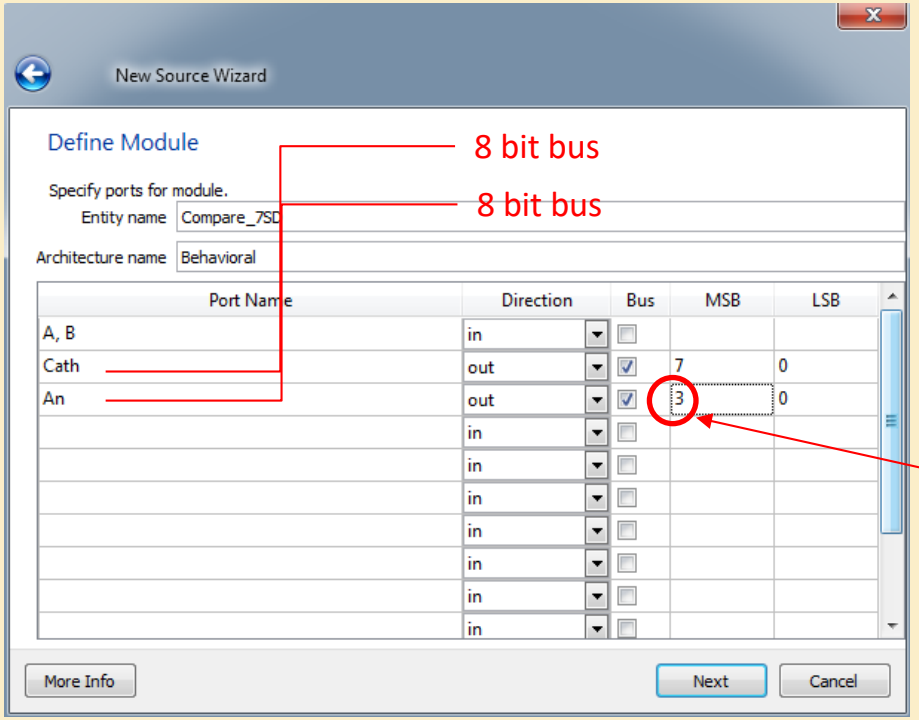
1. Create a new project. Name it **Compare**.
   - Write the VHDL that compares two input values to see if they are the same value.
   - Input signals A and B
   - Output signal F_E, F_L, F_G
2. Simulate Compare. Make stimulus inputs count from 00 – 11, waiting for 100 ns between each count.
3. Verify that your truth table corresponds with the simulation output.
4. Create a XDC for Compare; Assign:
   - A to SW1 ⇨ (J15)
   - B to SW0 ⇨ (L16)
   - F_E to LD0 ⇨ (H17)
   - F_G to LD1 ⇨ (K15)
   - F_L to LD2 ⇨ (J13)
5. Generate and download program file (.bit) to FPGA
6. Verify that the hardware works as expected based on your truth table from #3, #4, and #5.

PROJECT 2B

- 2 Inputs (Switches), 8 bit Cathode and 8 bit Anode Outputs.
- If both inputs are equal, display ‘E’ on 7 segment display.
- If first input is less than second input, display ‘L’ on 7 segment display.
- If first input is greater than second input, display ‘G’ on 7 segment display.

DECLARING BUSES

Buses are vectors or arrays of bits.

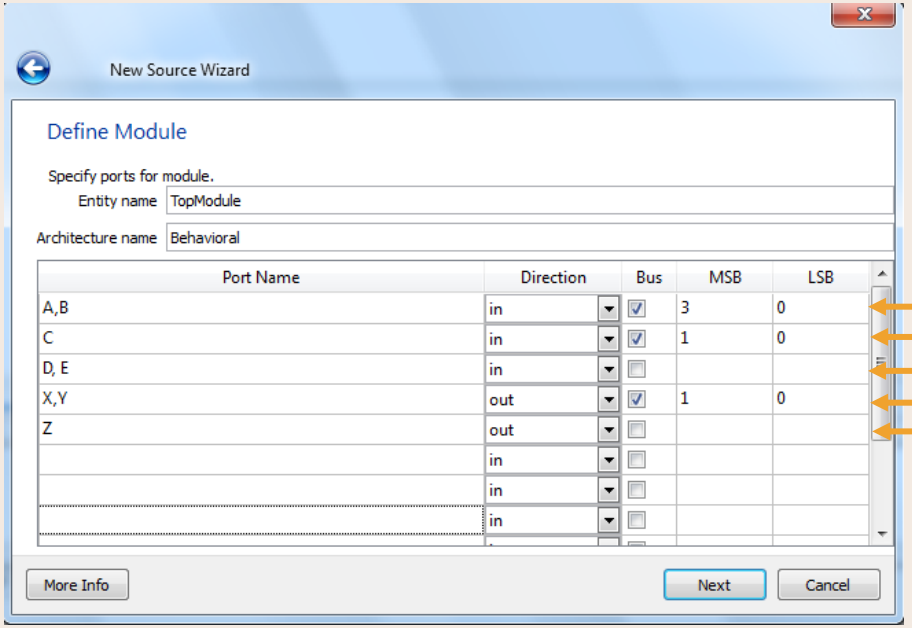



```
entity Compare_7SD is
   Port ( A, B : in  STD_LOGIC;
          Cath : out  STD_LOGIC_VECTOR (7 downto 0);
          An : out  STD_LOGIC_VECTOR (7 downto 0));
end Compare_7SD;

architecture Behavioral of Compare_7SD is

begin


end Behavioral;
```

Replace 3 with 7. Since it is an 8-bit bus

DECLARING ARRAYS FOR INPUTS/OUTPUTS

REVIEW Slide



A and B are 4-bit wide INPUT buses

C is 2-bit wide INPUT buses

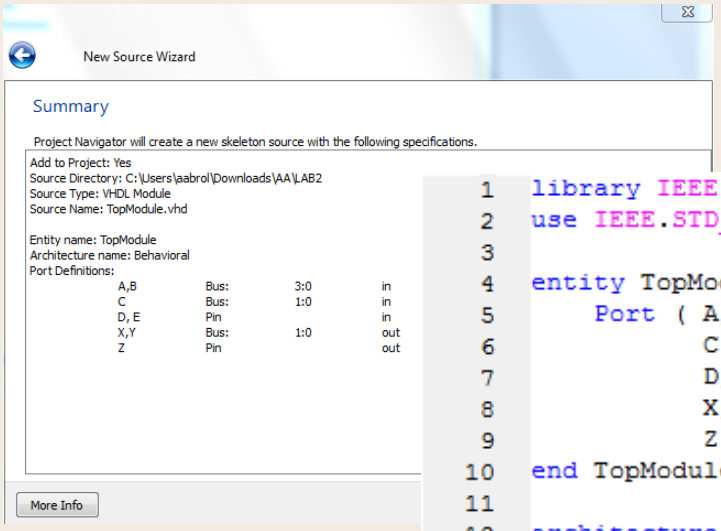
D and E are 1 bit Input Pins

X and Y are 2-bit wide OUTPUT buses

Z is a 1 bit OUTPUT Pin

DECLARING ARRAYS FOR INPUTS/OUTPUTS

REVIEW Slide




```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity TopModule is
5      Port ( A,B : in  STD_LOGIC_VECTOR (3 downto 0);
6             C : in  STD_LOGIC_VECTOR (1 downto 0);
7             D, E : in  STD_LOGIC;
8             X,Y : out  STD_LOGIC_VECTOR (1 downto 0);
9             Z : out  STD_LOGIC);
10 end TopModule;
11
12 architecture Behavioral of TopModule is
13
14 begin
15
16
17 end Behavioral;
```

DECLARING INTERNAL SIGNALS REVIEW Slide

Internal signals are neither inputs nor outputs, but only the internal links.
They are the bonding wires or substrate traces between each gate in the design.

```
entity Compare_7SD is
   Port ( A, B : in  STD_LOGIC;
          Cath : out  STD_LOGIC_VECTOR (7 downto 0);
          An : out  STD_LOGIC_VECTOR (3 downto 0));
end Compare_7SD;

architecture Behavioral of Compare_7SD is

signal F_E, F_L, F_G: STD_LOGIC;

begin

                F_E <= (not (A) and not (B)) or (A and B) ;
                F_L <= not (A) and B;
                F_G <= A and not (B) ;

end Behavioral;
```

The architecture contains three signals F_E, F_L and F_G, used internally within the architecture. A signal is declared before the 'begin', of an architecture, and has its own data type (e.g. STD_LOGIC). Technically, *ports* are *signals*, so signals and ports are read and assigned in the same way.

DECLARING INTERNAL SIGNALS

REVIEW Slide

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity TopModule is
5       Port ( A,B : in  STD_LOGIC_VECTOR (3 downto 0);
6              C : in  STD_LOGIC_VECTOR (1 downto 0);
7              D, E : in  STD_LOGIC;
8              X,Y : out  STD_LOGIC_VECTOR (1 downto 0);
9              Z : out  STD_LOGIC);
10  end TopModule;
11
12  architecture Behavioral of TopModule is
13
14  signal int1, int2: STD_LOGIC;
15  signal temp: STD_LOGIC_VECTOR(3 downto 0);
16
17  begin
18
19
20  end Behavioral;
```

int1 and int2 are 1 bit internal

Temp is a 4-bit wide internal signal bus

VHDL: SIGNAL ASSIGNMENTS

REVIEW Slide

(Ways of assigning different values to a signal, based on value of another signal)

"With/Select"

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

"When/Else"

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

"Case"

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

CONDITIONAL STRUCTURE (IF STATEMENT)

REVIEW Slide

```
[label: ] if  condition1  then
              sequence-of-statements
          elsif  condition2  then
              \_ optional sequence-of-statements  /
              elsif  condition3  then
              \_ optional sequence-of-statements  /

              ....
              else
              \_ optional sequence-of-statements  /
              end if [ label ] ;
```

```
if a=b then
    c <= '1';
elsif b<c then
    d <= '1';
    b <= '1';
else
    a <= '1';
end if;
```

PROCESSES: CONCURRENT VS. SEQUENTIAL EXECUTION

REVIEW Slide

- Sequential Execution
  - Most programming languages like C, C++, and C# use this approach where the statements are executed in the order written.
- Concurrent Execution
  - Because an FPGA is a large array of nand gates it is possible to execute all the statements at the same time.
  - This is possible if there is no time dependence between the signals being processed. But some hardware structures have propagation delays like the D-latch.

PROCESS REVIEW Slide

- Processes are used to control the execution
  - Contains a set of sequential statements to be executed
  - The whole process is a concurrent statement
  - Can be interpreted as a circuit part enclosed inside of a black box
- Two types
  - Process with sensitivity list.
    - This list states - exactly - which signals cause the process statement to be executed.
    - Only **changes** in these signals cause the process statement to be executed.
  - Process with wait statement (sensitivity list is a wait statement only)

A PROCESS WITH A SENSITIVITY LIST

REVIEW Slide

- Syntax:

```
process_name : process (sensitivity_list)
   declarations;
begin
   sequential statement;
   sequential statement;
  . . .
end process;
```

PROCESS WITH SENSITIVITY LIST REVIEW Slide

Interpretation: “black box, indivisible circuit part”.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   --Declaration of the module's inputs and outputs
5   ENTITY example IS PORT (
6
7   A: IN std_logic;
8   B: IN std_logic;
9   C: OUT std_logic
10
11  );
12  END example;
13
14  --Defining the modules behavior
15  ARCHITECTURE behavioral OF example IS
16
17  BEGIN
18
19  p1: PROCESS (A, B)
20  BEGIN
21     C <= A OR B;
22  END PROCESS;
23  END behavioral;
```

Sensitivity list

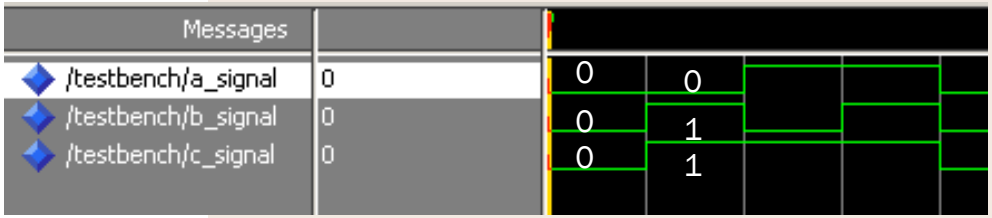
Note:

The execution of the process is initiated whenever an event occurs on any of the signals in the sensitivity list

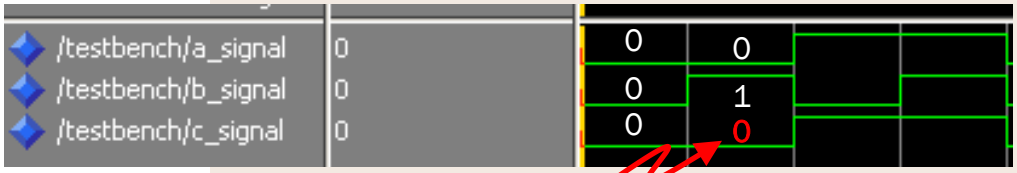
For practical purposes, you can regard a process as a “big” concurrent signal assignment statement

WAVEFORM FOR EXAMPLE 1 REVIEW Slide

```
14  --Defining the modules behavior
15  ARCHITECTURE behavioral OF example IS
16
17  BEGIN
18
19  p1: PROCESS (A, B)
20  BEGIN
21     C <= A OR B;
22  END PROCESS;
23  END behavioral;
24
```




```
14  --Defining the modules behavior
15  ARCHITECTURE behavioral OF example IS
16
17  BEGIN
18
19  p1: PROCESS (A)
20  BEGIN
21     C <= A OR B;
22  END PROCESS;
23  END behavioral;
```



Process not activated on B change

A PROCESS WITH WAIT STATEMENT

REVIEW Slide

- Process has no sensitivity list
- Process continues the execution until a wait statement is reached and then suspended
- Forms of wait statement:
  - wait on signals;
  - wait until boolean_expression;
  - wait for time_expression;

```
ENTITY example IS PORT (

A: IN std_logic;
B: IN std_logic;
SELECTOR: IN std_logic;
C: OUT std_logic

);
END example;

ARCHITECTURE behavioral OF example IS
SIGNAL S1: STD_LOGIC;
BEGIN
p1: PROCESS
BEGIN
   C <= A or B;
   WAIT ON A, B;
END PROCESS;
END behavioral;
```

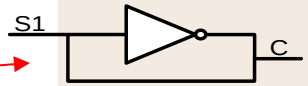
SEQUENTIAL SIGNAL ASSIGNMENT STATEMENT

REVIEW Slide

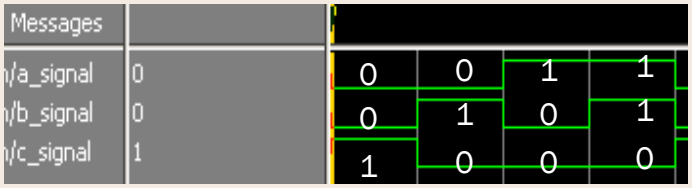
Syntax: Signal_name <= value_expression;

```
5   ENTITY example IS PORT (
6
7   A: IN std_logic;
8   B: IN std_logic;
9   C: OUT std_logic
10
11  );
12  END example;
13
14  --Defining the modules behavior
15  ARCHITECTURE behavioral OF example IS
16
17  SIGNAL S1: STD_LOGIC;
18  BEGIN
19
20  p1: PROCESS (A, B)
21  BEGIN
22     S1 <= NOT(A OR B);
23  END PROCESS;
24
25  C <= S1;
26
27
28  END behavioral;
29
```

```
5   ENTITY example IS PORT (
6
7   A: IN std_logic;
8   B: IN std_logic;
9   C: OUT std_logic
10
11  );
12  END example;
13
14  --Defining the modules behavior
15  ARCHITECTURE behavioral OF example IS
16
17  SIGNAL S1: STD_LOGIC;
18  BEGIN
19
20  p1: PROCESS (A, B)
21  BEGIN
22    S1 <= A OR B;
23    S1 <= NOT(S1);
24
25  END PROCESS;
26
27  C <= S1;
28
29
30  END behavioral;
```



undefined



VARIABLE ASSIGNMENT STATEMENT

REVIEW Slide

Syntax: Variable_name := value_expression;

Used inside processes. The assignment takes effect "immediately".

```
--Declaration of the module's inputs and outputs
ENTITY example IS PORT (

A: IN std_logic;
B: IN std_logic;
C: OUT std_logic

);
END example;

ARCHITECTURE behavioral OF example IS

BEGIN
p1: PROCESS (A, B)
variable tmp: std_logic;
BEGIN
   tmp:= '0';
   tmp:= tmp OR A;
   tmp:= tmp OR B;
   tmp:= NOT(tmp);
   C <= tmp;
END PROCESS;
END behavioral;
```

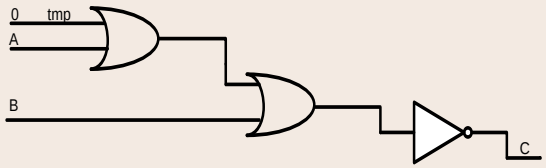
Conceptual implementation

Note:

Easy to understand, but not clear hardware mapping!

You can always use signals.

Rely on variables only for the characteristics that cannot be described by signals.

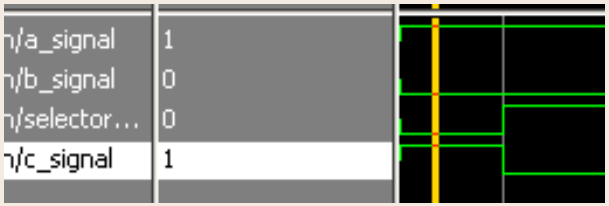


CASE STATEMENT

REVIEW Slide

Syntax:

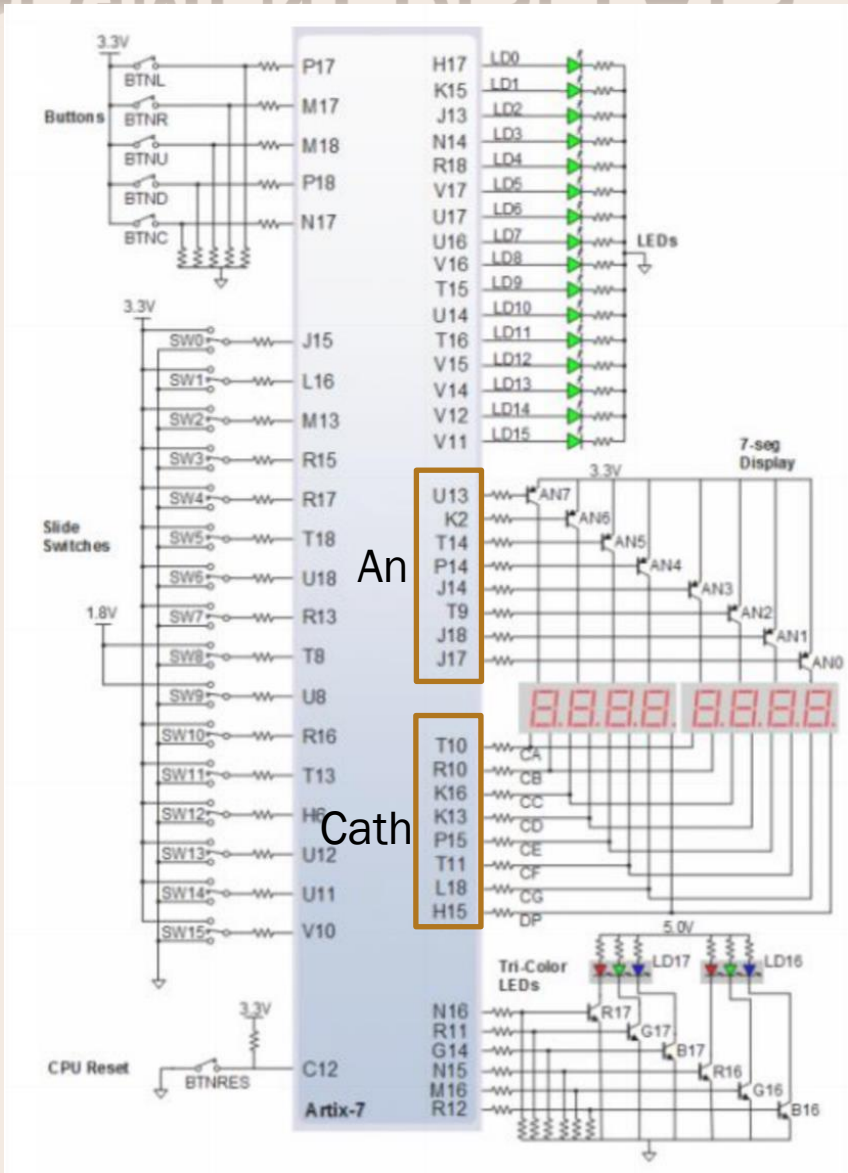
```
case case_expression is
  when choice_1 =>
    sequential statements;
  when choice_2 =>
    sequential statements;
  . . .
  when choice_n =>
    sequential statements;
end case;
```



Example:

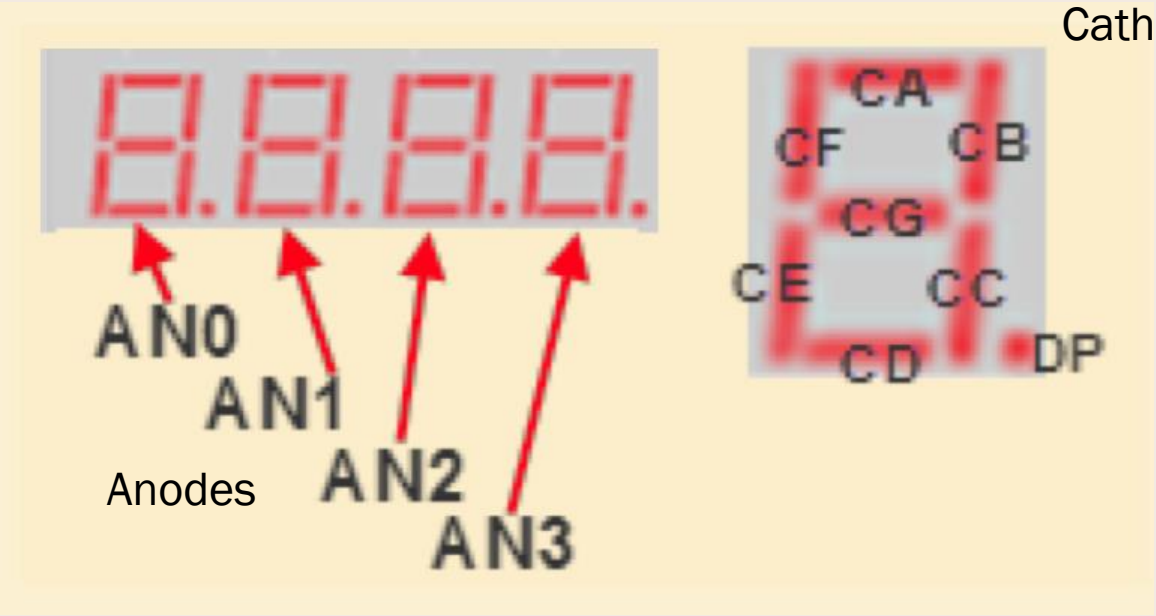
```
--Declaration of the module's inputs and outputs
ENTITY example IS PORT (

A: IN std_logic;
B: IN std_logic;
SELECTOR: IN std_logic;
C: OUT std_logic

);
END example;

ARCHITECTURE behavioral OF example IS
SIGNAL S1: STD_LOGIC;
BEGIN
p1: PROCESS (A, B, SELECTOR)
BEGIN
   CASE SELECTOR IS
      WHEN '0' => C <= A;
      WHEN others  => C <= B;
   END CASE;
END PROCESS;
END behavioral;
```

FOUR 7-SEGMENT DISPLAYS



FOUR 7-SEGMENT DISPLAYS

Anodes and Cathodes…



ANODES AND CATHODES

- “Anodes” (7-segment display) are “Active Low”.
  - A ‘0 (Low)’ lights the Anode on.
  - A ‘1 (High)’ turns the Anode off.

- “Cathodes” (common-cathode) are “Active High”.
  - A ‘1 (High)’ lights the Segment on.
  - A ‘0 (Low)’ turns the Segment off.

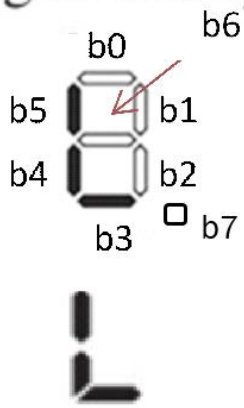
LETTER DISPLAY: SAMPLE CATHODE VALUES

REVIEW Slide

Letter display
(active low outputs)

Input	Display	B7	B6	B5	B4	B3	B2	B1	B0
0	L	0	0	1	1	1	0	0	0
1	H	0	1	1	1	0	1	1	0

7-segment display



WHEN, WHAT AND WHERE TO DISPLAY..

- When A = B, the 7-segment display should display E.
- When A > B, the 7-segment display should display G (Go for a 6).
- When A < B, the 7-segment display should display L.
- For all cases, display on one of the anodes (Set other anodes off). Remember, the anodes are active low.

LAB 2 PROJECT B COMPARE_7SD

1. Create a new project. Name it Compare_7SD.
   - Write the VHDL that compares two input values to see if they are the same value.
   - Input signals A and B
   - Output signals Cath (8 bits), An (4 bits).
2. Simulate Compare_7SD. Make stimulus inputs count from 00 – 11, waiting 100 ns between each count.
3. Verify the simulation output.
4. Create a XDC for Compare_7SD; Assign:
   - Input signals A and B: 2 switches
     Input A ⇨ J15
     Input B ⇨ L16
   - Output signals Cath (8 bits) to cathode ports,
   - Output An (4 bits) to anode ports.
5. Generate and download program file (.bit) to FPGA
6. Verify that the hardware works as expected.

An	Loc	Cath	Loc
An(0)	J17	Cath(0)	T10
An(1)	J18	Cath(1)	R10
An(2)	T9	Cath(2)	K16
An(3)	J14	Cath(3)	K13
		Cath(4)	P15
		Cath(5)	T11
		Cath(6)	L18
		Cath(7)	H15

LAB REPORT

- Cover page:
  - Course Title
  - Lab Number, Letter
  - Team Names
- Project 2A:
  - VHDL
  - Black Box
  - Truth Table
  - Simulation Code, Waveform
  - XDC
  - Picture of Implementation
- Project 2B:
  - VHDL
  - Truth Table
  - Simulation Code, Waveform
  - XDC
  - Picture of Implementation
- Summary Paragraph
  - Work completed
  - Any problems
  - Helpful Hints
  - Suggested Improvements