Class 03 Combinational Logic Functions

Selected Signal Assignment Statement vs. Conditional Signal Assignment Statement

Usually a port or a signal

```
WITH __expression SELECT
    __signal <= __expression WHEN __constant_value,
                __expression WHEN __constant_value,
                __expression WHEN others;
```

comma

Default case

Selected Signal Assignment Statement

```
__signal <= __expression WHEN __boolean_expression ELSE
            __expression WHEN __boolean_expression ELSE
            __expression;
```

Default case

Conditional Signal Assignment Statement

Can't be used in **PROCESS** statement

2-Line-to-4-Line Decoder with an Enable Input

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY decode IS
  PORT(
    d: IN STD_LOGIC_VECTOR (1 downto 0);
    g: IN STD_LOGIC;
    y: OUT STD_LOGIC_VECTOR (3 downto 0));
END decode;

ARCHITECTURE decoder OF decode IS
  SIGNAL inputs : STD_LOGIC_VECTOR (2 downto 0);
BEGIN
  inputs(2) <= g;
  inputs (1 downto 0) <= d;
  WITH inputs SELECT
    y <=  "0001" WHEN "000",
          "0010" WHEN "001",
          "0100" WHEN "010",
          "1000" WHEN "011",
          "0000" WHEN others;
END decoder;
```

Concatenate g (1 bit) and d (2 bits) to get a 3-bit vector

d(0)

d(1)

g: enable

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY decode IS
  PORT(
    d: IN INTEGER Range 0 to 3;
    g: IN STD_LOGIC;
    y: OUT STD_LOGIC_VECTOR (0 to 3));
END decode;

ARCHITECTURE decoder OF decode IS
BEGIN
 y <=  "1000" WHEN (d=0 and g='0') ELSE
       "0100" WHEN (d=1 and g='0') ELSE
       "0010" WHEN (d=2 and g='0') ELSE
       "0001" WHEN (d=3 and g='0') ELSE
       "0000";
END decoder ;
```

IF Statement

Boolean value

```
IF __boolean_expression THEN
    __statement;
    __statement;
ELSIF  __boolean_expression THEN
    __statement;
    __statement;
ELSE
    __statement;
    __statement;
END IF;
```

IF Statement

```
IF(nRBI = '0' and input = "0000") THEN
  output <= "1111111"; -- 0 suppressed
  nRBO <= '0'; -- Next 0 suppressed
ELSE
  nRBO <= '1'; -- Next 0 displayed
END IF;
```

Can be used in **PROCESS** statement

CASE Statement

Usually a port or signal

```
CASE __expression IS
   WHEN __constant_value =>
      __statement;
      __statement;
   WHEN __constant_value =>
      __statement;
      __statement;
   WHEN others =>
      __statement;
      __statement;
END CASE;
```

CASE Statement

Can be used in **PROCESS** statement

7-segment display

```
CASE input IS
  WHEN "0000" =>
    output <=  "00000011";        -- 0 displayed
  WHEN "0001"      =>
    output <=      "10011111";-- 1
  WHEN "0010"      =>
    output <=      "00100101";          -- 2
  WHEN "0011"      =>
    output <=      "00001101";          -- 3
  WHEN "0100"      =>
    output <=      "10011001";          -- 4
  WHEN "0101"      =>
    output <=      "01001001";          -- 5
  WHEN "0110"      =>
    output <=      "11000001";          -- 6
  WHEN "0111"      =>
    output <=      "00011111";-- 7
  WHEN "1000"      =>
    output <=      "00000001";          -- 8
  WHEN "1001"      =>
    output <=      "00011001";          -- 9
  WHEN others =>
    output <=      "11111111"; -- blank
END CASE;
```

PROCESS

- VHDL syntax requires an IF statement or a CASE statement to be contained within a PROCESS.
- IF statement and CASE statement can only be used in PROCESS statement
- A PROCESS is a construct containing statements that are executed if a signal in the sensitivity list of the PROCESS changes.
- A PROCESS is itself a *concurrent* statement, but the statements inside the PROCESS are *sequential*.

optional

```
[label:] PROCESS (sensitivity list)
BEGIN
   statements;
END PROCESS;
```

PROCESS

```
PROCESS (nRBI, input)
BEGIN
   IF(nRBI = '0' and input = "0000") THEN
      output <= "1111111"; -- 0 suppressed
      nRBO <= '0'; -- Next 0 suppressed
   ELSE
      nRBO <= '1'; -- Next 0 displayed
   END IF;
END PROCESS;
```

Possible Design Errors in PROCESS (Cont.)

- Only one instance of the EVENT express (e.g., clk'EVENT and clk='1') is allowed in a PROCESS statement.

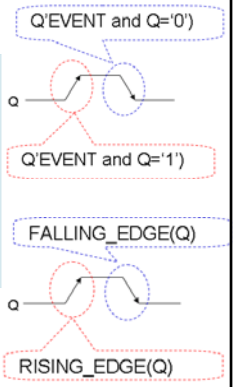
```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (load='1') THEN
         q <= p;
      END IF;
   END IF;
   IF (clk'EVENT and clk='1') THEN
      IF (count_enable='1') THEN
         q <= q+1;
      END IF;
   END IF;
END PROCESS;
```

Illegal syntax: more than one EVENT per process

$\Rightarrow$

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (load='1') THEN
         q <= p;
      ELSIF (count_enable='1') THEN
         q <= q+1;
      END IF;
   END IF;
END PROCESS;
```

Legal syntax



Possible Design Errors in PROCESS (Cont.)

- No other port, signal, or variable is allowed to be included with the expression that evaluates the clock.

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1' and load='1') THEN
      q <= p;
   ELSE
      q <= q+1;
   END IF;
END PROCESS;
```

Illegal syntax: load evaluated in same statement as clk

$\Rightarrow$

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (load='1') THEN
         q <= p;
      ELSE
         q <= q+1;
      END IF;
   END IF;
END PROCESS;
```

Legal syntax

Possible Design Errors in PROCESS (Cont.)

- The statements in a process should be exclusive such that it is only possible to assign one value to a port, variable, or signal for each time the process executes.

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (count_enable = '1') THEN
         q <= q+1;
      END IF;
      IF (load = '1') THEN
         q<= p;
      END IF;
      IF (clear = '0') THEN
         q <= (others => '0');
      END IF;
   END IF;
END PROCESS;
```

Ambiguous (but not illegal) syntax: q assigned more than once in a process. May have an unexpected result.

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (count_enable = '1') THEN
         q <= q+1;
      ELSIF (load = '1') THEN
         q<= p;
      ELSIF (clear = '0') THEN
         q <= (others => '0');
      END IF;
   END IF;
END PROCESS;
```

Legal syntax

Possible Design Errors in PROCESS (Cont.)

- The (others => '0') statement assigns '0's to a vector and doesn't have to care the width of the vector when using this statement.

```
ARCHITECTURE test_others OF XXX IS
  SIGNAL cnt: STD_LOGIC_VECTOR (3 downto 0)
BEGIN
   cnt <= (others => '0');
END test_others
```

```
ARCHITECTURE test_others OF XXX IS
  SIGNAL cnt: STD_LOGIC_VECTOR (3 downto 0)
BEGIN
  cnt(3) <= '0';
  cnt(2) <= '0';
  cnt(1) <= '0';
  cnt(0) <= '0';
END test_others
```

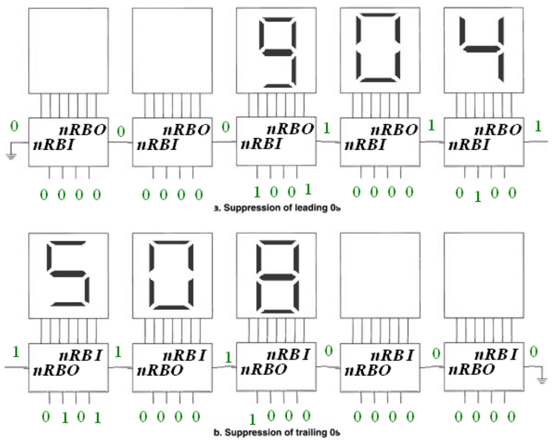
```
ARCHITECTURE test_others OF XXX IS
  SIGNAL cnt: STD_LOGIC_VECTOR (3 downto 0)
BEGIN
  cnt <= "0000";
END test_others
```

```
ARCHITECTURE test_others OF XXX IS
  SIGNAL cnt: STD_LOGIC_VECTOR (3 downto 0)
BEGIN
   cnt <= (1=>'1', others =>'0');
END test_others
```

```
ARCHITECTURE test_others OF XXX IS
  SIGNAL cnt: STD_LOGIC_VECTOR (3 downto 0)
BEGIN
  cnt(3) <= '0';
  cnt(2) <= '0';
  cnt(1) <= '1';
  cnt(0) <= '0';
END test_others
```

Ripple Blanking

- A technique used in a multiple-digit numerical display that suppresses leading or trailing zeros in the display, but allows internal zeros to be displayed.
- nRBI=0 and D=0
  → 7-segment blank
  → nRBO=0
  otherwise
  → show digit
  → nRBO=1
- Suppress leading zeros
  – Ground nRBI of the MSB digit
- Suppress trailing zeros
  – Groud nRBI of the LSB digit



BCD-to-7Segment with Ripple Blanking

Set up the initial value

```
ENTITY sevsegrb IS
  PORT(
    -- Use separate I/Os, not bus
    d3, d2, d1, d0: IN BIT;
    nRBI: IN BIT :='0'; -- set up the initial value
    a, b, c, d, e, f, g, nRBO: OUT BIT);
END sevsegrb;

ARCHITECTURE seven_segment OF sevsegrb IS
  -- Bit vectors for internal use
  SIGNAL input: BIT_VECTOR (3 DOWNTO 0);
  --   in decoder CASE statement
  SIGNAL output: BIT_VECTOR (6 DOWNTO 0);
BEGIN
-- Concatenate inputs to make bit vector
input <= d3 & d2 & d1 & d0;

--  Process Statement
assign_out: PROCESS (input, nRBI)
BEGIN
  IF(nRBI = '0' and input = "0000") THEN
    output <= "1111111";  -- 0 suppressed
    nRBO <= '0';  -- Next 0 suppressed
  ELSE
    nRBO <= '1';  -- Next 0 displayed
```

```
    CASE input IS
      WHEN "0000" =>
        output <= "0000001"; -- 0
      WHEN "0001" =>
        output <= "1001111"; -- 1
      WHEN "0010" =>
        output <= "0010010"; -- 2
      WHEN "0011" =>
        output <= "0000110"; -- 3
      WHEN "0100" =>
        output <= "1001100"; -- 4
      WHEN "0101" =>
        output <= "0100100"; -- 5
      WHEN "0110" =>
        output <= "1100000"; -- 6
      WHEN "0111" =>
        output <= "0001111"; -- 7
      WHEN "1000" =>
        output <= "0000000"; -- 8
      WHEN "1001" =>
        output <= "0001100"; -- 9
      WHEN others =>
        output <= "1111111"; -- blank
    END CASE;
  END IF;
END PROCESS assign_out;
```

```
-- pin outputs.
  a <= output(6);
  b <= output(5);
  c <= output(4);
  d <= output(3);
  e <= output(2);
  f <= output(1);
  g <= output(0);
END seven_segment;
```

Three 7-Segments with Ripple Blanking

```
ENTITY RippleBlanking IS
        PORT(sw: IN BIT_VECTOR(8 downto 0));
END RippleBlanking;

ARCHITECTURE a of RippleBlanking IS
        SIGNAL nRBI: BIT_VECTOR(2 downto 0) := "000";
BEGIN
        RB_process: PROCESS(sw)
        BEGIN
                -- First digit
                IF(nRBI(2) = '0' and sw(8 downto 6)="000") THEN -- check if leading zero
                        nRBI(1) <= '0'; -- Let Digit 1 blank
                ELSE
                        nRBI(1) <= '1'; -- Light up Digit 1
                END IF;
                -- Second digit
                IF(nRBI(1) = '0' and sw(5 downto 3)="000") THEN -- check if leading zero
                        nRBI(0) <= '0'; -- Let Digit 0 blank
                ELSE
                        nRBI(0) <= '1'; -- Light up Digit 0
                END IF;
                -- Third digit
                IF(nRBI(0) = '0' and sw(2 downto 0)="000") THEN -- check if leading zero
                        -- Let it blank
                ELSE
                        -- Let it blank
                END IF;
        END PROCESS RB_process;
END a;
```

Initial value

Encoders

• 3-bit binary encoder

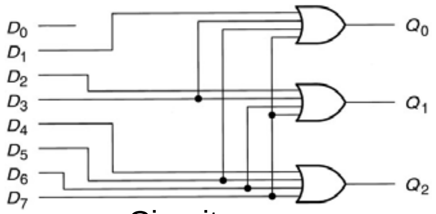
$$Q_2 = D_7 + D_6 + D_5 + D_4$$
$$Q_1 = D_7 + D_6 + D_3 + D_2$$
$$Q_0 = D_7 + D_5 + D_3 + D_1$$

Equation

Encoders might generate wrong codes: E.g., both D5 and D3 are on, the output of Q2Q1Q0 is 111.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	X	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	0	X	0	1	0
0	0	0	0	1	0	0	X	0	1	1
0	0	0	1	0	0	0	X	1	0	0
0	0	1	0	0	0	0	X	1	0	1
0	1	0	0	0	0	0	X	1	1	0
1	0	0	0	0	0	0	X	1	1	1

Truth Table



Circuit

Priority Encoder

- An encoder makes the output corresponding to the highest-priority input.
  - Step 1: A bit goes HIGH if it is part of the code for an active input.
  - Step 2: A bit goes LOW if it is a LOW of an input with a higher priority
- Development steps:

Step 1

$$Q_2 = D_7 + D_6 + D_5 + D_4$$
$$Q_1 = D_7 + D_6 + D_3 + D_2$$
$$Q_0 = D_7 + D_5 + D_3 + D_1$$

$$\Downarrow$$

Step 2

$$Q_2 = D_7 + D_6 + D_5 + D_4$$
$$Q_1 = D_7 + D_6 + \overline{D_5}\,\overline{D_4}D_3 + \overline{D_5}\,\overline{D_4}D_2$$
$$Q_0 = D_7 + \overline{D_6}D_5 + \overline{D_6}\,\overline{D_4}D_3 + \overline{D_6}\,\overline{D_4}\,\overline{D_2}D_1$$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	X	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Truth Table

Priority Encoder (Cont.)

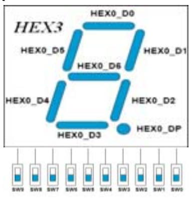
```
-- hi_pri8a.vhd
ENTITY hi_pri8a IS
  PORT(
    d: IN BIT_VECTOR(7 downto 0);
    q: OUT BIT_VECTOR (2 downto 0));
END hi_pri8a;

ARCHITECTURE a OF hi_pri8a IS
BEGIN
  --  Concurrent Signal Assignments
  q(2) <= d(7) or d(6) or d(5) or d(4);

  q(1) <= d(7) or d(6)
         or ((not d(5)) and (not d(4)) and d(3))
         or ((not d(5)) and (not d(4)) and d(2));

  q(0) <= d(7) or ((not d(6)) and d(5))
         or ((not d(6)) and (not d(4)) and d(3))
         or ((not d(6)) and (not d(4)) and (not d(2)) and d(1));
END a;
```

$\Rightarrow$

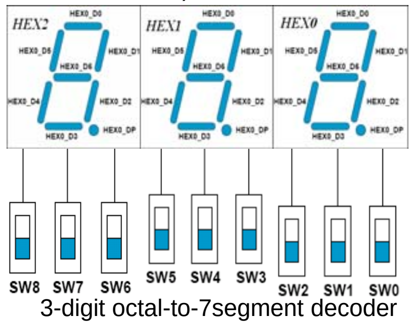
```
-- hi_pri8b.vhd
ENTITY hi_pri8b IS
  PORT(
    d: IN BIT_VECTOR(7 downto 0);
    q: OUT INTEGER RANGE (0 to 7);
END hi_pri8b;

ARCHITECTURE a OF hi_pri8b IS
BEGIN
--  Conditional Signal Assignment
q <= 7 WHEN d(7)='1' ELSE
       6 WHEN d(6)='1' ELSE
       5 WHEN d(5)='1' ELSE
       4 WHEN d(4)='1' ELSE
       3 WHEN d(3)='1' ELSE
       2 WHEN d(2)='1' ELSE
       1 WHEN d(1)='1' ELSE
       0;
END a;
```

Lab 03

- Part 1: Design a BCD priority Encoder. (7-segment shows 0~9) (Hint: use **WHEN … ELSE**)
  - The 7-segment shows the number corresponding to the switch that is ON and has the highest priority, where a switch with the larger numeric value has higher priority.
  - If all of the switches are OFF, turn off the 7-segment LED.
- Part 2: Design a 3-digit octal-to-7segment decoder with the leading zeros suppressed, as follow: (Hint: Use **PROCESS**, **IF**, and **CASE**)

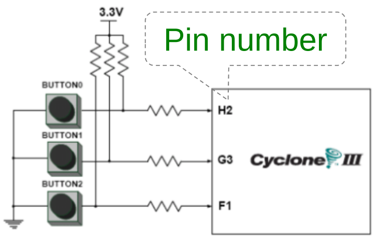


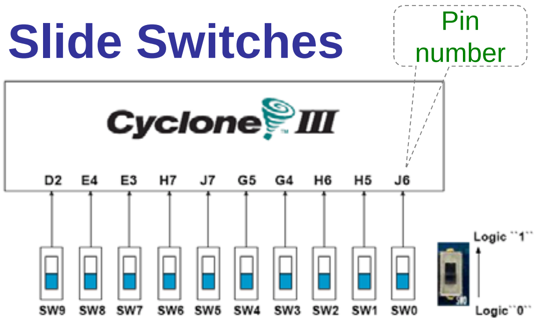
BCD priority encoder



3-digit octal-to-7segment decoder

Pushbutton and Slide Switches





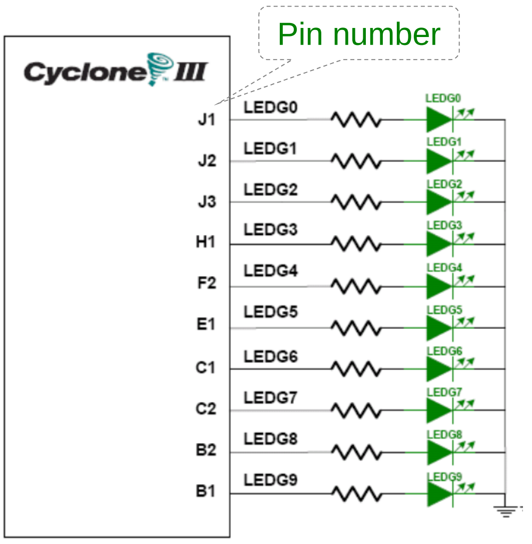
3 Pushbutton switches:
Not pressed → Logic High
Pressed → Logic Low

10 Slide switches (Sliders):
Up → Logic High
Down → Logic

Signal Name	FPGA Pin No.
BUTTON [0]	PIN_ H2
BUTTON [1]	PIN_ G3
BUTTON [2]	PIN_ F1

SW[0]	PIN_J6	SW[5]	PIN_J7
SW[1]	PIN_H5	SW[6]	PIN_H7
SW[2]	PIN_H6	SW[7]	PIN_E3
SW[3]	PIN_G4	SW[8]	PIN_E4
SW[4]	PIN_G5	SW[9]	PIN_D2

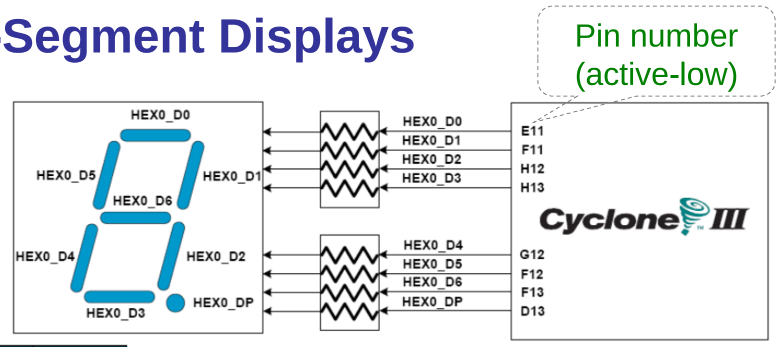
LEDs



10 LEDs
Opuput high → LED on
Output low → LED off

Signal Name	FPGA Pin No.
LEDG[0]	PIN_J1
LEDG[1]	PIN_J2
LEDG[2]	PIN_J3
LEDG[3]	PIN_H1
LEDG[4]	PIN_F2
LEDG[5]	PIN_E1
LEDG[6]	PIN_C1
LEDG[7]	PIN_C2
LEDG[8]	PIN_B2
LEDG[9]	PIN_B1

7-Segment Displays



Signal Name	FPGA Pin No.
HEX0_D[0]	PIN_E11
HEX0_D[1]	PIN_F11
HEX0_D[2]	PIN_H12
HEX0_D[3]	PIN_H13
HEX0_D[4]	PIN_G12
HEX0_D[5]	PIN_F12
HEX0_D[6]	PIN_F13
HEX0_DP	PIN_D13

HEX1_D[0]	PIN_A13
HEX1_D[1]	PIN_B13
HEX1_D[2]	PIN_C13
HEX1_D[3]	PIN_A14
HEX1_D[4]	PIN_B14
HEX1_D[5]	PIN_E14
HEX1_D[6]	PIN_A15
HEX1_DP	PIN_B15

HEX2_D[0]	PIN_D15
HEX2_D[1]	PIN_A16
HEX2_D[2]	PIN_B16
HEX2_D[3]	PIN_E15
HEX2_D[4]	PIN_A17
HEX2_D[5]	PIN_B17
HEX2_D[6]	PIN_F14
HEX2_DP	PIN_A18

HEX3_D[0]	PIN_B18
HEX3_D[1]	PIN_F15
HEX3_D[2]	PIN_A19
HEX3_D[3]	PIN_B19
HEX3_D[4]	PIN_C19
HEX3_D[5]	PIN_D19
HEX3_D[6]	PIN_G15
HEX3_DP	PIN_G16