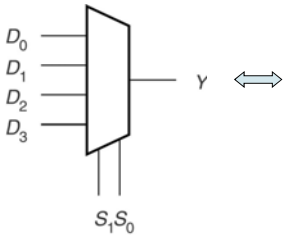
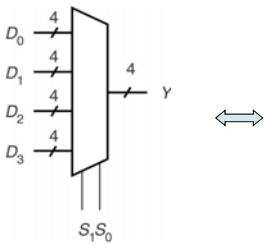
# Class 04
# MUX / DMUX
# and
# Full Adder



## Multiplexer (MUX)




| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |




| $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | $D_{03}$ | $D_{02}$ | $D_{01}$ | $D_{00}$ |
| 0 | 1 | $D_{13}$ | $D_{12}$ | $D_{11}$ | $D_{10}$ |
| 1 | 0 | $D_{23}$ | $D_{22}$ | $D_{21}$ | $D_{20}$ |
| 1 | 1 | $D_{33}$ | $D_{32}$ | $D_{31}$ | $D_{30}$ |

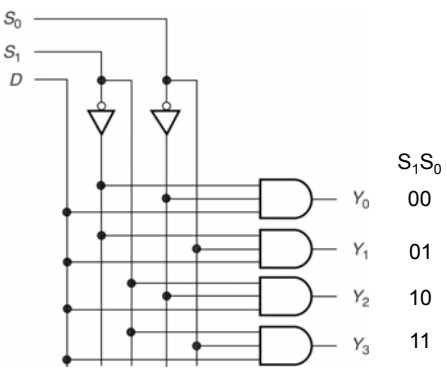
# Multiplexer (MUX)

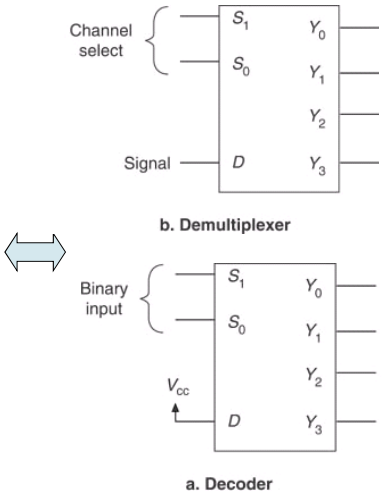
```
ENTITY mux4sel IS
  PORT(
    s: IN      BIT_VECTOR (1 downto 0);
    d: IN      BIT_VECTOR (3 downto 0);
    y: OUT   BIT);
END mux4sel;

ARCHITECTURE a OF mux4sel IS
BEGIN
-  Selected Signal Assignment
MUX4: WITH s SELECT
  y <=        d(0) WHEN "00",
              d(1) WHEN "01",
              d(2) WHEN "10",
              d(3) WHEN "11";
END a;
```

```
ENTITY mux4case IS
  PORT(
    d0, d1, d2, d3: IN BIT; -- data inputs
    s: IN BIT_VECTOR (1 downto 0); -- select inputs
    y: OUT BIT);
END mux4case;

ARCHITECTURE mux4to1 OF mux4case IS
BEGIN
  -- Monitor select inputs and execute if they change
  PROCESS(s)
  BEGIN
    CASE s IS
      WHEN "00"        =>      y <= d0;
      WHEN "01"        =>      y <= d1;
      WHEN "10"        =>      y <= d2;
      WHEN "11"        =>      y <= d3;
      WHEN others      =>      y <= '0';
    END CASE;
  END PROCESS;
END mux4to1;
```



# Demultiplexer (DMUX)



# Demultiplexer (DMUX) (Cont.)

```
ENTITY dmux8 IS
  PORT(
    s: IN      STD_LOGIC_VECTOR(2 downto 0);
    d: IN      STD_LOGIC;
    y: OUT   STD_LOGIC_VECTOR(0 to 7));
END dmux8;

ARCHITECTURE a OF dmux8 IS
  SIGNAL inputs : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
  inputs <= d & s;

  WITH inputs SELECT
    y <= "01111111" WHEN "0000",
         "10111111" WHEN "0001",
         "11011111" WHEN "0010",
         "11101111" WHEN "0011",
         "11110111" WHEN "0100",
         "11111011" WHEN "0101",
         "11111101" WHEN "0110",
         "11111110" WHEN "0111",
         "11111111" WHEN others;
END a;
```

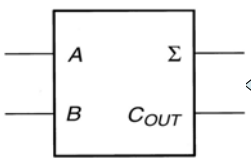
s: selector

"11" & d & "11111" WHEN "0010",

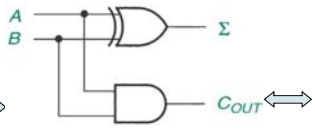
d: signal



# Half Adder




Half adder




$Cout = AB$

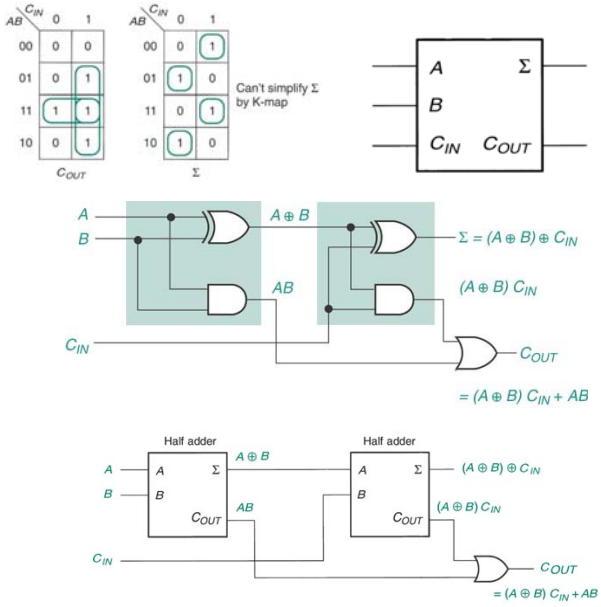
$\sum = \overline{A}B + A\overline{B} = A \oplus B$

| A | B | $C_{out}$ | Σ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Full Adder

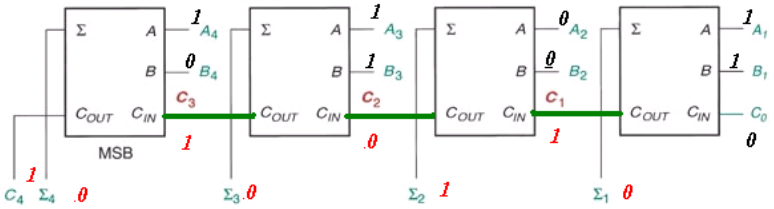
| A | B | $C_{IN}$ | $C_{out}$ | $\Sigma$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

$$Cout = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$= (\bar{A}B + A\bar{B})C + AB(\bar{C} + C)$$
$$= (A \oplus B)C + AB$$
$$\sum = \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$
$$= (\overline{A \oplus B})C + (A \oplus B)\bar{C}$$
$$= (A \oplus B) \oplus C$$



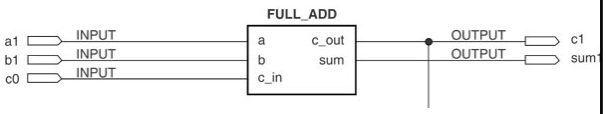



# Parallel Binary Adder (Ripple Carry Binary Adder)




$$A_4A_3A_2A_1 = 1101$$
$$B_4B_3B_2B_1 = 0101$$
$$A + B = 10010$$

# Full Adder

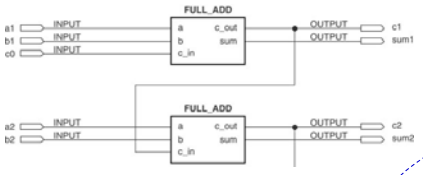



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY full_add IS
 PORT(
   a, b, c_in          : IN      STD_LOGIC;
   c_out, sum          : OUT     STD_LOGIC);
END full_add;

ARCHITECTURE adder OF full_add IS
BEGIN
 c_out <=((a xor b) and c_in) or (a and b);
 sum <=  (a xor b) xor c_in;
END adder;
```

$$Cout = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$
$$= (\overline{A}B + A\overline{B})C + AB(\overline{C} + C)$$
$$= (A \oplus B)C + AB$$
$$\sum = \overline{A}\overline{B}C + ABC + A\overline{B}\overline{C} + \overline{A}B\overline{C}$$
$$= (\overline{A}\overline{B} + AB)C + (\overline{A}B + A\overline{B})\overline{C}$$
$$= (\overline{A \oplus B})C + (A \oplus B)\overline{C}$$
$$= ((A \oplus B) \oplus C$$



# 2-Bit Full Adder




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

ENTITY full_add IS
 PORT(
   a, b, c_in          : IN      STD_LOGIC;
   c_out, sum          : OUT     STD_LOGIC);
END full_add;

ARCHITECTURE adder OF full_add IS
BEGIN
 c_out <=((a xor b) and c_in) or (a and b);
 sum <=  (a xor b) xor c_in;
END adder;
```

Label is needed

Each component should have different label name

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

ENTITY add2par IS
 PORT(
   c0: IN      STD_LOGIC;
   a, b: IN    STD_LOGIC_VECTOR(2 downto 1);
   c2: OUT     STD_LOGIC;
   sum: OUT STD_LOGIC_VECTOR(2 downto 1));
END add2par;

ARCHITECTURE adder OF add2par IS
  -- Component declaration
  COMPONENT full_add
   PORT(
      a, b, c_in: IN          STD_LOGIC;
      c_out, sum: OUT         STD_LOGIC);
  END COMPONENT;
  -- Define a signal for internal carry bits
  SIGNAL c : STD_LOGIC_VECTOR(1 downto 1);
BEGIN
  --  Two Component Instantiation Statements
  adder1: full_add
    PORT MAP ( a      => a(1),
               b      => b(1),
               c_in   => c0,
               c_out  => c(1),
               sum    => sum(1));
  adder2: full_add
    PORT MAP ( a      => a(2),
               b      => b(2),
               c_in   => c(1),
               c_out  => c2,
               sum    => sum(2));
END adder;
```

Connect c_out of adder1 to c_in of adder2

# 2-Bit Full Adder (Cont.)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY add4gen IS
  PORT(
    c0: IN      STD_LOGIC;
    a, b: IN    STD_LOGIC_VECTOR(2 downto 1);
    c2: OUT     STD_LOGIC;
    sum: OUT STD_LOGIC_VECTOR(2 downto 1));
END add4gen;

ARCHITECTURE adder OF add4gen IS
  -- Component declaration
  COMPONENT full_add
    PORT( a, b, c_in: IN      STD_LOGIC;
          c_out, sum: OUT STD_LOGIC);
  END COMPONENT;
  -- Define a signal for internal carry bits
  SIGNAL c : STD_LOGIC_VECTOR (2 downto 0);
BEGIN
  c(0) <= c0;
  adder1: full_add PORT MAP (a(1), b(1), c0, c(1), sum(1));
  adders:
  FOR i IN 1 to 2 GENERATE
    adder: full_add PORT MAP (a(i), b(i), c(i-1), c(i), sum(i));
  END GENERATE;
  c2 <= c(2);
END adder;
```

Label is needed

adder2: full_add PORT MAP (a(2), b(2), c(1), c2, sum(2));

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

ENTITY add2par IS
  PORT(
    c0: IN      STD_LOGIC;
    a, b: IN    STD_LOGIC_VECTOR(2 downto 1);
    c2: OUT     STD_LOGIC;
    sum: OUT STD_LOGIC_VECTOR(2 downto 1));
END add2par;

ARCHITECTURE adder OF add2par IS
  -- Component declaration
  COMPONENT full_add
    PORT(
      a, b, c_in: IN      STD_LOGIC;
      c_out, sum: OUT     STD_LOGIC);
  END COMPONENT;
  -- Define a signal for internal carry bits
  SIGNAL c : STD_LOGIC_VECTOR(1 downto 1);
BEGIN
  -- Two Component Instantiation Statements
  adder1: full_add
    PORT MAP ( a      => a(1),
               b      => b(1),
               c_in   => c0,
               c_out  => c(1),
               sum    => sum(1));
  adder2: full_add
    PORT MAP ( a      => a(2),
               b      => b(2),
               c_in   => c(1),
               c_out  => c2,
               sum    => sum(2));
END adder;
```



# Full Adder with Unspecified Width

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY addxgen IS
  GENERIC (width : INTEGER := 8);
  PORT(
    c0: IN STD_LOGIC;
    a, b: IN STD_LOGIC_VECTOR(width downto 1);
    c_max: OU TSTD_LOGIC;
    sum: OUT STD_LOGIC_VECTOR(width downto 1));
END addxgen;

ARCHITECTURE adder OF addxgen IS
  -- Component declaration
  COMPONENT full_add
    PORT(
      a, b, c_in : IN STD_LOGIC;
      c_out, sum: OUT STD_LOGIC);
  END COMPONENT;
  -- Define a signal for internal carry bits
  SIGNAL c : STD_LOGIC_VECTOR (width downto 0);
BEGIN
  c(0) <= c0;
  adders:
  FOR i IN 1 to width GENERATE
    adder: full_add PORT MAP (a(i), b(i), c(i-1), c(i), sum(i));
  END GENERATE;
  c_max <= c(width);
END adder;
```

Default value required, but can be redefined.

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

ENTITY add16gen IS
  PORT(
    c0: IN  STD_LOGIC;
    a, b:IN  STD_LOGIC_VECTOR(16 downto 1);
    c16: OUT STD_LOGIC;
    sum: OUT STD_LOGIC_VECTOR(16 downto 1));
END add16gen;

ARCHITECTURE adder of add16gen IS
  COMPONENT addxgen
    GENERIC (width : INTEGER);
    PORT(
      c0: IN  STD_LOGIC;
      a, b: IN  STD_LOGIC_VECTOR(width downto 1);
      c_max: OUT STD_LOGIC;
      sum: OUT STD_LOGIC_VECTOR(width downto 1));
  END COMPONENT;
BEGIN
  add16 : addxgen
  GENERIC MAP(width => 16)
  PORT MAP(c0, a, b, c16, sum);
END adder;
```

Included components should be in the same Quartus project as well.

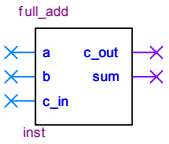
"width" is specified in GENERIC MAP

# The Procedure to Import VHDL Code to Block Diagram/Schematic File

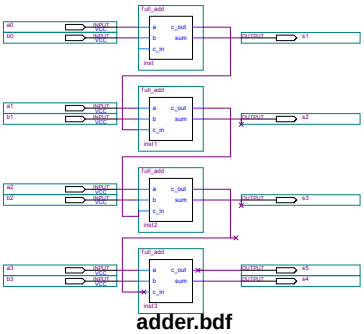
- The procedure to import a VHDL full-adder to a .bdf file to construct a four-bit full adder:
  - 1. Create a quartus project with entity name "adder"
  - 2. Create a new full_add.vhd file and save it as a full_add.bsf file. (File→Create/Update→Create Symbol File…)
  - 3. Create a new adder.bdf file (the file name is its entity name)
  - 4. Incude full_add.bsf file as a component into adder.bdf
  - 5. Pin assignment to complete the design

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

ENTITY full_add IS
  PORT(
    a, b, c_in     : IN      STD_LOGIC;
    c_out, sum     : OUT     STD_LOGIC);
END full_add;

ARCHITECTURE adder OF full_add IS
BEGIN
  c_out <=      ((a xor b) and c_in) or (a and b);
  sum <=        (a xor b) xor c_in;
END adder;
```

**full_add.vhd**




**full_add.bsf**

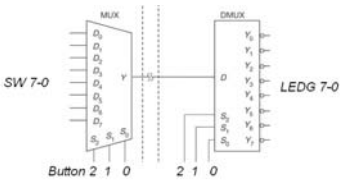


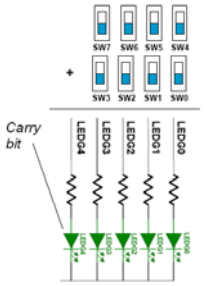
**adder.bdf**



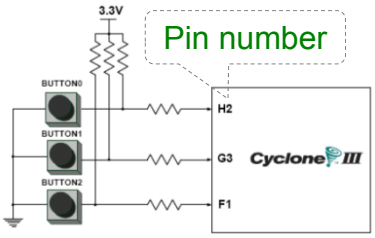
# Lab 04

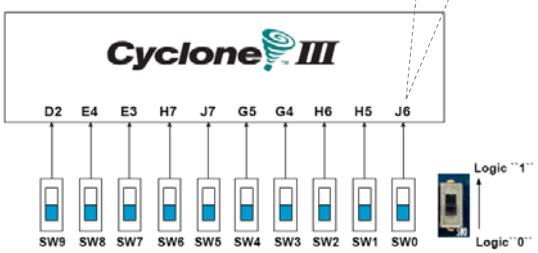
- Part 1: Design a MUX/DMUX
  - Use Button2-Button0 as the selectors to decide which slide switch among SW7-SW0 is selected to show its status on its corresponding LED. The LEDs that are not selected should be turned off. For example:
    - When Button2 is pushed, the status of SW4 is shown on LEDG4.
    - When Button2 and Button0 are both pushed, the status of SW5 is shown on LEDG5.
- Part 2: Full adder
  - Implement a 4-bit full adder:
    - SW7-4 is the first 4-bit operand, and SW3-0 is the second 4-bit operand.
    - Please show the result on LEDs, where LEDG4 is the carry of the MSB bit, and LEDG3-0 are ∑3-0, respectively.
      - An LED is on when the corresponding ∑ bit is 1.



# Pushbutton and Slide Switches




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

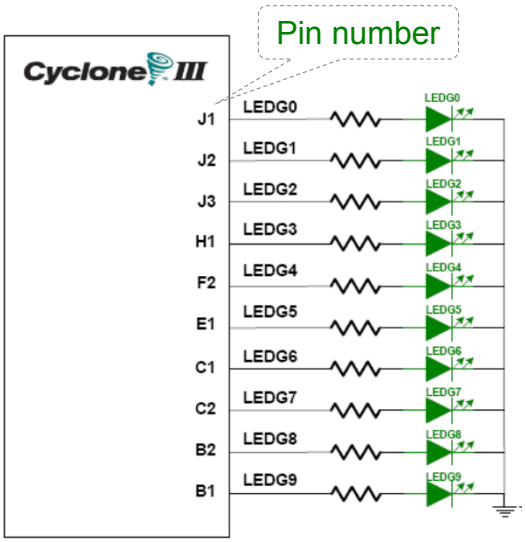
| Signal Name | FPGA Pin No. |
|---|---|
| BUTTON [0] | PIN_ H2 |
| BUTTON [1] | PIN_ G3 |
| BUTTON [2] | PIN_ F1 |

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

| | | | |
|---|---|---|---|
| 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

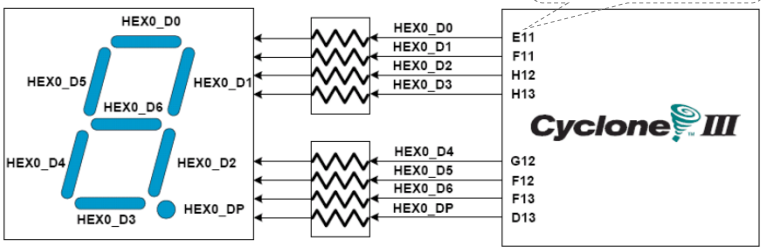



10 LEDs
Opuput high → LED on
Output low → LED off

| 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

Pin number (active-low)




| Signal Name | FPGA Pin No. | | | | | | |
|---|---|---|---|---|---|---|---|
| HEX0_D[0] | PIN_E11 | HEX1_D[0] | PIN_A13 | HEX2_D[0] | PIN_D15 | HEX3_D[0] | PIN_B18 |
| HEX0_D[1] | PIN_F11 | HEX1_D[1] | PIN_B13 | HEX2_D[1] | PIN_A16 | HEX3_D[1] | PIN_F15 |
| HEX0_D[2] | PIN_H12 | HEX1_D[2] | PIN_C13 | HEX2_D[2] | PIN_B16 | HEX3_D[2] | PIN_A19 |
| HEX0_D[3] | PIN_H13 | HEX1_D[3] | PIN_A14 | HEX2_D[3] | PIN_E15 | HEX3_D[3] | PIN_B19 |
| HEX0_D[4] | PIN_G12 | HEX1_D[4] | PIN_B14 | HEX2_D[4] | PIN_A17 | HEX3_D[4] | PIN_C19 |
| HEX0_D[5] | PIN_F12 | HEX1_D[5] | PIN_E14 | HEX2_D[5] | PIN_B17 | HEX3_D[5] | PIN_D19 |
| HEX0_D[6] | PIN_F13 | HEX1_D[6] | PIN_A15 | HEX2_D[6] | PIN_F14 | HEX3_D[6] | PIN_G15 |
| HEX0_DP | PIN_D13 | HEX1_DP | PIN_B15 | HEX2_DP | PIN_A18 | HEX3_DP | PIN_G16 |