# EE434
# ASIC & Digital Systems

## VHDL
## Practice (Combinational Logic)

Spring 2016
Dae Hyun Kim
daehyun@eecs.wsu.edu

# Data Types

- BIT
  - used for ‘0’ and ‘1’.
- std_logic
  - used for ‘0’, ‘1’, ‘X’, ‘U’, ‘Z’, ...
  - ‘U’: uninitialized
  - ‘X’: unknown (impossible to determine this value)
  - ‘Z’: high-impedance
- To use std_logic, you should include the following lines before each entity definition.
  - LIBRARY IEEE;
  - USE IEEE.std_logic_1164.all;

# Data Types

- std_logic_vector
  - used for buses
  - Definition
    - signal_name : direction std_logic_vector (n downto 0);
  - Example
    - S : OUT std_logic_vector (3 DOWNTO 0);
  - How to access the k-th element in the vector
    - S(k-1)

# Inverter

- Entity name
  - myINV
- Inputs
  - A
- Outputs
  - Y
- Function
  - $Y = \bar{A}$

# Inverter

- Entity name
  - myINV
- Inputs
  - A
- Outputs
  - Y
- Function
  - $Y = \bar{A}$

```
ENTITY  myINV  IS
   PORT  (  A : IN std_logic;
            Y : OUT std_logic );
END  myINV;

ARCHITECTURE  myINV_arch  OF  myINV  IS
BEGIN
   Y <= (NOT A);
END  myINV_arch;
```

# Buffer

- Entity name
  - myBUF
- Inputs
  - A
- Outputs
  - Y
- Function
  - $Y = A$

# Buffer

- Entity name
  - myBUF
- Inputs
  - A
- Outputs
  - Y
- Function
  - $Y = A$

```
ENTITY  myBUF  IS
   PORT  (  A : IN std_logic;
            Y : OUT std_logic );
END  myBUF;

ARCHITECTURE  myBUF_arch  OF  myBUF  IS
BEGIN
   Y <= A;
END  myBUF_arch;
```

# Buffer

- Entity name
  - myBUF
- Inputs
  - A
- Outputs
  - Y
- Function
  - $Y = A$

```
ENTITY myBUF IS
   PORT ( A : IN std_logic;
            Y : OUT std_logic );
END myBUF;

ARCHITECTURE myBUF_arch OF myBUF IS
   COMPONENT myINV PORT ( A : IN std_logic; Y : OUT std_logic );
   END COMPONENT;
   SIGNAL n1 : std_logic;
BEGIN
   u1 : myINV PORT MAP ( A => A, Y => n1 );
   u2 : myINV PORT MAP ( A => n1, Y => Y);
END myBUF_arch;
```

# Two-Input NAND

- Entity name
  - myNAND2
- Inputs
  - A, B
- Outputs
  - Y
- Function
  - $Y = \overline{A \cdot B}$

# Two-Input NAND

- Entity name
  - myNAND2
- Inputs
  - A, B
- Outputs
  - Y
- Function
  - $Y = \overline{A \cdot B}$

```
ENTITY  myNAND2  IS
   PORT  (  A, B : IN std_logic;
                Y : OUT std_logic );
END  myNAND2;

ARCHITECTURE  myNAND2_arch  OF  myNAND2  IS
BEGIN
   Y <= NOT (A AND B);
END  myNAND2_arch;
```

# Full Adder

- Entity name
  - myFA
- Inputs
  - A, B, CI
- Outputs
  - S, CO
- Function
  - $S = A \oplus B \oplus CI$
  - $CO = A \cdot B + B \cdot CI + A \cdot CI$

# Full Adder

- Entity name
  - myFA
- Inputs
  - A, B, CI
- Outputs
  - S, CO
- Function
  - $S = A \oplus B \oplus CI$
  - $CO = A \cdot B + B \cdot CI + A \cdot CI$

```
ENTITY  myFA  IS
   PORT  (  A, B, CI : IN std_logic;
            S, CO : OUT std_logic );
END  myFA;

ARCHITECTURE  myFA_arch  OF  myFA  IS
BEGIN
   S <= A XOR B XOR CI;
   CO <= (A AND B) OR (B AND CI) OR (A AND CI);
END  myFA_arch;
```

# 2:1 MUX

- Entity name
  - myMUX2
- Inputs
  - A, B, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S0} \cdot A + S0 \cdot B$

# 2:1 MUX

- Entity name
  - myMUX2
- Inputs
  - A, B, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S0} \cdot A + S0 \cdot B$

```
ENTITY  myMUX2  IS
   PORT  (  A, B, S0 : IN std_logic;
                Y : OUT std_logic );
END  myMUX2;

ARCHITECTURE  myMUX2_arch  OF  myMUX2  IS
BEGIN
   Y <= ((NOT S0) AND A) OR (S0 AND B);
END  myMUX2_arch;
```

# 2:1 MUX

- Entity name
  - myMUX2
- Inputs
  - A, B, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S0} \cdot A + S0 \cdot B$

```
ENTITY  myMUX2  IS
   PORT  (  A, B, S0 : IN std_logic;
                Y : OUT std_logic );
END  myMUX2;

ARCHITECTURE  myMUX2_arch  OF  myMUX2  IS
BEGIN
   Y <= A  WHEN  S0 = '0'
      ELSE  B;
END  myMUX2_arch;
```

# 4:1 MUX

- Entity name
  - myMUX4
- Inputs
  - A, B, C, D, S1, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S1} \cdot \overline{S0} \cdot A + \overline{S1} \cdot S0 \cdot B + S1 \cdot \overline{S0} \cdot C + S1 \cdot S0 \cdot D$

# 4:1 MUX

- Entity name
  - myMUX4
- Inputs
  - A, B, C, D, S1, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S1} \cdot \overline{S0} \cdot A + \overline{S1} \cdot S0 \cdot B + S1 \cdot \overline{S0} \cdot C + S1 \cdot S0 \cdot D$

```
ENTITY myMUX4 IS
   PORT ( A, B, C, D, S1, S0 : IN std_logic;
                Y : OUT std_logic );
END myMUX4;

ARCHITECTURE myMUX4_arch OF myMUX4 IS
BEGIN
   Y <= ((NOT S1) AND (NOT S0) AND A) OR
        ((NOT S1) AND S0 AND B) OR
        (S1 AND (NOT S0) AND C) OR
        (S1 AND S0 AND D);
END myMUX4_arch;
```

# 4:1 MUX

- Entity name
  - myMUX4
- Inputs
  - A, B, C, D, S1, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S1} \cdot \overline{S0} \cdot A + \overline{S1} \cdot S0 \cdot B + S1 \cdot \overline{S0} \cdot C + S1 \cdot S0 \cdot D$

```
ENTITY myMUX4 IS
   PORT ( A, B, C, D, S1, S0 : IN std_logic;
                Y : OUT std_logic );
END myMUX4;

ARCHITECTURE myMUX4_arch OF myMUX4 IS
BEGIN
   Y <= A WHEN ((S1 = '0') AND (S0 = '0))
      ELSE B WHEN ((S1 = '0') AND (S0 = '1'))
      ELSE C WHEN ((S1 = '1') AND (S0 = '0'))
      ELSE D;
END myMUX4_arch;
```

# 4:1 MUX

- Entity name
  - myMUX4
- Inputs
  - A, B, C, D, S1, S0
- Outputs
  - Y
- Function
  - $Y = \overline{S1} \cdot \overline{S0} \cdot A + \overline{S1} \cdot S0 \cdot B + S1 \cdot \overline{S0} \cdot C + S1 \cdot S0 \cdot D$

```
ENTITY myMUX4 IS
   PORT ( A, B, C, D, S1, S0 : IN std_logic;
          Y : OUT std_logic );
END myMUX4;

ARCHITECTURE myMUX4_arch OF myMUX4 IS
   COMPONENT myMUX2 PORT ( A, B, S0 : IN std_logic; Y : OUT std_logic );
   END COMPONENT;
   SIGNAL n1, n2 : std_logic;
BEGIN
   u1 : myMUX2 PORT MAP ( A => A, B => B, S0 => S0, Y => n1 );
   u2 : myMUX2 PORT MAP ( A => C, B => D, S0 => S0, Y => n2 );
   u3 : myMUX2 PORT MAP ( A => n1, B => n2, S0 => S1, Y => Y);
END myMUX4_arch;
```

# 4:1 MUX

- Entity name
  - myMUX4
- Inputs
  - A, B, C, D, S (1 downto 0)
- Outputs
  - Y
- Function
  - $Y = \overline{S(1)} \cdot \overline{S(0)} \cdot A + \overline{S(1)} \cdot S(0) \cdot B + S(1) \cdot \overline{S(0)} \cdot C + S(1) \cdot S(0) \cdot D$

# 4:1 MUX

- Entity name
  - myMUX4
- Inputs
  - A, B, C, D, S
- Outputs
  - Y
- Function
  - $Y = \overline{S1} \cdot \overline{S0} \cdot A + \overline{S1} \cdot S0 \cdot B + S1 \cdot \overline{S0} \cdot C + S1 \cdot S0 \cdot D$

```
ENTITY myMUX4 IS
   PORT ( A, B, C, D : IN std_logic;
                S : IN std_logic_vector (1 DOWNTO 0);
                Y : OUT std_logic );
END myMUX4;

ARCHITECTURE myMUX4_arch OF myMUX4 IS
BEGIN
   Y <= A WHEN S = "00"
      ELSE B WHEN S = "01"
      ELSE C WHEN S = "10"
      ELSE D;
END myMUX4_arch;
```

# Two-Bit Adder

- Entity name
  - myADD2
- Inputs
  - A (std_logic_vector, 1:0), B (std_logic_vector, 1:0), CI
- Outputs
  - S (std_logic_vector, 1:0), CO
- Function
  - Add two two-bit numbers

# Two-Bit Adder

```
ENTITY  myADD2  IS
   PORT  (  A, B : IN std_logic_vector (1 DOWNTO 0);
            CI : IN std_logic;
            S : OUT std_logic_vector (1 DOWNTO 0);
            Y : OUT std_logic );
END  myADD2;

ARCHITECTURE  myADD2_arch  OF  myADD2  IS
BEGIN
   S(0) <= A(0) XOR B(0) XOR CI;
   n1 <= (A(0) AND B(0)) OR (B(0) AND CI) OR (A(0) AND CI));
   S(1) <= A(1) XOR B(1) XOR n1;
   CO <= (A(1) AND B(1)) OR (B(1) AND n1) OR (A(1) AND n1));
END  myADD2_arch;
```

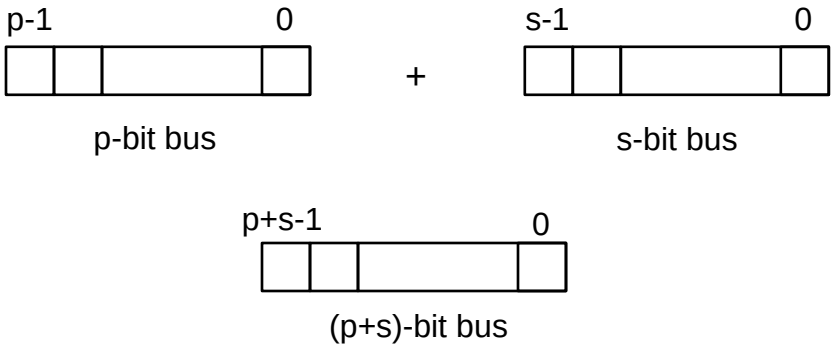
# Two-Bit Adder

```
ENTITY  myADD2  IS
   PORT  (  A, B : IN std_logic_vector (1 DOWNTO 0);
            CI : IN std_logic;
            S : OUT std_logic_vector (1 DOWNTO 0);
            Y : OUT std_logic );
END  myADD2;

ARCHITECTURE  myADD2_arch  OF  myADD2  IS
   COMPONENT  myFA  PORT  ( A, B, CI : IN std_logic;  S, CO : OUT std_logic );
   END COMPONENT;
   SIGNAL n1 : std_logic;
BEGIN
   u1 : myFA  PORT  MAP  ( A => A(0), B => B(0), CI => CI, S => S(0), CO => n1);
   u2 : myFA  PORT  MAP  ( A => A(1), B => B(1), CI => n1, S => S(1), CO => CO);
END  myADD2_arch;
```

# Signal Concatenation

- vector + vector




- Out <= v1 & v2;

# Two-Bit Adder

```
LIBRARY IEEE;
USE  IEEE.std_logic_1164.all;
USE  IEEE.std_logic_signed.all;

ENTITY  myADD2  IS
   ...
END  myADD2;

ARCHITECTURE  myADD2_arch  OF  myADD2  IS
   SIGNAL  n1, n2, n3 : std_logic_vector (2 DOWNTO 0);
BEGIN
   n1 <= '0' & A;       -- sign extension by concatenation
   n2 <= '0' & B;       -- sign extension by concatenation
   n3 <= n1 + n2 + CI;  <-- need "USE IEEE.std_logic_signed.all"
   S <= n3 (1 DOWNTO 0);  <-- partial element access of a vector
   CO <= n3 (2);
END  myADD2_arch;
```