

## Class 02

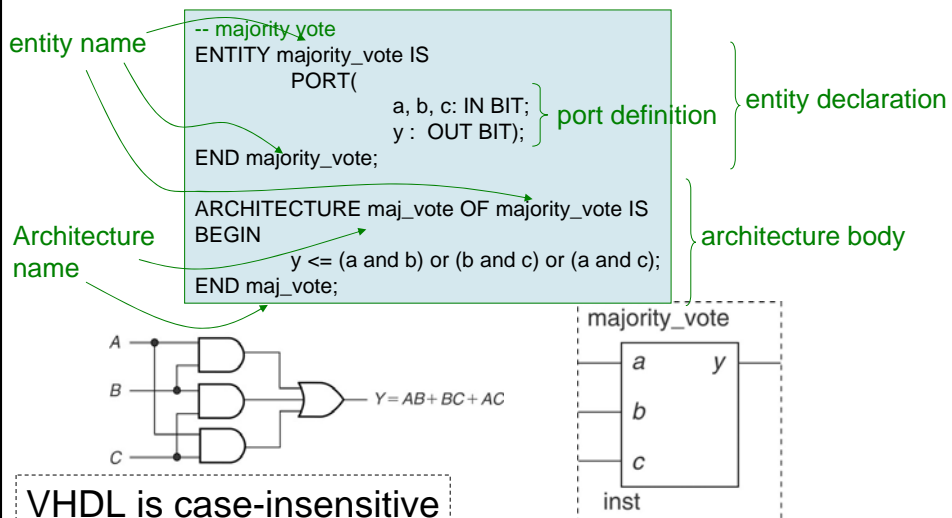
# VHDL Introduction

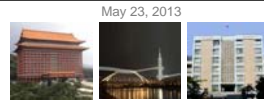


May 23, 2013

2

## VHDL ENTITY and ARCHITECTURE





May 23, 2013

3

## AOI

- Solve  $Y = \overline{AB} + \overline{AC} + D$

```

-- This is comment
ENTITY logic_circuit IS
    PORT(
        a, b, c, d: IN BIT;
        y: OUT BIT);
END logic_circuit;

ARCHITECTURE cct OF logic_circuit IS
BEGIN
    y <= not ((a and b) or ((not a) and (not c)) or d);
END cct;

```

Comment

Mode

Type

Logic operations:  
and, or, not, xor,  
nand, nor

assign



May 23, 2013

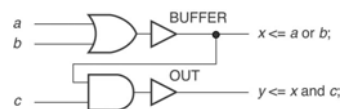
4

## Modes and Types

- Modes:

– IN, OUT, INOUT, BUFFER

BUFFER is the same as OUT, but allows to be fed back to the CPLD logic to be reused by another function.



- Types

– BIT:

– BIT, BIT\_VECTOR

– STD\_LOGIC:

– STD\_LOGIC, STD\_LOGIC\_VECTOR

– INTEGER

– INTEGER, NATURAL, POSITIVE

One bit

Multiple bits/Array of bits

Equal or larger than 0

Equal or larger than 1

Logic operations:  
and, or, not, xor,  
nand, nor



May 23, 2013

5

## 4-Bit AND Array

```

d(3) <= '0'; d(2) <= '1';
d(1) <= '0'; d(0) <= '1';
IN BIT_VECTOR (3 downto 0)
  d <= "0101";
IN BIT_VECTOR (0 to 3)
  d <= "1010";

```

```

-- 4-bit bitwise and function
-- y0 = a0 and b0; y1 = a1 and b1; etc.
ENTITY bitwise_and_4 IS
  PORT(
    a0, a1, a2, a3 : IN BIT;
    b0, b1, b2, b3 : IN BIT;
    y0, y1, y2, y3 : OUT BIT);
END bitwise_and_4;

```

Ports defined individually

```

ARCHITECTURE and_gate OF bitwise_and_4 IS
BEGIN

```

```

  y0 <= a0 and b0;
  y1 <= a1 and b1;
  y2 <= a2 and b2;
  y3 <= a3 and b3;
END and_gate;

```

Outputs assigned individually

```

-- 4-bit bitwise and function
-- y = a and b;

```

```

ENTITY bitwise_and_vec_4 IS
  PORT(
    a, b: IN BIT_VECTOR(3 downto 0);
    y: OUT BIT_VECTOR(3 downto 0));
END bitwise_and_vec_4;

```

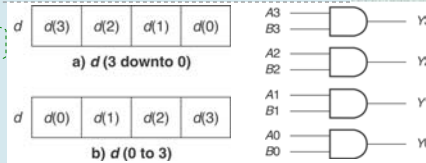
Ports defined as vectors

```

ARCHITECTURE and_gate OF bitwise_and_vec_4 IS
BEGIN
  y <= a and b;
END and_gate;

```

Outputs assigned as a vector



May 23, 2013

6

## WITH ... SELECT

D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

```

ENTITY select_example IS

```

```

  PORT(
    d: IN BIT_VECTOR(3 downto 0);
    y: OUT BIT);
END select_example;

```

Select y based on d

```

ARCHITECTURE cct OF select_example IS
BEGIN

```

```

  WITH d SELECT

```

```

    y <= '1' WHEN "0011",
        '1' WHEN "0110",
        '1' WHEN "1001",
        '1' WHEN "1100",
        '0' WHEN others;

```

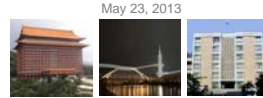
```

END cct;

```

Value of y

Default is required



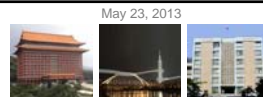
7

## STD\_LOGIC and STD\_LOGIC\_VECTOR

- STD\_LOGIC is also called **IEEE Std.1164 Multi-Valued Logic**
- To use STD\_LOGIC, we must include the package:

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

'U'	Uninitialized
'X'	Forcing Unknown
'0'	Forcing 0
'1'	Forcing 1
'Z'	High Impedance
'W'	Weak Unknown
'L'	Weak 0 (pull-down resistor)
'H'	Weak 1 (pull-up resistor)
'-'	Don't Care



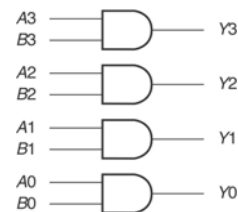
8

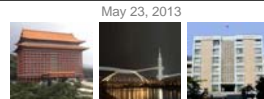
## STD\_LOGIC and STD\_LOGIC\_VECTOR (Cont.)

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

```
ENTITY bitwise_and_std_4 IS  
  PORT(  
    a, b: IN STD_LOGIC_VECTOR(3 downto 0);  
    y: OUT STD_LOGIC_VECTOR(3 downto 0));  
END bitwise_and_std_4;
```

```
ARCHITECTURE and_gate OF bitwise_and_std_4 IS  
  BEGIN  
    y <= a and b;  
  END and_gate;
```





9

## Tristate

```

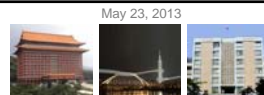
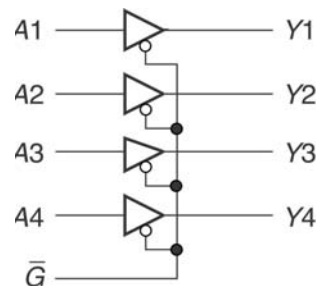
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY quad_tri IS
  PORT(
    a: IN  STD_LOGIC_VECTOR(3 downto 0);
    g: IN  STD_LOGIC;
    y: OUT STD_LOGIC_VECTOR(3 downto 0));
END quad_tri;

ARCHITECTURE quad_buff OF quad_tri IS
BEGIN
  WITH g SELECT
    y <=  a      WHEN '0',
          "ZZZZ"  WHEN others;
END quad_buff;

```

Y1	Y2	Y3	Y4	$\bar{G}$
A1	A2	A3	A4	0
'Z'	'Z'	'Z'	'Z'	1



10

## INTEGER

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY truth_table IS
  PORT(
    d: IN  INTEGER RANGE 0 to 7;
    y: OUT STD_LOGIC);
END truth_table;

ARCHITECTURE a OF truth_table IS
BEGIN
  WITH d SELECT
    y <=  '1'  WHEN 1,
          '1'  WHEN 5,
          '1'  WHEN 6,
          '0'  WHEN others;
END a;

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY truth_table IS
  PORT(
    d: IN  STD_LOGIC_VECTOR(2 downto 0);
    y: OUT STD_LOGIC);
END truth_table;

ARCHITECTURE a OF truth_table IS
BEGIN
  WITH d SELECT
    y <=  '1'  WHEN "001",
          '1'  WHEN "101",
          '1'  WHEN "110",
          '0'  WHEN others;
END a;

```

D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

STD\_LOGIC

integer



May 23, 2013

11

## SIGNAL

- SIGNAL can bundle inputs or outputs into a single group.

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

```
ENTITY signal_ex IS
  PORT(
    a, b, c : IN STD_LOGIC;
    w, x, y, z : OUT STD_LOGIC);
END signal_ex;
```

```
ARCHITECTURE sig OF signal_ex IS
```

```
-- Declaration area
```

```
-- Define signals here
```

```
SIGNAL inputs : STD_LOGIC_VECTOR(2 downto 0);
```

```
SIGNAL outputs : STD_LOGIC_VECTOR(3 downto 0);
```

```
BEGIN
```

```
-- Concatenate input ports into 3-bit signal
```

```
inputs <= a & b & c;
```

```
WITH inputs SELECT
```

```
outputs <=
```

```
"1000" WHEN "000",
```

```
"0100" WHEN "001",
```

```
"0110" WHEN "010",
```

```
"1001" WHEN "011",
```

```
"0110" WHEN "100",
```

```
"0001" WHEN "101",
```

```
"1001" WHEN "110",
```

```
"0010" WHEN "111",
```

```
"0000" WHEN others;
```

```
-- Separate signal
```

```
w <= outputs(3);
```

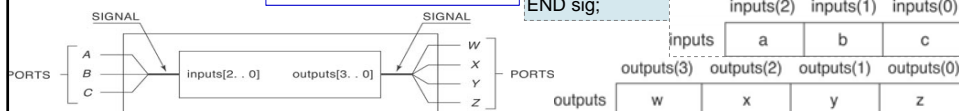
```
x <= outputs(2);
```

```
y <= outputs(1);
```

```
z <= outputs(0);
```

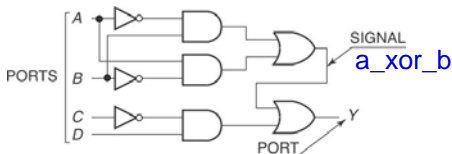
```
END sig;
```

A	B	C	W	X	Y	Z
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	1	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	1	0



## Single-Bit SIGNAL

$$Y = \overline{A}B + A\overline{B} + \overline{C}D$$



```
--Combine single-bit and multiple-bit signals:
```

```
d:IN STD_LOGIC_VECTOR(2 downto 0);
```

```
enable: IN STD_LOGIC;
```

```
...
```

```
SIGNAL inputs: STD_LOGIC_VECTOR (3 downto 0);
```

```
...
```

```
inputs <= enable & d; -- combine
```

```
LIBRARY ieee;
```

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

```
ENTITY signal_ex2 IS
```

```
  PORT(
```

```
    a, b, c, d : IN STD_LOGIC;
```

```
    y : OUT STD_LOGIC);
```

```
END signal_ex2;
```

```
ARCHITECTURE cct of signal_ex2 IS
```

```
-- Declare signal
```

```
SIGNAL a_xor_b : STD_LOGIC;
```

```
BEGIN
```

```
-- Define signal in terms of ports a and b
```

```
a_xor_b <= ((not a) and b) or (a and (not b));
```

```
-- Combine signal with ports c and d
```

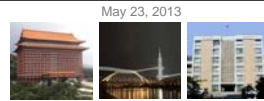
```
y <= a_xor_b or ((not c) and d);
```

```
END cct;
```



May 23, 2013

12



May 23, 2013

13

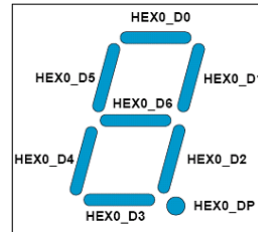
## 7-Segment Control

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

```
ENTITY SevenSegment IS
    PORT (
        sw: IN STD_LOGIC_VECTOR(2 downto 0);
        pb: IN STD_LOGIC;
        hex0: OUT STD_LOGIC_VECTOR(0 to 7));
END SevenSegment;
```

```
ARCHITECTURE a OF SevenSegment IS
BEGIN
    WITH pb & sw SELECT
        hex0 <=
            "00000011" WHEN "0000",
            "10011111" WHEN "0001",
            "00100101" WHEN "0010",
            "00001101" WHEN "0011",
            "10011001" WHEN "0100",
            "01001001" WHEN "0101",
            "01000001" WHEN "0110",
            "00011111" WHEN "0111",
            "11111111" WHEN others;
```

```
END a;
```



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



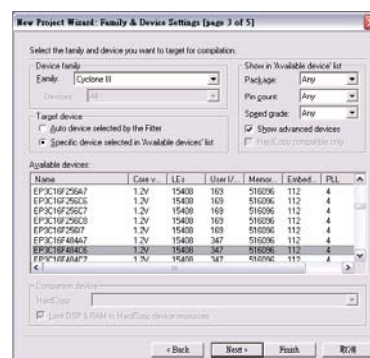
May 23, 2013

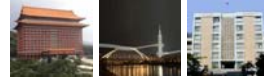
14

## VHDL Design with Quartus II

- Example: When the BUTTON0 is pressed,
  - LEDG0 shows the ANDed result of SW0 and SW1.
  - LEDG1 shows the ORed result of SW0 and SW1.

- Step 1: Start a new project
  - Select **File** → **New Project Wizard**
    - Working directory: Class02
    - Project name: Class02
    - Top-level design entry: Class02
  - Family & Device Settings
    - Device family: Cyclone III
    - Available device: EP3C16F484C6
  - EDA Tool Settings
    - Leave it alone at the moment





May 23, 2013

15

## VHDL Design with Quartus II (Cont.)

- Step 2: Design entry using the text editor

- Select **File** → **New** → **VHDL File (.vhd)**
- Save as “Class02.vhd” (check “**Add file to current project**”)
- Edit “Class02.vhd”

```

ENTITY Class02 IS
    PORT(
        A: IN BIT_VECTOR(1 downto 0);
        C: IN BIT;
        X: OUT BIT;
        Y: OUT BIT);
END Class02;

ARCHITECTURE and_or OF Class02 IS
BEGIN
    X <= A(1) and A(0) and (not C);
    Y <= (A(1) or A(0)) and (not C);
END and_or;

```

- Select “**Start Compilation**” to compile the circuit



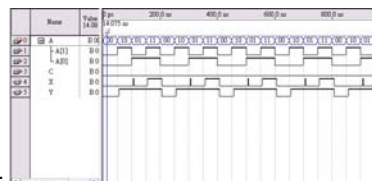
May 23, 2013

16

## VHDL Design with Quartus II (Cont.)

- Step 3: Simulation with Vector Waveform File (.vwf)

- Select **File** → **New** → **Vector Waveform File (.vwf)**
- Save as “Class02.vwf” (check “**Add file to current project**”)
- Select “**Edit** → **Insert** → **Insert Node or Bus** → **Node Finder**” to add input/output pins into the simulation.
- Select “**Edit** → **End Time**” and select “**Edit** → **Grid Size**” to config the simulation period and count period. (e.g., End time 4us, grid size: 50ns)
  - A(0): count value, binary, count every 50ns, multiplied by 1.
  - A(1): count value, binary, count every 50ns, start time: 0ns, multiplied by 2.
  - C: forcing high or forcing low.
- Select “**Start Simulation**” to simulate the circuit.
- Functional simulation
  - Select “**Assignments** → **Settings** → **Simulator Settings**” to set “**Simulation mode**” as **Functional**.
  - Select “**Processing** → **Generate Functional Simulation Netlist**”
  - Select “**Start Simulation**” to simulate the circuit.







May 23, 2013

17

## VHDL Design with Quartus II (Cont.)

- Step 3: Simulation with Vector Waveform File (.vwf)
  - Select “Assignments → Device” to configure the board settings.
    - Set Family as **Cyclone III** and Device as **EP316F484C6**
    - Select “Device and Pin Options”
      - Select and set “Unsigned Pins” as “As input tri-stated” and
      - Select “Configuration” to set configuration scheme as “Active Serial” and configuration device as “EPCS4”
  - Select “Assignments → Pins” to activate the “Pin Planner”.
  - Select “Start Compilation” to compile the circuit with circuit assignment.
  - Select “Tools → Programmer” to download the .soft file to the FPGA board for testing.

Node Name	Direction	Location
A[1]	Input	PIN_H5
A[0]	Input	PIN_J6
C	Input	PIN_H2
X	Output	PIN_J1
Y	Output	PIN_J2

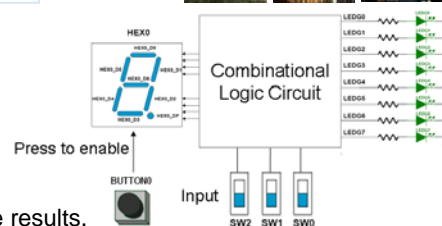


May 23, 2013

18

## Lab 02

- Part 1 - Simulation
  - Use VHDL to design a NAND gate with one output pin **f** and two input pins **a** and **b**. Then use Vector Waveform File (.vwf) to simulate the results.
    - A: count value, binary, simulation period=4us, advanced by 1 every 100ns
    - B: count value, binary, simulation period=4us, advanced by 1 every 200ns
- Part 2: When the BUTTON0 is pressed,
  - LEDG0 shows the ANDed result of SW0 and SW1.
  - LEDG1 shows the ORed result of SW0 and SW1.
- Part 3 - Transferring a Design to a Target FPGA
  - Use three slides (SW2-SW0) as the binary input value. Solve the following problems with VHDL.
    - The corresponding LED (LEDG0-7) is on when selected by the binary input. Other LEDs are off. E.g., 100 (SW2-SW0) lights LEDG4.
    - The first 7-segment LED (HEX0) shows the decimal value of the binary input when the first pushbutton (BUTTON0) is pressed. Otherwise, HEX0 is off. E.g., When BUTTON0 is pressed and the binary input is 101 (SW2-SW0), HEX0 shows 5.



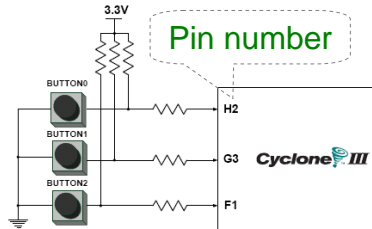


May 23, 2013

19

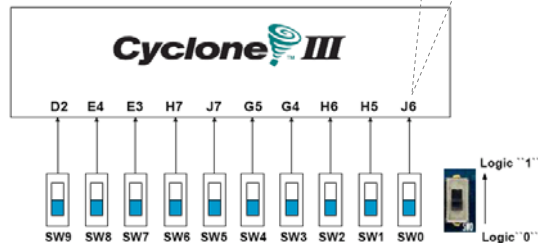
## Pushbutton and Slide Switches

Pin number



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

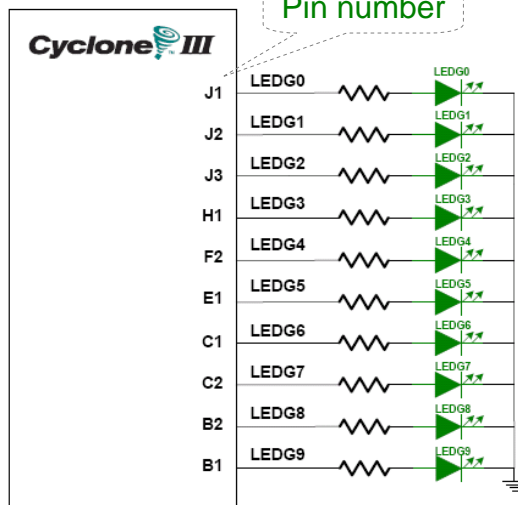


May 23, 2013

20

## LEDs

Pin number



10 LEDs  
Output 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

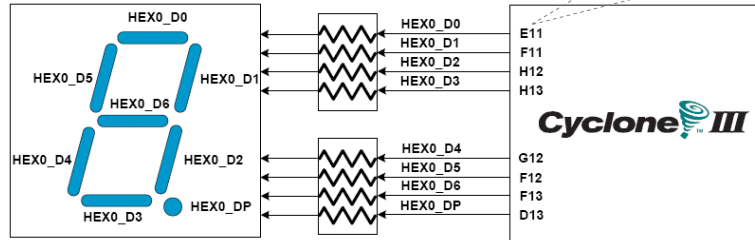


May 23, 2013

21

## 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