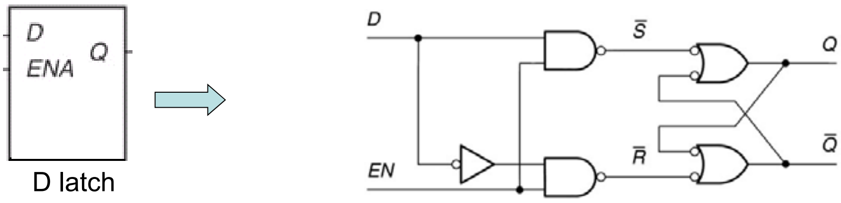
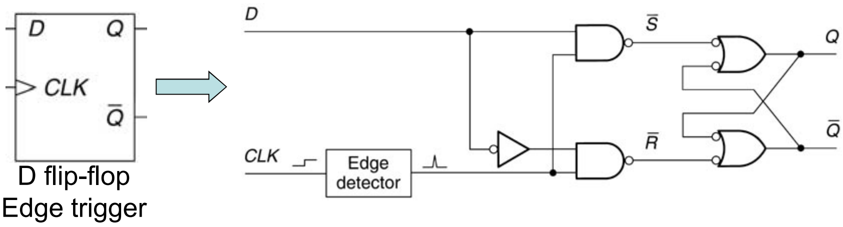
Class 06
Sequential Logic:
Flip-Flop

Differences between Latch and Flip-Flop

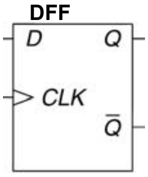


D latch
Level trigger

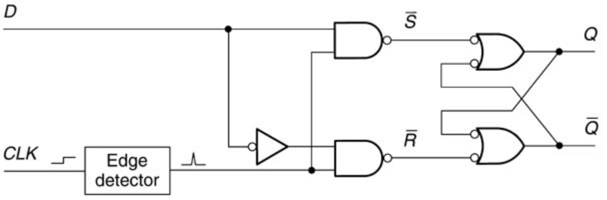


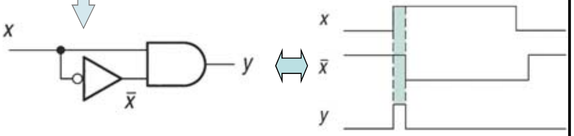
D flip-flop
Edge trigger

Function Table of D Flip-Flop



D flip-flop
Edge trigger

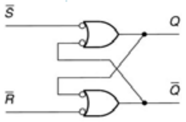




CLK	D	Q_{t+1}	nQ_{t+1}	Function
↑	0	0	1	Reset
↑	1	1	0	Set
0	X	Q_t	nQ_t	Inhibited
1	X	Q_t	nQ_t	Inhibited
↓	X	Q_t	nQ_t	Inhibited

Function table of D flip-flop

JK Flip-Flop

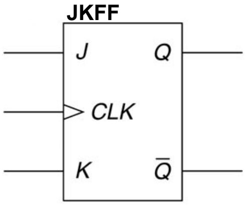


nS	nR	Q_{t+1}	nQ_{t+1}	Function
0	0	1	1	Forbidden
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q_t	nQ_t	No Change

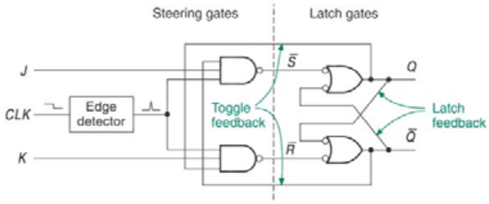
SR NAND Latch

EN	S	R	Q_{t+1}	nQ_{t+1}	Function
1	0	0	Q_t	nQ_t	No Change
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	1	1	Forbidden
0	X	X	Q_t	nQ_t	Inhibited

Gated SR NAND Latch



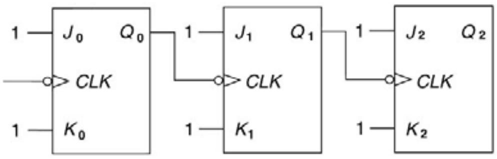
JK flip-flop

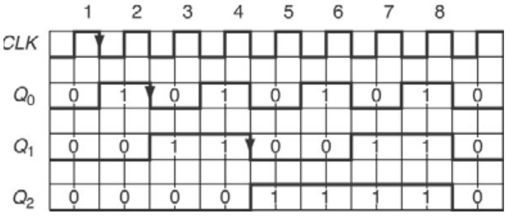


CLK	J	K	Q_{t+1}	nQ_{t+1}	Function
↑	0	0	Q_t	nQ_t	No Change
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	**1**	**1**	**nQ_t**	**Q_t**	**Toggle**
0	X	X	Q_t	nQ_t	Inhibited
1	X	X	Q_t	nQ_t	Inhibited
↓	X	X	Q_t	nQ_t	Inhibited

Function table of JK flip-flop

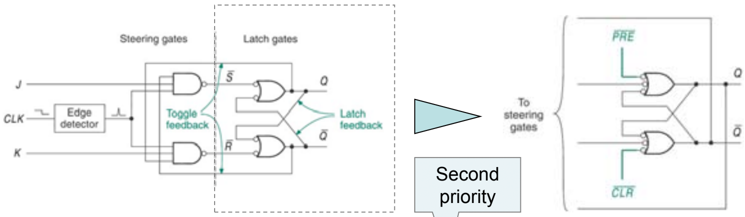
Frequency Divider or Counter



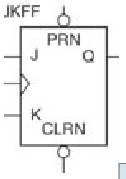


Clock Pulse	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

JK Flip-Flop with Asynchronous Inputs



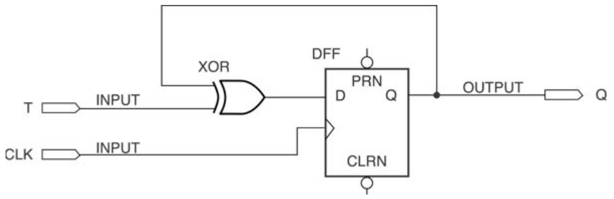
Second priority



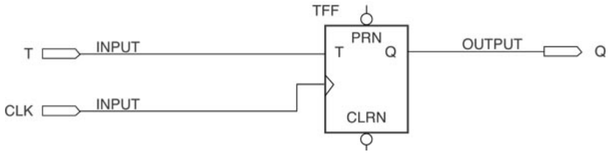
First priority

	nPRE	nCLR	CLK	J	K	Q_{t+1}	nQ_{t+1}	Function
Sync	1	1	↑	0	0	Q_t	nQ_t	No Change
	1	1	↑	0	1	0	1	Reset
	1	1	↑	1	0	1	0	Set
	1	1	↑	1	1	nQ_t	Q_t	Toggle
Async	0	1	X	X	X	1	0	Preset
	1	0	X	X	X	0	1	Clear
	0	0	X	X	X	1	1	Forbidden
	1	1	0	X	X	Q_t	nQ_t	Inhibited
	1	1	1	X	X	Q_t	nQ_t	Inhibited
	1	1	↓	X	X	Q_t	nQ_t	Inhibited

T Flip-Flop



D flip-flop with an XOR at the input



T flip-flop: the output Q toggles on each clock pulse when T is high

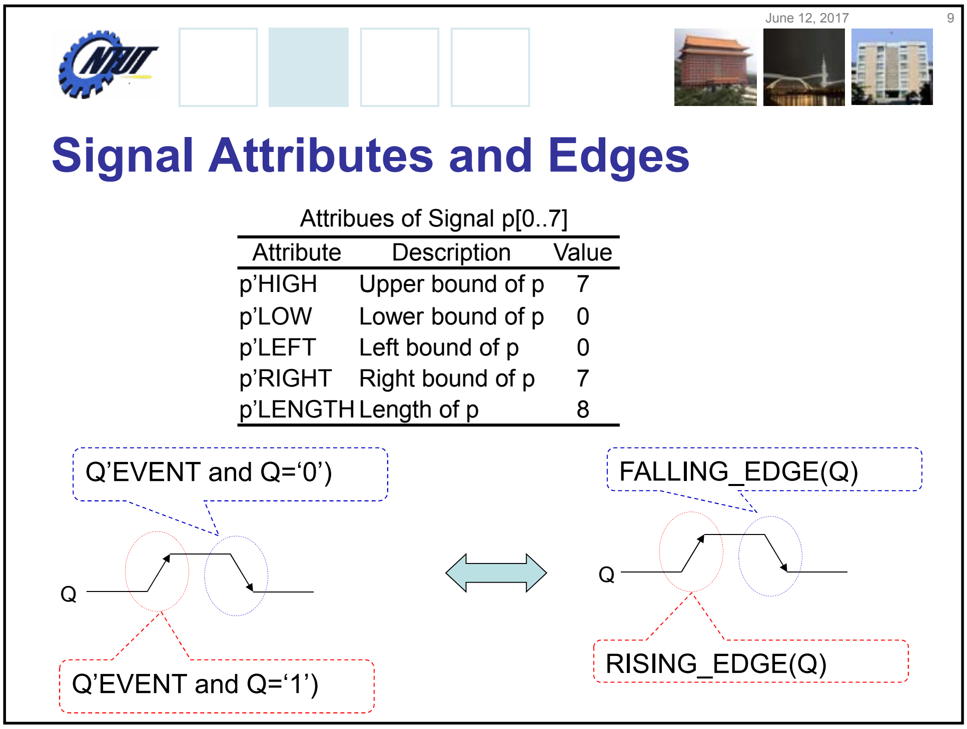
CLK	T	Q_{t+1}	Function
↑	0	Q_t	No change
↑	1	nQ_t	Toggle
0	X	Q_t	Inhibited
1	X	Q_t	Inhibited
↓	X	Q_t	Inhibited

Function table of T flip-flop

Integer vs. Unsigned STD_LOGIC

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY compare4 IS
  PORT(
              a, b : IN INTEGER RANGE 0 TO 15;
              agtb, aeqb, altb : OUT       STD_LOGIC);
END compare4;
ARCHITECTURE a OF compare4 IS
  SIGNAL compare : STD_LOGIC_VECTOR(2 downto 0);
BEGIN
  PROCESS (a,b)
  BEGIN
    IF a<b THEN
        compare <= "110";
    ELSIF a=b THEN
        compare <= "101";
    ELSIF a>b THEN
        compare <= "011";
    ELSE
        compare <= "111";
    END IF;
    agtb <=     compare(2);
    aeqb <=     compare(1);
    altb <=     compare(0);
  END PROCESS;
END a;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY compare4 IS
  PORT(
              a, b : IN STD_LOGIC_VECTOR (3 downto 0);
              agtb, aeqb, altb : OUT       STD_LOGIC);
END compare4;
ARCHITECTURE a OF compare4 IS
  SIGNAL compare : STD_LOGIC_VECTOR(2 downto 0);
BEGIN
  PROCESS (a,b)
  BEGIN
    IF a<b THEN
        compare <= "110";
    ELSIF a=b THEN
        compare <= "101";
    ELSIF a>b THEN
        compare <= "011";
    ELSE
        compare <= "111";
    END IF;
    agtb <=     compare(2);
    aeqb <=     compare(1);
    altb <=     compare(0);
  END PROCESS;
END a;
```



Signal Attributes and Edges

Attribues of Signal p[0..7]

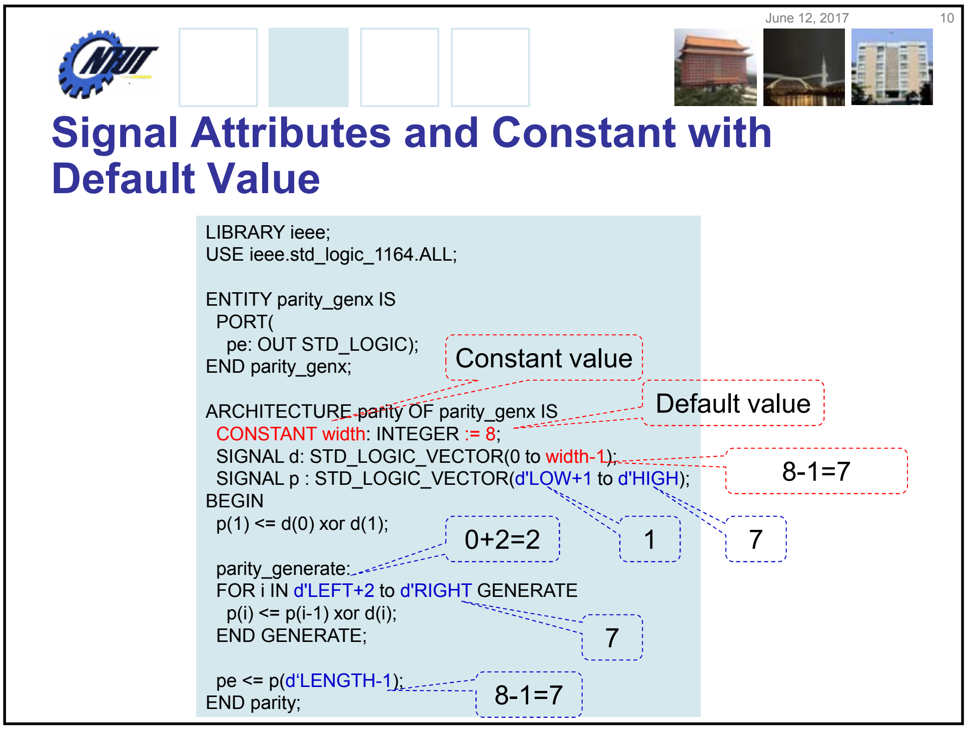
Attribute	Description	Value
p’HIGH	Upper bound of p	7
p’LOW	Lower bound of p	0
p’LEFT	Left bound of p	0
p’RIGHT	Right bound of p	7
p’LENGTH	Length of p	8

Q’EVENT and Q=‘0’)

Q’EVENT and Q=‘1’)

FALLING_EDGE(Q)

RISING_EDGE(Q)



Signal Attributes and Constant with Default Value

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

ENTITY parity_genx IS
  PORT(
    pe: OUT STD_LOGIC);
END parity_genx;

ARCHITECTURE parity OF parity_genx IS
  CONSTANT width: INTEGER := 8;
  SIGNAL d: STD_LOGIC_VECTOR(0 to width-1);
  SIGNAL p : STD_LOGIC_VECTOR(d'LOW+1 to d'HIGH);
BEGIN
  p(1) <= d(0) xor d(1);

  parity_generate:
  FOR i IN d'LEFT+2 to d'RIGHT GENERATE
    p(i) <= p(i-1) xor d(i);
  END GENERATE;

  pe <= p(d‘LENGTH-1);
END parity;
```

Constant value

Default value

8-1=7

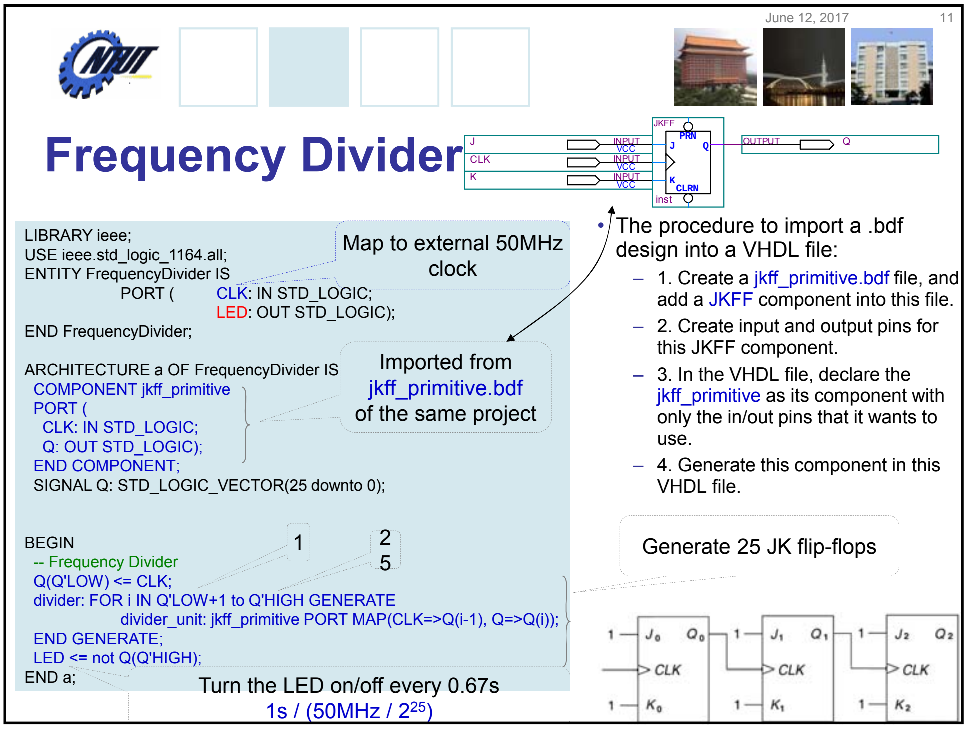
0+2=2

1

7

7

8-1=7



Frequency Divider

J
CLK
K
INPUT
VCC
INPUT
VCC
INPUT
VCC
JKFF
PRN
J
Q
K
CLRN
inst
OUTPUT
Q

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY FrequencyDivider IS
                PORT (        CLK: IN STD_LOGIC;
                              LED: OUT STD_LOGIC);
END FrequencyDivider;

ARCHITECTURE a OF FrequencyDivider IS
  COMPONENT jkff_primitive
  PORT (
    CLK: IN STD_LOGIC;
    Q: OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL Q: STD_LOGIC_VECTOR(25 downto 0);


BEGIN
  -- Frequency Divider
  Q(Q'LOW) <= CLK;
  divider: FOR i IN Q'LOW+1 to Q'HIGH GENERATE
                divider_unit: jkff_primitive PORT MAP(CLK=>Q(i-1), Q=>Q(i));
  END GENERATE;
  LED <= not Q(Q'HIGH);
END a;
```

Map to external 50MHz clock

Imported from jkff_primitive.bdf of the same project

1

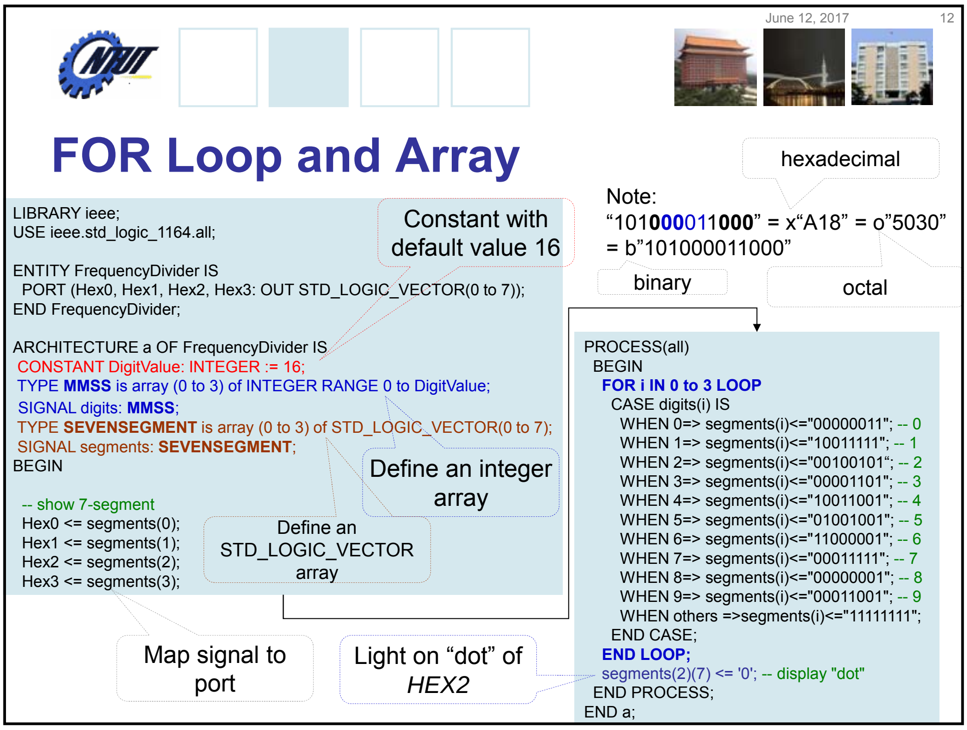
2
5

Turn the LED on/off every 0.67s
1s / (50MHz / 2^{25})

- The procedure to import a .bdf design into a VHDL file:
  - 1. Create a jkff_primitive.bdf file, and add a JKFF component into this file.
  - 2. Create input and output pins for this JKFF component.
  - 3. In the VHDL file, declare the jkff_primitive as its component with only the in/out pins that it wants to use.
  - 4. Generate this component in this VHDL file.

Generate 25 JK flip-flops

1
J_0
Q_0
CLK
1
K_0
1
J_1
Q_1
CLK
1
K_1
1
J_2
Q_2
CLK
1
K_2



FOR Loop and Array

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

ENTITY FrequencyDivider IS
  PORT (Hex0, Hex1, Hex2, Hex3: OUT STD_LOGIC_VECTOR(0 to 7));
END FrequencyDivider;

ARCHITECTURE a OF FrequencyDivider IS
 CONSTANT DigitValue: INTEGER := 16;
 TYPE MMSS is array (0 to 3) of INTEGER RANGE 0 to DigitValue;
 SIGNAL digits: MMSS;
 TYPE SEVENSEGMENT is array (0 to 3) of STD_LOGIC_VECTOR(0 to 7);
 SIGNAL segments: SEVENSEGMENT;
BEGIN

  -- show 7-segment
  Hex0 <= segments(0);
  Hex1 <= segments(1);
  Hex2 <= segments(2);
  Hex3 <= segments(3);
```

Constant with default value 16

Define an integer array

Define an STD_LOGIC_VECTOR array

Map signal to port

Light on "dot" of *HEX2*

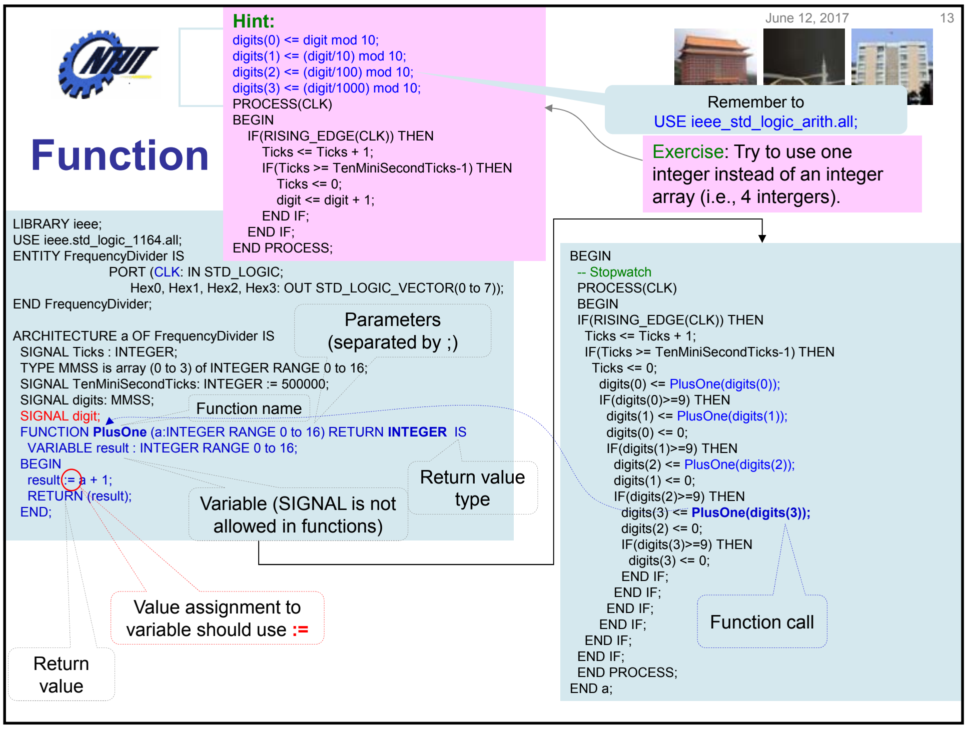
hexadecimal

Note:
"101000011000" = x"A18" = o"5030"
= b"101000011000"

binary

octal

```
PROCESS(all)
  BEGIN
    FOR i IN 0 to 3 LOOP
      CASE digits(i) IS
        WHEN 0=> segments(i)<="00000011"; -- 0
        WHEN 1=> segments(i)<="10011111"; -- 1
        WHEN 2=> segments(i)<="00100101“; -- 2
        WHEN 3=> segments(i)<="00001101"; -- 3
        WHEN 4=> segments(i)<="10011001"; -- 4
        WHEN 5=> segments(i)<="01001001"; -- 5
        WHEN 6=> segments(i)<="11000001"; -- 6
        WHEN 7=> segments(i)<="00011111"; -- 7
        WHEN 8=> segments(i)<="00000001"; -- 8
        WHEN 9=> segments(i)<="00011001"; -- 9
        WHEN others =>segments(i)<="11111111";
      END CASE;
    END LOOP;
    segments(2)(7) <= '0'; -- display "dot"
  END PROCESS;
END a;
```



Function

Hint:

```
digits(0) <= digit mod 10;
digits(1) <= (digit/10) mod 10;
digits(2) <= (digit/100) mod 10;
digits(3) <= (digit/1000) mod 10;
PROCESS(CLK)
BEGIN
   IF(RISING_EDGE(CLK)) THEN
      Ticks <= Ticks + 1;
      IF(Ticks >= TenMiniSecondTicks-1) THEN
         Ticks <= 0;
         digit <= digit + 1;
      END IF;
   END IF;
END PROCESS;
```

Remember to USE ieee_std_logic_arith.all;

Exercise: Try to use one integer instead of an integer array (i.e., 4 intergers).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY FrequencyDivider IS
                    PORT (CLK: IN STD_LOGIC;
                          Hex0, Hex1, Hex2, Hex3: OUT STD_LOGIC_VECTOR(0 to 7));
END FrequencyDivider;

ARCHITECTURE a OF FrequencyDivider IS
  SIGNAL Ticks : INTEGER;
  TYPE MMSS is array (0 to 3) of INTEGER RANGE 0 to 16;
  SIGNAL TenMiniSecondTicks: INTEGER := 500000;
  SIGNAL digits: MMSS;
  SIGNAL digit;
  FUNCTION PlusOne (a:INTEGER RANGE 0 to 16) RETURN INTEGER  IS
    VARIABLE result : INTEGER RANGE 0 to 16;
  BEGIN
    result := a + 1;
    RETURN (result);
  END;
```

```
BEGIN
  -- Stopwatch
  PROCESS(CLK)
  BEGIN
  IF(RISING_EDGE(CLK)) THEN
    Ticks <= Ticks + 1;
    IF(Ticks >= TenMiniSecondTicks-1) THEN
      Ticks <= 0;
        digits(0) <= PlusOne(digits(0));
        IF(digits(0)>=9) THEN
          digits(1) <= PlusOne(digits(1));
          digits(0) <= 0;
          IF(digits(1)>=9) THEN
            digits(2) <= PlusOne(digits(2));
            digits(1) <= 0;
            IF(digits(2)>=9) THEN
              digits(3) <= PlusOne(digits(3));
              digits(2) <= 0;
              IF(digits(3)>=9) THEN
                digits(3) <= 0;
              END IF;
            END IF;
          END IF;
        END IF;
    END IF;
  END IF;
  END PROCESS;
END a;
```

Parameters (separated by ;)

Function name

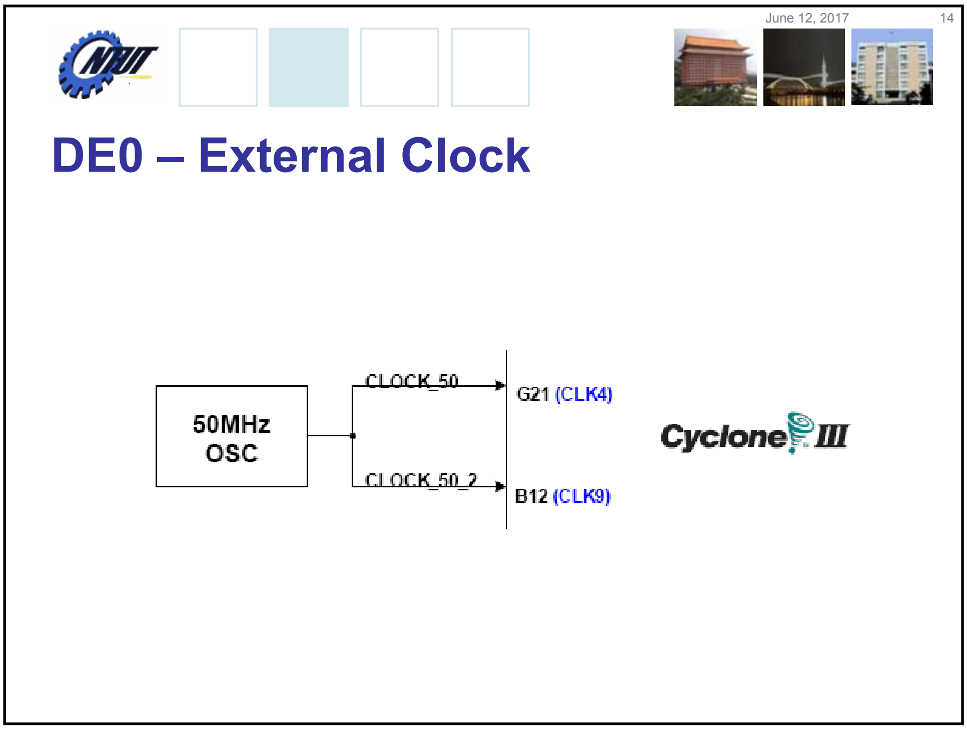
Return value type

Variable (SIGNAL is not allowed in functions)

Value assignment to variable should use :=

Return value

Function call

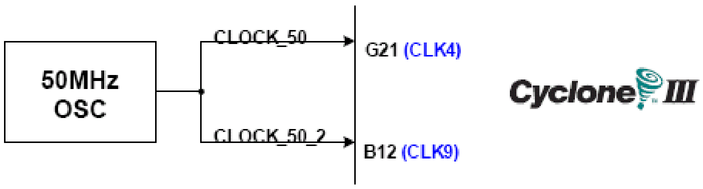


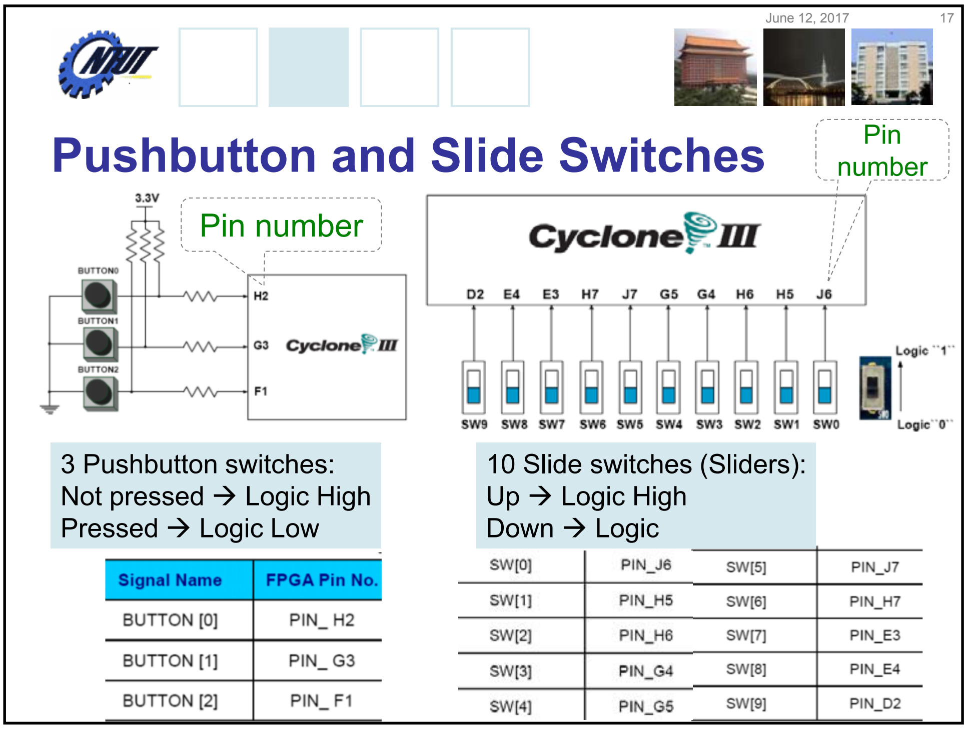
DE0 – External Clock

Lab 06

- Part 1: Frequency Divider with JK flip-flops or T flip-flops
  - Design a flashing light array with the 50MHz clock
    - LEDG[0..9] is toggled every $2^{[25..34]}$ clock ticks, respectively.
    - E.g., LEDG0 is toggled with 1.49Hz, LEDG1 is toggled with 0.745Hz, and so on.
- Part 2: Design a clock
  - Design a clock with four digits.
    - HEX[3210] represents MM.SS (MM: minutes, SS: seconds), where the “dot” between MM and SS should be always on.
    - Set the initial time to 34.56.

DE0 – External Clock





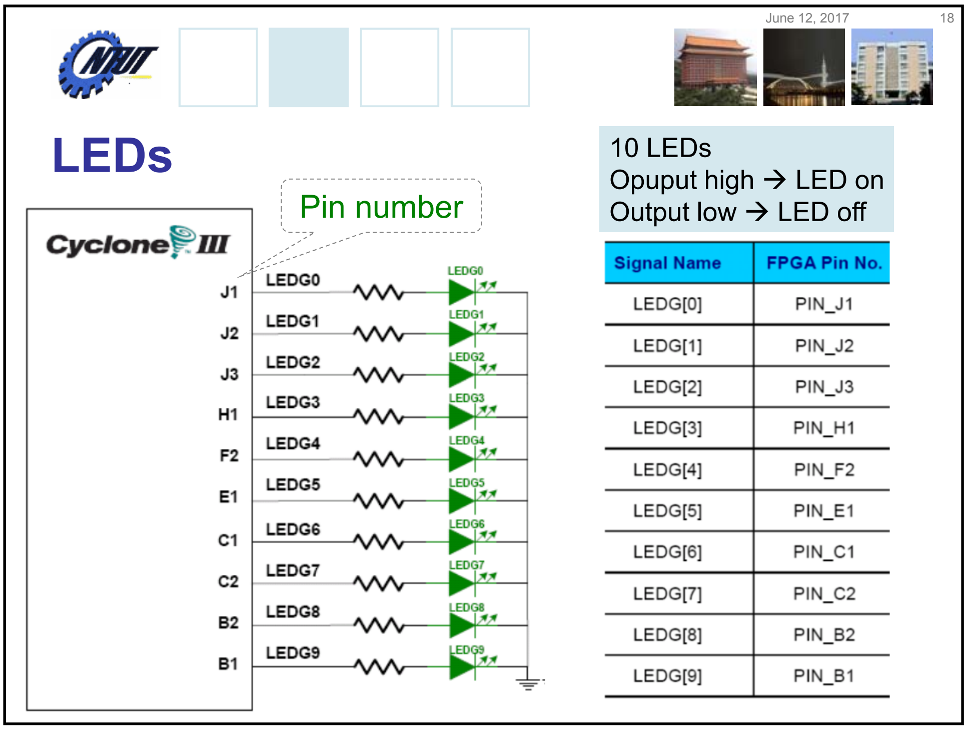
Pushbutton and Slide Switches

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

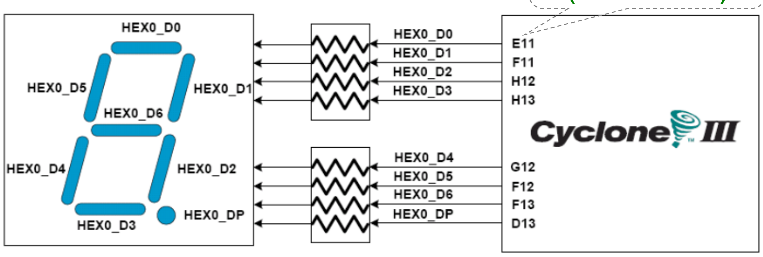
Cyclone III
Pin number
J1 J2 J3 H1 F2 E1 C1 C2 B2 B1
LEDG0 LEDG1 LEDG2 LEDG3 LEDG4 LEDG5 LEDG6 LEDG7 LEDG8 LEDG9

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)



HEX0_D0
HEX0_D1
HEX0_D2
HEX0_D3
HEX0_D4
HEX0_D5
HEX0_D6
HEX0_DP

HEX0_D0 E11
HEX0_D1 F11
HEX0_D2 H12
HEX0_D3 H13
HEX0_D4 G12
HEX0_D5 F12
HEX0_D6 F13
HEX0_DP D13

Cyclone III

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