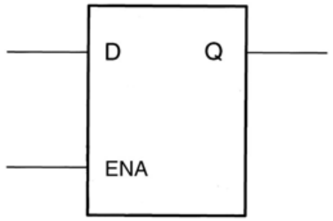
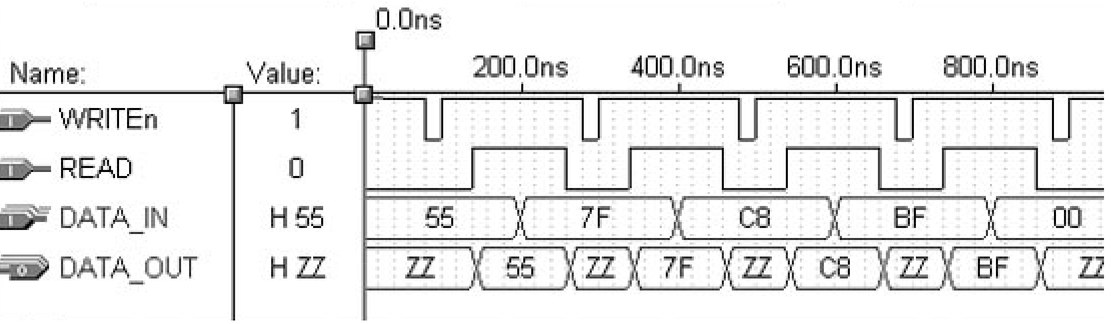
Class 13 Memory

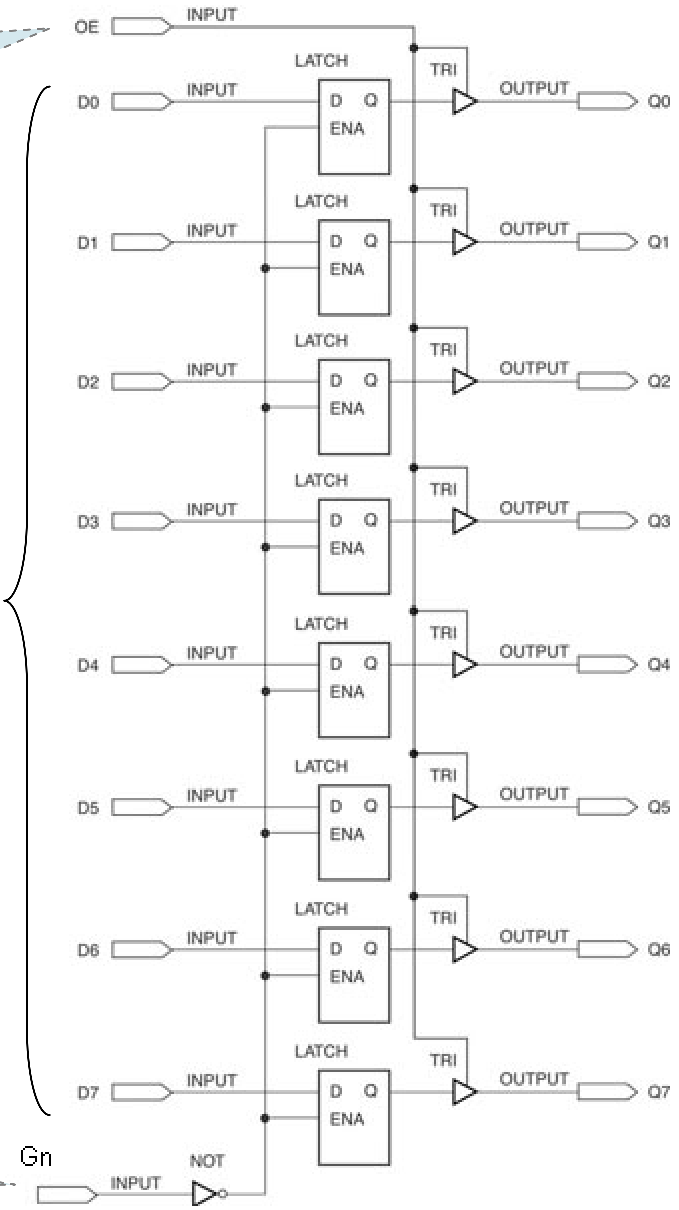
Memory

- A memory is a digital device or circuit that can store one or more bits of data.
- The simplest memory device is a D-type latch or a D flip-flop.

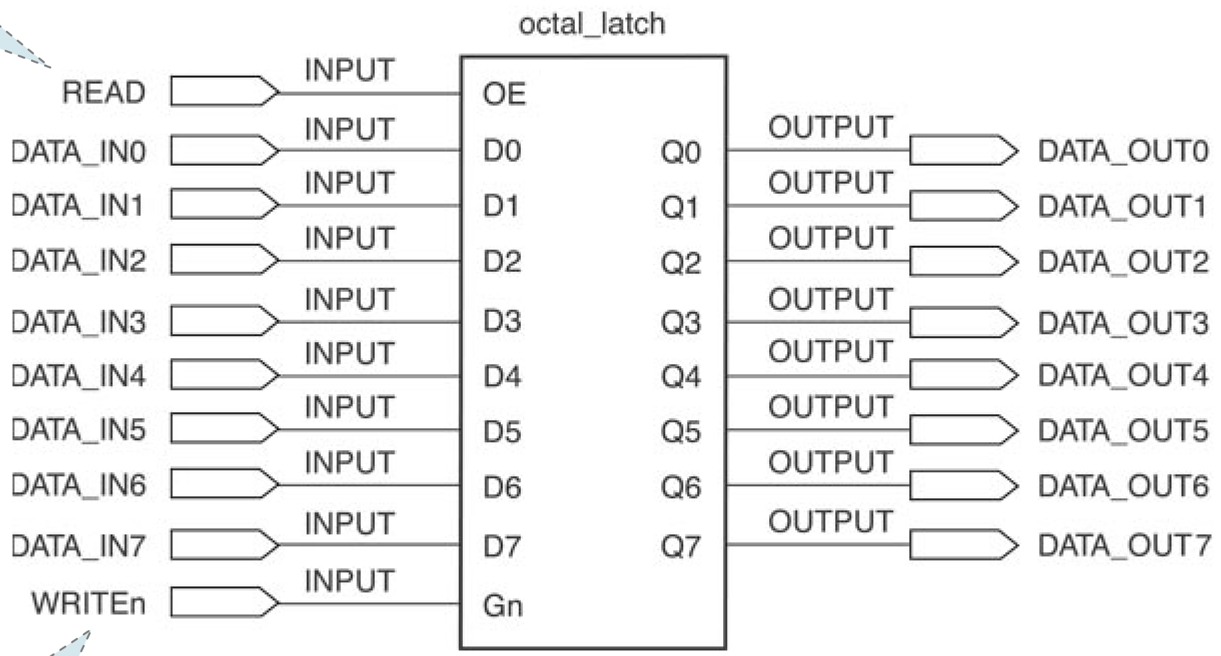


D-type latch





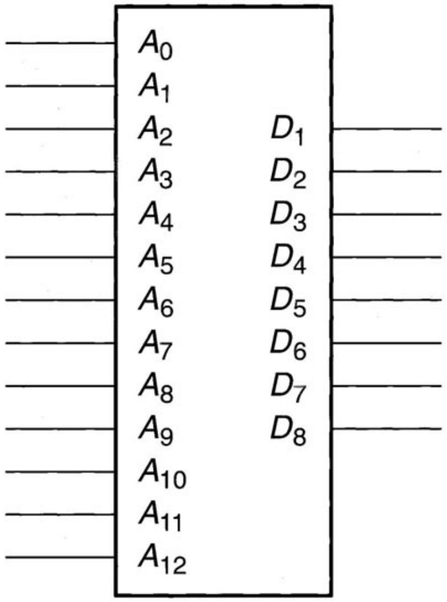
8-Bit Latch



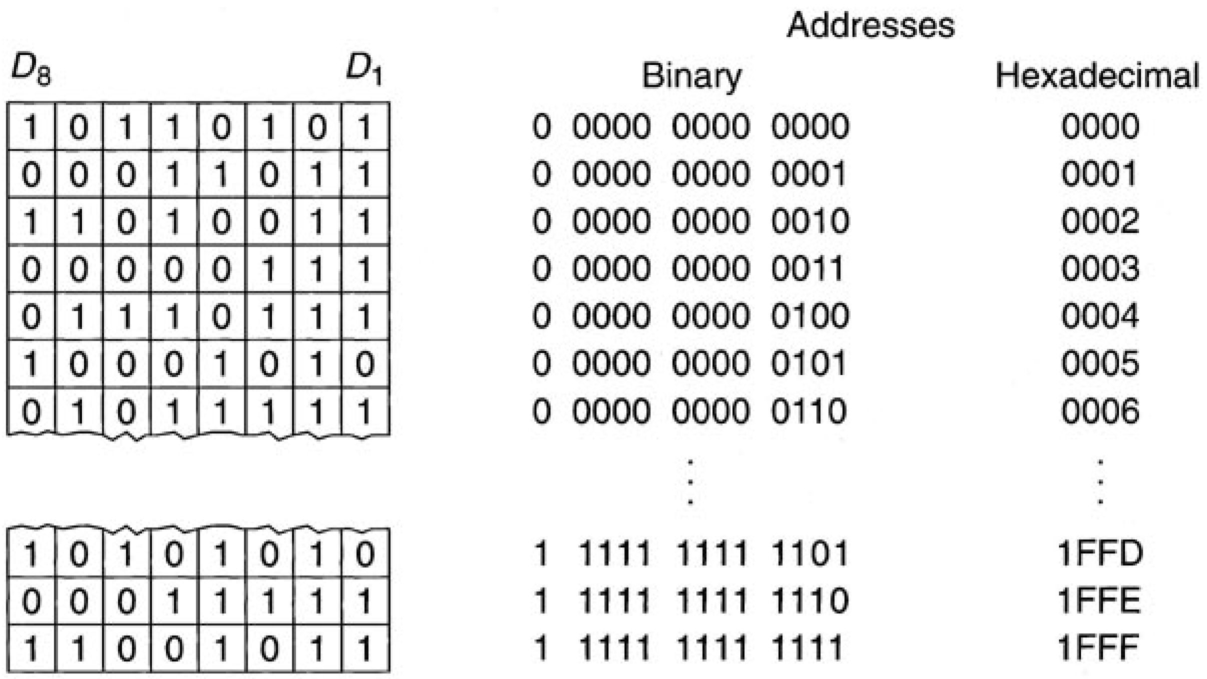
Output enable (Read)

Input gate enable (Write)

Address and Data Lines



a. Address and data lines



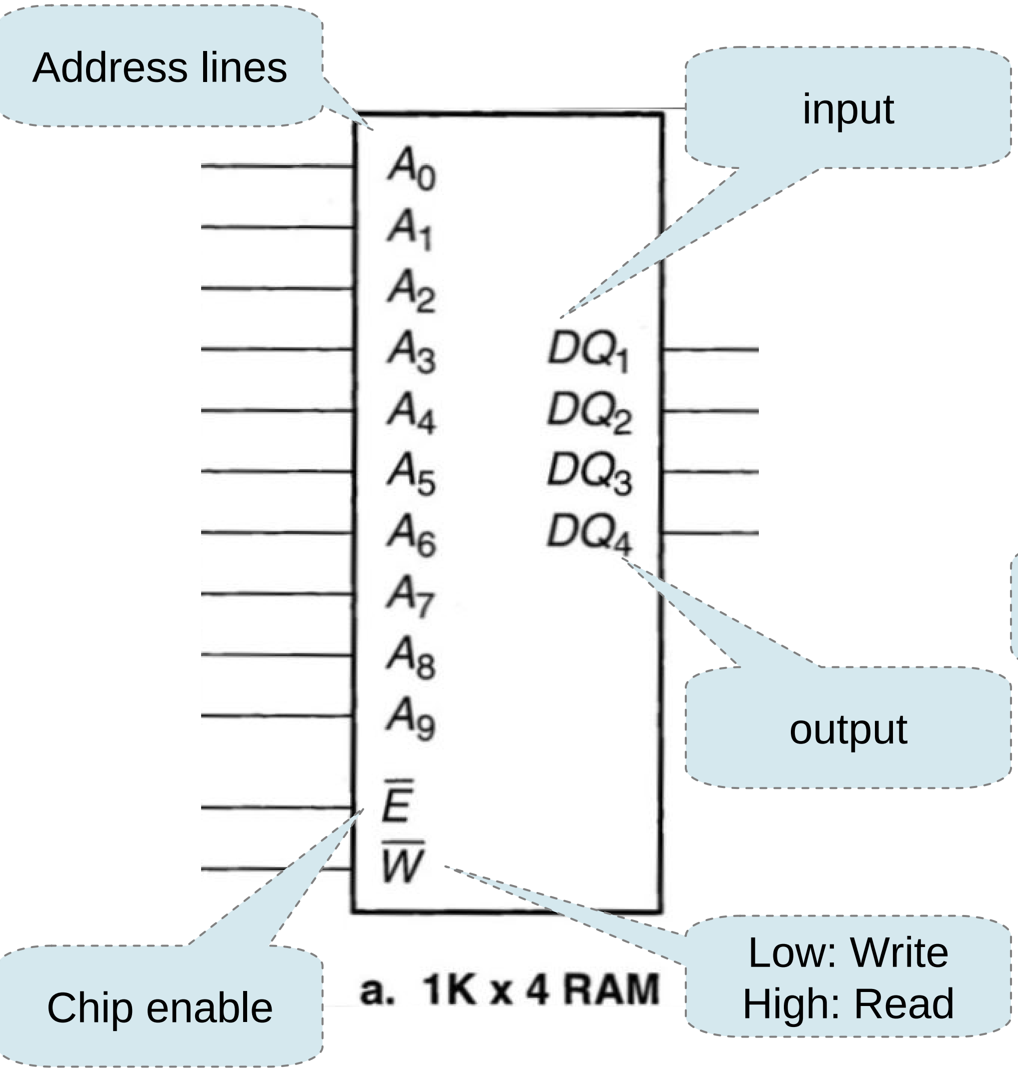
D_8							D_1	Addresses: Binary	Addresses: Hexadecimal
1	0	1	1	0	1	0	1	0 0000 0000 0000	0000
0	0	0	1	1	0	1	1	0 0000 0000 0001	0001
1	1	0	1	0	0	1	1	0 0000 0000 0010	0002
0	0	0	0	0	1	1	1	0 0000 0000 0011	0003
0	1	1	1	0	1	1	1	0 0000 0000 0100	0004
1	0	0	0	1	0	1	0	0 0000 0000 0101	0005
0	1	0	1	1	1	1	1	0 0000 0000 0110	0006
								⋮	⋮
1	0	1	0	1	0	1	0	1 1111 1111 1101	1FFD
0	0	0	1	1	1	1	1	1 1111 1111 1110	1FFE
1	1	0	0	1	0	1	1	1 1111 1111 1111	1FFF

b. Contents (data) and location (address)

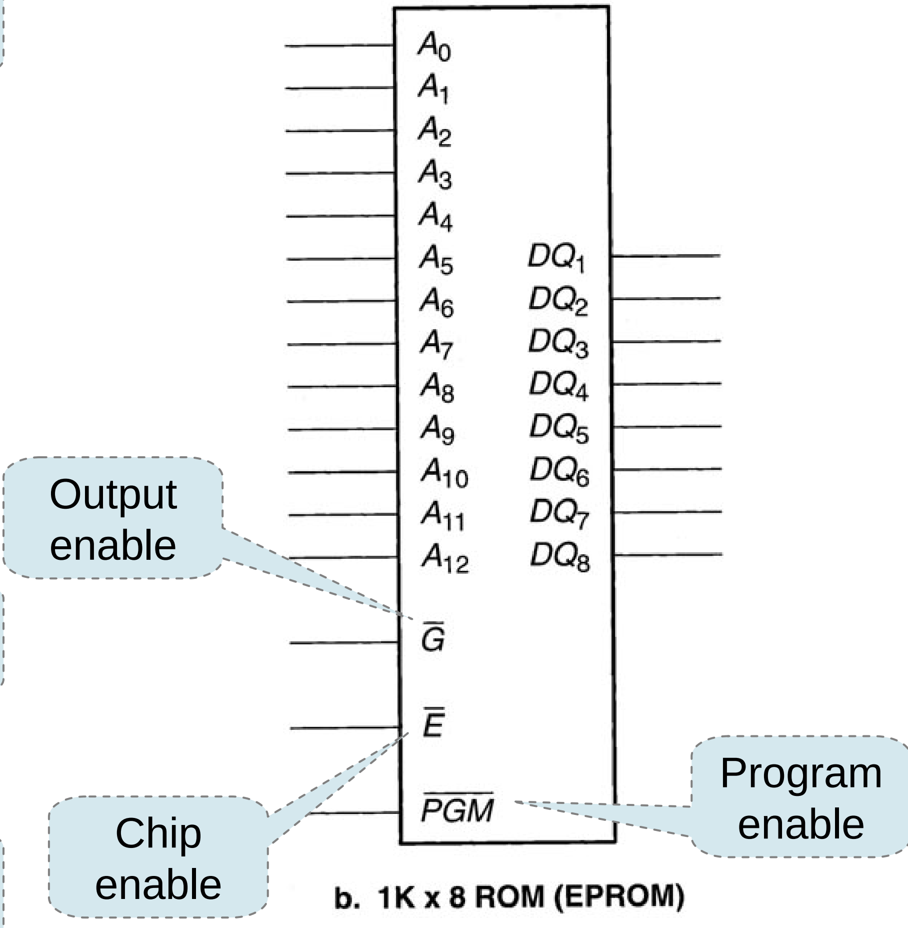
8K x 8 Memory
Address: 2^{13} = 8192 WORD
Data: 8 bits (= 1 WORD)

addr(0) = “10110101”, when $A_{12}…A_0$ = 0000000000000 = 0x000
addr(1) = “00011011”, when $A_{12}…A_0$ = 0000000000001 = 0x001

Address, Data, and Control Signals

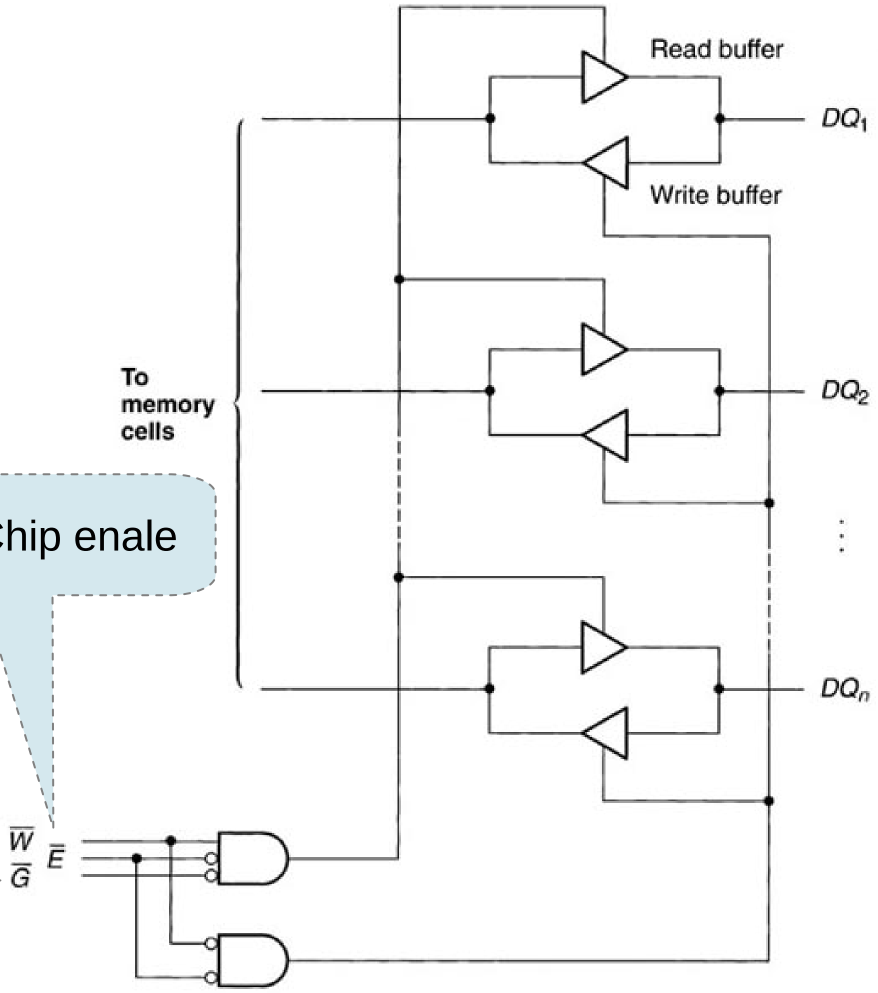

Address lines
input
output
Chip enable
Low: Write
High: Read
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9
DQ_1 DQ_2 DQ_3 DQ_4
$\overline{E}$
$\overline{W}$

a. 1K x 4 RAM


Output enable
Chip enable
Program enable
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12}
DQ_1 DQ_2 DQ_3 DQ_4 DQ_5 DQ_6 DQ_7 DQ_8
$\overline{G}$
$\overline{E}$
$\overline{PGM}$

b. 1K x 8 ROM (EPROM)

Memory Control Signals



Chip enale

Select the read or write function = R/Wn
Low; write enable
High: read enable

Output enable

4x8bit Memory - Behavioral Design

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

ENTITY Memory IS
   PORT(
      SlideSwitch: IN STD_LOGIC_VECTOR(9 downto 0);
      PushButton: IN STD_LOGIC_VECTOR(2 downto 0);
      Output: OUT STD_LOGIC_VECTOR(7 downto 0));
END Memory;

ARCHITECTURE memory OF Memory IS
   CONSTANT MAX_ADDR: INTEGER := 4;
   CONSTANT BUS_WIDTH: INTEGER := 8;
   SIGNAL addr: STD_LOGIC_VECTOR(1 downto 0);
   SIGNAL data_in: STD_LOGIC_VECTOR(BUS_WIDTH-1 downto 0);
   SIGNAL CEn, Wn, OEn: STD_LOGIC;
   TYPE MEMORY_ARRAY IS array(0 to MAX_ADDR-1) OF
         STD_LOGIC_VECTOR(BUS_WIDTH-1 downto 0);
   SIGNAL sram: MEMORY_ARRAY;
BEGIN
   -- port mapping
   addr <= SlideSwitch(9 downto 8);
   data_in <= SlideSwitch(7 downto 0);
   CEn <= PushButton(0);
   Wn <= PushButton(1);
   OEn <= PushButton(2);
```

Declare an 4x8bit RAM

```
PROCESS(ALL)
   BEGIN
      -- Memory
      IF(CEn = '0') THEN -- chip enabled
         IF(Wn = '0') THEN – Write enable
            sram(conv_integer(addr)) <= data_in;
            Output <= (others =>'1');
         ELSIF(OEn='0') THEN – Read enable and output enable
            Output <= sram(conv_integer(addr));
         ELSE
            Output <= (others =>'1');
         END IF;
      ELSE
         Output <= (others =>'1');
      END IF;
   END PROCESS;
END memory;
```

Write input data to the select address

During wirte operation, output is disabled

Read data from the selected address to the output port

Convert std_logic_vector to integer (need to include **USE ieee.std_logic_unsigned.all; USE ieee.std_logic_arith.all;**)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;
…
SIGNAL a: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL b: INTEGER RANGE 0 to 255;
a <= conv_std_logic_vector(b, 8);
b <= conv_integer(a);
```

Convert b into a 8-bit std_logic_vector

Convert a into an integer.

Counter Array with Fast Increments

```
ARCHITECTURE memory OF Memory IS
   CONSTANT TicksPerMilliSecond: INTEGER := 50000;
   CONSTANT DebounceTime: INTEGER := TicksPerMilliSecond * 5;
   CONSTANT FastCountDelay: INTEGER := TicksPerMilliSecond * 1000;
   CONSTANT FastCountInterval: INTEGER := TicksPerMilliSecond * 100;
   CONSTANT MAX_COUNTER: INTEGER := 16;
   TYPE COUNTER_ARRAY IS array(0 to MAX_COUNTER-1) OF INTEGER RANGE 0 to 99;
   SIGNAL PressedTime: NATURAL := 0;
   SIGNAL cnt: COUNTER_ARRAY;
   SIGNAL cnt_addr: STD_LOGIC_VECTOR(3 downto 0);
BEGIN
   PROCESS(ALL)
   BEGIN
      -- counters
      IF(clk'EVENT and clk = '1') THEN
         IF(CNTn='0') THEN
            IF(PressedTime < DebounceTime) THEN
               PressedTime <= PressedTime + 1; -- Wait for debounce time
            ELSIF(PressedTime = DebounceTime) THEN
               cnt(conv_integer(cnt_addr)) <= cnt(conv_integer(cnt_addr)) + 1;
               PressedTime <= FastCountInterval; -- Debounce time is reached
            ELSE -- fast and accumulated counting
               PressedTime <= PressedTime + 1; -- Keep accumulating the pressed time
               IF(PressedTime = FastCountDelay+FastCountInterval) THEN -- Reach the fast count point
                  cnt(conv_integer(cnt_addr)) <=cnt(conv_integer(cnt_addr)) + 1;
                  PressedTime <= FastCountDelay; -- Reset the pressed time to wait for the next delay
               END IF;
            END IF;
         ELSE
            PressedTime <= 0;
         END IF;
         -- reset counter
         IF(cnt(conv_integer(cnt_addr)) >99) THEN
                        cnt(conv_integer(cnt_addr)) <= 0;
         END IF;
      END IF;
   END PROCESS;
END memory;
```

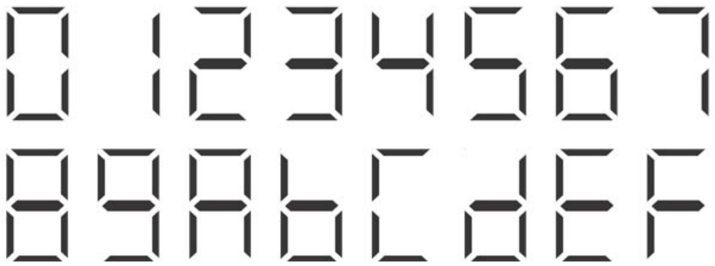
Define a data structure with 16 integers

Declare an array with 16 counters

Wait until the debouncing time is reached

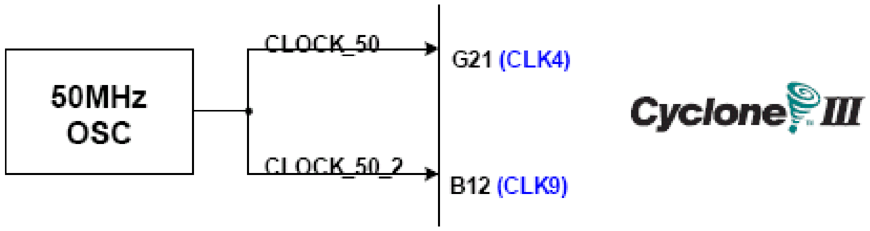
Wait until the fast increment time is reached

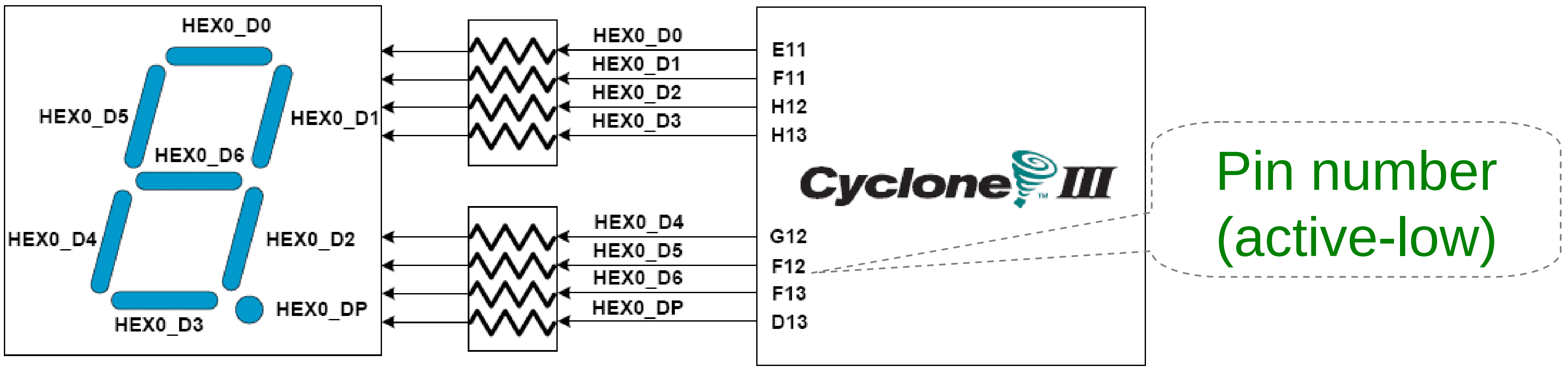
Lab 13



- Design a 4x8bit memory.
  - addr[1..0] is mapped to SW[9..8] (SlideSwitch);
  - data_in[7..0] is mapped to SW[7..0];
  - DQ[7..0] is converted into two-hexadecimal digits shown in Hex[1..0] (7-segments).
  - En (chip enable) is mapped to PushButton[0]
  - R/Wn (read/write selection) is mapped to PushButton[1];
  - OEn (or Gn: output enable) is mapped to PushButton[2];
- Design an array of 16 counters with fast increment support
  - Each counter is a two-digit counter that counts from 0 to 99.
  - Each counter is selected by the decoded value of SW[3..0].
  - When a counter is selected, its content is shown in Hex[3..2].
  - When PushButton[1] is pushed, the selected counter is advanced by one.
  - If PushButton[1] is pushed for more than one second, the selected counter is advanced by one every 100ms.
- Report:
  - Write down what you have learned from this lab. (實驗心得)

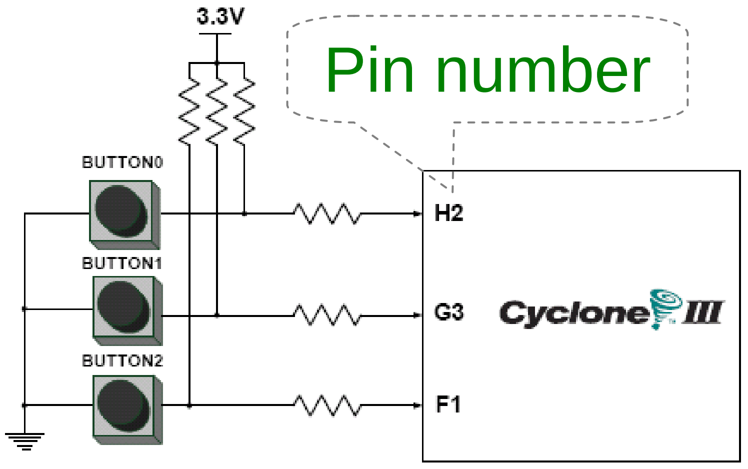
7-Segment Displays & DE0 – External Clock

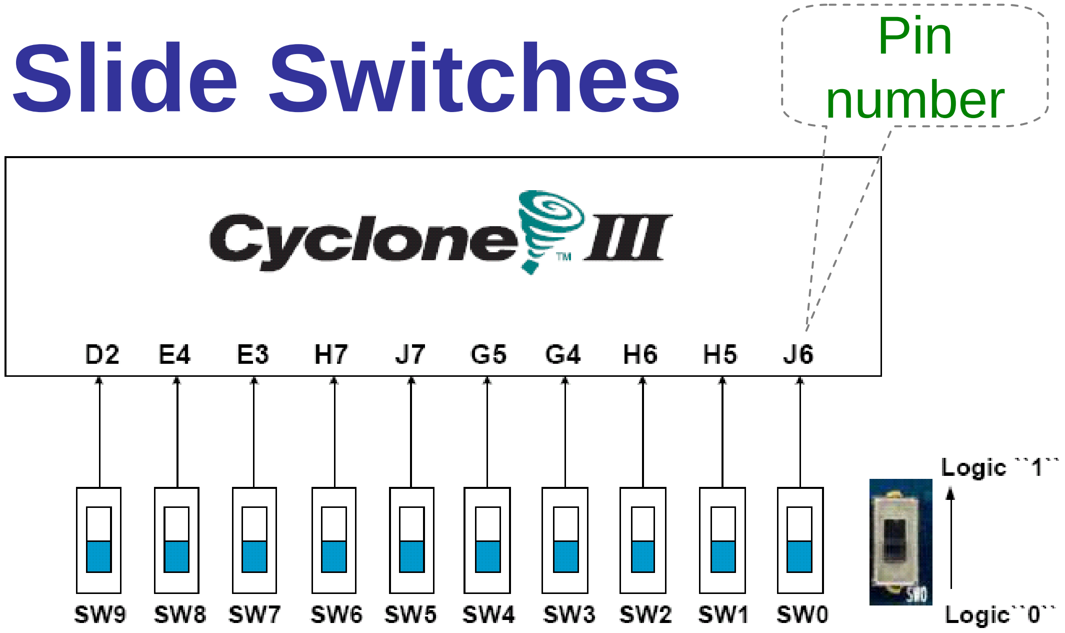




Signal Name	FPGA Pin No.
HEX0_D[0]	PIN_E11
HEX0_D[1]	PIN_F11
HEX0_D[2]	PIN_H12
HEX0_D[3]	PIN_H13
HEX0_D[4]	PIN_G12
HEX0_D[5]	PIN_F12
HEX0_D[6]	PIN_F13
HEX0_DP	PIN_D13
HEX1_D[0]	PIN_A13
HEX1_D[1]	PIN_B13
HEX1_D[2]	PIN_C13
HEX1_D[3]	PIN_A14
HEX1_D[4]	PIN_B14
HEX1_D[5]	PIN_E14
HEX1_D[6]	PIN_A15
HEX1_DP	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

Pushbutton and Slide Switches





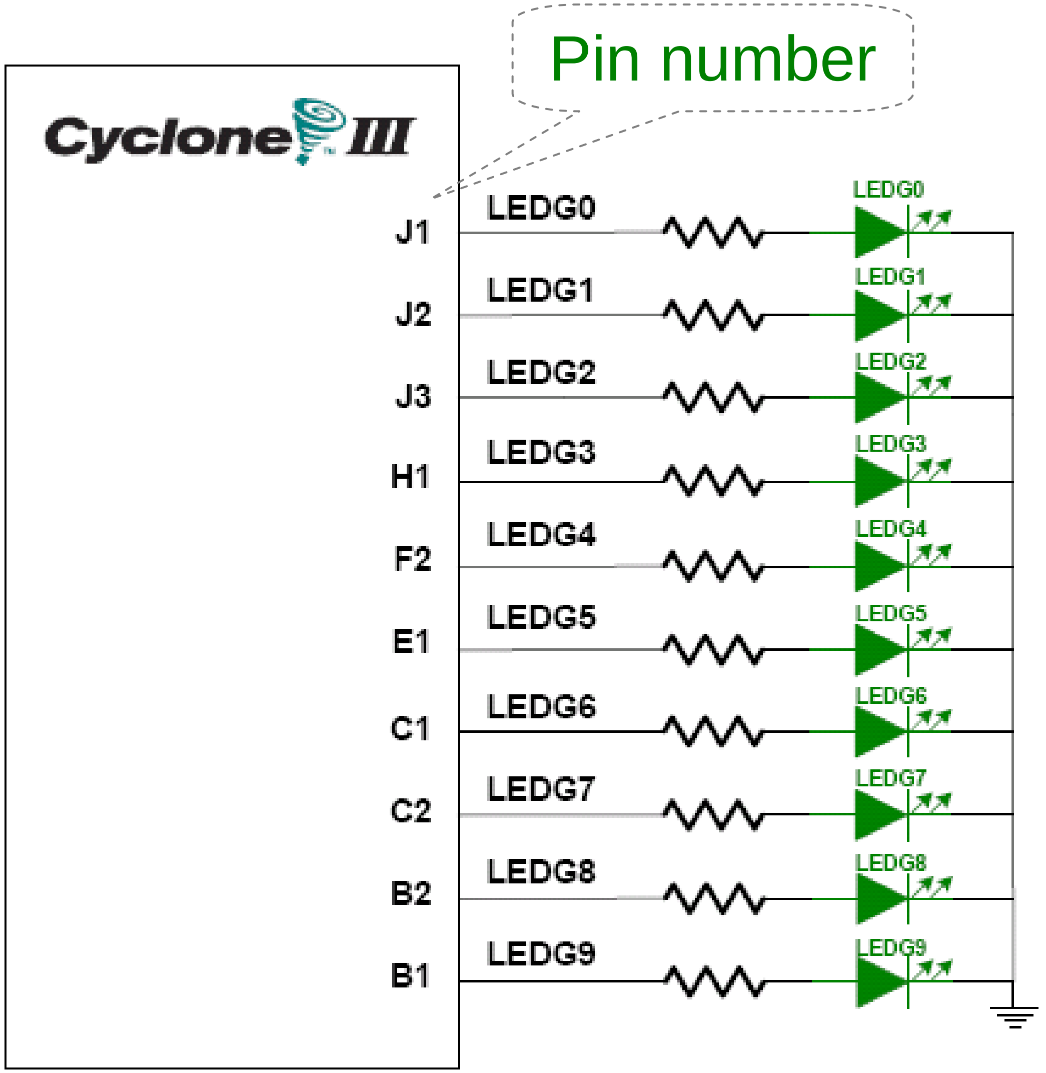
3 Pushbutton switches:
Not pressed → Logic High
Pressed → Logic Low

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