



# Islamic University of Technology

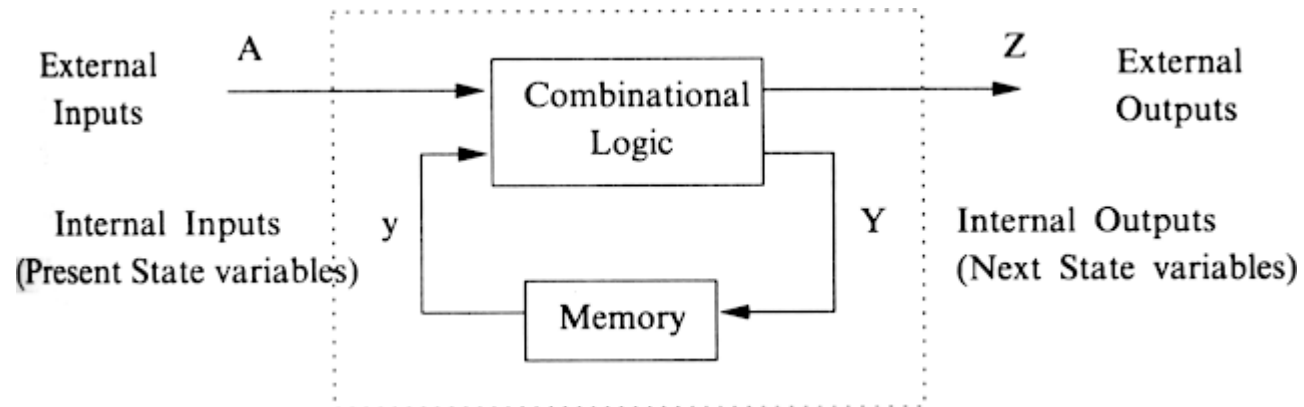
EEE 4483

Digital Electronics & Pulse Techniques

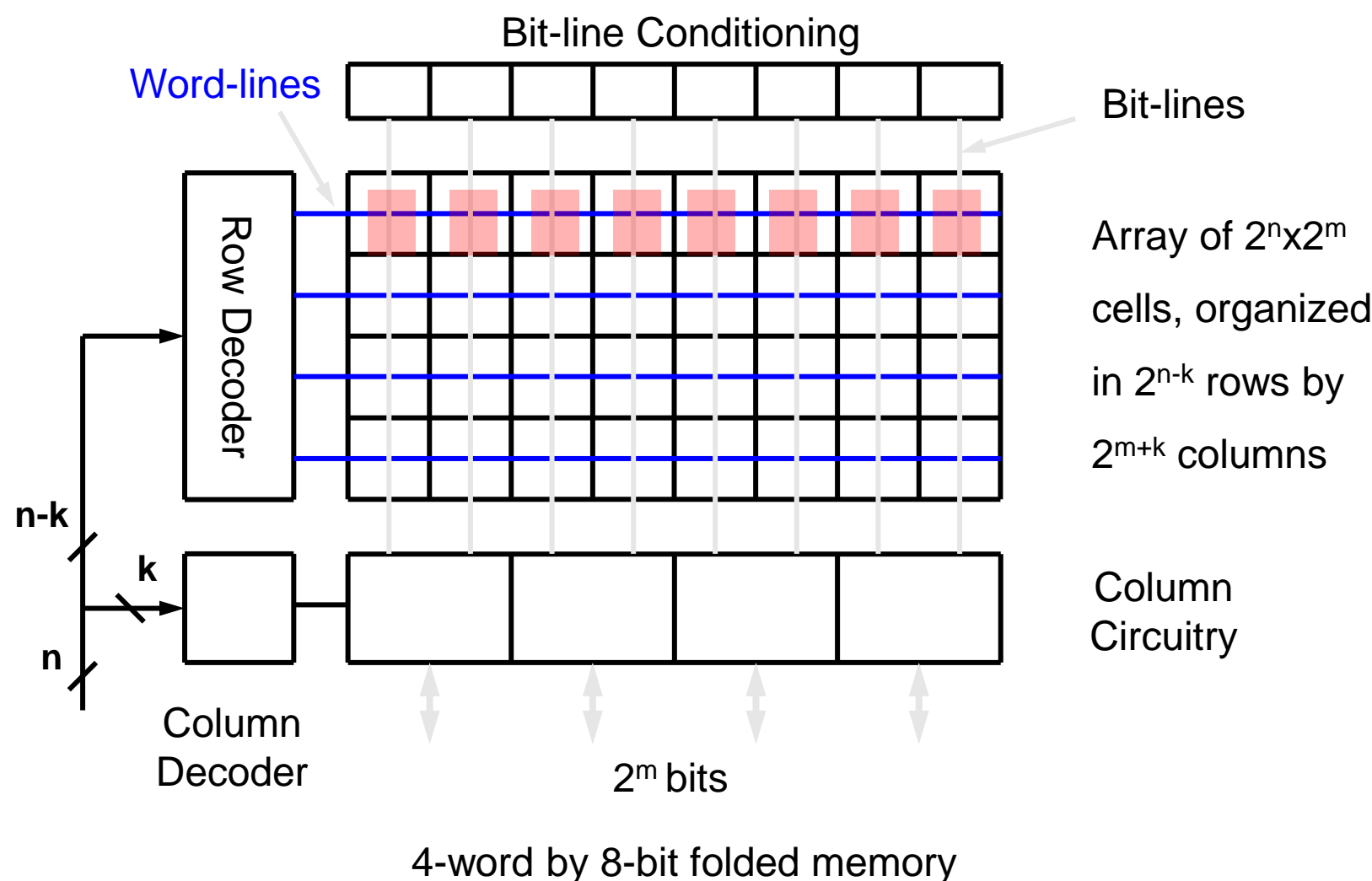
Lecture- 13

# Memory Elements

- ❑ Combinational logic cannot remember
  - Output logic values are function of inputs only
  - Feedback is needed to be able to remember a logic value
- ❑ Memory elements are needed in most digital logic circuits to hold (remember) logic values
- ❑ 2 basic types of memory elements
  - Latches *Level-sensitive to inputs*
  - Flip-flops *Edge-triggered on active edge of clock*



# General Memory Architecture



# General Memory Architecture : continued ...

A memory has  $2^n$  words of  $2^m$  bits each. Usually  $2^n \gg 2^m$ , (e.g. 1M Vs. 64) which will result a very tall structure.

The array is therefore folded into  $2^{n-k}$  rows, each containing  $2^k$  words, namely, every row contains  $2^{m+k}$  bits.

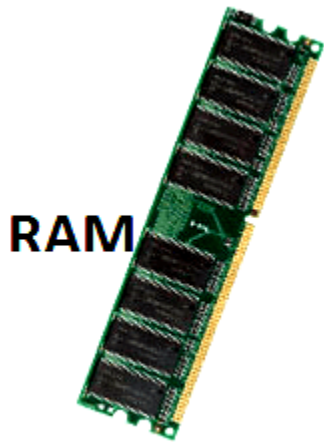
Consider 8-words of 4-bit memory. We'd like to organize it in 4 lines and 8 columns. The memory is folded into 4-word by 8-bit, so  $n=3$ ,  $m=2$  and  $k=1$ .

Larger memories are built from smaller sub-arrays to maintain short word and bit lines.

# Read-only Memory

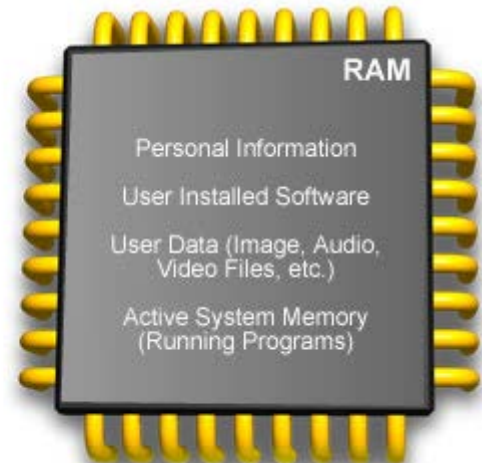
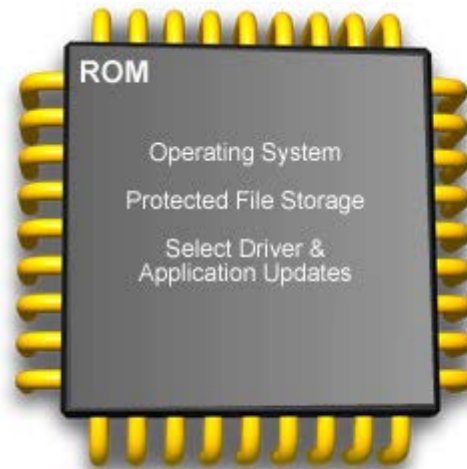
**Read-only memory** (ROM) is a type of storage medium that permanently stores data on personal computers (PCs) and other electronic devices. It contains the programming needed to start a PC, which is essential for boot-up; it performs major input/output tasks and holds programs or software instructions.

Because ROM is read-only, it cannot be changed; it is permanent and non-volatile, meaning it also holds its memory even when power is removed. By contrast, random access memory (RAM) is volatile; it is lost when power is removed. There are numerous ROM chips located on the motherboard and a few on expansion boards. The chips are essential for the basic input/output system (BIOS), boot up, reading and writing to peripheral devices, basic data management and the software for basic processes for certain utilities.




RAM

ROM



# SRAM Basics

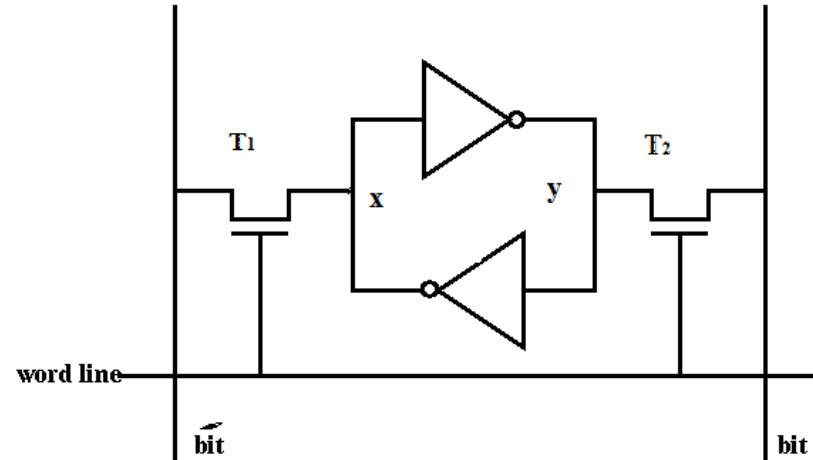
SRAM or Static Random Access Memory is a form of semiconductor memory widely used in electronics, microprocessor and general computing applications. This form of semiconductor memory gains its name from the fact that data is held in there in a static fashion, and does not need to be dynamically updated as in the case of DRAM memory. While the data in the SRAM memory does not need to be refreshed dynamically, it is still volatile, meaning that when the power is removed from the memory device, the data is not held, and will disappear.

 There are two key features to SRAM - Static random Access Memory, and these set it out against other types of memory that are available:

=> **The data is held statically:** This means that the data is held in the semiconductor memory without the need to be refreshed as long as the power is applied to the memory.

=> **SRAM memory is a form of random access memory:** A random access memory is one in which the locations in the semiconductor memory can be written to or read from in any order, regardless of the last memory location that was accessed.

# SRAM Circuit



SRAM cell is made up of 2 inverters cross-connected to form a latch. The latch is connected to 2 bit lines by transistors T1 and T2. T1 and T2 can switch open or closed under control of the word line. When the word line is at ground level, the transistors are turned off and the latch retains its state.

During the **Read Operation**, the word line is activated to close switches T1 and T2.

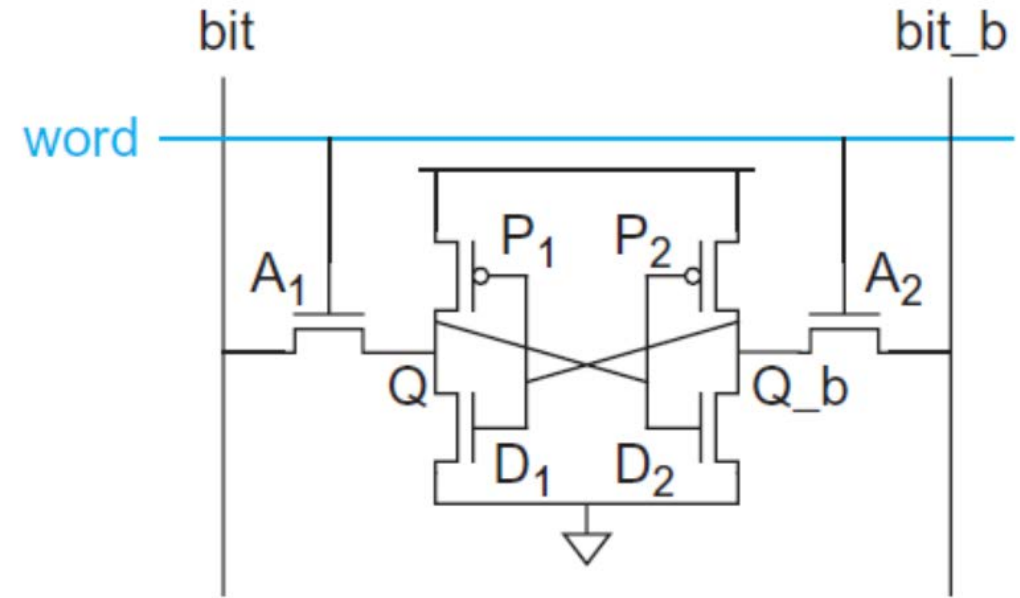
If the cell is in state 1; the signal on **b** line is high and the signal on **b'** line is low. The opposite is true if the cell is in state 0. Thus, b and b' are always complements of each other's.

The sense/write circuit at the end of the two bit lines monitors their state and sets the corresponding output accordingly.

While in **Write Operation**, the Sense/Write circuit drives bit lines **b** and **b'**, instead of sensing their state. It places the appropriate value on bit line **b** and its complement on **b'** and activates the word line. This forces the cell into the corresponding state, which the cell retains when the word line is deactivated.

## 6T SRAM Cell : detailed

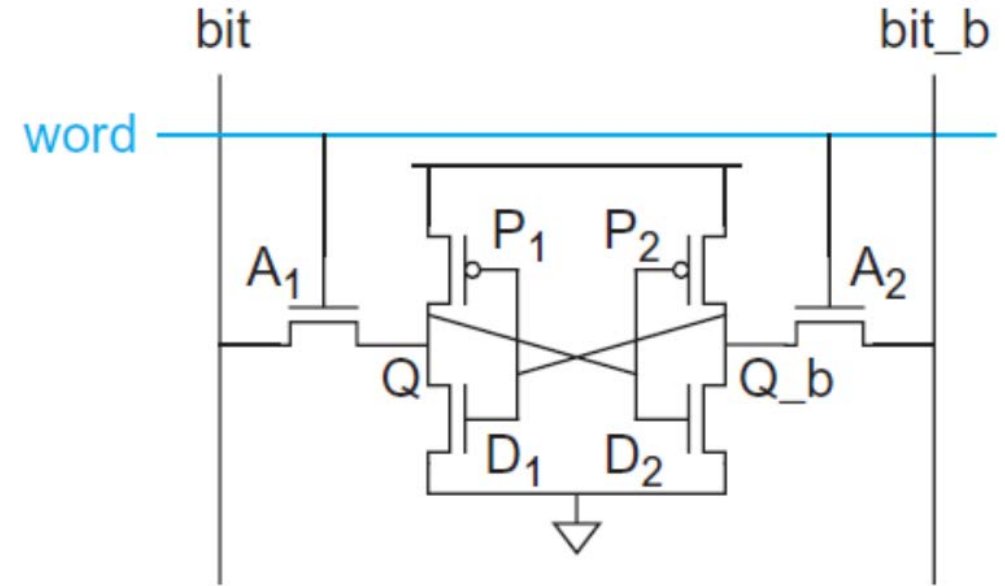
In SRAM, for any operation to be performed, the word line should be high. To perform **read operation**, initially memory should have some value. Therefore let us consider memory has  $Q=1$  and  $Q'=0$ . Raise the word line to high, to perform the read operation. bit and bit\_b acts as output lines, and these bit lines are initially pre-charged i.e. there will be a node voltage  $V_{dd}$  at bit and bit\_b. As  $Q$  and bit are high, there will be no discharge in the circuit. As  $Q'$  is 0, there will be a voltage difference between the  $Q'$  and the node voltage at bit\_b, hence bit\_b voltage decreases. Therefore there will be discharge in the circuit and current flows. Bit and bit\_b are connected to the sense amplifier, this sense amplifier acts as a comparator, so When bit' is low the output will be 1. Hence input  $Q=1$  and we got the output as 1, read operation verified. In the same way consider  $Q=0$  and  $Q'=1$  in the memory. There will be a discharge in the circuit at  $Q$  and bit, since there is voltage difference. The transistors must have ratio such that  $Q$  lies below the threshold region of  $P2/D2$ . This is called read constraint. As bit voltage decreases the output will be 0. when input  $Q=0$ , the output we get is 0. Therefore in both the cases read operation is verified.





# 6T SRAM Cell : detailed

To perform **write operation** consider the memory bits consists of  $Q=0$  and  $Q'=1$ . Initially word line is high and hence write operation can be performed. In the write operation  $bit$  and  $bit'$  are input lines. As we have control on the bit lines, initially make the  $bit\_b$  connected to ground so that we can have the voltage difference between  $Q'$  and  $bit\_b$ . To write  $1$  into the SRAM cell,  $D2$  must be stronger than  $P2$ , this can be achieved by changing the aspect ratio of the transistors. Hence  $Q$  will be  $1$ . Initially  $Q=0$  after the operation  $Q=1$ , hence we write successfully into the memory.



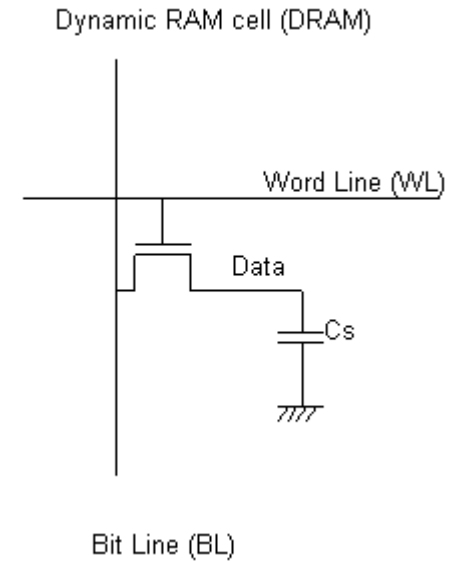
# DRAM

Dynamic random access memory (DRAM) is a type of memory that is typically used for data or program code that a computer processor needs to function. DRAM is a common type of random access memory (RAM) used in personal computers (PCs), workstations and servers. Random access allows the PC processor to access any part of the memory directly rather than having to proceed sequentially from a starting place. RAM is located close to a computer's processor and enables faster access to data than storage media such as hard disk drives and solid-state drives.

DRAM stores each bit of data or program code in a storage cell consisting of a capacitor and a transistor, and is typically organized in a rectangular configuration of storage cells. A DRAM storage cell is dynamic in that it needs to be refreshed or given a new electronic charge every few milliseconds to compensate for charge leaks from the capacitor.

# DRAM Operation

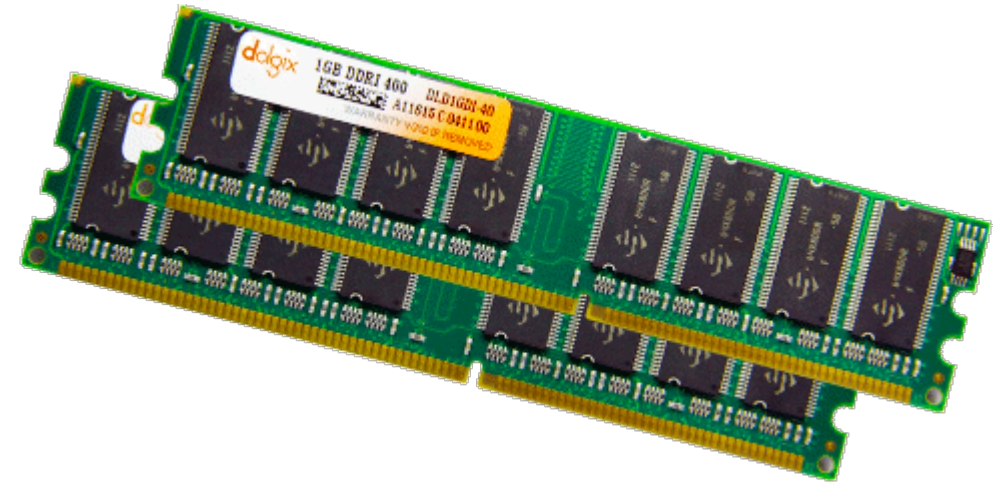
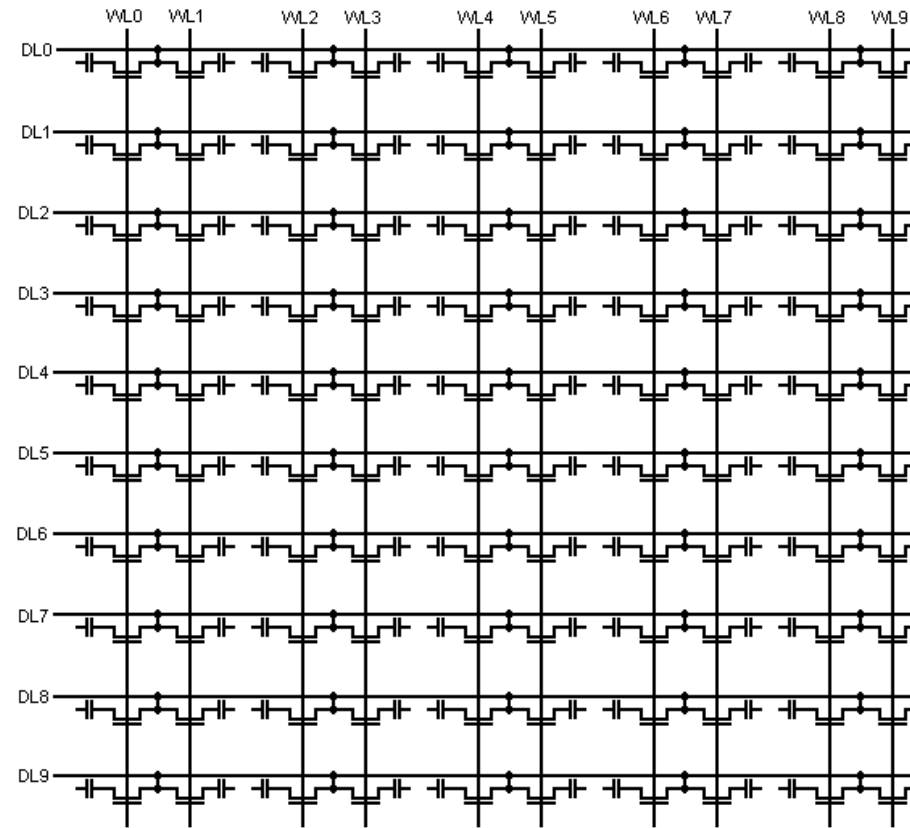
In **Dynamic Random Access Memory** CELL , transistor acts as a switch to Close (allowing current to flow) when voltage applied in address line or Open (no current flow) when no voltage applied in address line. Address Line also known as word line .Which use to signal the transistor to close or open.



During the **Write** operation, a voltage is applied on the bit line and a signal applied to the address line to close the transistor. Then the voltage applied on the bit line will transfer to capacitor and store in the capacitor. However the capacitor has tendency to discharge and has to refresh to maintain the bit.

While in **Reading** Operation ,the instruction find the bit store using the address line to read the data or bit. When the address line is selected ,the transistor turns on and the charge stored on the capacitor is fled out onto a bit line and to sense amplifier. Sense amplifiers compare the capacitor voltage to reference value to determine the logic 1 or logic 0.The read out from cell must be restored to complete the operation.

# Cascaded DRAM



# Feature Comparison Between Memory Types

Memory Type	SRAM	DRAM	Flash
Speed	Very Fast	Fast	Very slow
Density	Low	High	Very high
Endurance	Better	Better	Poor
Power	Low	High	Very low
Refresh	No	Yes	No
Retention	Volatile	Volatile	Non-volatile
Scalable	Good	Bad	Good
Mechanism	Bi-stable latch	Capacitor	FN-tunneling HCI

# VHDL Keyword 'Generic'

Oftentimes we want to be able to specify a property separately for each instance of a component


- Delay
- Bit width

VHDL allows models to be parameterized with generics.

Allows one to make general models instead of making specific models for many different configurations of inputs, outputs, and timing information.

➡ Information passed into a design description from its environment.

```
entity NAND_GATE is
    generic (N: Natural := 2;
            D: Time := 10 ns);
    port (A: in Bit_Vector (1 to N);
          Z: out Bit);
end NAND_GATE;
```



Example how Generic can be used while compiling NAND\_GATE in VHDL.

# Common Conversions in VHDL

## Convert from **Std\_Logic\_Vector** to **Integer** using **Numeric\_Std**

First the coder should think about the data that is represented by `std_logic_vector`. Is it signed data or is it unsigned data? Signed data means that your `std_logic_vector` can be a positive or negative number. Unsigned data means that your `std_logic_vector` is only a positive number. The example below uses the `unsigned()` typecast, but if the used data can be negative then it is required to use the `signed()` **typecast**. Once the input is cast to `std_logic_vector` as unsigned or signed, then it can be converted to integer as shown below:

```
signal input_4      : std_logic_vector(3 downto 0);
signal output_4a    : integer;
signal output_4b    : integer;

-- This line demonstrates the unsigned case
output_4a <= to_integer(unsigned(input_4));

-- This line demonstrates the signed case
output_4b <= to_integer(signed(input_4));
```

# VHDL composite type declaration

Declare a type for creating array, record or unit objects.

```
type identifier is composite_type_definition ;
```

```
type word is array (0 to 31) of bit;
```

```
type data is array (7 downto 0) of word;
```

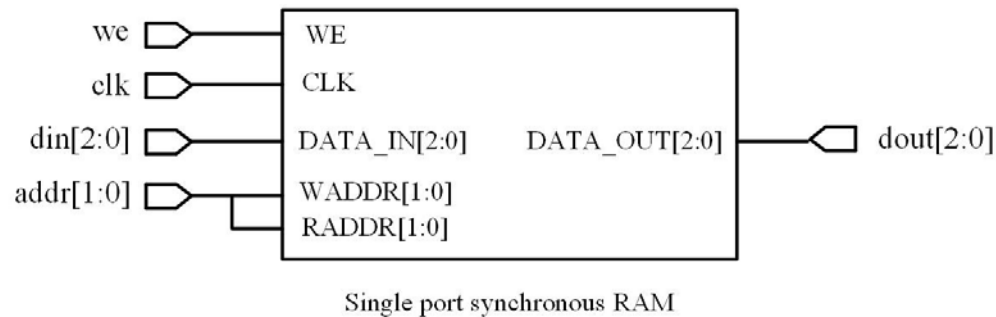
```
type mem is array (natural range <>) of word;
```

```
type matrix is array (integer range <>,
                      integer range <>) of real;
```



# Single Port RAM using VHDL

Single port RAM has one input port (i.e. address line) which is used for both storing and retrieving the data, as shown in the following figure. Here 'addr[1:0]' port is used for both 'read' and 'write' operations.



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity single_port_RAM is
  generic (
    addr_width : integer := 2;
    data_width  : integer := 3
  );
  port(
    clk: in std_logic;
    we  : in std_logic;
    addr : in std_logic_vector(addr_width-1 downto 0);
    din  : in std_logic_vector(data_width-1 downto 0);
    dout : out std_logic_vector(data_width-1 downto 0)
  );
end single_port_RAM;

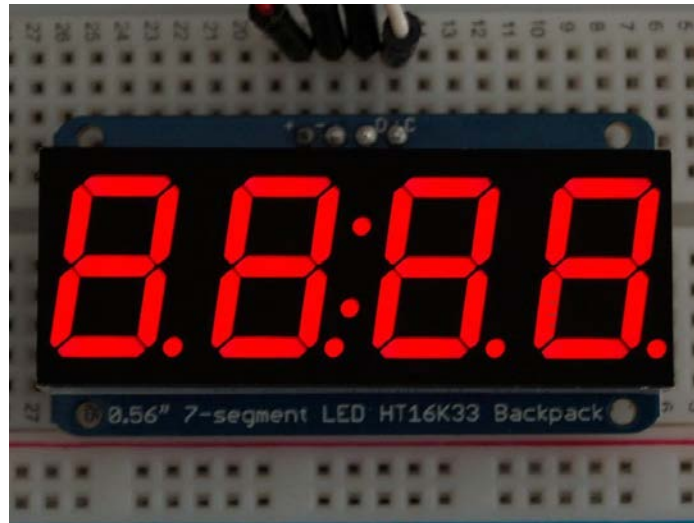
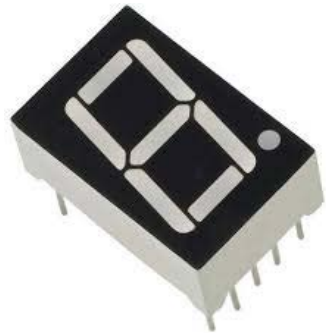
architecture behavioral of single_port_RAM is
  type ram_type is array ((2**addr_width)-1 downto 0) of std_logic_vector (data_width-1 downto 0);
  signal ram_single_port : ram_type;
begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then -- write data to address 'addr'
        --convert 'addr' type to integer from std_logic_vector
        ram_single_port(to_integer(unsigned(addr))) <= din;
      end if;
    end if;
  end process;

  -- read data from address 'addr'
  -- convert 'addr' type to integer from std_logic_vector
  dout <= ram_single_port(to_integer(unsigned(addr)));
end architecture behavioral;
```

# Seven Segment Display

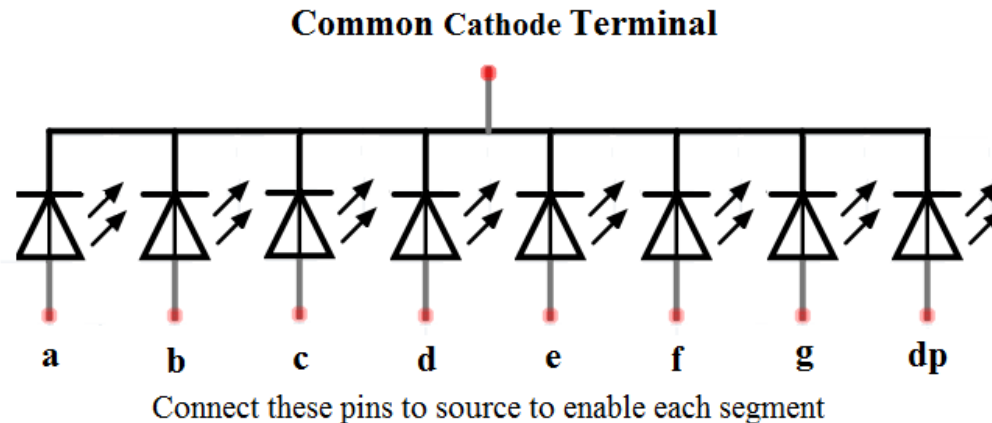
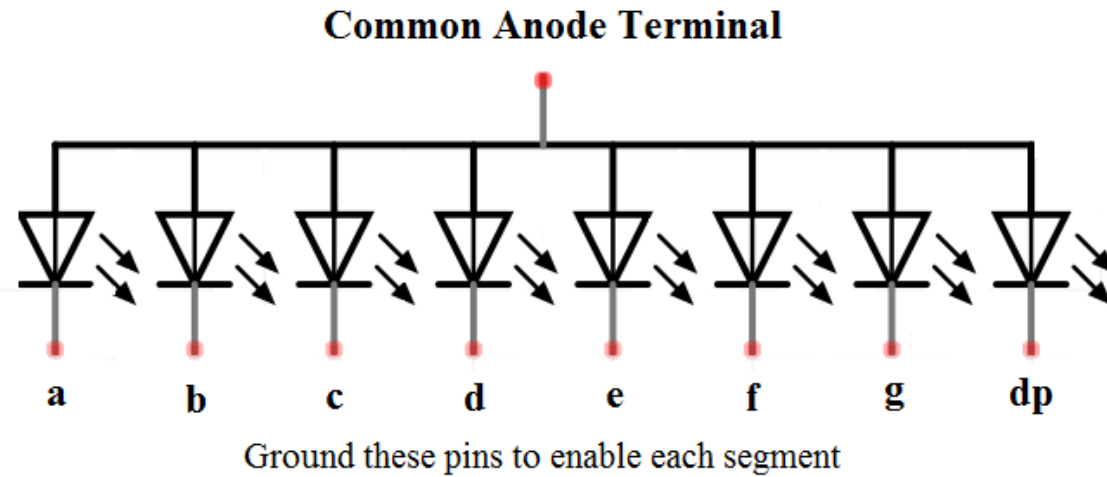
A **seven-segment display** is a set of seven bar-shaped LED (light-emitting diode) or LCD (liquid crystal display) elements, arranged to form a squared-off figure 8. A few seven-segment displays use other illumination devices, such as incandescent or gas-plasma ("neon") lamps. If all elements are activated, the display shows a numeral 8. When some of the elements are activated but not others, any single-digit numeral from 0 to 9, as well as most uppercase and lowercase letters of the English alphabet, can be portrayed.

**Seven-segment displays** are commonly used in digital clocks, clock radios, timers, wristwatches, and calculators. They can also be found in motor-vehicle odometers, speedometers, radio frequency indicators, and practically any other display that makes use of alphanumeric characters alone (without the need for graphics). Some seven-segment displays produce an "italicized" (slanted) set of characters.

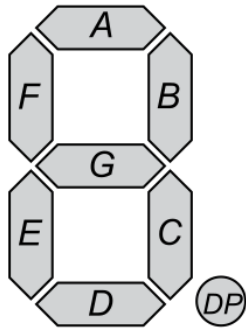
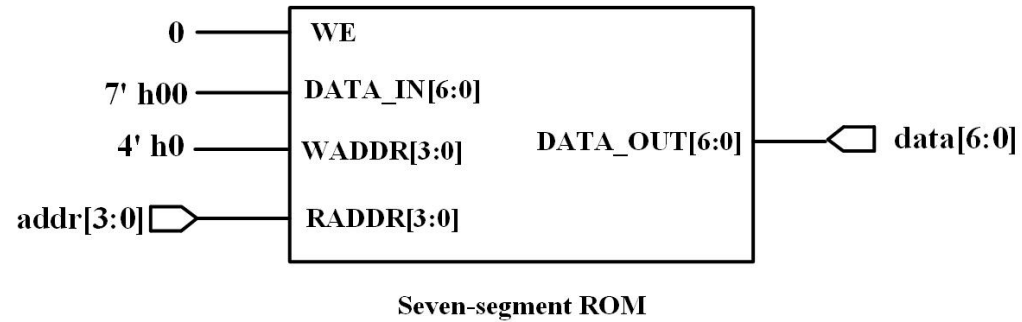


# Seven Segment Display : Continued ..

In common anode type, all the anodes of 8 LED's are connected to the common terminal and cathodes are left free. Thus, in order to glow the LED, these cathodes have to be connected to the logic '0' and anode to the logic '1'.



# Seven segment (Common Anode) ROM using VHDL



Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
0	0	0	0	0	0	1	0
1	0	0	1	1	1	1	1
0	0	1	0	0	1	0	2
0	0	0	0	1	1	0	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	0	5
0	1	0	0	0	0	0	6
0	0	0	1	1	1	1	7
0	0	0	0	0	0	0	8
0	0	0	0	1	1	0	9

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ROM_sevenSegment is
    generic(
        addr_width : integer := 16; -- store 16 elements
        addr_bits   : integer := 4;  -- required bits to store 16 elements
        data_width  : integer := 7;  -- each element has 7-bits
    );
    port(
        addr : in std_logic_vector(addr_bits-1 downto 0);
        data : out std_logic_vector(data_width-1 downto 0)
    );
end ROM_sevenSegment;

architecture behavioral of ROM_sevenSegment is

    type rom_type is array (0 to addr_width-1) of std_logic_vector(data_width-1 downto 0);

    signal sevenSegment_ROM : rom_type := (
        "1000000", -- 0, active low i.e. 0:display & 1:no display
        "1111001", -- 1
        "0100100", -- 2
        "0110000", -- 3
        "0011001", -- 4
        "0010010", -- 5
        "0000010", -- 6
        "1111000", -- 7
        "0000000", -- 8
        "0010000", -- 9
        "0001000", -- a
        "0000011", -- b
        "1000110", -- c
        "0100001", -- d
        "0000110", -- e
        "0001110"  -- f
    );

begin
    data <= sevenSegment_ROM(to_integer(unsigned(addr)));
end architecture behavioral;
```

<https://codeshare.io/5vz79m>

Click Here