Digital Logic Circuit (SE273 – Fall 2020)

Lecture 10: Memory

Jaesok Yu, Ph.D. (jaesok.yu@dgist.ac.kr)

Assistant Professor

Department of Robotics Engineering, DGIST

Goal of this lecture

- Learn different types of memory units
  - Random-access memory (RAM)
  - Read-only memory (ROM)

- Learn how memory decoding works
  - Internal structure of memory cell and array
  - Address multiplexing
  - Error detection and correction

Introduction to Memory Unit

- A memory unit is a device to which binary information is transferred for storage
  - Data is retrieved for processing when needed
  - It is transferred to selected registers in a processor
  - Intermediate and final results obtained are transferred back to memory

- Memory stores new information for later use
  - **Storing new information** into memory is referred to as a memory ***write***
  - Process of transferring the stored information out of memory is referred to as a memory *read*

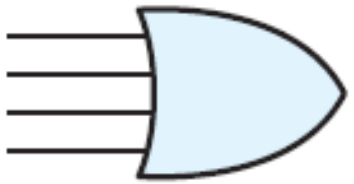
Types of Memories

- Two types of memories are used in digital systems
  - Random-access memory (RAM): can perform both write and read operations
  - Read-only memory (ROM): can perform only the read operation

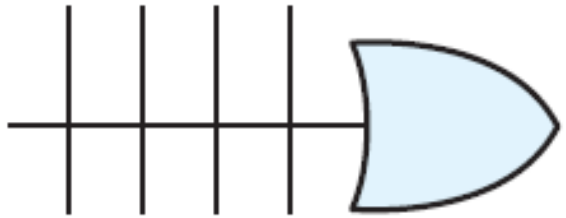
- More details on ROM
  - The information stored in ROMs cannot be altered by writing
  - It is a programmable logic device (PLD)
  - Binary information is embedded within hardware by programming the device

Programmable Logic Device (PLD)

- ROM is one example of a PLD
  - Others include programmable logic array (PLA), programmable array logic (PAL), and field-programmable gate array (FPGA)
  - PLD is an IC with internal logic gates connected through programmable electronic paths
  - A particular configuration is programmed for the desired logic function



Conventional symbol

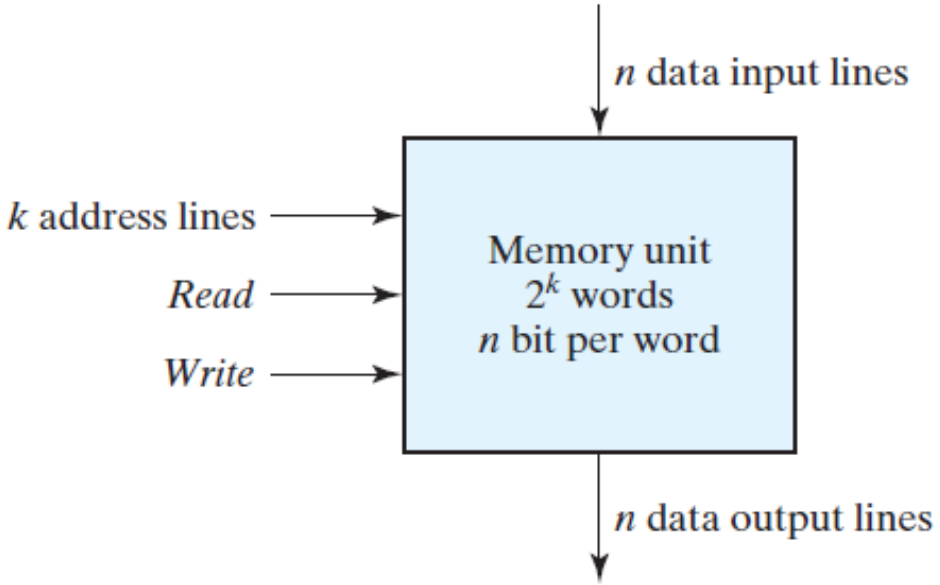


Array logic symbol

Random-Access Memory (RAM)

Block Diagram of Memory Unit

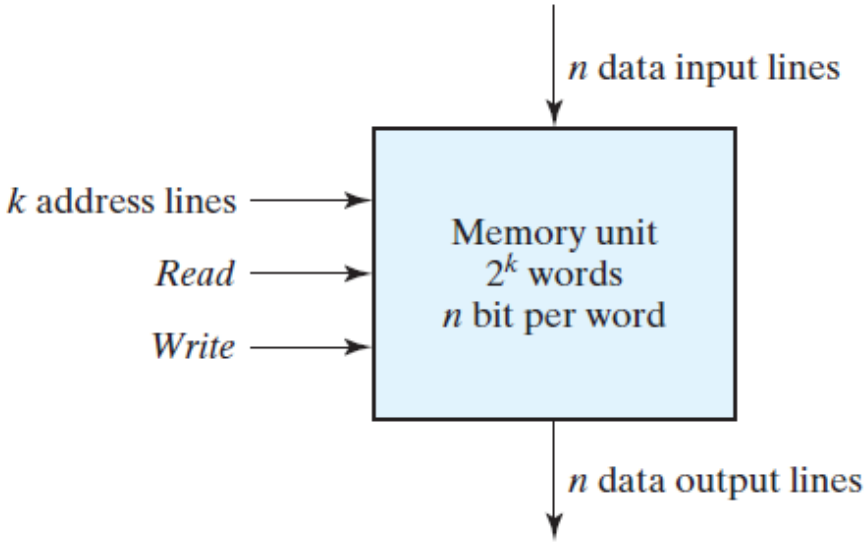
- Memory unit is a collection of storage cells
  - Associated circuits are needed to transfer information into and out of a device
  - The **time** **it takes to transfer information** to or from **any desired random location** is always the **same** → why it is called **random-access**
  - What about a magnetic-tape based storage?



- Groups of bits are called **words**
- A group of 8bits is called a **byte**
- Most computers use words that are **multiples of 8bits** in length

Memory Address and Its Content

- Read/Write signal control the direction of data transfer
  - Let's consider a 1K x 16bit memory unit
  - 10 bits are used for the address
  - 'k' bit allows access to 0 to 2^k -1



Memory address		Memory content
Binary	Decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
⋮	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1011111000100101

Write and Read Operations

- Two operations that RAM can perform are write and read
  - **Write** signal specifies a **transfer-in** operation
  - **Read** signal specifies a **transfer-out** operations

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

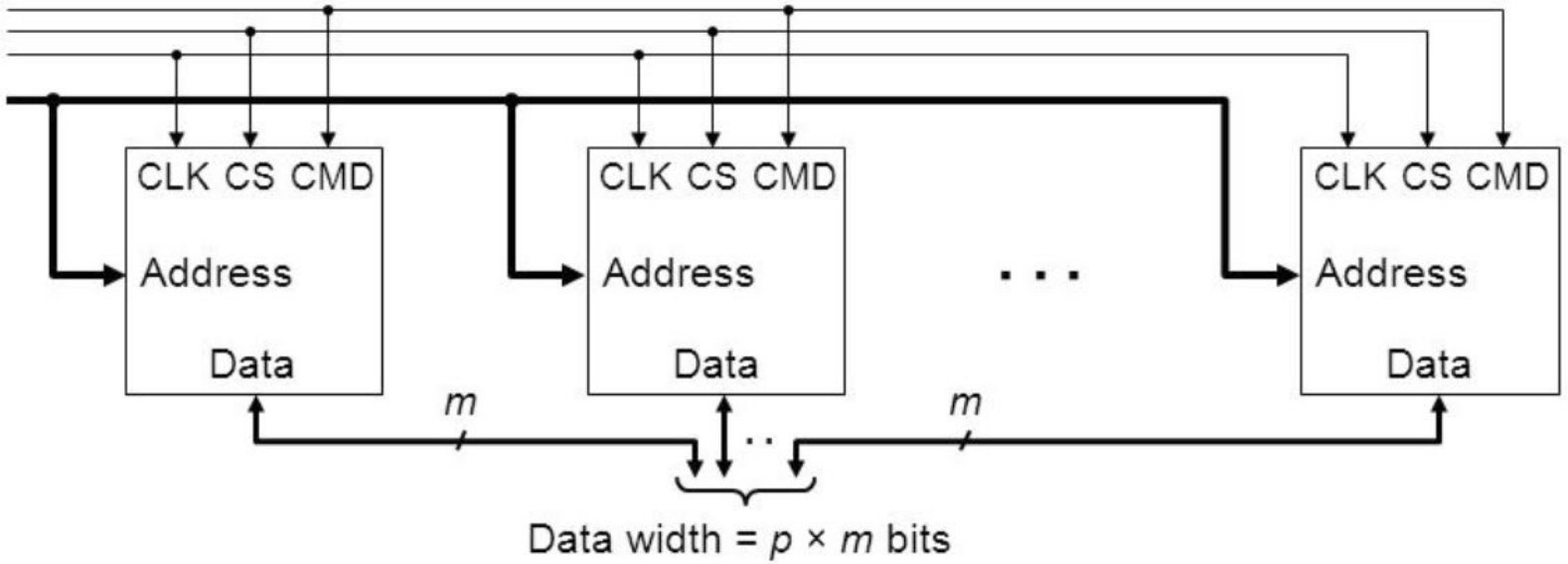
Write and Read Operations

- Instead of having separate read and write inputs to control the two operations, most integrated circuits provide two other control inputs
  - One input selects the memory unit (**chip select**)
  - The other determines the operation (**read/write**)

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

What is Chip Select?

- Memory Rank: Set of DRAM chips accessed in parallel
  - Same Chip Select (CS) and Command (CMD)
  - Same address, but different data lines
  - Increases memory capacity and bandwidth
  - Example: 64-bit data bus using 4 × 16-bit DRAM chips



Memory Simulation Model in HDL

```
ENTITY K4S641633H IS
    GENERIC (

        tAC       : TIME    :=  7.0 ns;        -- Access time from CLK (pos. edge)
        tHZ       : TIME    :=  7.0 ns;        -- [tSAC] max 7ns Data-out high-impedance time
        tOH       : TIME    :=  2.5 ns;        -- [tOH] min 2.5 Data-out hold time
        tMRD      : INTEGER :=  2;             -- [MRS 2clk] LOAD MODE REGISTER command to ACTIVE or REFRESH command 2 Clk Cycles
        tRAS      : TIME    := 45.0 ns;        -- [tRAS] 60ns -100us ACTIVE to PRECHARGE command
        tRC       : TIME    := 64.0 ns;        -- [tRC] min 84ns ACTIVE to ACTIVE command period
        tRCD      : TIME    := 19.0 ns;        -- [RCD] min 24ns ACTIVE to READ or WRITE delay
        tRFC      : TIME    := 64.0 ns;        -- AUTO REFRESH period = tRC(min)
        tRP       : TIME    := 19.0 ns;        -- PRECHARGE command period
        tRRD      : TIME    := 15.0 ns;        -- [tRRD] min 20ns ACTIVE bank a to ACTIVE bank b command

        tWRa      : TIME    := 18.0 ns;        -- write recovery time : tRDL=2clk=9*2ns=18ns
        tWRm      : TIME    := 37.0 ns;        -- tRDL + tRP = (18 + 19) ns

        tAH       : TIME    :=  1.0 ns;        -- [tSH] 1.5ns Address hold time
        tAS       : TIME    :=  2.0 ns;        -- [tSS] 2.5ns Address setup time
        tCH       : TIME    :=  2.5 ns;        -- [tCH]3ns CLK high-level width
        tCL       : TIME    :=  2.5 ns;        -- [tCL]3ns CLK low-level width
        tCK       : TIME    :=  9.5 ns;        -- [tcc]10-1000 Clock cycle time
        tDH       : TIME    :=  1.0 ns;        -- [tSH] 1.5ns Data-in hold time
        tDS       : TIME    :=  2.0 ns;        -- [tSS] 2.5ns Data-in setup time
        tCKH      : TIME    :=  1.0 ns;        -- [tSH] 1.5ns CKE hold time
        tCKS      : TIME    :=  2.0 ns;        -- [tSS] 2.5ns CKE setup time
        tCMH      : TIME    :=  1.0 ns;        -- [tSH] 1.5nsCS#, RAS#, CAS#, WE#, DQM hold time
        tCMS      : TIME    :=  2.0 ns;        -- [tSS] 2.5ns CS#, RAS#, CAS#, WE#, DQM setup time

        addr_bits : INTEGER := 12;
        data_bits : INTEGER := 16;
        col_bits  : INTEGER := 8
    );
```

Memory Simulation Model in HDL

```
PORT (
        Dq    : INOUT STD_LOGIC_VECTOR (data_bits - 1 DOWNTO 0) := (OTHERS => 'Z');
        Addr  : IN    STD_LOGIC_VECTOR (addr_bits - 1 DOWNTO 0) := (OTHERS => '0');
        Ba    : IN    STD_LOGIC_VECTOR (1 DOWNTO 0) := "00";
        Clk   : IN    STD_LOGIC := '0';
        Cke   : IN    STD_LOGIC := '1';
        Cs_n  : IN    STD_LOGIC := '1';
        Ras_n : IN    STD_LOGIC := '1';
        Cas_n : IN    STD_LOGIC := '1';
        We_n  : IN    STD_LOGIC := '1';
        Dqm   : IN    STD_LOGIC_VECTOR (1 DOWNTO 0) := "00"
    );
END K4S641633H;
```

Memory in FPGA

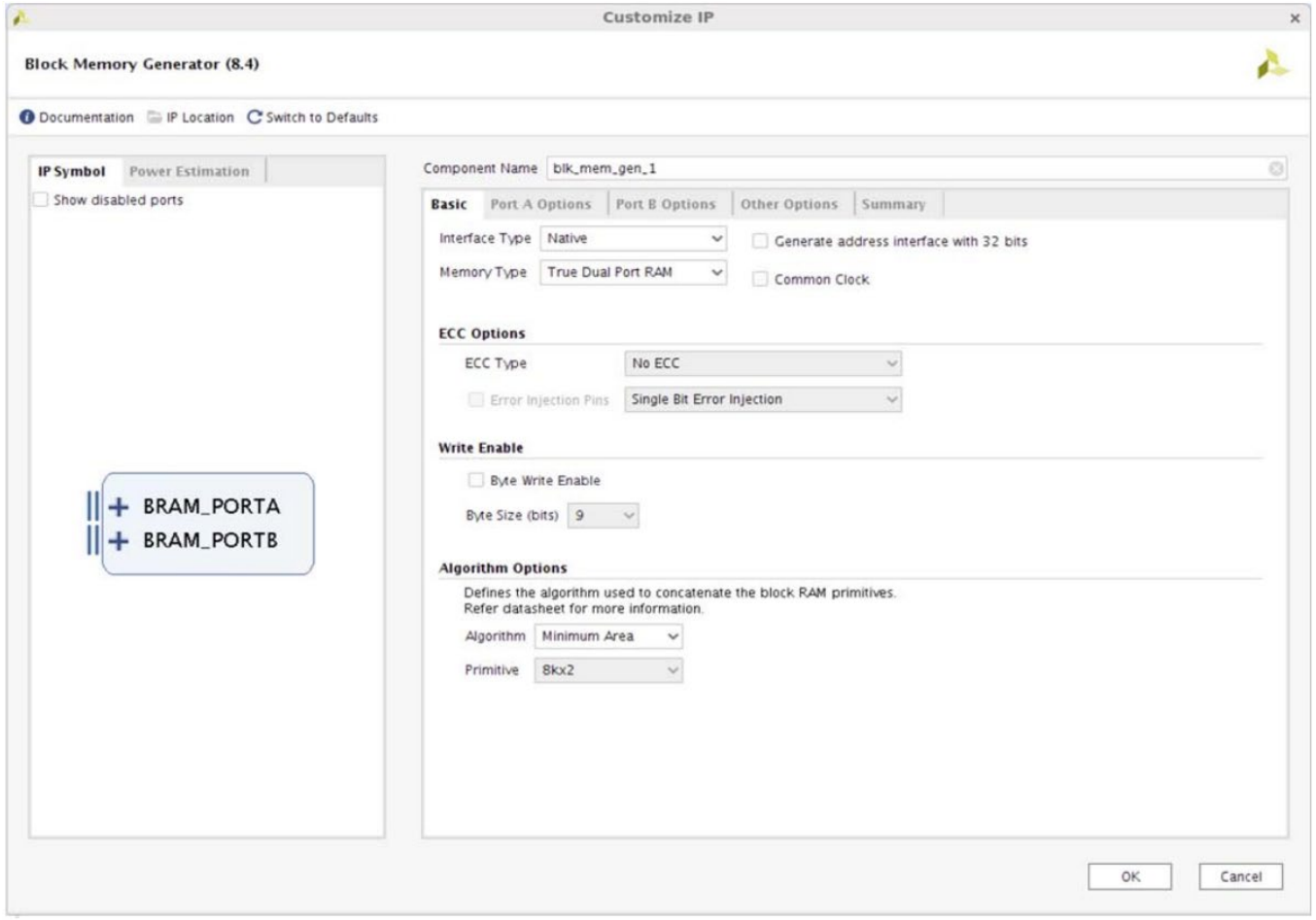


Figure 4-1: **Block Memory Generator Basic Tab**

Memory in FPGA

ROM (IP gen)

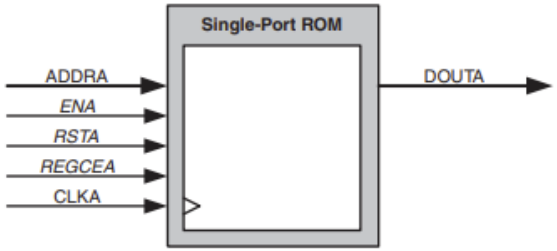


Figure 3-1: **Single-port ROM**

The Dual-port ROM allows Read access to the memory space through two ports, as shown in Figure 3-2.

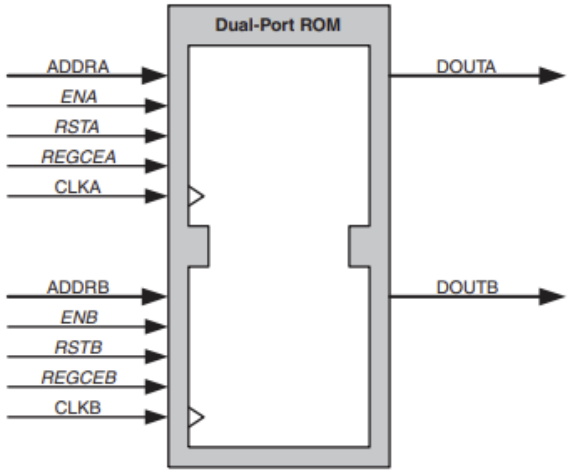


Figure 3-2: **Dual-port ROM**

The Single-port RAM allows Read and Write access to the memory through a single port, as shown in Figure 3-3.

RAM (IP gen)

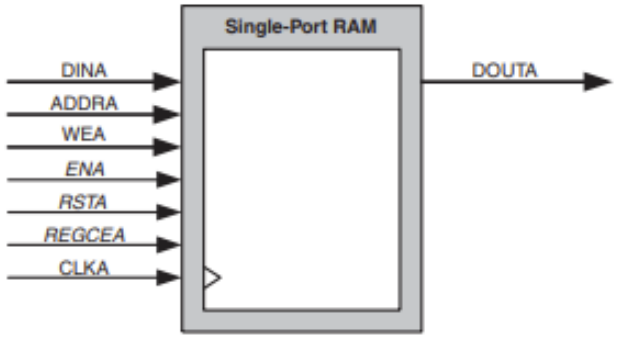


Figure 3-3: **Single-port RAM**

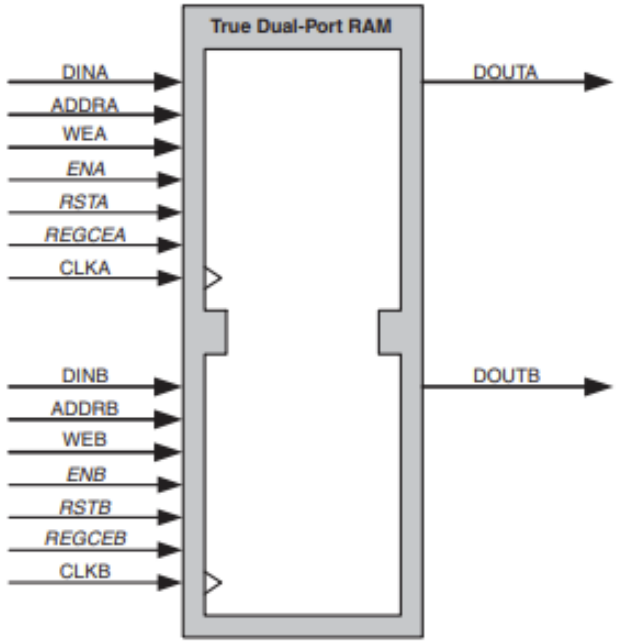


Figure 3-5: **True Dual-port RAM**

Memory in FPGA

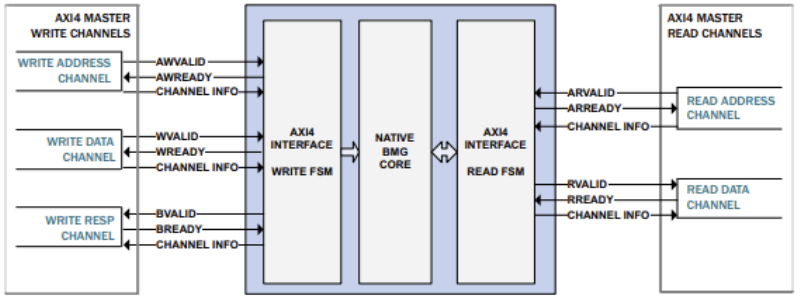


Figure 1-1: **AXI4 Interface BMG Block Diagram**

All communication in the AXI protocol is performed using five independent channels. Each of the five independent channels consists of a set of information signals and uses a two-way **valid** and **ready** handshake mechanism. The information source uses the **valid** signal to show when valid data or control information is available on the channel. The information destination uses the **ready** signal to show when it can accept the data.

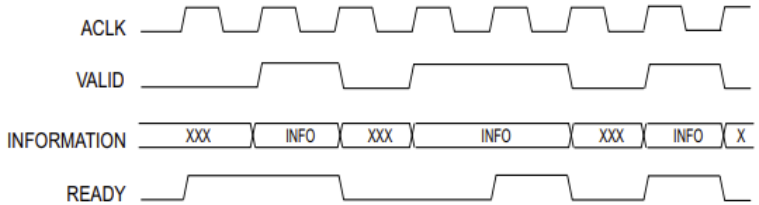


Figure 1-2: **AXI4 Interface Handshake Timing Diagram**

In Figure 1-2, the information source generates the **valid** signal to indicate when data is available.

The destination generates the **ready** signal to indicate that it can accept the data, and transfer occurs only when both the **valid** and **ready** signals are High.

The AXI4 Block Memory Generator core is an AXI4 endpoint Slave IP and can communicate with multiple AXI4 Masters in an AXI4 System or with Standalone AXI4 Masters in point to point applications. The core supports Simple Dual-Port RAM configurations. Because AXI4 Block Memories are built using Native interface Block Memories, they share many common features.

All Write operations are initiated on the Write Address Channel (AW) of the AXI bus. The AW channel specifies the type of Write transaction and the corresponding address information. The Write Data Channel (W) communicates all Write data for single or burst Write

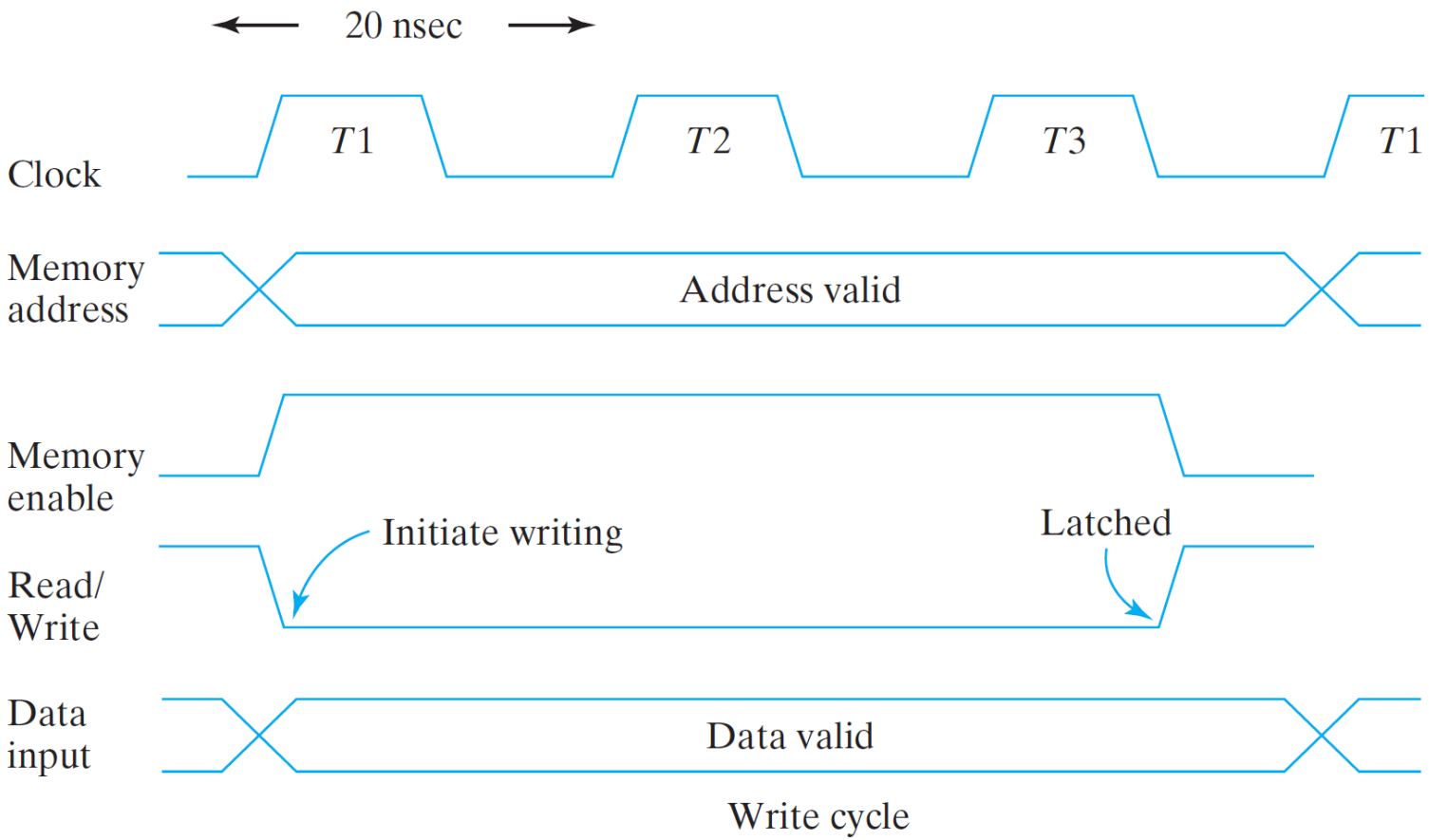
Timings in Memory Unit

- The operation of memory unit is controlled by a CPU
  - CPU is synchronized by its own clock
  - The memory, however, does not employ an internal clock

- Access time and cycle time
  - Access time: time required to select a word and read it
  - Cycle time: time required to complete a write operation
  - They should be within a time equal to a **fixed number of CPU clock cycles**

Timing Diagram - Write

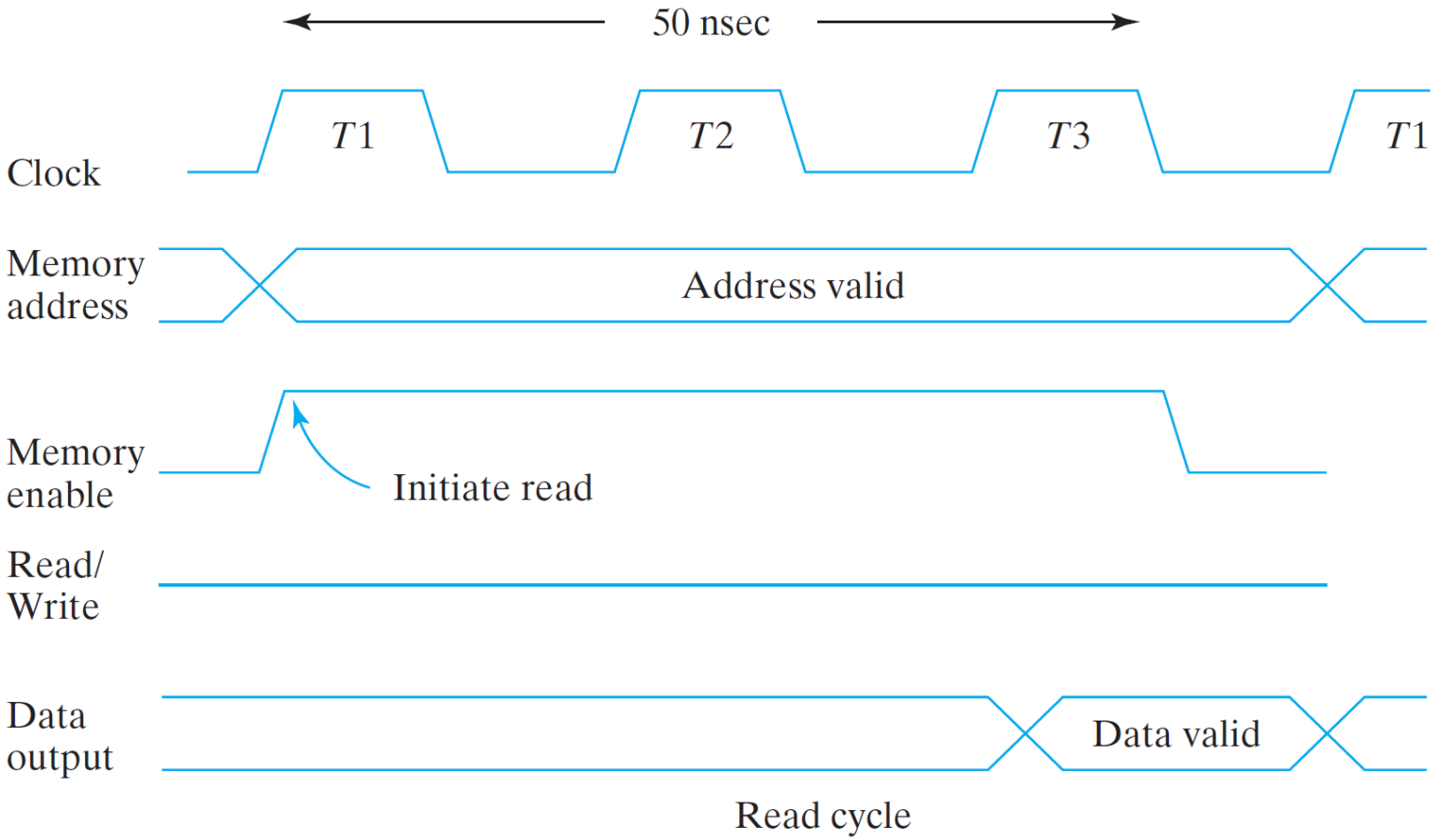
- Suppose that CPU operates at 50MHz (20ns for one clock cycle)
  - Also, let's assume that access time or cycle time of a memory unit does not exceed 50ns



Write cycle

Timing Diagram - Read

- Suppose that CPU operates at 50MHz (20ns for one clock cycle)
  - Also, let's assume that access time or cycle time of a memory unit does not exceed 50ns



Read cycle

Properties of Memory

- Static RAM vs. Dynamic RAM
  - SRAM: the stored information **remains valid** as long as power is applied
  - DRAM: the stored charge on the capacitors tends to **discharge** over time (requires **periodical refreshing** - every few ms)
  - Both are volatile memory

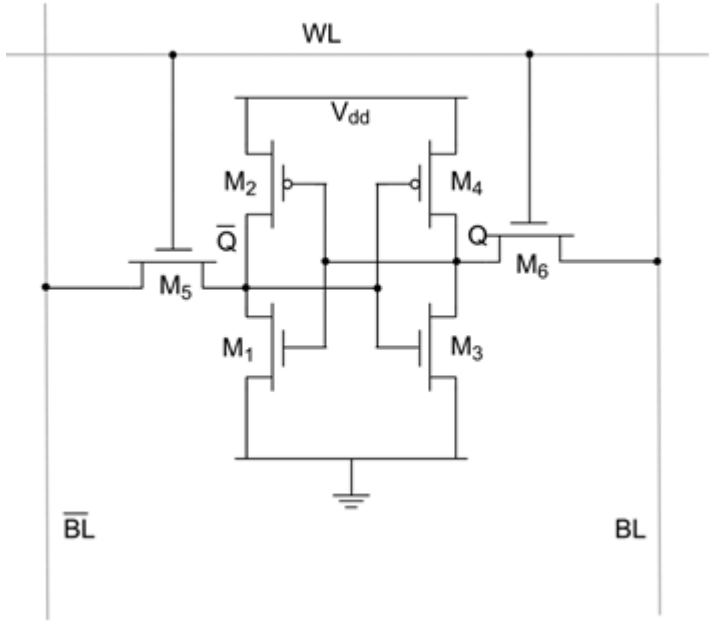
- Nonvolatile memory
  - It retains its stored information after the removal of power
  - Ex) Magnetic disk, ROM

SRAM vs DRAM

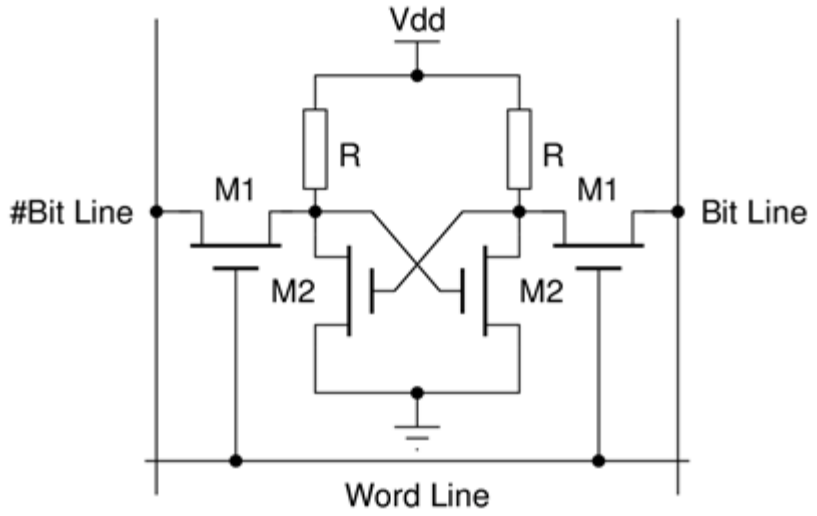
	SRAM	DRAM
Speed	Faster	Slower
Capacity	Small	Large
Cost	Expensive	Cheap
Usage	Cache Memory	Main Memory
Density	Less Dense	Highly Dense
Construction	Complex and Uses TRs and Latches	Simple and Uses Capacitors and very few TRs
Single Memory Cell	6 TRs (or 4 TRs)	1 TR
Power Consumption	Low	High (but, becomes comparable in higher frequencies)

SRAM Cell

- SRAM cell
  - The basic binary storage cell
  - Typically designed as an electronic circuit rather than a logic circuit
  - For convenience, let's model the RAM chip w/ a logic model



6 Transistors

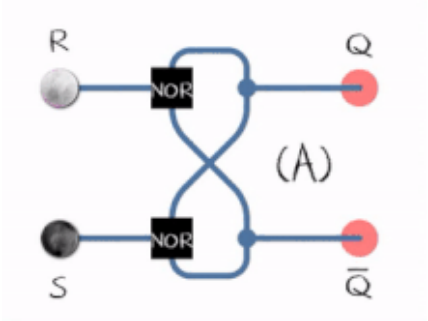


4 Transistors

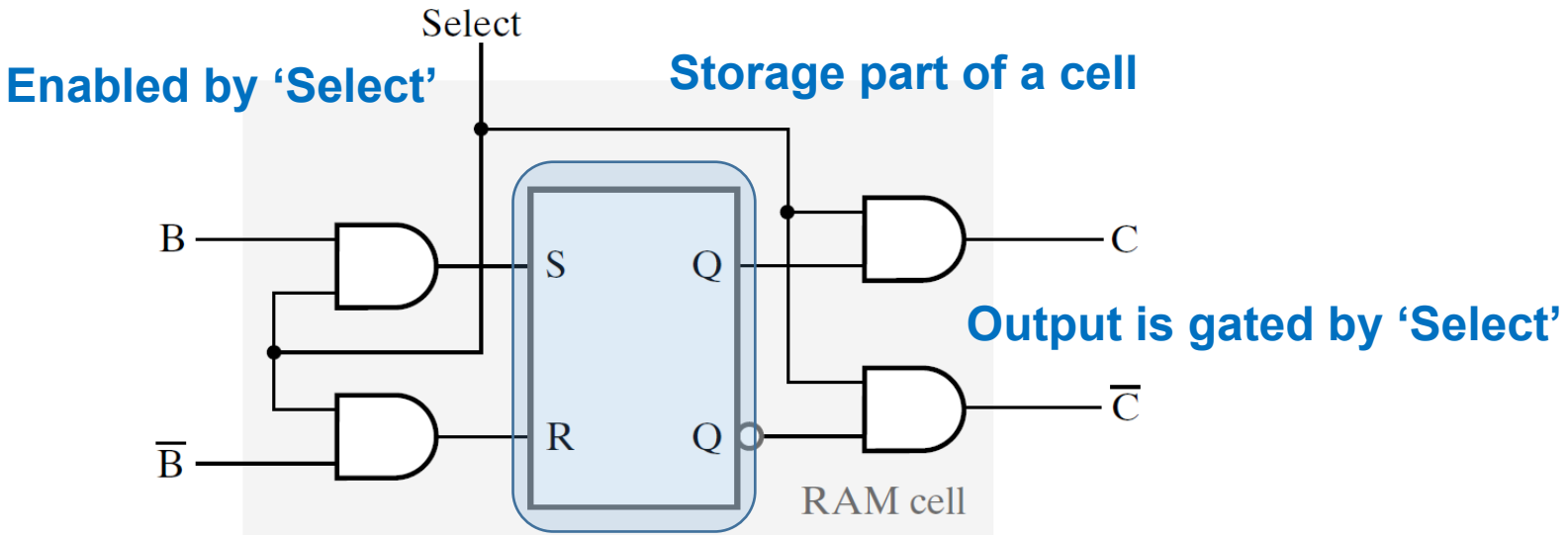
SRAM Cell Using a Logic Model

- SRAM cell
  - The basic binary storage cell
  - Typically designed as an electronic circuit rather than a logic circuit
  - For convenience, let's model the RAM chip w/ a logic model

Characteristic table				Excitation table			
S	R	Q_{next}	Action	Q	Q_{next}	S	R
0	0	Q	Hold state	0	0	0	X
0	1	0	Reset	0	1	1	0
1	0	1	Set	1	0	0	1
1	1	X	Not allowed	1	1	X	0



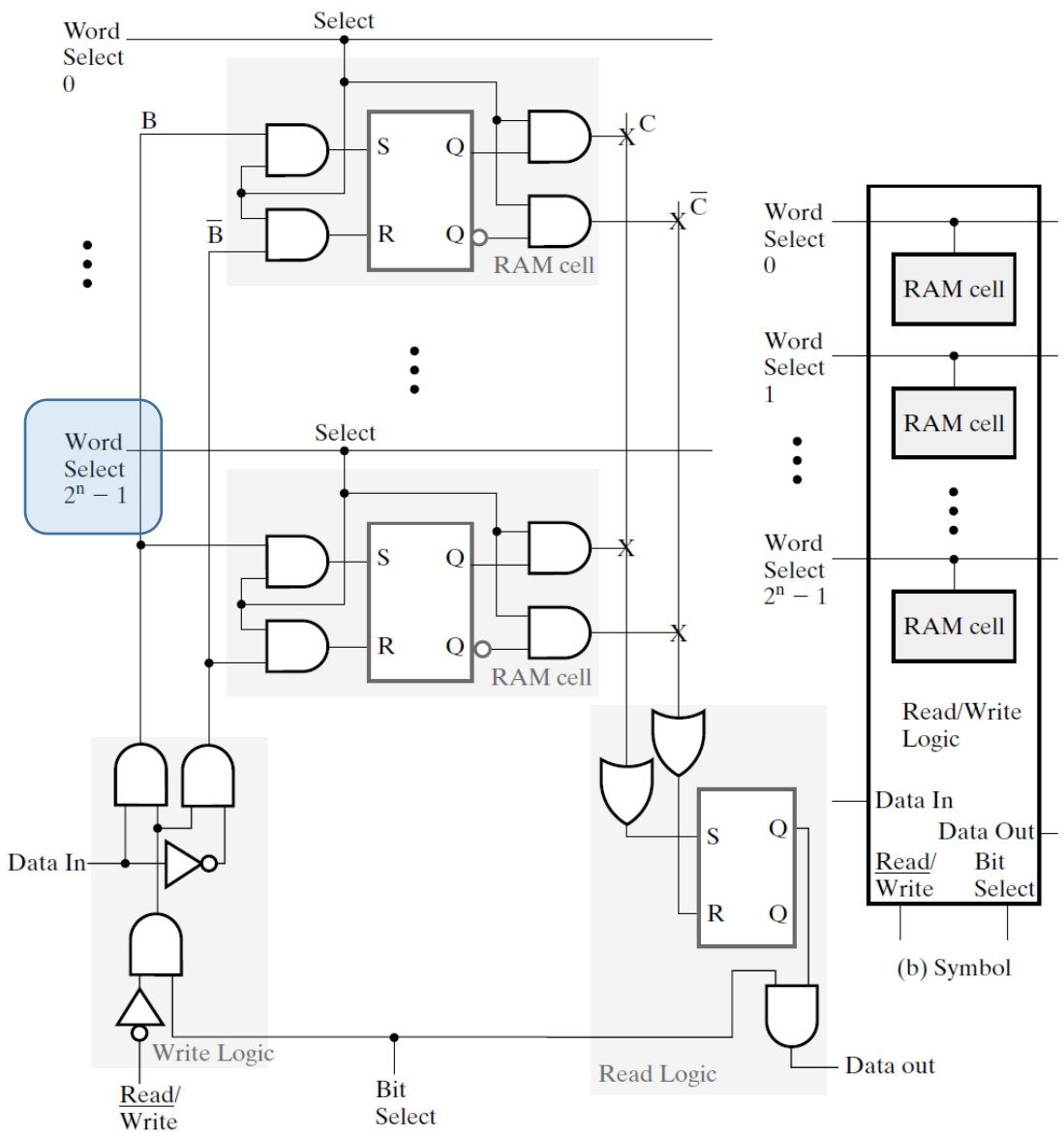
Black: 1/White: 0



Static RAM Cell

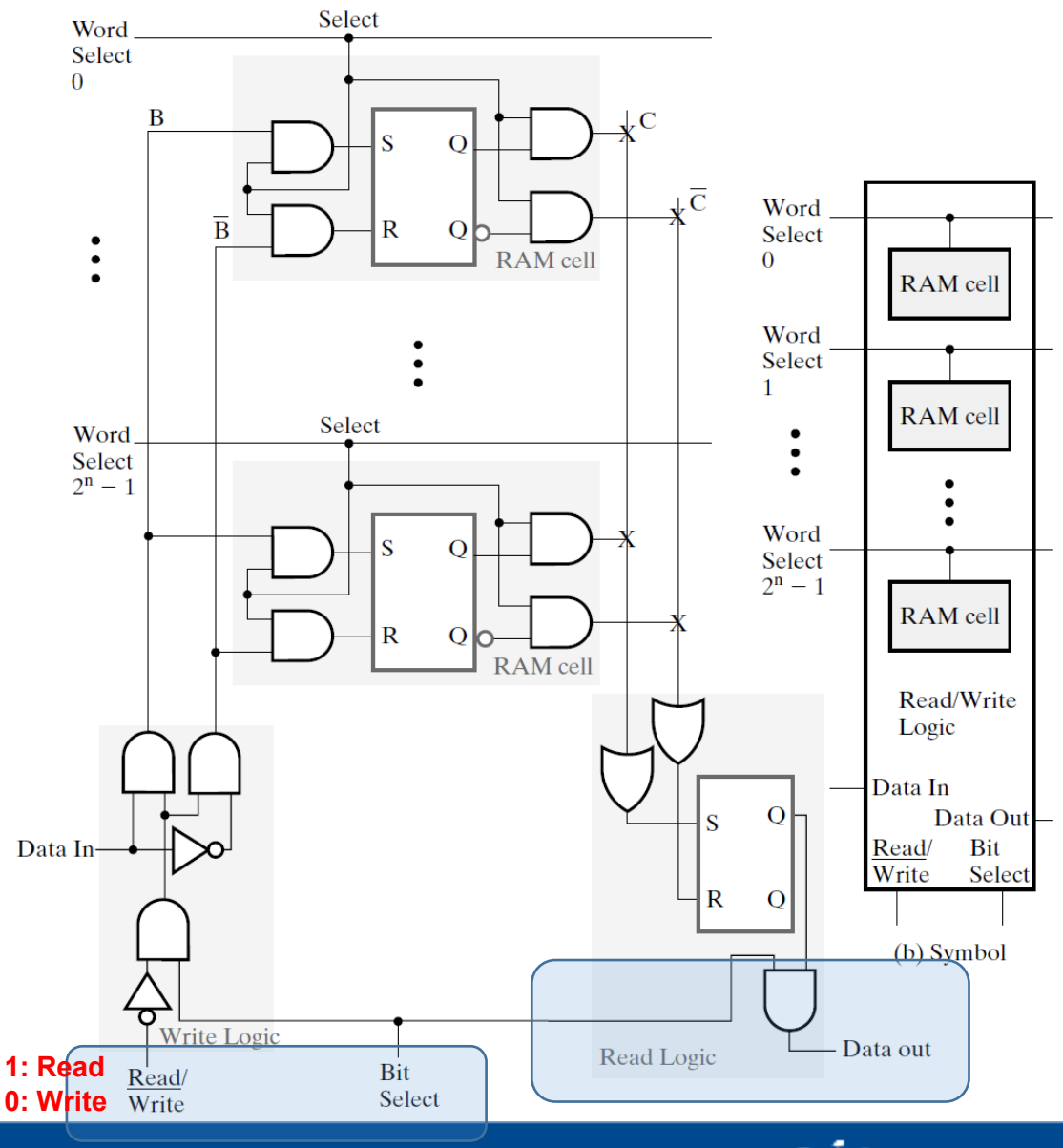
SRAM Bit Slice Model

- SRAM bit slice
  - A set of RAM cells
  - Plus, read and write circuits
- Loading of a cell latch is controlled by ‘Word Select’



SRAM Bit Slice Model

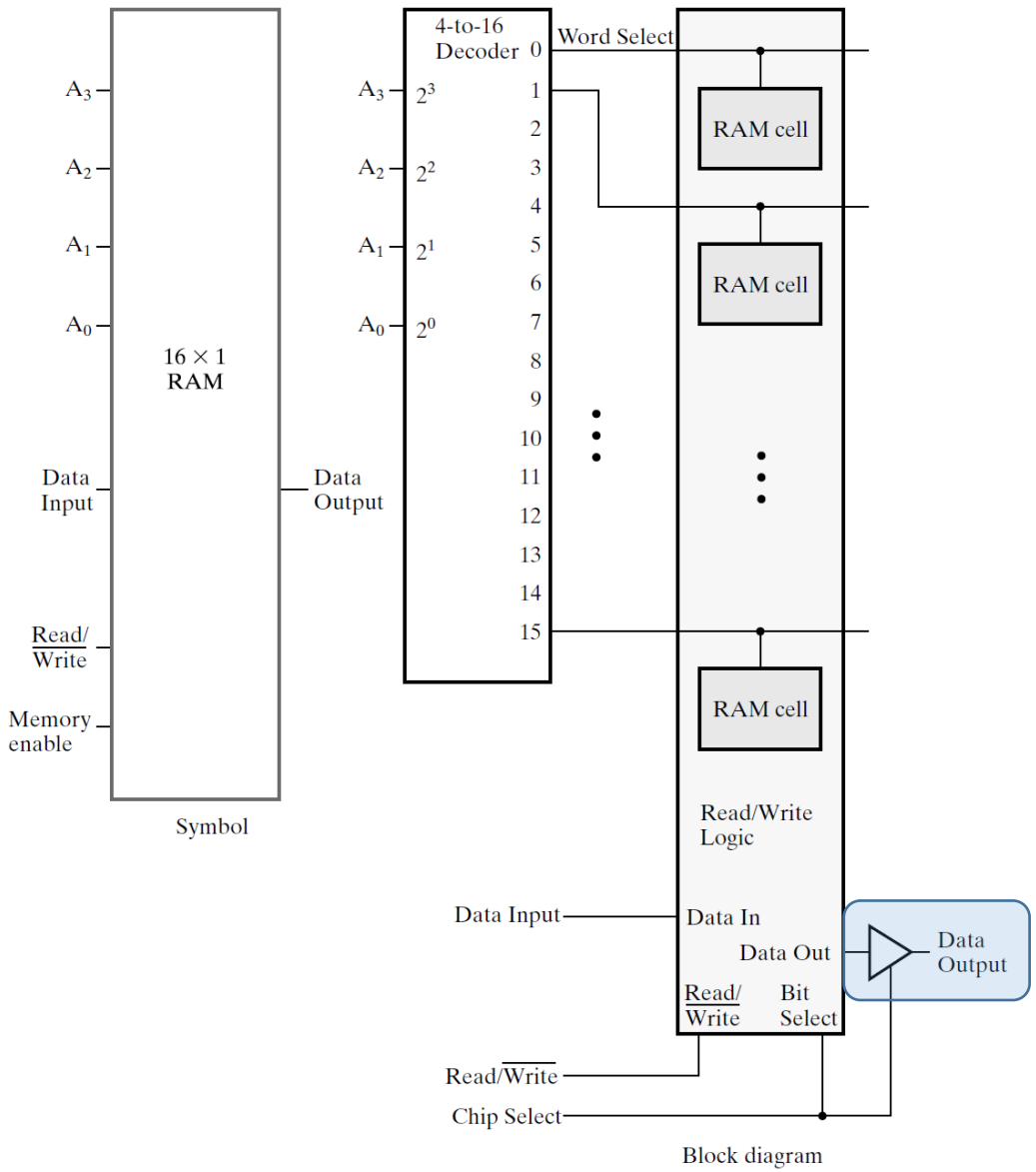
- To allow write to a cell, ‘Read’ must be 0 + ‘Bit Select’ set to 1
  - Only one word is written at a time (only one ‘Word Select’ line is 1)
- The ‘Word Select’ also controls the reading of the RAM cells, using shared Read Logic.
  - The read occurs regardless of value of Read/Write`



16 x 1-Bit SRAM Chip

- Let's look at a block diagram of a 16x1 RAM chip
  - 'Chip Select' at the chip level
  - 16 RAM cells
  - 16 Word Select lines
  - To address a single line at a time, 4-to-16-line decoder is used

- Why do we need a three-state buffer prior to 'Data Output'?



Coincident Selection

- Let's look at decoder complexity
  - A decoder w/ *k* inputs to 2^k outputs requires 2^k AND gates
  - If # of words per bit slice is large, # of RAM cells sharing a read/write circuitry becomes large

- These overhead can be reduced by employing two decoders w/ **coincident selection**

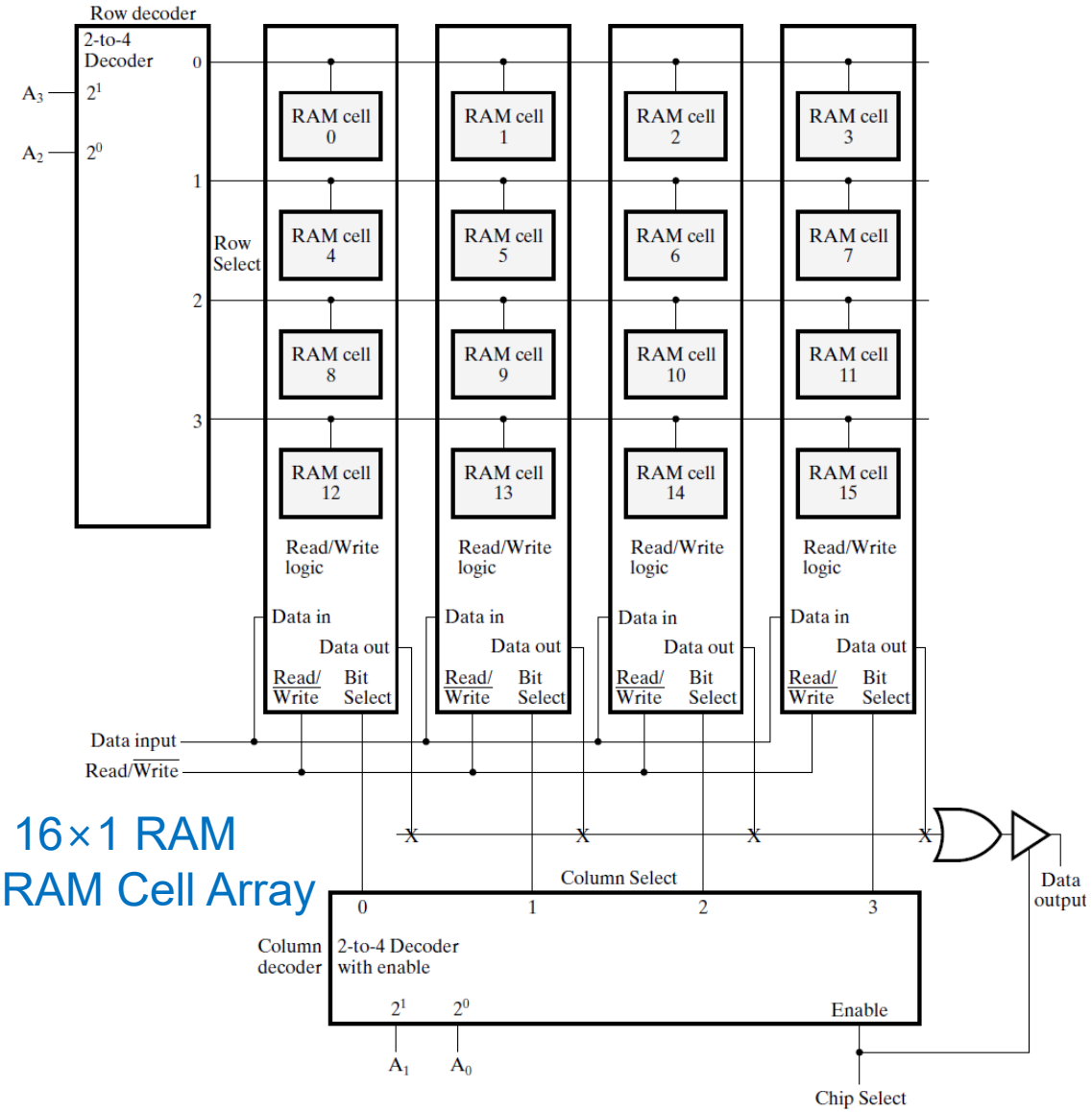
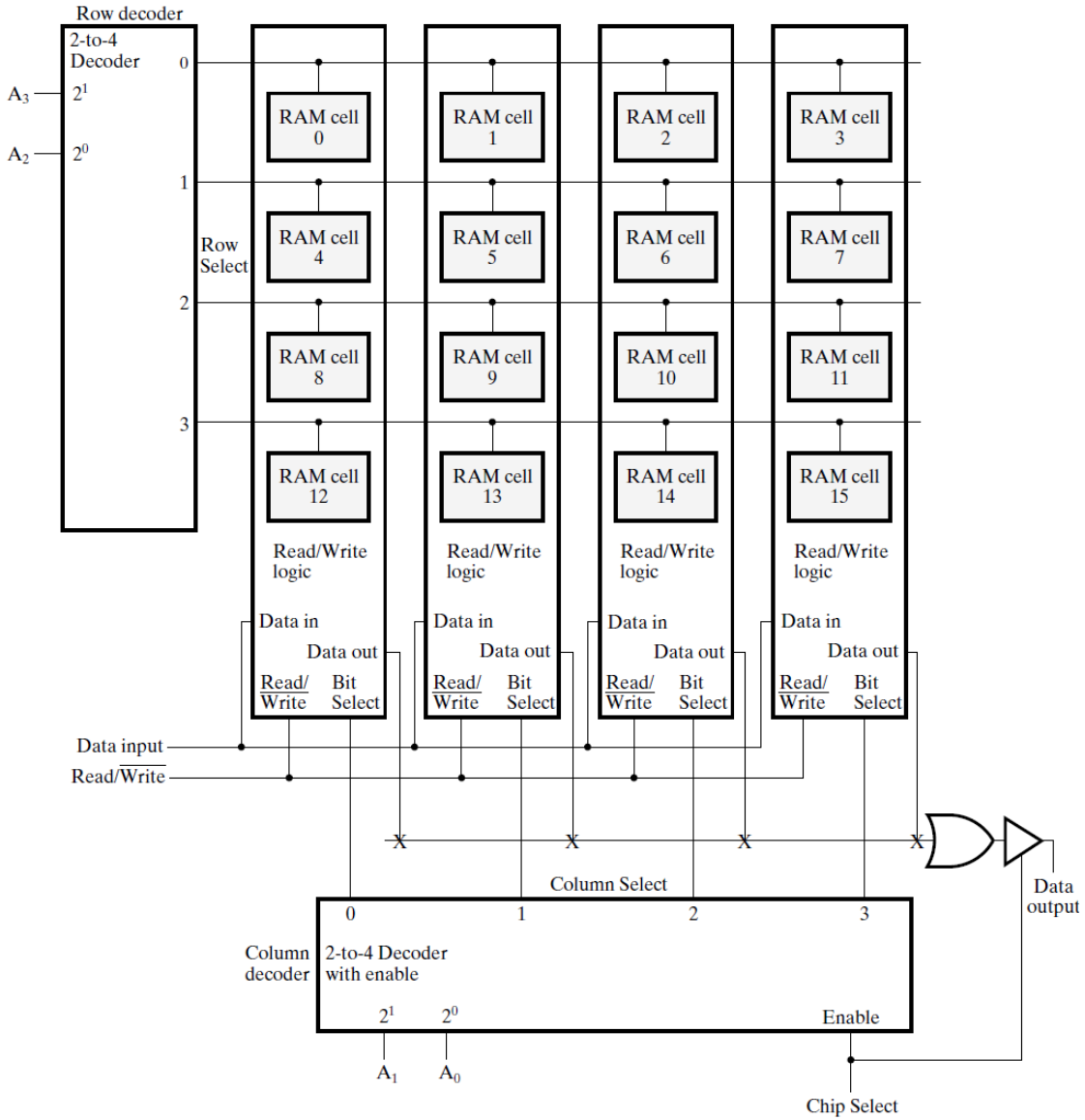


Diagram of a 16×1 RAM
Using a 4×4 RAM Cell Array

Row/Column Selection

- Now, two *k*/2-input decoders can be used
  - One decoder controls the word select lines and the other controls the bit select lines
  - Instead of 4-to-16, we need two 2-to-4 line decoders
  - ‘Word Select’ is separated ‘Row Select’ and ‘Column Select’



Multiple Bits per Word

- Let's consider eight words of two bits each
  - Row decoding is unchanged
  - The only changes are in the column and output logic
  - Only one address bit is handles by the column decoder

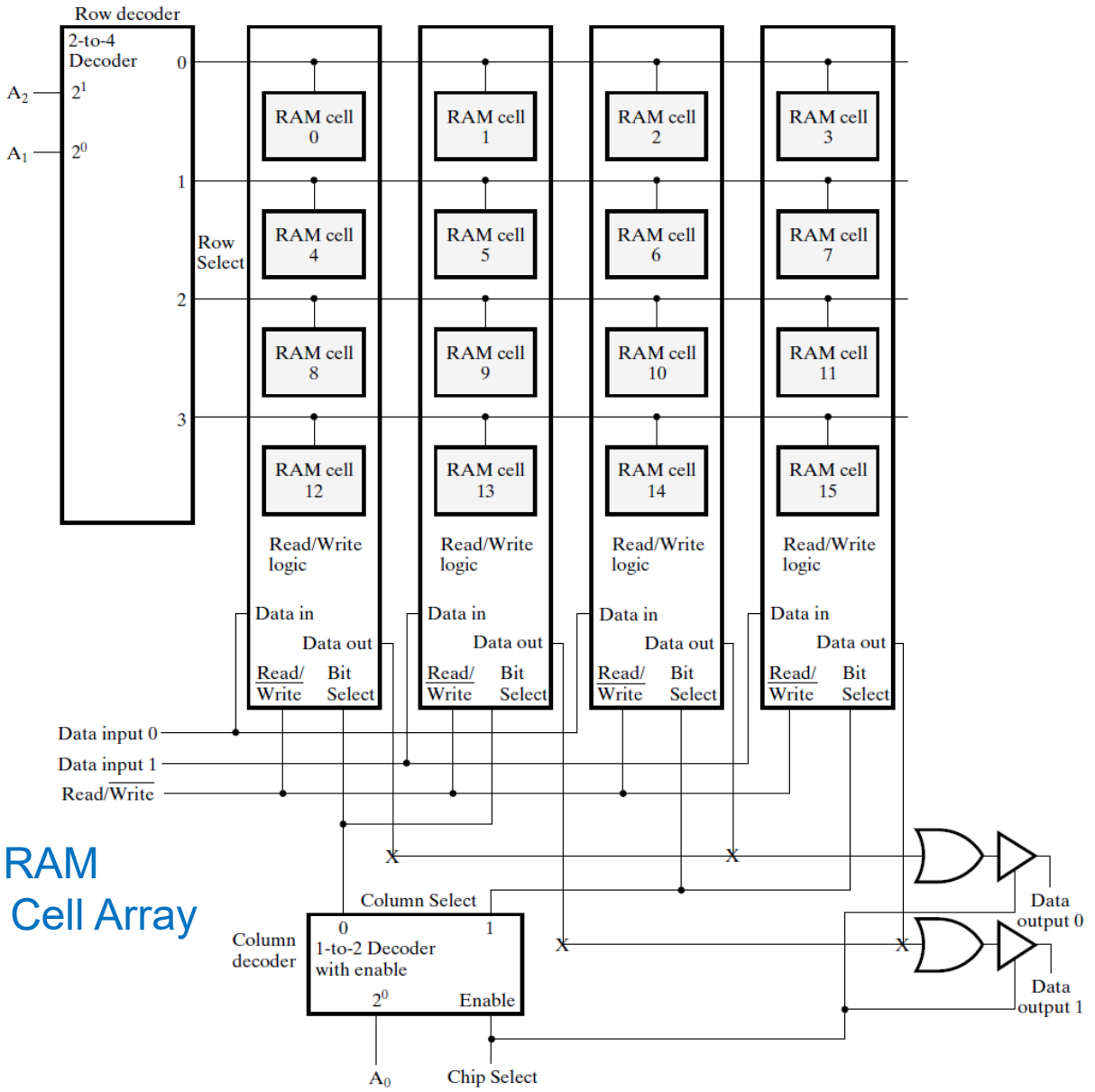


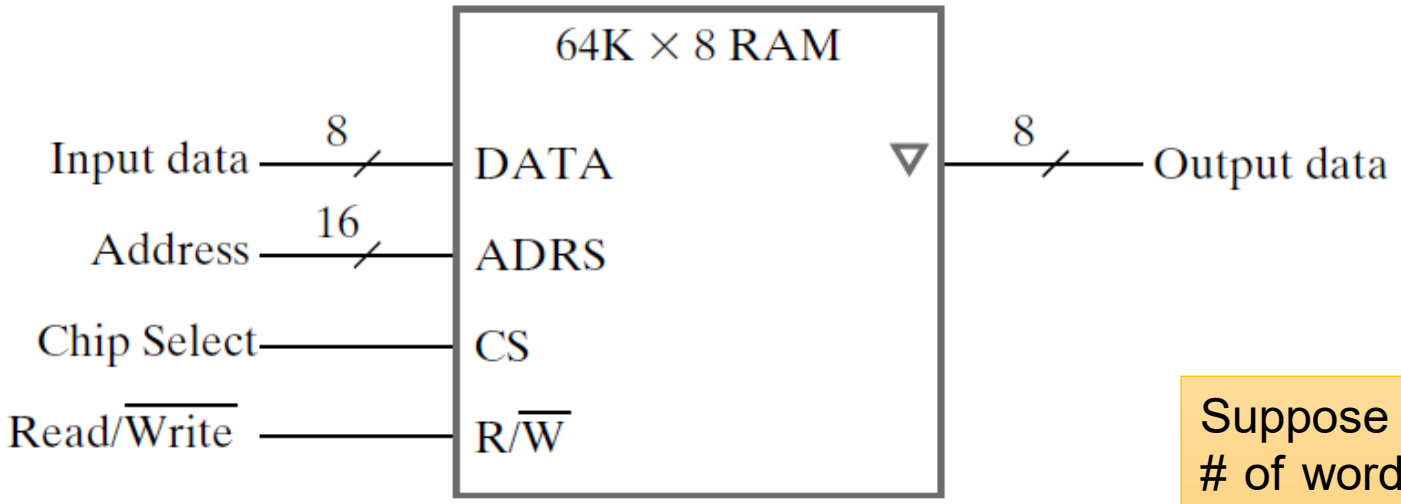
Diagram of a 8×2 RAM
Using a 4×4 RAM Cell Array

Array of SRAM ICs

- Integrated-circuit RAM chips are available in a variety of sizes
  - Sometimes, it is necessary to combine a number of chips in an array to form the required size of memory
  - An increase in the # of words requires that we increase the address length
  - An increase in the # of bits per word requires that we increase the # of data input and output lines

Representation of RAM chip

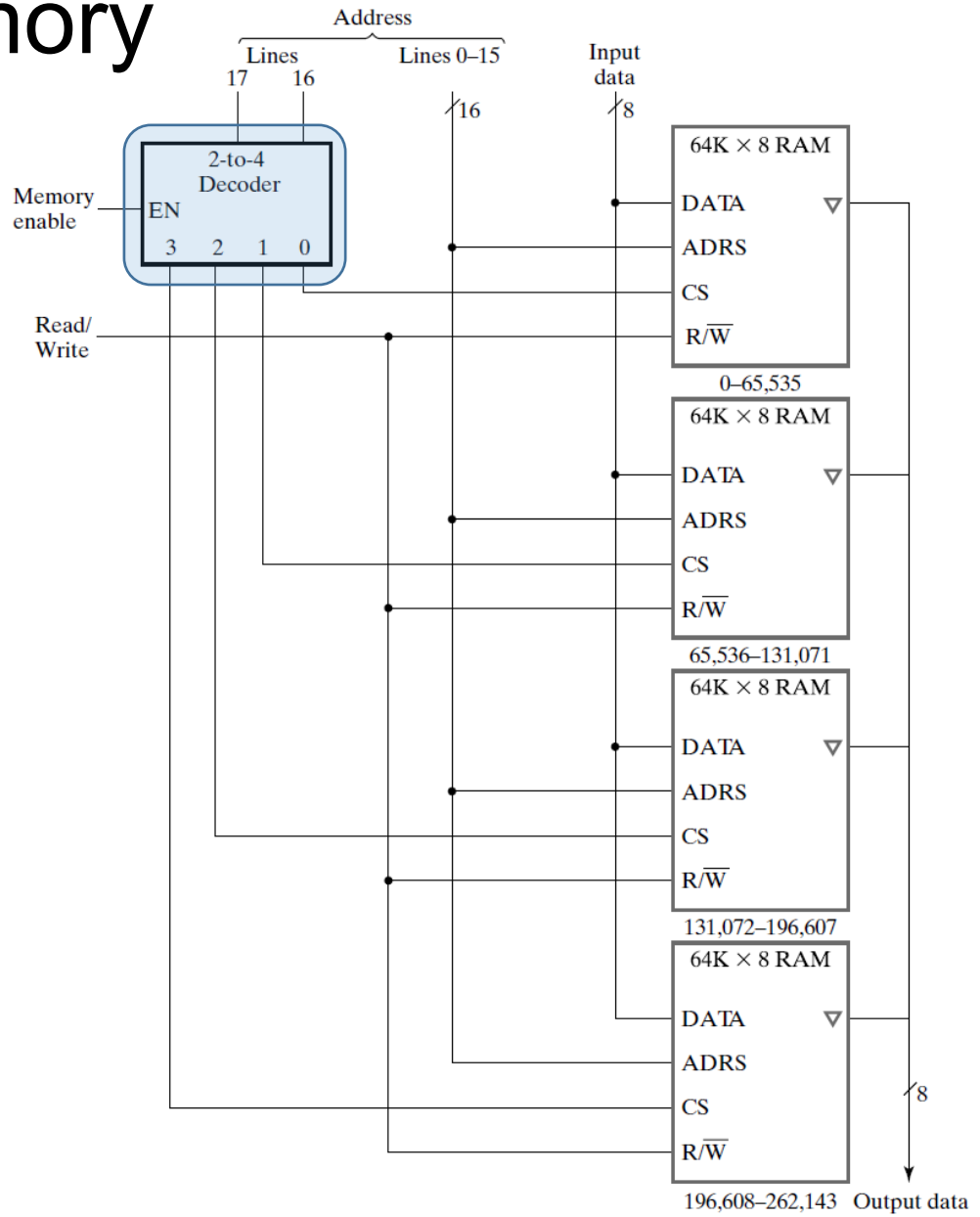
- The capacity of this chip is 64K words of 8 bits each
  - 16-bit address and 8 input/output lines are required
  - CS (Chip Select) input selects the particular RAM chip



Suppose that we want to increase # of words in the memory by using two or more RAM chips

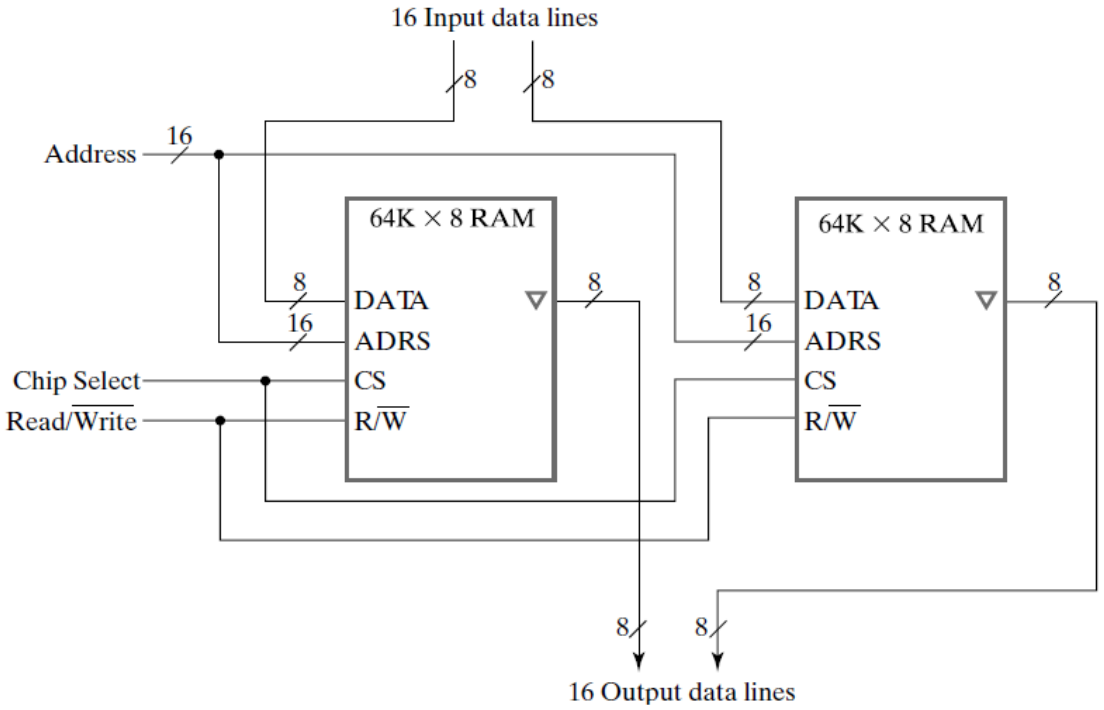
Expanding the Capacity of Memory

- Consider constructing a 256K x 8 RAM w/ four 64K x 8 RAM chips
  - The eight data input lines go to all the chips
  - Three-state outputs can be connected together to from the eight common data output lines
  - The 256K-word memory requires an 18-bit address
  - The two most significant bits are applied to a 2×4 decoder



Expanding the Capacity of Memory

- Consider constructing a 64K x 16 RAM w/ two 64K x 8 RAM chips
  - Doubling the # of bits per word
  - The 16 data input and output lines are split btw two chips
  - Both receive the same 16-bit address and the common CS and R/W` control inputs

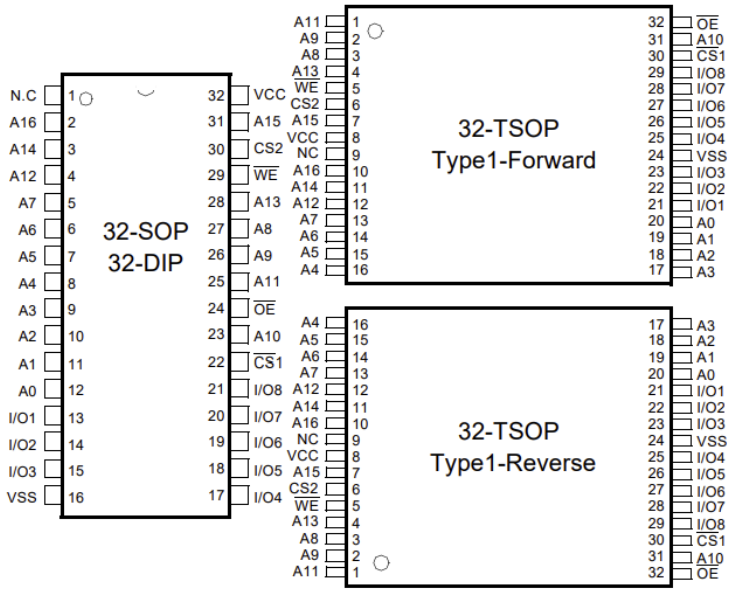


These two techniques just described may be combined to assemble an array of identical chips into a large-capacity memory

How can we reduce # of pins on the chip package? (can we combine input and output data lines?)

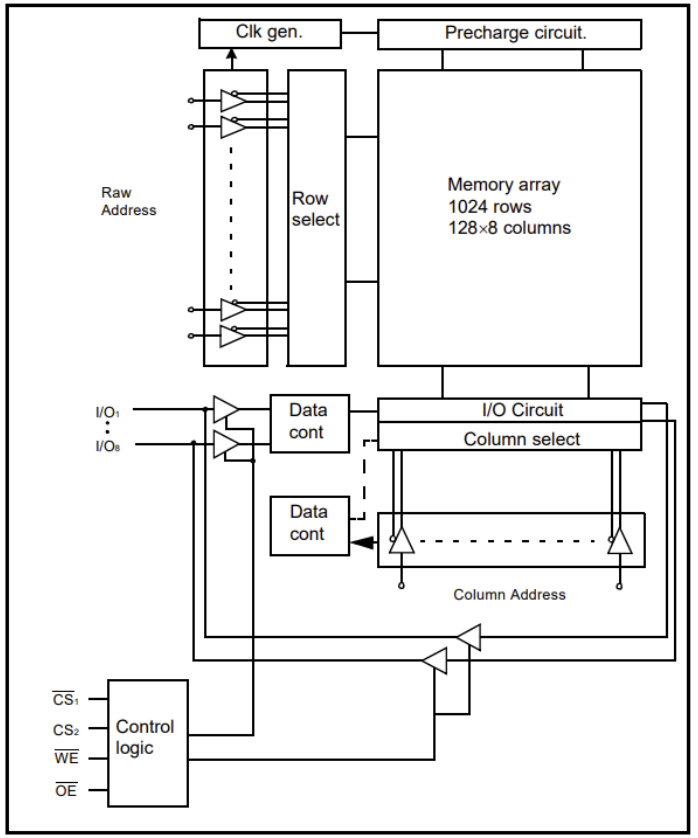
Example: SRAM (CMOS SRAM K6T1008C2E Family)

PIN DESCRIPTION



Name	Function
$\overline{CS}_1$, CS_2	Chip Select Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
I/O_1~I/O_8	Data Inputs/Outputs
A_0~A_{16}	Address Inputs
Vcc	Power
Vss	Ground
N.C.	No Connection

FUNCTIONAL BLOCK DIAGRAM



The attached datasheets are provided by SAMSUNG Electronics. SAMSUNG Electronics CO., LTD. reserves the right to change the specifications and products. SAMSUNG Electronics will answer to your questions. If you have any questions, please contact the SAMSUNG branch offices.

Example: SRAM (CMOS SRAM K6T1008C2E Family)

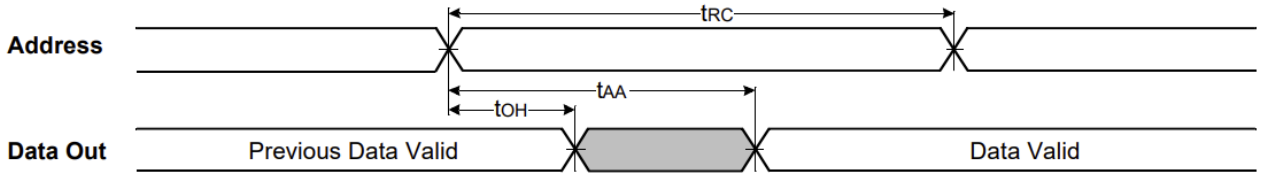
AC CHARACTERISTICS (V_{CC}=4.5~5.5V, Commercial Product: T_A=0 to 70°C, Industrial Product: T_A=-40 to 85°C)

Parameter List		Symbol	Speed Bins				Units
			55ns		70ns		
			Min	Max	Min	Max	
Read	Read Cycle Time	t_{RC}	55	-	70	-	ns
	Address Access Time	t_{AA}	-	55	-	70	ns
	Chip Select to Output	t_{CO}	-	55	-	70	ns
	Output Enable to Valid Output	t_{OE}	-	25	-	35	ns
	Chip Select to Low-Z Output	t_{LZ}	10	-	10	-	ns
	Output Enable to Low-Z Output	t_{OLZ}	5	-	5	-	ns
	Chip Disable to High-Z Output	t_{HZ}	0	20	0	25	ns
	Output Disable to High-Z Output	t_{OHZ}	0	20	0	25	ns
	Output Hold from Address Change	t_{OH}	10	-	10	-	ns
Write	Write Cycle Time	t_{WC}	55	-	70	-	ns
	Chip Select to End of Write	t_{CW}	45	-	60	-	ns
	Address Set-up Time	t_{AS}	0	-	0	-	ns
	Address Valid to End of Write	t_{AW}	45	-	60	-	ns
	Write Pulse Width	t_{WP}	40	-	50	-	ns
	Write Recovery Time	t_{WR}	0	-	0	-	ns
	Write to Output High-Z	t_{WHZ}	0	20	0	25	ns
	Data to Write Time Overlap	t_{DW}	20	-	25	-	ns
	Data Hold from Write Time	t_{DH}	0	-	0	-	ns
	End Write to Output Low-Z	t_{OW}	5	-	5	-	ns

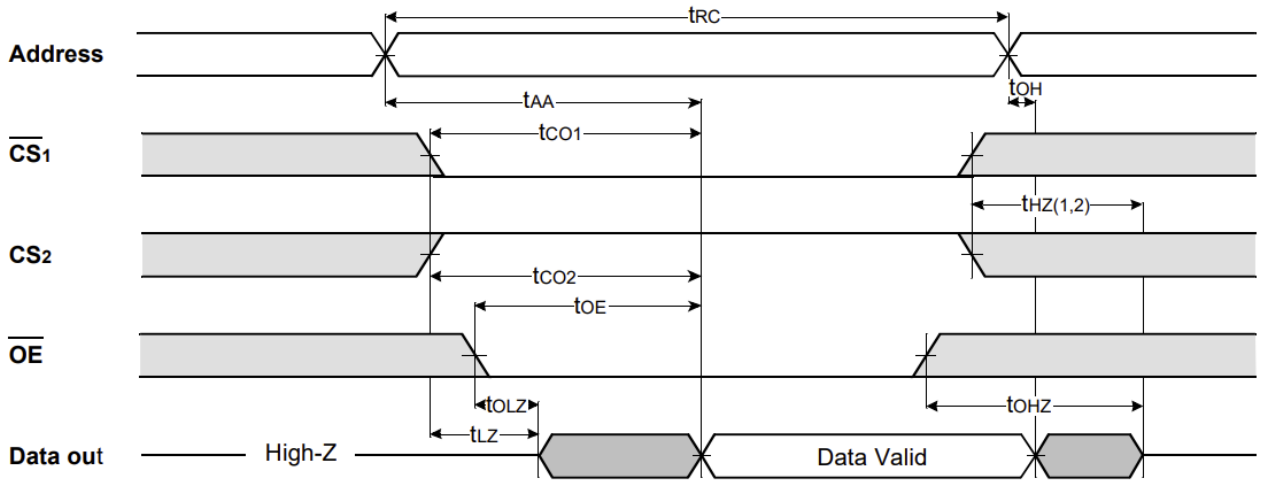
Example: SRAM (CMOS SRAM K6T1008C2E Family)

TIMMING DIAGRAMS

TIMING WAVEFORM OF READ CYCLE(1) (Address Controlled, $\overline{CS1}$=$\overline{OE}$=V_{IL}, CS2=$\overline{WE}$=V_{IH})



TIMING WAVEFORM OF READ CYCLE(2) ($\overline{WE}$=V_{IH})

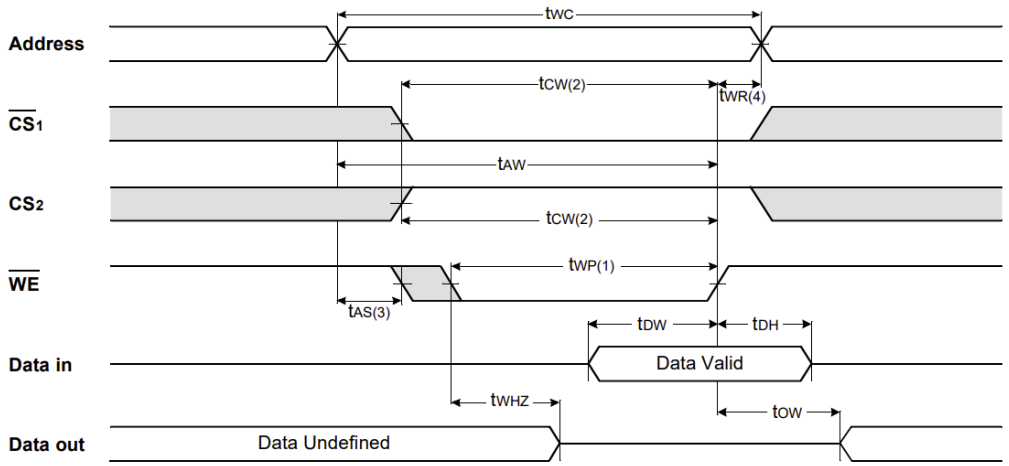


NOTES (READ CYCLE)

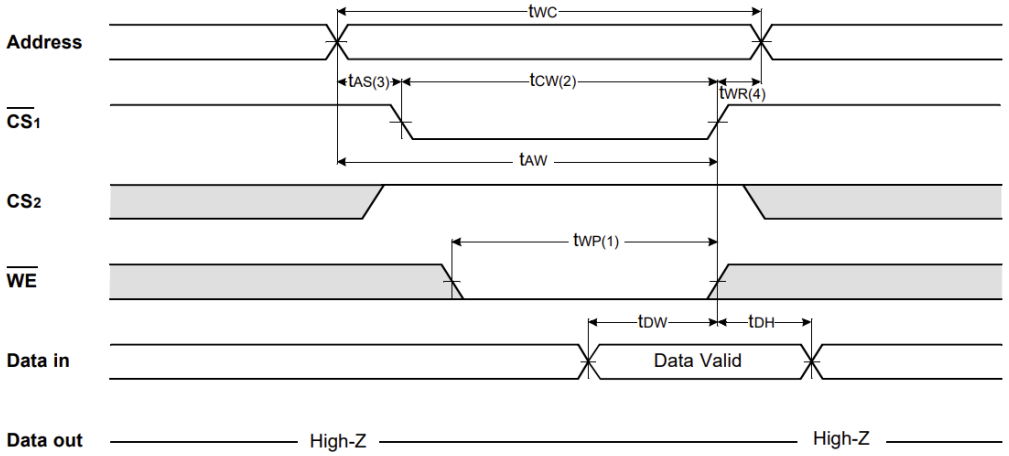
1. t_{HZ} and t_{OHZ} are defined as the time at which the outputs achieve the open circuit conditions and are not referenced to output voltage levels.
2. At any given temperature and voltage condition, t_{HZ}(Max.) is less than t_{LZ}(Min.) both for a given device and from device to device interconnection.

Example: SRAM (CMOS SRAM K6T1008C2E Family)

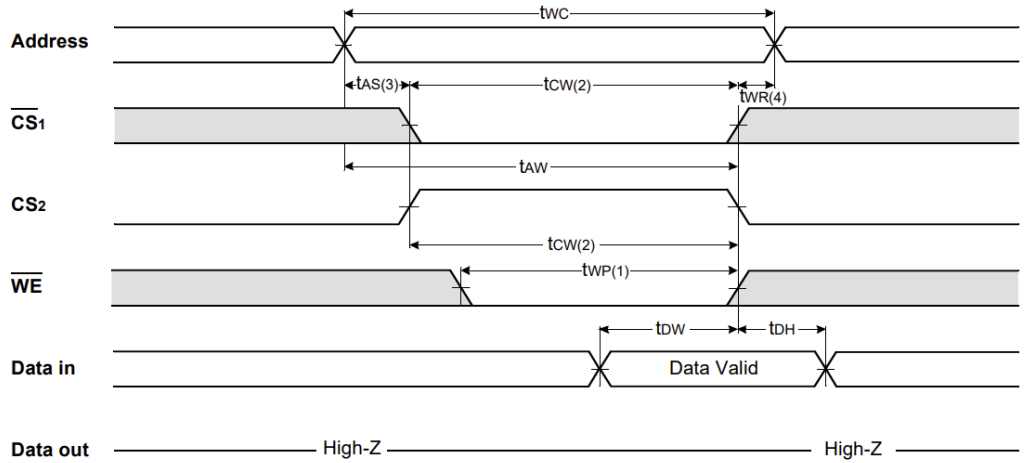
TIMING WAVEFORM OF WRITE CYCLE(1) ($\overline{WE}$ Controlled)



TIMING WAVEFORM OF WRITE CYCLE(2) ($\overline{CS_1}$ Controlled)



TIMING WAVEFORM OF WRITE CYCLE(3) (CS_2 Controlled)



NOTES (WRITE CYCLE)

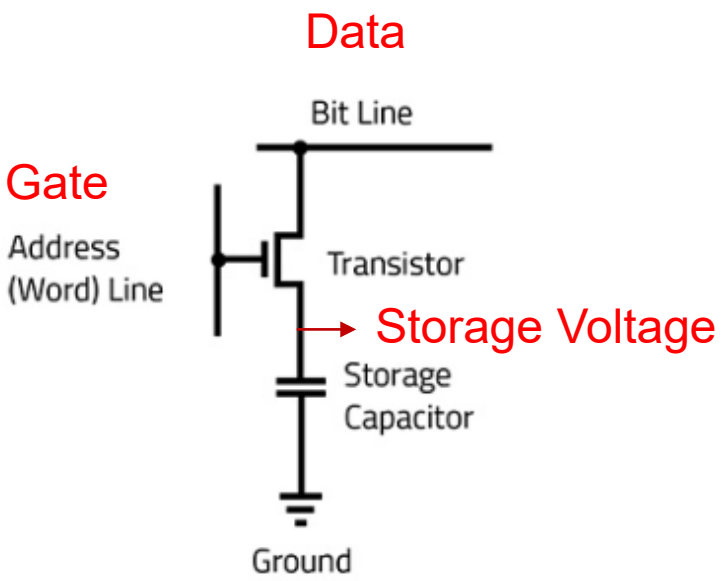
1. A write occurs during the overlap of a low $\overline{CS_1}$, a high CS_2 and a low $\overline{WE}$. A write begins at the latest transition among $\overline{CS_1}$ goes low, CS_2 going high and $\overline{WE}$ going low: A write end at the earliest transition among $\overline{CS_1}$ going high, CS_2 going low and $\overline{WE}$ going high, t_{WP} is measured from the begining of write to the end of write.
2. t_{CW} is measured from the $\overline{CS_1}$ going low or CS_2 going high to the end of write.
3. t_{AS} is measured from the address valid to the beginning of write.
4. t_{WR} is measured from the end of write to the address change. t_{WR1} applied in case a write ends as $\overline{CS_1}$ or $\overline{WE}$ going high t_{WR2} applied in case a write ends as CS_2 going to low.

DRAM ICs

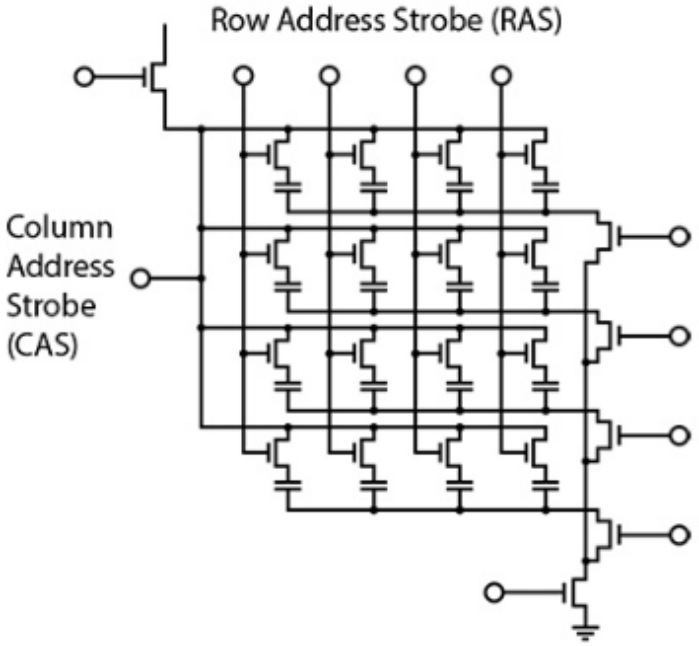
- It provides high storage capacity at low cost
  - DRAM dominates the high-capacity memory applications
  - Logically it is similar to SRAM
  - As "**dynamic**" implies, the storage of information is inherently temporary

Thus, it requires a periodic "**refresh**"

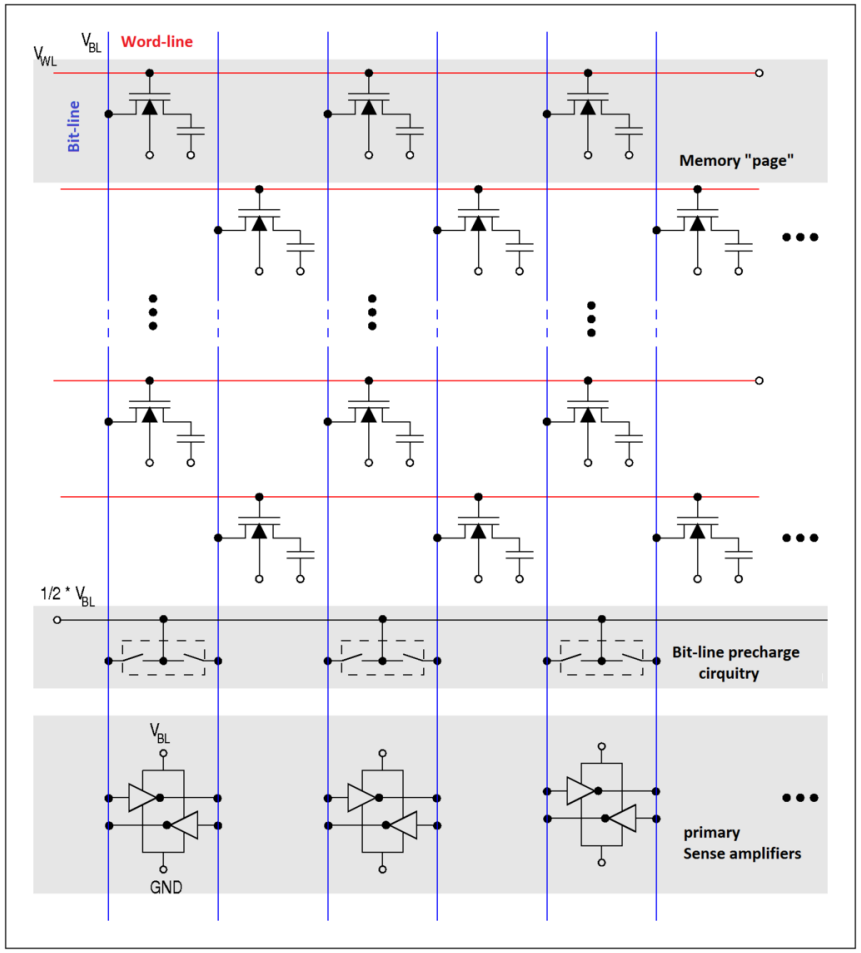
DRAM Cell



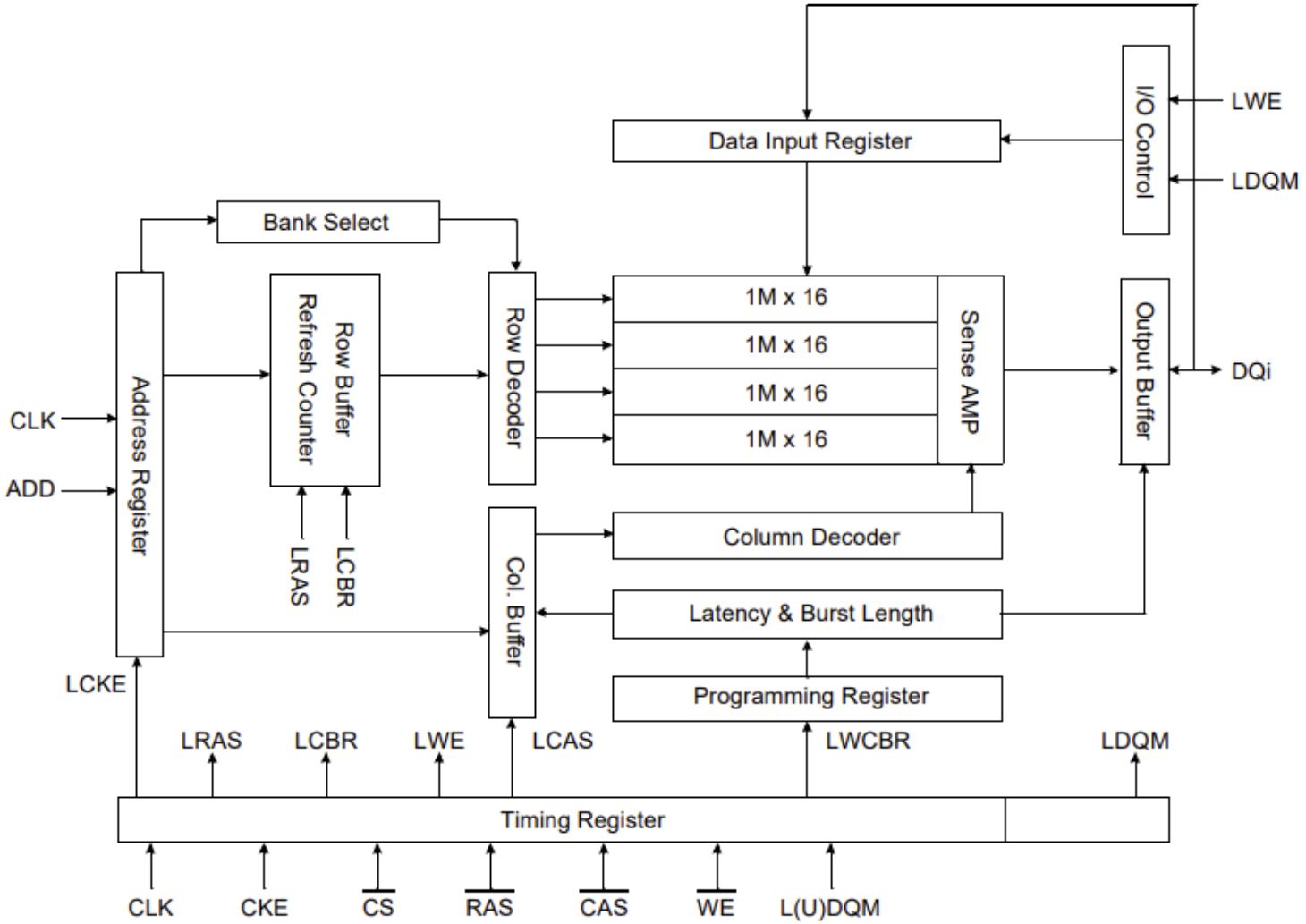
Single Memory Cell



Memory Cell Array



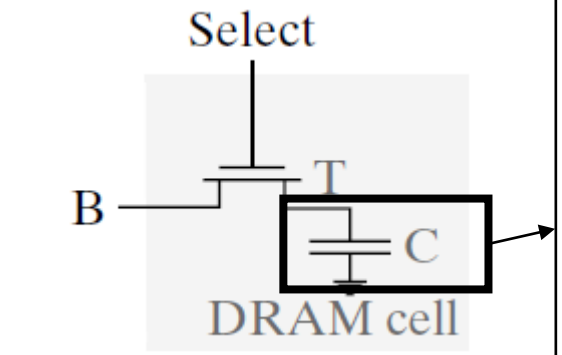
SDRAM Block Diagram

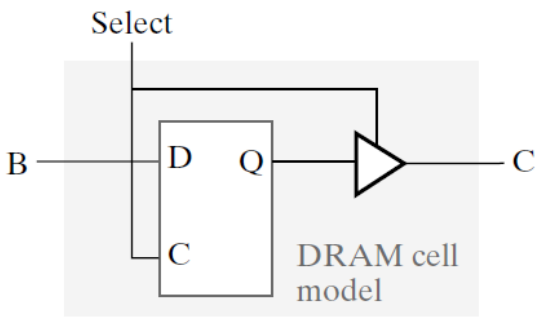


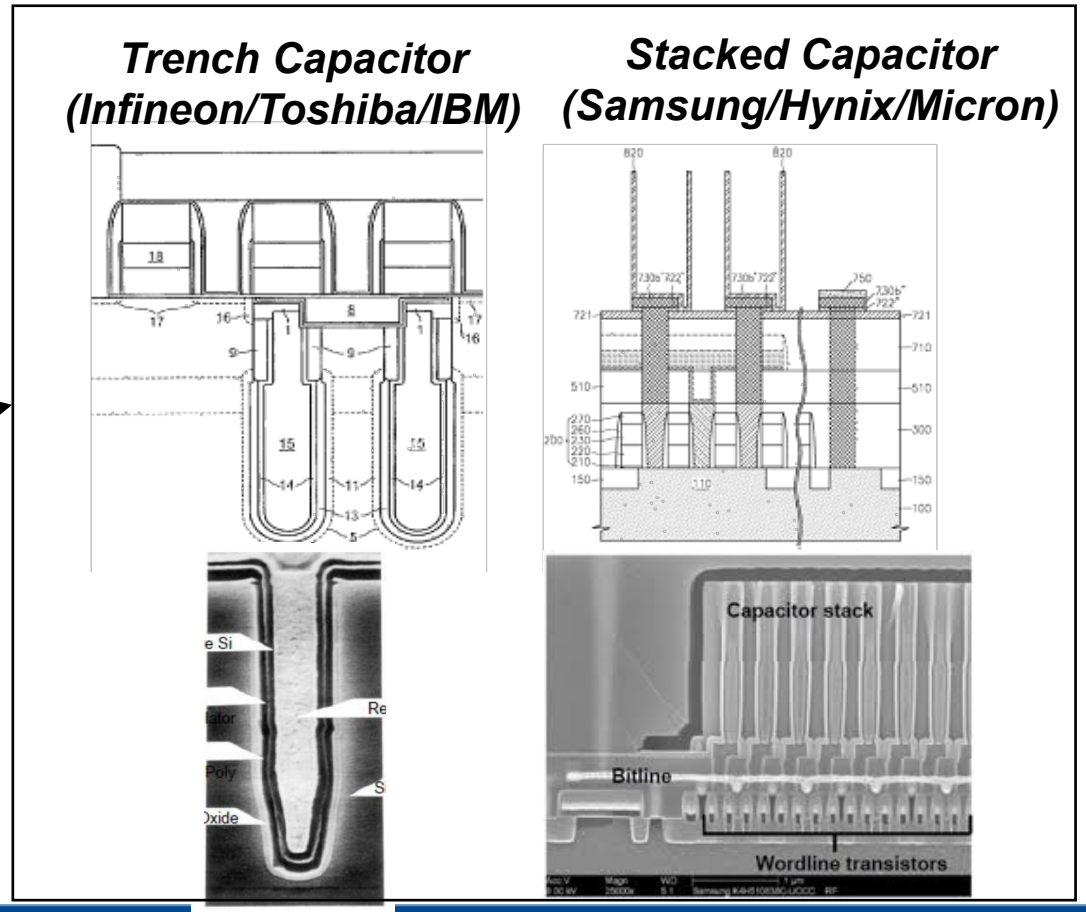
*** Samsung Electronics reserves the right to change products or specification without notice.**

DRAM cell

- It consists of a capacitor C and a transistor T
  - The capacitor is used to store electrical charge
  - The transistor acts much like a switch

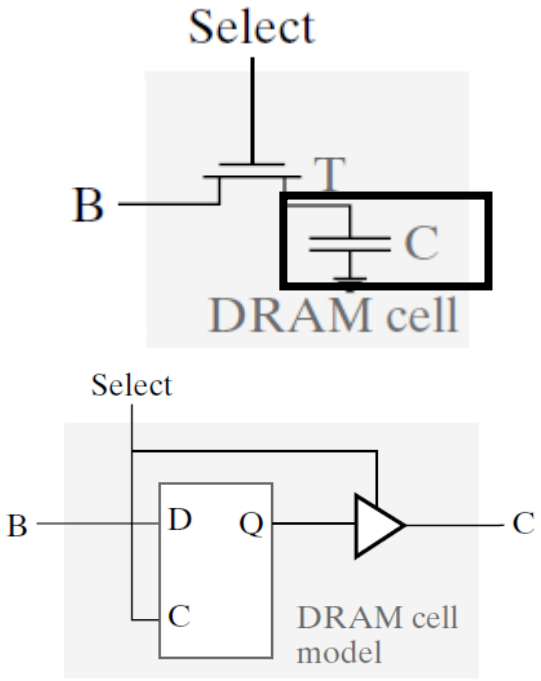






DRAM cell

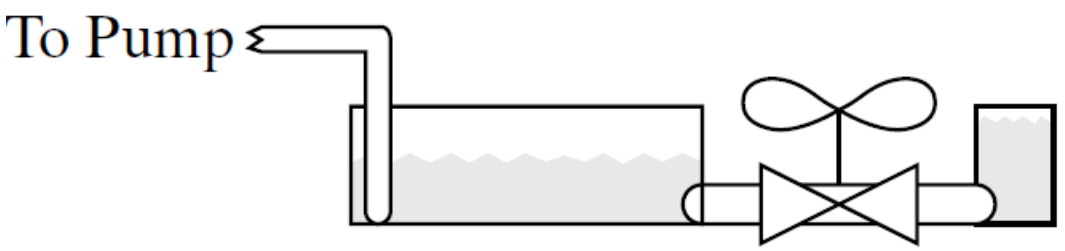
- It consists of a capacitor C and a transistor T
  - The capacitor is used to store electrical charge
  - The transistor acts much like a switch



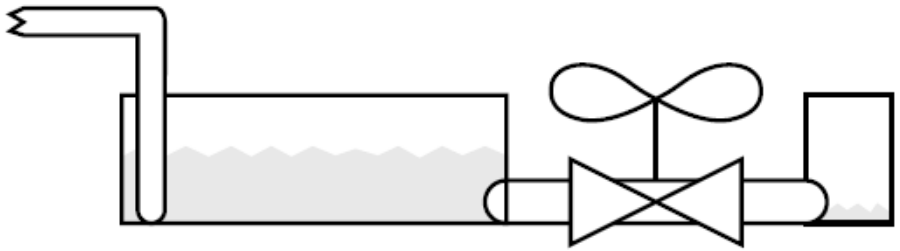
	Trench	Stacked
Structure	Deep trench	3-D stacked structure
Pros	Easy to downsize the chip size Easy to make a planar chip	Easy to verify the chip
Cons	Hard to make a deep trench in smaller manufacturing process, Hard to find flaws during manufacturing	Hard to make a planar chip
Company	Infineon, IBM, Toshiba	Samsung, Hynix, Micron

Hydraulic Analogy

- To understand the read/write operation, we use a hydraulic analogy with charge replaced by water
  - The capacitor then becomes a small storage tank
  - The transistor works as a valve



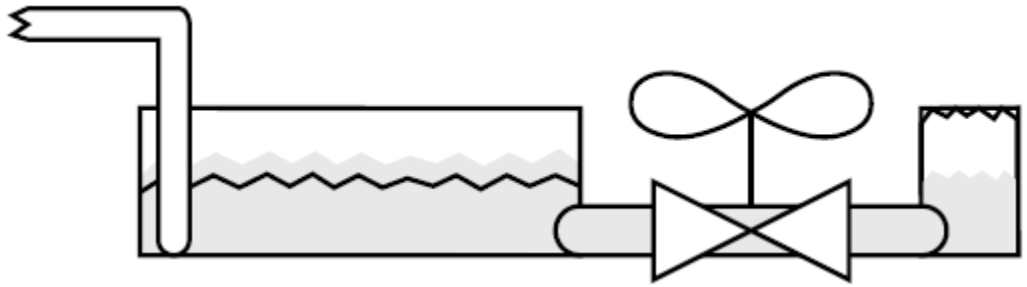
Logic ‘1’



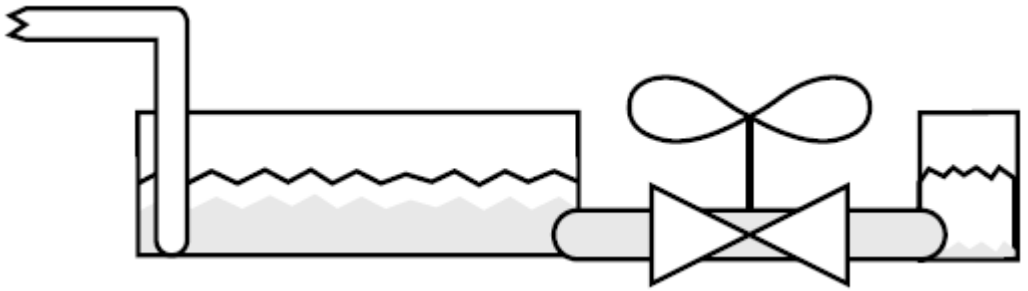
Logic ‘0’

Hydraulic Analogy

- Suppose we want to read a stored value 1
  - With the large tank at a known intermediate level, the valve is opened
  - Since the small storage tank is full, water flows from the small tank to the large tank (increasing the level of water surface)
  - This increase is observed as reading ‘1’



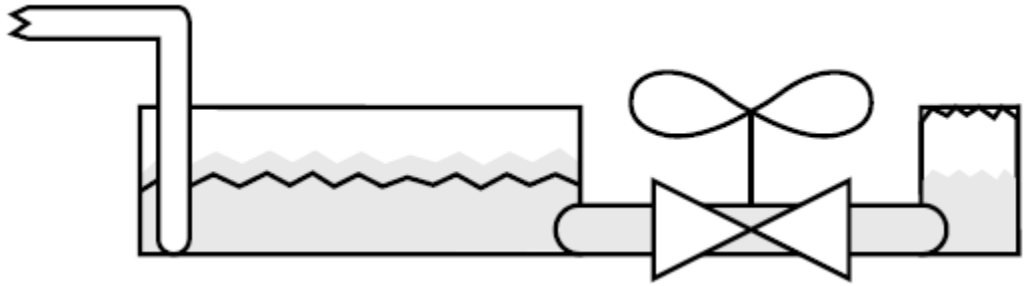
Reading Logic ‘1’



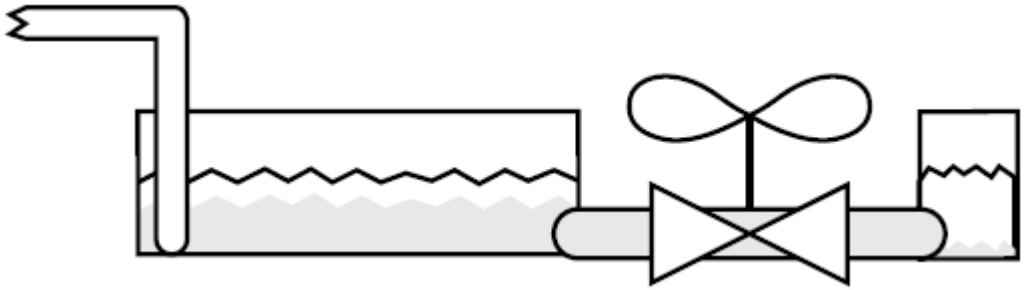
Reading Logic ‘0’

Hydraulic Analogy

- After the read operation, the storage tank now contains an inter-mediate value
  - Thus, read operation in DRAM is “**Destructive**”
  - We **must restore** the value **to its original level (by Sense Amplifier)**



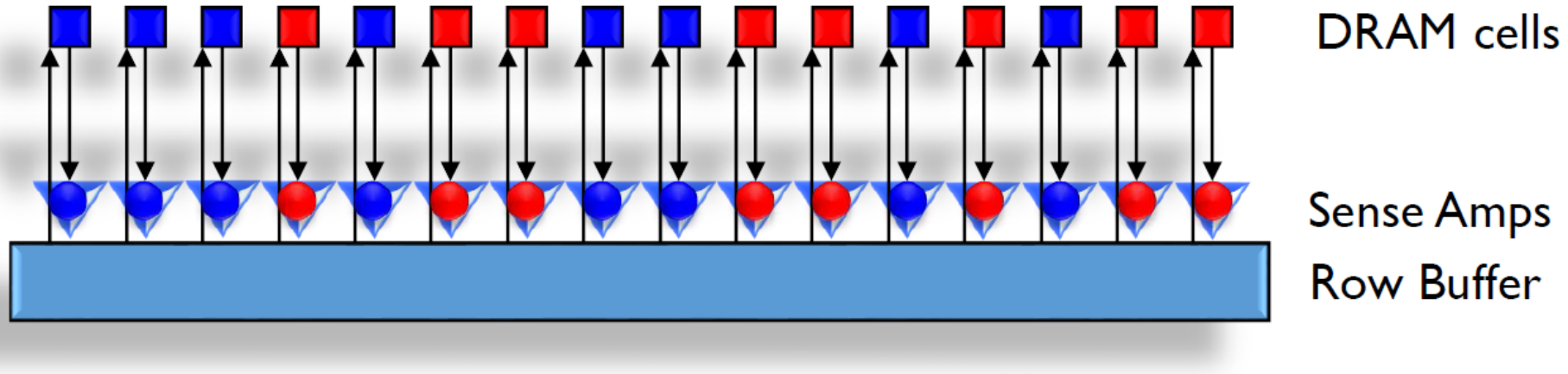
Reading Logic ‘1’



Reading Logic ‘0’

Re-write DRAM Cell After Read

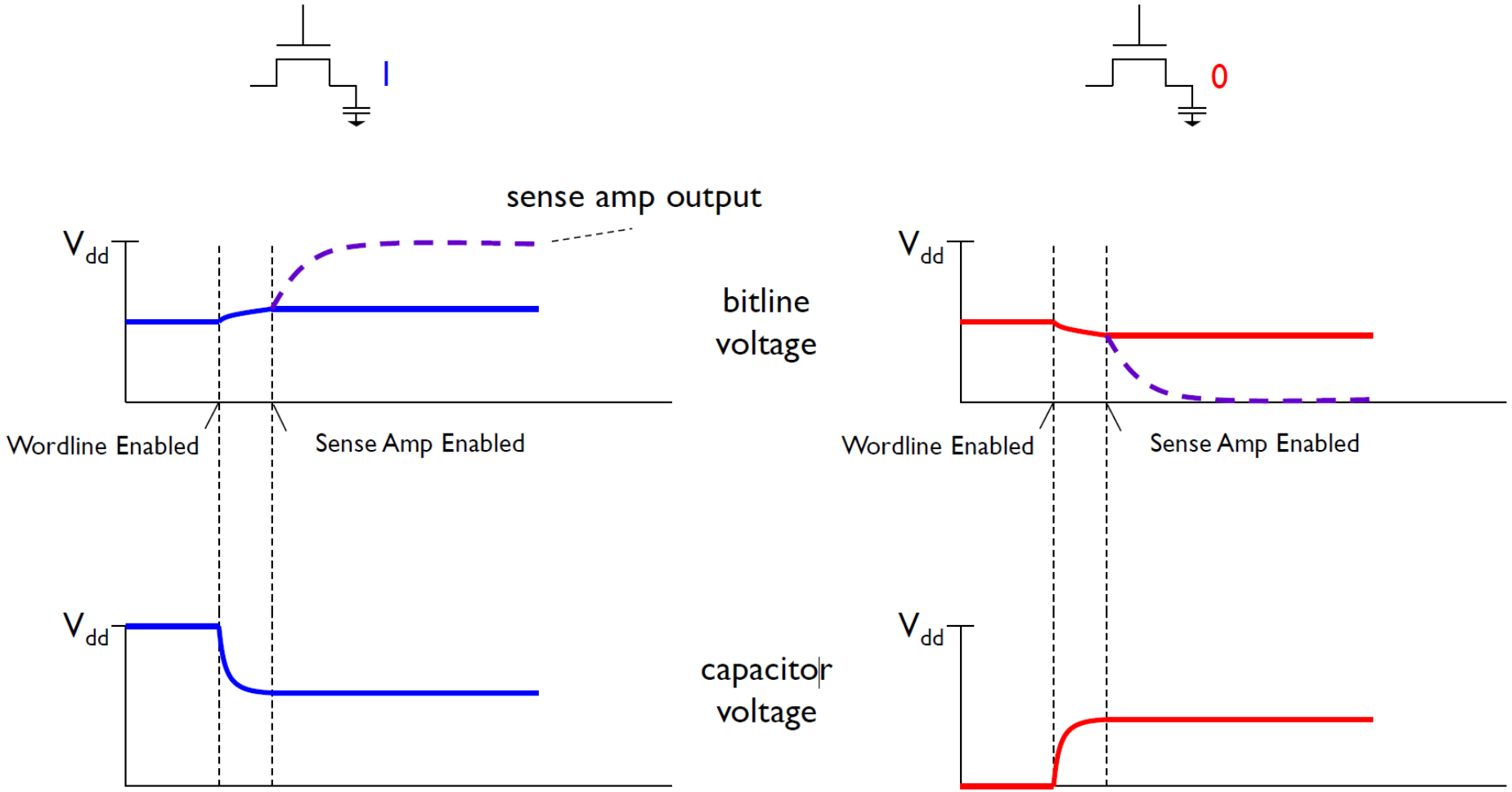
- After a read, the contents of DRAM cell are gone
  - But still "**safe**" in the row buffer
- Write bits back before doing another read
- Reading into buffer is slow, but reading buffer is fast
  - Try reading multiple lines from buffer (row-buffer hit)



Opening or **closing** a row

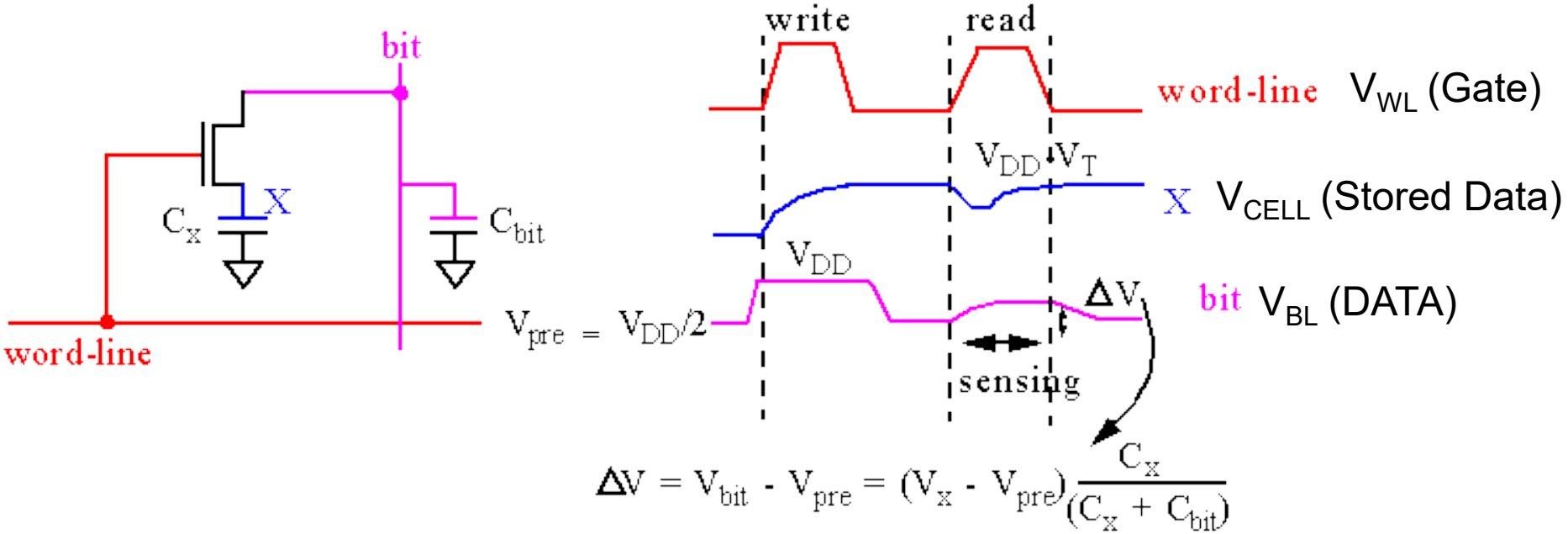
DRAM Destructive Read

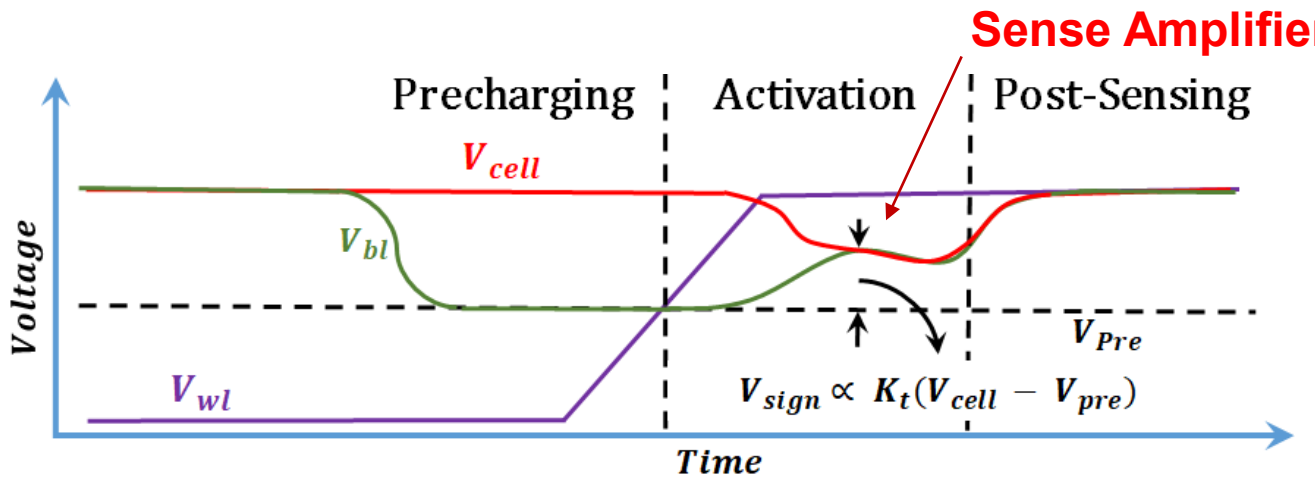
- After read of 0 or 1, cell contents close to 1/2



DRAM Operations

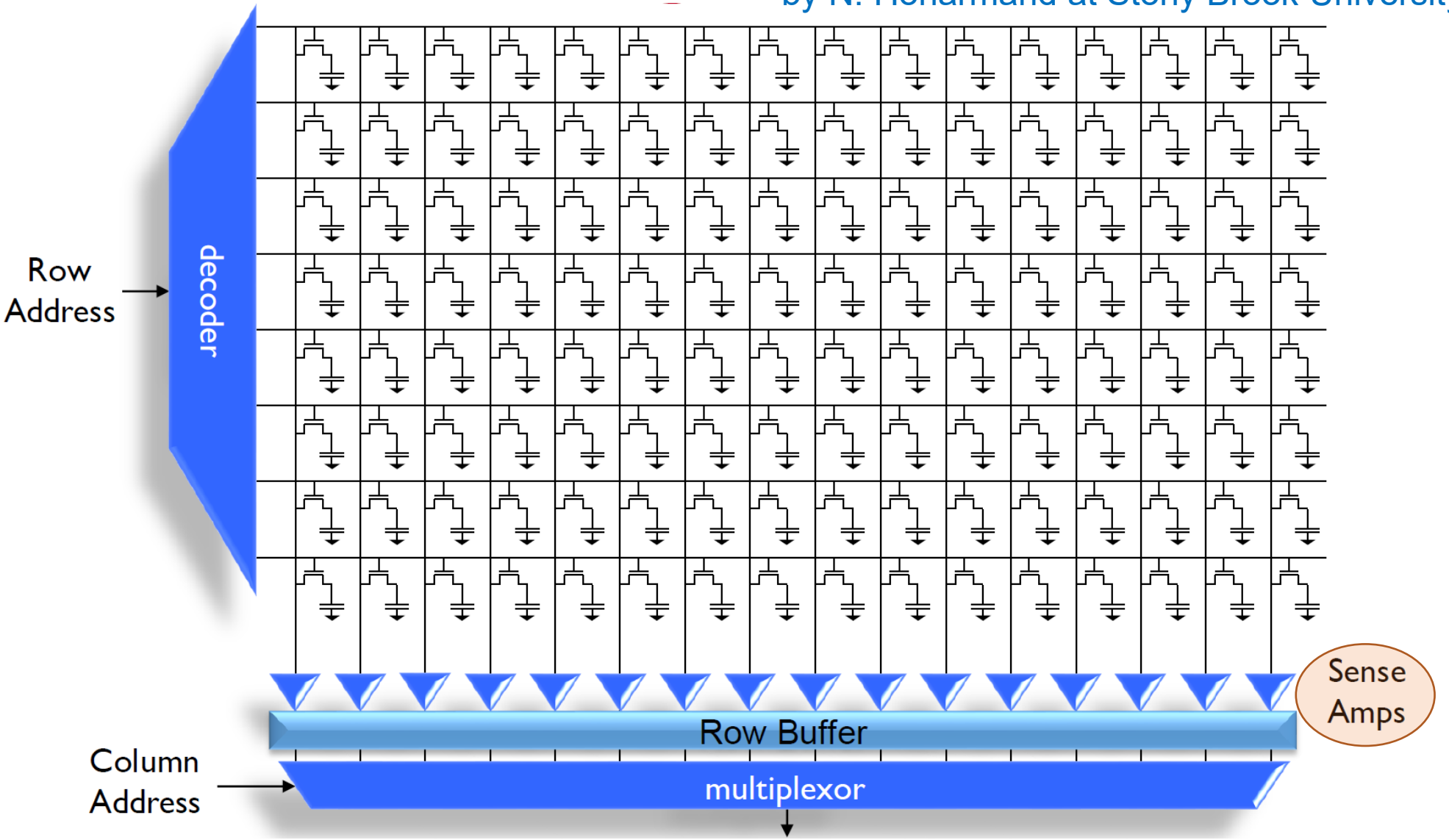
Figure from "http://ece-research.unm.edu/"





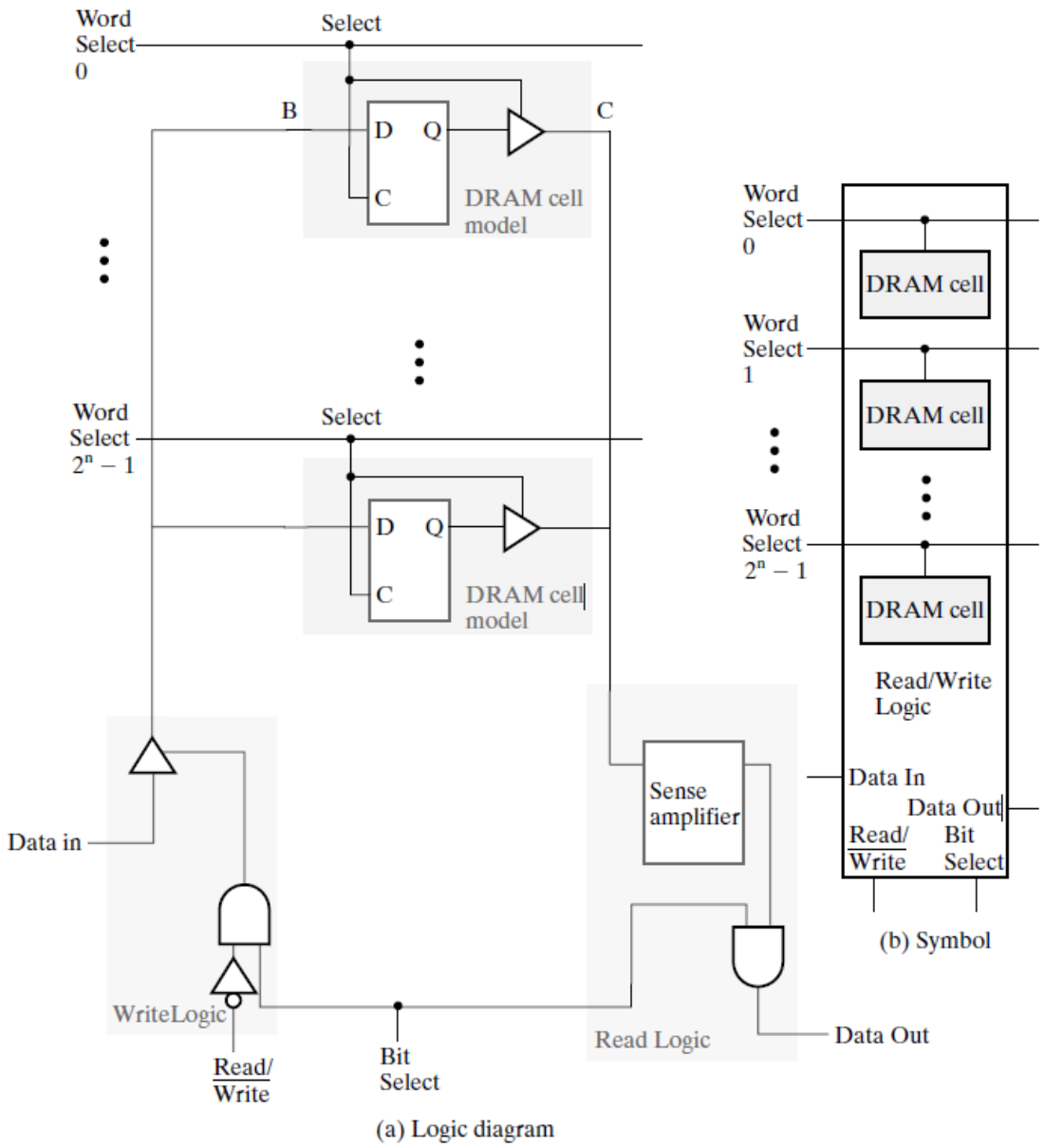
DRAM Chip Organization

Some slides are from “Main Memory and DRAM” by N. Honarmand at Stony Brook University



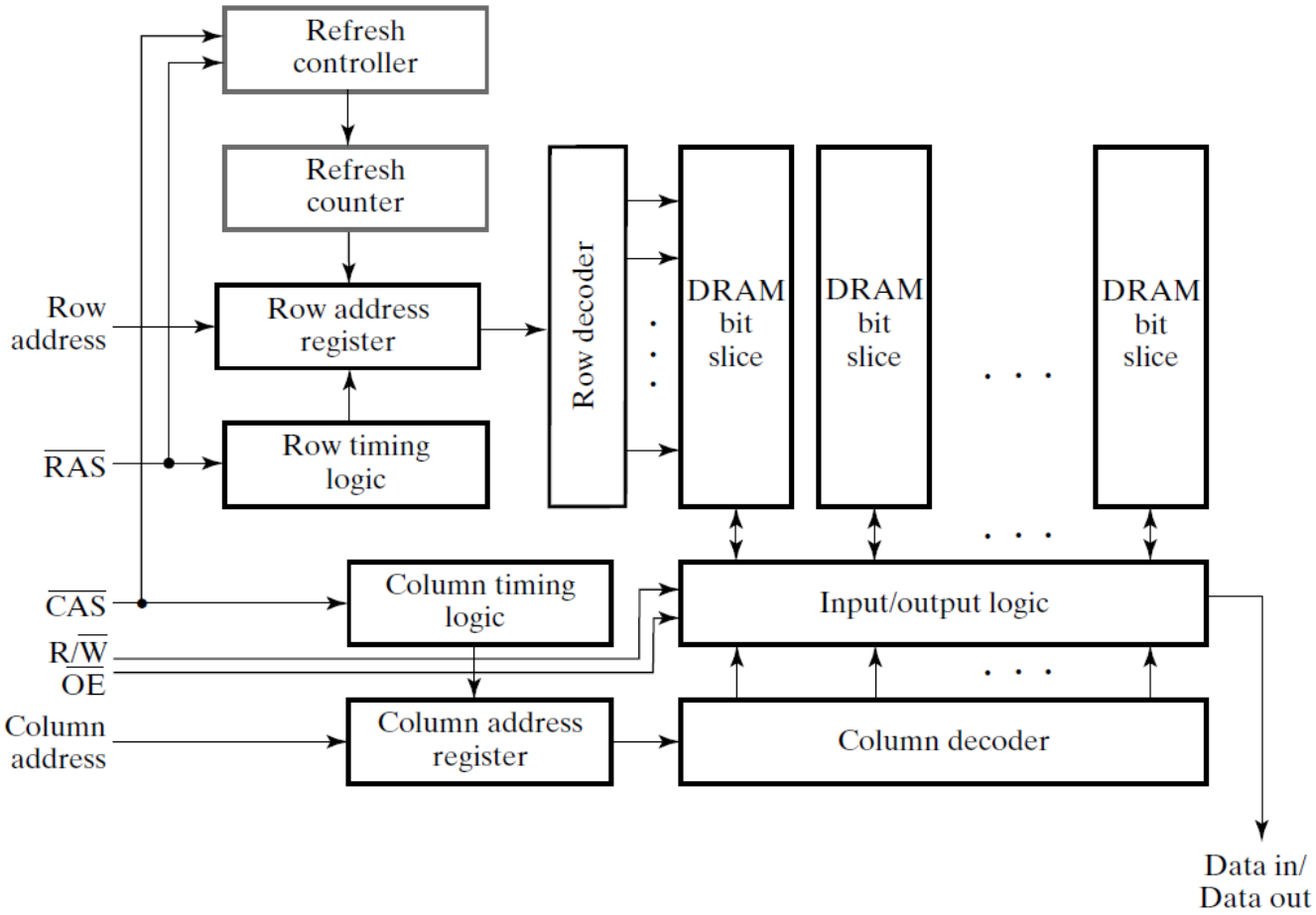
DRAM Bit Slice

- The bit slice model of DRAM is similar to that of SRAM
  - The main difference is the cost of a cell design
  - A SRAM cell is roughly three times larger than a DRAM cell

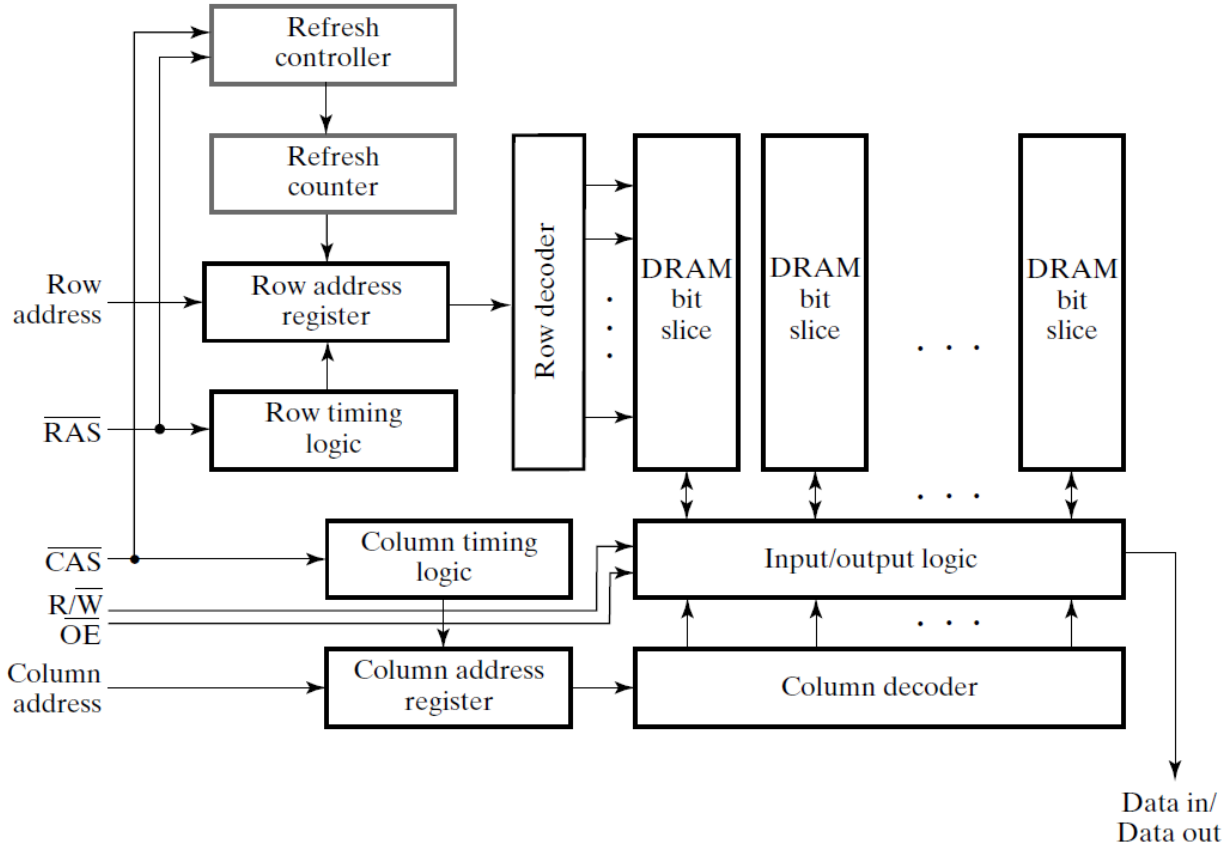


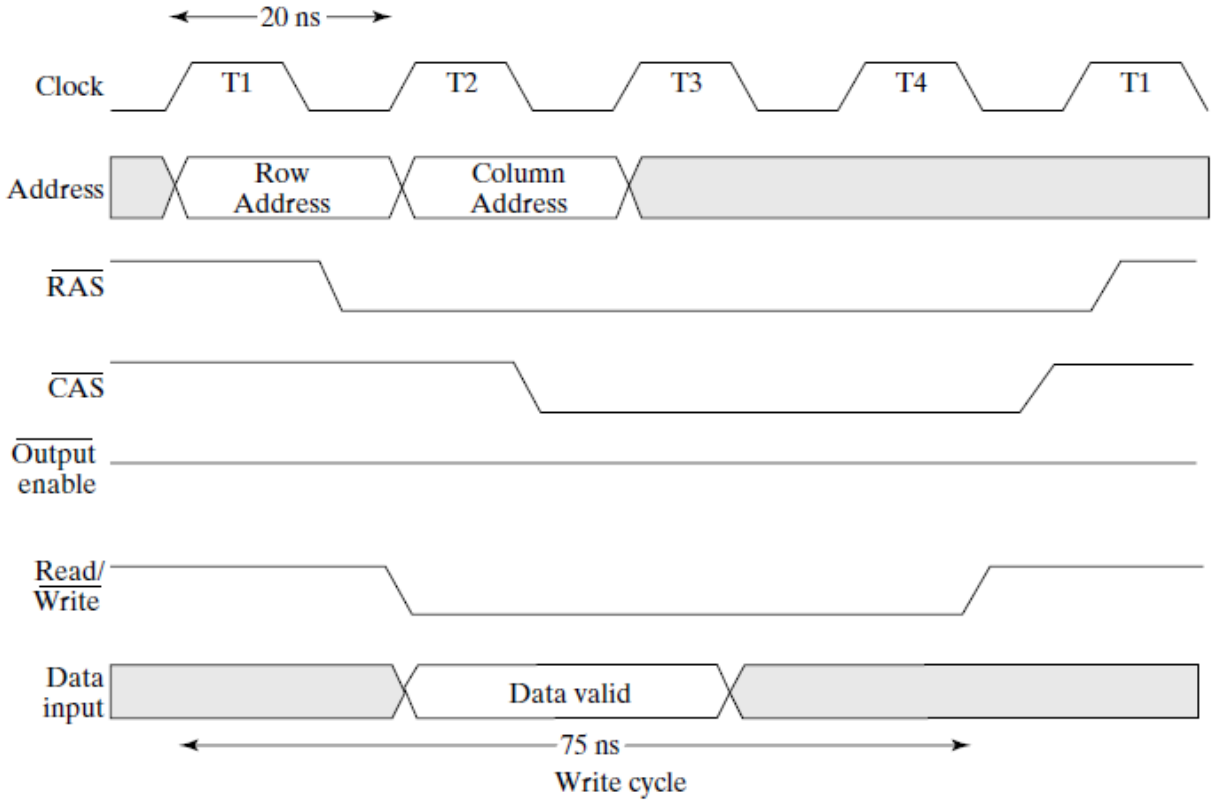
Block Diagram of a DRAM w/ Refresh Logic

- To reduce the physical size of the DRAM chips, we try to reduce # of pins
  - Address is applied serially in two parts
  - First, a row address
  - Then, a column address
- There is a refresh counter and a controller
  - The counter is used to provide the address of the row of DRAM cells to be refreshed
  - Max refresh time: 16-64 ms

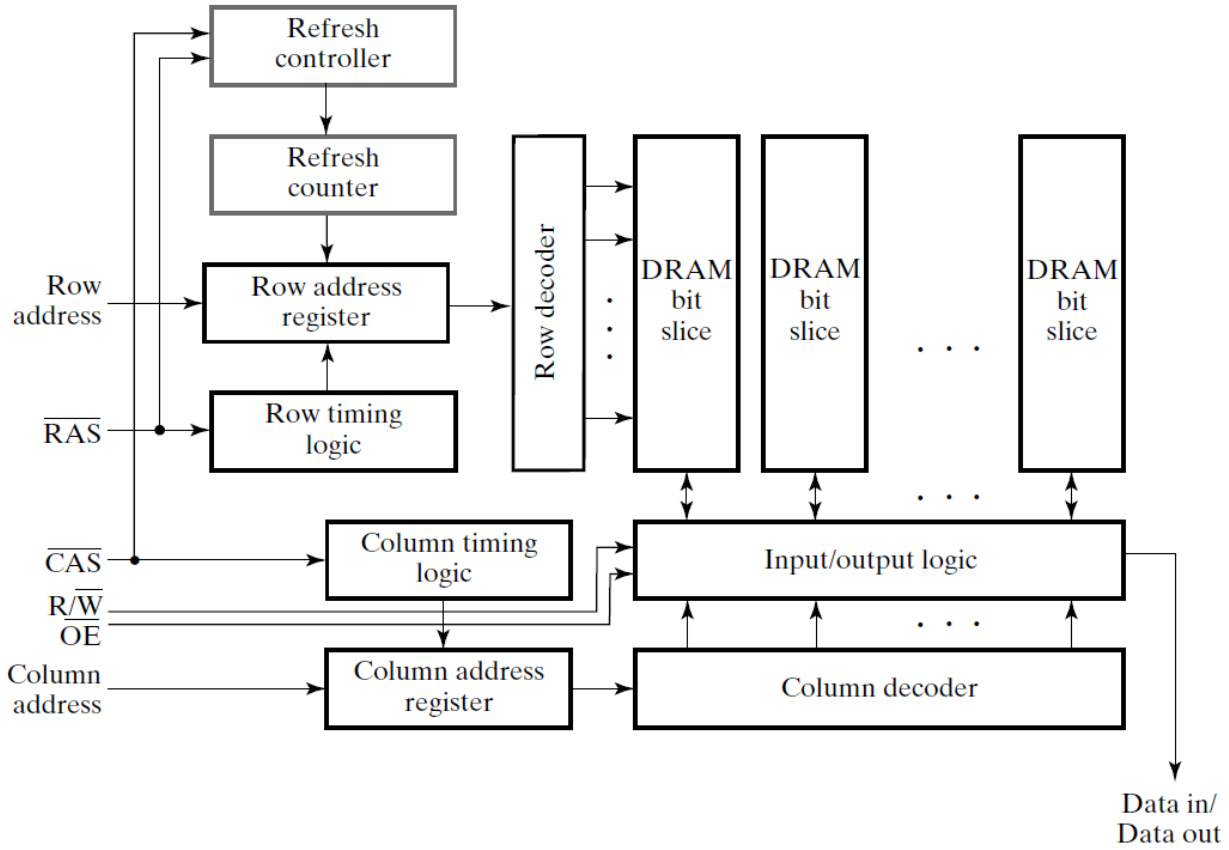


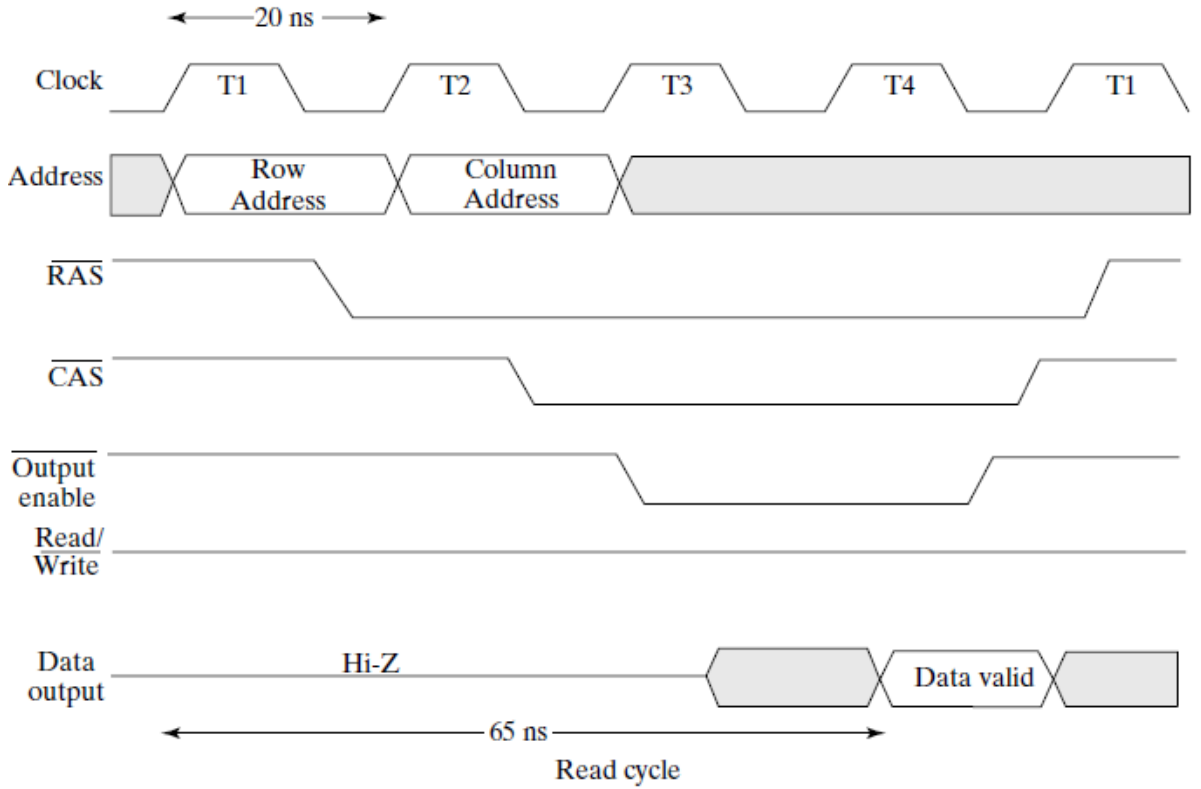
Timing of Write Operation





Timing of Read Operation





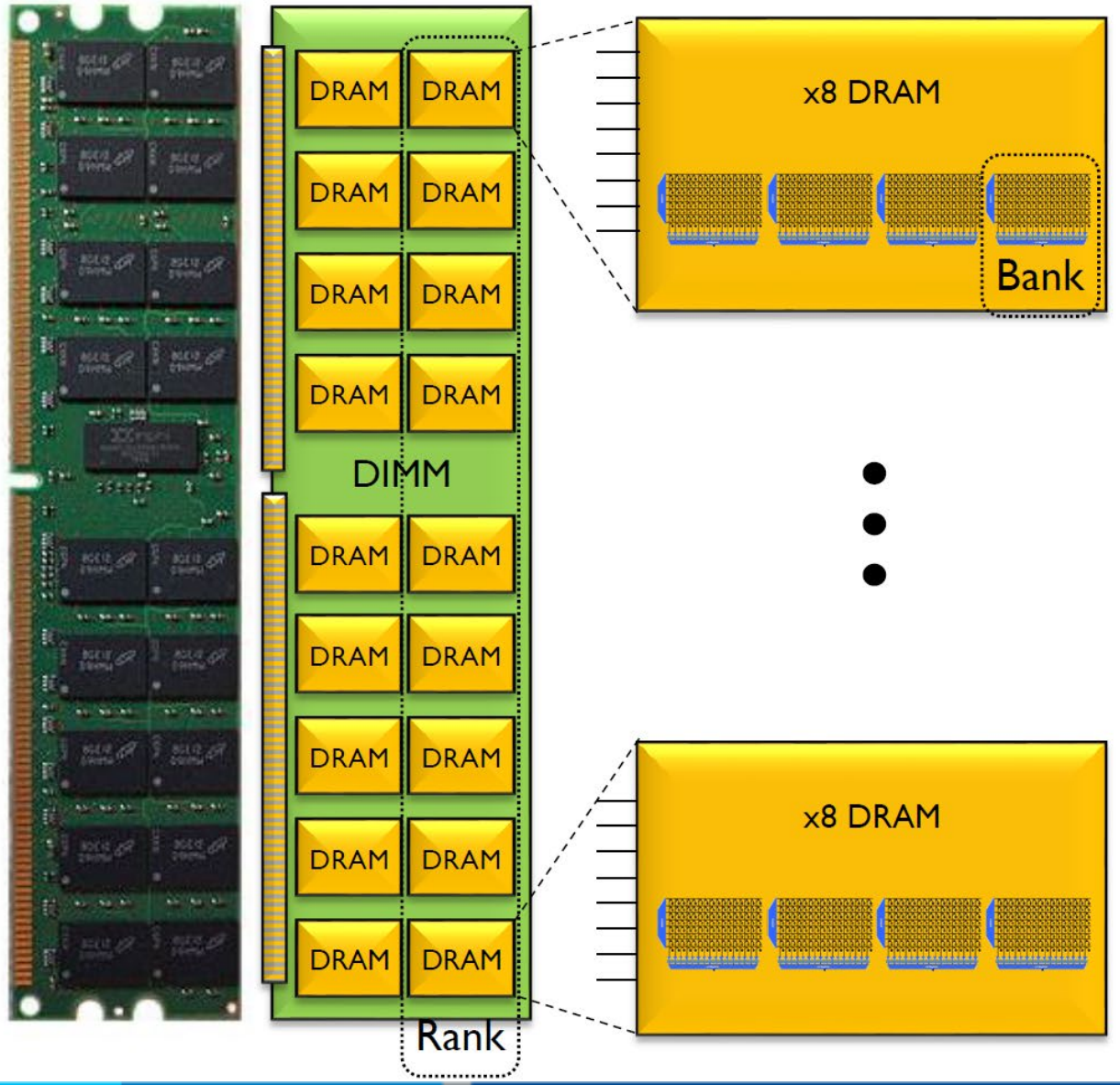
Row Buffer

- Row buffer holds read data
  - Data in row buffer is called a DRAM row
    - Often called “page” – do not confuse with virtual memory page
  - Read gets entire row into the buffer
  - Block reads always performed out of the row buffer
    - Reading a whole row, but accessing one block

DRAM Refresh Modes

- Burst refresh
  - Stop the world, refresh all memory
- Distributed refresh
  - Space out refresh one (or a few) row(s) at a time
  - Avoids blocking memory for a long time
- Self-refresh (low-power mode)
  - Tell DRAM to refresh itself
  - Turn off memory controller
  - Takes some time to exit self-refresh

DRAM Organization



All banks within the rank share all address and control pins

All banks are independent, but can only talk to one bank at a time

x8 means each DRAM outputs 8 bits, need 8 chips for DDRx (64-bit)

Why 9 chips per rank? 64 bits data, 8 bits ECC

Error Detection and Correction

- Dynamic physical interaction of electrical signals affecting datapath of a memory unit may cause errors in retrieving binary information
  - The reliability of memory unit is extremely important
  - Recall the parity bit to detect bit errors

- Error-correcting code generates multiple parity check bits that are stored with data word in memory
  - Each check bit is a parity over a group of bits
  - When the word is read back, the associated parity bits are also read from memory and compared
  - If the check bits do not match the stored parity, they generate a unique pattern, called **syndrome**

Hamming Code

- One of the most common error-correcting codes used in RAMs
  - ‘k’ parity bits are added to an n-bit data word
  - Forming a new word of ‘n+k’ bits
  - Those positions numbered as a power of 2 are reserved for parity bits

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		...
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

- Consider the 8-bit data word 11000100 with 4 parity bits

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

Hamming Code

- Consider the 8-bit data word 11000100 with 4 parity bits

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

- Each parity bit is calculated as follows:

$$P_1 = \text{XOR of bits } (3, 5, 7, 9, 11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$P_2 = \text{XOR of bits } (3, 6, 7, 10, 11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4 = \text{XOR of bits } (5, 6, 7, 12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$P_8 = \text{XOR of bits } (9, 10, 11, 12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

	0	0	1	1	1	0	0	1	0	1	0	0
Bit position:	1	2	3	4	5	6	7	8	9	10	11	12

Read Memory with Parity Bits

- When 12bits are read from memory, they are checked for errors
  - Note that bits were stored w/ even parity
  - C = 0000 indicates no error

$$C_1 = \text{XOR of bits } (1, 3, 5, 7, 9, 11) \qquad C_4 = \text{XOR of bits } (4, 5, 6, 7, 12)$$

$$C_2 = \text{XOR of bits } (2, 3, 6, 7, 10, 11) \qquad C_8 = \text{XOR of bits } (8, 9, 10, 11, 12)$$

Detecting Errors

- Evaluating XOR of the corresponding bits,

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

C_1 = XOR of bits (1, 3, 5, 7, 9, 11)

C_2 = XOR of bits (2, 3, 6, 7, 10, 11)

C_4 = XOR of bits (4, 5, 6, 7, 12)

C_8 = XOR of bits (8, 9, 10, 11, 12)

	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

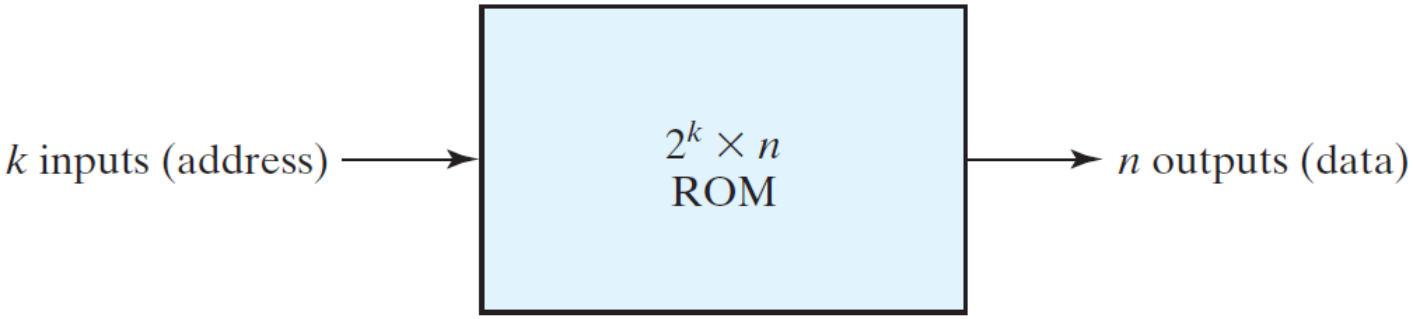
Another Example : Hamming Code of (00101110)

	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	D_1	P_4	D_2	D_3	D_4	P_8	D_5	D_6	D_7	D_8
DATA			0		0	1	0		1	1	1	0
P_1	0		0		0		0		1		1	
P_2	↓	1	0			1	0			1	1	
P_4	↓	↓		1	0	1	0					0
P_8	↓	↓		↓				1	1	1	1	0
	↓	↓		↓				↓				
Final	0	1	0	1	0	1	0	1	1	1	1	0

Read-Only Memory (ROM)

Read-Only Memory

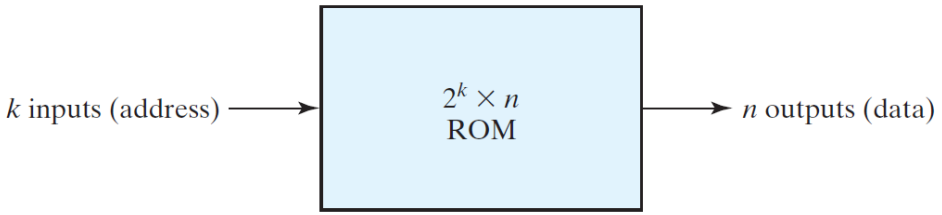
- ROM is essentially a memory device in which permanent binary information is stored
  - Once the interconnection pattern is established, it stays within the unit even when power is turned off

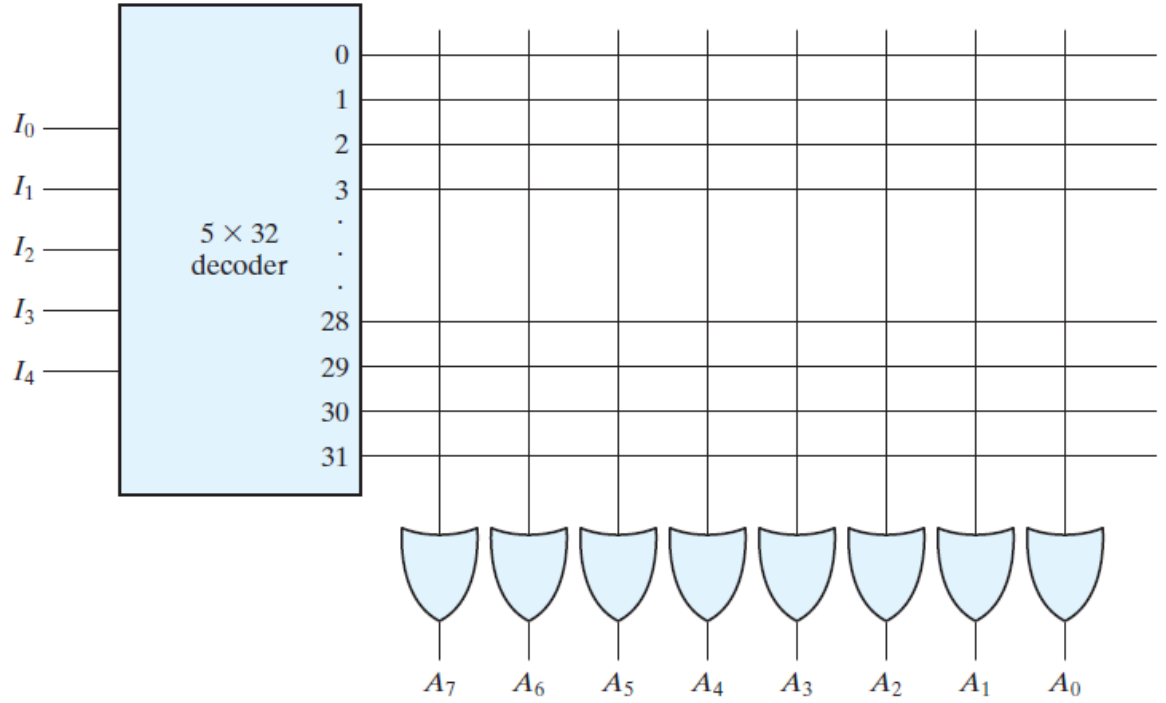


A block diagram of a ROM consisting of ‘k’ inputs and ‘n’ outputs

32 x 8 ROM

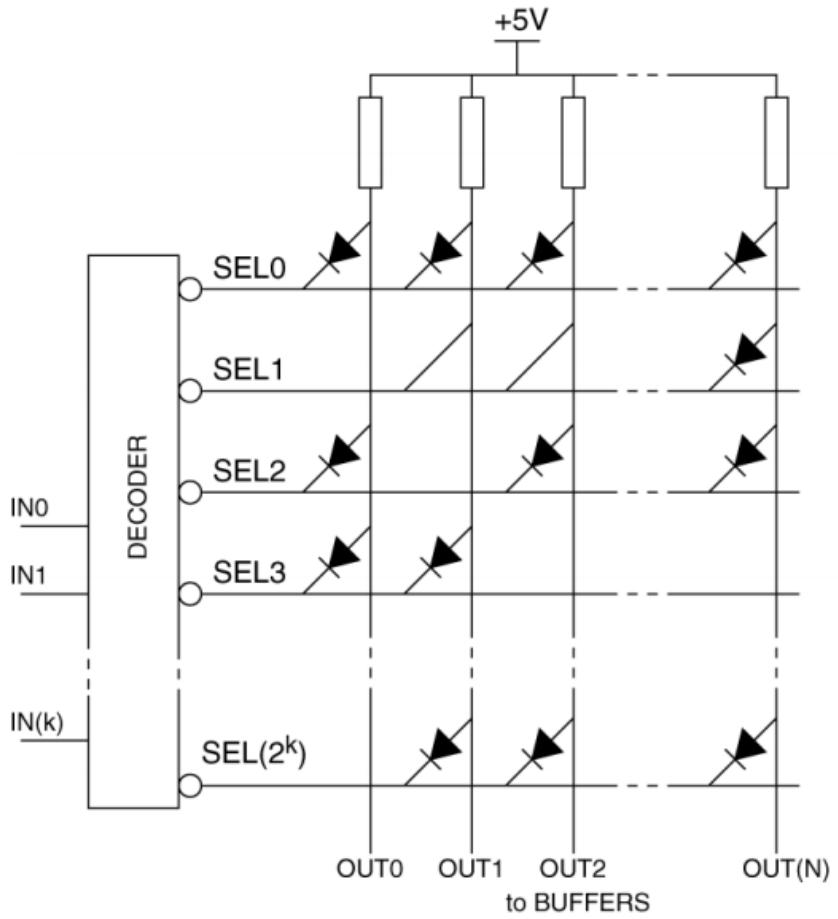
- The unit consists of 32 words of 8bits each
  - It requires a 5 x 32 decoder to process 5bit input address
  - 32 outputs of the decoder are connected to each of eight OR gates
  - The ROM contains 32 * 8 = 256 internal connections

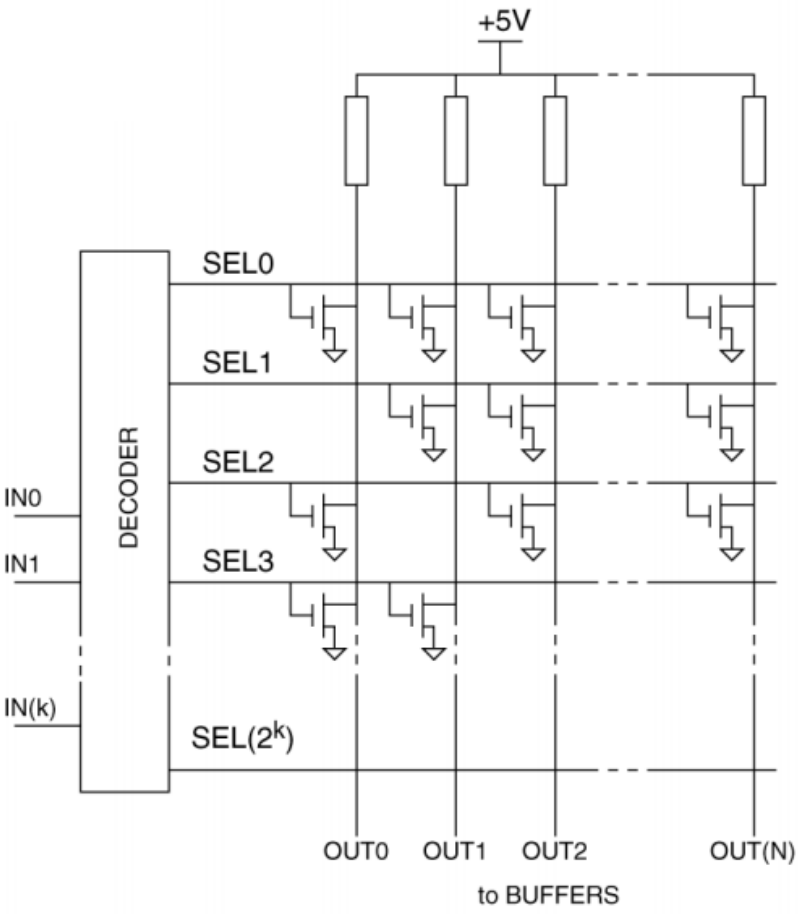




Programmable Connection

- A crosspoint btw two lines behaves as a switch that can be either closed or open





ROM Truth Table

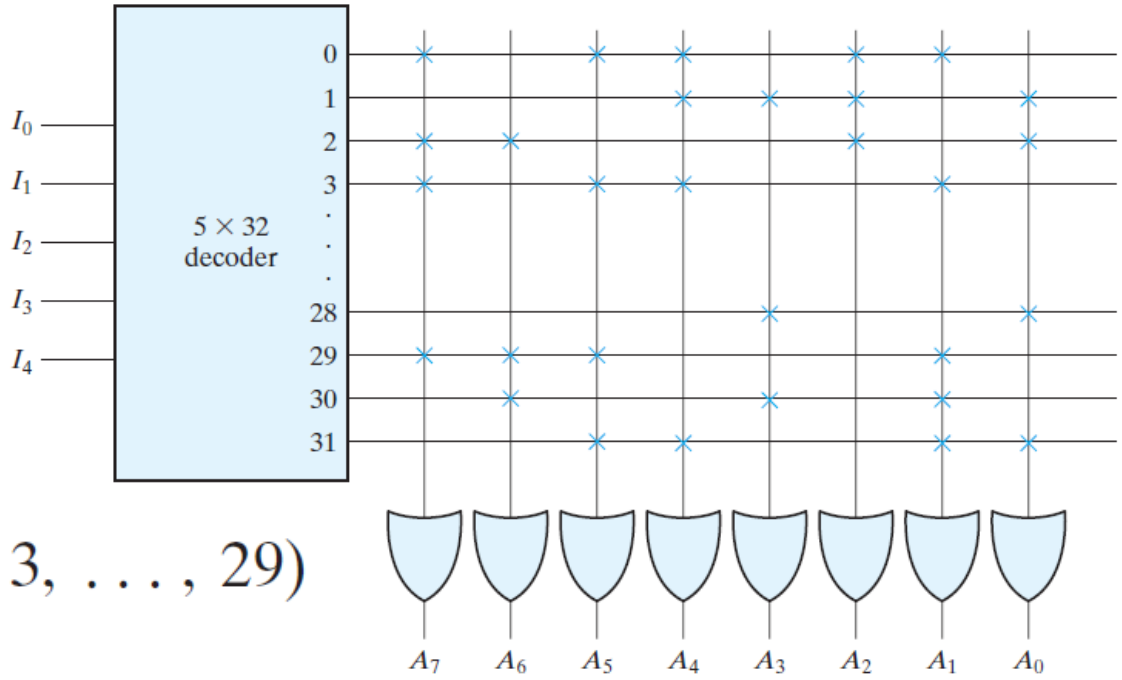
- The content of a 32 x 8 ROM may be specified with a truth table
  - The hardware procedure that programs the ROM blows fuse links in accordance with a given truth table
  - Every 0 listed in the truth table specifies the absence of a connection

ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮							⋮			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Combinational Circuit Implementation

- A decoder generates the 2^k minterms of ‘k’ input variables
  - By inserting OR gates to sum the minterms, we are able to generate any desired combinational circuit
  - The ROM can be used as a minterm generator
  - Each output terminal is considered separately as the output of a Boolean function

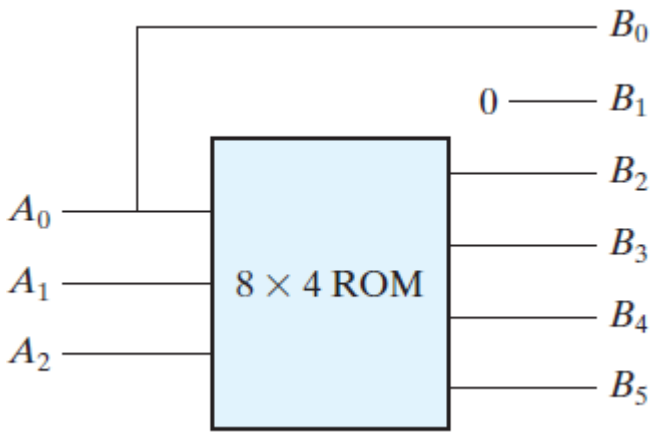


$$A_7(I_4, I_3, I_2, I_1, I_0) = \Sigma(0, 2, 3, \ldots, 29)$$

Combinational Circuit Implementation

- All that the designer has to do is provide the applicable truth table to the particular ROM
  - Truth table gives all the info for programming ROM
  - Let's design a circuit that accepts 3-bit number 'n' and outputs 'n^2' as a binary number

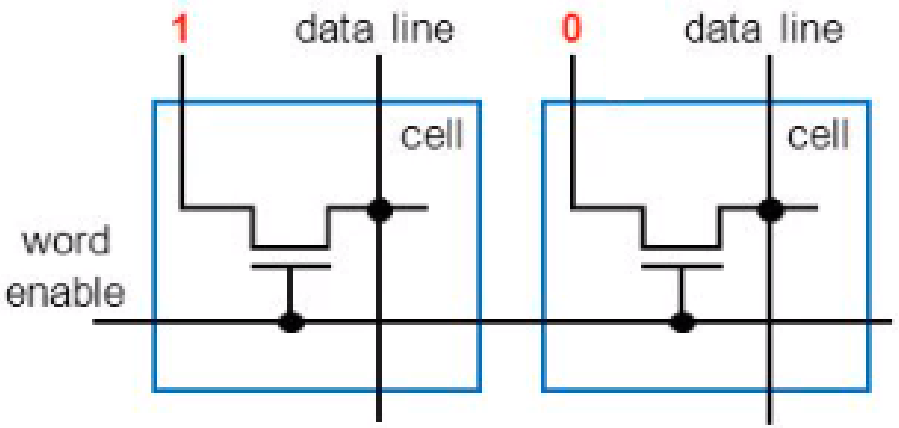
Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

Types of ROMs

- Mask-programmed ROM
  - Bits are hardwired as 0s or 1s during chip manufacturing
    - 2-bit word on right stores “10”
    - ‘word enable’ (from decoder) simply passes hardwired value through transistor
  - Notice how compact, and fast, this memory would be

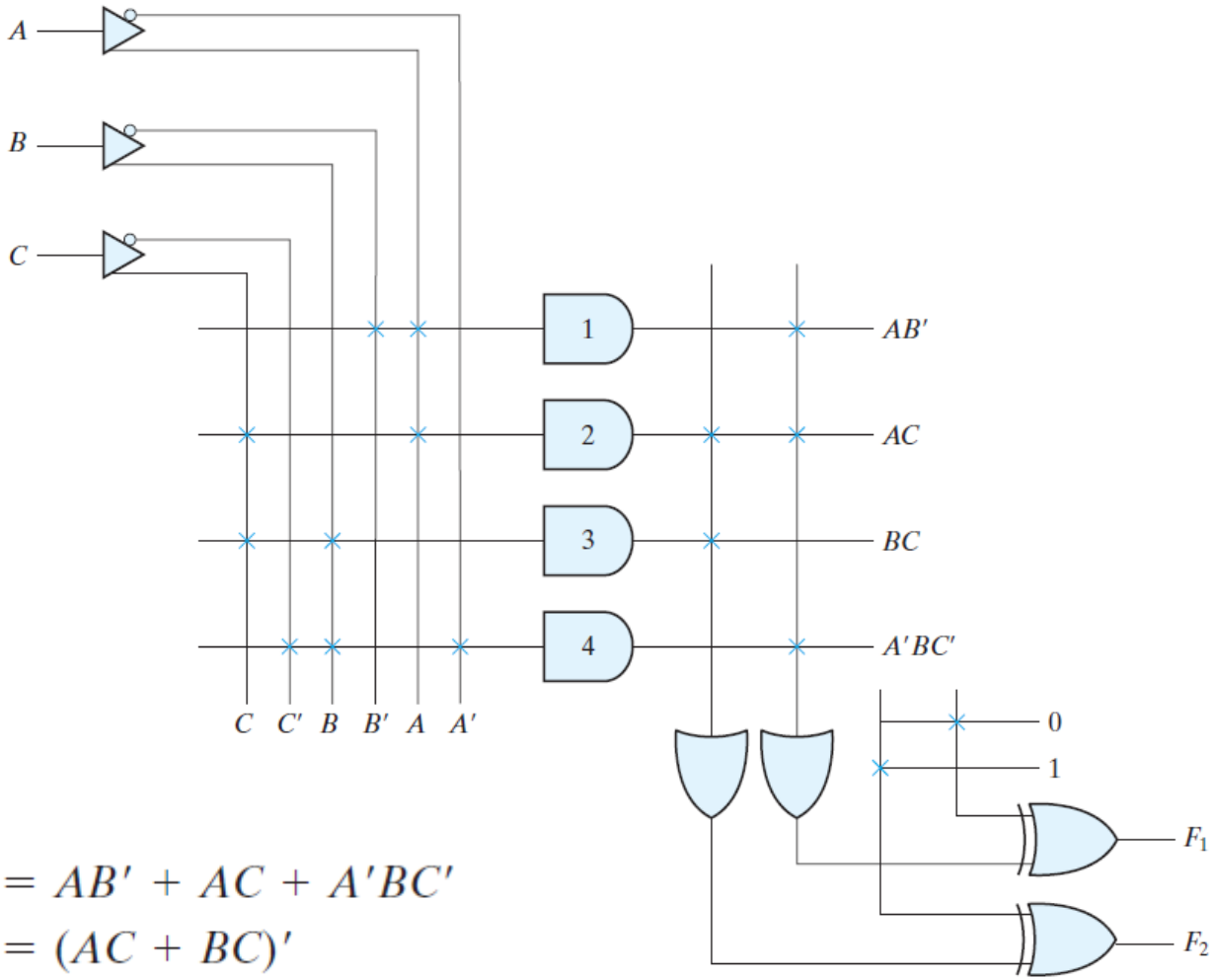


Types of ROMs

- Programmable read-only memory (PROM)
  - PROM units contain all the fuses intact, giving all 1s in the bits of stored words
  - The fuses are blown by the application of a high-voltage pulse through a special pin
    - Blown fuse: defines 0
    - Intact fuse: defines 1

Programmable Logic Array (PLA)

- PLA is similar to ROM, except it does not provide full decoding of variables and does not generate all minterms
  - The decoder is replaced by an array of AND gates
  - The AND gate array generates any product term of input variables



$$F_1 = AB' + AC + A'BC'$$
$$F_2 = (AC + BC)'$$

Programmable Logic Array (PLA)

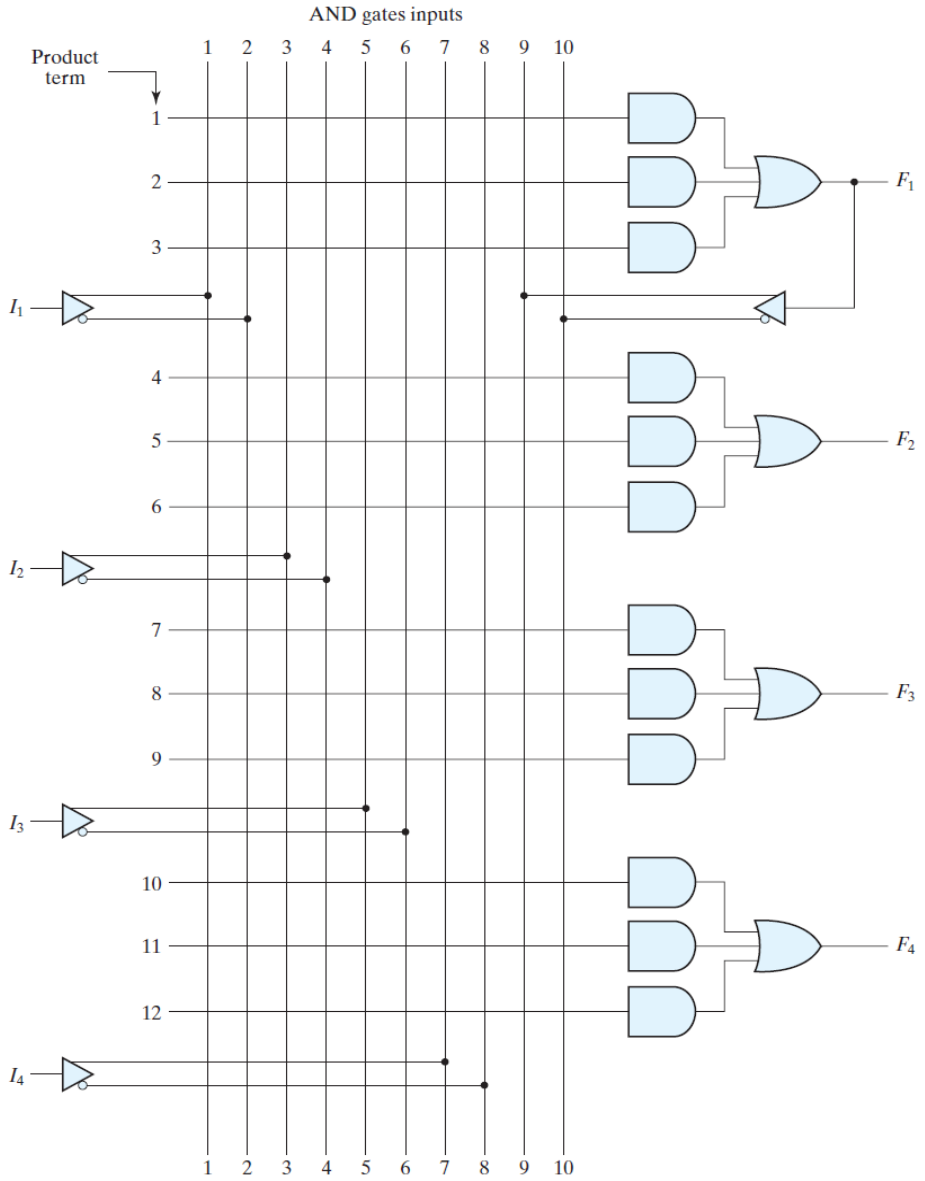
- The size of a PLA is specified by # of inputs, # of product terms, # of outputs
  - For ‘n’ inputs, ‘k’ product terms, and ‘m’ outputs, the internal logic consists of ‘n’ buffer-inverter gates, ‘k’ AND gates, ‘m’ OR gates, and ‘m’ XOR gates
  - ‘2n x k’ connections btw inputs and AND array
  - ‘k x m’ connections btw AND and OR arrays

- We need to reduce # of distinct product terms, since there is a finite number of AND gates
  - Simplifying each Boolean function is required!

Programmable Array Logic (PAL)

- It is a programmable device with a **fixed OR array** and a **program-mable AND array**
  - PAL is easier to program than, but is not as flexible as, the PLA
  - Each input has a buffer-inverter gate
  - Each output is generated by a fixed OR gate

Each AND gate has 10 programmable input connections



Programmable Array Logic (PAL)

- A typical PAL circuit may have 8 inputs, 8 outputs, and 8 sections (each consisting of an eight-wide AND-OR array)
- Boolean functions must be simplified to fit into each section
  - Unlike PLA, **a product term cannot be shared** among two or more OR gates
  - If **# of terms** in the function **is too large**, **we need two sections** to implement one Boolean function

Programmable Array Logic (PAL)

- Consider the following Boolean functions

$$w(A, B, C, D) = \sum(2, 12, 13)$$
$$x(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$
$$= w + AC'D' + A'B'C'D$$

Programmable Array Logic (PAL)

- Now, we establish PAL programming table

$$w = ABC' + A'B'CD'$$

$$x = A + BCD$$

$$y = A'B + CD + B'D'$$

$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$

$$= w + AC'D' + A'B'C'D$$

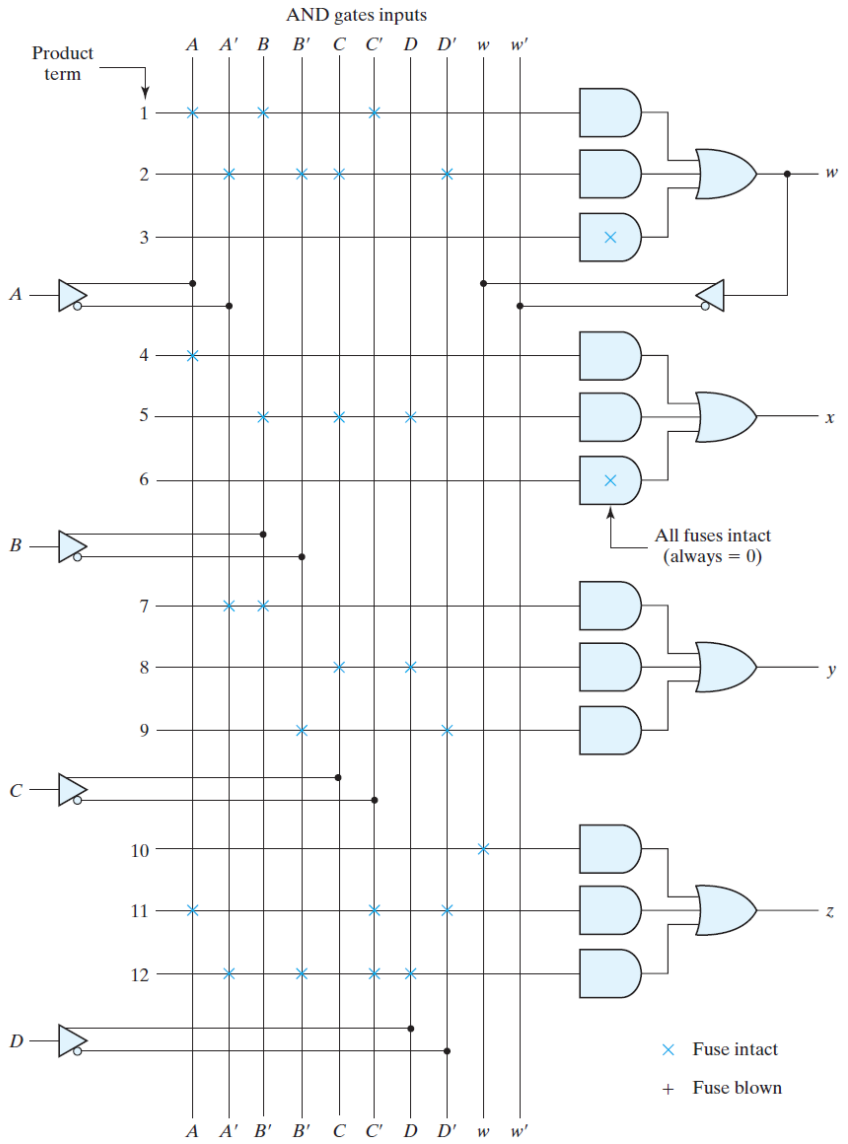
Product Term	AND Inputs A	B	C	D	w	Outputs
1	1	1	0	—	—	$w = ABC' + A'B'CD'$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	$x = A + BCD$
5	—	1	1	1	—	
6	—	—	—	—	—	
7	0	1	—	—	—	$y = A'B + CD + B'D'$
8	—	—	1	1	—	
9	—	0	—	0	—	
10	—	—	—	—	1	$z = w + AC'D' + A'B'C'D$
11	1	—	0	0	—	
12	0	0	0	1	—	

Three product terms each

Programmable Array Logic (PAL)

- The fuse map for the PAL becomes

Product Term	AND Inputs A	B	C	D	w	Outputs
1	1	1	0	—	—	$w = ABC' + A'B'CD'$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	$x = A + BCD$
5	—	1	1	1	—	
6	—	—	—	—	—	
7	0	1	—	—	—	$y = A'B + CD + B'D'$
8	—	—	1	1	—	
9	—	0	—	0	—	
10	—	—	—	—	1	$z = w + AC'D' + A'B'C'D$
11	1	—	0	0	—	
12	0	0	0	1	—	



Summary

- Learned different types of memory units
  - Random-access memory (RAM)
  - Read-only memory (ROM)

- Learned how memory decoding works
  - Internal structure of memory cell and array
  - Address multiplexing
  - Error detection and correction