Chapter 10: Memory

This chapter introduces the basic concepts, terminology, and roles of memory in digital systems. The material presented here will not delve into the details of the device physics or low-level theory of operation. Instead, the intent of this chapter is to give a general overview of memory technology and its use in computer systems in addition to how to model memory in VHDL. The goal of this chapter is to give an understanding of the basic principles of semiconductor-based memory systems.

Learning Outcomes—After completing this chapter, you will be able to:

10.1 Describe the basic architecture and terminology for semiconductor-based memory systems.
10.2 Describe the basic architecture of non-volatile memory systems.
10.3 Describe the basic architecture of volatile memory systems.
10.4 Design a VHDL behavioral model of a memory system.

10.1 Memory Architecture and Terminology

The term *memory* is used to describe a system with the ability to store digital information. The term *semiconductor memory* refers to systems that are implemented using integrated circuit technology. These types of systems store the digital information using transistors, fuses, and/or capacitors on a single semiconductor substrate. Memory can also be implemented using technology other than semiconductors. Disk drives store information by altering the polarity of magnetic fields on a circular substrate. The two magnetic polarities (north and south) are used to represent different logic values (i.e., 0 or 1). Optical disks use lasers to burn pits into reflective substrates. The binary information is represented by light either being reflected (no pit) or not reflected (pit present). Semiconductor memory does not have any moving parts, so it is called *solid-state memory* and can hold more information per unit area than disk memory. Regardless of the technology used to store the binary data, all memory has common attributes and terminology that are discussed in this chapter.

10.1.1 Memory Map Model

The information stored in memory is called the **data**. When information is placed into memory, it is called a **write**. When information is retrieved from memory, it is called a **read**. In order to access data in memory, an **address** is used. While data can be accessed as individual bits, in order to reduce the number of address locations needed, data is typically grouped into *N-bit words*. If a memory system has N = 8, this means that 8 bits of data are stored at each address. The number of address locations is described using the variable *M*. The overall size of the memory is typically stated by saying "MxN." For example, if we had a 16×8 memory system, that means that there are 16 address locations, each capable of storing a byte of data. This memory would have a **capacity** of 16×8 = 128 bits. Since the address is implemented as a binary code, the number of lines in the address bus (n) will dictate the number of address locations that the memory system will have ($M = 2^n$). Figure 10.1 shows a graphical depiction of how data resides in memory. This type of graphic is called a *memory map model*.

B. J. LaMeres, *Introduction to Logic Circuits & Logic Design with VHDL*,
https://doi.org/10.1007/978-3-030-12489-2_10

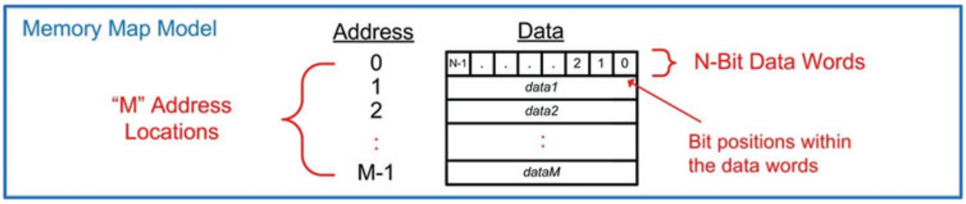


Fig. 10.1
Memory map model

10.1.2 Volatile Versus Non-volatile Memory

Memory is classified into two categories depending on whether it can store information when power is removed or not. The term **non-volatile** is used to describe memory that *holds* information when the power is removed, while the term **volatile** is used to describe memory that loses its information when power is removed. Historically, volatile memory is able to run at faster speeds compared to non-volatile memory, so it is used as the primary storage mechanism while a digital system is running. Non-volatile memory is necessary in order to hold critical operation information for a digital system such as start-up instructions, operations systems, and applications.

10.1.3 Read-Only Versus Read/Write Memory

Memory can also be classified into two categories with respect to how data is accessed. **Read-only memory (ROM)** is a device that cannot be written to during normal operation. This type of memory is useful for holding critical system information or programs that should not be altered while the system is running. **Read/write** memory refers to memory that can be read and written to during normal operation and is used to hold temporary data and variables.

10.1.4 Random Access Versus Sequential Access

Random access memory (RAM) describes memory in which any location in the system can be accessed at any time. The opposite of this is **sequential access** memory, in which not all address locations are immediately available. An example of a sequential access memory system is a tape drive. In order to access the desired address in this system, the tape spool must be spun until the address is in a position that can be observed. Most semiconductor memory in modern systems is random access. The terms RAM and ROM have been adopted, somewhat inaccurately, to also describe groups of memory with particular behavior. While the term ROM technically describes a system that cannot be written to, it has taken on the additional association of being the term to describe non-volatile memory. While the term RAM technically describes how data is accessed, it has taken on the additional association of being the term to describe volatile memory. When describing modern memory systems, the terms RAM and ROM are used most commonly to describe the characteristics of the memory being used; however, modern memory systems can be both read/write and non-volatile, and the majority of memory is random access.

CONCEPT CHECK

CC10.1 An 8-bit wide memory has 8 address lines. What is its capacity in bits?

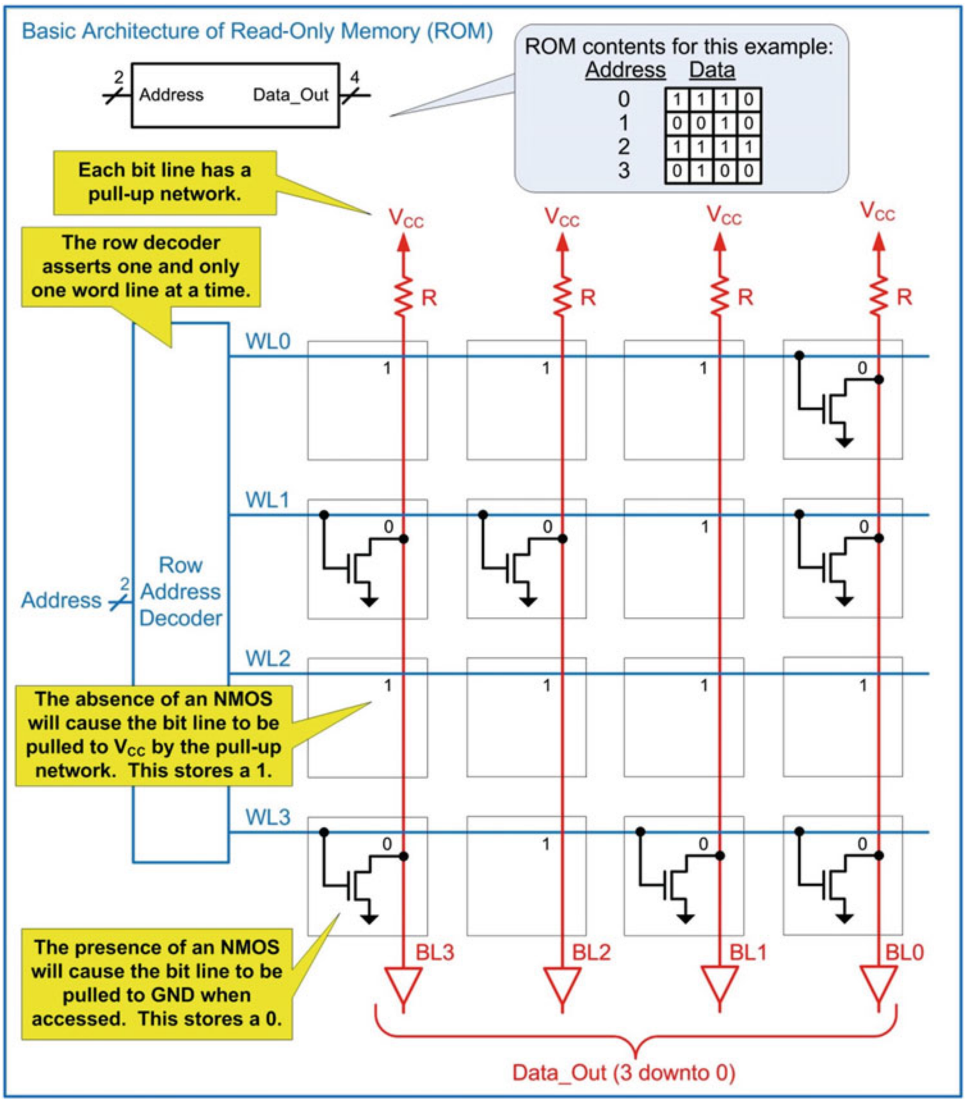
A) 64 B) 256 C) 1024 D) 2048

10.2 Non-volatile Memory Technology

10.2.1 ROM Architecture

This section describes some of the most common non-volatile memory technologies used to store digital information. An address decoder is used to access individual data words within the memory system. The address decoder asserts one and only one *word line* (WL) for each unique binary address that is present on its input. This operation is identical to a binary-to-one-hot decoder. For an n-bit address, the decoder can access 2^n, or M words in memory. The word lines historically run horizontally across the memory array; thus, they are often called *row lines*, and the word line decoder is often called the *row decoder. Bit lines* (BL) run perpendicular to the word lines in order to provide individual bit storage access at the intersection of the bit and word lines. These lines typically run vertically through the memory array; thus, they are often called *column lines*. The output of the memory system (i.e., Data_Out) is obtained by providing an address and then reading the word from buffered versions of the bit lines. When a system provides individual bit access to a row or access to multiple data words sharing a row line, a column decoder is used to route the appropriate bit line(s) to the data out port.

In a traditional ROM array, each bit line contains a pull-up network to V_{CC}. This provides the ability to store a logic 1 at all locations within the array. If a logic 0 is desired at a particular location, an NMOS pull-down transistor is inserted. The gate of the NMOS is connected to the appropriate word line, and the drain of the NMOS is connected to the bit line. When reading, the word line is asserted and turns on the NMOS transistor. This pulls the bit line to GND and produces a logic 0 on the output. When the NMOS transistor is excluded, the bit line remains at a logic 1 due to the pull-up network. Figure 10.2 shows the basic architecture of a ROM.



Basic Architecture of Read-Only Memory (ROM)
2 Address
Data_Out 4
ROM contents for this example:
Address Data
0 1 1 1 0
1 0 0 1 0
2 1 1 1 1
3 0 1 0 0
Each bit line has a pull-up network.
The row decoder asserts one and only one word line at a time.
Vcc
R
WL0
WL1
WL2
WL3
Address 2
Row Address Decoder
The absence of an NMOS will cause the bit line to be pulled to Vcc by the pull-up network. This stores a 1.
The presence of an NMOS will cause the bit line to be pulled to GND when accessed. This stores a 0.
BL3
BL2
BL1
BL0
Data_Out (3 downto 0)

Fig. 10.2
Basic architecture of read-only memory (ROM)

Figure 10.3 shows the operation of a ROM when information is being read.

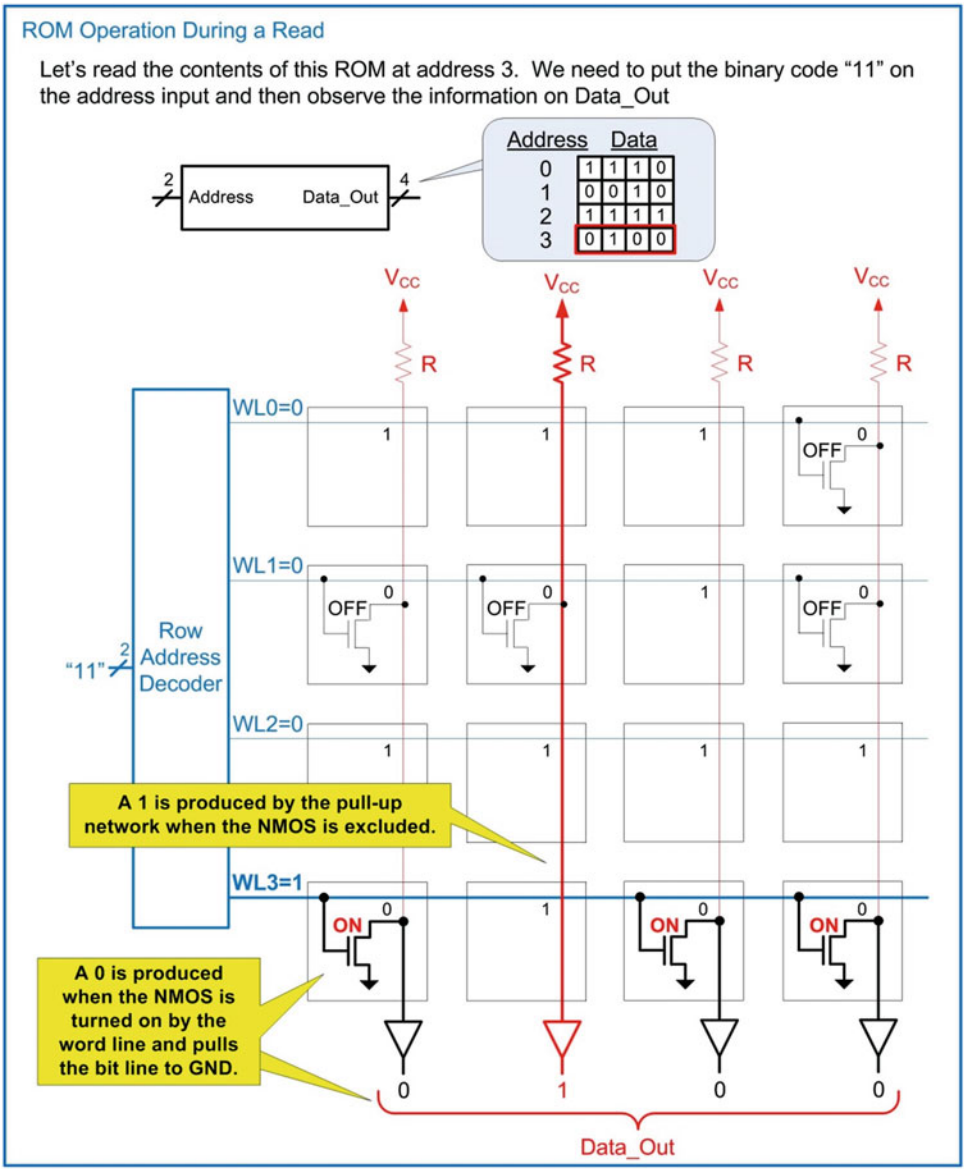

ROM Operation During a Read
Let's read the contents of this ROM at address 3. We need to put the binary code "11" on the address input and then observe the information on Data_Out
Address
Data_Out
2
4
Address Data
0 1 1 1 0
1 0 0 1 0
2 1 1 1 1
3 0 1 0 0
Vcc
R
WL0=0
WL1=0
WL2=0
WL3=1
OFF
ON
Row Address Decoder
"11"
A 1 is produced by the pull-up network when the NMOS is excluded.
A 0 is produced when the NMOS is turned on by the word line and pulls the bit line to GND.
0 1 0 0
Data_Out

Fig. 10.3
ROM operation during a read

Memory can be designed to be either asynchronous or synchronous. **Asynchronous memory** updates its data outputs immediately upon receiving an address. **Synchronous memory** only updates its data outputs on the rising edge of a clock. The term *latency* is used to describe the delay between when a signal is sent to the memory (either the address in an asynchronous system or the clock in a synchronous system) and when the data is available. Figure 10.4 shows a comparison of the timing diagrams between asynchronous and synchronous ROM systems during a read cycle.

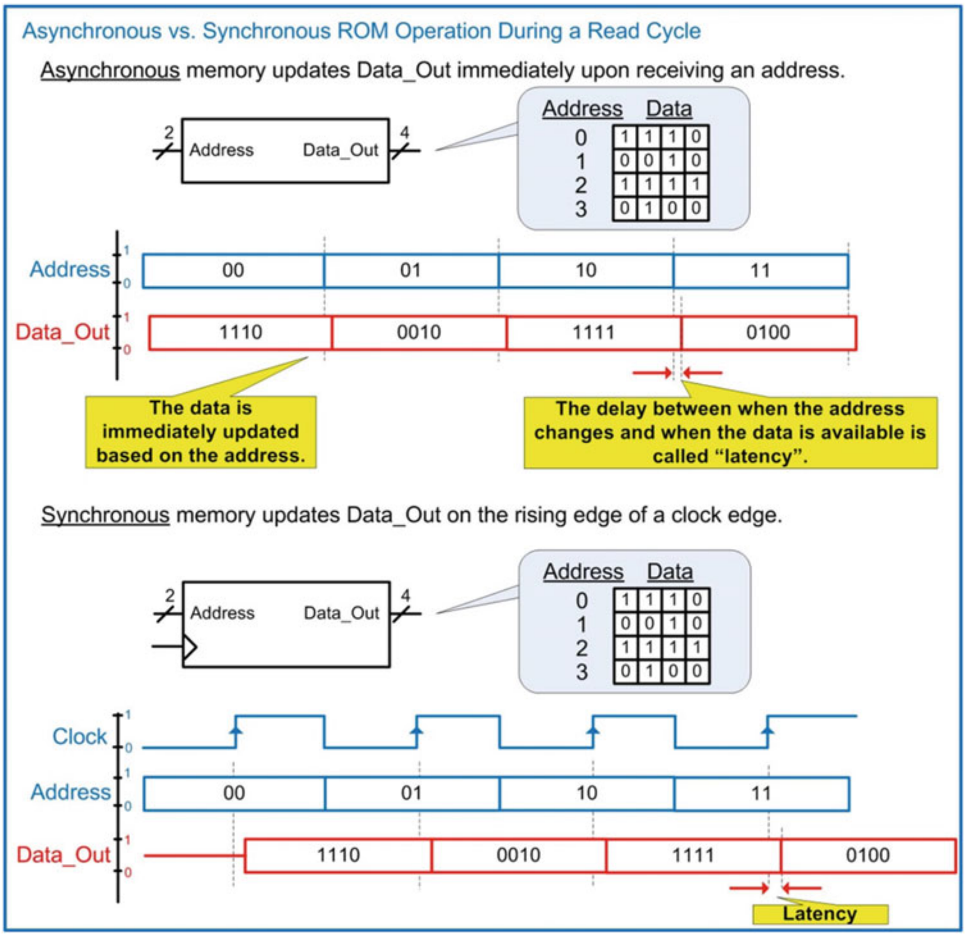

Asynchronous vs. Synchronous ROM Operation During a Read Cycle
Asynchronous memory updates Data_Out immediately upon receiving an address.
Address
Data_Out
Address Data
0 1110
1 0010
2 1111
3 0100
Address 00 01 10 11
Data_Out 1110 0010 1111 0100
The data is immediately updated based on the address.
The delay between when the address changes and when the data is available is called "latency".
Synchronous memory updates Data_Out on the rising edge of a clock edge.
Clock
Latency

Fig. 10.4
Asynchronous vs. synchronous ROM operation during a read cycle

10.2.2 Mask Read-Only Memory (MROM)

A mask read-only memory (MROM) is a non-volatile device that is programmed during fabrication. The term *mask* refers to a transparent plate that contains patterns to create the features of the devices on an integrated circuit using a process called photolithography. An MROM is fabricated with all of the features necessary for the memory device with the exception of the final connections between the NMOS transistors and the word and bit lines. This allows the majority of the device to be created prior to knowing what the final information to be stored is. Once the desired information to be stored is provided by the customer, the fabrication process is completed by adding connections between certain NMOS transistors and the word/bit lines in order to create logic 0's. Figure 10.5 shows an overview of the MROM programming process.

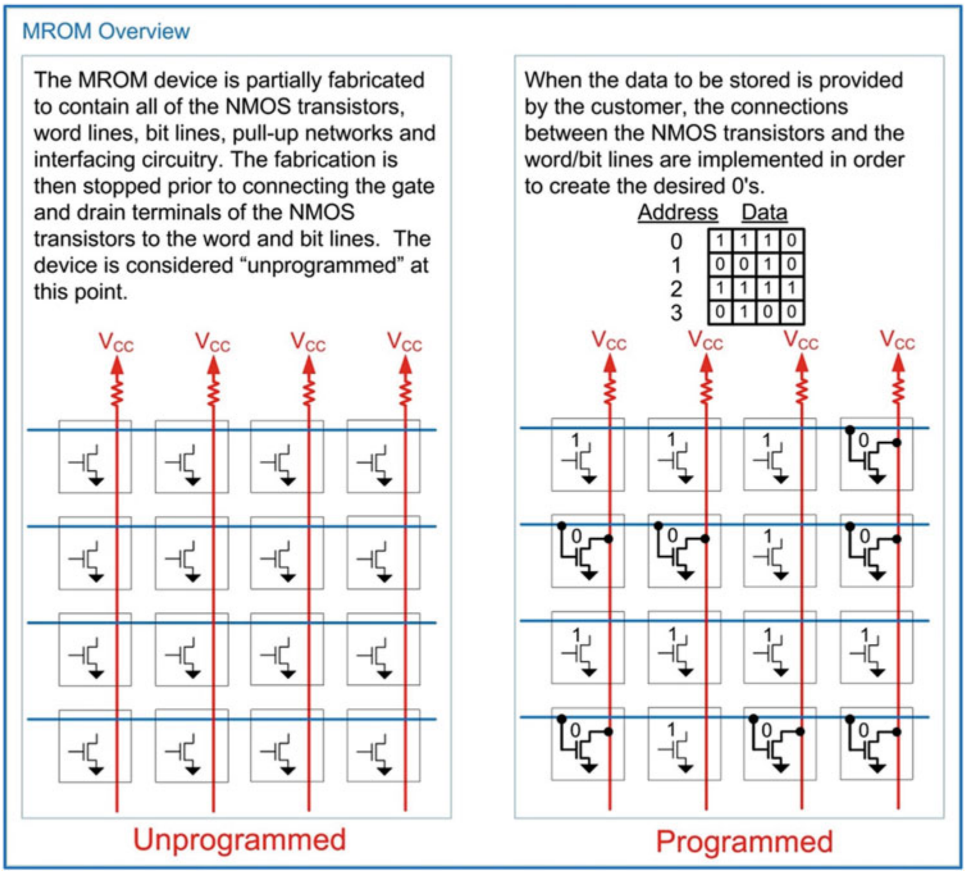

MROM Overview
The MROM device is partially fabricated to contain all of the NMOS transistors, word lines, bit lines, pull-up networks and interfacing circuitry. The fabrication is then stopped prior to connecting the gate and drain terminals of the NMOS transistors to the word and bit lines. The device is considered "unprogrammed" at this point.
When the data to be stored is provided by the customer, the connections between the NMOS transistors and the word/bit lines are implemented in order to create the desired 0's.
Address
Data
0 1110
1 0010
2 1111
3 0100
Vcc
Unprogrammed
Programmed

Fig. 10.5
MROM overview

10.2.3 Programmable Read-Only Memory (PROM)

A programmable read-only memory (PROM) is created in a similar manner as an MROM except that the programming is accomplished post-fabrication through the use of fuses or anti-fuses. A **fuse** is an electrical connection that is normally conductive. When a certain amount of current is passed through the fuse, it will melt, or *blow*, and create an open circuit. The amount of current necessary to open the fuse is much larger than the current the fuse would conduct during normal operation. An **anti-fuse** operates in the opposite manner as a fuse. An anti-fuse is normally an open circuit. When a certain amount of current is forced into the anti-fuse, the insulating material breaks down and creates a conduction path. This turns the anti-fuse from an open circuit into a wire. Again, the amount of current necessary to close the anti-fuse is much larger than the current the anti-fuse would experience during normal operation. A PROM uses fuses or anti-fuses in order to connect/disconnect the NMOS transistors in the ROM array to the word/bit lines. A PROM programmer is used to *burn* the fuses or anti-fuses. A PROM can only be programmed once in this manner; thus, it is a read-only memory and non-volatile. A PROM has the advantage that programming can take place quickly as opposed to an MROM that must be programmed through device fabrication. Figure 10.6 shows an example PROM device based on fuses.

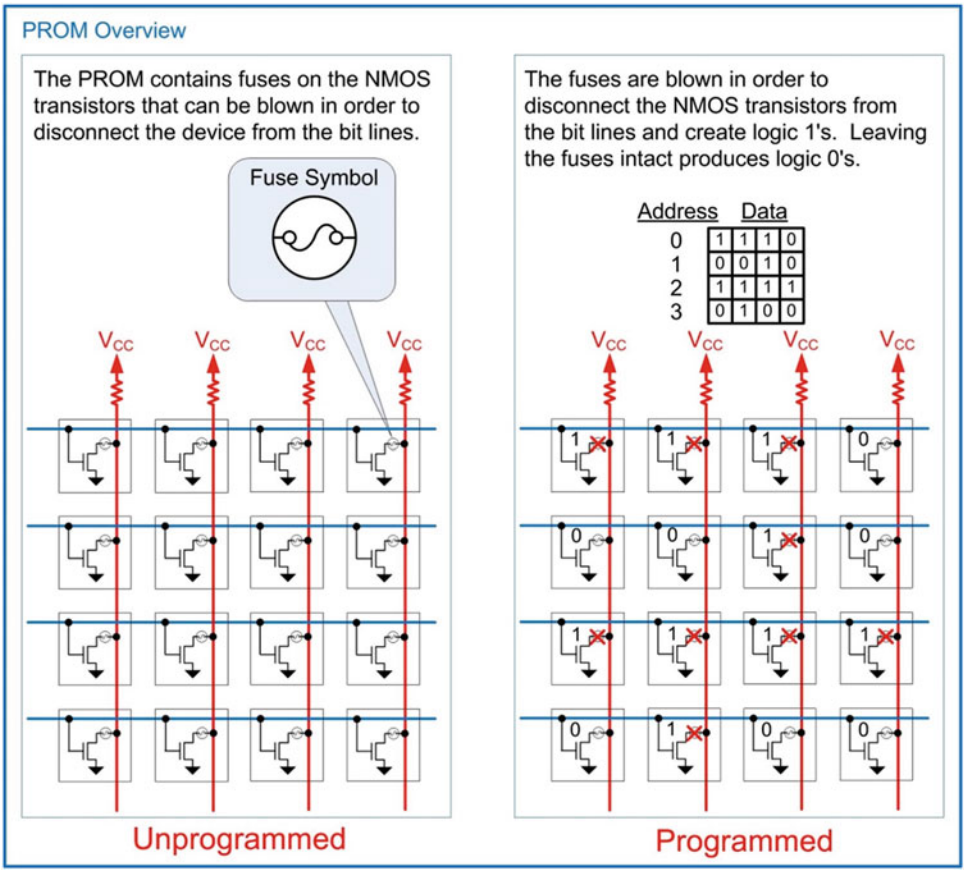
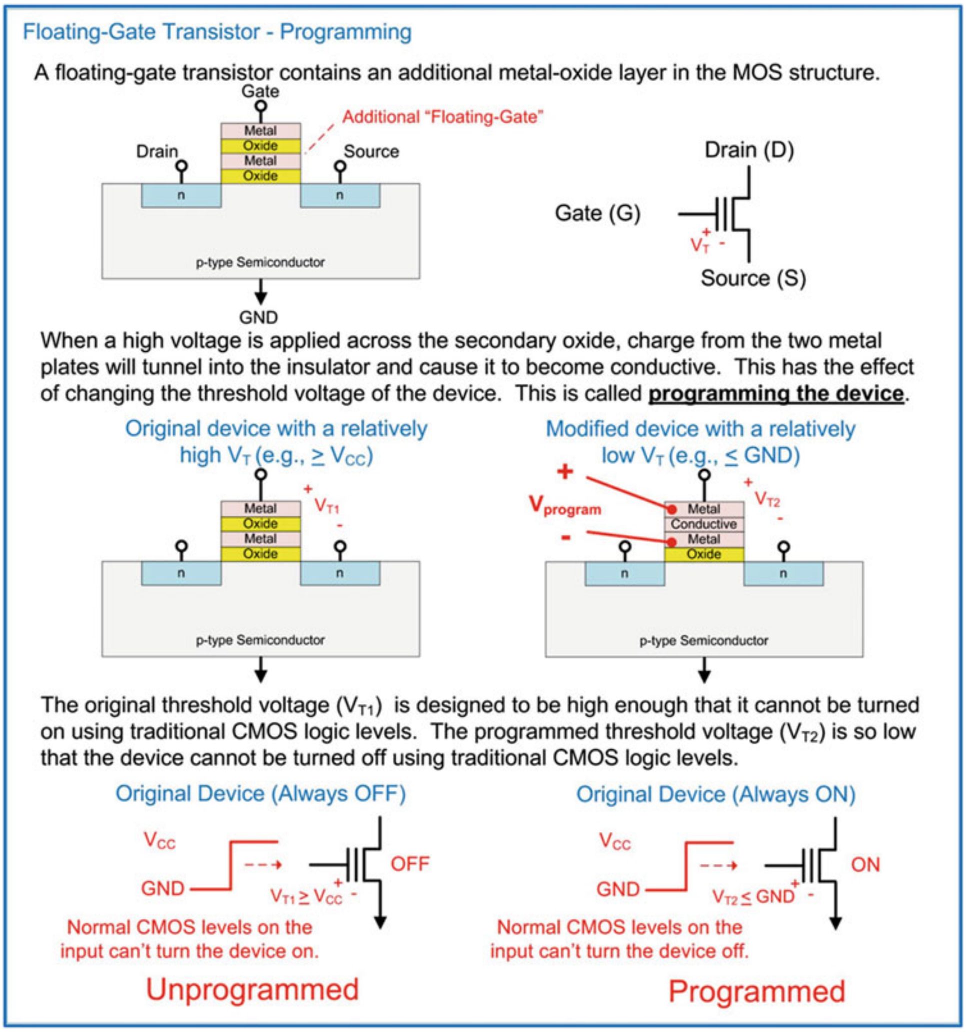

PROM Overview
The PROM contains fuses on the NMOS transistors that can be blown in order to disconnect the device from the bit lines.
Fuse Symbol
The fuses are blown in order to disconnect the NMOS transistors from the bit lines and create logic 1's. Leaving the fuses intact produces logic 0's.
Address
Data
0 1110
1 0010
2 1111
3 0100
VCC
Unprogrammed
Programmed

Fig. 10.6
PROM overview

10.2.4 Erasable Programmable Read-Only Memory (EPROM)

As an improvement to the one-time programming characteristic of PROMs, an electrically programmable ROM with the ability to be erased with ultraviolet (UV) light was created. The erasable programmable read-only memory (EPROM) is based on a **floating-gate transistor**. In a floating-gate transistor, an additional metal-oxide structure is added to the gate of an NMOS. This has the effect of increasing the threshold voltage. The geometry of the second metal oxide is designed such that the threshold voltage is high enough that normal CMOS logic levels are not able to turn the transistor on (i.e., $V_{T1} \geq V_{CC}$). This threshold can be changed by applying a large electric field across the two metal structures in the gate. This causes charge to tunnel into the secondary oxide, ultimately changing it into a conductor. This phenomenon is called Fowler-Nordheim tunneling. The new threshold voltage is low enough that normal CMOS logic levels are not able to turn the transistors off (i.e., $V_{T2} \leq \text{GND}$). This process is how the device is *programmed*. This process is accomplished using a dedicated programmer; thus, the EPROM must be removed from its system to program. Figure 10.7 shows an overview of a floating-gate transistor and how it is programmed.


Floating-Gate Transistor - Programming

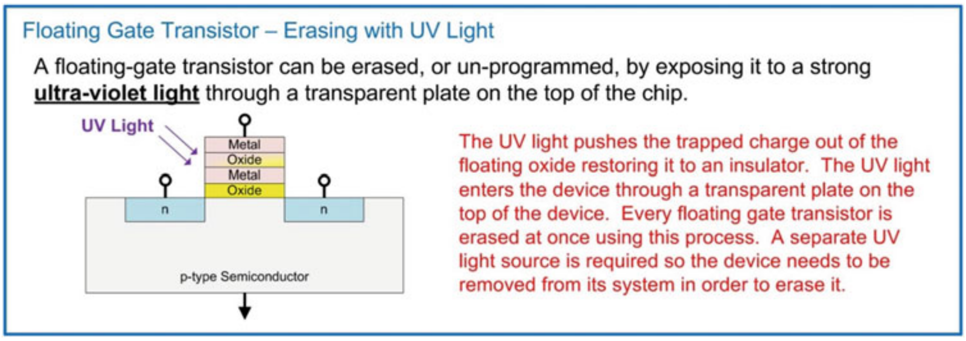
A floating-gate transistor contains an additional metal-oxide layer in the MOS structure.

When a high voltage is applied across the secondary oxide, charge from the two metal plates will tunnel into the insulator and cause it to become conductive. This has the effect of changing the threshold voltage of the device. This is called **programming the device**.

The original threshold voltage (V_{T1}) is designed to be high enough that it cannot be turned on using traditional CMOS logic levels. The programmed threshold voltage (V_{T2}) is so low that the device cannot be turned off using traditional CMOS logic levels.
Gate
Additional "Floating-Gate"
Drain
Source
Metal
Oxide
Metal
Oxide
n
n
p-type Semiconductor
GND
Drain (D)
Gate (G)
V_T + -
Source (S)
Original device with a relatively high V_T (e.g., $\geq V_{CC}$)
Modified device with a relatively low V_T (e.g., $\leq$ GND)
V_{T1}
$V_{program}$
Conductive
V_{T2}
Original Device (Always OFF)
Original Device (Always ON)
V_{CC}
GND
$V_{T1} \geq V_{CC}$
OFF
$V_{T2} \leq$ GND
ON
Normal CMOS levels on the input can't turn the device on.
Normal CMOS levels on the input can't turn the device off.
Unprogrammed
Programmed

Fig. 10.7
Floating-gate transistor – programming

In order to change the floating-gate transistor back into its normal state, the device is exposed to a strong ultraviolet light source. When the UV light strikes the trapped charge in the secondary oxide, it transfers enough energy to the charge particles that they can move back into the metal plates in the gate. This, in effect, *erases* the device and restores it back to a state with a high threshold voltage. EPROMs contain a transparent window on the top of their package that allows the UV light to strike the devices. The EPROM must be removed from its system to perform the erase procedure. When the UV light erase procedure is performed, every device in the memory array is erased. EPROMs are a significant improvement over PROMs because they can be programmed multiple times; however, the programming and erase procedures are manually intensive and require an external programmer and external eraser. Figure 10.8 shows the erase procedure for a floating-gate transistor using UV light.



Floating Gate Transistor – Erasing with UV Light

A floating-gate transistor can be erased, or un-programmed, by exposing it to a strong **ultra-violet light** through a transparent plate on the top of the chip.

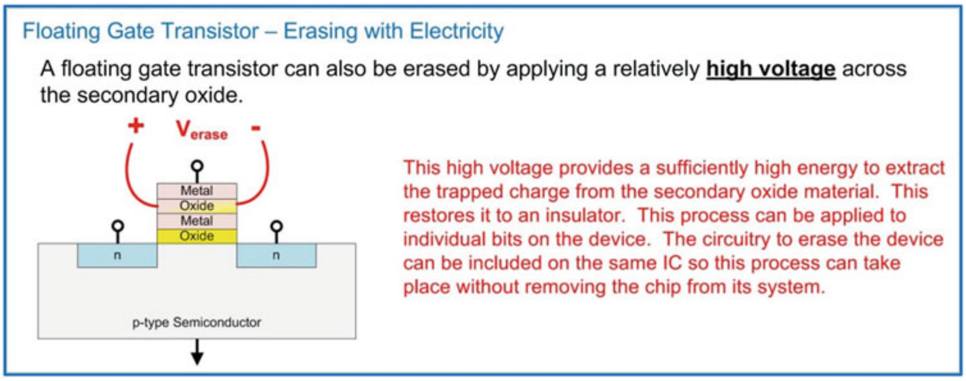
The UV light pushes the trapped charge out of the floating oxide restoring it to an insulator. The UV light enters the device through a transparent plate on the top of the device. Every floating gate transistor is erased at once using this process. A separate UV light source is required so the device needs to be removed from its system in order to erase it.

Fig. 10.8
Floating-gate transistor – erasing with UV light

An EPROM array is created in the exact same manner as in a PROM array with the exception that additional programming circuitry is placed on the IC and a transparent window is included on the package to facilitate erasing. An EPROM is non-volatile and read-only since the programming procedure takes place outside of its destination system.

10.2.5 Electrically Erasable Programmable Read-Only Memory (EEPROM)

In order to address the inconvenient programming and erasing procedures associated with EPROMs, the electrically erasable programmable ROM (EEPROM) was created. In this type of circuit, the floating-gate transistor is erased by applying a large electric field across the secondary oxide. This electric field provides the energy to move the trapped charge from the secondary oxide back into the metal plates of the gate. The advantage of this approach is that the circuitry to provide the large electric field can be generated using circuitry on the same substrate as the memory array, thus eliminating the need for an external UV light eraser. In addition, since the circuitry exists to generate large on-chip voltages, the device can also be programmed without the need for an external programmer. This allows an EEPROM to be programmed and erased while it resides in its target environment. Figure 10.9 shows the procedure for erasing a floating-gate transistor using an electric field.



Floating Gate Transistor – Erasing with Electricity

A floating gate transistor can also be erased by applying a relatively **high voltage** across the secondary oxide.

This high voltage provides a sufficiently high energy to extract the trapped charge from the secondary oxide material. This restores it to an insulator. This process can be applied to individual bits on the device. The circuitry to erase the device can be included on the same IC so this process can take place without removing the chip from its system.

Fig. 10.9
Floating-gate transistor – erasing with electricity

Early EEPROMs were very slow and had a limited number of program/erase cycles; thus, they were classified into the category of non-volatile, read-only memory. Modern floating-gate transistors are now capable of access times on scale with other volatile memory systems; thus, they have evolved into one of the few non-volatile, read/write memory technologies used in computer systems today.

10.2.6 FLASH Memory

One of the early drawbacks of EEPROM was that the circuitry that provided the capability to program and erase individual bits also added to the size of each individual storage element. FLASH EEPROM was a technology that attempted to improve the density of floating-gate memory by programming and erasing in large groups of data, known as *blocks*. This allowed the individual storage cells to shrink and provided higher-density memory parts. This new architecture was called *NAND FLASH* and provided faster write and erase times coupled with higher-density storage elements. The limitation of NAND FLASH was that reading and writing could only be accomplished in a block-by-block basis. This characteristic precluded the use of NAND FLASH for run-time variables and data storage but was well suited for streaming applications such as audio/video and program loading. As NAND FLASH technology advanced, the block size began to shrink, and the software adapted to accommodate the block-by-block data access. This expanded the applications that NAND FLASH could be deployed in. Today, NAND FLASH memory is used in nearly all portable devices (e.g., smartphones, tablets), and its use in solid-state hard drives is on pace to replace hard disk drives and optical disks as the primary non-volatile storage medium in modern computers.

In order to provide individual word access, NOR FLASH was introduced. In NOR FLASH, circuitry is added to provide individual access to data words. This architecture provided faster read times than NAND FLASH, but the additional circuitry causes the write and erase times to be slower and the individual storage cell size to be larger. Due to NAND FLASH having faster write times and higher density, it is seeing broader-scale adoption compared to NOR FLASH despite only being able to access information in blocks. NOR FLASH is considered random access memory, while NAND FLASH is typically not; however, as the block size of NAND FLASH is continually reduced, its use for variable storage is becoming more attractive. All FLASH memory is non-volatile and read/write.

Concept Check

CC10.2 Which of the following is suitable for implementation in a read-only memory?

A) Variables that a computer program needs to continuously update.

B) Information captured by a digital camera.

C) A computer program on a spacecraft.

D) Incoming digitized sound from a microphone.

10.3 Volatile Memory Technology

This section describes some common volatile memory technologies used to store digital information.

10.3.1 Static Random Access Memory (SRAM)

Static random access memory (SRAM) is a semiconductor technology that stores information using a cross-coupled inverter feedback loop. Figure 10.10 shows the schematic for the basic SRAM storage cell. In this configuration, two access transistors (M1 and M2) are used to read and write from the storage cell. The cell has two complementary ports called Bit Line (BL) and Bit Line' (BLn). Due to the inverting functionality of the feedback loop, these two ports will always be the complement of each other. This behavior is advantageous because the two lines can be compared to each other to determine the data value. This allows the voltage levels used in the cell to be lowered while still being able to detect the stored data value. Word lines are used to control the access transistors. This storage element takes six CMOS transistors to implement and is often called a 6T configuration. The advantage of this memory cell is that it has very fast performance compared to other sub-systems because of its underlying technology being simple CMOS transistors. SRAM cells are commonly implemented on the same IC substrate as the rest of the system, thus allowing a fully integrated system to be realized. SRAM cells are used for cache memory in computer systems.

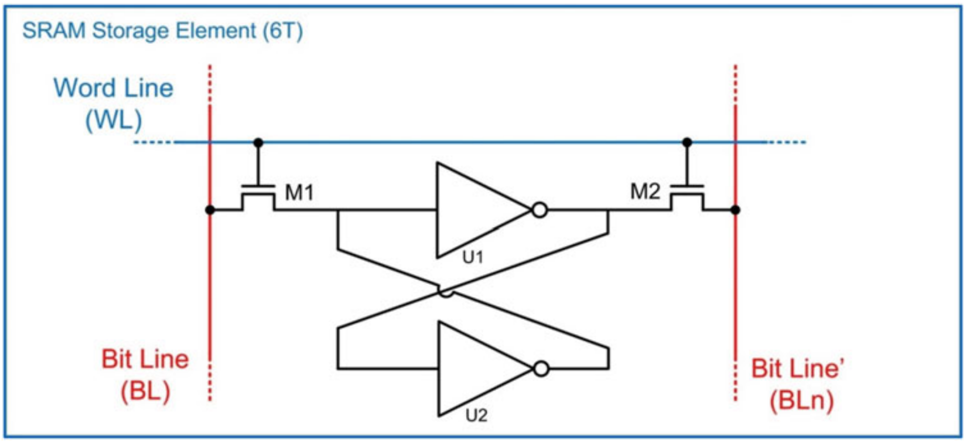


Fig. 10.10
SRAM Storage Element (6T)

To build an SRAM memory system, cells are arranged in an array pattern. Figure 10.11 shows a 4×4 SRAM array topology. In this configuration, word lines are shared horizontally across the array in order to provide addressing capability. An address decoder is used to convert the binary-encoded address into the appropriate word line assertions. N storage cells are attached to the word line to provide the desired data word width. Bit lines are shared vertically across the array in order to provide data access (either read or write). A data line controller handles whether data is read from or written to the cells based on an external write enable (WE) signal. When WE is asserted (WE = 1), data will be written to the cells. When WE is de-asserted (WE = 0), data will be read from the cells. The data line controller also handles determining the correct logic value read from the cells by comparing BL to BLn. As more cells are added to the bit lines, the signal magnitude being driven by the storage cells diminishes due to the additional loading of the other cells. This is where having complementary data signals (BL and BLn) is advantageous because this effectively doubles the magnitude of the storage cell outputs. The comparison of BL to BLn is handled using a *differential amplifier* that produces a full logic level output even when the incoming signals are very small.

SRAM is volatile memory because when the power is removed, the cross-coupled inverters are not able to drive the feedback loop and the data is lost. SRAM is also read/write memory because the storage cells can be easily read from or written to during normal operation.

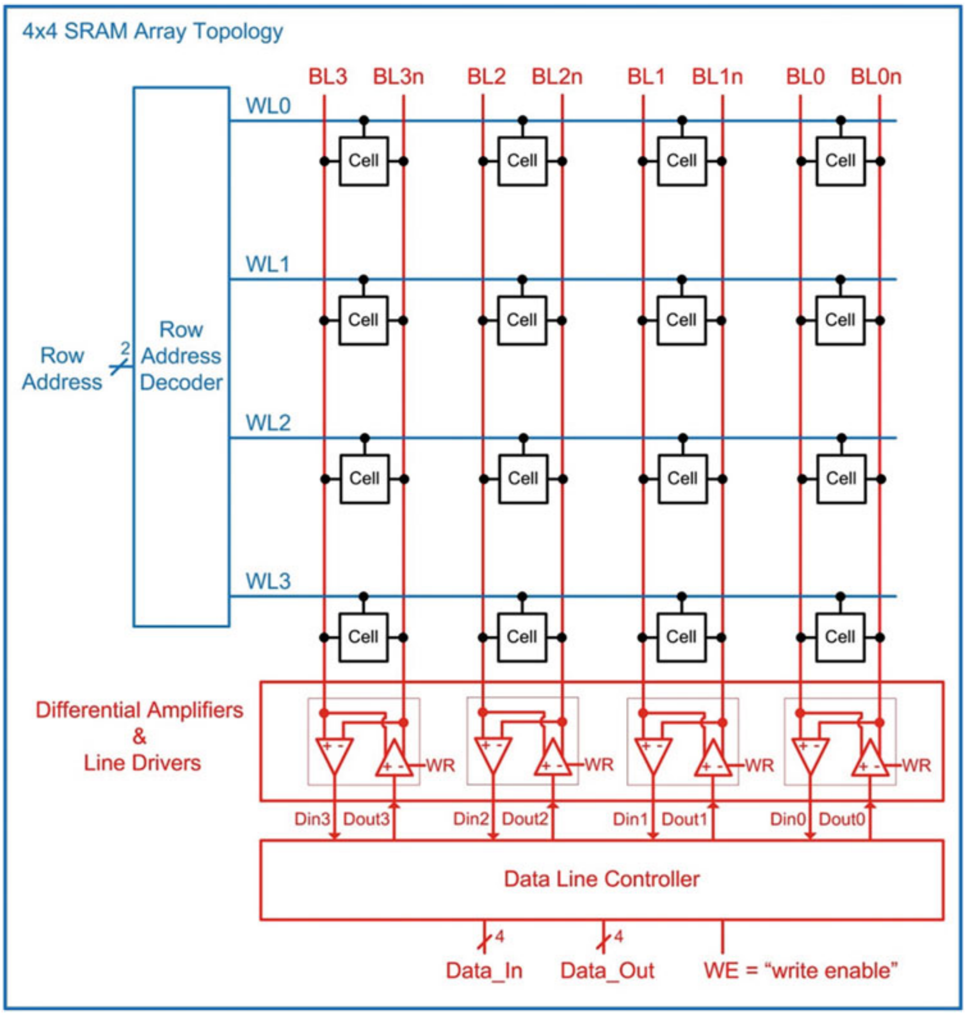


Fig. 10.11
4×4 SRAM array topology

Let's look at the operation of the SRAM array when writing the 4-bit word "0111" to address "01." Figure 10.12 shows a graphical depiction of this operation. In this write cycle, the row address decoder observes the address input "01" and asserts WL1. Asserting this word line enables all of the access transistors (i.e., M1 and M2 in Fig. 10.10) of the storage cells in this row. The line drivers are designed to have a stronger drive strength than the inverters in the storage cells so that they can override their values during a write. The information "0111" is present on the Data_In bus, and the write enable control line is asserted (WE = 1) to indicate a write. The data line controller passes the information to be stored to the line drivers, which in turn converts each input into complementary signals and drives the bit lines. This overrides the information in each storage cell connected to WL1. The address decoder then de-asserts WL1 and the information is stored.

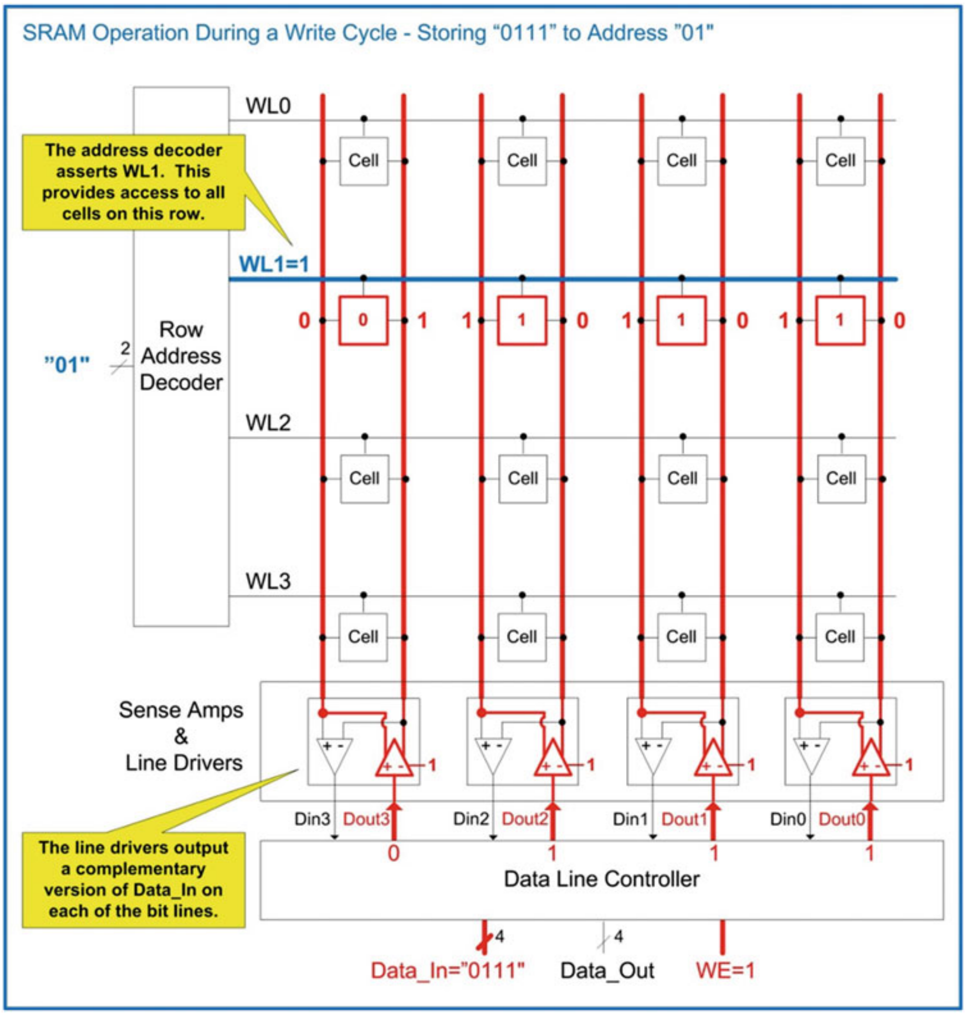

SRAM Operation During a Write Cycle - Storing "0111" to Address "01"
The address decoder asserts WL1. This provides access to all cells on this row.
WL0
WL1=1
WL2
WL3
Row Address Decoder
"01"
2
Cell
0
1
Sense Amps & Line Drivers
The line drivers output a complementary version of Data_In on each of the bit lines.
Din3
Dout3
Din2
Dout2
Din1
Dout1
Din0
Dout0
Data Line Controller
4
Data_In="0111"
Data_Out
WE=1

Fig. 10.12
SRAM operation during a write cycle – storing "0111" to address "01"

Now let's look at the operation of the SRAM array when reading a 4-bit word from address "10." Let's assume that this row was storing the value "1010." Figure 10.13 shows a graphical depiction of this operation. In this read cycle, the row address decoder asserts WL2, which allows the SRAM cells to drive their respective bit lines. Note that each cell drives a complementary version of its stored value. The input control line is de-asserted (WE = 0), which indicates that the sense amplifiers will read the BL and BLn lines in order to determine the full logic value stored in each cell. This logic value is then routed to the Data_Out port of the array. In an SRAM array, reading from the cell does not impact the contents of the cell. Once the read is complete, WL2 is de-asserted and the read cycle is complete.

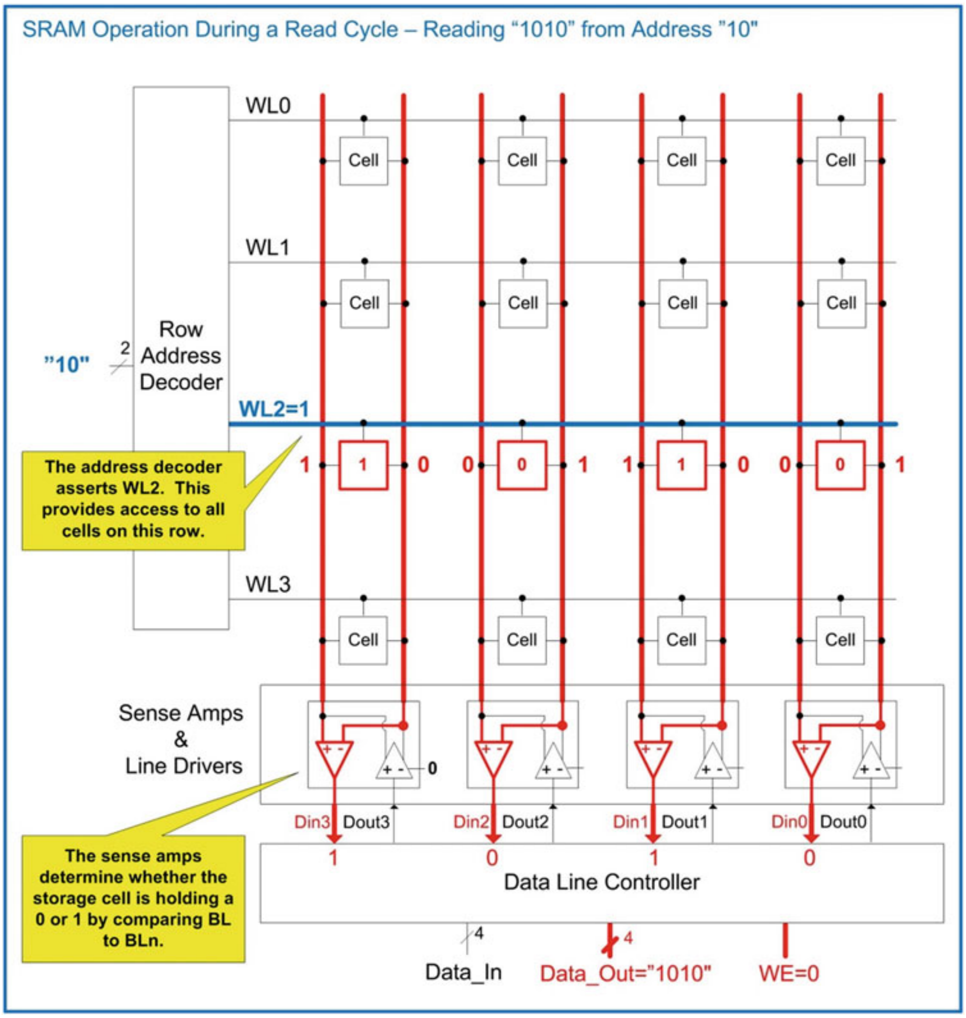

SRAM Operation During a Read Cycle – Reading “1010” from Address ”10”
WL0
WL1
WL2=1
WL3
Cell
Row Address Decoder
”10”
2
1 1 0
0 0 1
1 1 0
0 0 1
The address decoder asserts WL2. This provides access to all cells on this row.
Sense Amps & Line Drivers
0
The sense amps determine whether the storage cell is holding a 0 or 1 by comparing BL to BLn.
Din3 Dout3
Din2 Dout2
Din1 Dout1
Din0 Dout0
1 0 1 0
Data Line Controller
4
Data_In
Data_Out="1010"
WE=0

Fig. 10.13
SRAM operation during a read cycle – reading “0101” from address “10”

10.3.2 Dynamic Random Access Memory (DRAM)

Dynamic random access memory (DRAM) is a semiconductor technology that stores information using a capacitor. A capacitor is a fundamental electrical device that stores charge. Figure 10.14 shows the schematic for the basic DRAM storage cell. The capacitor is accessed through a transistor (M1). Since this storage element takes one transistor and one capacitor, it is often referred to as a 1T1C configuration. Just as in SRAM memory, word lines are used to access the storage elements. The term *digit line* is used to describe the vertical connection to the storage cells. DRAM has an advantage over SRAM in that the storage element requires less area to implement. This allows DRAM memory to have much higher density compared to SRAM.

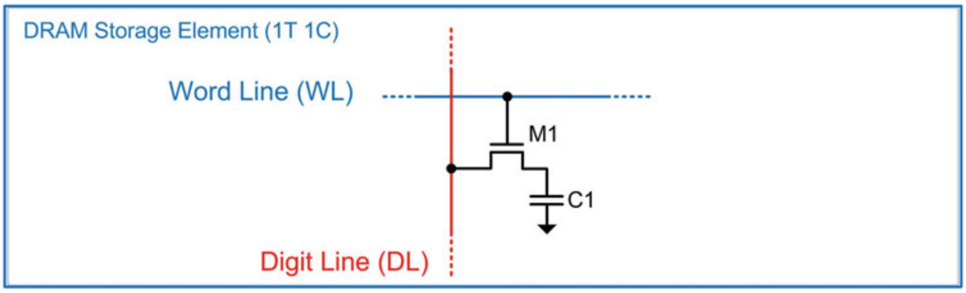


Fig. 10.14
DRAM Storage Element (1T1C)

There are a variety of considerations that must be accounted for when using DRAM. First, the charge in the capacitor will slowly dissipate over time due to the capacitors being non-ideal. If left unchecked, eventually the data held in the capacitor will be lost. In order to overcome this issue, DRAM has a dedicated circuit to *refresh* the contents of the storage cell. A refresh cycle involves periodically reading the value stored on the capacitor and then writing the same value back again at full signal strength. This behavior also means that that DRAM is volatile because when the power is removed, and the refresh cycle cannot be performed, the stored data is lost. DRAM is also considered read/write memory because the storage cells can be easily read from or written to during normal operation.

Another consideration when using DRAM is that the voltage of the word line must be larger than V_{CC} in order to turn on the access transistor. In order to turn on an NMOS transistor, the gate terminal must be larger than the source terminal by at least a threshold voltage (V_T). In traditional CMOS circuit design, the source terminal is typically connected to ground (0v). This means the transistor can be easily turned on by driving the gate with a logic 1 (i.e., V_{CC}) since this creates a V_{GS} voltage much larger than V_T. This is not always the case in DRAM. In DRAM, the source terminal is not connected to ground but rather to the storage capacitor. In the worst-case situation, the capacitor could be storing a logic 1 (i.e., V_{CC}). This means that in order for the word line to be able to turn on the access transistor, it must be equal to or larger than ($V_{CC} + V_T$). This is an issue because the highest voltage that the DRAM device has access to is V_{CC}. In DRAM, a *charge pump* is used to create a voltage larger than $V_{CC} + V_T$ that is driven on the word lines. Once this voltage is used, the charge is lost so the line must be pumped up again before its next use. The process of "pumping up" takes time that must be considered when calculating the maximum speed of DRAM. Figure 10.15 shows a graphical depiction of this consideration.

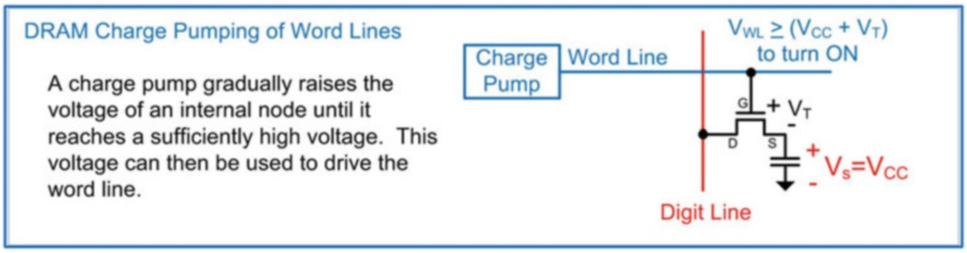

DRAM Charge Pumping of Word Lines
A charge pump gradually raises the voltage of an internal node until it reaches a sufficiently high voltage. This voltage can then be used to drive the word line.
Charge Pump
Word Line
$V_{WL} \geq (V_{CC} + V_T)$ to turn ON
G
+ V_T -
D
S
+ $V_s = V_{CC}$ -
Digit Line

Fig. 10.15
DRAM charge pumping of word lines

Another consideration when using DRAM is how the charge in the capacitor develops into an actual voltage on the digital line when the access transistor is closed. Consider the simple 4x4 array of DRAM cells shown in Fig. 10.16. In this topology, the DRAM cells are accessed using the same approach as in the SRAM array from Fig. 10.11.

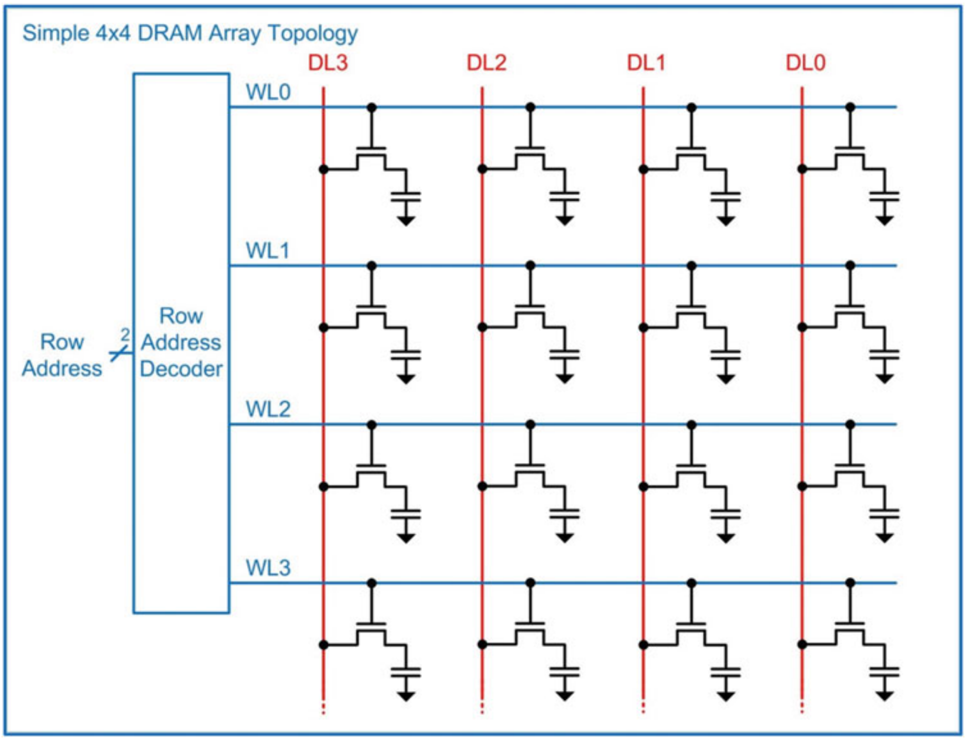
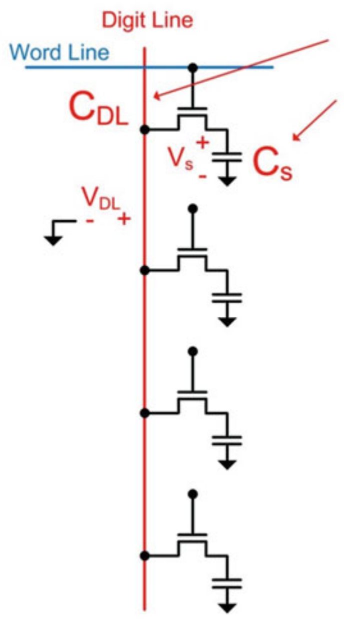


Fig. 10.16
Simple 4×4 DRAM array topology

One of the limitations of this simple configuration is that the charge stored in the capacitors cannot develop a full voltage level across the digit line when the access transistor is closed. This is because the digit line itself has capacitance that impacts how much voltage will be developed. In practice, the capacitance of the digit line (C_{DL}) is much larger than the capacitance of the storage cell (C_S) due to having significantly more area and being connected to numerous other storage cells. This becomes an issue because when the storage capacitor is connected to the digit line, the resulting voltage on the digit line (V_{DL}) is much less than the original voltage on the storage cell (V_S). This behavior is known as *charge sharing* because when the access transistor is closed, the charge on both capacitors is distributed across both devices and results in a final voltage that depends on the initial charge in the system and the values of the two capacitors. Example 10.1 shows an example of how to calculate the final digit line voltage when the storage cell is connected.

Example: Calculating the Final Digit Line Voltage in a DRAM Based on Charge Sharing



To illustrate how charge sharing limits the voltage that is developed on the digit line, let's consider a simple example where the cell is storing a logic 1 (V_S=+3.3v) and the digit line is initially set to V_{DL}=1.65v. The capacitance of the storage cell is C_S=10 pF while the capacitance of the digit line is C_{DL}=150 pF. We want to solve for the voltage on the digit line <u>after</u> the access transistor is closed.

The principle that guides this problem is "charge conservation". This means that the total amount of charge in the system can neither be created nor destroyed. The amount of charge in the system is dictated by the initial voltage across the capacitors. Since the definition of capacitance is "Charge per Volt", or C=Q/V, we can solve for the total amount of charge in the system prior to the access transistor being closed.

$$Q_{init} \rightarrow \underline{\text{Q in the storage cell}} + \underline{\text{Q on the digital line}}$$

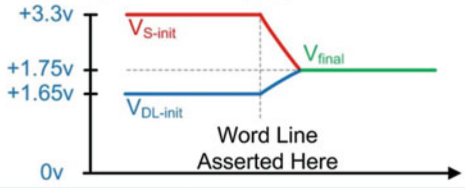
$$\begin{aligned} C_S &= Q_S / V_S & \qquad C_{DL} &= Q_{DL} / V_{DL} \\ 10\text{ pF} &= Q_S / 3.3\text{ v} & \qquad 150\text{ pF} &= Q_{DL} / 1.65\text{ v} \\ Q_S &= 33\text{ pC} & \qquad Q_{DL} &= 247.5\text{ pC} \end{aligned}$$

$$Q_{init} = Q_s + Q_{DL} = 33\text{ pC} + 247.5\text{ pC} = \underline{280.5\text{ pC}}$$

Once the access transistor closes, the two voltages (V_S and V_{DL}) are connected together and are forced to the same voltage (V_S=V_{DL}=V_{final}). Also after the access transistor closes, the final capacitance of the system is the sum of the two capacitors (C_{final}=C_S+C_{DL}=160 pF) since capacitors in parallel are additive. Using charge conservation, the initial charge in the system is equivalent to the final charge in the system (Q_{final}=Q_{init}=280.5 pC). From these values we can calculate the final voltage in the system after the access transistor closes.

$$\begin{aligned} &C_{final} = Q_{final} / V_{final} \\ &160\text{ pF} = 280.5\text{ pC} / V_{final} \\ &V_{final} = 1.75\text{ v} \end{aligned}$$

This means that when storage cell is connected to the digit line, it only moves the voltage by 0.1v (1.76v-1.65v), or 100mv. This is a problem because this voltage difference is not sufficient to be detected using a standard logic gate.



Example 10.1
Calculating the final digit line voltage in a DRAM based on charge sharing

The issue with the charge sharing behavior of a DRAM cell is that the final voltage on the word line is not large enough to be detected by a standard logic gate or latch. In order to overcome this issue, modern DRAM arrays use complementary storage cells and sense amplifiers. The complementary cells store the original data and its complement. Two digit lines (DL and DLn) are used to read the contents of the storage cells. DL and DLn are initially pre-charged to exactly $V_{CC}/2$. When the access transistors are closed, the storage cells will share their charge with the digit lines and move them slightly away from $V_{CC}/2$ in different directions. This allows twice the voltage difference to be developed during a read. A sense amplifier is then used to boost this small voltage difference into a full logic level that can be read by a standard logic gate or latch. Figure 10.17 shows the modern DRAM array topology based on complementary storage cells.

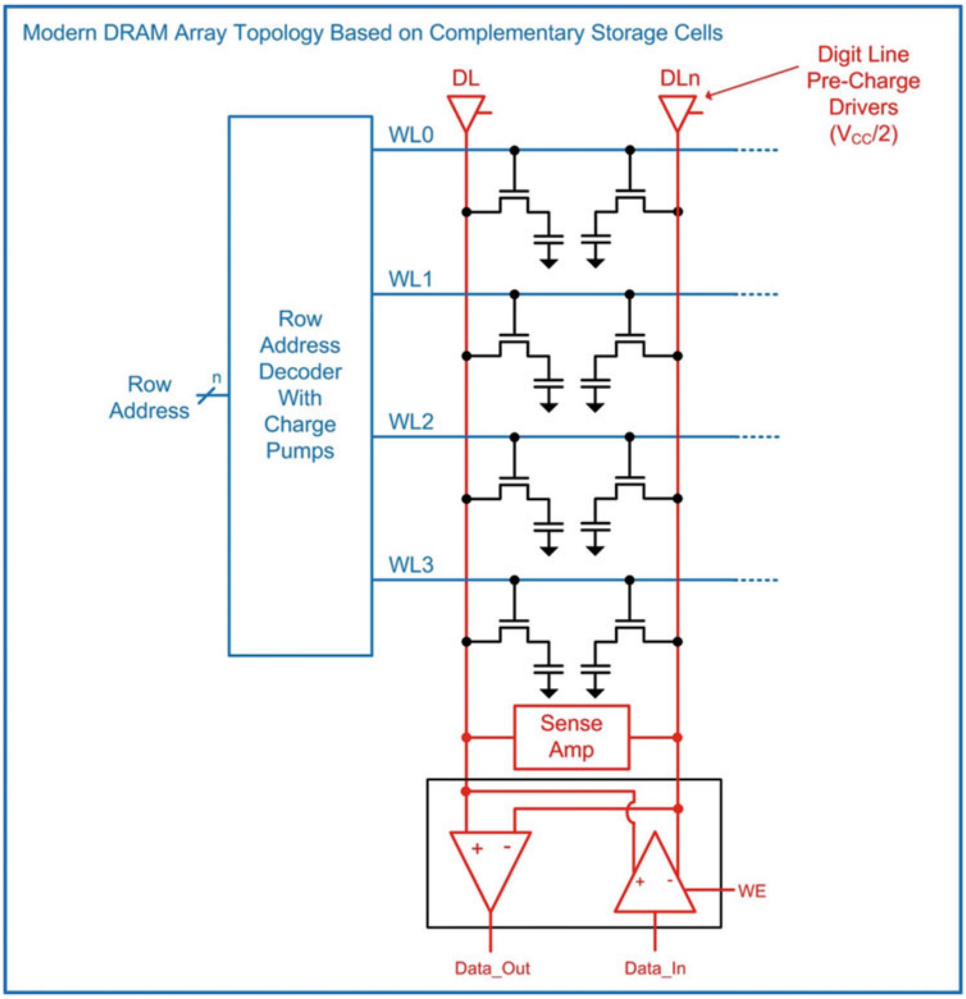

Modern DRAM Array Topology Based on Complementary Storage Cells
Digit Line Pre-Charge Drivers ($V_{CC}/2$)
DL
DLn
WL0
WL1
WL2
WL3
Row Address Decoder With Charge Pumps
Row Address
n
Sense Amp
+
-
WE
Data_Out
Data_In

Fig. 10.17
Modern DRAM array topology based on complementary storage cells

The sense amplifier is designed to boost small voltage deviations from $V_{CC}/2$ on DL and DLn to full logic levels. The sense amplifier sits in between DL and DLn and has two complementary networks, the N-sense amplifier and the P-sense amplifier. The N-sense amplifier is used to pull a signal that is below $V_{CC}/2$ (either DL or DLn) down to GND. A control signal (N-Latch or NLATn) is used to turn on this network. The P-sense amplifier is used to pull a signal that is above $V_{CC}/2$ (either DL or DLn) up to V_{CC}. A control signal (active pull-up or ACT) is used to turn on this network. The two networks are activated in a sequence with the N-sense network activating first. Figure 10.18 shows an overview of the operation of a DRAM sense amplifier.

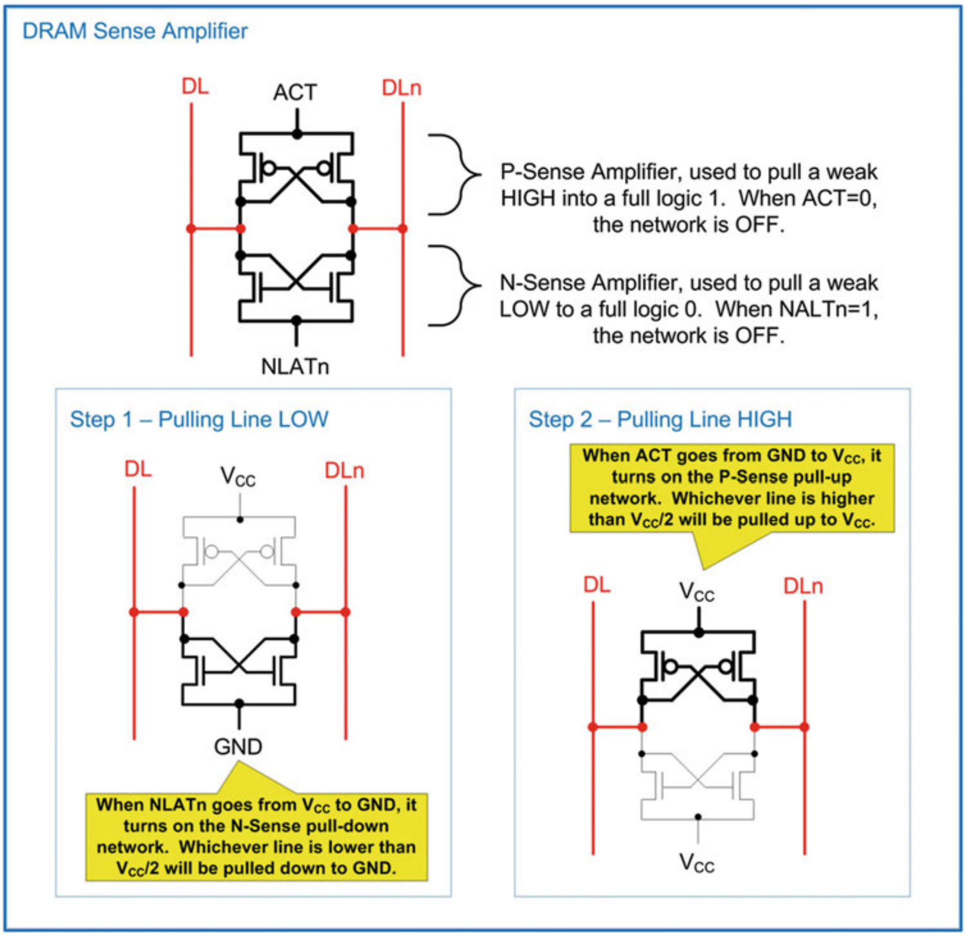

DRAM Sense Amplifier
DL
ACT
DLn
P-Sense Amplifier, used to pull a weak HIGH into a full logic 1. When ACT=0, the network is OFF.
N-Sense Amplifier, used to pull a weak LOW to a full logic 0. When NALTn=1, the network is OFF.
NLATn
Step 1 – Pulling Line LOW
DL
V_{CC}
DLn
GND
When NLATn goes from V_{CC} to GND, it turns on the N-Sense pull-down network. Whichever line is lower than $V_{CC}/2$ will be pulled down to GND.
Step 2 – Pulling Line HIGH
When ACT goes from GND to V_{CC}, it turns on the P-Sense pull-up network. Whichever line is higher than $V_{CC}/2$ will be pulled up to V_{CC}.
DL
V_{CC}
DLn
V_{CC}

Fig. 10.18
DRAM sense amplifier

Let's now put everything together and look at the operation of a DRAM system during a read operation. Figure 10.19 shows a simplified timing diagram of a DRAM read cycle. This diagram shows the critical signals and their values when reading a logic 1. Notice that there is a sequence of steps that must be accomplished before the information in the storage cells can be retrieved.

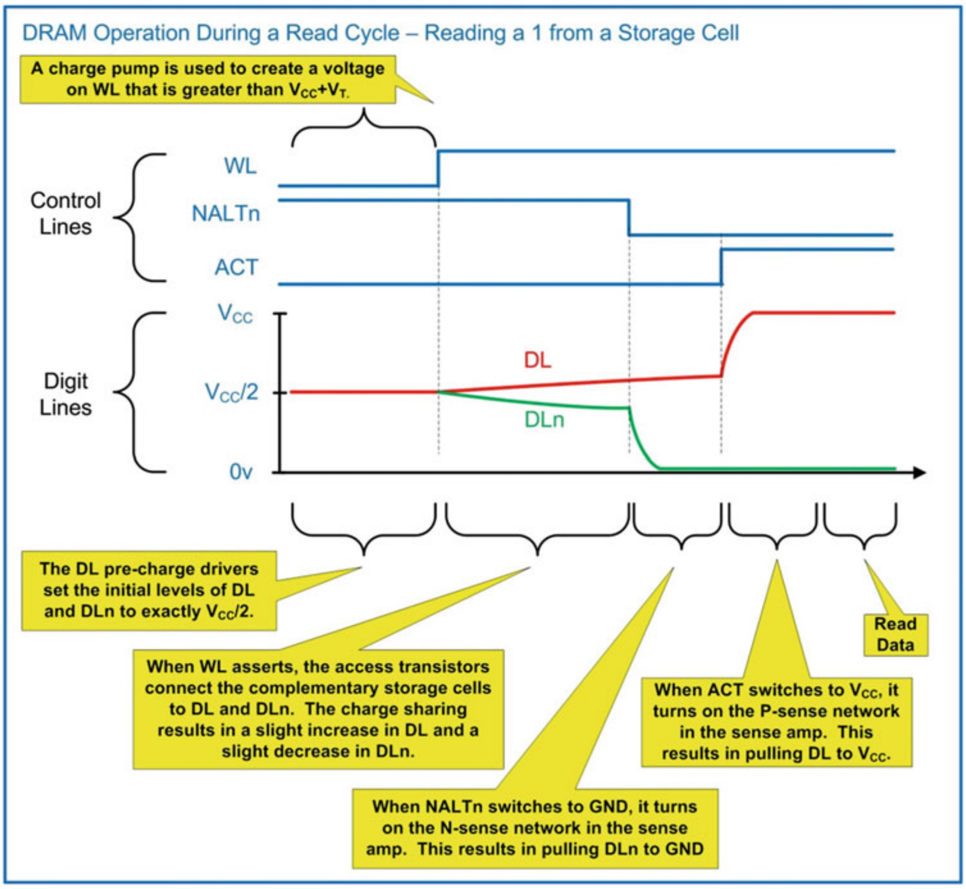

DRAM Operation During a Read Cycle – Reading a 1 from a Storage Cell
A charge pump is used to create a voltage on WL that is greater than $V_{CC}+V_T$.
Control Lines
WL
NALTn
ACT
Digit Lines
V_{CC}
$V_{CC}/2$
0v
DL
DLn
The DL pre-charge drivers set the initial levels of DL and DLn to exactly $V_{CC}/2$.
When WL asserts, the access transistors connect the complementary storage cells to DL and DLn. The charge sharing results in a slight increase in DL and a slight decrease in DLn.
When NALTn switches to GND, it turns on the N-sense network in the sense amp. This results in pulling DLn to GND
When ACT switches to V_{CC}, it turns on the P-sense network in the sense amp. This results in pulling DL to V_{CC}.
Read Data

Fig. 10.19
DRAM operation during a read cycle – reading a 1 from a storage cell

A DRAM write operation is accomplished by opening the access transistors to the complementary storage cells using WL, disabling the pre-charge drivers and then writing full logic level signals to the storage cells using the Data_In line driver.

Concept Check

CC10.3 Which of the following is suitable for implementation in a read/write memory?

A) A look up table containing the values of sine.

B) Information captured by a digital camera.

C) The boot up code for a computer.

D) A computer program on a spacecraft

10.4 Modeling Memory with VHDL

10.4.1 Read-Only Memory in VHDL

Modeling of memory in VHDL is accomplished using the *array* data type. Recall the syntax for declaring a new array type below:

```
type name is array (<range>) of <element_type>;
```

To create the ROM array, a new type is declared (e.g., ROM_type) that is an array. The *range* represents the addressing of the memory array and is provided as an integer. The *element_type* of the array specifies the data type to be stored at each address and represents the data in the memory array. The type of the element should be std_logic_vector with a width of N. To define a 4×4 array of memory, we would use the following syntax.

Example:

```
type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);
```

Notice that the address is provided as an integer (0–3). This will require 2 address bits. Also notice that this defines 4-bit data words. Next, we define a new constant of type ROM_type. When defining a constant, we provide the contents at each address.

Example:

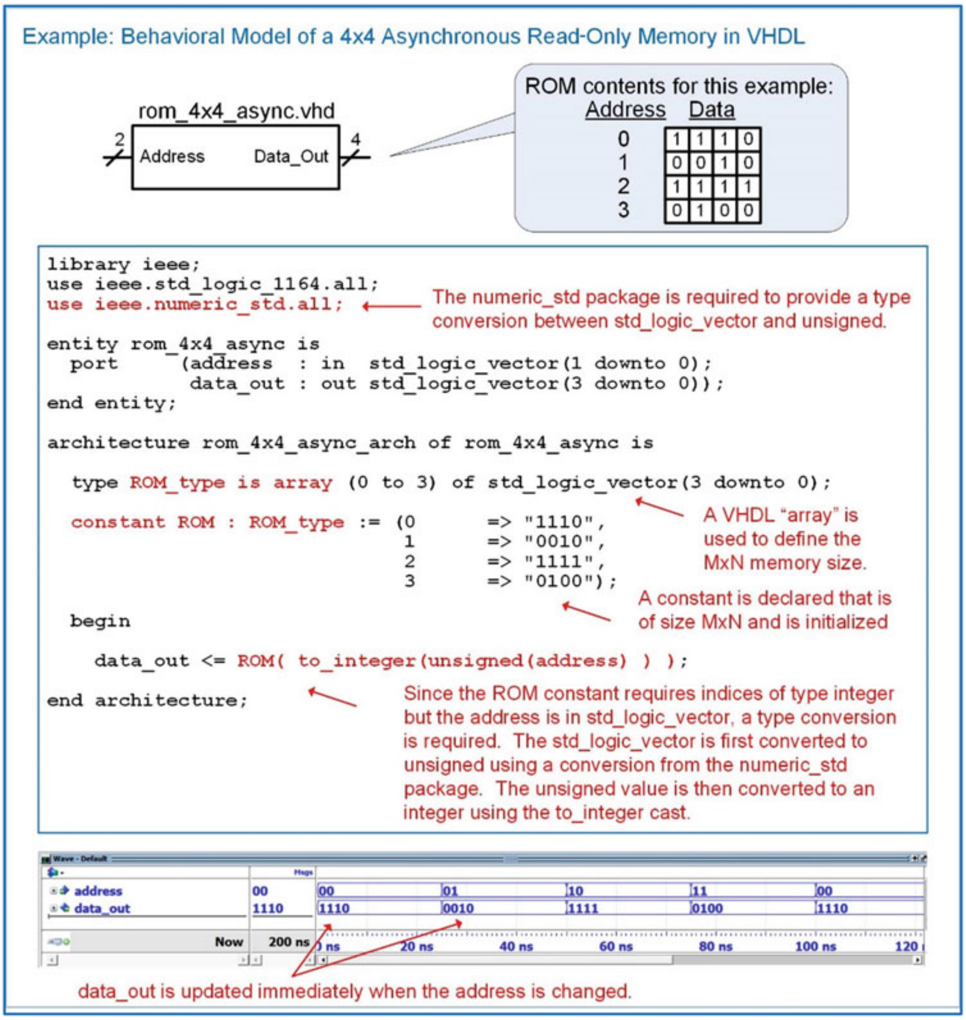
```
constant ROM : ROM_type := (0 => "1110",
                            1 => "0010",
                            2 => "1111",
                            3 => "0100");
```

At this point, the ROM array is declared and initialized. In order to model the read behavior, a concurrent signal assignment is used. The assignment will be made to the output data_out based on the incoming address. The assignment to data_out will be the contents of the constant ROM at a particular address. Since the index of a VHDL array needs to be provided as an integer (e.g., 0, 1, 2, 3) and the address of the memory system is provided as a std_logic_vector, a type conversion is required. Since there is no direct conversion from type std_logic_vector to integer, two conversions are required. The first step is to convert the address from std_logic_vector to unsigned using the *unsigned* type conversion. This conversion exists within the numeric_std package. The second step is to convert the address from unsigned to integer using the *to_integer* conversion. The final assignment is as follows:

Example:

```
data_out <= ROM(to_integer(unsigned(address)));
```

Example 10.2 shows the entire VHDL model for this memory system and the simulation waveform. In the simulation, each possible address is provided (i.e., “00,” “01,” “10,” and “11”). For each address, the corresponding information appears on the data_out port. Since this is an asynchronous memory system, the data appears immediately upon receiving a new address.


Example: Behavioral Model of a 4x4 Asynchronous Read-Only Memory in VHDL

Address	Data
0	1110
1	0010
2	1111
3	0100

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

entity rom_4x4_async is
  port     (address  : in   std_logic_vector(1 downto 0);
            data_out : out  std_logic_vector(3 downto 0));
end entity;

architecture rom_4x4_async_arch of rom_4x4_async is

  type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);

  constant ROM : ROM_type := (0      => "1110",
                              1      => "0010",
                              2      => "1111",
                              3      => "0100");

  begin

    data_out <= ROM( to_integer(unsigned(address) ) );

end architecture;
```

The numeric_std package is required to provide a type conversion between std_logic_vector and unsigned.

A VHDL "array" is used to define the MxN memory size.

A constant is declared that is of size MxN and is initialized

Since the ROM constant requires indices of type integer but the address is in std_logic_vector, a type conversion is required. The std_logic_vector is first converted to unsigned using a conversion from the numeric_std package. The unsigned value is then converted to an integer using the to_integer cast.

```
Wave - Default
                 Msgs
address          00      00        01        10        11        00
data_out         1110    1110      0010      1111      0100      1110

Now              200 ns  ns        20 ns     40 ns     60 ns     80 ns     100 ns    120
```

data_out is updated immediately when the address is changed.

Example 10.2
Behavioral model of a 4×4 asynchronous read-only memory in VHDL

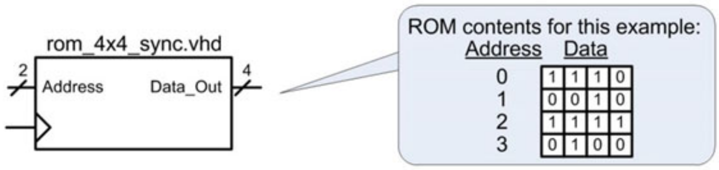
Latency can be modeled in memory systems by using delayed signal assignments. In the above example, if the memory system had a latency of 5 ns, this could be modeled using the following approach:

Example:

```
data_out <= ROM(to_integer(unsigned(address))) after 5 ns;
```

A synchronous ROM can be created in a similar manner. In this approach, a clock edge is used to trigger when the data_out port is updated. A sensitivity list is used that contains only the signal clock to trigger the assignment. A rising edge condition is then used in an if/then statement to make the assignment only on a rising edge. Example 10.3 shows the VHDL model and simulation waveform for this system. Notice that prior to the first clock edge, the simulator does not know what to assign to data_out, so it lists the value as *uninitialized*.

Example: Behavioral Model of a 4x4 Synchronous Read-Only Memory in VHDL

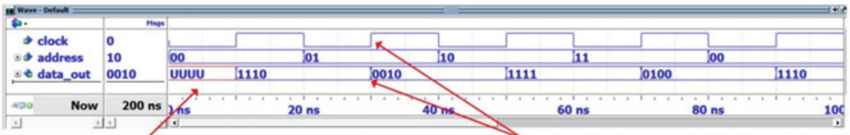


Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

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

entity rom_4x4_sync is
  port      (clock    : in  std_logic;
             address  : in  std_logic_vector(1 downto 0);
             data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rom_4x4_sync_arch of rom_4x4_sync is

  type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);

  constant ROM : ROM_type := (0      => "1110",
                              1      => "0010",
                              2      => "1111",
                              3      => "0100");
  begin

   MEMORY : process (clock)
     begin
       if (clock'event and clock='1') then
          data_out <= ROM( to_integer(unsigned(address) ) );
      end if;
   end process;

end architecture;
```

To model synchronous behavior, the clock is placed in the sensitivity list, and a rising edge condition is used to trigger the assignment.

When there is not a clock edge, the memory will hold its last output on data_out.



Before the first clock edge, the simulator doesn't know what the output should be.

The data does not appear on the output until a rising edge of clock.

Example 10.3
Behavioral model of a 4×4 synchronous read-only memory in VHDL

10.4.2 Read/Write Memory in VHDL

In a read/write memory model, a new type is created using a VHDL *array* (e.g., RW_type) that defines the size of the storage system. To create the memory, a new signal is declared with the array type.

Example:

```
type RW_type is array (0 to 3) std_logic_vector(3 downto 0);
signal RW : RW_type;
```

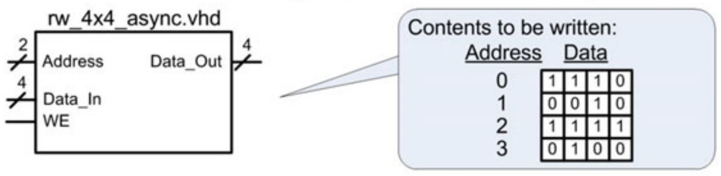
Note that a signal is used in a read/write system as opposed to a constant as in the read-only memory system. This is because a read/write system is uninitialized until it is written to. A process is then used to model the behavior of the memory system. Since this is an asynchronous system, all inputs are listed in the sensitivity list (i.e., address, WE, and data_in). The process first checks whether the write enable line is asserted (WE = 1), which indicates a write cycle is being performed. If it is, then it makes an assignment to the RW signal at the location provided by the address input with the data provided by the data_in input. Since the RW array is indexed using integers, type conversions are required to convert the address from std_logic_vector to integer. When WE is not asserted (WE = 0), a read cycle is being performed. In this case, the process makes an assignment to data_out with the contents stored at the provided address. This assignment also requires type conversions to change the address from std_logic_vector to integer. The following syntax implements this behavior.

Example:

```
MEMORY: process (address, WE, data_in)
  begin
    if (WE = '1') then
      RW(to_integer(unsigned(address))) <= data_in;
    else
      data_out <= RW(to_integer(unsigned(address)));
    end if;
end process;
```

A read/write memory does not contain information until its storage locations are written to. As a result, if the memory is read from before it has been written to, the simulation will return *uninitialized*. Example 10.4 shows the entire VHDL model for an asynchronous read/write memory and the simulation waveform showing read/write cycles.

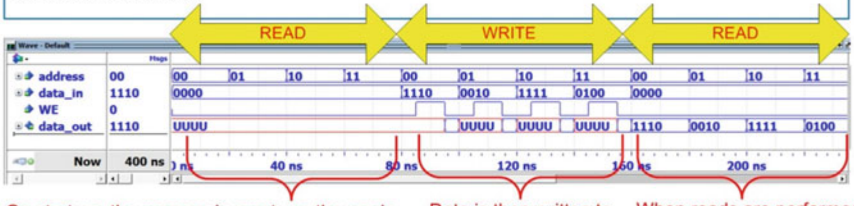
Example: Behavioral Model of a 4x4 Asynchronous Read/Write Memory in VHDL



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

entity rw_4x4_async is
  port      (address   : in  std_logic_vector(1 downto 0);
             data_in   : in  std_logic_vector(3 downto 0);
             WE        : in  std_logic;
             data_out  : out std_logic_vector(3 downto 0));
end entity;

architecture rw_4x4_async_arch of rw_4x4_async is

  type RW_type is array (0 to 3) of std_logic_vector(3 downto 0);

  signal RW : RW_type;

  begin

   MEMORY: process (address, WE, data_in)
     begin
        if (WE = '1') then
          RW(to_integer(unsigned(address))) <= data_in;
        else
          data_out <= RW(to_integer(unsigned(address)));
        end if;
   end process;

end architecture;
```

A signal is used since the read/write memory is uninitialized until it is written to.

Type conversions are needed for both reads and writes to RW.



On start-up, the memory is empty so the reads from the four addresses yield "uninitialized".

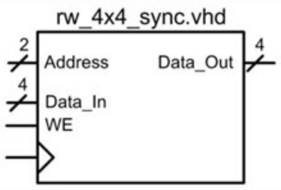
Data is then written to the four addresses.

When reads are performed again, the data that was written appears.

Example 10.4
Behavioral model of a 4×4 asynchronous read/write memory in VHDL

A synchronous read/write memory is made in a similar manner with the exception that a clock is used to trigger the signal assignments in the sensitivity list. The WE signal acts as a synchronous control signal indicating whether assignments are read from or written to the RW array. Example 10.5 shows the entire VHDL model for a synchronous read/write memory and the simulation waveform showing both read and write cycles.

Example: Behavioral Model of a 4x4 Synchronous Read/Write Memory in VHDL



Contents to be written:

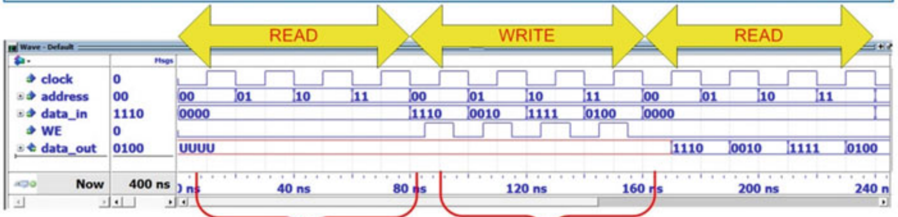
Address	Data
0	1110
1	0010
2	1111
3	0100

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

entity rw_4x4_sync is
  port      (clock    : in  std_logic;
             address  : in  std_logic_vector(1 downto 0);
             data_in  : in  std_logic_vector(3 downto 0);
             WE       : in  std_logic;
             data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rw_4x4_sync_arch of rw_4x4_sync is

  type RW_type is array (0 to 3) of std_logic_vector(3 downto 0);

  signal RW : RW_type;

  begin

   MEMORY : process (clock)
     begin
        if (clock'event and clock='1') then
           if (WE = '1') then
             RW(to_integer(unsigned(address))) <= data_in;
           else
             data_out <= RW(to_integer(unsigned(address)));
           end if;
        end if;
   end process;

end architecture;
```

Synchronous behavior is modeled by listing clock in the sensitivity list and using a rising edge condition.

The WE control signal dictates whether information is read or written to the RW array.

Type conversions are needed for both reads and writes to RW.



Reads are performed on the rising edge of clock when WE=0.

Data is written on the rising edge of clock when WE=1.

Example 10.5
Behavioral model of a 4×4 synchronous read/write memory in VHDL

Concept Check

CC10.4 Explain the advantage of modeling memory in VHDL without going into the details of the storage cell operation.

A) It allows the details of the storage cell to be abstracted from the functional operation of the memory system.

B) It is too difficult to model the analog behavior of the storage cell.

C) There are too many cells to model so the simulation would take too long.

D) It lets both ROM and R/W memory to be modeled in a similar manner.

Summary

- The term memory refers to large arrays of digital storage. The technology used in memory is typically optimized for storage density at the expense of control capability. This is different from a D-Flip-Flop, which is optimized for complete control at the bit level.
- A memory device always contains an address bus input. The number of bits in the address bus dictates how many storage locations can be accessed. An n-bit address bus can access 2^n (or M) storage locations.
- The width of each storage location (N) allows the density of the memory array to be increased by reading and writing vectors of data instead of individual bits.
- A memory map is a graphical depiction of a memory array. A memory map is useful to give an overview of the capacity of the array and how different address ranges of the array are used.
- A read is an operation in which data is retrieved from memory. A write is an operation in which data is stored to memory.
- An asynchronous memory array responds immediately to its control inputs. A synchronous memory array only responds on the triggering edge of clock.
- Volatile memory will lose its data when the power is removed. Non-volatile memory will retain its data when the power is removed.
- Read-only memory (ROM) is a memory type that cannot be written to during normal operation. Read/write (R/W) memory is a memory type that can be written to during normal operation. Both ROM and R/W memory can be read from during normal operation.
- Random access memory (RAM) is a memory type in which any location in memory can be accessed at any time. In sequential access memory, the data can only be retrieved in a linear sequence. This means that in sequential memory the data cannot be accessed arbitrarily.
- The basic architecture of a ROM consists of intersecting bit lines (vertical) and word lines (horizontal) that contain storage cells at their crossing points. The data is read out of the ROM array using the bit lines. Each bit line contains a pull-up resistor to initially store a logic 1 at each location. If a logic 0 is desired at a certain location, a pull-down transistor is placed on a particular bit line with its gate connected to the appropriate word line. When the storage cell is addressed, the word line will assert and turn on the pull-down transistor producing a logic 0 on the output.
- There are a variety of technologies to implement the pull-down transistor in a ROM. Different ROM architectures include MROMs, PROMs, EPROMs, and EEPROMs. These memory types are non-volatile.
- A R/W memory requires a storage cell that can be both read from and written to during normal operation. A DRAM (dynamic RAM) cell is a storage element that uses a capacitor to hold charge corresponding to a logic value. An SRAM (static RAM) cell is a storage element that uses a cross-coupled inverter pair to hold the value being stored in the positive feedback loop formed by the inverters. Both DRAM and SRAM are volatile and random access.
- The floating-gate transistor enables memory that is both non-volatile and R/W. Modern memory systems based on floating-gate transistor technology allow writing to take place using the existing system power supply

levels. This type of R/W memory is called FLASH. In FLASH memory, the information is read out in blocks; thus, it is not technically random access.

❖ Memory can be modeled in VHDL using the array data type.

Exercise Problems

Section 10.1: Memory Architecture and Terminology

10.1.1 For a 512 k × 32 memory system, how many unique address locations are there? Give the exact number.

10.1.2 For a 512 k × 32 memory system, what is the data width at each address location?

10.1.3 For a 512 k × 32 memory system, what is the *capacity* in bits?

10.1.4 For a 512 k × 32-bit memory system, what is the *capacity* in bytes?

10.1.5 For a 512 k × 32 memory system, how wide does the incoming address bus need to be in order to access every unique address location?

10.1.6 Name the type of memory with the following characteristic: *when power is removed, the data is lost.*

10.1.7 Name the type of memory with the following characteristic: *when power is removed, the memory still holds its information.*

10.1.8 Name the type of memory with the following characteristic: *it can only be read from during normal operation.*

10.1.9 Name the type of memory with the following characteristic: *during normal operation, it can be read and written to.*

10.1.10 Name the type of memory with the following characteristic: *data can be accessed from any address location at any time.*

10.1.11 Name the type of memory with the following characteristic: *data can only be accessed in consecutive order; thus, not every location of memory is available instantaneously.*

Section 10.2: Non-volatile Memory Technology

10.2.1 Name the type of memory with the following characteristic: *this memory is non-volatile and read/write and only provides data access in blocks.*

10.2.2 Name the type of memory with the following characteristic: *this memory uses a floating-gate transistor, can be erased with electricity, and provides individual bit access.*

10.2.3 Name the type of memory with the following characteristic: *this memory is non-volatile and read/write and provides word-level data access.*

10.2.4 Name the type of memory with the following characteristic: *this memory uses a floating-gate transistor that is erased with UV light.*

10.2.5 Name the type of memory with the following characteristic: *this memory is programmed by blowing fuses or anti-fuses.*

10.2.6 Name the type of memory with the following characteristic: *this memory is partially fabricated prior to knowing the information to be stored.*

Section 10.3: Volatile Memory Technology

10.3.1 How many transistors does it take to implement an SRAM cell?

10.3.2 Why doesn't an SRAM cell require a refresh cycle?

10.3.3 Design a VHDL model for the SRAM system shown in Fig. 10.20. Your storage cell should be designed such that its contents can be overwritten by the line driver. Consider using a resolved data type for this behavior that models drive strength (e.g., in std_logic, a 1 has a higher drive strength than an H). You will need to create a system for the differential line driver with enable. This driver will need to contain a high impedance state when disabled. Both your line driver (Din) and receiver (Dout) are differential. These systems can be modeled using simple if/then statements. Create a test bench for your system that will write a 0 to the cell, then read it back to verify the 0 was stored, and then repeat the write/read cycles for a 1.

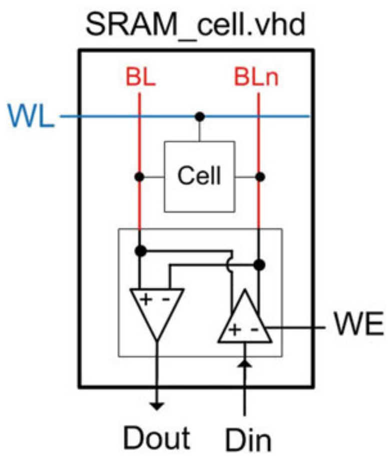


Fig. 10.20
SRAM cell block diagram

10.3.4 Why is a DRAM cell referred to as a 1T/1C configuration?

10.3.5 Why is a charge pump necessary on the word lines of a DRAM array?

10.3.6 Why does a DRAM cell require a refresh cycle?

10.3.7 For the DRAM storage cell shown in Fig. 10.21, solve for the final voltage on the digit line after the access transistor (M1) closes if initially V_S=V_{CC} (**i.e., the cell is storing a 1**). In this system, $C_S = 5$ pF, $C_{DL} = 10$ pF, and $V_{CC} = +3.4$v. Prior to the access transistor closing, the digit line is pre-charged to $V_{CC}/2$.

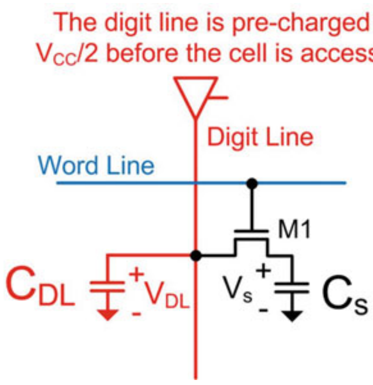
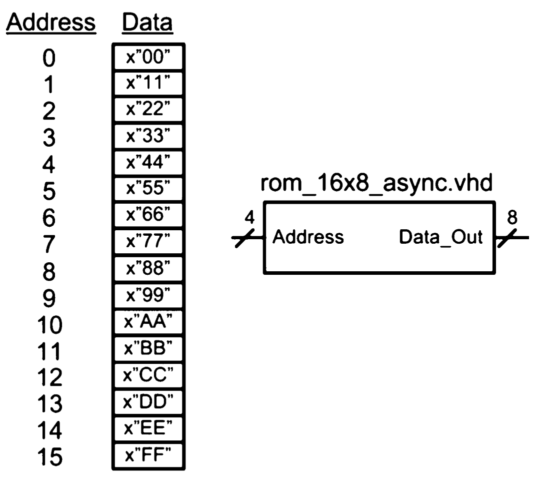

The digit line is pre-charged to
$V_{CC}/2$ before the cell is accessed.
Digit Line
Word Line
M1
C_{DL}
+
V_{DL}
-
V_s
+
-
C_S

Fig. 10.21
DRAM charge sharing exercise

10.3.8 For the DRAM storage cell shown in Fig. 10.21, solve for the final voltage on the digit line after the access transistor (M1) closes if initially V_S = GND (**i.e., the cell is storing a 0**). In this system, $C_S = 5$ pF, $C_{DL} = 10$ pF, and $V_{CC} = +3.4$v. Prior to the access transistor closing, the digit line is pre-charged to $V_{CC}/2$.

Section 10.4: Modeling Memory with VHDL

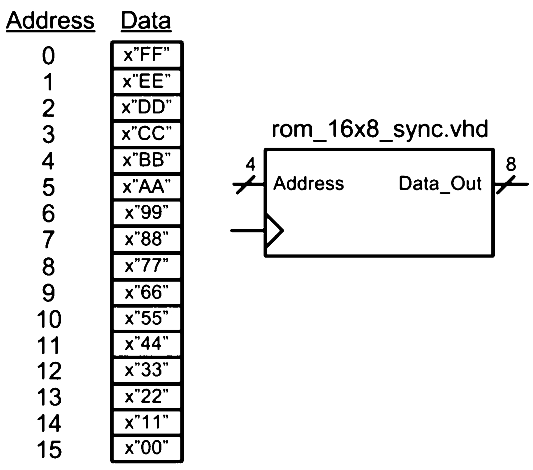
10.4.1 Design a VHDL model for the 16×8, asynchronous, read-only memory system shown in Fig. 10.22. The system should contain the information provided in the memory map. Create a test bench to simulate your model by reading from each of the 16 unique addresses and observing Data_Out to verify it contains the information in the memory map.



Address	Data
0	x"00"
1	x"11"
2	x"22"
3	x"33"
4	x"44"
5	x"55"
6	x"66"
7	x"77"
8	x"88"
9	x"99"
10	x"AA"
11	x"BB"
12	x"CC"
13	x"DD"
14	x"EE"
15	x"FF"

Fig. 10.22
16×8 asynchronous ROM block diagram

10.4.2 Design a VHDL model for the 16×8, synchronous, read-only memory system shown in Fig. 10.23. The system should contain the information provided in the memory map. Create a test bench to simulate your model by reading from each of the 16 unique addresses and observing Data_Out to verify it contains the information in the memory map.



Address	Data
0	x"FF"
1	x"EE"
2	x"DD"
3	x"CC"
4	x"BB"
5	x"AA"
6	x"99"
7	x"88"
8	x"77"
9	x"66"
10	x"55"
11	x"44"
12	x"33"
13	x"22"
14	x"11"
15	x"00"

Fig. 10.23
16×8 synchronous ROM block diagram

10.4.3 Design a VHDL model for the 16×8, asynchronous, read/write memory system shown in Fig. 10.24. Create a test bench to simulate your model. Your test bench should first read from all of the address locations to verify they are uninitialized. Next, your test bench should write unique information to each of the address locations. Finally, your test bench should read from each address location to verify that the information that was written was stored and can be successfully retrieved.

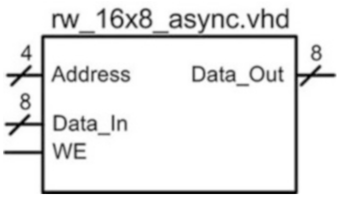


Fig. 10.24
16×8 asynchronous R/W memory block diagram

10.4.4 Design a VHDL model for the 16×8, synchronous, read/write memory system shown in Fig. 10.25. Create a test bench to simulate your model. Your test bench should first read from all of the address locations to verify they are uninitialized. Next, your test bench should write unique information to each of the address locations. Finally, your test bench should read from each address location to verify that the information that was written was stored and can be successfully retrieved.

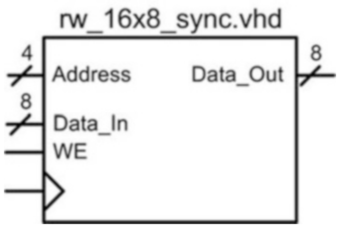


Fig. 10.25
16×8 synchronous R/W memory block diagram