Memory Devices, Circuits, and Subsystem Design

INTRODUCTION

In the previous chapter, we began our study of the hardware architecture of the 8088-and 8086-based microcomputer systems. This included a study of the MPU's memory interface and the circuits needed to connect to a memory subsystem. We examined the memory interface signals, read and write bus cycles, hardware organization of the memory address space, and memory interface circuits. In this chapter, we continue our study of microcomputer hardware by examining the devices, circuits, and techniques used in the design of memory subsystems. For this purpose, this chapter explores the following topics:

9.1 Program and Data Storage Memory
9.2 Read-Only Memory
9.3 Random Access Read/Write Memories
9.4 Parity, the Parity Bit, and Parity-Checker/Generator Circuit
9.5 FLASH Memory
9.6 Wait-State Circuitry
9.7 8088/8086 Microcomputer System Memory Circuitry
9.1 PROGRAM AND DATA STORAGE MEMORY

Memory provides the ability to store and retrieve digital information and is one of the key elements of a microcomputer system. By digital information, we mean that instructions and data are encoded with 0s and 1s and then saved in memory. The ability to store information is made possible by the part of the microcomputer system known as the memory unit. In Chapter 1, we indicated that the memory unit of the microcomputer is partitioned into a primary storage section and secondary storage section. Figure 9-1 illustrates this subdivision of the memory unit.

Secondary storage memory is used for storage of data, information, and programs that are not in use. This part of the memory unit can be slow speed, but it requires very large storage capacity. For this reason, it is normally implemented with magnetic storage devices, such as the floppy disk and hard disk drive. Hard disk drives used in today's personal computers have the ability to store 10 gigabytes (Gbyte) to 80Gbytes of information.

The other part, primary storage memory, is used for working information, such as the instructions of the program currently being run and data that it is processing. This section normally requires high-speed operation but does not normally require very large storage capacity. Therefore, it is implemented with semiconductor memory devices. Most modern personal computers have 128 megabytes (128Mbytes) of primary storage memory.

Figure 9-1 shows that the primary storage memory is further partitioned into program storage memory and data storage memory. The program storage part of the memory subsystem is used to hold information such as the instructions of the program. That is, when a program is executed by the microcomputer, it is read one byte or word at a time from the program storage part of the memory subsystem. These programs can be either permanently stored in memory, which makes them always available for execution, or temporarily loaded into memory before execution. The program storage memory section does not normally contain only instructions, it can also store other fixed information such as constant data and lookup tables.

The program storage memory in a personal computer is implemented exactly this way. It has a fixed part of program memory that contains the basic input/output system

![Figure 9-1](image-url)
(BIOS). These programs are permanently held in a read-only memory device mounted on
the main processor board. Programs held this way in ROM are called firmware because of
their permanent nature. The typically size of a BIOS ROM used in a PC today is 2 megabits
(MB), which equal 256Kbytes.

The much larger part of the program storage memory in a PC is built with dynamic
random access read/write memory devices (DRAMs). They may be either mounted on
the main processor board or on an add-in memory module or board. Use of DRAMs
allows this part of the program storage memory to be either read from or written into. Its
purpose is again to store programs that are to be executed, but in this case they are loaded
into memory only when needed. Programs are normally read in from the secondary stor-
age device, stored in the program storage part of memory, and then run. When the pro-
gram is terminated, the part of the program memory where it resides is given back to the
operating system for reuse. Moreover, if power is turned off, the contents of the RAM-
based part of the program storage memory are lost. Due to the temporary nature of these
programs, they are referred to as software.

Earlier we indicated that the primary storage memory of a microcomputer is typi-
cally 128Mbytes. This number represented the total of the DRAM part of the memory
subsystem and is given as the size of memory because the ROM BIOS is almost negligi-
ble when compared to the amount of DRAM. In the PC, a major part of primary storage
is available for use as program storage memory.

In other microcomputer applications, such as an electronic game or telephone, the
complete program storage memory is implemented with ROM devices.

Information that frequently changes is stored in the data storage part of the
microcomputer's memory subsystem. For instance, the data to be processed by the micro-
computer is held in the data storage part of the primary storage memory. When a pro-
gram is run by the microcomputer, the values of the data can change repeatedly. For
this reason, data storage memory must be implemented with RAM. In a PC, the data
does not automatically reside in the data storage part of memory. Just like software, it
is read into memory from a secondary storage device, such as the hard disk. Any part
of the PCs DRAM can be assigned for data storage. The operating system software
does this.

When a program is run, data are modified while in DRAM and writing them to the
disk saves the new values. Data does not have to be numeric in form; they can also be
alphanumeric characters, codes, and graphical patterns. For instance, when running a
word processor application, the data are alphanumeric and graphical information.

\section{9.2 READ-ONLY MEMORY}

We begin our study of semiconductor memory devices with the \textit{read-only memory}
(ROM). ROM is one type of semiconductor memory device. It is most widely used in
microcomputer systems for storage of the program that determines overall system opera-
tion. The information stored within a ROM integrated circuit is permanent—or
\textit{nonvolatile}. This means that when the power supply of the device is turned off, the stored
information is not lost.
For some ROM devices, information (the microcomputer program) must be built in during manufacturing, and for others the data must be electrically entered. The process of entering the data into a ROM is called programming. As the name ROM implies, once entered into the device this information can be read only. For this reason, these devices are used primarily in applications where the stored information would not change frequently.

Three types of ROM devices are in wide use today: the mask-programmable read-only memory (ROM), the one-time-programmable read-only memory (PROM), and the erasable programmable read-only memory (EPROM). Let us continue by looking more closely into the first type of device, the mask-programmable read-only memory. This device has its data pattern programmed as part of the manufacturing process and is known as mask programming. Once the device is programmed, its contents can never be changed. Because of this fact and the cost for making the programming masks, ROMs are used mainly in high-volume applications where the data will not change.

The other two types of read-only memories, the PROM and EPROM, differ from the ROM in that the user electrically enters the bit pattern for the data. Programming is usually done with an instrument known as an EPROM programmer. Both the PROM and EPROM are programmed in the same way. Once a PROM is programmed, its contents cannot be changed. This is the reason they are sometimes called one-time programmable PROMs. On the other hand, exposing an EPROM to ultraviolet light erases the information it holds. That is, the programmed bit pattern is cleared out to restore the device to its unprogrammed state. In this way, the device can be used over and over again simply by erasing and reprogramming. PROMs and EPROMs are most often used during the design of a product, for early production, when the code of the microcomputer may need frequent changes, and for production in low-volume applications that do not warrant making a mask programmed device.

Figure 9-2(a) shows a typical EPROM programmer unit. Programming units like this usually have the ability to verify that an EPROM is erased, program it with new data, verify correct programming, and read the information out of a programmed EPROM. An erasing unit such as that shown in Fig. 9-2(b) can be used to erase a number of EPROM ICs at one time.

Block Diagram of a Read-Only Memory

Figure 9-3 shows a block diagram of a typical read-only memory. Here we see that the device has three sets of signal lines: the address inputs, data outputs, and control inputs. This block diagram is valid for a ROM, PROM, or EPROM. Let us now look at the function of each of these sets of signal lines.

The address bus is used to input the signals that select between the storage locations within the ROM device. In Fig. 9-3 we find that this bus consists of 11 address lines, A0 through A10. The bits in the address are arranged so that A10 is the MSB and A0 is the LSB. With an 11-bit address, the memory device has $2^{11} = 2048$ unique byte-storage locations. The individual storage locations correspond to consecutive addresses over the range 000000000002 through 111111111112 = 7FF16.
Earlier we pointed out that information is stored inside a ROM, PROM, or EPROM as either a binary 0 or binary 1. Actually, 8 bits of data are stored at every address. Therefore, the organization of the ROM is described as $2048 \times 8$. The total storage capacity of the ROM is identified as the number of bits of information it can hold. We know 2048 bytes corresponds to 16,384 bits; therefore, the device we are describing is actually a 16K bit or 16KB ROM.
By applying the address of a storage location to the address inputs of the ROM, the byte of data held at the addressed location is read out onto the data lines. The block diagram in Fig. 9–3 shows that the data bus consists of eight lines labeled as $O_0$ through $O_7$. Here $O_7$ represents the MSB and $O_0$ the LSB. For instance, applying the address $A_{10} \ldots A_1 A_0 = 10000000000_2 = 400_{16}$ will cause the byte of data held in this storage location to be output as $O_7 O_6 O_5 O_4 O_3 O_2 O_1 O_0$.

**EXAMPLE 9.1**

Suppose the block diagram in Fig. 9–3 had 15 address lines and eight data lines. How many bytes of information can be stored in the ROM? What is its total storage capacity?

**Solution**

With 8 data lines, the number of bytes is equal to the number of locations, which is

$$2^{15} = 32,768 \text{ bytes}$$

This gives a total storage of

$$32,768 \times 8 = 262,144 \text{ bits}$$

The control bus represents the control signals required to enable or disable the ROM, PROM, or the EPROM device. The block diagram in Fig. 9–3 identifies two control inputs: output enable (OE) and chip enable (CE). For example, logic 0 at OE enables the three state outputs, $O_0$ through $O_7$, of the device. If OE is switched to the 1 logic level, these outputs are disabled (put in the high-Z state). Moreover, CE must be at logic 0 for the device to be active. Logic 1 at CE puts the device in a low-power standby mode. When in this state, the data outputs are in the high-Z state independent of the logic level of OE. In this way we see that both OE and CE must be at their active 0 logic levels for the device to be ready for operation.
Read Operation

It is the role of the MPU and its memory interface circuitry to provide the address and control input signals and to read the output data at the appropriate times during the memory-read bus cycle. The block diagram in Fig. 9-4 shows a typical read-only memory interface. For a microprocessor to read a byte of data from the device, it must apply a binary address to inputs \( A_0 \) through \( A_{10} \) of the EPROM. This address gets decoded inside the device to select the storage location of the byte of data that is to be read. Remember that the microprocessor must switch \( \overline{CE} \) and \( \overline{OE} \) to logic 0 to enable the device and its outputs. Once done, the byte of data is made available at \( O_0 \) through \( O_7 \) and the microprocessor can read the data over its data bus.

Standard EPROM ICs

A large number of standard EPROM ICs are available today. Figure 9-5 lists the part numbers, bit densities, and byte capacities of nine popular devices. They range in size from the 16KB density (2K \( \times \) 8) 2716 device, to the 4MB (512K \( \times \) 8) 27C040 device. Higher-density devices, such as the 27C256 through 27C020, are now popular for system designs. In fact, many manufacturers have already discontinued some of the older devices, such as the 2716 and 2732. Let us now look at some of these EPROMs in more detail.

The 27C256 is an EPROM IC manufactured with the CMOS technology. Looking at Fig. 9-5, we find that it is a 256KB device, and its storage array is organized as 32K \( \times \) 8 bits. Figure 9-6 shows the pin layout of the 27C256. Here we see that it has

![Figure 9-4  Read-only memory interface.](image)
15 address inputs, labeled \(A_0\) through \(A_{14}\), eight data outputs, identified as \(O_0\) through \(O_7\), and two control signals \(CE\) and \(OE\).

From our earlier description of the read operation, it appears that after the inputs of the EPROM are set up, the output is available immediately; however, in practice this is not true. A short delay exists between address inputs and data outputs. This leads us to three important timing properties defined for the read cycle of an EPROM: access time (\(t_{ACC}\)), chip-enable time (\(t_{CE}\)), and chip-deselect time (\(t_{DF}\)). The values of these timing properties are provided in the read-cycle switching characteristics shown in Fig. 9-7(a) and identified in the switching waveforms shown in Fig. 9-7(b).

Access time tells us how long it takes to access data stored in an EPROM. Here we assume that both \(CE\) and \(OE\) are already at their active 0 levels, and then the address is applied to the inputs of the EPROM. In this case, the delay \(t_{ACC}\) occurs before the data stored at the addressed location are stable at the outputs. The microprocessor must wait at least this long before reading the data; otherwise, invalid results may be obtained. Figure 9-7(a) shows that the standard EPROMs are available with a variety of access time ratings. The maximum values of access time are given as 170 ns, 200 ns, and 250 ns. The speed of the device is selected to match that of the MPU. If the access time of the fastest standard device is too long for the MPU, wait state circuitry needs to be added to the interface. In this way, wait states can be inserted to slow down the memory read bus cycle.

Chip-enable time is similar to access time. In fact, for most EPROMs they are equal in value. They differ in how the device is set up initially. This time the address is applied and \(OE\) is switched to 0, then the read operation is initiated by making \(CE\) active. Therefore, \(t_{CE}\) represents the chip-enable-to-output delay instead of the address-to-output delay. Looking at Fig. 9-7(a), we see that the maximum values of \(t_{CE}\) are also 170 ns, 200 ns, and 250 ns.

Chip-deselect time is the opposite of access or chip-enable time. It represents the amount of time the device takes for the data outputs to return to the high-Z state after \(OE\) becomes inactive—that is, the recovery time of the outputs. Figure 9-7(a) shows that the maximum values for this timing property are 55 ns, 55 ns, and 60 ns.

In an erased EPROM, all storage cells hold logic 1. The device is put into the programming mode by switching on the \(V_{pp}\) power supply. Once in this mode, the address of the storage location to be programmed is applied to the address inputs, and the byte of
Figure 9–6  Pin layouts of standard EPROMs.
<table>
<thead>
<tr>
<th>Versions</th>
<th>27C256-120V05</th>
<th>27C256-135V05</th>
<th>27C256-159V05</th>
<th>27C256-1 P27C256-1 N27C256-1</th>
<th>27C256-2 P27C256-2 N27C256-2</th>
<th>27C256 P27C256 N27C256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(V_{CC} \pm 5%)</td>
<td>(V_{CC} \pm 10%)</td>
<td>(V_{CC} \pm 10%)</td>
<td>(V_{CC} \pm 10%)</td>
<td>(V_{CC} \pm 10%)</td>
<td>(V_{CC} \pm 10%)</td>
</tr>
<tr>
<td>Symbol</td>
<td>Parameter</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>(t_{ACC})</td>
<td>Address to output delay</td>
<td>120</td>
<td>135</td>
<td>150</td>
<td>170</td>
<td>200</td>
</tr>
<tr>
<td>(t_{CE})</td>
<td>CE to output delay</td>
<td>120</td>
<td>135</td>
<td>150</td>
<td>170</td>
<td>200</td>
</tr>
<tr>
<td>(t_{OE})</td>
<td>OE to output delay</td>
<td>60</td>
<td>85</td>
<td>70</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>(t_{OE}^{(2)})</td>
<td>OE high to output high-Z</td>
<td>30</td>
<td>35</td>
<td>45</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>(t_{OH}^{(2)})</td>
<td>Output hold from addresses, CE or OE change—whichever is first</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes:
1. A.C. characteristics tested at \(V_{IH} = 2.4\) V and \(V_{IL} = 0.45\) V.
2. Timing measurements made at \(V_{OL} = 0.8\) V and \(V_{OH} = 2.0\) V.
3. Guaranteed and sampled.
4. Package Prefixes: No Prefix = CERDIP; N = PLCC; P = Plastic DIP.

Figure 9–7 (a) EPROM device timing characteristics. (Reprinted by permission of Intel Corporation; Copyright Intel Corp. 1989) (b) EPROM switching waveforms. (Reprinted by permission of Intel Corporation; Copyright Intel Corp. 1989)
Figure 9-7 (continued)

data to be loaded into this location is supplied as inputs to the data leads. Note that the data outputs act as inputs when the EPROM is set up for programming mode of operation. Next the CE input is pulsed to load the data. Actually, a complex series of program and verify operations are performed to program each storage location in an EPROM. The two widely used programming sequences are the Quick-Pulse Programming Algorithm and the Intelligent Programming Algorithm. Flowcharts for these programming algorithms are given in Figs. 9-8(a) and (b), respectively.

Figure 9-9 presents another group of important electrical characteristics for the 27C256 EPROM. They are the device's dc electrical operating characteristics. CMOS EPROMs are designed to provide TTL-compatible input and output logic levels. Here we find the output logic level ratings are $V_{OH_{min}} = 3.5$ V and $V_{OL_{max}} = 0.45$ V. Also provided is the operating current rating of the device, identified as $I_{CC} = 30$ mA. This shows that if the device is operating at 5 V, it will consume 150 mW of power.

Figure 9-6 also shows the pin layouts for the 2716 through 27C512 EPROM devices. In this diagram, we find that both the 27C256 and 27C512 are available in a 28-pin package. A comparison of the pin configuration of the 27C512 with that of the 27C256 shows that the only differences between the two pinouts are that pin 1 on the 27C512 becomes the new address input $A_{13}$, and $V_{pp}$, which was at pin 1 on the 27C256, becomes a second function performed by pin 22 on the 27C512.

Expanding EPROM Word Length and Word Capacity

In many applications, the microcomputer system requirements for EPROM are greater than what is available in a single device. There are two basic reasons for expand-
ing EPROM capacity: first, the byte-wide length is not large enough; and second, the total storage capacity is not enough bytes. Both of these expansion needs can be satisfied by interconnecting a number of ICs.

For instance, the 8086 microprocessor has a 16-bit data bus. Therefore, its program memory subsystem needs to be implemented with two 27C256 EPROMs connected, as shown in Fig. 9–10(a). Notice that the individual address inputs, chip enable lines, and output enable lines on the two devices are connected in parallel. On the other hand, the eight data outputs of each device are used to supply eight lines of the MPU’s 16-bit data bus. This circuit configuration has a total storage capacity equal to 32K words or 512KB.

Figure 9–10(b) shows how two 27C256s can be interconnected to expand the number of bytes of storage. Here the individual address inputs, data outputs, and output enable lines of the two devices are connected in parallel. However, the CE inputs of the individual devices remain independent and can be supplied by different chip enable outputs.
identified as $CS_0$ and $CS_1$, of an address decoder circuit. In this way, only one of the two devices is enabled at one time. This configuration results in a total storage capacity of 64Kbytes or 512KB. When several EPROMs are used in an 8088-based microcomputer, they are connected in this way. To double the word capacity of the circuit in Fig. 9-10(a), this same connection must be made for each of the EPROMs.

Figure 9-8  (continued)
### Table 9.1 DC electrical characteristics of the 27C256

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Notes</th>
<th>Min</th>
<th>Typ(%)</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_{D}</td>
<td>Input load current</td>
<td></td>
<td>0.01</td>
<td>1.0</td>
<td>μA</td>
<td>V_{IN} = 0V to V_{CC}</td>
<td></td>
</tr>
<tr>
<td>I_{LO}</td>
<td>Output leakage current</td>
<td></td>
<td>± 10</td>
<td>μA</td>
<td>V_{OUT} = 0V to V_{CC}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{PP}</td>
<td>V_{PP} read current</td>
<td>5</td>
<td>200</td>
<td>μA</td>
<td>V_{PP} = V_{CC}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{ST}</td>
<td>V_{CC} current standby</td>
<td></td>
<td>8</td>
<td>1.0</td>
<td>mA</td>
<td>CE = V_{IH}</td>
<td></td>
</tr>
<tr>
<td>I_{PP}</td>
<td>V_{PP} read current</td>
<td>4</td>
<td>100</td>
<td>μA</td>
<td>CE = V_{CC}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{CC}</td>
<td>V_{CC} current active</td>
<td>5,8</td>
<td></td>
<td>30</td>
<td>mA</td>
<td>CE = V_{IL}</td>
<td></td>
</tr>
<tr>
<td>V_{L}</td>
<td>Input low voltage (±10% supply) (TTL)</td>
<td></td>
<td>−0.5</td>
<td>0.8</td>
<td>V</td>
<td>f = 5 MHz</td>
<td></td>
</tr>
<tr>
<td>V_{L}</td>
<td>Input low voltage (CMOS)</td>
<td></td>
<td>−0.2</td>
<td>0.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_{H}</td>
<td>Input high voltage (±10% supply) (TTL)</td>
<td></td>
<td>2.0</td>
<td>V_{CC} + 0.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_{H}</td>
<td>Input high voltage (CMOS)</td>
<td></td>
<td>0.7 V_{CC}</td>
<td>V_{CC} + 0.2</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_{OL}</td>
<td>Output low voltage</td>
<td></td>
<td></td>
<td>0.45</td>
<td>V</td>
<td>I_{OL} = 2.1 mA</td>
<td></td>
</tr>
<tr>
<td>V_{OH}</td>
<td>Output high voltage</td>
<td></td>
<td>3.5</td>
<td>V</td>
<td>I_{OH} = −2.5 mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{OS}</td>
<td>Output short circuit current</td>
<td></td>
<td>6</td>
<td>100</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_{PP}</td>
<td>V_{PP} read voltage</td>
<td>7</td>
<td>V_{CC} − 0.7</td>
<td>V_{CC}</td>
<td>V</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. Minimum D.C. input voltage is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns. Maximum D.C. voltage on output pins is V_{CC} + 0.5V which may overshoot to V_{CC} + 2V for periods less than 20 ns.
2. Operating temperature is for commercial product defined by this specification. Extended temperature options are available in EXPRESS and Military version.
3. Typical limits are at V_{CC} = 5V, T_{A} = +25°C.
4. CE is V_{CC} ± 0.2V. All other inputs can have any value within spec.
5. Maximum Active power usage is the sum I_{PP} + I_{CC}.
   The maximum current value is with outputs O_{0} to O_{7} unloaded.
6. Output shorted for no more than one second. No more than one output shorted at a time. I_{OS} is sampled but not 100% tested.
7. V_{PP} may be one diode voltage drop below V_{CC}. It may be connected directly to V_{CC}. Also, V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
8. V_{IL}, V_{IH} levels at TTL inputs.

---

**Figure 9-9** DC electrical characteristics of the 27C256. (Reprinted by permission of Intel Corporation; Copyright Intel Corp. 1989)

### 9.3 RANDOM ACCESS READ/WRITE MEMORIES

The memory section of a microcomputer system is normally formed from both read-only memories and random access read/write memories (RAM). Earlier we pointed out that the ROM is used to store permanent information such as the microcomputer’s hardware control program. RAM is similar to ROM in that its storage location can be accessed in a random order, but it is different from ROM in two important ways. First, data stored in
Figure 9-10  (a) Expanding word length.  (b) Expanding word capacity.
RAM is not permanent in nature—that is, it can be altered. RAM can be used to store data by writing to it, and later the data can be read back for additional processing. Because of its read and write features, RAM finds wide use where data and programs need to be placed in memory only temporarily. For this reason, it is normally used to store data and application programs for execution. The second difference is that RAM is volatile—that is, if power is removed from RAM, the stored data are lost.

Static and Dynamic RAMs

Two types of RAMs are in wide use today: the static RAM (SRAM) and dynamic RAM (DRAM). For a static RAM, data, once entered, remain valid as long as the power supply is not turned off. On the other hand, to retain data in a DRAM, it is not sufficient just to maintain the power supply. For this type of device, we must both keep the power supply turned on and periodically restore the data in each storage location. This added requirement is necessary because the storage elements in a DRAM are capacitive nodes. If the storage nodes are not recharged within a specific interval of time, data are lost. This recharging process is known as refreshing the DRAM.

Block Diagram of a Static RAM

Figure 9–11 shows a block diagram of a typical static RAM IC. By comparing this diagram with the one shown for a ROM in Fig. 9–3, we see that they are similar in many ways. For example, they both have address lines, data lines, and control lines. These signal buses perform similar functions when the RAM is operated. Because of the RAMs read/write capability, data lines, however, act as both inputs and outputs. For this reason, they are identified as a bidirectional bus.

A variety of static RAM ICs are currently available. They differ both in density and organization. The most commonly used densities in system designs are the 64KB and 256KB devices. The structure of the data bus determines the organization of the SRAMs storage array. Figure 9–11 shows an 8-bit data bus. This type of organization is known as a byte-wide SRAM. Devices are also manufactured with by 1 and by 4 data I/O organizations. The 64KB density results in three standard device organizations: 64K X 1, 16K X 4, and 8K X 8.

The address bus on the SRAM in Fig. 9–11 consists of the lines labeled A₀ through A₁₂. This 13-bit address is what is needed to select between the 8K individual storage locations in an 8K X 8-bit SRAM IC. The 16K X 4 and 64K X 1 devices require a 14-bit and 16-bit address, respectively.

![Block diagram of a static RAM.](image-url)
To either read from or write to SRAM, the device must first be chip enabled. Just like for a ROM, this is done by switching the CE input of the SRAM to logic 0. Earlier we indicated that data lines I/O₀ through I/O₇ in Fig. 9-11 are bidirectional. This means that they act as inputs when writing data into the SRAM or as outputs when reading data from the SRAM. The setting of a new control signal, the write enable (WE) input, determines how the data lines operate. During all write operations to a storage location within the SRAM, the appropriate WE inputs must be switched to the 0 logic level. This configures the data lines as inputs. On the other hand, if data are to be read from a storage location, WE is left at the 1 logic level. When reading data from the SRAM, output enable (OE) must be active. Applying the active memory signal, logic 0, at this input, enables the device's three-state outputs.

A Static RAM System

Three-state data bus lines of SRAM devices allow for the parallel connection needed to expand data memory using multiple devices. For example, Fig. 9-12 shows how four 8K x 8-bit SRAMS are interconnected to form a 16K x 16-bit memory system. In this circuit, the separate CE inputs of the SRAM ICs in bank 0 are wired together and connected to a common chip-select input CS₀. The same type of connection is used for the SRAMS in bank 1 using chip-select input CS₁. These inputs are activated by the chip-select output of the address decoder circuit and must be logic 0 to select a bank of SRAMS for operation. The OE inputs of the individual SRAMS are connected in parallel. The combined output-enable input that results is driven by the MEMR output of the memory interface circuit and enables the outputs of all SRAMS during all memory-read bus cycles. Similarly, the write enables of all SRAMS are supplied from MEMW to write to the selected bank. Note that the memory system allows only word writes and reads.

Standard Static RAM ICs

Figure 9-13 lists a number of standard static RAM ICs. Here we find their part numbers, densities, and organizations. For example, the 4361, 4363, and 4364 are all 64KB density devices; however, they are each organized differently. The 4361 is a 64K x 1-bit device, the 4363 is a 16K x 4-bit device, and the 4364 is an 8K x 8-bit device.

The pin layouts of the 4364 and 43256A ICs are given in Figs. 9-14(a) and (b), respectively. Looking at the 4364 we see that it is almost identical to the block diagram shown in Fig. 9-11. The one difference is that it has two chip-enable lines instead of one. They are labeled CE₁ and CE₂. Note that logic 0 activates one, and logic 1 activates the other. Both of these chip-enable inputs must be at their active logic levels to enable the device for operation.

EXAMPLE 9.2

How does the 43256A SRAM differ from the block diagram in Fig. 9-11?

Solution

It has two additional address inputs, A₁₃ and A₁₄, and the chip-enable input is labeled CS instead of CE.
Figure 9-12  16K x 16-bit SRAM circuit.
As Fig. 9–15 shows, the 4364 is available in four speeds. For example, the minimum read cycle and write cycle time for the 4364-10 is 100 ns. Figure 9–16 is a list of the 4364’s dc electrical characteristics. Note (1) shows that the 100 ns device draws a maximum of 45 mA when operating at maximum frequency (minimum cycle time).

<table>
<thead>
<tr>
<th>SRAM</th>
<th>Density (bits)</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>4361</td>
<td>64K</td>
<td>64K x 1</td>
</tr>
<tr>
<td>4363</td>
<td>64K</td>
<td>16K x 4</td>
</tr>
<tr>
<td>4364</td>
<td>64K</td>
<td>8K x 8</td>
</tr>
<tr>
<td>43254</td>
<td>256K</td>
<td>64K x 4</td>
</tr>
<tr>
<td>43256A</td>
<td>256K</td>
<td>32K x 8</td>
</tr>
<tr>
<td>431000A</td>
<td>1M</td>
<td>128K x 8</td>
</tr>
</tbody>
</table>

**Figure 9–13** Standard SRAM devices.

![Diagram of pin layout](image)

Figure 9–14 (a) 4364 pin layout. (b) 43256A pin layout.
SRAM Read and Write Cycle Operation

Figure 9-17 illustrates the waveforms for a typical write cycle. Let us trace the events that take place during the write cycle. Here we see that all critical timing is referenced to the point at which the address becomes valid. Note that the minimum duration of the write cycle is identified as $t_{WC}$. This is the 100-ns write cycle time of the 4364-10. The address must remain stable for this complete interval of time.

Next, $CE_1$ and $CE_2$ become active and must remain active until the end of the write cycle. The duration of these pulses are identified as $CE_1$ to end of write time ($t_{CWE1}$) and $CE_2$ to end of write time ($t_{CWE2}$). As the waveforms show, we are assuming here that they begin at any time after the occurrence of the address but before the leading edge of WE. The minimum value for both of these times is 80 ns. On the other hand, WE is shown not to occur until the interval $t_{AS}$ elapses. This is the address-setup time and represents the minimum amount of time the address inputs must be stable before WE can be switched to logic 0. For the 4364, however, this parameter is equal to 0 ns. The width of the write enable pulse is identified as $t_{WP}$, and its minimum value equals 60 ns.

Data applied to the D_in data inputs are written into the device synchronous with the trailing edge of WE. Note that the data must be valid for an interval equal to $t_{DW}$ before this edge. This interval, called data valid to end of write, has a minimum value of 40 ns for the 4364-10. Moreover, it is shown to remain valid for an interval of time equal to $t_{DH}$ after this edge. This data-hold time, however, just like address-setup time, equals 0 ns for the 4364. Finally, a short recovery time takes place after WE returns to logic 1 before the write cycle is complete. This interval is identified as $t_{WR}$ in the waveforms, and its minimum value equals 5 ns.

The read cycle of a static RAM, such as the 4364, is similar to that of a ROM. Figure 9-18 gives waveforms of a read operation.

Standard Dynamic RAM ICs

Dynamic RAMs are available in higher densities than static RAMs. Currently, the most widely used DRAMs are the 64K-bit, 256K-bit, 1M-bit, and 4M-bit devices. Figure 9-19 lists a number of popular DRAM ICs. Here we find the 2164B, organized as 64K $\times$ 1 bit; the 21256, organized as 256K $\times$ 1 bit; the 21464, organized as 64K $\times$ 4 bits; the 421000, organized as 1M $\times$ 1 bit; and the 424256, organized as 256K $\times$ 4 bits. Pin layouts for the 2164B, 21256, and 421000 are shown in Figs. 9-20(a), (b), and (c), respectively.

Some other benefits of using DRAMs over SRAMs are that they cost less, consume less power, and their 16- and 18-pin packages take up less space. For these reasons, DRAMs are normally used in applications that require a large amount of memory. For example, most systems that support at least 1Mbyte of data memory are designed using DRAMs.

<table>
<thead>
<tr>
<th>Part number</th>
<th>Read/write cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4364-10</td>
<td>100 ns</td>
</tr>
<tr>
<td>4364-12</td>
<td>120 ns</td>
</tr>
<tr>
<td>4364-15</td>
<td>150 ns</td>
</tr>
<tr>
<td>4364-20</td>
<td>200 ns</td>
</tr>
<tr>
<td>Parameter</td>
<td>Symbol</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Input leakage current</td>
<td>$I_{UL}$</td>
</tr>
<tr>
<td>Output leakage current</td>
<td>$I_{LO}$</td>
</tr>
<tr>
<td>Operating supply current</td>
<td>$I_{CCA1}$</td>
</tr>
<tr>
<td></td>
<td>$I_{CCA2}$</td>
</tr>
<tr>
<td></td>
<td>$I_{CCA3}$</td>
</tr>
<tr>
<td>Standby supply current</td>
<td>$I_{SB}$</td>
</tr>
<tr>
<td></td>
<td>$I_{SB1}$</td>
</tr>
<tr>
<td></td>
<td>$I_{SB2}$</td>
</tr>
<tr>
<td>Output voltage, low</td>
<td>$V_{OL}$</td>
</tr>
<tr>
<td>Output voltage, high</td>
<td>$V_{OH}$</td>
</tr>
</tbody>
</table>

Notes:

(1) μPD4364-10/11L: 45 mA max
    μPD4364-12/12L/12LL: 40 mA max
    μPD4364-15/15L/15LL: 40 mA max
    μPD4364-20/20L/20LL: 35 mA max

(2) μPD4364-xx: 5 mA max
    μPD4364-xxL: 3 mA max
    μPD4364-xxLL: 3 mA max
(3) μPD4364-xx: 2 mA max
    μPD4364-xxL: 100 μA max
    μPD4364-xxLL: 50 μA max

Figure 9–16  DC electrical characteristics of the 4364.
1. A write occurs during the overlap of a low CE₁ and a high CE₂ and a low WE.
2. CE₁ or WE (or CE₂) must be high [low] during any address transaction.
3. If CE is high the I/O pins remain in a high-impedance state.

The 2164B is one of the older NMOS DRAM devices. Figure 9-21 presents a block diagram of the device. Looking at the block diagram, we find that it has eight address inputs, A₀ through A₇, a data input and a data output marked D and Q, respectively, and three control inputs, row-address strobe (RAS), column-address strobe (CAS), and read/write (W).

The storage array within the 2164B is capable of storing 65,536 (64K) individual bits of data. To address this many storage locations, we need a 16-bit address; however,
this device's package has just 16 pins. For this reason, the 16-bit address is divided into two separate parts: an 8-bit row address and an 8-bit column address. These two parts of the address are time-multiplexed into the device over a single set of address lines, A₀ through A₇. First the row address is applied to A₀ through A₇. Then RAS is pulsed to logic 0 to latch it into the device. Next, the column address is applied and strobes CAS to logic 0. This 16-bit address selects which one of the 64K storage locations is to be accessed.

Data are either written into or read from the addressed storage location in DRAMs. Write data are applied to the D input and read data are output at Q. The logic levels of control signals W, RAS, and CAS tell the DRAM whether a read or write data transfer is taking place and control the three-state outputs. For example, during a write operation, the logic level at D is latched into the addressed storage location at the falling edge of either CAS or W. If W is switched to logic 0 by an active MWTC signal before CAS, an
Figure 9-20  (a) 2164B pin layout. (b) 21256 pin layout. (c) 421000 pin layout.
early write cycle is performed. During this type of write cycle, the outputs are maintained in the high-Z state throughout the complete bus cycle. The fact that the output is put in the high-Z state during the write operation allows the D input and Q output of the DRAM to be tied together. The Q output is also in the high-Z state whenever CAS is logic 1. This is the connection and mode of operation normally used when attaching DRAMs to the bidirectional data bus of a microprocessor. Figure 9-22 shows how 16 2164B devices are connected to make up a 64K \times 16\text{-}bit DRAM array.

The 2164B also has the ability to perform what are called *page-mode* accesses. If RAS is left at logic 0 after the row address is latched inside the device, the address is maintained within the device. Then, by simply supplying successive column addresses, data cells along the selected row are accessed. This permits faster access of memory by eliminating the time needed to set up and strobe additional row addresses.

Earlier we pointed out that the key difference between the DRAM and SRAM is that the storage cells in the DRAM need to be periodically refreshed; otherwise, they lose their data. To maintain the integrity of the data in a DRAM, each of the rows of the storage array must typically be refreshed periodically, such as every 2 ms. All the storage cells in an array are refreshed by simply cycling through the row addresses. As long as CAS is held at logic 1 during the refresh cycle, no data are output.

External circuitry is required to perform the address multiplexing, RAS/CAS generation, and refresh operations for a DRAM subsystem. DRAM-refresh controller ICs are available to permit easy implementation of these functions.

**Battery Backup for the RAM Subsystem**

Even though RAM ICs are volatile, in some equipment it is necessary to make all or part of the RAM memory subsystem nonvolatile (e.g., an electronic cash register). In this application, a power failure could result in the loss of irreplaceable information about the operation of the business.

To satisfy the nonvolatile requirement, additional circuitry can be included in the RAM subsystem. These circuits must sense the occurrence of a power failure and automatically switch the memory subsystem over to a backup battery. An orderly transition must take place from system power to battery power. Therefore, when a loss of power is detected, the power-fail circuit must permit the completion of any read or write cycle that
Figure 9-22 64K × 16-bit DRAM circuit.
is in progress and then lock the memory against the occurrence of additional read/write operations. The memory subsystem remains in this state until power is restored. In this way, the RAM subsystem can be made at least temporarily nonvolatile.

9.4 PARITY, THE PARITY BIT, AND PARITY-CHECKER/GENERATOR CIRCUIT

In microcomputer systems, the data exchanges that take place between the MPU and the memory must be done without error. However, problems such as noise, transient signals, or even bad memory bits can produce errors in the transfer of data and instructions. For instance, the storage location for 1 bit in a large DRAM array may be bad and stuck at the 0 logic level. This will not present a problem if the logic level of the data written to the storage location is 0, but if it is 1, the value will always be read as 0. To improve the reliability of information transfer between the MPU and memory, a parity bit can be added to each byte of data. To implement data transfers with parity, a parity-checker/generator circuit is required.

Figure 9-23 shows a parity-checker/generator circuit added to the memory interface of a microcomputer system. Note that the data passed between the MPU and memory subsystem is applied in parallel to the parity-checker/generator circuit. Assuming that the microprocessor has an 8-bit data bus, data words read from or written to memory by the

![Figure 9-23 Data-storage memory interface with parity-checker generator.](image)
MPU over the data bus are still byte-wide, but the data stored in memory is 9 bits long. The data in memory consists of 8 bits of data and 1 parity bit. Assuming that the memory array is constructed with 256K x 1-bit DRAMs, then the memory array for a memory subsystem with parity would have nine DRAM ICs instead of eight. The extra DRAM is needed for storage of the parity bit for each byte of data stored in the other eight DRAM devices.

The parity-checker/generator circuit can be set up to produce either even parity or odd parity. The 9-bit word of data stored in memory has even parity if it contains an even number of bits that are at the 1 logic level and odd parity if the number of bits at logic 1 is odd.

Let us assume that the circuit in Fig. 9-23 is used to generate and check for even parity. If the byte of data written to memory over the MPU’s data bus is FFH, the binary data is 11111111. This byte has 8 bits at logic 1—that is, it already has even parity. Therefore, the parity-checker/generator circuit, which operates in the parity generate mode, outputs logic 0 on the parity bit line (PB), and the 9 bits of data stored in memory is 0111111111. On the other hand, if the byte written to memory is 7FH, the binary word are 01111111. Since only 7 bits are at logic 1, parity is odd. In this case, the parity-checker/generator circuit makes the parity bit logic 1, and the 9 bits of data saved in memory is 1011111111. Notice that the data held in memory has even parity. In this way, we see that during all data memory write cycles, the parity-checker/generator circuit simply checks the data that are to be stored in memory and generates a parity bit. The parity bit is attached to the original 8 bits of data to make it 9 bits. The 9 bits of data stored in memory have even parity.

The parity-checker/generator works differently when data are read from memory. Now the circuit must perform its parity-check function. Note that the 8 bits of data from the addressed storage location in memory are sent directly to the MPU. However, at the same time, this byte and the parity bit are applied to the inputs of the parity-checker/generator circuit. This circuit checks to determine whether there is an even or odd number of logic 1s in the word with parity. Again we will assume that the circuit is set up to check for even parity. If the 9 bits of data read from memory are found to have an even number of bits at the 1 logic level, parity is correct. The parity-checker/generator signals this fact to the MPU by making the parity error (PE) output inactive logic 1. This signal is normally sent to the MPU to identify whether or not a memory parity error has occurred. If an odd number of bits are found to be logic 1, a parity error has been detected and PE is set to 0 to tell the MPU of the error condition. Once alerted to the error, the MPU can do any one of a number of things under software control to recover. For instance, it could simply repeat the memory-read cycle to see if it takes place correctly the next time.

The 74AS280 device implements a parity-checker/generator function similar to that just described. Figure 9-24(a) shows a block diagram of the device. Note that it has nine data-input lines, which are labeled A through H. In the memory interface, lines A through H are attached to data bus lines D0 through D7, respectively, and during a read operation the parity bit output of the memory array, DpB, is applied to the I input.

The function table in Fig. 9-24(b) describes the operation of the 74AS280. It shows how the $\Sigma_{EVEN}$ and $\Sigma_{ODD}$ outputs respond to an even or odd number of data inputs at logic 1. Note that if there are 0, 2, 4, or 8 inputs at logic 1, the $\Sigma_{EVEN}$ output switches to logic 1 and $\Sigma_{ODD}$ to logic 0. This output response signals the even parity condition.
Figure 9–24  (a) Block diagram of the 74AS280. (Texas Instruments Incorporated) (b) Function table. (Texas Instruments Incorporated) (c) Even-parity checker/generator connection.
In practical applications, the $\Sigma_{EVEN}$ and $\Sigma_{ODD}$ outputs are used to produce the parity bit and parity error signal lines. Figure 9-24(c) is an even parity-checker/generator configuration. Note that $\Sigma_{ODD}$ is used as the parity bit ($D_{PB}$) output that gets applied to the data input of the parity bit DRAM in the memory array. During a write operation MEMR is 0, which makes the I input 0, and therefore the parity of the byte depends only on data bits $D_0$ through $D_7$, which are applied to the A through H inputs of the 74AS280. As long as the input at A through H has an even number of bits at logic 1 during a memory write cycle, $\Sigma_{ODD}$, which is $D_{PB}$, is at logic 0 and the 9 bits of data written to memory retain an even number of bits that are 1, or even parity. On the other hand, if the incoming byte at A through H has an odd number of bits that are logic 1, $\Sigma_{ODD}$ switches to logic 1. The logic 1 at $D_{PB}$ along with the odd number of 1s in the original byte again give the 9 bits of data stored in memory an even parity.

Let us next look at what happens in the parity-checker/generator circuit during a memory-read cycle for the data-storage memory subsystem. When the MPU is reading a byte of data from memory, the 74AS280 performs the parity-check operation. In response to the MPU's read request, the memory array outputs 9 bits of data. They are applied to inputs A through I of the parity-checker/generator circuit. The 74AS280 checks the parity and adjusts the logic levels of $\Sigma_{EVEN}$ and $\Sigma_{ODD}$ to represent this parity. If parity is even as expected, $\Sigma_{EVEN}$, which represents the parity error (PE) signal, is at logic 1. This tells the MPU that a valid data transfer is taking place. However, if the data at A through H has an odd number of bits at logic 1, $\Sigma_{ODD}$ switches to logic 0 and informs the MPU that a parity error has occurred.

In a 16-bit microcomputer system, such as that built with the 8086 MPU, there are normally two 8-bit banks of DRAM ICs in the data-storage memory array. In this case, a parity bit DRAM is added to each bank. Therefore, parity is implemented separately for each of the two bytes of a data word stored in memory. This is important because the 8086 can read either bytes or words of data from memory. For this reason, two parity-checker/generator circuits are also required, one for the upper eight lines of the data bus and one for the lower eight lines. Gating them together combines the parity error outputs of the two circuits and the resulting parity error signal is supplied to the MPU. In this way, the MPU is notified of a parity error if it occurs in an even-addressed byte data transfer, odd-addressed byte data transfer, or in either or both bytes of a 16-bit data transfer.

9.5 FLASH MEMORY

Another memory technology important to the study of microcomputer systems is what is known as FLASH memory. FLASH memory devices are similar to EPROMs in many ways, but are different in several very important ways. In fact, FLASH memories act just like EPROMs: they are nonvolatile, are read just like an EPROM, and program with an EPROM-like algorithm.

The key difference between a FLASH memory and an EPROM is that its memory cells are erased electrically, instead of by exposure to ultraviolet light. That is, the storage array of a FLASH memory can be both electrically erased and reprogrammed with new data. Unlike RAMs, they are not byte erasable and writeable. When an erase operation is performed on a FLASH memory, either the complete memory array or a large block of storage locations, not just one byte, is erased. Moreover, the erase process is complex and

Sec. 9.5     FLASH Memory
can take as long as several seconds. This erase operation can be followed by a write operation—a programming cycle—that loads new data into the storage location. This write operation also takes a long time when compared to the write cycle times of a RAM.

Even though FLASH memories are writeable, like EPROMs they find their widest use in microcomputer systems for storage of firmware. However, their limited erase/rewrite capability enables their use in applications where data must be rewritten, though not frequently. Some examples: implementation of a nonvolatile writeable lookup table, in-system programming for code updates, and solid state drives. An example of the use of FLASH memory as a lookup table is the storage of a directory of phone numbers in a cellular phone.

Block Diagram of a FLASH Memory

Earlier we pointed out that FLASH memories operate in a way very similar to an EPROM. Figure 9–25 shows a block diagram of a typical FLASH memory device. Let us compare this block diagram to that of the ROM in Fig. 9–3. Address lines A₀ through A₁₇, chip enable (CE), and output enable (OE) serve the exact same function for both devices. That is, the address picks the storage location that is to be accessed, CE enables the device for operation, and OE enables the data to the outputs during read cycles.

We also see that they differ in two ways. First, the data bus is identified as bidirectional, because the FLASH memory can be used in an application where it is written into as well as read from. Second, another control input, write enable (WE), is provided. This signal must be at its active 0 logic level during all write operations to the device. In fact, this block diagram is exactly the same as that given for SRAM in Fig. 9–11.

Bulk-Erase, Boot Block, and FlashFile FLASH Memories

FLASH memory devices are available with several different memory array architectures. These architectures relate to how the device is organized for the purpose of erasing. Earlier we pointed out that when an erase operation is performed to a FLASH memory device, either all or a large block of memory storage locations are erased. The three standard FLASH memory array architectures, bulk-erase, boot block, and FlashFile, are shown in Fig. 9–26. In a bulk-erase device, the complete storage array is arranged as a single block. Whenever an erase operation is performed, the contents of all storage loca-
tions are cleared. This is the architecture used in the design of the earliest FLASH memory devices.

More modern FLASH memory devices employ either the boot block or FlashFile architecture for their memory array. They add granularity to the programming process. Now the complete memory array does not have to be erased. Instead, each of the independent blocks of storage locations erases separately. Note that the blocks on a boot block device are asymmetrical in size. There is one small block known as the boot block. This block is intended for storage of the boot code for the system. Two small blocks that are called parameter blocks follow it. Their intended use is for storage of certain system parameters, for instance, a system configuration table or lookup time. Finally, there are a number of much larger blocks of memory identified as main blocks, where the firmware code is stored.

Boot block devices are intended for use in a variety of applications that require smaller memory capacity and benefit from the asymmetrical blocking. One such application is known as in-system programming. In this type of application, the boot code used to start up the microcomputer is held in the boot block part of the FLASH memory. When the system is powered on, a memory-loading program is copied from the boot area of FLASH into RAM. Then, program execution is transferred to this program; the firmware is then loaded into the FLASH memory is downloaded from a communication line or external storage device such as a drive; the firmware is written into the main blocks of the FLASH memory devices; and finally the program is executed out of FLASH. In this way, we see that the FLASH memory devices are not loaded with the microcomputer's program in advance; instead, they are programmed while in the system.
FlashFile architecture FLASH memory devices differ from boot block devices in that the memory array is organized into equal-sized blocks. For this reason, it is said to be symmetrically blocked. This type of organization is primarily used in the design of high-density devices. High-density flash devices are used in applications that require a large amount of code or data to be stored (e.g., a FLASH memory drive).

Standard Bulk-Erase FLASH Memories

Bulk-erase FLASH memories are the oldest type of FLASH devices and are available in densities similar to those of EPROMs. Figure 9–27 lists the part number, bit density, and storage capacity of some of the popular devices. Note that the part numbers of FLASH devices are similar to those used for the EPROMs described earlier. The differences are that the 7 in the EPROM part number is replaced by an 8, representing FLASH, and instead of a C, which is used to identify that the circuitry of the EPROM is made with a CMOS process, an F identifies FLASH technology. Remember the 2MB EPROM was labeled 27C020; therefore, the 2MB FLASH memory is labeled as 28F020. This device is organized as 256K byte-wide storage locations.

Since FLASH memories are electrically erased, they do not need to be manufactured in a windowed package. For this reason, and the trend toward the use of surface-mount packaging, the most popular package for housing FLASH memory ICs is the plastic leaded chip carrier, or PLCC as it is commonly known. Figure 9–28(a) shows the PLCC pin layout of the 28F020. All of the devices listed in Fig. 9–27 are manufactured in this same-size package and with compatible pin layouts.

Looking at the signals identified in the pin layout, we find that the device is exactly the same as the block diagram in Fig. 9–25. To select between its 256K byte-wide storage locations, it has 18 address inputs, A0 through A17, and to support byte-wide data-read and -write transfers, it has an 8-bit data bus, DQ0 through DQ7. Finally, to enable the chip and its outputs and distinguish between read- and write-data transfers, it has control lines CE, OE, and WE, respectively. As Fig. 9–28(b) shows, the device is available with read access times ranging from 70 ns for the 28F020-70 to 150 ns for the 28F020-150.

The power supply voltage and current requirements depend on whether the FLASH memory is performing a read, erase, or write operation. During read mode of operation, the 28F020 is powered by 5V ±10% between the Vcc and Vss pins, and it draws a maximum current of 30 mA. On the other hand, when either an erase or write cycle is taking place, 12V ±5% must also be applied to the Vpp power supply input.

The 28F256, 28F512, 28F010, and 28F020 employ a bulk-erase storage array. For this reason, when an erase operation is performed to the device, all bytes in the storage
array are restored to FF₁₆, which represents the erased state. The method employed to erase the 28F020 FLASH memory IC is known as the quick-erase algorithm. A flowchart that outlines the sequence of events that must take place to erase a 28F020 is given in Fig. 9-29. This programming sequence can be performed either with a FLASH memory-programming instrument or by the software of the microprocessor to which the FLASH device is attached. Let us next look more closely at how a 28F020 is erased.

To change the contents of a memory array—that is, either erase the storage array or write bytes of data into the array—commands must be written to the FLASH memory device. Unlike an EPROM, a FLASH memory has an internal command register. Figure 9-30 lists the commands that can be issued to the 28F020. Note that they include a read (read memory), set up and erase (set up erase/erase), and erase verification (erase verify) commands. These three are used as part of the quick-erase algorithm process. The command register can be accessed only when +12V is applied to the Vpp pin of the 28F020.

Figure 9-29 also includes a table of the bus operation and command activity that takes place during an erase operation of the 28F020. From the bus operation and command columns, we see that as part of the erase process, the microprocessor (or programming instrument) must issue a set up erase/erase command, followed by an erase verify command, and then a read command to the FLASH device. It does this by executing a
Figure 9-29 Quick-erase algorithm of the 28F020. (Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995)
FLASH memory programming control program that causes the write of the command to the FLASH memory device at the appropriate time. Actually, all storage locations in the memory array must always be programmed with 00₁₆ before initiating the erase process. Figure 9–29 shows that two consecutive set up erase/erase commands are used in the quick-erase sequence. The first one prepares the 28F020 to be erased, and the second initiates the erase process. Figure 9–30 shows that this sequence is identified as two write cycles. During both of these bus cycles, a value of data equal to 20₁₆ is written to any address in the address range of the FLASH device being erased. Once these commands have been issued, a state-machine within the device automatically initiates and directs the erase process through completion.

The next step in the quick-erase process is to determine whether or not the device has erased completely. This is done with the erase verify command. Figure 9–30 shows that this operation requires a write cycle followed by a read cycle. During the write cycle, the data bus carries the erase verify command, A₀₁₆, and the address bus carries the address of the storage location that is to be tested, EA. The read cycle that follows is used to transfer the data from the storage location corresponding to EA to the MPU. This data is identified as EVD in Fig. 9–30. The flowchart shows that the MPU must verify that the value of data read out of FLASH is FF₁₆. This erase verify step is repeated for every storage location in the 28F020. If any storage location does not verify erase by reading back FF₁₆, the complete erase process is immediately repeated.

After complete erasure has been verified, the software must issue a read command to the device. From Fig. 9–30, we find that it requires a single write bus cycle and is accompanied by any address that corresponds to the FLASH device being erased and a command data value of 00₁₆. Issuing this command puts the device into the read mode and reads it for read operation. Figure 9–31 outlines the quick-pulse programming algorithm of the 28F020. This process is similar to that just described for erasing devices; however, it uses the set up and program (set up program/program), program verification (program verify), and read (read memory) commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Bus Cycles Req'd</th>
<th>First Bus Cycle</th>
<th>Second Bus Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Operation</td>
<td>Address</td>
</tr>
<tr>
<td>Read memory</td>
<td>1</td>
<td>Write</td>
<td>X</td>
</tr>
<tr>
<td>Read intelligent identifier codes</td>
<td>3</td>
<td>Write</td>
<td>X</td>
</tr>
<tr>
<td>Setup erase/erase</td>
<td>2</td>
<td>Write</td>
<td>X</td>
</tr>
<tr>
<td>Erase verify</td>
<td>2</td>
<td>Write</td>
<td>EA</td>
</tr>
<tr>
<td>Setup program/program</td>
<td>2</td>
<td>Write</td>
<td>X</td>
</tr>
<tr>
<td>Program verify</td>
<td>2</td>
<td>Write</td>
<td>X</td>
</tr>
<tr>
<td>Reset</td>
<td>2</td>
<td>Write</td>
<td>X</td>
</tr>
</tbody>
</table>

Figure 9–30 28F020 command definitions. (Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995)

Sec. 9.5 FLASH Memory
Standard Boot Block FLASH Memories

Earlier we pointed out that boot block FLASH memories are designed for use in embedded microprocessor application. These newer devices are available in higher densities than bulk-erase devices. Fig. 9-32 shows the pin layouts for three compatible standard densities: the 2MB, 4MB, and 8MB devices. Notice that the corresponding devices are identified with the part numbers 28F002, 28F004, and 28F008, respectively. These devices have different densities but have a common set of operating features and capabilities. The pinout information given is for a 40-pin thin small outline package (TSOP).
These devices offer a number of new architectural features when compared to the bulk-erase devices just described. One of the most important of these new features is what is known as SmartVoltage. This capability enables the device to be programmed with either a 5-V or 12-V value of \( V_{pp} \). In fact, the device can be installed into a circuit using either value of \( V_{pp} \). This is because the device has the ability to automatically detect and adjust its programming operation to the value of the programming supply voltage in use.

These devices are available with either of two read voltage, \( V_{cc} \) supply ratings: Smart 5, which operates off a 5V \( V_{cc} \), and Smart 3, which operates off a 3V \( V_{cc} \).

A second important difference is that devices are available at each of these three densities that can be organized with either an 8-bit or 16-bit bus. A block diagram of such a device is shown in Fig. 9–33(a). This device is identified as a 28F004/28F400. The 28F004 device is available in the 40-pin TSOP package and only operates in the 8-bit data bus mode. The 28F400 device has 16 data lines, \( D_0 \) through \( D_{15} \), and can be configured to operate with either an 8-bit or 16-bit data bus. This is done with the BYTE input. Logic 0 at BYTE selects byte-wide mode of operation and logic 1 chooses word-wide operation. To permit the extra data lines, the 28F400 is housed in a 56-lead TSOP.

Remember that the storage array of a boot block device is arranged as multiple asymmetrically sized, independently erasable blocks. In fact, the 28F004 has one 16Kbyte boot block, two 8Kbyte parameter blocks, three 128Kbyte main blocks, and a fourth main block that is only 96Kbytes. Two different organizations of these blocks are available, as shown in Fig. 9–33(b). The configuration on the left is known as the top boot (T), and that on the right is known as the bottom boot (B). Note that they differ in how the blocks are assigned to the address space. That is, the T version has the 16Kbyte boot block at the top of the address space (highest address), followed by the parameter blocks, and then the main blocks. The address space of the B version is a mirror image; therefore, the 16Kbyte boot block starts at the bottom of the address space (lowest address).

Another new feature introduced with the boot block architecture is that of a hardware-lockable block. In the 28F004/28F400, the 16Kbyte boot block section can be
locked. If external hardware applies logic 0 to the write protect (WP) input, the boot block is locked. Any attempt to erase or program this block when it is locked results in an error condition. Therefore, we say that the boot block is write protected. In an in-system programming application, the boot block part of the storage array typically would contain the part of the microcomputer program (boot program) that is used to load the system software into FLASH memory. For this reason, it would be locked and should remain that way.

Looking at Fig. 9–33(a), we find one more new input on the 28F004/28F400, the reset/deep power-down (RP) input. This input must be at logic 1 to enable normal read,
erase, and program operations. During read operations, the device can draw as much as 60 mA of current. If the device is not in use, it can be put into the deep power-down mode to conserve power. To do this, external circuitry must switch RP to logic 0. In this mode, it draws just 0.2 μA.

The last difference we will describe is that the 28F004/28F400 is equipped with what is known as automatic erase and write. No longer do we need to implement the complex quick-erase and quick-pulse programming algorithms in software as done for the 28F020. Instead, the 28F004/28F400 uses a command user interface (CUI), status register, and write-state machine to initiate an internally implemented and highly automated method of erasing and programming the blocks of the storage array.

Let us now look briefly at how an erase operation is performed. The command bus definitions of the 28F004/28F400 are shown in Fig. 9–34(a), the bit definitions of its status register are given in Fig. 9–34(b), and its erase cycle flowchart in Fig. 9–34(c). Here we see that all that needs to be done to initiate an erase operation is to write to the device a command bus definition that includes an erase setup command and an erase confirm command. These commands contain an address that identifies the block to be erased. In response to these commands, the write state machine drives a sequence that automatically programs all of the bits in this block to logic 0, verifies that they have been programmed, erases all of the bits in the block, and then verifies that each bit in the block has been erased. While it is performing this process, the write state machine status bit (WSMS) of the status register is reset to 0 to say that the device is busy. The microcomputer’s software can simply poll this bit to see if it is still busy. When WSMS is read as logic 1 (ready), the erase operation is complete and all the bits in the erased block are at the 1 logic level. In this way, we see that the new programming software only has to initiate the automatic erase process and then poll the status register to determine when the erase operation is finished.

Standard FLASHFile FLASH Memories

The highest-density FLASH memories available today are those designed with the FLASHFile architecture. As pointed out earlier, they use a symmetrically sized, independently erasable organization for blocking of their storage array. Two popular devices, the 8MB 28F008S5 and the 16MB 28F016SA/SV, are intended for use in large-code storage applications and to implement solid-state mass-storage devices such as the FLASH card and FLASH drive.

A block diagram of the 28F016SA/SV FLASHFile memory device is shown in Fig. 9–35(a) and its pin layout for a shrink small outline package (SSOP) is given in Fig. 9–35(b). Comparing this device to the 28F004/28F400 in Fig. 9–33(a), we find many similarities. For instance, both devices have an address bus, data bus, and control signals OE, WE, WP, RP, and BYTE, and they serve similar functions relative to device operation. One difference is that there are now two chip-enable inputs, labeled CE₀ and CE₁, instead of just one. Both of these inputs must be at logic 0 to enable the device for operation.

Another change found on the 28F016SA/SV is the addition of the ready/busy (RY/BY) output. This output has been provided to further reduce the software overhead.
Command | Notes | First Bus Cycle | Second Bus Cycle
--- | --- | --- | ---
Read array | 8 | Write | X | FFH
Intelligent identifier | 1 | Write | X | 90H | Read | IA | IID
Read status register | 2,4 | Write | X | 70H | Read | X | SRD
Clear status register | 3 | Write | X | 50H
Word/byte write | | Write | WA | 40H | Write | WA | WD
Alternate word/byte write | 6,7 | Write | WA | 10H | Write | WA | WD
Block erase/confirm | 6,7 | Write | BA | 20H | Write | BA | D0H
Erase suspend/resume | 5 | Write | X | B0H | Write | X | D0H

Address
BA = Block Address
IA = Identifier Address
WA = Write Address
X = Don’t Care

Data
SRD = Status Register Data
IID = Identifier Data
WD = Write Data

Notes:
1. Bus operations are defined in Tables 4 and 5.
2. IA = Identifier Address: A0 = 0 for manufacturer code, A0 = 1 for device code.
3. SRD = Data read from Status Register.
4. IID = Intelligent Identifier Data. Following the Intelligent Identifier command, two Read operations access manufacturer and device codes.
5. BA = Address within the block being erased.
6. WA = Address to be written. WD = Data to be written at location WD.
7. Either 40H or 10H commands is valid.
8. When writing commands to the device, the upper data bus [DQ8–DQ15] = X (28F400 only) which is either VIL or VIH, to minimize current draw.

Figure 9-34 (a) 28F004 command bus definitions. (Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995) (b) Status register bit definitions. (Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995) (c) Erase operation flowchart and bus activity. (Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995)

on the MPU during the erase and programming processes. When this output is 0, it signals that the on-chip write-state machine of the FLASH memory is busy performing an operation. Logic 1 means that it is ready to start a new operation. For the boot block devices we introduced earlier, the busy condition had to be determined through software by polling the WSMS bit of the status register. One approach for the 28F016SA/SV is that software could poll RY/BY as an input waiting for the FLASH device to be ready. On the other hand, this signal could be used as an interrupt input to the MPU. In this way, no software and MPU overhead is needed to recognize when the FLASH memory is ready to perform another operation. This is the default mode of operation and is known as level mode.
The function of the RY/BY output can be configured for a number of different modes of operation under software control. Writing a device configuration code to the 28F016SA/SV does this. For instance, it can be set to produce a pulse on write or a pulse on erase or even be disabled.

The last signal line in the block diagram that is new is the 3/5 input. Note that this input is implemented only on the 28F016SA IC. Logic 1 at this input selects 3.3V operation for $V_{cc}$, and logic 0 indicates that a 5V supply is in use. Since the 28F016SV is a SmartVoltage device, this input is not needed.

Figure 9–35(c) illustrates the blocking of the 28F016SA/SV configured for byte-wide mode of operation. Here we find that the 16MB address space is partitioned into 32 independent 64K byte blocks. Note that block 0 is in the address range from 000000₁₆ through 00FFFF₁₆ and block 31 corresponds to the range 1F0000₁₆ through 1FFFFF₁₆. If
the device is strapped for word-mode operation with logic 1 at the \text{BYTE} input, there are still 32 blocks, but they are now 32K words in size.

Just as for the 28F004/28F400, the 28F016SA/SV supports block locking. However, in these devices, the 32 blocks are independently programmable as locked or unlocked. In fact, there is a separate block status register for each of the 32 blocks. This block status register contains both control and status bits related to a corresponding block. The block-lock status (BLS) bit, bit 6 in this register, is an example of a control bit. When it is set to logic 1 under software control, the corresponding block is configured as unlocked and write and erase operations is permitted. Changing it to logic 0 locks the block so that it cannot be written into or erased. Bit 7, block status (BS), is an example of a status bit that can be read by the MPU. Logic 1 in this bit means that the block is ready, and logic 0 signals that it is busy. When the write-protect (WP) input is active (logic 0), write and erase operations are not permitted to those blocks marked as locked with an 0 in the BLS bit in their corresponding block status register.

Finally, the internal algorithms and hardware of the 28F016SA/SV have been expanded to improve programming performance. For instance, two 256-byte (128-word) write buffers have been added into the architecture to enable paged data writes. Moreover, the programming algorithm has been enhanced to support queuing of commands and overlapping of erase and write operations. Therefore, additional commands can be sent to a device while it is still executing a prior command. They are held in the queue until processed. The overlapping write/erase capability enables the devices to erase one block while writing data to another. All these features result in easier and faster programming for the 28F016SA/SV.
Figure 9-35  (a) Block diagram of the 28F016SA/SV FlashFile memory.  (b) Pin layout.  
(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995)  
(c) Byte-wide mode memory map.  (Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1995)
9.6 WAIT-STATE CIRCUITRY

Depending on the access time of the memory devices used and the clock rate of the MPU, a number of wait states may need to be inserted into external memory read and write operations. In our study of 8088/8086 bus cycles in Chapter 8, we found that the memory subsystem signals the MPU whether or not wait states are needed in a bus cycle with the logic level applied to its READY input.

The circuit that implements this function for a microcomputer system is known as a wait-state generator. Figure 9-36(a) shows a block diagram of this type of circuit. Note that the circuit has six inputs and just one output. The two inputs located at the top of the circuit, CS₀ and CS₁, are outputs of the memory chip-select logic. They could represent

\[ \overline{CS₀}, \overline{CS₁}, MRDC, MWTC, \text{ and RESET} \]

(CLK

(a)

\[ \overline{CS₀}, \overline{CS₁}, \overline{MRDC}, \overline{MWTC}, \text{ and } \overline{\text{SELECT}} \]

(CLK

(b)

**Figure 9-36** (a) Wait-state generator circuit block diagram. (b) Typical wait-state generator circuit.
the chip selects for the program storage and data-storage memory subsystems, respectively, and tell the circuit whether or not these parts of the memory subsystem are being accessed. The two middle inputs are the memory read command (MRDC) and memory write command (MWTC) outputs of the bus controller. They indicate that a read or write operation is taking place to the memory subsystem. The last two inputs are the system reset (RESET) signal and system clock (CLK) signals. The output is READY.

The circuit in Fig. 9–36(b) can be used to implement a wait-state generator for an 8088/8086-based microcomputer system. To design this circuit, we must select the appropriate D-type flip-flop, shift register, and gates.

Let us look briefly at how this wait-state generator circuit works. The READY output is returned directly to the READY input of the MPU. Logic 1 at this output tells the MPU that the current read/write operation is to be completed. Logic 0 means that the memory bus cycle must be extended by inserting wait states.

Whenever an external memory bus cycle is initiated, the D-type flip-flop is used to start the wait-state generation circuit. Before either CS₀, CS₁, or RESET becomes active, the Q output of the flip-flop is held at logic 1 and signals that wait states are not needed. The Q output, logic 0, is applied to the CLR input of the shift register. Logic 0 at CLR holds it in the reset state and holds outputs 0 through 7 all at logic 0.

Whenever a read or write operation takes place to a storage location in the external memory’s address range, a logic 0 is produced at either CS₀ or CS₁ and a logic 0 is produced at either MRDC or MWTC. The active chip-select input makes the D input of the flip-flop logic 1 and the transition to logic 0 by the read/write command signal causes the flip-flop to set. This makes the Q output switch to logic 1 and Q to logic 0. Now READY tells the MPU to start inserting wait states into the current memory bus cycle. The Q output makes both the CLR and data input of the shift register logic 1. Therefore, it is released and the logic 1 at the data input shifts up through the register synchronous with clock pulses from the MPU’s system clock. When the select wait-state output becomes logic 1, it makes the RS input of the flip-flop active, thereby resetting the Q output to logic 0 and Q to logic 1. Thus, the READY output returns to logic 1 and terminates the insertion of wait states, and the MPU completes the bus cycle. The number of wait states inserted depends on how many clock periods READY remains at logic 0. Simply attaching the select wait-state line to a different output of the shift register can change this. For instance, the connection shown in Fig. 9–36(b) represents operation with two wait states.

9.7 8088/8086 MICROCOMPUTER SYSTEM MEMORY CIRCUITRY

In Chapter 8 we introduced the bus cycles, hardware organization of the memory address space, and memory interface circuits of the 8088/8086-based microcomputer system. The earlier sections of this chapter covered the types of memory devices used in the memory subsystem. Here we will show how the memory interface circuits and memory subsystem are interconnected in a simple microcomputer system, shown in Fig. 9–37(a). We will use the information we have acquired to analyze the memory circuits of this microcomputer system.
Figure 9-37  (a) Minimum-mode 8088 system memory interface. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1981)  (b) Minimum-mode 8086 system memory interface. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1979)  (c) Maximum-mode 8088 system memory interface. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1981)
Figure 9-37 (continued)
Program Storage Memory

Earlier we found that program storage memory is used to store fixed information such as instructions of the program or tables of data. This part of the microcomputer's memory subsystem is typically implemented with nonvolatile memory devices, such as the ROM, PROM, EPROM, or FLASH memory. EPROM devices, such as the 2716, 2764, and 27C256, are organized with a byte-wide output; therefore, a single device is required to supply the 8-bit data bus of the 8088. They need to be arranged to provide a word-wide output when used in an 8086 system.

Figure 9-37(a) shows how a 2716 is connected to the demultiplexed system bus of a minimum-mode 8088-based microcomputer. This device supplies 2Kbytes of program storage memory. To select one of the 2K storage locations within the 2716, 11 bits of address are applied to address inputs A0 through A10 of the EPROM. If A0 through A10 of the 8088's address bus supply these inputs, the address range corresponding to program memory is from

\[ A_{10}A_9 \ldots A_0 = 000000000002 = 0000_{16} \]

to

\[ A_{10}A_9 \ldots A_0 = 111111111112 = 007F_{16} \]

assuming that \( A_{11} = 0 \ldots 0 \) generates the chip select signal \( \overline{CS}_x \). Data outputs \( D_0 \) through \( D_7 \) of the EPROM are applied to data bus lines \( D_0 \) through \( D_7 \), respectively, of the 8088's system data bus. Data held at the addressed storage location are enabled onto the data bus by the control signal MEMR (memory read), which is applied to the OE (output enable) input of the EPROM.

In most applications, attaching several EPROM devices to the system bus expands the capacity of program storage memory. In this case, high-order bits of the 8088's address are decoded to produce chip-select signals. For instance, two address bits, \( A_{11} \) and \( A_{12} \), can be decoded to provide four chip-select signals. Each of these chip-selects is applied to the CE (chip-enable) input of one EPROM. When an address is on the bus, just one of the outputs of the decoder becomes active and enables the corresponding EPROM for operation. By using four 2716s, the program storage memory is increased to 8Kbytes.

Now that we have explained how EPROMs are attached to the 8088's system bus, let us trace through the operation of the circuit for a bus cycle in which a byte of code is fetched from program storage memory. During an instruction acquisition bus cycle, the instruction fetch sequence of the 8088 causes the instruction to be read from memory byte by byte. The values in CS and IP are combined within the 8088 to give the address of a storage location in the address range of the program storage memory. This address is output on \( A_0 \) through \( A_{10} \) and latched into the address latches synchronously with the signal ALE. Bits \( A_0 \) through \( A_{10} \) of the system address bus are applied to the address inputs of the 2716. This part of the address selects the byte of code to be output. When the 8088 switches RD to logic 0 and IO/M to logic 0, the control signal generation circuit switches MEMR to logic 0. Logic 0 at MEMR enables the outputs of the 2716 and the byte of data at the addressed storage location is output onto system data bus lines \( D_0 \) through \( D_7 \). Early
in the read bus cycle, the 8088 switches DT/R to logic 0 to signal the bus transceiver that
data are to be input to the microprocessor, and later in the bus cycle DEN is switched to
logic 0 to enable the transceiver for operation. Now the byte of data is passed from the sys-
tem data bus onto the multiplexed address/data bus from which it is read by the MPU.

The circuit in Fig. 9–37(b) shows a similar circuit for a minimum-mode 8086
microcomputer system. Note that because of the 16-bit data bus, two octal transceivers
and two EPROMs are required.

Figure 9–37(c) shows the program storage memory implementation for a maximum-
mode 8088 microcomputer system. Let us look at how this circuit differs from the
minimum-mode circuit of Fig. 9–37(a). The key difference in this circuit is that the 8288
bus controller is used to produce the control signals for the memory interface. Remember
that in maximum mode the code output on status lines S₀ through S₂ identifies the type of
bus cycle that is in progress. During all read operations of program memory, the 8088
outputs the instruction fetch memory bus status code, S₂S₁S₀ = 101, to the 8288. In
response to this input, the bus controller produces the memory read command (MRDC)
output, which is used as the OE input of the 2716 EPROM and enables it for data output.

In the maximum-mode circuit, the 8288, rather than the 8088, produces the control
signals for the address latches and data bus transceiver. Notice that three address latches
are again used, but this time the ALE output of the 8288 is used to strobe the memory
address into these latches. ALE is applied to the STB inputs of all three latch devices in
parallel. The direction of data transfer through the data bus transceiver is set by the DT/R
output of the bus controller and the DEN output is used to generate the OE input of the
transceiver. Since DEN, not \( \overline{DEN} \), is produced by the 8288, an inverter is constructed
from the NAND gate that drives OE of the transceiver.

Data Storage Memory

Information that frequently changes is stored in the data storage part of the micro-
computer's memory subsystem (e.g., application programs and data). This part of
the memory subsystem is normally implemented with random access read/write memory
(RAM). If the amount of memory required in the microcomputer is small, for instance,
less than 32Kbytes, the memory subsystem will usually be designed with static RAMs.
On the other hand, systems that require a larger amount of data storage memory normally
use dynamic RAMs (DRAMs), which provide larger storage capacity in the same size
package. DRAMs require refresh support circuits. This additional circuitry is not war-
ranted if storage requirements are small.

A 1Kbyte random access read/write memory is also supplied in the minimum-mode
8088-based microcomputer circuit in Fig. 9–37(a). This part of the memory subsystem is
implemented with two 2142 static RAM ICs. Each 2142 contains 1K, 4-bit storage loca-
tions; therefore, they both supply storage for just 4 bits of the byte. The storage location
to be accessed is selected by a 10-bit address, which is applied to both RAMs in parallel
over address lines A₀ through A₀. Data are read from or written into the selected storage
location over data bus lines D₀ through D₇. Of course, through software, the 8088 can
read data from memory either as bytes, words, or double words. The logic level of
MEMWR (memory write), which is applied to the write enable (WE) input of both RAMs
in parallel, signals whether a read or write bus cycle is in progress. MEMR is applied to the OD (output disable) input of both RAMs in parallel. When a write cycle is in progress, RD is at logic 1, which disables the outputs of the RAMs. Now the data lines act as inputs.

Just as for program storage memory, simply attaching additional banks of static RAMs to the system bus can expand data storage memory. Once again, high-order address bits can be decoded to produce chip-select signals. Each chip-select output is applied to the chip-enable input of both RAMs in a bank and, when active, enables that bank of RAMs for operation.

Let us assume that the value of a byte-wide data operand is to be updated in memory. In this case, the 8088 must perform a write bus cycle to the address of the operand’s storage location. First, the address of the operand is formed and output on the multiplexed address/data bus. When the address is stable, a pulse at ALE is used to latch it into the address latches. Bits A0 through A9 of the system address bus are applied to the address inputs of the 2142s. This part of the address selects the storage location into which the byte of data is to be written. Next the 8088 switches DT/R to logic 1 to signal the octal transceivers that data are to be output to memory. Later in the bus cycle, DEN is switched.

Figure 9-38  (a) Devices to be used in the system design of Example 9.3 (p. 458). (b) Memory map of the system to be designed. (c) Memory organization for the system design. (d) Address range analysis for the design of chip select signals CS0, CS1, CS2, and CS3. (e) Chip-select logic.
to logic 0 to enable the data bus transceiver for operation. Now the byte of data is output on the multiplexed address/data bus and passed through the transceiver to the system data bus and data inputs of the RAMs. Finally, the byte of data is written into the addressed storage location synchronously with the occurrence of the MEMW control signal.

The data storage memory circuitry of a minimum-mode 8086 system is also shown in Fig. 9–37(b). Here we see that two banks of RAM ICs are required.

Figure 9–37(c) shows the data storage memory circuit of a maximum-mode 8088 microcomputer. Similar to our description of the program storage memory part of this circuit, the difference between the maximum-mode and minimum-mode data storage memory circuits lies in the fact that the 8288 bus controller produces the control signals for the memory and bus interface logic devices. When the 8088 is accessing data storage memory, it outputs either the read memory (101) or write memory (110) bus status code. These codes are decoded by the 8288 to produce appropriate memory control signals. For instance, the status code 110 (write memory) causes the memory write command (MWTC) and advanced memory write command (AMWC) outputs to become active during all write bus cycles. Figure 9–37(c) shows that MWTC or MEMW is used to drive the WE input of the 2142 SRAMs. When MWTC is at its active 0 logic level, the input
Figure 9-38 (continued)
buffers of the SRAMs are enabled for operation. On the other hand, during read bus cycles, MRDC or MEMR is used to enable the outputs of the SRAMs.

**EXAMPLE 9.3**

Design a memory system consisting of 32Kbytes of R/W memory and 32Kbytes of ROM memory. Use SRAM devices to implement R/W memory and EPROM devices to implement ROM memory. The memory devices to be used are shown in Fig. 9-38(a) (p. 455). R/W memory is to reside over the address range 0000016 through 07FF16 and the address range of ROM memory is to be F800016 through FFFFF16. Assume that the 8088 microprocessor system bus signals that follow are available for use: A0 through A19, D0 through D7, MEMR, and MEMW.

Solution

First let us determine the number of SRAM devices needed to implement the R/W memory. Since each device provides $2^{14} \times 4$ or 16K $\times$ 4 of storage, the number of SRAM devices needed to implement 32Kbytes of storage is

$$\text{No. of SRAM devices} = \frac{32\text{Kbytes}}{(16K \times 4)} = 4$$

To provide an 8-bit data bus, two SRAMs must be connected in parallel. Two pairs connected in this way are then placed in series to implement the R/W address range, and each
pair implements 16Kbytes. The first pair, SRAM<sub>1</sub> and SRAM<sub>2</sub>, implements the address range 00000<sub>16</sub> through 03FFF<sub>16</sub>, and the second pair, SRAM<sub>3</sub> and SRAM<sub>4</sub>, implements addresses 04000<sub>16</sub> through 07FFF<sub>16</sub>. The memory map in Fig. 9–38(b) shows the device allocation for this implementation.

Next let us determine the number of EPROM devices that are needed to implement the ROM memory. In this case, each device provides $2^{14} \times 8$ or 16Kbytes of storage. To implement 32Kbytes of storage, the number of EPROM devices needed is

$$\text{No. of EPROM devices} = \frac{32\text{Kbyte}}{16\text{Kbyte}} = 2$$

These two devices must be connected in series to implement the ROM address range and each device implements 16Kbytes of storage. As shown in the memory map in Fig. 9–38(b), the first device, EPROM<sub>1</sub>, implements the address range F8000<sub>16</sub> through FBFFF<sub>16</sub>. The second device, EPROM<sub>2</sub>, implements the address range FC000<sub>16</sub> through FFFFF<sub>16</sub>.
The memory organization based on the preceding allocation of devices is shown in Fig. 9-38(c). Notice that we have used the various 8088 system bus signals (A₀ – A₁₉, D₀ – D₇, MEMR, and MEMW) to draw the circuit diagram. For example, the MEMW signal is applied to the WE input of all four SRAMs in parallel, but it is not connected to the EPROMs.

The four chip select signals, \( \overline{CS}_0, \overline{CS}_1, \overline{CS}_2, \) and \( \overline{CS}_3 \), that are used in the circuit need to be produced for the appropriate address ranges. To design the circuit for generating the chip-select signals, we first analyze the address ranges as shown in Fig. 9-38(d) to determine the address bits that should be used. For instance, to generate the range represented by SRAM₁ and SRAM₂, \( \overline{CS}_0 \) should be active for \( A_{19}A_{18}A_{17}A_{16}A_{15}A_{14} = 000000₂ \). Similarly the other address ranges tell us which address bits are needed to produce the other chip-select signals. This information is used in Fig. 9-38(e) to design the chip-select logic circuit with 74F138 three-line to eight-line decoders.

REVIEW PROBLEMS

Section 9.1

1. Which part of the primary storage memory is used to store instructions of the program and fixed information such as constant data and lookup tables? Data that changes frequently?
2. What does BIOS stand for?
3. What term is used to refer to programs stored in ROM?
4. Can DRAMs be used to construct a program storage memory?

Section 9.2

5. What is meant by the term nonvolatile memory?
6. What does PROM stand for? EPROM?
7. What must an EPROM be exposed to in order to erase its stored data?
8. If the block diagram in Fig. 9-3 has address lines A₀ through A₁₆ and data lines D₀ through D₇, what are its bit density and byte capacity?
9. Summarize the read cycle of an EPROM. Assume that both \( \overline{CE} \) and \( \overline{OE} \) are active before the address is applied.
10. Which standard EPROM stores 64K 8-bit words?
11. What is the difference between a 27C64A and a 27C64A-1?
12. What are the values of \( V_{cc} \) and \( V_{pp} \) for the intelligent programming algorithm?
13. What is the duration of the programming pulses used for the intelligent programming algorithm?

Section 9.3

14. What do SRAM and DRAM stand for?
15. Are RAM ICs examples of nonvolatile or volatile memory devices?
16. What must be done to maintain valid data in a DRAM device?

17. Find the total storage capacity of the circuit similar to Fig. 9–12 if the memory devices are 43256As.

18. List the minimum values of each of the write cycle parameters that follow for the 4364-10 SRAM: tWC, tCW1, tCW2, tWP, tDW, and tWR.

19. Give two benefits of DRAMs over SRAMs.

20. Name the two parts of a DRAM address.

21. Show how the circuit in Fig. 9–22 can be expanded to 128K × 16 bits.

22. Give a disadvantage of the use of DRAMs in an application that does not require a large amount of memory.

Section 9.4

23. What type of circuit can be added to the data storage memory interface to improve the reliability of data transfers over the data bus?

24. If in Fig. 9–23 the data read from memory is 100100100₂, what is its parity? Repeat the same if the data is 011110000₂?

25. If the input to a 74AS280 parity-checker/generator circuit that is set up for odd parity checking and generation is IH. . . A = 111111111₂, what are its outputs?

26. What changes must be made to the circuit in Fig. 9–24(c) to convert it to an odd parity configuration?

27. Make a drawing similar to that shown in Fig 9–24(c) that can be used as the parity-checker/generator in the data storage memory subsystem of an 8086 microcomputer system. Assume that parity checking is performed independently for the upper and lower banks of the memory array and that the parity error outputs for the two banks are combined to form a single parity error signal.

Section 9.5

28. What is the key difference between a FLASH memory and an EPROM?

29. What is the key difference between the bulk-erase architecture and the boot block or FlashFile architectures?

30. What is the key difference between the boot block architecture and the FlashFile architecture?

31. What architecture is used in the 28F010 FLASH memory IC?

32. What power supply voltage must be applied to a 28F010 device when it is being erased or written into?

33. Give the names of two boot block FLASH devices.

34. What value Vcc power supply voltages can be applied to a Smart 5 boot block FLASH IC? What value Vpp power supply voltages?

35. Name the three types of blocks used in the storage array of the 28F004. How many of each is provided? What are their sizes?

36. Name two FlashFile FLASH memory devices.

37. What is the function of the RY/BY output of the 28F016SA/SV?
Section 9.6
38. What function does a wait-state generator circuit perform?
39. What output signal does the wait-state generator produce?
40. Does the circuit in Fig. 9–36(b) produce the same number of wait states for the memory subsystems corresponding to both chip selects?
41. What is the maximum number of wait states that can be produced with the circuit in Fig. 9–36(b)?

Section 9.7
42. Make a diagram showing how 2764 EPROMs can be connected to form a 16Kbyte program storage memory subsystem. Also show a 16Kword program memory subsystem.
43. If we assume that the high-order address bits in the circuits formed in problem 42 are all logic 0, what is the address range of the program memory subsystems?
44. How many 2142 static RAMs would be needed in the memory array of the circuit in Fig. 9–37(a) if the capacity of data storage memory were to be expanded to 64Kbytes?
45. How many 2716 EPROMs would be needed in the program memory array in the circuit of Fig. 9–37(a) to expand its capacity to 96K bits? If 2732s were used instead of 2716s, how many devices are needed to implement the 96K-bit program memory?
46. Repeat the design in Example 9.3 for the 8086 microprocessor system bus signals A₀ through A₁₉, D₀ through D₁₅, MEMR, MEMW, and BHE. Use the same memory and device specifications.