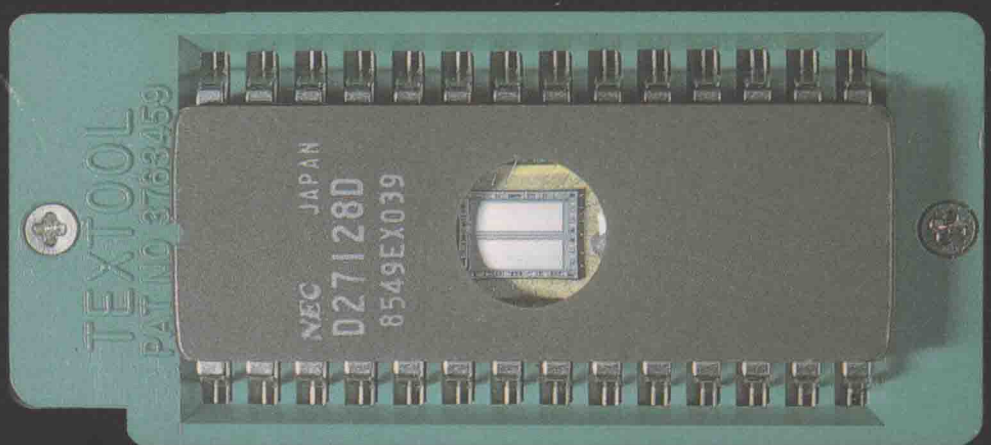
2962

EXPERIMENTS WITH EPROMS



DAVE PROCHNOW

ADVANCED TECHNOLOGY SERIES

Experiments with EPROMS

By Dave Prochnow

TAB Books
Division of McGraw-Hill, Inc.
New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

Trademark List

The following trademarked products are mentioned in *Experiments with EPROMs*.

Apple Computer, Inc.: Apple *II*e
Autodesk, Inc.: AutoCAD 2
Bishop Graphics, Inc.: E-Z Circuit
Wintek Corporation: HiWIRE
smARTWORK
Borland International: Turbo BASIC
MicroPro International, Inc.: WordStar

To my patient and understanding Kathy

TAB Books is a division of McGraw-Hill, Inc.

pbk 10 11 12 13 14 15 16 FGR/FGR 9 9 8
hc 1 2 3 4 5 6 7 8 9 10 FGR/FGR 8 9 8

Library of Congress Cataloging-in-Publication Data

Prochnow, Dave.
Experiments with EPROMS / by Dave Prochnow.
p. cm.
Includes index.
ISBN 0-8306-0362-X ISBN 0-8306-2962-9 (pbk.)
1. Computer storage devices—Experiments. I. Title.
TK7895.M4P76 1988
621.397'3—dc19 88-1508
CIP

Acknowledgments

Significant contributions were made by several outstanding manufacturers during the preparation of this book. Borland International, CAD Software, Inc., Heath/Zenith, Kepro Circuit Systems, Inc., and Wintek Corporation all made generous hardware and software contributions which served as important references in developing the following text.

Introduction

EPROM programming is one of the great mysteries in advanced circuit design. Here is a discrete, dual in-line package that is capable of holding any user-supplied programming in a nonvolatile fashion. Furthermore, this silicon-housed carrier can be quickly erased at the discretion of the user and reprogrammed at a modest cost. Unfortunately, there is a fixed set of perennial problems which continually hamper the widespread usage of EPROM technology.

Basically, these EPROM restrictions can be broken down into three standard questions. How do I program an EPROM? How can I incorporate an EPROM into my circuit design? And, what are my alternatives to using an EPROM? Up until now, these questions were only answered by a select group of individuals who had braved the uncharted reaches of EPROM programming. Even their answers were far from definitive, however. Extensive pioneering work in EPROM technology has yet to be performed at the experimenter's level. This is a field that is ripe for discovery.

In actual practice, the implementation of EPROMs into digital circuits is a relatively easy task. Virtually, any design that uses microprocessor control or requires external data for its operation can be modified or adapted to accept EPROM programming. To achieve this desired digital memory goal, you will need three things: an EPROM programmer, an EPROM eraser, and an extensive guidebook. At this moment, you are holding the direct solution to the latter requirement and the indirect answer to the previous two necessary EPROM elements.

This book contains complete circuit information for building a number of EPROM programmers, erasers, and dedicated projects. Each of these circuits

is based on a varying level of design need. These programmers, erasers, and projects represent working solutions to every major EPROM usage, including stand-alone and computer-based units. But this book doesn't stop at hardware answers. There is also a complete introduction into the theory of digital circuitry and digital memory devices. Therefore, not only will you learn how to construct several powerful EPROM programmers and erasers, but you also receive an education in the technology of programmable memories.

Following a brief introduction into human memory, there is a complete examination of the current state of digital memory. Once you have mastered the basics of memory fabrication, three chapters list the major types of digital memories that can be found on today's market. After you have gained the necessary background information in EPROM technology, the remainder of this book covers the actual programming and erasing of EPROMs.

Fifteen different programmers, erasers, and EPROM-based circuits are fully detailed in four valuable chapters. All of these projects are advanced in the methods that they use for their construction and operation. Therefore, beginning readers might need some background training prior to attempting the construction of these EPROM projects. There are three appendices, however, that can guide you through many of these areas. Two important considerations that you should make before attempting to build any of these projects are the purchases of both a quality soldering iron and a reliable digital logic probe. While the need for the former is obvious, you will find that the logic probe's ability to pinpoint logic status errors is invaluable when troubleshooting EPROM wiring problems.

If the construction of an EPROM programmer and/or eraser doesn't satisfy your needs, then this book's chapter dealing with commercial programmers and erasers can help you explore EPROM technology. The inclusion of this chapter expands the potential of incorporating EPROMs into your next circuit design. Thus, you have two different methods for acquiring the benefits of nonvolatile memory.

Finally, after all of your creative juices have been stimulated with the knowledge of creating your own EPROM circuits, this book's last chapter gives a thumbnail look at alternate memory technologies. So all three of the EPROM programming requirements have been met by this book. Both an EPROM programmer and an EPROM eraser can either be built from the circuits that are contained within this book or purchased from a manufacturer. Additionally, the information in this book supplies all of the reference material that is necessary for mastering EPROM technology. Therefore, the conclusion of this book leaves you with only one remaining task:

54 68 61 6E 6B 73 20
66 6F 72 20
74 68 65 20
6D 65 6D 6F 72 69 65 73 2E

Contents

List of Projects vii

Acknowledgments viii

Introduction ix

1 EPROM Technology 1

Digital Memory • Binary Codes • Binary Mathematics • Binary Logic • Project 1: Boole's Box • Project 2: Keyboard Encoder

2 Programmable Memory Structures 47

Multi-bit Registers • Digital Memory • ROM Technology • PROM Technology • EAROM Technology • EPROM Technology • EEPROM Technology • PLA Technology • Building the X-3

3 Popular PROMs 65

4 Popular EEPROMs 73

5 Popular EPROMs 79

6 The Bit Smasher 92

EPROM Programming Considerations • Project 3: Construction of the Bit Smasher • Testing the Bit Smasher • Using the Bit Smasher • Project 4: The Bit Smasher II

7 EPROgraMmer 113

Project 5: Construction of the EPROgraMmer • Testing the EPROgraMmer • Using the EPROgraMmer • Project 6: The EPROgraMmer II • Project 7: The Three-Line Burner • Project 8: The ABeEP I • Project 9: The EPROM Program Tester • Project 10: Building a ROM Drive

8 Programming an EPROM 137

Intel Hex • Simple EPROM Programming • Synthesized Speech Data Table (Project 11: Speech Synthesizer) • Music Data Table (Project 12: Music Synthesizer) • Character Data Table (Project 13: Message Center) • Advanced EPROM Programming

9 Erasing an EPROM 154

Erasing Mathematics • Project 14: A 4-Watt EPROM Eraser • Project 15: An 8-Watt EPROM Eraser

10 Commercial Programmers and Erasers 161

The Heath ID-4801 Programmer • Building the Heath ID-4803 Eraser • Expanding the Heath EPROM Programmer

11 SAM Technology 177

Charge-Coupled Device • Bubble Memory • Josephson Junction Device

A Building an EPROM Project 181

B IC Data Sheets 209

C Supply Source Guide 217

Glossary 220

Bibliography 223

Index 227

Includes list of devices mentioned in this text.

List of Projects

Project 1: Boole's Box 41
Project 2: Keyboard Encoder 45
Project 3: Bit Smasher 99
Project 4: Bit Smasher II 107
Project 5: EPROgraMmer 116
Project 6: EPROgraMmer II 123
Project 7: Three-Line Burner 126
Project 8: ABeEP I 127
Project 9: EPROM Program Tester 130
Project 10: ROM Drive 135
Project 11: Speech Synthesizer 142
Project 12: Music Synthesizer 144
Project 13: Message Center 145
Project 14: 4-watt Eraser 157
Project 15: 8-watt Eraser 159

1

EPROM Technology

Perform this series of three simple tests: What was the name of your kindergarten teacher? Now, what was the name of your college advisor? Finally, what is your social security number? How'd you do? Your ability to correctly recall each of these answers depends largely on your command of a relatively mysterious process which inhabits the human brain—the memory.

Memory is an ephemeral product derived from the interactions between over 15 billion neurons inside the average human brain (see Bibliography for further reference materials supporting this study). These interactions form a neurochemical change that is retained over lengthy periods of time in spite of other brain activities. There are several unknown variables in this simplified explanation of human memory, however. For example, how are these neuronal interactions initiated? What length of exposure is required for commiting a name, idea, or concept to memory? And what is the duration of a "memorized" thought?

Any study of memory must begin with the derivation of its definition. The etymology of this noun, "memory," has its origin in Latin's *memoria* and *memor*. Complementing this literary definition, the biological derivation of memory stems from the reactions within the single-celled neuron. This small grayish cell is the basis for all neurological activity. Unfortunately, the action of these neurons in the formation of memory is far more complicated.

There are two general schools of thought on the structure of the neuronal biology of the brain. The first is the *net theory* of neuron placement. As espoused by Camillo Golgi, this net theory concluded that the brain's neurons were all interconnected in dense, multiple layers of vast nerve cell networks. Golgi further stated that impulses, ideas, and memory traveled this enormous

neuronal network similar to trains through a railway system. The complicated mechanics of this type of structure made this theory difficult for some biologists to swallow and alternate theories were quickly formulated.

One of these opposing views came from Ramon y Cajal of Spain. Cajal's theory stated that neurons were discrete individual cells with no distinct point of articulation. In this theory, an impulse or idea travels from neuron to neuron through a proximate space known as a *synaptic cleft* or *synapse* (see Fig. 1-1). Therefore, Cajal's synapse serves as a communication point between individual neurons with impulses able to travel in any direction to any nearby neuron. The flexible concept of synaptic communication between neurons has made Cajal's theory more widely accepted than Golgi's nerve cell network theory. Support in the scientific community can be fickle, however.

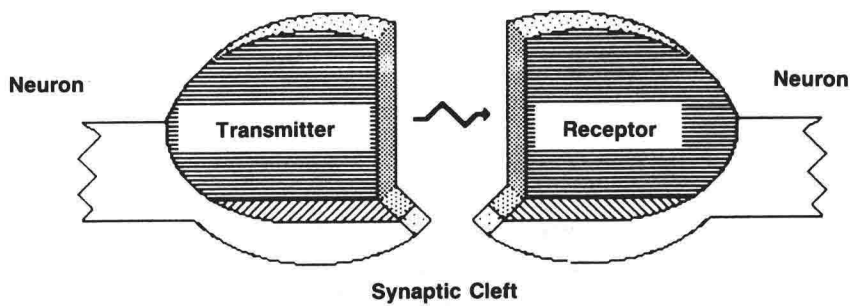


Fig. 1-1. Diagram of a neuronal synapse in the human brain.

This point is illustrated by the 1906 Nobel Prize awards ceremony. In 1906, the Nobel Prize Committee couldn't make a definitive decision on the "best" neuronal theory. Therefore, the Nobel Prize in Medicine was awarded to both Cajal and Golgi. In spite of this dual selection, Cajal's theory was quickly adopted by the medical world as the most sound explanation of impulse and idea transmission within the human brain. An interesting footnote to this widespread refutation of nerve network impulse conveyance is that several new studies are now lending a degree of credence to Golgi's theory (see Bibliography). Who knows, maybe the Nobel Prize Committee was correct after all in dividing the 1906 award between both Cajal and Golgi?

Restricting our neurological study to the more generally accepted Cajal theory, there are three main components of a neuron: the cell *body*, the *axon*, and the *dendrite* (see Fig. 1-2). Each of these neuronal parts serves a key role in sending an impulse through the brain. Basically, an impulse travels from one neuron's axon to another cell's dendrite. This process is carried out between any neuron across any synapse. The resultant impulse can culminate in an action, a reflex, or an idea. But what exactly is an impulse?

Neuronal impulses are electrochemical signals that are carried along neurons through the various axon/dendrite synaptic joints. The nature of these

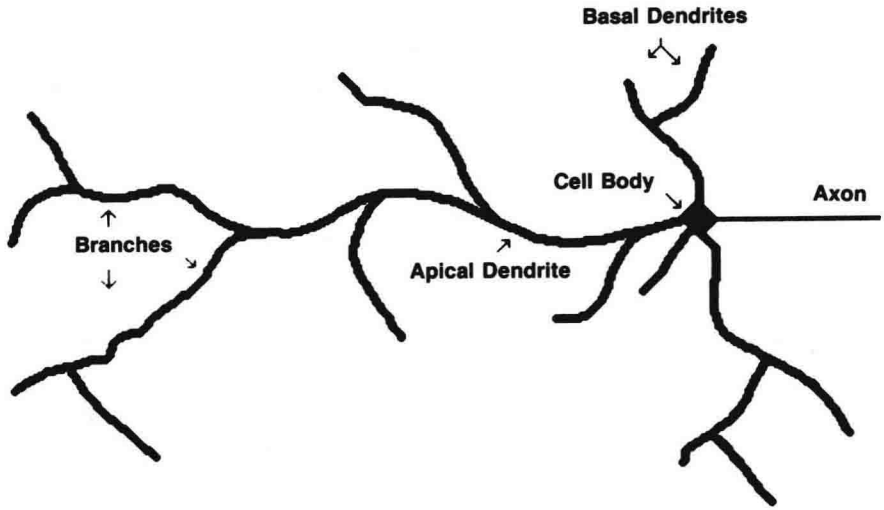


Fig. 1-2. The cellular parts of a human neuron.

impulses can be demonstrated through a simple biological experiment. By sending a low-voltage electrical charge through the exposed nerve fibers of a dissected frog leg, the inert leg can be made to twitch. Furthermore, the fluid that is produced from these electrically charged nerves will cause another muscular contraction when applied directly to the leg muscle. Therefore, the sequence of events during impulse transmission consists of:

1. An electrical charge.
2. The production of a neuronal chemical.
3. The transmission of the neuronal chemical.
4. The absorption of the chemical by another neuron.
5. The production of an electrical charge.

This process is repeated thousands of times until the final neuron causes the muscle to contract.

In terms of a single neuron, this sequence is initiated in the cell body through the production of a weak electrical current. Once this electrical charge has been generated, the signal travels down the neuron's axon to the synapse. At this junction, the impulse releases special compounds in the transmitting neuron's cell membrane and the electrical signal is changed into a chemical signal.

A neuron's chemical signal is composed of neurotransmitters that float across the synaptic cleft and latch onto another neuron's dendrite. This dendritic attachment selectively alters the receiving neuron's cell membrane and forms an electrical potential. Once again, this electrical potential travels

through the neuron, along the axon, and the entire process is repeated thousands, millions, or billions of times for each and every impulse.

This elementary introduction into the biology of memory has some interesting parallels in the electronics of digital memory. Like its neuronal equivalent, digital memory can be either *volatile* or *nonvolatile* (i.e., permanent). Similarly, memory is conveyed as an electrical impulse in both systems. One sharp difference between human memory and digital memory, however, is that digital memory will never forget the name of your kindergarten teacher; that is, not unless you erase it.

DIGITAL MEMORY

Any discussion of digital memory must begin with a solid introduction to digital logic. By definition, digital logic is the sequence of events within a digital circuit. This sequencing is governed by a strict application of mathematics. Entering into this mathematical logic scheme are the two possible conditions or states that can be present in a digital circuit: OFF or ON.

Several different names are given to this dual state depending on their circumstances of occurrence. For example, the two conditions of a digital signal are labeled as LOW and HIGH for the OFF and ON states, respectively. Alternatively, in a graphics system like a computer video display, the digital representation used for indicating the OFF and ON status of the individual pixels found on a monitor screen is either dark or light, respectively. Finally, in a microcomputer's MPU (Microprocessing Unit), these OFF and ON conditions are interpreted numerically as a 0 and a 1. This final numeric symbolic definition is used throughout the ensuing discussions of digital logic and introduction to digital memory.

These 0's and 1's of digital logic are manipulated with the *binary* or base-2 number system. Like other number systems, the selective combination of the binary number system's 0 and 1 can be used for expressing any numeric value. Table 1-1 compares the same sequence of values for four different number systems. One unfortunate side effect to writing numbers in binary notation is their unwieldy dimensions. For example, consider the following decimal value along with its binary equivalent:

222 (decimal) = 11011110 (binary)

In this example, the binary value is a lengthy eight digits versus three digits for the decimal representation. Therefore, a more practical means for dealing with the binary states of digital logic is through a higher-level number system.

While the decimal or base-10 number system is more comprehensible to the human user, the *octal* (base-8) and *hexadecimal* (base-16) number systems mesh more easily with the digital circuit's multiples-of-four data and address requirements. The handling of these data and address conditions is

Table 1-1. Four Different Numbering Systems.

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

principally executed by several single-binary-digit switching circuits or through a single multi-binary digit register. In an average digital circuit, these registers range in size from 4 to 64 binary digits. This dependence on factors of four makes translating between octal and hexadecimal values and binary digits a necessary talent that must be acquired by the digital circuit designer. Even so, the use of octal programming has recently fallen into disfavor and the hexadecimal number system has become a veritable standard in the standardless microcomputer industry.

As an exercise in number system manipulation, perform the following experiment:

Purpose: Write a decimal-to-binary conversion program.
Materials: Any microcomputer system with a high-level language interface (e.g., BASIC, FORTRAN).
Procedure:

- Make your program short with a limited number of jumps or GOTO statements.
- Your program should be able to calculate any bit-size binary value.
- Test your final program with known binary numbers and compare your achieved results with your acquired results.

Results:

✤ This is one possible solution to this experiment in BASIC:

```
5 REM BINARY CALCULATOR PROGRAM
10 INPUT ''HOW MANY DIGITS '';A
20 B=0:C=0:D=0:Z=0:PRINT ''ENTER YOUR'';A;''-DIGIT
BINARY NUMBER'':INPUT QB$
30 D=A-1
40 FOR B=1 TO A
50 IF MID$(QB$,B,1)=''1'' THEN LET C=2^D:Z=Z+C:D=D-1
60 IF MID$(QB$,B,1)=''0'' THEN LET C=0:Z=Z+C:D=D-1
70 NEXT B
80 PRINT QB$;''=''
90 PRINT TAB(A+1) Z
100 INPUT ''DO ANOTHER: Y OR N ? '';X$
110 IF X$=''Y'' THEN GOTO 10
120 IF X$=''N'' THEN END
130 IF X$<>''Y'' OR X$<>''N'' THEN GOTO 100
```

✤ After you have completed this experiment, add an octal and hexadecimal conversion option to your final program.

Referring to a register size as multi-binary digit can be almost as cumbersome as writing large binary numbers. Therefore, another means for expressing binary digits is with bits. A bit can equal either a 0 or a 1. Applying this definition to the previously mentioned 8-digit binary number example 11011110 yields an 8-bit number. When dealing with register bit size, however, the final register value can represent the computational strength of a microcomputer.

At the heart (or brain) of every microcomputer is the MPU. This single chip or IC (integrated circuit) is the repository of the CPU's registers. Based on the bit width of these registers, the data handling ability of the MPU can be fairly judged (or can it?). Unfortunately, not all registers are created equal and many MPUs are difficult to pigeonhole into an accurate statement of their true computing power.

Take the Motorola MC68000 MPU, for example. The MC68000 has 32-bit internal registers, but this same MPU also has a 16-bit data bus, a 23-bit address bus, and a 16-bit ALU (arithmetic logic unit). Furthermore, this IC is able to address 16M bytes of unsegmented memory with a 32-bit program counter. So where does this mixed bag of bit width leave us? Granted, the MC68000 can handle 32-bit-sized instructions, internally. On the other hand, it can only

receive data in 16-bit slices. Therefore, the MC68000 is best classified as a 16/32-bit MPU.

Another MPU that falls into this dual personality mold is the Intel 8088. The 8088 is best labeled an 8/16-bit MPU. In this case, there are 16-bit internal registers with an 8-bit data bus, a 20-bit address bus, and 16-bit instruction pointer (the function of the instruction pointer is similar to the role of the program counter in the MC68000). Additionally, the ALU is 16 bits wide. Once again, however, the 16-bit 8088's ability to receive data in 8-bit pieces mandates the 8/16-bit qualifier in describing its computational strength.

In addition to describing the amount of data that can be processed, the bit size of a register also indicates the number of logical states that are possible. All possible logic states for two different registers are listed in Table 1-2. Based on the states listed in these two examples, an X-bit register will contain 2^X logic states.

Table 1-2. Register Logic States.

2-bit Register	
bit 1	bit 0
0	0
0	1
1	0
1	1

3-bit Register		
bit 2	bit 1	bit 0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Yet another means for expressing the computational strength of a microcomputer is in its CPU's (central processing unit) memory size. Memory size is usually measured in 8-bit segments. Each 8-bit chunk is defined as a *byte*. Bytes, in turn, are frequently described in terms of kilobytes (K bytes), megabytes (M bytes), and gigabytes (G bytes). A 1K-byte CPU memory is equal

to 2^{10} or 1024 bytes. This same 1K-byte CPU memory also contains 8192 bits (8 X 1024 bytes). Continuing with these equivalencies:

$$1M \text{ bytes} = 2^{100} \text{ or } 1{,}048{,}576 \text{ bytes}$$
$$1G \text{ bytes} = 2^{1000} \text{ or } 1{,}073{,}741{,}824 \text{ bytes}$$

In turn, these same values equal:

$$1M \text{ bytes} = 8{,}388{,}608 \text{ bits}$$
$$1G \text{ bytes} = 8{,}589{,}934{,}592 \text{ bits}$$

Bit strings of a byte's width are also useful for indicating the CPU's status. Depending on its register location, a byte can have a variety of different meanings.

For example,

00111111

is equal to,

63 (decimal)

3F (hexadecimal)

77 (octal)

and can represent the 8088 instruction code AAS. This code is a mnemonic representation of "ASCII adjust for subtraction." The use of the term *ASCII* points to another possible interpretation of this byte. ASCII (American Standard Code for Information Interchange) is a coding system that uses 7 bits of data for describing alphabetic, numeric, and punctuation characters in a digital circuit. Continuing with our previously mentioned byte,

00111111

represents the ASCII character

?

Another alphanumeric character description code is *EBCDIC* (Extended Binary Coded Decimal Interchange Code). This code is used primarily on mainframe IBM systems such as the IBM 360/370. In addition to being restricted to mainframe usage, EBCDIC data is 8 bits wide. Therefore, the byte 00111111 is equal to the EBCDIC control character SUB or substitute.