

the Textbook of

# Video Game Logic

volume I

# Video Game Logic

First Printing 1976

Copyright © 1976 Kush N' Stuff Amusement Electronics, Inc.

All rights reserved. No part of this book may be reproduced by any mechanical, photographic or electronic process, or in the form of a phonograph recording, nor may it be stored in a retrieval system, transmitted or otherwise copied for public or private use without written permission from the Publisher.

Kush N' Stuff Amusement Electronics 60 Dillon Avenue, Campbell, California

# Table of Contents

<b>CHAPTER 1</b>	<b>Digital Integrated Circuits</b>	<b>Page</b>
1.1	Introduction . . . . .	1
1.2	Logic Families . . . . .	1
1.3	Integration . . . . .	2
1.4	Levels of Integration . . . . .	3
1.5	The Binary Counting System . . . . .	4
1.6	Timing Diagrams & Wave Form Nomenclature . . . . .	6
1.7	The Basic Digital Rules . . . . .	7
1.8	Active HI Vs. Active LO . . . . .	8
1.9	Mathematical Notation . . . . .	8
1.10	Inverters . . . . .	8
1.11	AND & NAND Gates . . . . .	9
1.12	OR & NOR Gates . . . . .	10
1.13	Chip Enables . . . . .	11
1.14	The Exclusive OR . . . . .	12
1.15	More Complicated Devices . . . . .	13
1.16	Clock Inputs . . . . .	14
1.17	Functions of the Flip-Flops . . . . .	14
1.18	R-S Flip-Flops . . . . .	15
1.19	D-Type Flip-Flops . . . . .	15
1.20	J-K Flip-Flops . . . . .	16
1.21	MSI Devices . . . . .	17
1.22	Counters . . . . .	17
1.23	Constructing a Simple 4-Bit Binary Asynchronous Counter . . . . .	18
1.24	7493 Four Bit Binary Asynchronous Counter . . . . .	19
1.25	Synchronous Counters . . . . .	20
1.26	Presettable Counters . . . . .	20
1.27	9316 Synchronously Presettable Four Bit Counter . . . . .	21
1.28	74193 Presettable Up/Down Counter . . . . .	22
1.29	9314 Quad Latch . . . . .	23
1.30	Shift Registers . . . . .	23
1.31	Serial-In/Serial-Out Shift Register . . . . .	24
1.32	Serial to Parallel Converter . . . . .	24
1.33	Parallel to Serial Converter . . . . .	24
1.34	Multiplexers . . . . .	26
1.35	Decoders . . . . .	27
1.36	Other Decoder Applications . . . . .	27
1.37	Adders . . . . .	29
<b>CHAPTER 2</b>	<b>Theory of TV Monitor Operation</b>	
2.1	Introduction . . . . .	31
2.2	General . . . . .	31
2.3	Types of Scanning Systems . . . . .	31
2.4	Raster Scan Monitor Operation . . . . .	31
2.5	Making Your Own Monitor . . . . .	34
2.6	Types of Raster Scans . . . . .	36
2.7	The Non-Interlaced Raster Scan . . . . .	36
2.8	The Interlaced Raster Scan . . . . .	37

## Table of Contents (cont.)

<b>CHAPTER 3</b>	<b>Video Game Architecture</b>	<b>Page</b>
3.1	Common Themes . . . . .	39
3.2	A Simplified Overview . . . . .	39
3.3	Functions & Relationships of Common Circuits . . . . .	40
3.4	Power Supplies . . . . .	40
3.5	Coin & Credit Circuitry . . . . .	41
3.6	Master Timing . . . . .	42
3.7	Motion Circuitry . . . . .	43
3.8	Score Circuits . . . . .	43
3.9	Memory Circuits . . . . .	45
3.10	Video Summation Networks . . . . .	46
3.11	Sound Circuits . . . . .	46
<b>CHAPTER 4</b>	<b>Power Supplies</b>	
4.1	Introduction . . . . .	48
4.2	The LM309K . . . . .	48
4.3	A Simple +5 Supply . . . . .	48
4.4	A More Complicated Supply . . . . .	49
4.5	Another Approach . . . . .	50
4.6	Ripple in the +5 Line . . . . .	51
<b>CHAPTER 5</b>	<b>Master Timing</b>	
5.1	Introduction . . . . .	53
5.2	The Oscillator . . . . .	53
5.3	Horizontal Submultiples . . . . .	55
5.4	Horizontal Reset . . . . .	57
5.5	Horizontal Blanking & Sync. . . . .	57
5.6	Vertical Submultiples & Reset . . . . .	58
5.7	Vertical Blanking & Sync. . . . .	59
5.8	Sync Summing . . . . .	59
5.9	An Interlaced Game . . . . .	61
5.10	Horizontal Main Timing . . . . .	61
5.11	Vertical Main Timing . . . . .	64
5.12	V INFO . . . . .	65
<b>CHAPTER 6</b>	<b>Motion</b>	
6.1	Introduction . . . . .	68
6.2	The Illusion of Motion . . . . .	69
6.3	Vectored Motion . . . . .	69
6.4	A Typical Motion Circuit . . . . .	75
6.5	Vertical Motion Control . . . . .	77
<b>CHAPTER 7</b>	<b>Creating Images</b>	
7.1	Introduction . . . . .	80
7.2	Generated Vs. Stored Displays . . . . .	80
7.3	Building a Generated Display . . . . .	81
7.4	The Window Concept . . . . .	81
7.5	An Actual Generated Display . . . . .	82
7.6	Addressing Memory Locations . . . . .	87

## Table of Contents (cont.)

	<b>Page</b>
7.7 A Diode Matrix Display . . . . .	88
7.8 How It Works . . . . .	90
7.9 Integrated ROMs . . . . .	93
7.10 An Integrated ROM Circuit . . . . .	95
 <b>CHAPTER 8            Score Circuitry</b>	
8.1 Introduction . . . . .	102
8.2 A 7-Segment Score Display . . . . .	103
8.3 Score Storage . . . . .	103
8.4 Score Window . . . . .	106
8.5 Score Display . . . . .	106
 <b>CHAPTER 9        Paddle Generation and Control</b>	
9.1 Introduction . . . . .	107
9.2 The 555 Timer . . . . .	107
9.3 Player Selectable Paddle Sizes . . . . .	110
9.4 Paddle Size and Summing . . . . .	110

## INTRODUCTION

The instant success of the first video games has given birth to a significant new industry overnight and thrust a complex new technology into the largely unprepared hands of amusement machine operators. The popular appeal of video games is such that no operator of coin-op devices can afford to overlook their potential. On the other hand, the myriad of circuit designs and the complexity of video game computers has increased service problems by an order of magnitude for operators without trained digital technicians.

Today's coin-op amusement industry is becoming increasingly more competitive for each locale generally has several operators of electro-mechanical and electronic games all vying for essentially the same clientele. Those operators running the newest and most stimulating games will inevitably come out ahead when all the quarters are counted. Obviously, an operator cannot ignore video games and still attempt to compete in the amusement marketplace. But while the introduction of electronic games has been a marvelous tonic for the entire industry, the undesirable side effects of this stimulant — namely service problems — have still not been adequately dealt with.

The circuit designs and overall architecture of the first video games seemed totally incomprehensible to operators long accustomed to servicing their simpler electro-mechanical counterparts. However, the truth of the matter is that these first games were excellent examples of design simplicity and component minimization because at that time only relatively crude effects were enough to fascinate eager players. Today, jaded players have become bored by the myriad of variations of these first games and increasingly more dramatic game action is required to stimulate the average player who might still play a fifteen year old pin ball machine, but is not at all interested in last year's video game.

This being true, electronic games designers are incessantly on the prowl for new effects and innovative circuit designs. The simpler type of computer architecture found in the early games is clearly giving way to vastly more complex circuit designs, particularly as the semiconductor industry makes greater and greater strides in the areas of large and very large scale integration and inexpensive micro-processor chips which now make possible revolutionary new approaches to video games.

Unfortunately, this trend is having a more serious impact on operators and game service personnel each day. Technicians who familiarized themselves with the first video games and who have kept current with recent innovations are now in a unique position because as the technological gap increases, fewer and fewer game operators are able to retain the capability of servicing their own game computers. And, as circuit design evolves into evermore complex directions, the prospect of starting up a repair facility becomes less and less attractive.

However, it is not yet too late to begin learning about video games. In fact, now is probably the best time ever as certain future trends are becoming quite clear. It is still feasible for service personnel to jump into the relatively simple medium-scale random-logic designs currently being produced. But in a few years or less, as game computer architecture moves

inexorably into complex large-scale micro-processor based systems, it will become extremely difficult for the novice technician to learn to repair the new varieties of video games. The changeover from today's random-logic designs to tomorrow's micro-computer games is a quantum leap. Since this change has only just recently begun, the opportunity to become involved with electronic games is now at its ripest stage.

Games manufacturers, in their eagerness to protect valuable circuit designs, have been quite reluctant to educate and orient others to their products. Fearful of training a new generation of competing engineers, games manufacturers have, in general, played their cards fairly close with the logical result that few individuals at the operating end of the industry are familiar with how these games really work. In the beginning, it was not always even possible to obtain the schematics for a particular game. Now, most manufacturers freely distribute their circuit designs, however this information is of little value to operators unfamiliar with the new technology.

So there is a clear need for practical educational materials which will enable the entrance and intermediate level technicians to make use of documentation now available from manufacturers of electronic games. The function of this book is to provide a vehicle for familiarizing video game technicians with some primary knowledge of digital electronics and an intermediate level of understanding of video game computer operation. Obviously it cannot be within the scope of this book to train expert level digital technicians. But almost anybody with the proper motivation should be able to fully comprehend video game computers and fix nearly all malfunctions encountered simply by reading this book, attending a few relevant and informative seminars and by getting some genuine hands-on experience.

Successful video game troubleshooting depends on having the right test instruments, a basic knowledge of digital electronics (which is not nearly as complicated as you might have imagined), a thorough understanding of video game fundamentals and specific documentation such as schematics and test point data for the malfunctioning computer. Clearly, the most important of these ingredients is fully understanding how the various circuits of a game work for an on-the-ball technician needs very little in the way of test equipment. A firm comprehension of digital devices and circuit operation eliminates the need for expensive "hand-holding" instrumentation such as logic comparators. Believe it or not, a knowledgeable technician armed only with a voltmeter, a logic probe and a video probe (\$100 total cost) will be able to solve just about all the problems normally associated with video game computers.

To be sure, oscilloscopes and a few other more esoteric instruments are desirable, especially for the professional technician but they are not essential. What is absolutely essential is the knowledge of why the video game computer is constructed the way it is and how the specific devices and circuits actually operate.

Technicians totally unfamiliar with the entire technology will obviously need to begin by first learning the basics of digital electronics. Unfortunately, the scope of this book simply will not allow us to delve too deeply into the realm of semiconductor devices other than to review the basics of computer logic and the operation of commonly used integrated circuits. The subject of analog devices can only be covered superficially in this text for it is a whole subject unto itself. Obviously, video games will make little sense without some basic know-

ledge of transistors, resistors, capacitors and other analog components, however this information is easily obtained from a great number of other sources. Although we will discuss the functions of these components in their circuits, we will not unduly concern ourselves with the specifics of analog device operation.

After reviewing the fundamentals of digital electronics and devices, we will move into a discussion of the TV monitor. While knowledge of monitor construction and operation is not absolutely essential to repair video game computers, it is extremely useful for comprehension of computer architecture. Since the entire game computer is essentially designed around the signal requirements of the TV monitor, the general design of game computers will make little sense unless the operation of the monitor is somewhat familiar. The greatest emphasis in the monitor section will center around the concept of TV raster operation relative to image display for it is this aspect which bears the most on computer design.

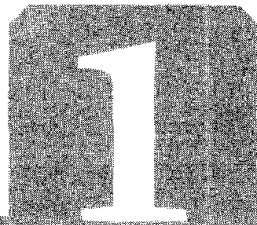
After building a solid foundation in digital devices and TV monitors, the section dealing with video game computer architecture should be considerably easier to comprehend. In this section, the functions of various circuits will be described so the computer can be visualized as a whole system before delving in the operation of specific circuits. The operation and construction of these specific circuits should then be much easier to understand since the reasons they are required and the components they are built from will have already been analyzed. Moreover, since much of a game's computer circuitry is interdependent, conceptualizing the system as a whole becomes an important troubleshooting factor.

Successive chapters will deal with important circuit types and the specifics of circuit operation. All games share a great deal of similar circuitry since every game must operate via certain established parameters. This commonality is manifested in the design of coin, player control, sync and score circuitry. So even though the circuits discussed in the theory of operation chapters will probably never be exactly duplicated, there will still be a great deal of positive transference between these example circuits and malfunctioning ones encountered in the real world. All the major types of circuits including power supplies, sync generation, motion, credit circuitry, playfield image generation and paddle circuits will be discussed in depth. Wherever possible, we have contrasted a simpler circuit from an early game with a more complex contemporary scheme. Additionally, test points and corresponding test data have been included for many circuits so that you may become familiar with the established test point format and typical kinds of test point data.

The final step in the process is actually getting down to repairing malfunctioning video games. To do this, knowledge and equipment are two very important ingredients. However, another variable known as support documentation comes into play at this point. Without a complete set of computer schematics and wiring diagrams, troubleshooting all but the most obvious malfunctions is virtually impossible. To an experienced digital technician familiar with most games, a set of schematics may be the only documentation necessary. However, to many entrance and intermediate level technicians, a set of schematics often raises more questions than answers in terms of how various circuits actually operate when functional and what information should be expected from reading applicable test instruments.



An excellent source for the answers to these questions is the Video Game Data Library's *Data Books* and *Computer Service Manuals*. Each manual is an exhaustive analysis of a particular game and contains a great amount of information available from no other source. In addition to providing a set of logically organized and clearly drawn schematics, the manuals contain simply explained theory-of-operation descriptions for every circuit. All circuits are keyed to a block diagram for quick familiarization with the general architecture of the game and the subtle relationships between circuits. The theory-of-operation sections discuss every component in each circuit and describe all the events and signals which occur in that circuit. But most important, each section schematic contains a number of test points adequate to fully troubleshoot the circuit where each numbered test point has a corresponding explanation detailing the types of test instruments required and what the readings should be.



# **digital integrated circuits**

## 1.1 INTRODUCTION

The advent of digital integrated circuits has not only completely revolutionized the electronics industry, but it has touched almost every facet of modern human life. Even if you were not involved with video computers, you would still be surrounded with digitally-operated devices and the products they output. Almost everyone has an electronic calculator or a digital clock and even kitchen stoves are currently being built with a custom large scale integrated circuit which does everything from telling you when the roast is done to performing simple calculations. Recent advances in large scale integrated circuits have made powerful and inexpensive home computers a reality and the day will soon arrive when even your car will be controlled by a small electronic brain. Every time you pick up a phone, make a bank deposit or step into an elevator, an electronic device makes sure you are serviced quickly and properly. Computers are inescapable, so you just might as well sit down and learn to enjoy them.

Fortunately, almost all computerized devices operate by the same rules and processes even though the actual circuit configuration may vary greatly. This means that once you have learned the basics of digital devices you should be able to move into any field where they are applied and quickly pick up the particular application. Therefore, if you have already had some digital experience, video game computers should be quite easy to understand once the general architecture of the system has been explained. Conversely, once you have learned about video games, you should be able to move into other fields such as micro-computers for although these devices are somewhat more complex, they still follow the same basic rules explained in this chapter. And if you become familiar with the new generation of micro-computer games, you will be in an excellent career position for almost every device needing control will be implemented with a micro and the possibilities for employment will be unlimited.

## 1.2 Logic Families

Although there are a great many types or *families* of digital integrated circuits, only one is used in the construction of the games discussed in this text. This family is known as transistor-transistor-logic or TTL for short. Other logic families have been around a great deal longer than TTL, but are not used in contemporary video games for a number of reasons.

RTL (resistor-transistor-logic) predates TTL by quite a number of years and was widely used in its day. Although still found in some devices, RTL has become an obsolete technology because of its low fan-out capability (ability to drive other devices), its relatively low speed (about 30 ns propagation delay), its high noise susceptibility (picks up spurious signals from other places) and its inability to lend itself to medium and large scale levels of integration. DTL (diode-transistor-logic) is another ancestor of TTL and is still used to some extent since its output impedance is quite a bit lower than that of RTL and it is capable of driving an acceptable number of other devices. Although the speed of DTL is approximately the same as RTL, it does find some applications because of the fact that it is about three times more noise immune. However, in spite of this, DTL is rapidly becoming an extinct family. ECL (emitter-coupled-logic) technology is relatively old but still widely used because of its high speed (propagation delay on the order of 3 ns) and it has even seen some limited video game action because of this factor. However, ECL is extremely noise susceptible which necessitates special printed circuit board construction, layout and shielding to keep extraneous signals off input lines. Due to the relative difficulty of implementing ECL designs, it is used only where very high speed is an absolute necessity. MOS (metal-oxide-semiconductor) technology is a relatively new field with many branches such as CMOS, NMOS, PMOS and VMOS where each type finds a particular application. MOS technology has quickly become the leader of the pack because, although it is considerably slower than many other families, MOS devices are capable of

greater size reduction and therefore find application in large and soon-to-be very large scale integrated circuits.

Since the trend of the semiconductor industry is currently to produce chips which provide complete logic systems, MOS technology has become the most important new family, or at least the most glamorous. Calculator chips, watch circuits, micro-processors and custom LSI designs are all examples of MOS implementation. Although not currently used in the random-logic video game designs found in this text, it is conceivable that alert video game manufacturers could have complex MOS circuits custom built for more or less standard circuits such as sync circuits where one MOS chip would replace both counter chains and all necessary gates and flip-flops. Unfortunately, MOS devices are not directly compatible with TTL voltage levels, however they may be interfaced to TTL components through the use of level shifters which change the MOS output voltages to levels acceptable to TTL devices. Some MOS devices are even built with the level shifters on the chip itself which makes interfacing considerably easier.

This brings us to TTL and why TTL is so widely used today. Everything else being equal, the most important consideration for any manufacturer when selecting a logic family is price. TTL is a relatively old family and has been around long enough that development costs incurred by the semiconductor manufacturers have long ago been recovered. Hence, these devices may be produced quite cheaply. Moreover, the semiconductor manufacturers have acquired a great deal of experience in TTL process technology with the result that TTL yields are high. It is interesting to note that initially even TTL devices were quite expensive. It was not too long ago that the simple 7483 4-bit adder was the latest hot item and carried an appropriate price tag (approaching \$80 each, believe it or not!). When you compare this with today's micro-processor (i. e. Intel's 8080A) which is at least a thousand times more complex yet costs under \$30, you can see the great strides which have been made in the last few years.

TTL technology has many other advantages other than price in terms of its operational characteristics and ease of manufacturing and it is these factors which contributed to making TTL such a popular family to begin with. TTL is both relatively fast and noise immune which makes it acceptable for almost all circuit designs where normal speeds are required. Furthermore, TTL lends itself to medium scale levels of integration meaning that all devices from simple gates to more complex counters, shift registers and multiplexers are available in the same family. But most important, there are more TTL circuit types currently available than any other family which means the logic designer can usually find just about everything he needs already inside a TTL chip. This results in tremendous savings in terms of printed circuit board real estate and assembly costs.

### 1.3 Integration

The word *integration* refers to the process by which a great number of *discrete* or separate components are combined into large and complex circuits all contained within a single package. The original circuit design may be very large and contain thousands of transistors and other elements, however the final chip may only measure 1/8 of an inch on each side.

The manufacturing processes by which integrated circuits are formed is an extremely complex field and we do not mean to demean it by the following cursory discussion, however the fact remains that the scope of this book simply will not allow an in-depth exploration into this fascinating area. At any rate, integrated circuits are produced by depositing layers of semiconductor materials on highly polished silicon wafers using photographic processes. The initial design is produced by a computerized drafting machine directed to cut areas out of a piece of rubylith (red plastic) material so a pattern or *mask* results.

These masks are greatly reduced in size and *step-photographed* so a large number of them can be grouped together on a glass slide. Layers of semi-

conductor materials are then deposited on the wafer and the mask is used to selectively deposit a *resist* so only certain areas of the semiconductor material remain under the resist after the rest is etched away with another compound. The resist is cleaned off and the process repeated a number of times until all the layers have been deposited and etched away. Needless to say, each step of the process requires a different mask.

The result of this process is a wafer (generally 3" in diameter) containing hundreds of individual chips or *dice* where each die is separated by a thin strip of unused silicon. At this point, each chip on the wafer is automatically tested by a computerized machine which places microscopically small contacts on the pads of the chip and functionally tests all parameters of the device. Bad chips are marked with a small paint spot before the wafer is sent on to the next step. Generally, the yield of good chips is surprisingly small and usually falls into the range of 30% to 50% for a process the manufacturer has well under control. The entire tested wafer is then sent to an area where a diamond scribe is used to make a minute scratch between each chip. The wafer is clasped between two flexible sheets and rolled over a device which causes the chips to break apart where scribed, much the same way as sheets of glass are cut apart. The good chips are separated from the bad ones by a mechanized sorting machine operated remotely.

The good chips are taken to another area for a process known as *wire bonding*. Here, the chip is mounted to the *lead frame* and super-fine wires are welded from the legs of the lead frame to the pads on the chip. Then, a plastic or metal top is mounted to the frame so that only the legs or pins of the lead frame protrude. The legs are bent down and the chip is complete.

Depending on the intended use of the chip, it is tested a number of different ways. Commercial and industrial ICs are run through a number of functional electrical tests to determine that they still work after wire bonding. Military and aerospace quality chips are further subjected to a number

of grueling environmental tests to verify the hermeticity of the package (ability to resist the intrusion of moisture and gasses) and the operational characteristics for the specific temperature range. A typical MILSPEC test involves subjecting the poor IC to live steam at a very high temperature and pressure inside a modified pressure cooker.

Ceramic packages sealed with glass are used for the higher grades of ICs to increase the hermeticity of the device while plastic packages are normally used for the cheaper grades. Since almost all dice are glass passified, a small amount of package leaking will not usually affect the device.

#### 1.4 Levels Of Integration

The physical size of an IC is absolutely no clue to the degree of complexity of the circuit contained within it. For example, a 14-pin DIP (dual in-line package) may contain 6 inverters, the simplest of all circuits where each inverter has one input and output pin (and two pins for power and ground make 14 pins in all). Or, a slightly larger package may contain a huge and amazingly complex circuit such as the new 4K RAM chips where the input lines are multiplexed together to reduce the number of pins required.

Small and simple devices such as inverters, gates and flip-flops fall into the classification of *small-scale integration* (SSI). Since these devices generally require fewer control inputs, several of them may be found in a single 14 or 16 pin DIP. SSI devices occupy little area on the wafer, hence they are less likely to fail because of imperfections in the wafer itself. Also, since they are smaller, there is less likelihood of imperfections in depositing and etching away the layers of semiconductor materials with the logical result that there is a significantly higher yield from SSI production and the per chip prices are consequently much lower.

More complex *medium scale integrated* (MSI) devices such as counters, multiplexers, latches and shift registers require more control lines so there is usually only one in a package. Since these cir-

cuits are considerably larger physically than SSI chips, there is a greater likelihood of imperfections occurring the whole way down the manufacturing process which greatly reduces yields. Whereas acceptable yields for SSI may be 30% to 50%, manufacturers are content to get only 20% to 40% yields from their MSI processes. Since it still takes the same amount of effort to produce fewer MSI devices from the same size wafer, MSI chip prices are somewhat higher.

Since TTL technology does not lend itself to large circuit configurations, manufacturers have turned to MOS processes for *large scale integrated* (LSI) circuits. MOS technology requires less power and space than TTL so it is suited to larger designs. But it does tend to be quite a bit slower. The relatively low speed of MOS has encouraged the development of another process known as I<sup>2</sup>L (or *integrated injection logic*) which is much faster than MOS yet still has a relatively small gate size permitting the implementation of large circuit designs. Although most LSI designs are currently being implemented with various MOS processes, I<sup>2</sup>L technology may well move into a dominant position in years to come.

LSI chips have only just recently been introduced into the field of video games where two completely different approaches have been taken. One manufacturer has had an entire game reduced to a single custom LSI chip. Other manufacturers producing microprocessor-based designs are also using LSI chips for the *central processing units* (CPUs) and other functions as these chips are now available.

The future will bring us *very large scale integration* (VLSI) within the next several years. For example, process engineers are currently working on a 65K dynamic RAM process. Ordinarily, this product would not be feasible because of the large size of the chip which, if it were produced in accordance with existing technology, would be bound to encounter an imperfection in the substrate material. However, engineers are working on a process which stacks the refresh capacitor on top of the

gate thereby reducing the size of the circuit to one-fourth of what it takes today. If the substrate material can be made to somewhat higher tolerances, this product will be a reality.

### 1.5 The Binary Counting System

Hopefully you were paying attention in grammar school instead of throwing spit balls and picked up some useful information on changing number bases because computers do not count like humans. If not, understanding the following material is absolutely essential before any of the digital device section will make sense.

Everybody is familiar with the decimal counting system because it is the one we normally use everyday. We humans have ten fingers or *digits* so we have learned to count by tens or *decades*. Computers, on the other hand, have only two "fingers" — *on and off* — and therefore must count by groups of two. The *binary* or *base 2* system follows the same rules as the base ten system except there are only two digits — 0 and 1 — to work with whereas the decimal system allows us a total of ten digits to describe our number. Binary counting may take a bit of practice before you feel completely comfortable with it, but it is by no means difficult.

But before getting into the binary system, let's make sure we all agree on how the decimal system works. In the decimal system, each *place* contains successively higher multiples or powers of ten (Figure 1.5-1). The first is the 1s place or 10<sup>0</sup> (any number raised to the zero power is 1). The second is the 10s place or 10<sup>1</sup> (any number raised to the power of one is simply that number). The

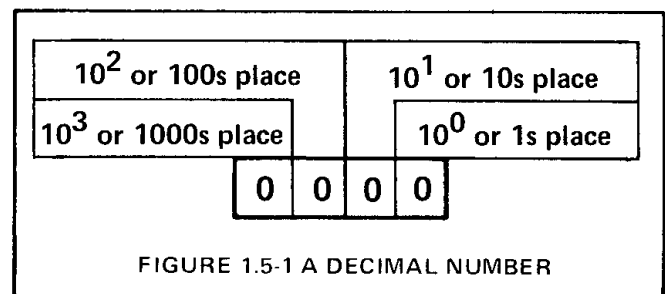


FIGURE 1.5-1 A DECIMAL NUMBER

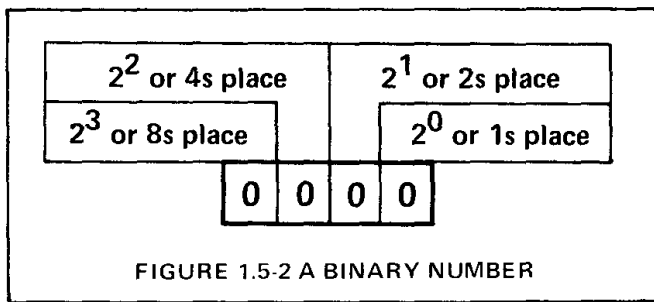
third is for 100s or multiples of  $10^2$  (10 x 10). The fourth is for multiples of  $10^3$  or 1000s (10 x 10 x 10).

Now let's illustrate this with a real decimal number. For example, the decimal number 3256 consists of:

3 in the  $10^3$  place or . . . . 3000  
 2 in the  $10^2$  place or . . . . 200  
 5 in the  $10^1$  place or . . . . 50  
 6 in the  $10^0$  place or . . . . 6  
 3256

The binary system works just the same way except that each place contains successive multiples of two instead of ten. The first place is the 1s or  $2^0$  place. The second is the 2s or  $2^1$  place. The third is the 4s or  $2^2$  place and the fourth is the 8s or  $2^3$  place. You figure the next two out. The fifth is the  $2^4$  place. The fifth is the  $2^4$  or the 16s place. The next after that is the  $2^5$  or 32s place.

We can visualize this with the following simulated binary number.



For practice, let's parse the binary number 0 1 0 1. In this number we have a

0 in the  $2^3$  place or . . . . 0  
 1 in the  $2^2$  place or . . . . 4  
 0 in the  $2^1$  place or . . . . 0  
 1 in the  $2^0$  place or . . . . 1  
 5

Figure 1.5-3 illustrates the binary count up to the decimal number 15. The first number, zero, has no value so "0" appears in all places. The second is decimal one and it is expressed by the binary equivalent of 0 0 0 1 ( a 1 in the 1s place). The number 2 is 0 0 1 0 (a 1 in the 2s place). Three is the addition of 1 (0 0 0 1) and 2 (0 0 1 0) or 0 0 1 1.

DECIMAL EQUIVALENT	BINARY NUMBER			
	$2^0$	$2^1$	$2^2$	$2^3$
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

FIGURE 1.5-3 BINARY TRUTH TABLE Notice the symmetry in the columns of 1s and 0s. You can see that each column contains alternating groups of 1s and 0s which double with each increasing bit.

Before moving on, let's notice a few relevant things about this truth table. The most significant item is that the term *place* is known as a *bit* in digital lingo and the complete expression is generally known as a *word*. A word may begin with either the *most* significant digit (MSB) or the *least* significant digit (LSB). Although not frequently associated with random-logic video games, the term *byte* usually refers to one-half of a 16-bit word. And somebody has even come up with *nybble* which is

one-half of a byte. Also, notice that the numbers in the  $2^0$  column alternate single 1s and 0s, that the numbers in the  $2^1$  column alternate double 1s and 0s and so on. The last item will take on a bit more significance when we get to the operation of binary counters.

You can see from the truth table that only one bit is needed to express the binary equivalent of the decimal number 1, that two bits are required to describe numbers 2 and 3. Three bits are sufficient for numbers 4 through 7, but it takes all four bits for 8 through 15. The number 16 would be 1 0 0 0. Now here's a heavy question: how is the binary equivalent of decimal 17 expressed? What about 18? Okay, those were pretty easy, so try 29.

If you said "1 1 1 0 1" to the last question, you were quite right. To find the binary equivalent of any decimal number, you start by subtracting the highest possible power of two less than the decimal number. Since 16 is the highest power we can subtract from 29 (32 would put us over the edge), the result of the subtraction is 13. So, we put a 1 in the 16s or  $2^4$  place and our number looks like 1 ? ? ? ?. We then subtract the next highest power of two which happens to be 8 and in fact 8 can be subtracted from 13 leaving 5, so we put a one in the 8s or  $2^3$  place so our number now looks like: 1 1 ? ? ?. The next power is 4 which is also subtractable from 5, so we drop a 1 in the 4s place and get 1 1 1 ? ?. This leaves a result of 1 and we can't subtract 2 from 1 so we put a 0 in the 2s place for 1 1 1 0 ?. Since only 1 is left, we complete the process by subtracting a 1 and by placing a 1 in the  $2^0$  bit for 1 1 1 0 1. The result is and always must be 0 after all the subtractions.

Numbers greater than nine bits are rarely used in video games since, generally speaking, 454 is about the largest number needed and we will see exactly why once we get to the sync section. With a total of nine bits we could conceivably count all the way up to 1 1 1 1 1 1 1 1 1 or 511.

Although we humans can count and express num-

bers by using numerals, computers must use LO (voltage) for binary 0 and HI (voltage) for binary 1. In a moment, we will see how a digital counter chip can count from 0 0 0 0 to 1 1 1 1 (decimal zero to fifteen) at its four outputs by producing the proper combinations of HIs and LOs at the appropriate output pins. But in the meantime, we need to discuss more basic digital devices.

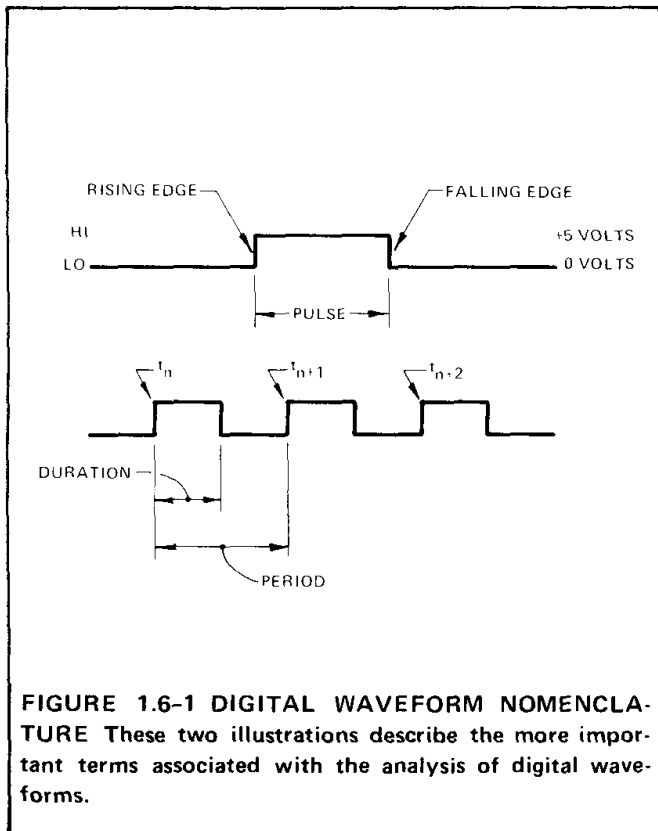
## 1.6 Timing Diagrams & Waveform Nomenclature

Timing diagrams are often used to illustrate important timing relationships between signals as they are processed by the computer and are especially useful where the developmental process is fairly complicated. Timing diagrams more or less simulate the appearance of the waveform on an oscilloscope CRT which facilitates troubleshooting. Progressive timing diagrams are used extensively in this text to show significant relationships between developmental signals and the final processed output. Most of these diagrams will contrast digital signals, however there is an occasional need for illustration of analog waveform development.

Figure 1.6-1 illustrates some of the more important terms associated with digital waveforms. The *pulse* is that period of time in which the signal changes or is *active*. By changing, we mean that process by which the signal passes from one logic level to another. Figure 1.6-1 shows a pulse which changes from the LO or 0 volt logic level to the HI or +5 volt level. The total amount of time the signal changes, or pulses HI in this case, is known as the *duration* of pulse. The part of the waveform which rises from the LO to the HI level is the *rising edge* of the pulse and the term *risetime* describes the amount of time required to make this transition. The *falling edge* is that part of the waveform which provides transition back from the HI to the LO level. These two terms may take on additional significance when dealing with the operation of rising or falling *edge triggered* devices. A rising edge triggered device becomes active during the rising edge of the input pulse.

When dealing with *pulse trains* or signals in which





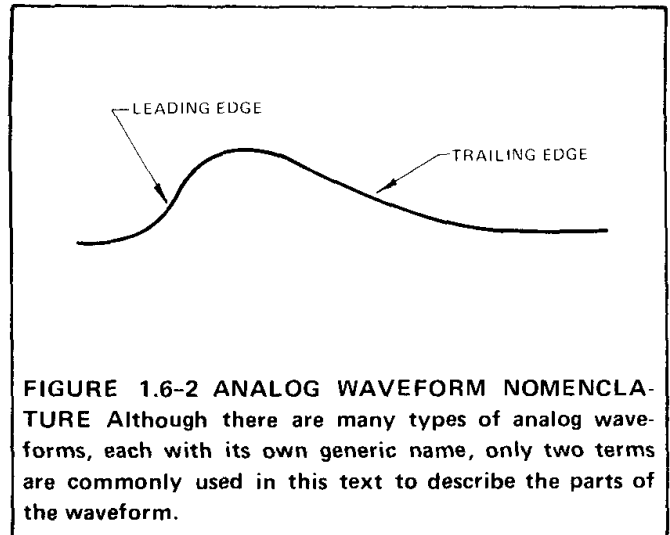
pulses are being produced on a regular and continuous basis, a few other terms take on significance. The term *frequency* refers to the *rate* at which the pulses occur and *period* describes the amount of time between the rising edge of one pulse and the rising edge of the one immediately following. The *off-time* of the signal is the period minus the duration. This then allows us to describe the *duty cycle* of the signal which is defined as the ratio of on-time to off-time.

When discussing events initiated by CLOCK (more about this term later in the chapter), a particular timing notation is often used. In this case,  $t^n$  refers to the rising edge of the first clock pulse and  $t^{n+1}$  describes the second clock pulse.

CLOCK waveforms have equal periods of on and off time, hence they are often called *square waves* and for this reason, a waveform with this appearance is often said to be *symmetrical*.

There are many different types of analog waveforms, however only a few analog waveform terms

are relevant to our discussion. First of all, the rising edge of this type of waveform is called the *leading edge* and the falling part is known as the *trailing edge*. The terms *attack* and *decay* are synonymous with the leading and trailing edges. The waveform in Figure 1.6-2 has a sharp attack and a long decay.



### 1.7 The Basic Digital Rules

All logic devices and circuits are constructed from three basic building blocks governed by these rules: AND, OR and NOT. The first states that when two signals are ANDed together, the output is HI only when *both* inputs are HI. The OR principle states that the output will be HI if *either or both* inputs are HI. A variation of the OR principle is the exclusive OR and, in this case, the output will be HI if *either, but not both*, signals are HI. The NOT rule states that the output of an inverter is always inverted to the logic level opposite the input.

Inverted or *active LO* signals are indicated by a bar over the signal name. For example,  $\overline{\text{START}}$  (pronounced "start not") drops to the logic LO level when the start button is pressed, and START rises HI simultaneously. Overscored signals are *always* at the logic level opposite their non-overscored counterparts.

## 1.8 Active HI vs Active LO

Before moving on to the section on digital device operation, one more point needs clarification. Inputs to devices which affect when or how an event takes place are known as control inputs. The event takes place when the control line changes states and this may occur if a HI pulse occurs on a normally LO line (active HI) or when a LO pulse appears at a normally HI input (active LO). It is easier to understand circuits where signals go HI to initiate events, however the active LO system is considerably more flexible. Consequently, most MSI and some SSI chips are provided with active LO control inputs. Since active LO controls are so prevalent, most chips also have inverted control outputs so that devices will be directly compatible without using inverters.

## 1.9 Mathematical Notation

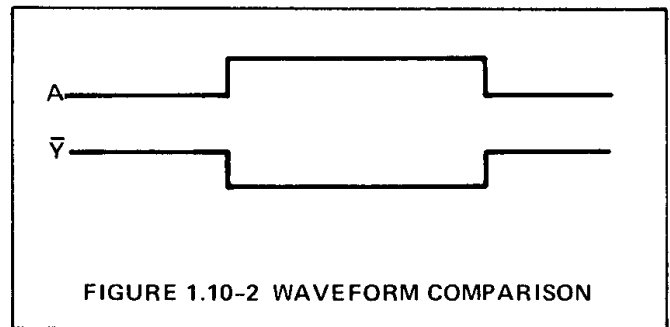
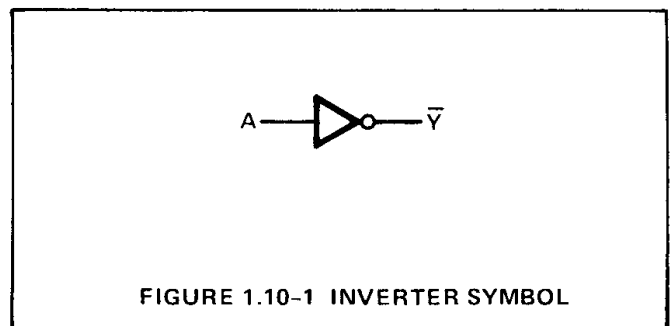
The operation of logic gates and more complicated systems can be expressed using mathematical notation. The primary usefulness of this notation is in logic design where logic circuits are occasionally engineered by implementing mathematical equations in hardware. This would bear little significance to troubleshooting video games if it were not for the fact that outputs are often named with their mathematical expressions. When using this notation,  $(\cdot)$  indicates the AND operation,  $(+)$  denotes the OR functions and  $(\oplus)$  indicates the exclusive OR operation. The logic of an AND gate is expressed by the statement  $A \cdot B = Y$  where A and B are the inputs and Y is the output. A two-input AND gate would function by using the equation  $A \cdot B = Y$ . Consequently, the output signal of an AND gate where the input signals are named CAR and CRASH might be named CAR · CRASH.

The logic of a three-input OR gate may be expressed by a similar equation,  $A+B+C = Y$  or by COIN+START+REPLAY. If inversion is used in the equation, it is expressed by an overscore on the appropriate signal. For example, if three signals are NORed, the result may be expressed by  $A+B+C = \bar{Y}$  or simply by  $\overline{A+B+C}$ . Additionally,

combinations of these symbols may be used for more complicated expressions such as  $\bar{B} \cdot (A+D)$ . Combinatorial logic expressions follow the same rules as combinatorial algebraic equations.

## 1.10 Inverters

The output of an inverter simply provides the inverse logic level of the input. If we enter signal A at the inverter input, the output signal  $\bar{Y}$  will be LO for the period of time A is HI and visa versa. The operation of the inverter may also be expressed by the simple equation  $A = \bar{Y}$  or  $\bar{A} = Y$ .



The operation of logic gates and even entire systems is often demonstrated by truth tables. The primary usefulness of truth tables is in the study of complex systems where the table allows easier visualization of the effects of changing input signals. But first, we need to start off with some simpler tables and the inverter truth table is the most basic of all.

A more visual demonstration of inverter logic is light colored column in 4a symbolizes when signal

A	$\bar{Y}$
0	1
1	0

FIGURE 1.10-3 INVERTER TRUTH TABLE

A is Hi, 4b will reveal the inverse of 4a, hence there will be dark areas for  $\bar{Y}$  where ever A is light.

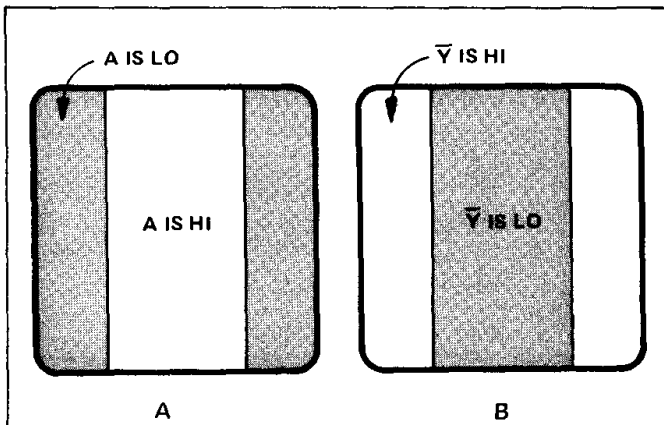


FIGURE 1.10-4 INVERTER OPERATION This type of illustration is a symbolic diagram where light areas correspond to the time the signal is HI and dark areas represent LO periods. This kind of diagram is useful not only because it demonstrates gating operations in a visual way, but also because it simulates how the signal would actually appear on a TV monitor.

Since the inverter circuit requires only one input and one output, six of them are packaged in a single 14-pin DIP, hence the name “hex inverter”. The remaining two pins are used for power and ground.

### 1.11 AND & NAND Gates

Signals are ANDed together when an output event is to occur only if two or more other conditions exist. For example, let’s say we want to trigger an explosion sequence when two objects collide. The objects can be represented by the HI pulses of input signals A and B in Figure 1.11-2. The output of the AND gate is denoted by signal Y. Since the AND rule states that the output of the AND gate will be HI only during the time both inputs are HI,

the Y pulse will represent the period of time both object signals exist simultaneously or are concurrent. If we want the explosion sequence to last only for the duration of the collision, then we use the Y pulse to set the time limit. If we would like the explosion to last for a longer period of time, the HI Y pulse can be used to trigger another, longer lasting pulse from a device such as a flip-flop or a one-shot.

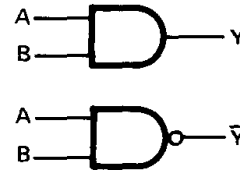


FIGURE 1.11-1 AND & NAND SYMBOLS

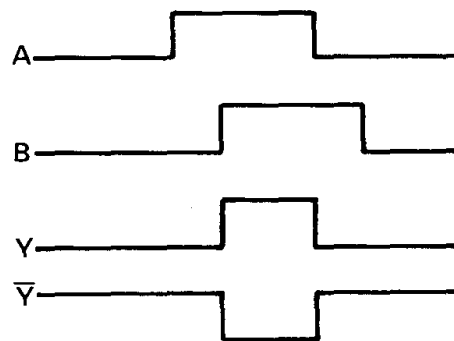


FIGURE 1.11-2 AND & NAND LOGIC

Since a NAND gate is simply an AND gate with an internally inverted output, its output is denoted  $\bar{Y}$  and it pulses LO during the time the output of the AND gate is HI.

The AND and NAND truth table is also quite

A	B	Y	$\bar{Y}$
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

FIGURE 1.11-3 AND & NAND TRUTH TABLE

simple. Notice that the  $Y$  and  $\bar{Y}$  outputs are always at the logic levels opposite each other.

To illustrate AND logic in a more graphic manner, let's call the white columns in Figures 1.11-4a and 1.11-4b the times when input signals A and B are HI. The output of the gate can be HI only when both inputs are HI which occurs at the intersection of signals A and B. Consequently, a single white square remains in Figure 1.11-c. Had we desired to illustrate NAND logic, the square would appear LO or black.

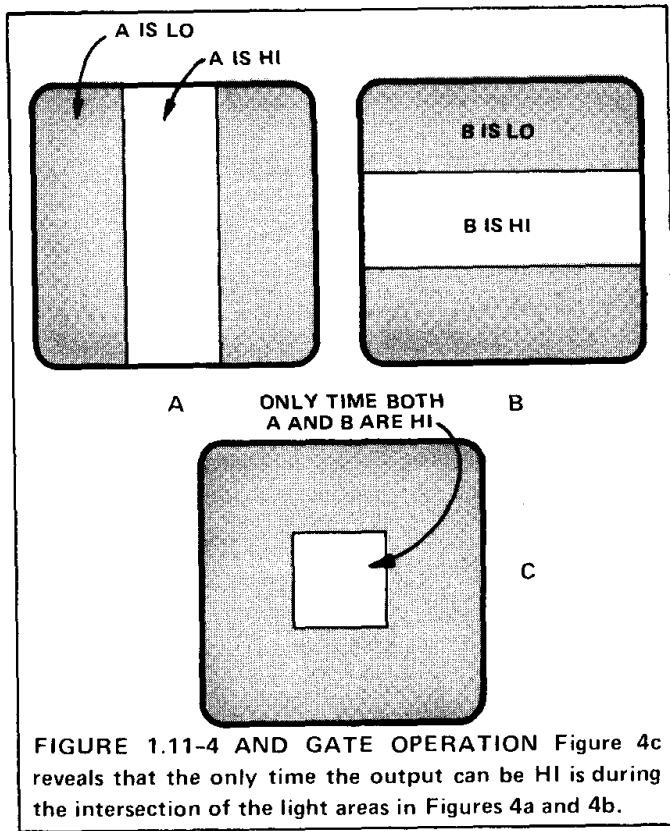


FIGURE 1.11-4 AND GATE OPERATION Figure 4c reveals that the only time the output can be HI is during the intersection of the light areas in Figures 4a and 4b.

Depending on the number of inputs to an AND or NAND gate there may be as many as four separate gates per package as in the case of the two-input gates. AND and NAND gates are manufactured with as many as eight inputs per gate, however there can only be one per package.

To illustrate a realistic situation using AND logic, let's say we are building a video driving game and we want to subtract points from a player if he crashes his car into a roadblock. In order to sub-

tract the points, we must have a circuit which detects when a collision occurs. We can detect if the car is driven into a roadblock by the following simple circuit.

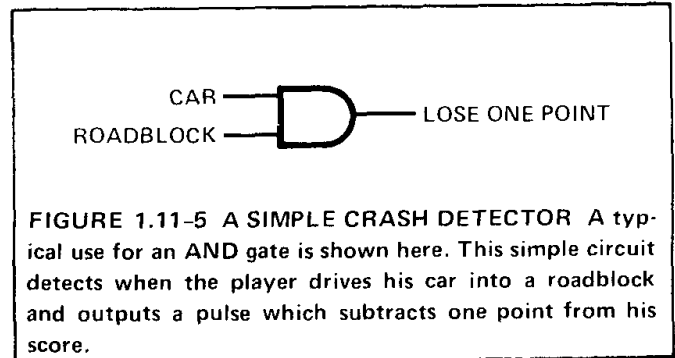


FIGURE 1.11-5 A SIMPLE CRASH DETECTOR A typical use for an AND gate is shown here. This simple circuit detects when the player drives his car into a roadblock and outputs a pulse which subtracts one point from his score.

If the player drives his car into a roadblock, CAR and ROADBLOCK occur simultaneously and a LOSE ONE POINT signal is generated which can be taken to the player's score counters to decrement his score.

## 1.12 OR & NOR Gates

Signals are ORed together if an output event is to occur when either or both of two conditions are present. This is illustrated in Figure 1.12-2 by waveforms where you can easily see that the output pulse  $Y$  is HI during the time either A or B is HI. Since a NOR is no more than an inverted OR, its output pulse  $\bar{Y}$  is LO during the time  $Y$  is HI.

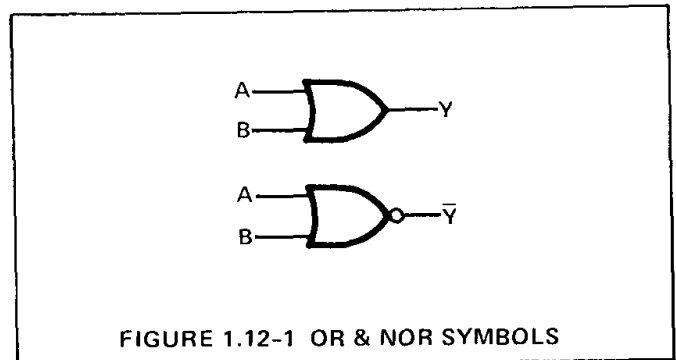
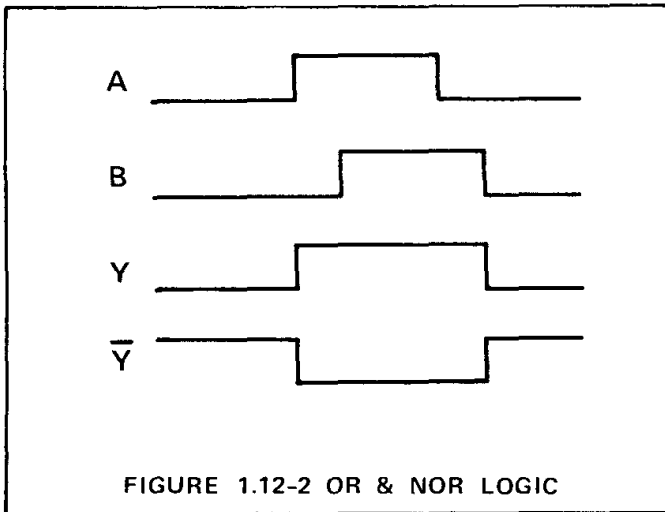


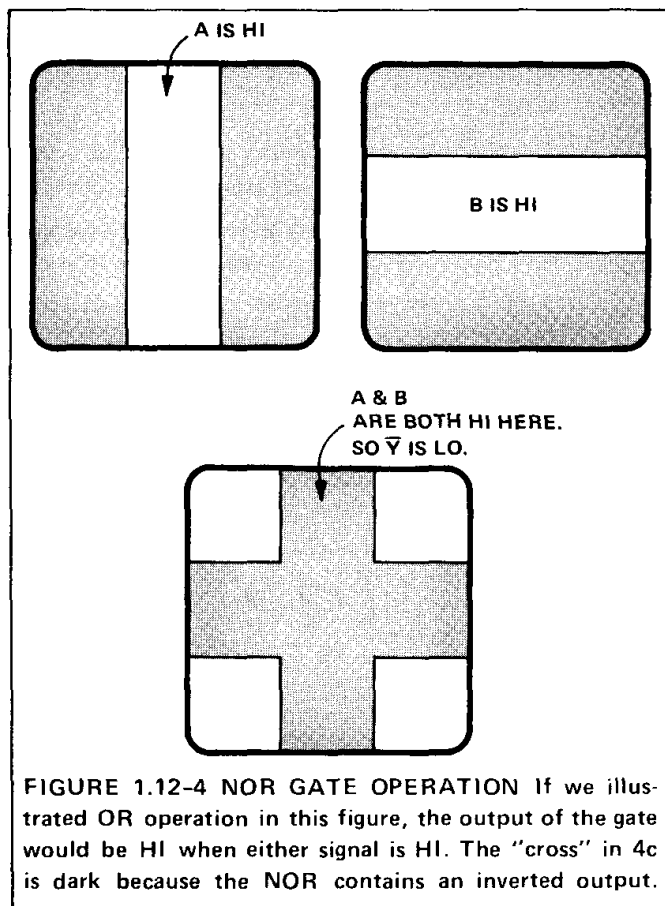
FIGURE 1.12-1 OR & NOR SYMBOLS

The truth table for OR and NOR logic is quite similar to that of the AND and NAND gates except the active output event occurs when either signal is HI.



A	B	Y	$\bar{Y}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

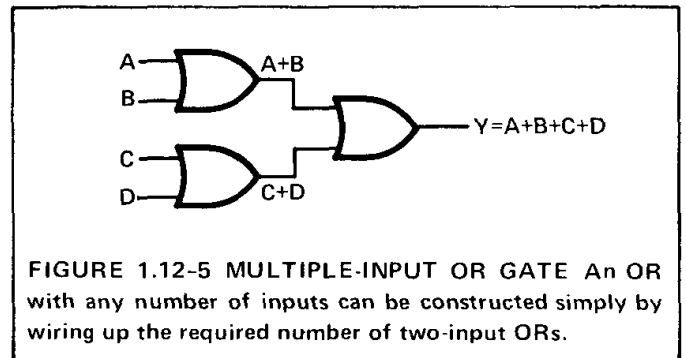
FIGURE 1.12-3 OR & NOR TRUTH TABLE



OR and NOR logic can also be illustrated by the same scheme of light and dark areas which correspond to the times when the signals are HI and LO.

Figures 4a to 4c illustrate the logic of a two-input NOR. Had we used an OR gate instead, Figure 4c would contain a white cross surrounded by a dark field.

Although theoretically important, OR gates are seldom used because active LO logic is far more prevalent. Consequently, many NOR varieties are used to the exclusion of the ORs and IC manufacturers produce more NOR varieties. The only easily available OR gate has but two inputs, whereas NORs can be had with up to five inputs. If it



becomes necessary to have an OR with more than two inputs, it can be constructed simply by wiring together several of the two-input ORs found in a single DIP. For example, Figure 1.12-5 shows how a four input OR is configured.

Now back to our driving game. Let's say we want to liven up the game a bit by adding some trees along the race course in addition to our roadblocks. If we want to subtract a point if the player crashes his car into either a roadblock or a tree, we can build one of two equivalent crash detectors, however note that one is considerably simpler.

### 1.13 Chip Enables

Hopefully, you have absorbed all the foregoing material so we can introduce a new idea which is that of the chip enable. Depending on the exact

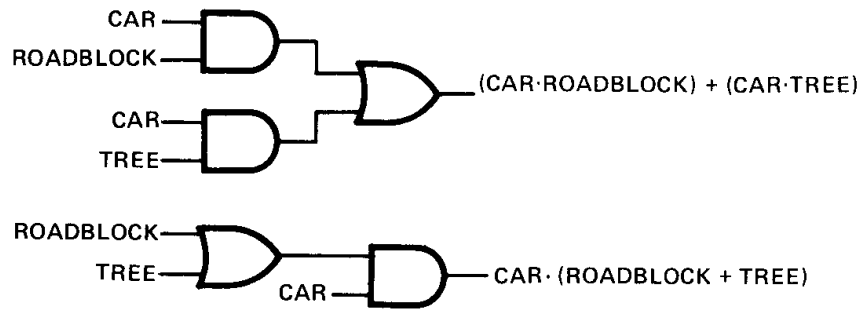


FIGURE 1.12-6 ANOTHER CRASH DETECTOR Both circuits perform equivalent logic functions of detecting when the player has driven into either a roadblock or a tree.

application this function may be called *chip enable*, *strobe* or *chip select*. No matter how it's known, an enable is an input used to turn a device on and off and almost every MSI device has one. However, even a simple four-input AND gate can be drawn with one of its inputs used as an enable.

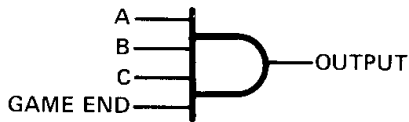


FIGURE 1.13-1 ENABLE FUNCTION The enable in this case is simply an input designated as such.

For example, let's say we have signals A, B and C which we want to turn off when the game is over. To make a simple enable, we assign one of the inputs an enable function. If GAME END drops LO when the game is over, then none of the other inputs will be allowed or enabled through. The use of the enable pin is extremely prevalent when dealing with larger logic circuits as we shall see a bit later in this book.

In the previous example, we simply assigned an enable function to one of the normal inputs. The

following example shows an enable provided by the manufacturer, however it is called a strobe in this case.

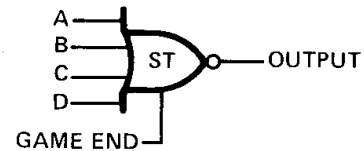


FIGURE 1.13-2 NOR WITH STROBE In this case, an enable is provided by the IC manufacturer and called a "strobe". While this type of enable is more frequently seen in MSI devices, a NOR with strobe is occasionally seen in video games. If a NOR of this variety is found to be malfunctioning, it cannot be replaced by a regular NOR gate.

Without the strobe, the output of this NOR would drop LO anytime one or more of the inputs were to rise HI. Since all the inputs are first NORed and then wired to an internal AND gate, the strobe input can be used to "lock out" the LO output whenever desired.

#### 1.14 The Exclusive OR

Two signals are exclusively ORed when we want the output event to occur if either but not both

input conditions exist. The waveform illustration in Figure 1.14-2 shows that when both inputs are HI, the output drops LO. Otherwise, this type of logic is identical to that of the OR gate.

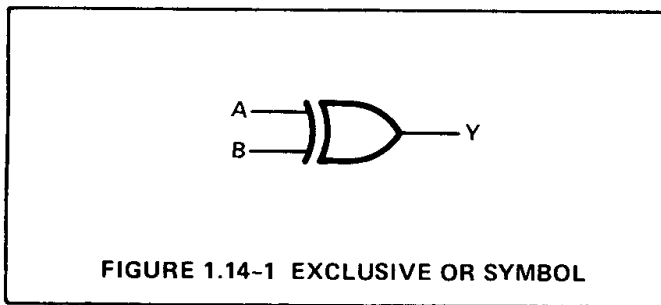


FIGURE 1.14-1 EXCLUSIVE OR SYMBOL

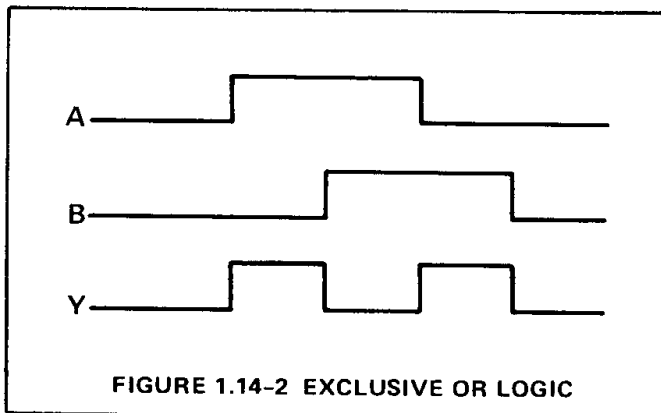


FIGURE 1.14-2 EXCLUSIVE OR LOGIC

The truth table is also quite similar, except when both A and B are HI.

A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

FIGURE 1.14-3 EXCLUSIVE OR TRUTH TABLE

We can also use the same light and dark bar example to illustrate exclusive OR operation.

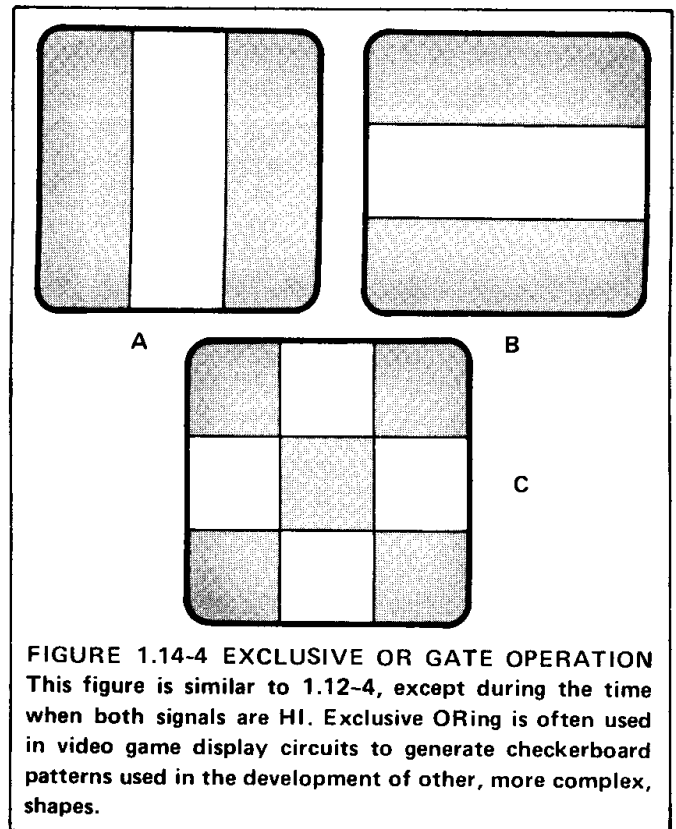


FIGURE 1.14-4 EXCLUSIVE OR GATE OPERATION  
This figure is similar to 1.12-4, except during the time when both signals are HI. Exclusive ORing is often used in video game display circuits to generate checkerboard patterns used in the development of other, more complex, shapes.

You can see that the two HIs at the inputs are passed except when they intersect or, in other words, occur *simultaneously*.

### 1.15 More Complicated Devices

All other digital devices, no matter how complex they may seem initially, are constructed from the basic building blocks already described. The logic characteristics of any device described in the rest of this chapter can be duplicated by assembling the various combinations of AND, OR and NOR gates. To introduce some of these circuits we will do just that. Devices above the level of gates are symbolized by the "black box" symbol which is simply a rectangle containing the proper number of input and output pins. Many of the actual logic diagrams showing the individual gates are included here, however some are not. If you wish to acquaint yourself with these devices on a more intimate level, the manufacturer's data book should

be consulted for almost all IC manufacturers include logic diagrams with their data sheets.

### 1.16 Clock Inputs

Logic systems or elements within a system may be operated either *asynchronously* or *synchronously* depending on the application and the architecture of the system. Asynchronous circuits are generally simpler and used where the sequencing of events is the prime design consideration. The gate applications in the previous pages are excellent examples of asynchronous operation. In these examples, the output event occurred whenever the input conditions were met.

Synchronous operation becomes necessary as system complexity grows and may even be necessary in simple circuits for special applications. Logic elements are synchronized together by the use of a master timing signal known throughout the digital world as **CLOCK**.

Essentially, the clock input is just another form of chip enable. If all the enables of an array of logic elements are connected to the same clock signal, then system events can only proceed when enabled by **CLOCK**. If the operations of the individual elements are all referenced to the frequency of **CLOCK**, each element within the array will also bear a synchronous relationship to every other element. Since the resulting events can be controlled at definite temporal intervals, many kinds of complex operations such as extremely accurate counting are possible.

The frequency of **CLOCK** can also be used as a standard against which measurement can be taken. For example, if a certain event lasts for a known number of clock pulses and the duration of the clock pulse is also known, then the duration of the event can be calculated simply by multiplying the number of clock pulses by the duration of the clock pulse.

All video game computers must operate synchronously. Without extremely accurate timing signals,

coherent TV monitor display would be impossible. The clock is used to provide a standard reference for all logic elements so the electron beam of the monitor can be modulated when it is in the correct part of the CRT and thereby display an image which is correctly positioned. But more about all this later.

Even though all video games operate synchronously, it is important to keep in mind that elements of the system may still be operated on an asynchronous basis. Coin and score circuits are excellent examples of asynchronous operation within a synchronous system. For example, if the score counting process of a game is defined by how many times a race car passes the starting line within a given game length, asynchronous operation may be used. In this case, a signal generated each time the car passes the starting line is used to increment a score counter by connecting it to the clock input.

### 1.17 Functions Of Flip-Flops

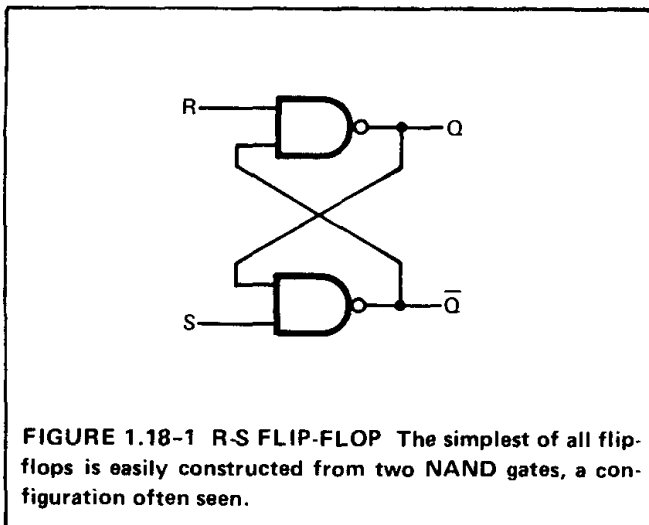
Flip-flops are one of the most important digital building blocks because not only do they have the ability to store information for any length of time desired but they can also be used as frequency dividers. Several different types of flip-flops can be constructed from the basic gates to provide control function variations on the basic theme and suit the flip-flop to a particular application. Flip-flops may be used alone to store information and provide related functions, or they may be combined together in various ways to create other, more complex devices such as counters, latches and shift registers.

The name "flip-flop" derives from the unique operation of the circuit. All flip-flops have two outputs which are known as the **Q** and the  $\overline{Q}$ . When stable in one configuration, the **Q** and  $\overline{Q}$  outputs will be HI and LO respectively, however, as input conditions change, the device "toggles" or "flips" and the **Q** and  $\overline{Q}$  outputs reflect LO and HI.



## 1.18 R-S Flip-Flops

A simple variety of flip-flop, known as the R-S type (reset-set), is often constructed from two of the four two-input NAND gates found in a single package. The R-S type finds application in coin and other asynchronous video game circuits.



R	S	Q	$\bar{Q}$
1	1	1	0
1	0	0	1
1	X	0	1
0	1	1	0
X	1	1	0

**FIGURE 1.18-2 R-S TRUTH TABLE** You can see that, once the device is stable, the outputs will remain the same regardless of what signal S does.

The initial or starting condition for this circuit occurs when the R(reset), S(set) and Q output are all HI. This places a HI at the other input of the set gate and since there are now two HIs here, its output must drop LO. Since this LO is entered at the other input of the reset gate, the output of this gate must remain HI and the flip-flop stabilizes in this configuration. Now, let's say we feed a LO pulse into the S input. Since there is already a HI at the other S gate input, a HI results from the  $\bar{Q}$  output. This HI is connected to the other R gate input and elicits a LO from the R gate since

both inputs are now HI. Therefore, the Q and  $\bar{Q}$  outputs are now LO and HI respectively and the flip-flop is again stable in the toggled configuration. No matter what the signal at S does at this point, the flip-flop will remain stable and continue to output LO and HI from Q and  $\bar{Q}$ . It has therefore "remembered" the last information *regardless* of how S may change. Now, if the reset input should pulse LO, the flip-flop will toggle back the other way. Before R goes LO, there are two HIs at the inputs of the reset gate and its output must be LO. But when R drops LO, the output rises HI and since it is connected to the other set gate input, the  $\bar{Q}$  output returns LO and the flip-flop is again stable. Only now Q and  $\bar{Q}$  are again HI and LO respectively and the flip-flop remains stable in this configuration regardless of how the signal at R may change.

## 1.19 D-Type Flip-Flops

The D-type flip-flop is essentially the same as the previous example, except that two control inputs have been added. These controls are the *preset* and *clear* and both are active LO inputs. Also, we will demonstrate how this device may be used both as a memory element or as a frequency divider.

As long as the preset and clear inputs are held HI, the information at the D input will appear at the Q output (and the inverse will appear at the  $\bar{Q}$ ) *on the rising edge of the next clock pulse*. For example, if D is HI when CLK rises HI, Q and  $\bar{Q}$  will output HI and LO respectively. If D is LO when CLK rises, Q and  $\bar{Q}$  will be LO and HI. Since the information at the D input is both entered into the device and transferred to the outputs on the rising edge of CLOCK, the D-type flip-flop is known as an *edge triggered* device and the significance of this will become more apparent later.

The function of the control inputs is to *force* the outputs to a desired configuration regardless of the state of the conditional inputs. A LO pulse at the preset forces Q HI and  $\bar{Q}$  LO. A LO pulse at the clear input forces Q LO and  $\bar{Q}$  HI. The clear control overrides all other inputs.

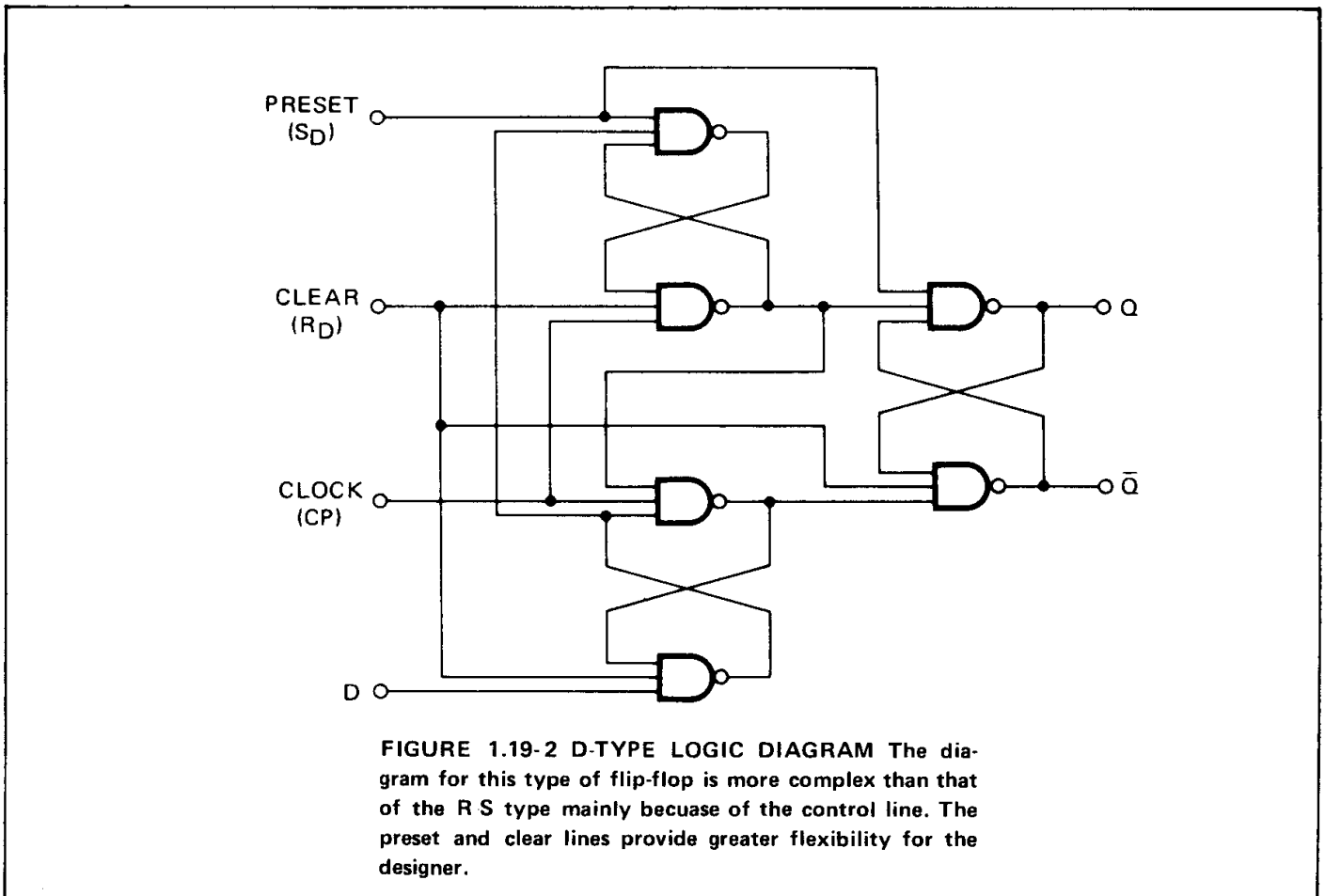
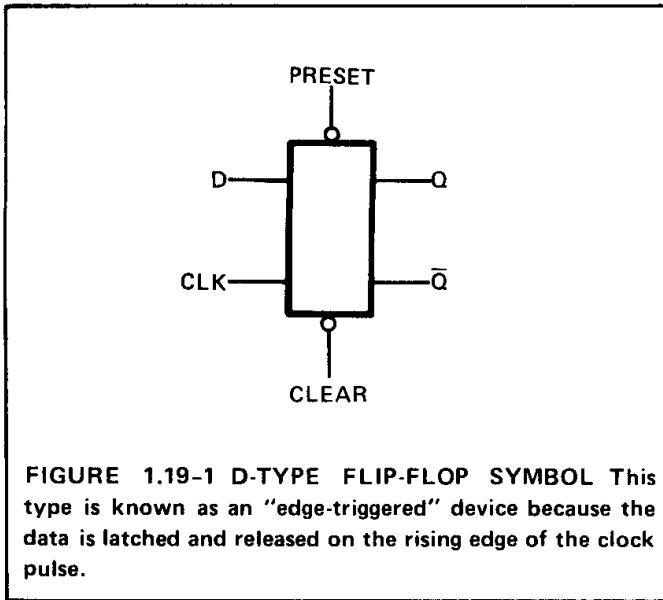
Any D-type flip-flop may be wired into the “toggle” configuration by connecting the  $\bar{Q}$  output back to the D input. This will enter the logic state opposite to the previous one at the D input each

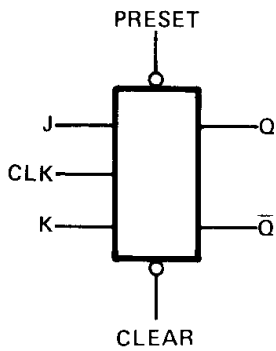
time the clock pulses so the outputs will toggle every time the device is clocked.

### 1.20 J-K Flip-Flops

The J-K type may also be used in either the memory or toggle mode, however it is not an edge triggered device. In this case, the information is entered on the rising edge of CLOCK but not released until the *falling edge*, hence the J-K variety is often called a *master-slave* flip-flop.

The J and K inputs must be at *opposite logic levels* for the device to be used in the memory mode. In this case, the Q and  $\bar{Q}$  outputs will reflect those levels when the next clock pulse is received. If J and K are both LO, the *outputs will not change*. If J and K are both held HI, the flip-flop will operate in the *toggle mode* and the outputs will flip after every clock pulse. The preset and clear inputs have the same functions in th J-K flip-flop as in the D-type.





**FIGURE 1.20-1 J-K FLIP-FLOP SYMBOL** The J-K flip-flop is called a "master-slave" device because data is latched on the rising edge of CLOCK, but not released until the falling edge.

### 1.21 MSI Devices

So far, all the devices discussed have been relatively simple basic building blocks and excellent examples of the SSI level of integration. Since the functions these circuits provide are relatively simple, few input and output pins are required so several may be stuffed into a single package. The following MSI devices, although constructed from gates and flip-flops, produce more complicated functions, hence multiple control, input and output pins are required.

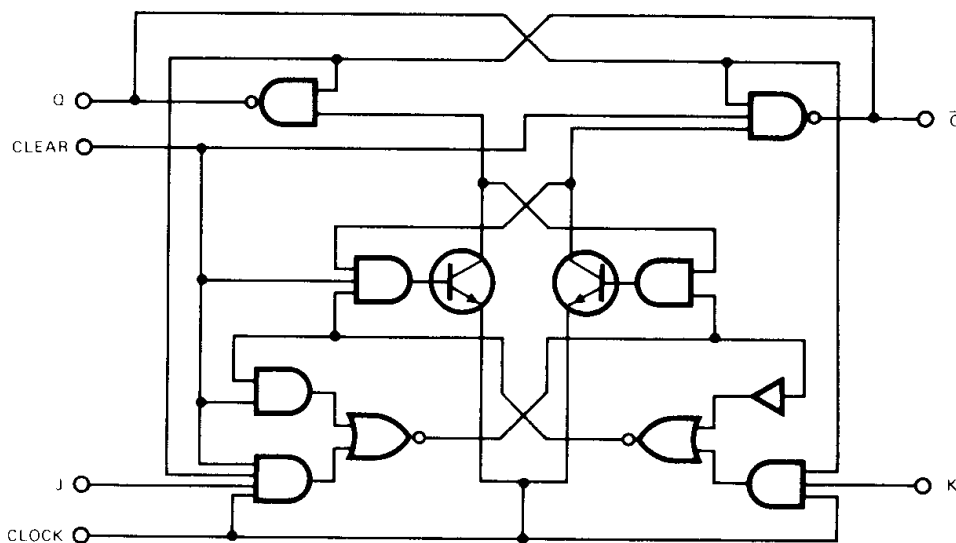
### 1.22 Counters

Back when things were still fairly primitive in the

digital world, counters were actually constructed by wiring up a series of flip-flops. Now, of course, a great variety of integrated counters are available with a wide choice of control capability and method of operation. Obviously, counting is one of the most important functions of digital circuitry and this fact is evidenced by their widespread use in video games.

Counters operate by expressing binary numbers in terms the circuitry can understand, namely HIs and LOs. To read out the binary number 9, the counter's outputs must be 1 0 0 1 mathematically speaking or HI LO LO HI electrically. Aside from MOS circuits, all normally available counters are 4-bit devices which means they can count from 0 0 0 0 (decimal zero) to 1 1 1 1 (decimal 15). Although a 4-bit counter can only reach a terminal count of 15, it performs sixteen actual counting operations since it begins at zero. If more than four bits are needed for a particular application such as counting to 256, additional bits are easily added forming a *chain* of the desired length. Since all integrated counters are constructed from flip-flops, extra bits are supplied by tacking on one or two more flip-flops or even another counter.

Counters are constructed according to one of two basic architectural types. *Synchronous counters* (i. e. 9316s) are built from D-type flip-flops where



**FIGURE 1.20-2 J-K LOGIC DIAGRAM**

the internal CLK inputs are all tied to an external CLOCK so their outputs all change states *simultaneously*. Asynchronous counters such as the 7493 are constructed from J-K type flip-flops where one flip-flop toggles the next one down the line. Hence, there is room for propagation delay between stages which can stack up in a longer chain and cause significant problems in a fast system. Although the internal construction of the counter is either synchronous or asynchronous, either type may be used within a synchronous or asynchronous system. For example, the 7493 is an asynchronous counter, however a number of 7493s may be wired up in a chain where all the counters are tied to the same clock. The result is an asynchronous device operation within a synchronous system architecture.

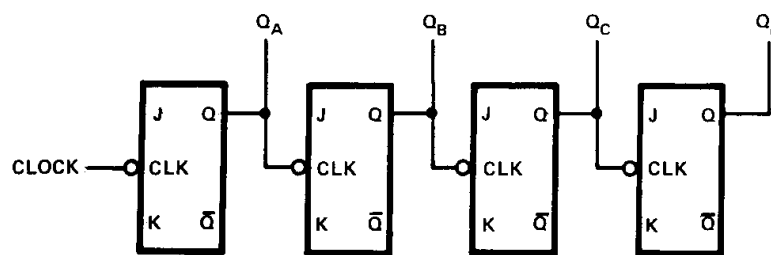
Also, keep in mind that counters are available in both a straight binary version which counts from 0 to 15 or in a BCD (Binary Coded Decimal) configuration which counts from 0 to 9 before being reset.

### 1.23 Constructing A Simple 4-Bit Asynchronous Counter

Let's build a simple 4-bit asynchronous counter so you can see how really basic their operation is. First, we take four J-K flip-flops and wire the CLK input of the first to CLOCK. Then we take the Q output of each flip-flop and wire it to the CLK input of the next one in line so that it looks like the chain in Figure 1.23-1.

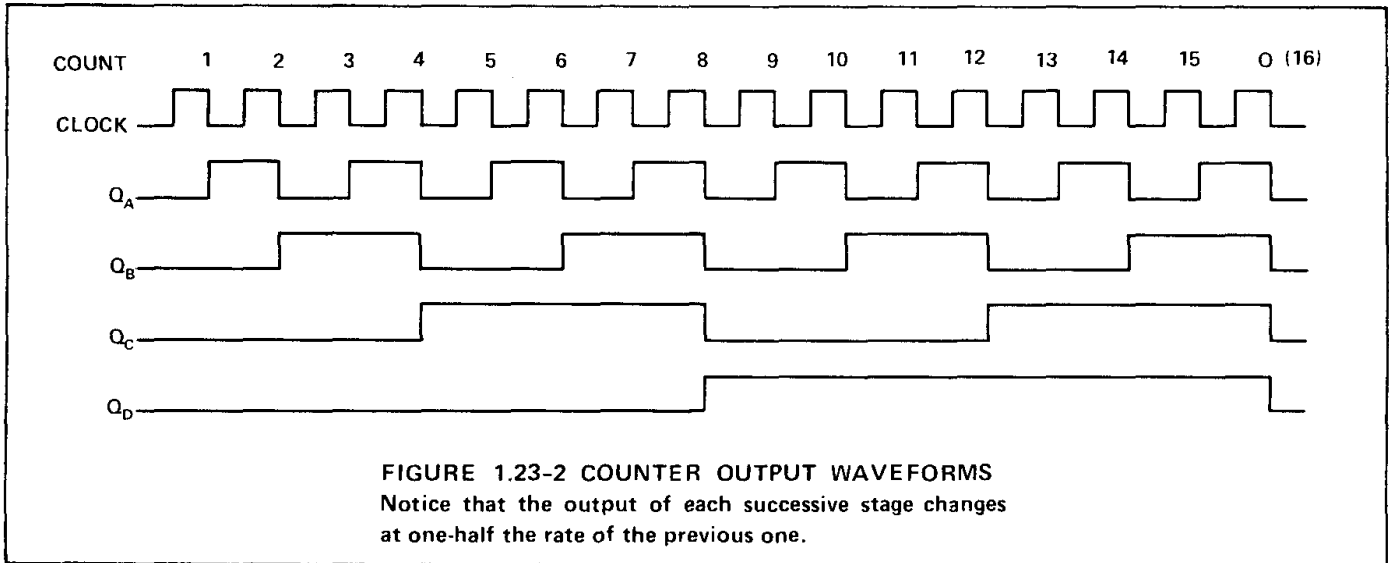
Before the counter begins, all its outputs are LO and the resulting number is 0 0 0 0. On the falling edge of the first clock pulse, the Q output of the first flip-flop rises HI and remains HI until the next falling edge of CLOCK so the number 1 0 0 0 (decimal one-the LSB is the leftmost bit) is generated. At this point, the falling edge of the next clock pulse causes the  $Q_A$  output to drop LO and the falling edge of this signal clocks the next flip-flop causing its Q output to rise HI. This results in the number 0 1 0 0 or decimal two at the outputs. On the third clock pulse, both the  $Q_A$  and the  $Q_B$  outputs are HI so the number 3 (1 1 0 0) is developed.  $Q_A$  and  $Q_B$  both return LO on the falling edge of the fourth clock pulse so the falling edge of the signal from  $Q_B$  causes the Q output of the third flip-flop ( $Q_C$ ) to rise HI and produce the number 0 0 1 0 (4).  $Q_C$  changes every fourth clock pulse or count and since  $Q_D$  divides this frequency in half,  $Q_D$  will change every eighth count. On the 15th count, all the outputs are HI producing the number 1 1 1 1 or 15. On the next or 16th falling edge of CLOCK, all the outputs return to LO to produce the number zero again.

In the figure above, the top row of numbers corresponds to the actual number the device is producing where each number occurs on the falling edge of the clock pulse. Also notice that although the last count is numbered zero for all the outputs are LO, it is actually equivalent to a count of 16. However, the most important idea to understand from this figure is that the frequency of clock is continuously being divided in half by successive stages of flip-flops.



**FIGURE 1.23-1 ASYNCHRONOUS COUNTER CHAIN**  
Since each successive stage is clocked by the previous out-

put, all the outputs do not change simultaneously. This factor may cause problems in high speed circuits.

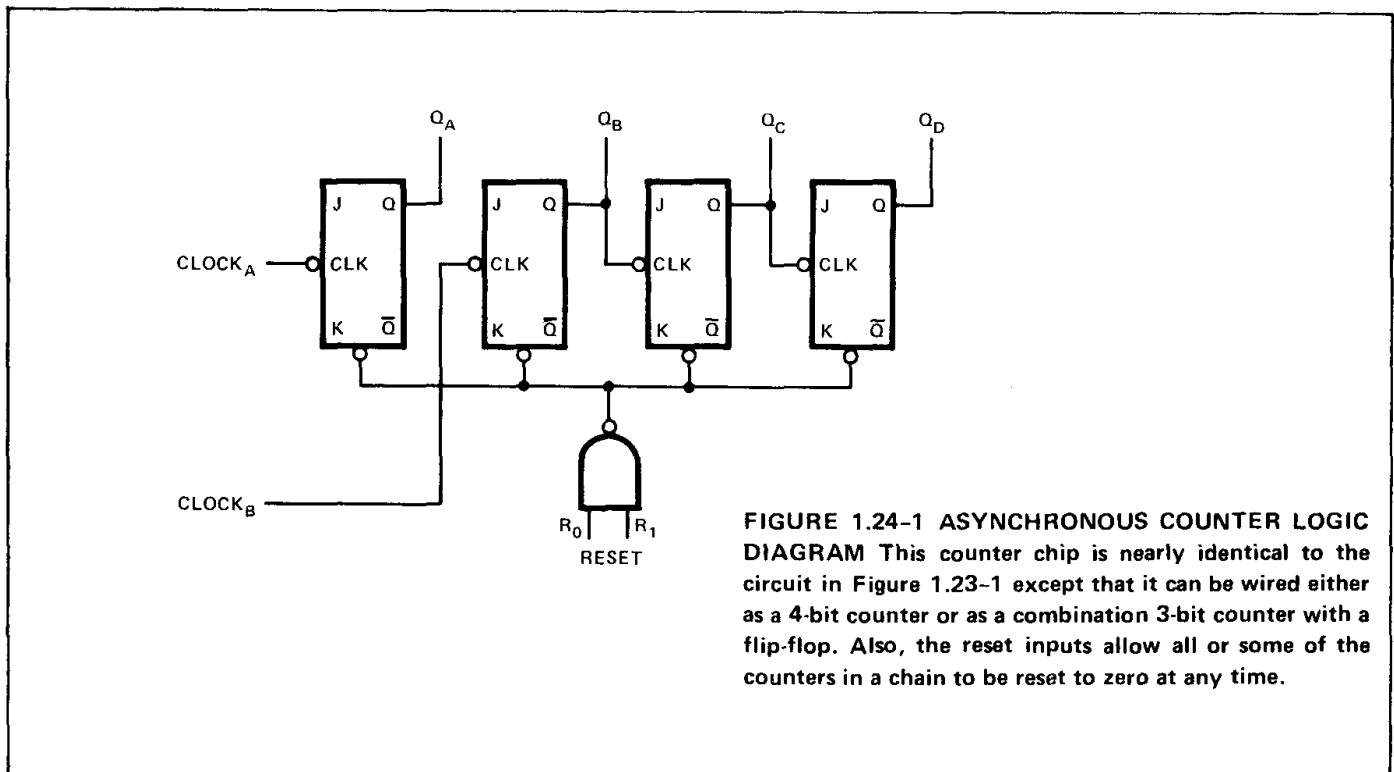


Now you see how counter chains are constructed. If we needed to count up to 32, another flip-flop could simply be tacked on to our four-bit chain. Or, if an eight-bit chain is required, two integrated counter chips are simply wired together. Also, you should be able to more clearly see how propagation delay develops. Since the clock for each stage must come from the previous flip-flop, the total amount of propagation delay is equal to the number of bits multiplied by the delay for a flip-flop

(between 10 and 20 ns.).

#### 1.24 7493 Four Bit Binary Asynchronous Counter

This 4-bit counter finds its major video game application in sync chains and it is constructed *almost* identically to Figure 1.23-1. Figure 1.24-1 is the logic diagram for the 7493 and it shows two minor additions. First, notice that two clock inputs are



provided so the device may be used as a divide-by-two and a divide-by-eight simultaneously. However, the counter is normally wired as a divide-by-sixteen by connecting pin 1 ( $CP_A$ ) to pin 14 ( $CP_B$ ). Also, the internal flip-flops are provided with clear inputs so the counter may be reset before it reaches the 16th count. When the reset input drops LO, all the outputs are reset to 0 0 0 0. Two reset pins are provided (and internally NANDed) so that counters in a chain may be selectively reset. Normally, both reset pins are tied together to a single reset input line. The 7492 is a very similar counter, except that it is a simultaneously divide-by-two and divide-by-six which can be wired as a divide-by-twelve.

### 1.25 Synchronous Counters

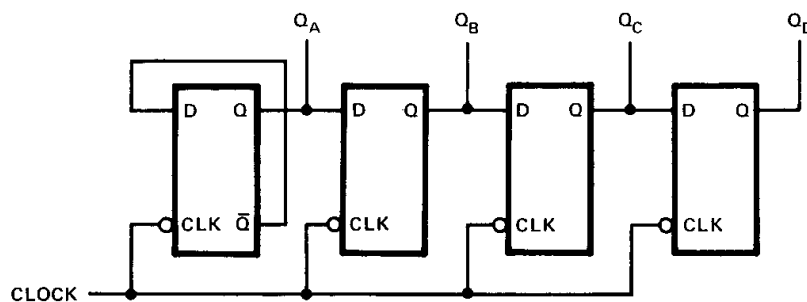
Synchronous counters are also quite easily assembled from flip-flops, except in this case a chain of D-type flip-flops is used so that all elements of the chain are clocked simultaneously. If you will remember the section on D-type flip-flops, you will recall that the information is entered and released on the rising edge of the clock pulse whereas the J-K type enters the data on the rising edge and clocks it out on the falling edge.

Since all the flip-flops in Figure 1.25-1 are tied to clock, the  $Q_A$ - $Q_D$  outputs must all change simultaneously. As a result, this type of counter may be used in high speed applications without encountering annoying propagation delay problems.

### 1.26 Presettable Counters

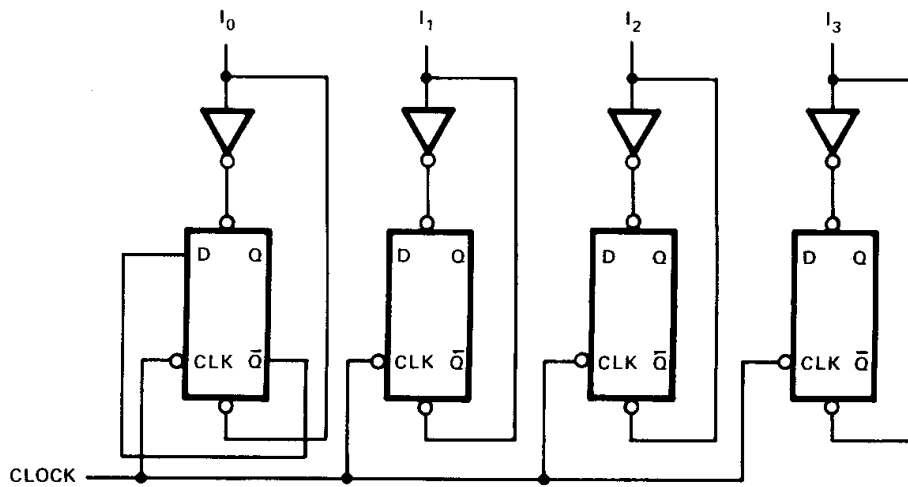
Synchronously presettable counters are the key to the "slipping counter" motion chains which enable video game computers to move images about the monitor CRT. Although presettable counters are available in both synchronous and asynchronous versions, the synchronous varieties are more widely used to keep glitches at a minimum.

Notice that the flip-flops in Figure 1.26-1 are equipped with both preset and clear inputs and that an inverter is placed between the parallel load inputs ( $I_0$ - $I_3$ ) and the preset input. Let's say we want our counter to start counting from any number other than zero, but less than fifteen. To accomplish this, all we do is enter the binary equivalent at the parallel inputs and the chain must start from that number. For example, if a 1 is entered at  $I_0$ , the preset will receive a LO input and that flip-flop will output a 1 from its Q output. The



**FIGURE 1.25-1 SYNCHRONOUS COUNTER CHAIN**

Since this type of chain uses edge triggered flip-flops where all the clock inputs are tied to the same signal, all the outputs change simultaneously.



**FIGURE 1.26-1 SYNCHRONOUSLY PRESETTABLE COUNTER CHAIN** This chain is very similar to the one in Figure 1.25-1 except that the count can be started from any number input into the  $I_0 - I_3$  terminals. Synchronously presettable counters are used extensively in motion chains.

reason for the inverters is to assure that the preset and clear inputs receive complementary signals. Therefore, the clear input of the first flip-flop will receive a HI and will not be cleared. So, to enter a complete number, a four-bit binary code is dropped into all the parallel inputs and only then is that counter enabled to count.

### 1.27 9316 Synchronously Presettable Four Bit Counter

This device is similar to the diagram in Figure 1.26-1, except that the actual logic diagram is considerably more complex since we did not include the outputs and a number of the control inputs. However, the operation of the device is still relatively simple. The preset number is loaded when  $\overline{PE}$  is LO and the counter begins counting up from that number when  $\overline{PE}$  returns HI. When the number is loaded, it will appear immediately at the parallel outputs. Generally speaking both CEP and CET are tied HI permanently. In any case, they must both be HI before the counter will begin to count.

PIN NAMES	
$\overline{PE}$	Parallel Enable (Active LOW) Input
$P_0, P_1, P_2, P_3$	Parallel Inputs
CEP	Count Enable Parallel Input
CET	Count Enable Trickle Input
CP	Clock (Active HIGH Going Edge) Input
$\overline{MR}$	Master Reset (Active LOW) Input
$Q_0, Q_1, Q_2, Q_3$	Parallel Outputs
TC	Terminal Count Outputs

**FIGURE 1.27-1 PIN NAMES**

MODE SELECTION			
$\overline{PE}$	CEP	CET	MODE
L	L	L	Preset
L	L	H	Preset
L	H	L	Preset
L	H	H	Preset
H	L	L	No Change
H	L	H	No Change
H	H	L	No Change
H	H	H	Count

( $\overline{MR} = \text{HIGH}$ )

**FIGURE 1.27-2 MODE SELECTION CHART**

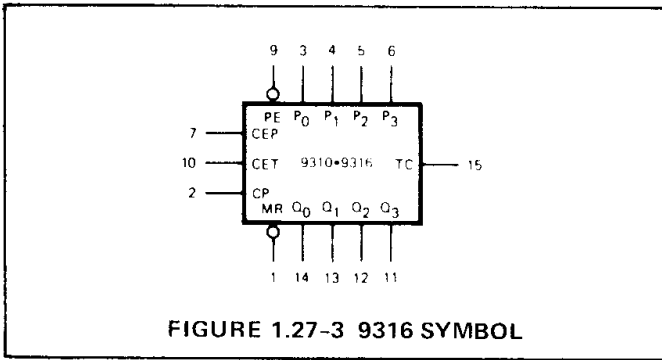


FIGURE 1.27-3 9316 SYMBOL

The master reset line ( $\overline{MR}$ ) is also an active LO input. It must be HI for the counter to count and when a LO pulse is received on this line the device

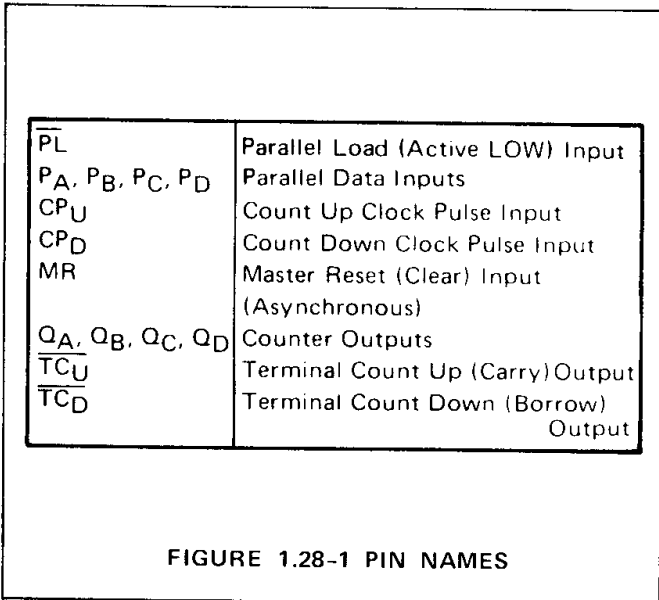


FIGURE 1.28-1 PIN NAMES

is completely reset. The terminal count (TC) pin is also known as the carry output and it is used to clock the next counter in a chain. This type of counter is also available in a BCD version (the 9310), however this device is used infrequently in video games.

### 1.28 74193 Presettable Up/Down Counter

This counter is also presettable, however the number is loaded asynchronously. In addition, it can count *up or down* from the preset number. The direction of counting is controlled pulsing one clock input while holding the other HI. For example, to force the counter to count up the  $CP_U$  input is clocked while  $CP_D$  is held HI. The preset number is loaded while  $\overline{PL}$  is held LO but the counter cannot begin until it is returned HI. Notice that this device has two terminal count out-

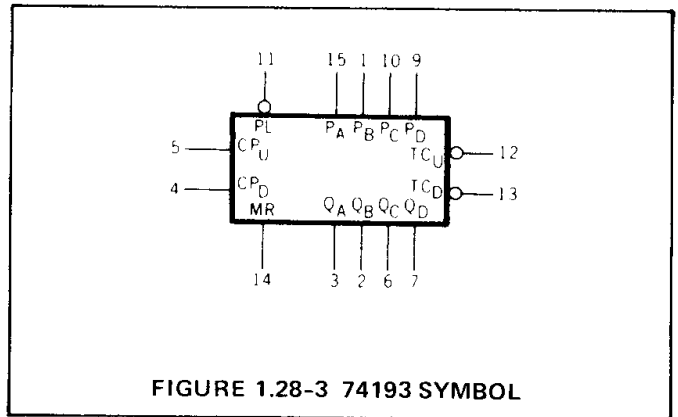


FIGURE 1.28-3 74193 SYMBOL

#### MODE SELECTION

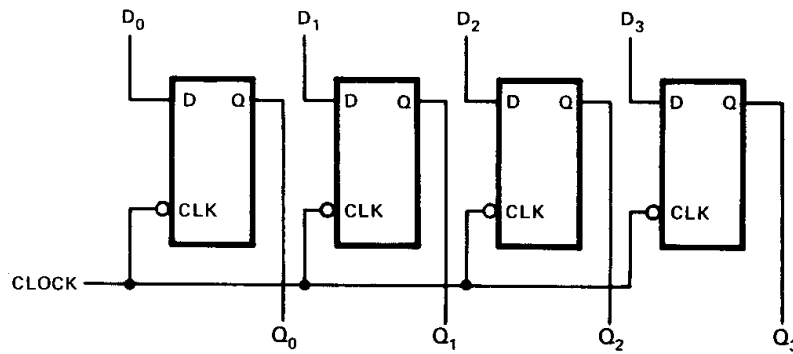
MR	$\overline{PL}$	$CP_U$	$CP_D$	MODE
H	X	X	X	Preset (Asyn.)
L	L	X	X	Preset (Asyn.)
L	H	H	H	No Change
L	H	CP	H	Count Up
L	H	H	CP	Count Down

H = HIGH Voltage Level  
L = LOW Voltage Level

X = Don't Care Condition  
CP = Clock Pulse

FIGURE 1.28-2 MODE SELECTION CHART





**FIGURE 1.29-1 QUAD LATCH LOGIC DIAGRAM**  
 This quad latch consists of four single-bit latches used to store a 4-bit binary number.

puts ( $TC_U$  and  $TC_D$ ). This is necessary so the next counter in the chain will have the correct clock input (up or down). The 74193 is the BCD version of this device.

### 1.29 9314 Quad Latch

Counters are not the only devices constructed from flip-flops. Latches, shift registers, serial-to-parallel converters, etc. are all built from the basic flip-flops in much the same way as counters.

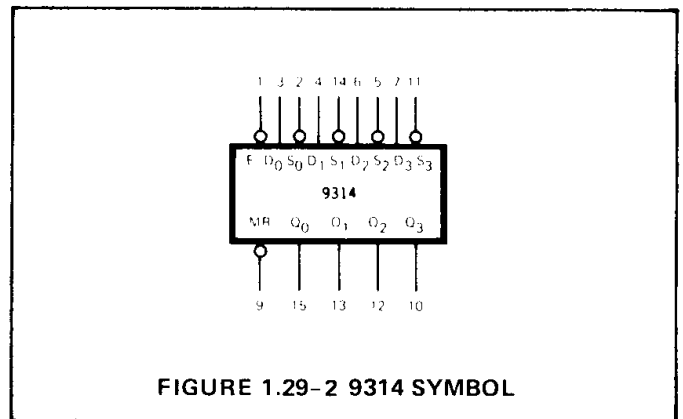
The quad latch is used to store up to four bits of data and so it can retain a number as large as 1 1 1 1. In reality, it consists of four single-bit latches connected by a common clock. The simplified logic diagram in Figure 1.29-1 shows how data is entered at the  $D_0$ - $D_3$  inputs and released by the clock pulse or enable.

The actual logic diagram for the 9314 is more complicated as there are some additional control lines. First of all, the master reset line ( $\overline{MR}$ ) overrides all other input conditions and forces the outputs LO when it receives a LO pulse. Also, this device may be connected in either the D-type or R-S mode. In the D mode, the  $\overline{S}$  inputs are held LO and the latch outputs reflect the D inputs when the enable is active. If the enable is LO and if the latch is in the R-S mode, the device is reset by a LO input and can be set by a LO on the  $\overline{S}$  line if the D input returns HI. If both  $\overline{S}$  and D are LO,

the D input will dominate and the latch will be reset. When the enable goes HI, the latch remains in the last state prior to the LO to HI transition.

### 1.30 Shift Registers

Often data must be manipulated so it appears in a usable format and occurs at the proper time before it can be entered into another device. If the other



**FIGURE 1.29-2 9314 SYMBOL**

$\overline{E}$	(Active LOW) Enable Input
$D_0, D_1, D_2, D_3$	Data Inputs
$\overline{S}_0, \overline{S}_1, \overline{S}_2, \overline{S}_3$	Set (Active LOW) Inputs
$\overline{MR}$	Master Reset (Active LOW)
$Q_0, Q_1, Q_2, Q_3$	Latch Outputs

**FIGURE 1.29-3 9314 PIN NAMES**

MR	E	D	S	$Q_n$	OPERATION
H	L	L	L	L	D MODE
H	L	H	L	H	
H	H	X	X	$Q_{n-1}$	
H	L	L	L	L	R/S MODE
H	L	H	L	H	
H	L	L	H	L	
H	L	H	H	$Q_{n-1}$	
H	H	X	X	$Q_{n-1}$	
L	X	X	X	L	RESET

X= Don't Care     $Q_{n-1}$  = Previous Output State  
 L= LO Voltage  
 H= HI Voltage     $Q_n$  = Present Output State

FIGURE 1.29-4 TRUTH TABLE

device must complete a task before data can be read into it, a serial-in/serial-out shift register is used to hold the information until the right time arrives. Other variations of shift registers are used to convert parallel information (data occurring simultaneously on a number of output lines) to a serial (sequentially occurring information on a single line) format or visa versa.

For example, many games use various types of memories to hold information which appears from the memory as an eight-bit word occurring in parallel form from all eight output lines. This format

is perfect for other parallel-loading devices but it is not compatible with the operation of the TV monitor. Since the electron beam of the monitor sweeps the screen and generates images by illuminating a series of dots, a burst of simultaneously occurring information would appear as a blotch on the CRT. However, if this information is converted to a serial format, the electron beam can illuminate one point at a time and form a coherent image.

### 1.31 Serial-In/Serial-Out Shift Register

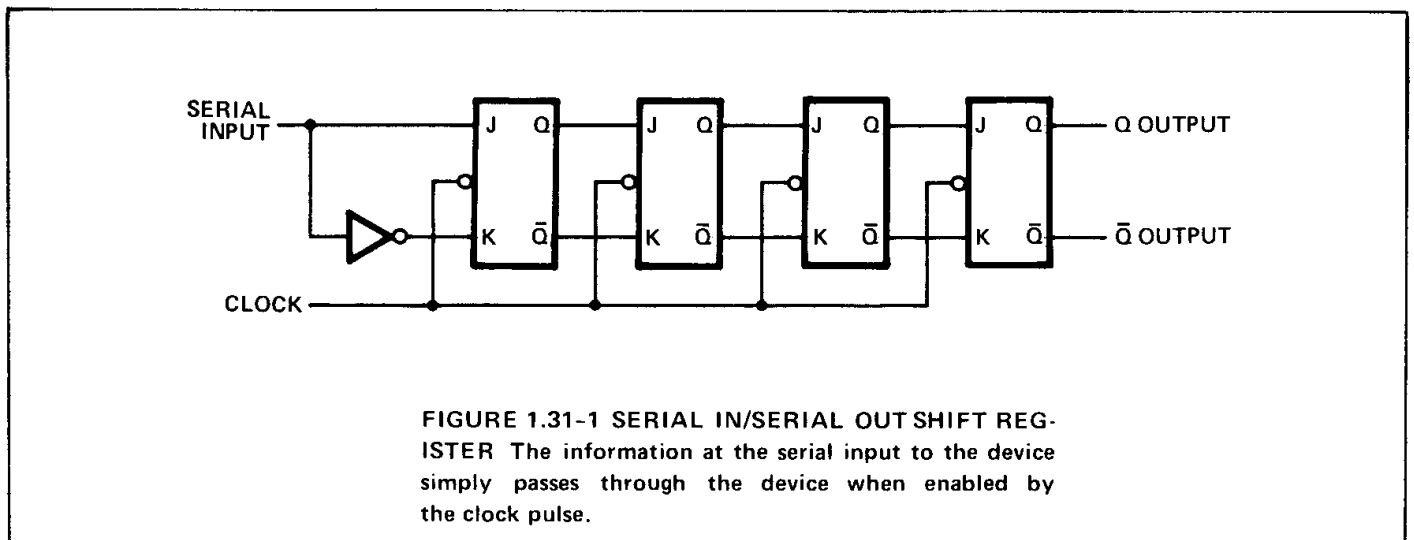
This is probably the simplest variety which consists of four master-slave flip-flops wired so the information simply passes through the device when enabled. The internal inverter provides the complement of the input data so the K input will be at the opposite state.

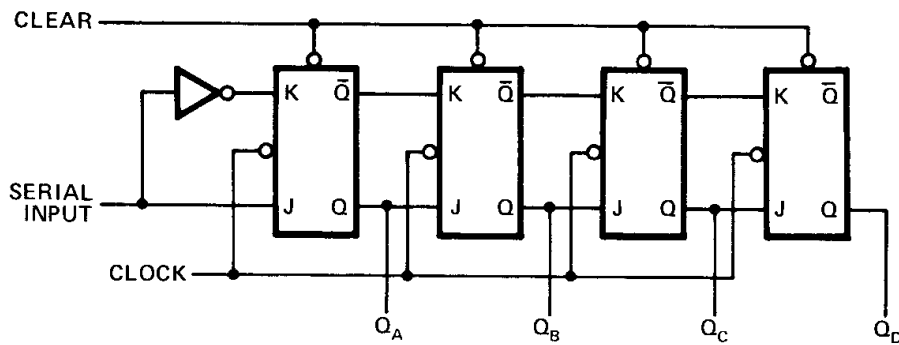
### 1.32 Serial To Parallel Converter

As you can see from Figure 1.32-1, this device is nearly identical to the serial-in/serial-out shift register except that the Q outputs have been brought out and the flip-flops are all wired to a clear input which forces the parallel outputs LO.

### 1.33 Parallel To Serial Converter

While the circuit below may look complicated, it is very simple — only a bit cluttered with control inputs. The data is loaded at the parallel inputs

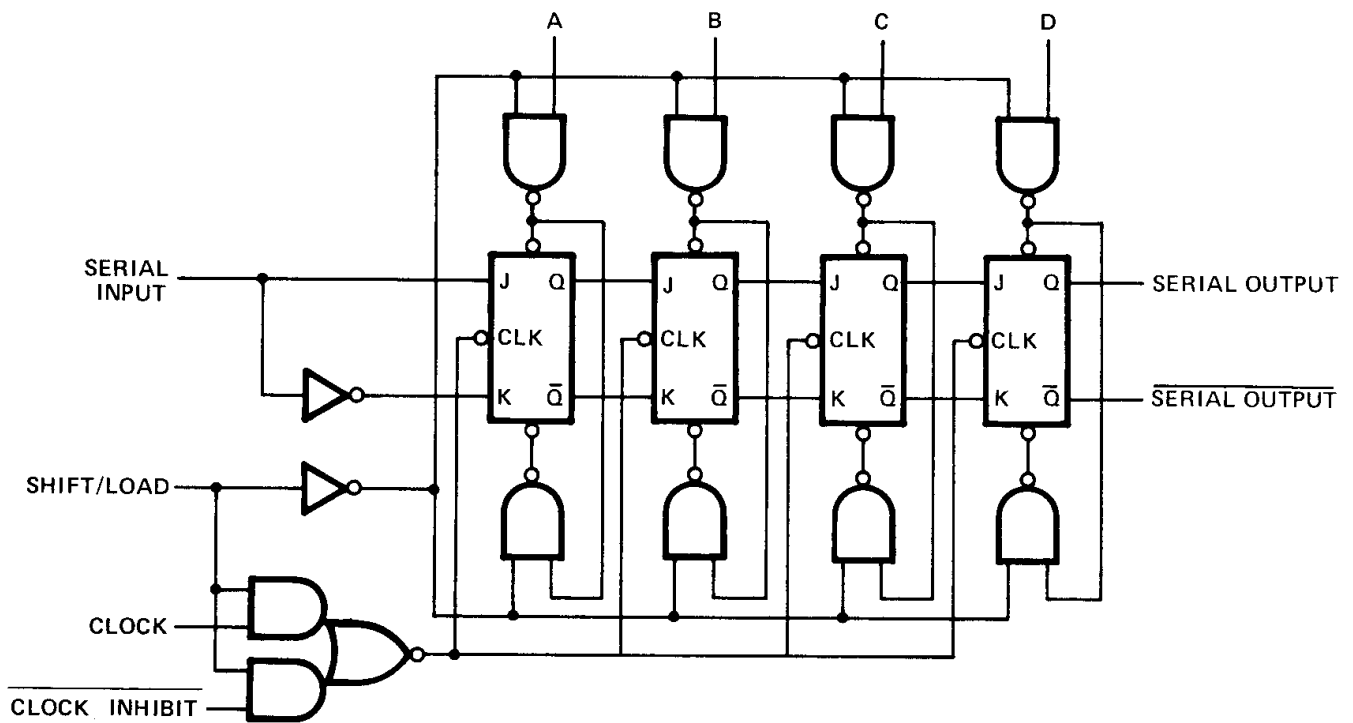




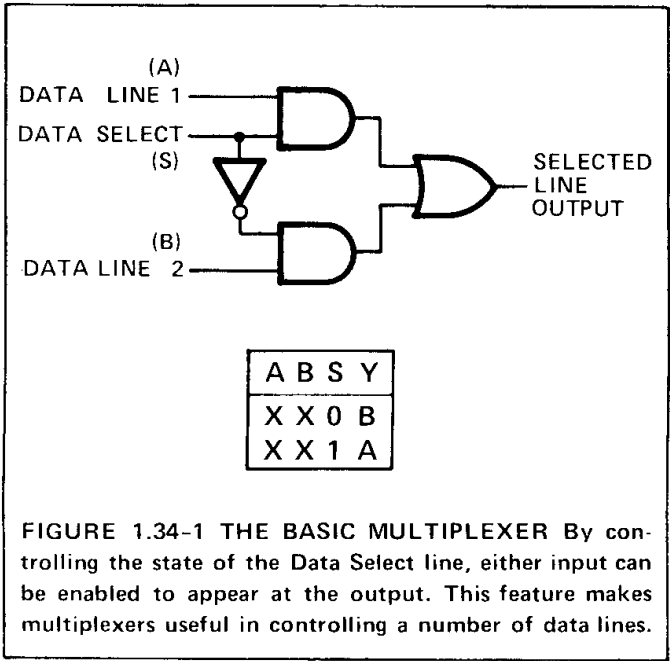
**FIGURE 1.32-1 SERIAL TO PARALLEL CONVERTER** This device takes a serial or sequentially-ordered data stream and converts it to a parallel or simultaneous format.

(A-D) independently of the clock but only when the top row of NAND gates is enabled by a LO at the shift/load input. As long as CLOCK INHIBIT

is held HI, the information remains latched within the register. However, when this input drops LO, the clock reads the information out serially. If a



**FIGURE 1.33-1 PARALLEL TO SERIAL CONVERTER** This device provides both the serial in/serial out function as well as parallel to serial conversion. The parallel data can be loaded only during the time the Shift/Load control is LO.



devices for further processing. For example, the score signals for two players are often multiplexed together so only one score display circuit is necessary. Multiplexing both score signals allows the display circuit to work on one player's data first and generates the numbers for his score and then deals with the other player's information. The applications of multiplexers in video games is almost limitless. They are used to select certain areas of the CRT of parts of an image and extensively used in the address of memories to retrieve information.

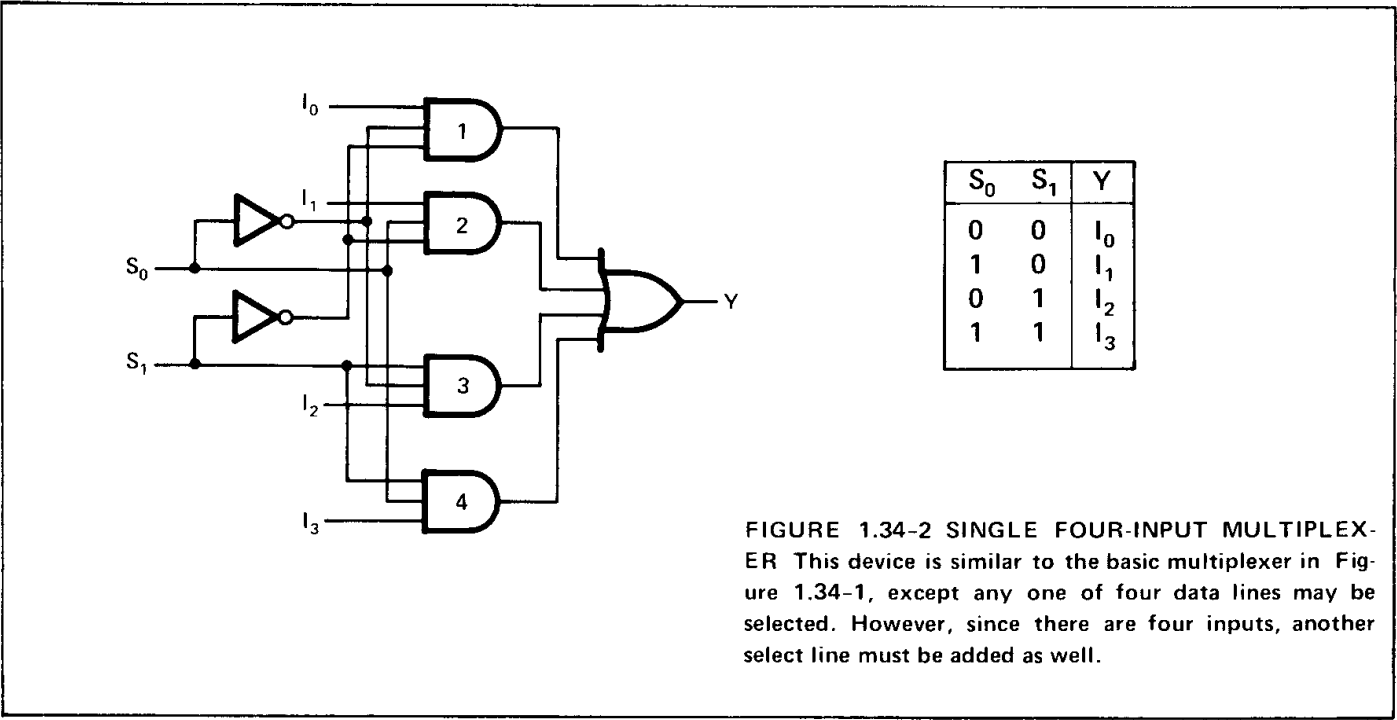
Unlike counters and shift registers, there is only one basic type of multiplexer *cell* or circuit which can be enlarged to accommodate up to eight or more inputs. But first, let's illustrate the basic multiplexer in its simplest form.

longer word needs conversion, two shift registers can be used simply by connecting the serial output of one to the serial input of the next.

**1.34 Multiplexers**

When manipulating more than one data line, it is often necessary to *select* some lines from a group so the desired information can be passed to other

Figure 1.34-1 shows the simplest multiplexer, the single two-input. The input data lines A and B are selected by the state of signal S. When S is HI, the top AND gate is enabled and it passes whatever signal is at the data input. Simultaneously, the HI at S is being inverted LO to disable the lower gate and lock out the other data line. But if S drops LO, the top gate is disabled and the second data line is enabled through to the output.



A single select line can only be in one of two states: HI or LO, hence it can select only one of two signals. But if we have four signals from which we need to choose one, a second select line is required. Two selects then have the capability of four states: 0 0, 0 1, 1 0 and 1 1 where each different code can be used to select one of four data lines. Figure 1.34-2 shows how the simplest four-input multiplexer is constructed and notice that there are two selects,  $S_0$  and  $S_1$  which control four inputs ( $I_0-I_3$ ).

If both  $S_0$  and  $S_1$  are LO, Gates 2, 3 and 4 are disabled since a LO  $S_0$  disables Gates 2 and 4 and a LO  $S_1$  disables Gates 3 and 4. Gate 1 is receiving two HIs because the LOs at  $S_0$  and  $S_1$  are inverted first before being connected to it. Consequently, Gate 1 is enabled and it passes whatever signal is at  $I_0$ . The second count occurs when  $S_0$  and  $S_1$  are HI and LO respectively which enables Gate 2 to pass input  $I_1$  and so on.

Actual integrated four-input multiplexer chips such as the 74153 are a bit different. First of all, there are two complete multiplexers per 16-pin DIP. Also, an additional enable input is provided which is connected to all the gates so that the entire chip may be turned off. Generally, this enable input is known as the strobe and it may be used to turn off some of the multiplexers in a circuit while the others are allowed to continue.

### 1.35 Decoders

Although decoders have ancillary functions such as providing seven-segment code, their primary function is as reverse multiplexers. Whereas multiplexers take a number of input lines and route the desired one to a single output, the decoder takes a single input and routes it to one of a number of outputs.

The figure above illustrates the simplest decoder configuration where A is the address line and E is the input. If A is LO, Gate 2 is disabled but since A is also inverted, Gate 1 is enabled and passes E through output 1.

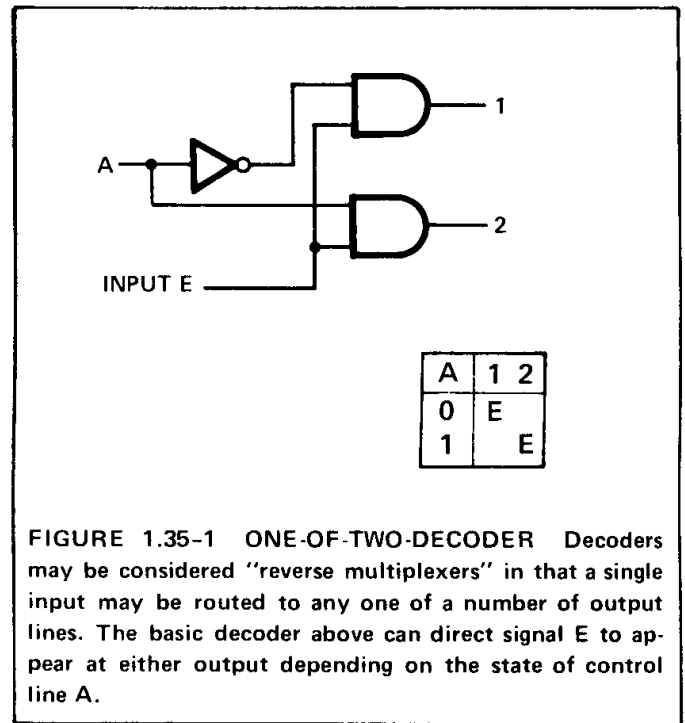


FIGURE 1.35-1 ONE-OF-TWO-DECODER Decoders may be considered "reverse multiplexers" in that a single input may be routed to any one of a number of output lines. The basic decoder above can direct signal E to appear at either output depending on the state of control line A.

Generally, more elaborate decoders are necessary to decode one of four, eight or even sixteen outputs. Figure 1.35-2 shows how a one-of-four decoder is constructed and notice that it must have an additional address bit just like the four-to-one multiplexer. For instance, if both address bits  $A_0$  and  $A_1$  are LO, all but the fourth gate are disabled.

The combination of a multiplexer and a decoder is very useful for transmitting data over long distances. To make it simple, let's say we have two different sets of data which need to be transmitted over a single telephone line. All we need to do essentially is to multiplex both signals together at the transmit end and decode them at the receive end. However, provisions do have to be made so the address for both the multiplex and decode is the same and this can be done by sending additional information along the same line.

### 1.36 Other Decoder Applications

Back in the olden days before the advent of LEDs, nixie tubes were used to read out numbers in various types of equipment. The nixie tube contained 10 different neon numbers (0 - 9) where one was illuminated at a time. If the number nine was

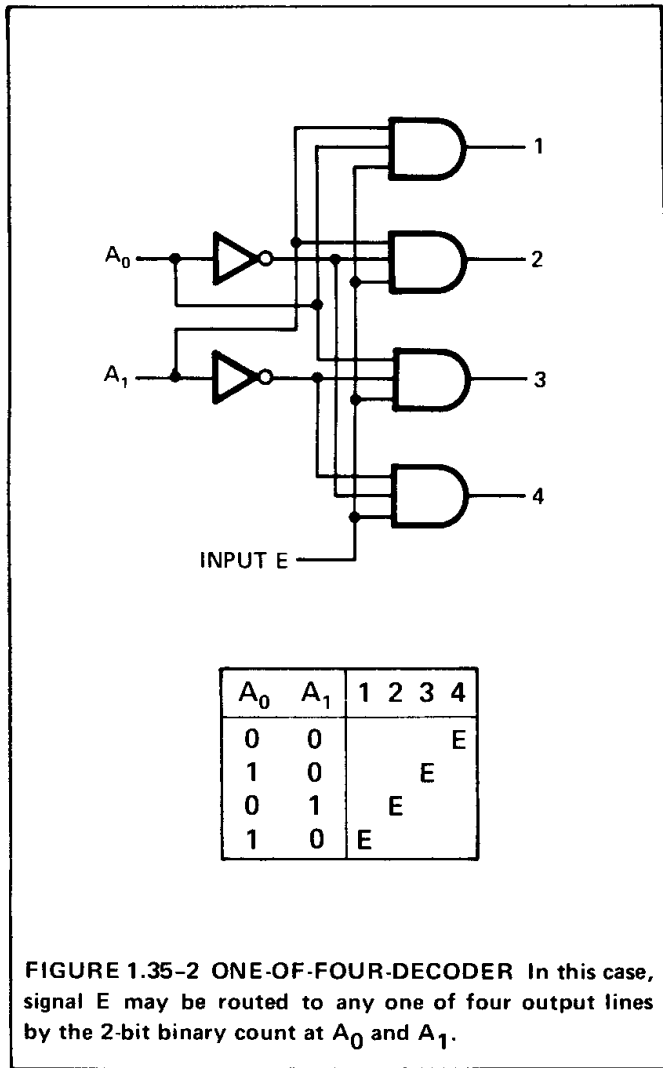


FIGURE 1.35-2 ONE-OF-FOUR-DECODER In this case, signal E may be routed to any one of four output lines by the 2-bit binary count at A<sub>0</sub> and A<sub>1</sub>.

needed, power was entered into the 9 pin to light the neon number. However, the computer doesn't output the decimal number 9 for it can only talk in binary. So a circuit was needed to convert the BCD number 1 0 0 1 to the decimal equivalent of 9. This process is called BCD to decimal decoding and this function is now available in a single IC. In fact, decoders are available which also contain the lamp drivers thereby eliminating the need for a bunch of external transistors to power the lamps.

But in video games, the most common need for this type of conversion occurs when score numbers need to be displayed on the monitor CRT or sometimes by an LED display. In this case, the device takes a binary-coded-decimal number in the range of 0 0 0 0 - 1 0 0 1 and decodes it to a decimal equivalent in the range of 0 - 9. Figure 1.36-1

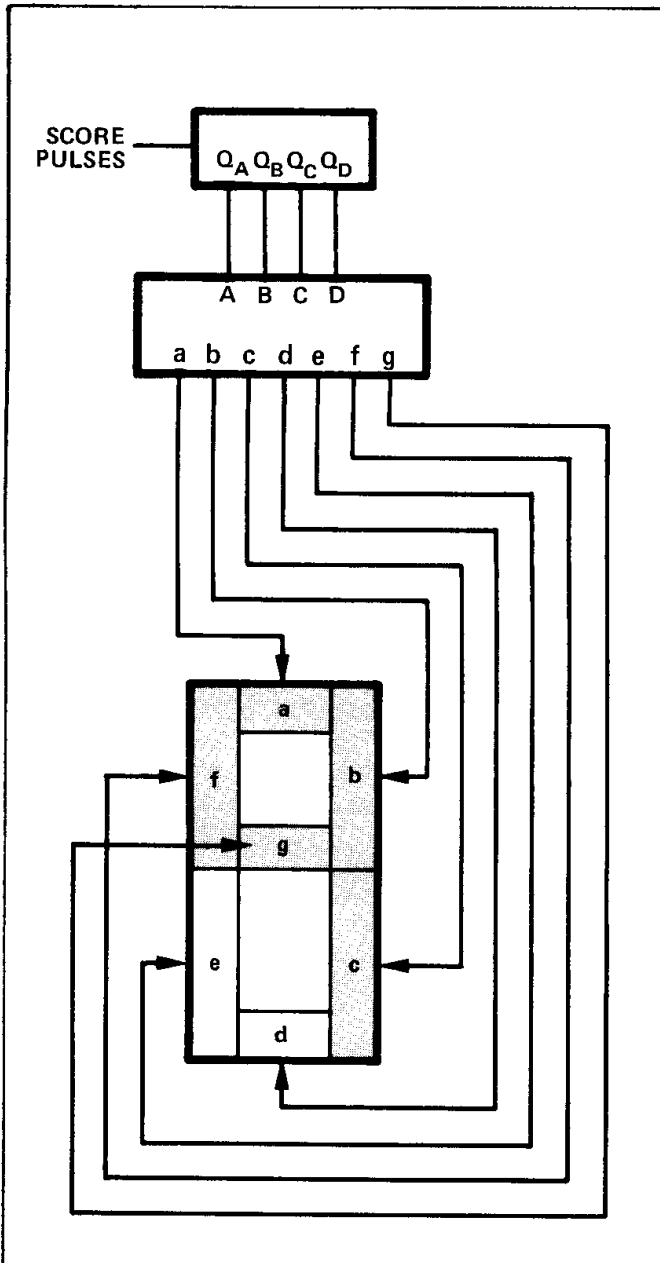
shows how a BCD number entered at the A, B, C, and D inputs is converted to the code necessary for 7-segment display.

DECIMAL EQUIVALENT	BINARY NUMBER				7-SEGMENT CODE						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

FIGURE 1.36-1 7448 TRUTH TABLE When a 4-bit binary number is entered into the D, C, B, and A inputs, the decoder outputs a code at the a, b, c, d, e, f and g outputs to light the proper segments of a 7-segment display.

Now let's apply this truth table to a game situation where we want to display a player's score on a 7-segment LED display. First, we need to count the player's scoring events so we use a detector circuit (not shown) to generate a pulse each time the player completes the specified task. These pulses are fed into the clock input of a BCD counter which counts the pulses and outputs the appropriate code at the Q<sub>A</sub>, Q<sub>B</sub>, Q<sub>C</sub> and Q<sub>D</sub> outputs. The counter outputs are connected directly to the A, B, C and D BCD inputs of a 7446 7-segment decoder and it then outputs the code necessary to illuminate the proper segments.

For example, the ninth time the player scores the counter will output 1 0 0 1. If you examine the line of the truth table to the right of decimal 9, you will see that 1 0 0 1 is decoded to 1 1 1 0 0 1 1 at the decoder's a, b, c, d, e, f and g outputs. The only two outputs which do not go HI are those for the d and e segments so all the rest of the



**FIGURE 1.36-2 TYPICAL 7-SEGMENT DECODER USE** This is a common scheme where a 4-bit counter counts score pulses to form a binary number equivalent to the player's decimal score value. The decoder takes the binary number and lights the proper segments of the display.

segments are illuminated producing the number 9 which is indicated by shaded segments.

But in most video games, the number must be displayed on the CRT of the monitor and this is done by using multiplexers to divide up the CRT into areas which contain seven invisible segments. A de-

coder is then incorporated so that the proper segments will be illuminated by the electron beam when it is enabled by segment information from the multiplexers. But we will go into greater depth on how this is actually done in the chapter on score displays.

### 1.37 Adders

So far, we have talked about some of the ways data is generated (gates, flip-flops and counters) and handled (shift registers, multiplexers and decoders). But if we want to perform arithmetic operations such as adding and subtracting, we need to use other types of devices such as adders and ALUs. Although the logic diagram (Figure 1.37-1) of the four-bit adder may appear fairly complex, its functions are really very simple.

The two four-bit numbers are entered in the A and B inputs and the  $\Sigma$  outputs immediately reflect the sum without being clocked. The carry input ( $C_{IN}$ ) and carry output ( $C_4$ ) are only used when more than one adder are used to deal with numbers greater than four bits or 15. The  $C_4$  bit provides five-bit capability which is necessary when two large four-bit numbers are added. For example, if 10 and 15 are added, the result in binary is 1 1 0 0 1 which is a five bit number. In this case, the  $C_4$  output occupies the 16s place. If two adders are used, the carry output of the first is wired to the carry input of the next to provide overflow. If only one adder is used, the carry input must be held LO so it will begin with zero overflow.

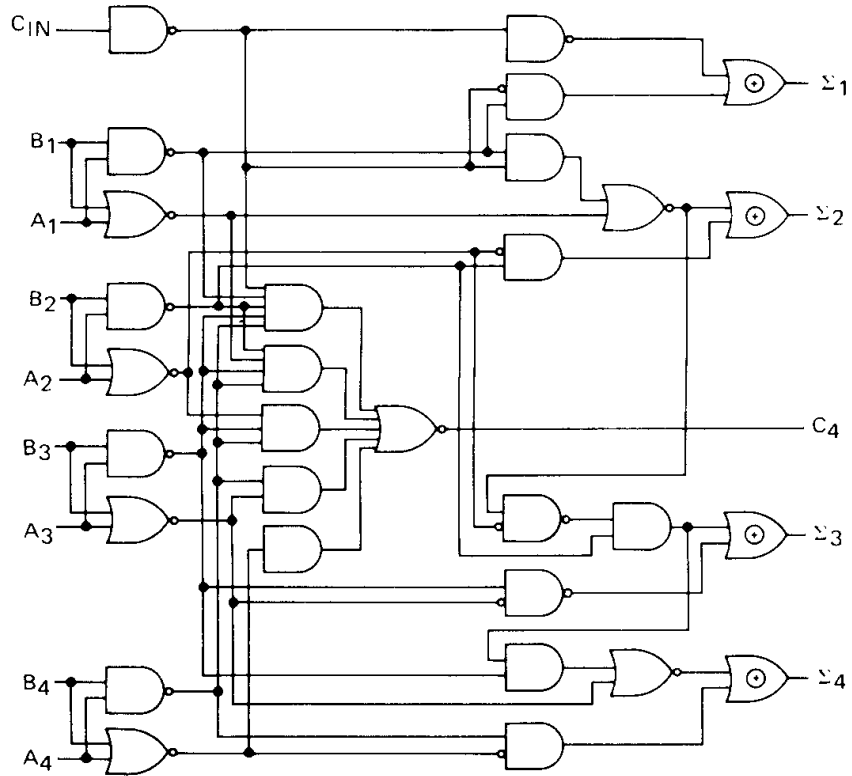


FIGURE 1.37-1 4-BIT ADDER LOGIC DIAGRAM Although this logic diagram may appear somewhat complex, all this adder does is to add two 4-bit binary numbers

appearing at the A and B inputs. The  $C_{IN}$  and  $C_4$  pins allow several adders to be "ganged" to accommodate numbers larger than four bits.

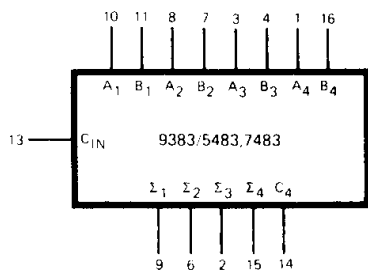


FIGURE 1.37-2 7483 SYMBOL

$A_1, B_1, A_3, B_3$	Data Inputs
$A_2, B_2, A_4, B_4$	Data Inputs
$C_{IN}$	Carry Input
$\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4$	Sum Outputs
$C_4$	Carry Out Bit 4

FIGURE 1.37-3 7483 PIN NAMES

INPUT				OUTPUT					
				WHEN $C_{IN} = 0$			WHEN $C_{IN} = 1$		
				WHEN $C_2 = 0$			WHEN $C_2 = 1$		
$A_1$	$B_1$	$A_2$	$B_2$	$\Sigma_1$	$\Sigma_2$	$C_2$	$\Sigma_1$	$\Sigma_2$	$C_2$
$A_3$	$B_3$	$A_4$	$B_4$	$\Sigma_3$	$\Sigma_4$	$C_4$	$\Sigma_3$	$\Sigma_3$	$C_4$
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	L	H	L	H	H	L
L	H	L	H	H	L	H	L	H	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

FIGURE 1.37-4 TRUTH TABLE



# 2

## **theory of tv monitor operation**

## 2.1 Introduction

It comes to our attention that the majority, or at least a large percentage, of those of you beginning to learn about video games are experienced “tube jockies” with a strong background in television repair. If this is true for you, please skip over this chapter as there is little value in wasting time on basic information you already know. The only function of this chapter is to orient those of you who do not know *anything* about TVs.

Since this book is intended as an aid to repairing video games, some understanding of monitor operation is necessary as the architecture of every game is designed around the signal requirements of the monitor. However, we are not attempting to train TV repairmen here, so we have included only that information relevant to our primary objective. Also, we have tried to keep the information presented in this chapter relatively simple so we do not lose anybody early in the game because it is really not necessary to have a vast amount of monitor knowledge to fully understand and successfully repair video games.

This chapter is organized basically into two sections. The first part contains a brief description of how the monitor functions electronically. Since all video games use monitors rather than receivers, no explanation of receiving circuitry, tuners, etc. is necessary. The second part of the chapter is devoted to the operation of different types of raster scans, for a full understanding of this aspect is truly critical to video game operation as we shall see in successive chapters. If you come to feel the depth of the information presented here is too shallow for your particular needs, drop by your local electronics store where any number of TV repair manuals are generally available.

## 2.2 General

The TV monitor performs a function most people have never really appreciated: the *temporal to spatial conversion* of an electrical signal. An electrical signal essentially intangible and temporal in

nature is entered into the monitor and converted by the circuitry into a spatial or physical display. The input signal contains nothing more than carefully timed electrical pulses which are converted by the circuitry to form a realistic image. The input signal may be considered *one-dimensional* for the only factor distinguishing one pulse from another is a certain amount of sequential timing. However, the monitor takes this single dimensional signal and converts it into a *two-dimensional* display wherein an image has both height and width. The process and circuitry by which causes this is what monitors and TV receivers are all about.

The signal entered into the monitor contains two types of electrical information: *sync* and *video*. The sync portion contains both horizontal and vertical instructions which help control the way the electron beam sweeps the CRT. The video signal contains an electrical representation of the actual information or image to be displayed. This video information contains precisely timed pulses of the correct amplitude synchronized with the movements of the electron beam so that points of light are illuminated by the electron beam's excitation of the phosphorescent coating on the inside of the monitor CRT. Illumination is controlled and images are formed by modulating or intensifying the electron beam just as it reaches the exact point desired. Depending on the degree of modulation, points of different intensity may be illuminated. In other words, the greater the amplitude of the incoming video signal, the more powerful the electron beam becomes, hence the affected areas of phosphor glow more brightly.

## 2.3 Types Of Scanning Systems

Two completely different methods have been devised for displaying images on a CRT. The more familiar method is the *raster scan* method used in TV receivers and video games. In this case, the electron beam scans the CRT in an endlessly repeated pattern or *raster*. Images can then be created by modulating the electron beam as it reaches certain points in the raster pattern.

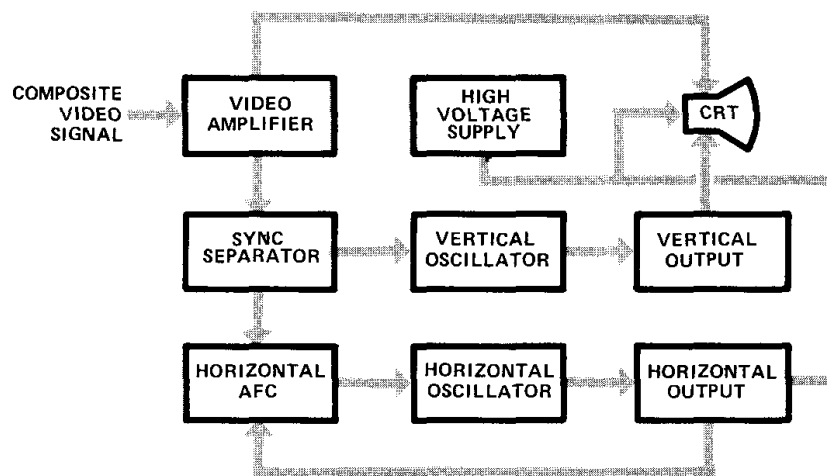
*Vector scan* is a more elaborate technique found in sophisticated systems such as graphics terminals, oscilloscopes and radar displays. The circuitry of a vector scan monitor positions the electron beam directly where the image is to be generated and only then modulates the beam. When it has completed the image, or part of the image, it is vectored off to a new location. The electron beam does not follow a repetitious pattern; it only moves where instructed. Since this type of scan has a higher resolution (clarity of image), the expense of a vector scan system is justified for applications where a crisp, well defined outline is an absolute necessity.

Although the vector scan technique delivers higher resolution, the raster scan system has been adopted for TV broadcasting because it requires considerably less circuitry and electrical power. Enormous demand has caused the price of raster scan receivers and monitors to drop to a level where almost anybody can afford one, however vector scan systems with the necessary back-up circuitry remain a relatively specialized and expensive item.

## 2.4 Raster Scan Monitor Operation

Figure 2.4-1 is the block diagram for the type of monitor found in most video games and it has not been overly simplified in case you are wondering about the seeming general simplicity of the system. If you have any doubt as to how simply these monitors really are, we have reprinted the Motorola Schematic (Figure 2.5-2) so the entire circuit can be examined.

The *composite video* signal generated by the game computer contains both the sync and video information. This signal is entered into the *video amplifier* section where it is amplified to a level high enough to drive the rest of the monitor's circuitry, however the waveform retains its original shape even after amplification. This amplified composite signal is then taken directly to the CRT where it is used to modulate the intensity of the electron beam and also sent to the *sync separator* where an important process known as *video stripping* takes place.



**FIGURE 2.4-1 TV MONITOR BLOCK DIAGRAM** This type of TV has no receiving circuitry at all since the video game signal is wired directly into the video amplifier. Regular TVs can also be used as monitors by cutting the output of the third I.F. stage and inputting the digital game signal instead.

The sync separator is designed in such a way that it strips off the video information, leaving only the composite sync signal which includes both the horizontal and vertical sync pulses. The resulting sync signal (known as COMP SYNC) is now in a form usable to both the horizontal and vertical oscillators.

The vertical oscillator section is designed in such a way that it "sees" only the vertical sync pulses and ignores the horizontal ones. It uses the vertical pulses to generate the *sawtooth* waveform illustrated in Figure 2.4-3.

This sawtooth waveform is amplified by the Vertical Output Section to a level high enough to enter the yoke of the CRT and *deflect* the electron beam vertically down the screen.

The composite sync signal also enters the Horizontal AFC section (Automatic Frequency Control) which compares the incoming sync signal with the amplified horizontal oscillator output and uses any resulting phase differential to pull the oscillator frequency up or down to correct any error which might have crept into the system. The resulting *phase corrected* signal is further amplified by the Horizontal Output Section and sent to the yoke to drive the electron beam horizontally.

Notice that the wave form from the Horizontal Output Section is also used in the High Voltage Section. This section contains a *flyback transformer* which requires a series of very sharp input pulses to function and the waveform from the Horizontal Output Section is simply a convenient source of a sharp pulse. Otherwise, a special circuit

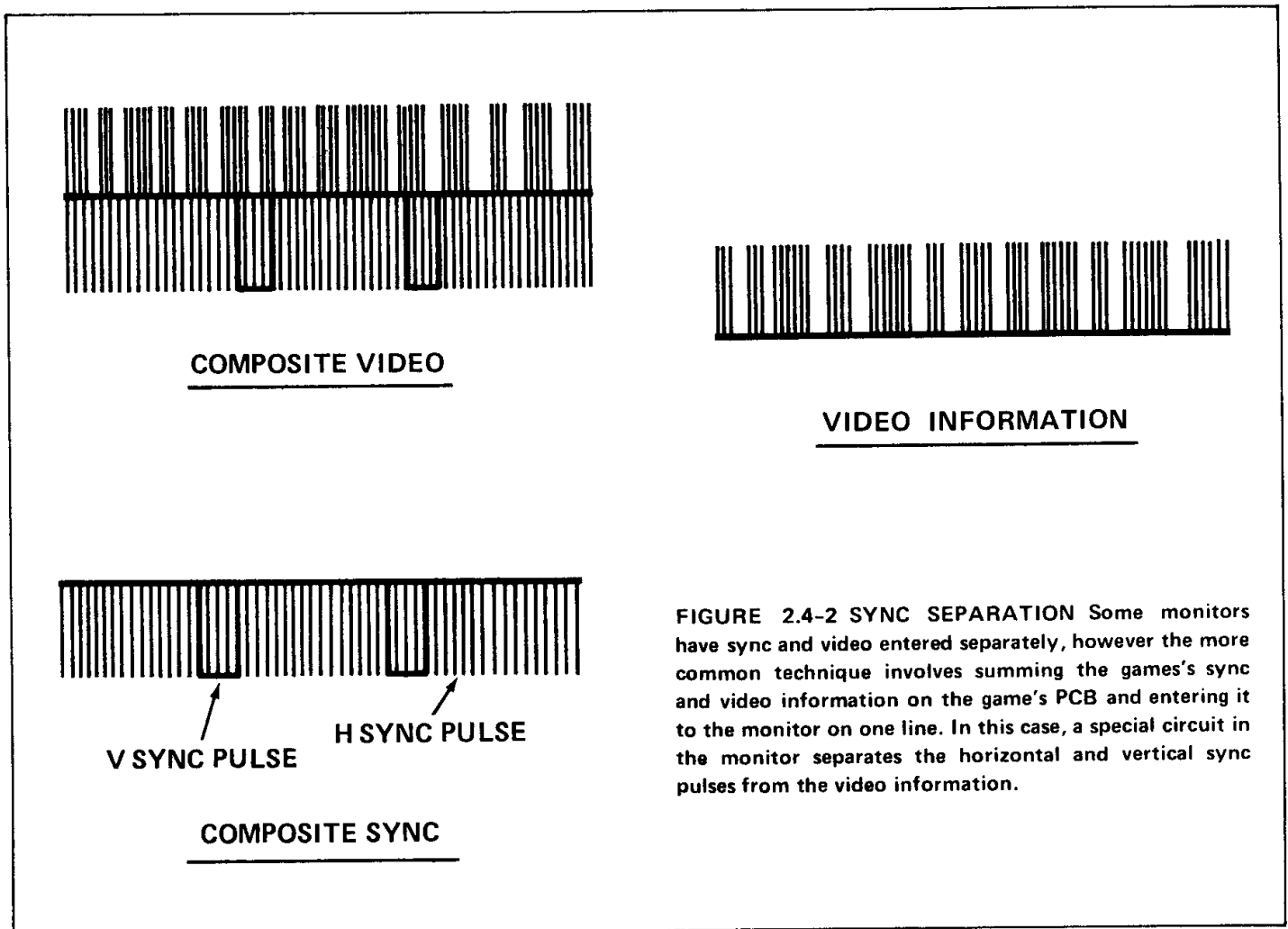
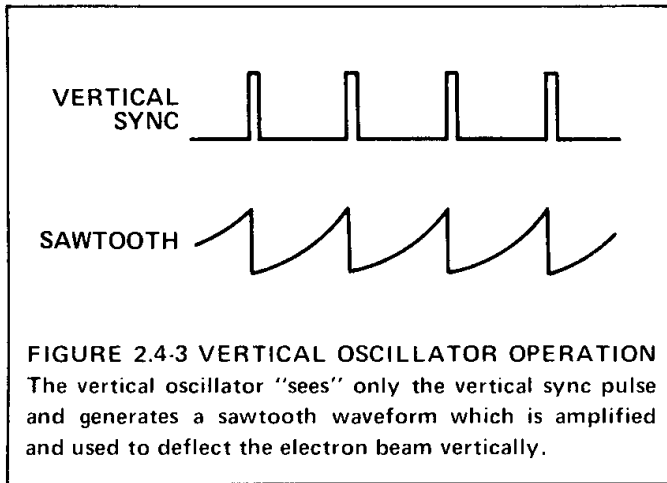


FIGURE 2.4-2 SYNC SEPARATION Some monitors have sync and video entered separately, however the more common technique involves summing the games's sync and video information on the game's PCB and entering it to the monitor on one line. In this case, a special circuit in the monitor separates the horizontal and vertical sync pulses from the video information.



would be needed solely to operate the flyback transformer so that it can produce the high frequency, high voltage required. A nice visual demonstration of this high frequency stuff can be had by bringing the point of an insulated screwdriver within about a half inch of the high voltage supply output (i. e. where it connects to the CRT).

Even when no input signal is connected to the video amplifier of the monitor, it still produces the raster and — since it is not receiving any incoming sync — it operates in the *free running* mode where it simply generates a raster not timed with the operation of any other device. However, when connected to the incoming sync from the game computer, it *locks* onto the sync signals and becomes fully *synchronized* with the operation of the computer.

## 2.5 Making Your Own Monitor

Any transistorized black and white TV receiver is capable of being quickly converted to a monitor. Essentially, all that is required is that the receiving circuitry (3rd I.F. stage) output be taken to a switch mounted on the outside of the TV so it may be turned off. Then, an external connector (i. e. BNC) is connected directly to the input of the video amplifier so the game computer signal can be entered directly to the amplifier. And that's all there is to it.

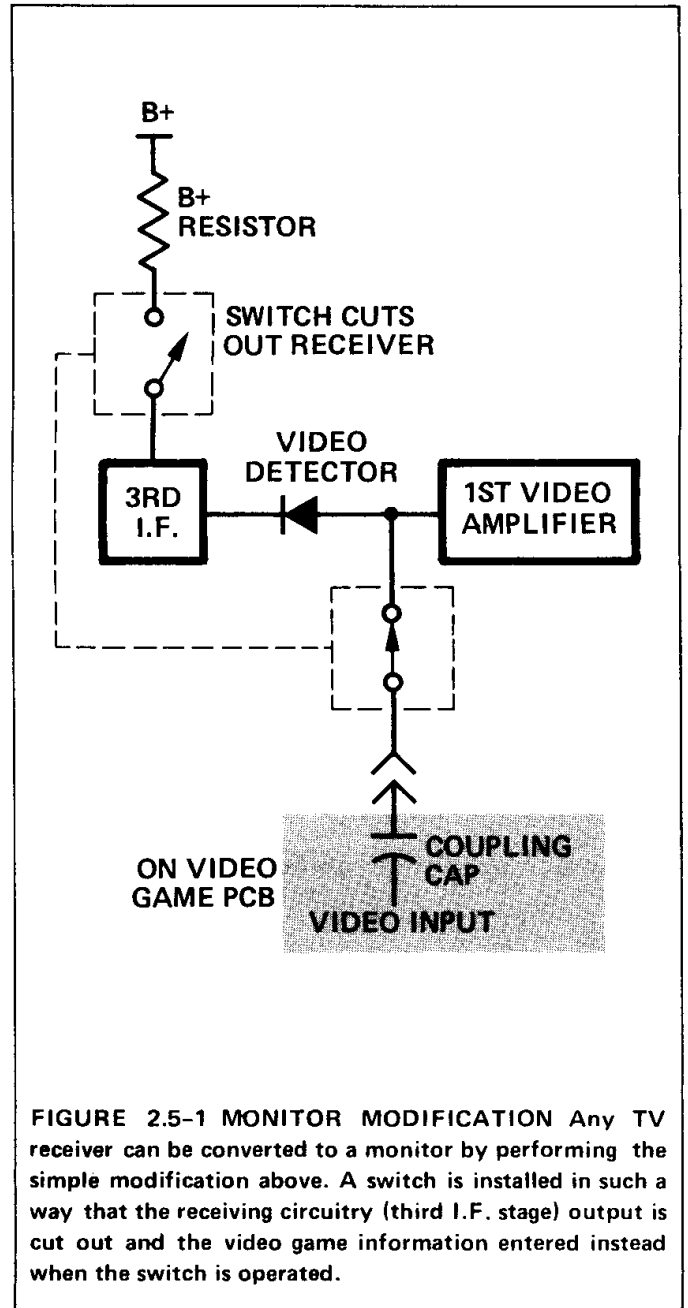
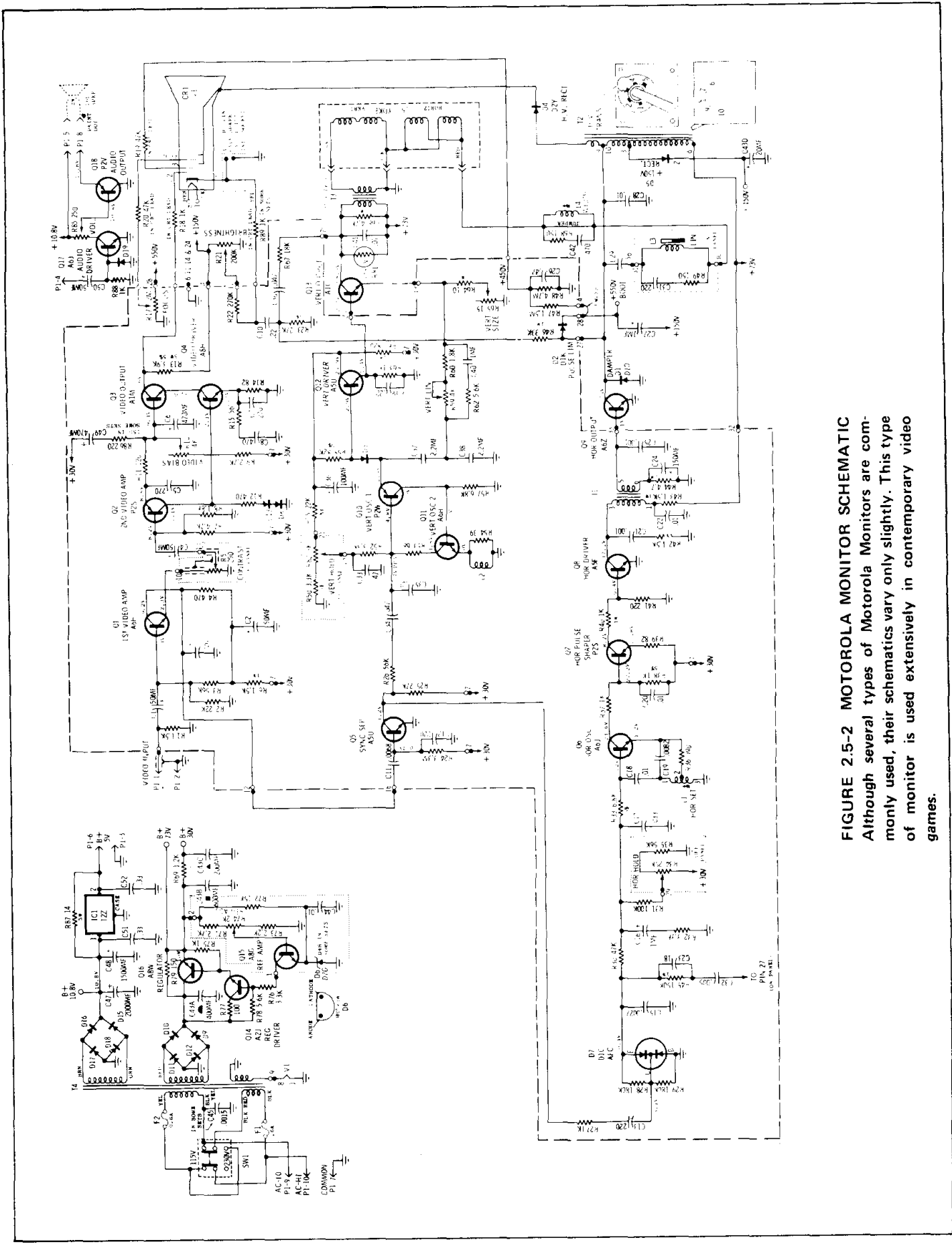


FIGURE 2.5-1 MONITOR MODIFICATION Any TV receiver can be converted to a monitor by performing the simple modification above. A switch is installed in such a way that the receiving circuitry (third I.F. stage) output is cut out and the video game information entered instead when the switch is operated.

An alternative technique involves building a RF modulator which is a simple device that takes the computer output and converts it to a radio frequency the monitor is capable of receiving. The RF modulator can usually be made to work adequately after a little tweaking, however the preferable method is the direct connection through a piece of coax. Please keep in mind that some TVs are made with a "hot chassis" which will not conveniently accept a direct input making the mod-



**FIGURE 2.5-2 MOTOROLA MONITOR SCHEMATIC**  
 Although several types of Motorola Monitors are commonly used, their schematics vary only slightly. This type of monitor is used extensively in contemporary video games.

ulator a necessity. Consult your TV schematic to be sure.

## 2.6 Types Of Raster Scans

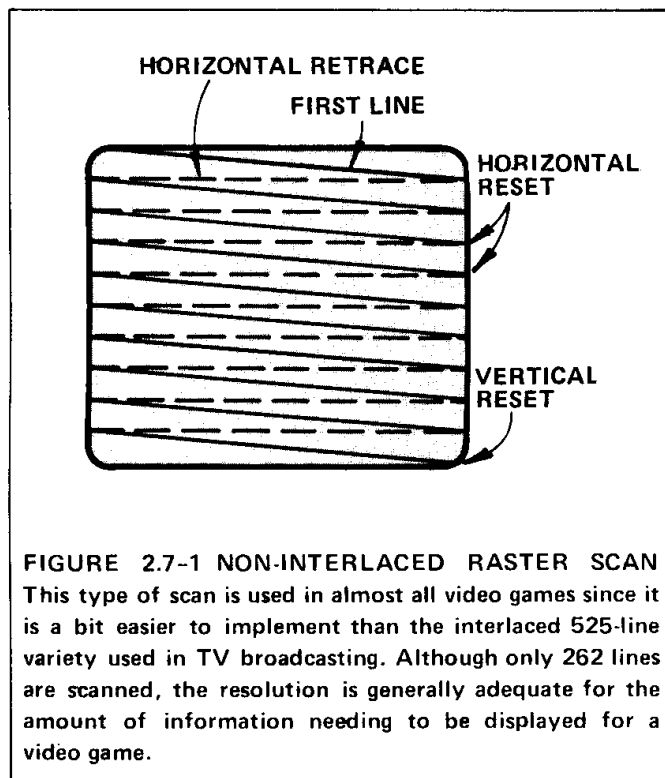
While the actual electrical operation of the monitor is not especially important to understanding how a video game computer functions, the end result — the raster pattern — is a subject which must be fully grasped before video game circuitry will make any sense at all. Since the majority of the circuitry found in the game computer is precisely timed with the movements of the electron beam, it becomes extremely important to become familiar not only with the timing of the raster, but also with the various terms associated with electron beam movement.

Most of you are probably familiar with the standard broadcast system used in this country for TV reception which employs a *fully interlaced* scan of 525 lines per frame. But for simplicity, almost all video games use a *non-interlaced* scheme where identical fields of 262 lines each are repeated twice per frame to eliminate the additional circuitry required for a full interlace. However, a small minority of video games does use the interlaced method to achieve greater resolution for the display of very small images, so our discussion of raster generation will cover both methods.

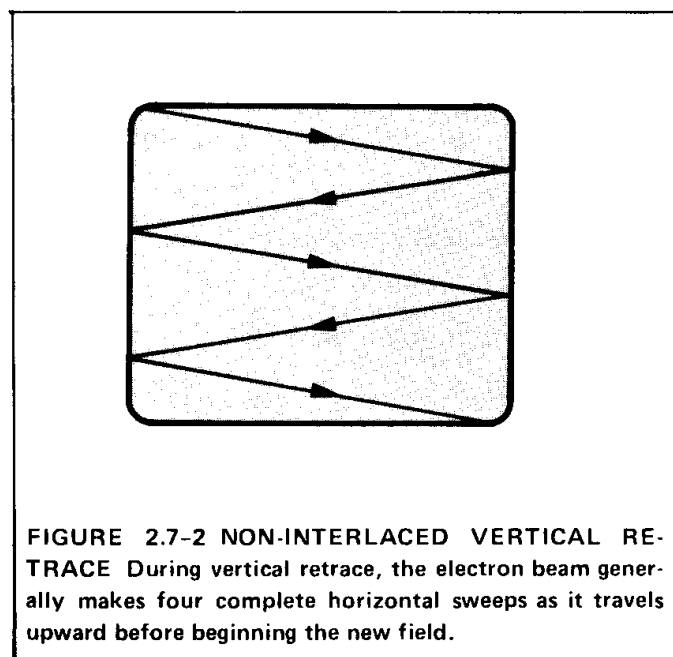
## 2.7 The Non-Interlaced Raster Scan

Figure 2.7-1 is a simplified illustration of non-interlaced operation. To start a new frame, the electron beam begins in the upper left corner of the CRT and sweeps out a *slanting horizontal line* ending at the right side. The electron beam moves to the right because it is being deflected horizontally by the output of the horizontal oscillator and the line slants downward slightly because the beam is also constantly being deflected vertically by the vertical oscillator output.

After scanning the first line, the electron beam is repositioned back to the left side by a process known as *horizontal retrace*. During retrace, the



electron beam is turned off (the video information blanked out) so no undesired illumination will occur. Blanking during retrace is an internal function of the monitor and is not related to blanking signals produced by the computer in any way other than by the fact that computer signals V BLANK and H BLANK occur along with retrace



blanking. V BLANK and H BLANK are signals used to load certain kinds of information into various computer circuits before the electron beam begins a new line or frame.

After the electron beam has been reset to the left side of the CRT, it sweeps the second horizontal line. If the monitor is synchronized to the computer-generated sync signal, it begins this line as the horizontal sync pulse occurs. Since the electron beam is still being deflected vertically, the second line appears beneath the first and the beam continues to scan the CRT in this fashion until it reaches the end of the last line which is located in the lower right corner of the CRT.

At this point, both a vertical and horizontal sync pulse occur and the electron beam is reset back to the left edge and to the top of the CRT to the point of its beginning. If you carefully examine a vertical sync pulse, you will notice that it is *ser-rated* by four horizontal sync pulses. This causes the electron beam to sweep the CRT four times while it is being deflected back up the CRT (Figure 2.7-2).

Each time the electron beam has completely scanned the CRT, it has completed one full *field* of 262 horizontal lines. Since two fields constitute a *frame*, there are actually 525 lines per frame. However, since fields are laid approximately on top of one another, the effective resolution is only 262 lines per frame and the monitor has a frame rate of 30/second.

## 2.8 The Interlaced Raster Scan

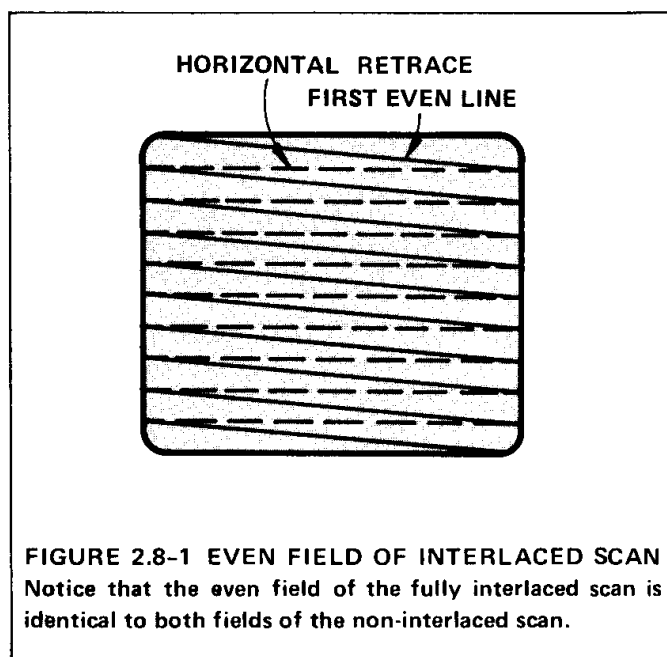
As we have already explained, the fully interlaced scan is used for applications where small or very detailed images are displayed and greater resolution is required. This becomes particularly important when image memory area is limited and a detailed image needs to be generated using very little actual data. If memory area is at a premium, the interlaced scheme will yield an image with a better appearance, however the same resolution could be more effectively obtained by doubling

the amount of image data stored within memory. Faced with this choice, the designer has two choices: (1) he can design a slightly more complicated sync circuit to interlace the scan and optimize resolution cheaply or (2) he can double the size of his ROM and obtain a much more detailed image and let the manufacturer worry about the cost of programming the ROMs.

The essential difference between the two types of scanning techniques is that in the interlaced scheme successive fields are laid *in between* each other rather than on top of one another. The net result is twice as many separately displayable raster lines per frame where each line also contains twice as many separately addressable points.

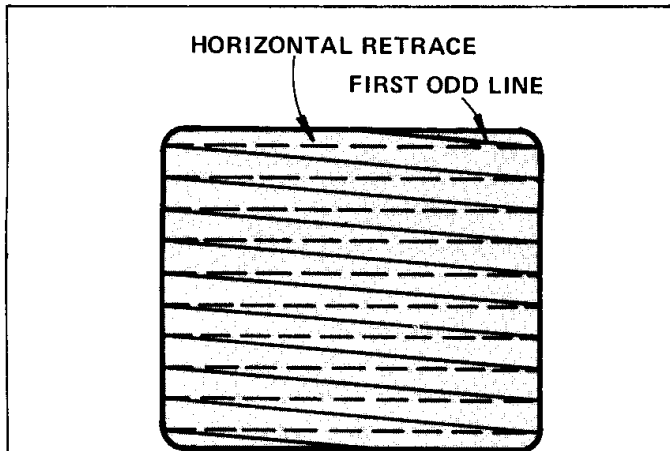
The two fields to be interlaced are known as the *even* and *odd* fields. The even field is scanned exactly like both fields of the non-interlaced scan where the electron beam begins in the upper left corner of the CRT and sweeps out a full field of 262 downward-slanting lines. There is an important bit of difference during the vertical retrace between the even and odd fields, but we will get to this aspect in a moment.

In any case, after retrace does occur, the odd field begins at the *top middle* of the CRT so the first





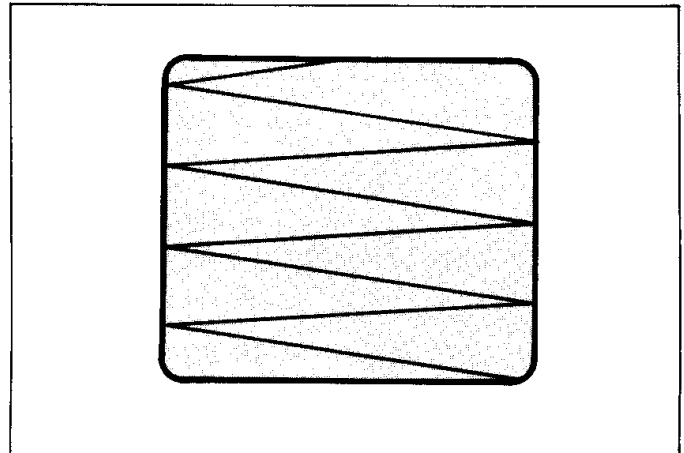
line is one-half the usual length and extends from the center to the right edge of the CRT. The last line also terminates in the center, however this point occurs at the bottom edge of the CRT.



**FIGURE 2.8-2 ODD FIELD OF INTERLACED SCAN**  
By beginning the odd scan in the center of the CRT, the lines of the odd field are staggered in between those of the even field. This technique becomes necessary when greater resolution is required.

Interlace is achieved by counting an odd number of half-lines to force the start of the odd field in the center. This necessitates a 14 MHz clock rate so each line can be divided into half-lines of 452 clock pulses each (the normal non-interlaced line is 454 clock pulses long total). In case you are completely confused as to how this is actually accomplished, be patient until the sync section so concepts built in the early part of that chapter can be used to more adequately describe the interlacing process.

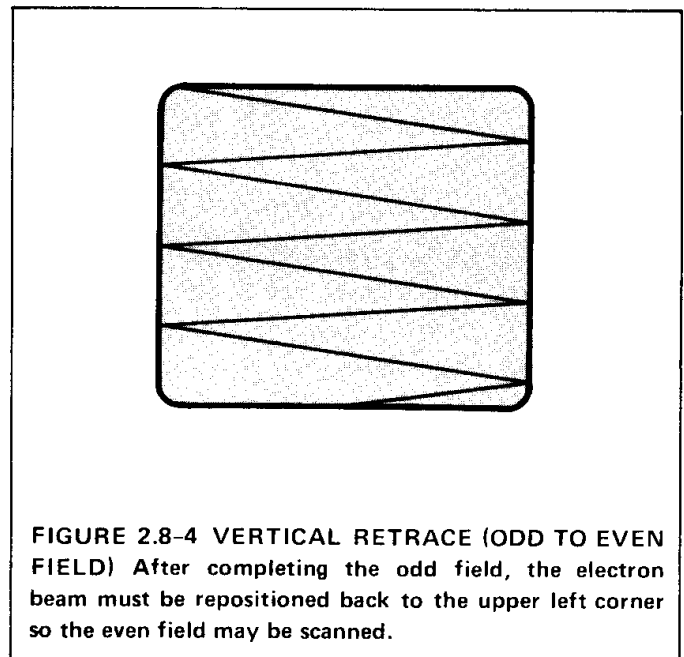
At any rate, the nomenclature associated with the interlaced scan remains basically the same as for the non-interlaced version. The only significant difference is the fact that — since the interlace is achieved by using half-lines — horizontal reset occurs both in the center of the CRT and at the right edge, however vertical reset still occurs when the electron beam is at the end of the last line of each field. And, since there is a difference of a half-line in the start and end of each field, vertical



**FIGURE 2.8-3 VERTICAL RETRACE (EVEN TO ODD FIELD)** This figure illustrates how the electron beam is repositioned at the top center of the CRT prior to sweeping the odd field. In the non-interlaced scan, the electron beam is repositioned back to the upper left corner prior to scanning the next field.

retrace must follow a slightly different route. Between the even and odd fields, the electron beam must be repositioned from the lower right corner to the center top of the CRT.

However, since the end of the odd field occurs at the bottom middle of the CRT, the electron beam must be reset from this point back to the top left corner before the even field can begin.



**FIGURE 2.8-4 VERTICAL RETRACE (ODD TO EVEN FIELD)** After completing the odd field, the electron beam must be repositioned back to the upper left corner so the even field may be scanned.

**3**

**video game  
architecture**

### 3.1 Common Themes

On a general level, all random logic video games share a basic circuit design or system architecture where some circuitry is actually identical from game to game. One finds striking similarities in the design of power supply, sync, game credit, motion and score circuits. Other areas such as playfield generation, memory techniques and sound circuits are also found to have a number of common themes.

These similarities stem from the fact that, while each game may appear somewhat different, all video games perform the same functions, hence the circuitry implemented to effect those functions also takes on a common architecture. Essentially, a video game boils down to an eye-hand coordination task displayed on a CRT where a player has some sort of control — usually switches or pots — with which he must move an object in such a way as to increase his score. Since a video monitor must — by definition — be used, a great deal of circuitry must be specifically included just to drive the monitor. And, since all monitors are operationally identical, we invariably find one of two basic types of sync circuitry which controls monitor-related functions. Another area which shows striking similarity from game to game is the power supply. Since all components require electrical power in a very closely regulated state, a specific type of power supply is associated with the game computer. In the early days of the video game industry, a particular version of power supply dominated; for these games all used TTL circuits exclusively. Now, designers are using more esoteric components so power supply design has evolved into designs which output more voltages and a greater amount of current, however most power supplies still remain quite similar.

There are a number of other similar areas as well. Since a video game must be coin operated, obviously there must be a coin and a credit circuit in each game. A number of standard coin circuit parameters such as detection of false signals, ability to set the game for one or two players or

one or two coins per play and the necessary provisions for ending the game have manifested themselves in common designs for coin and credit circuits. Score circuits also share in design similarity since there are only two basic approaches to displaying score numbers on a CRT. And even beyond this we find that all games must have playfield generation circuits, motion circuits, sound circuits and a game timer, so we find common themes here as well.

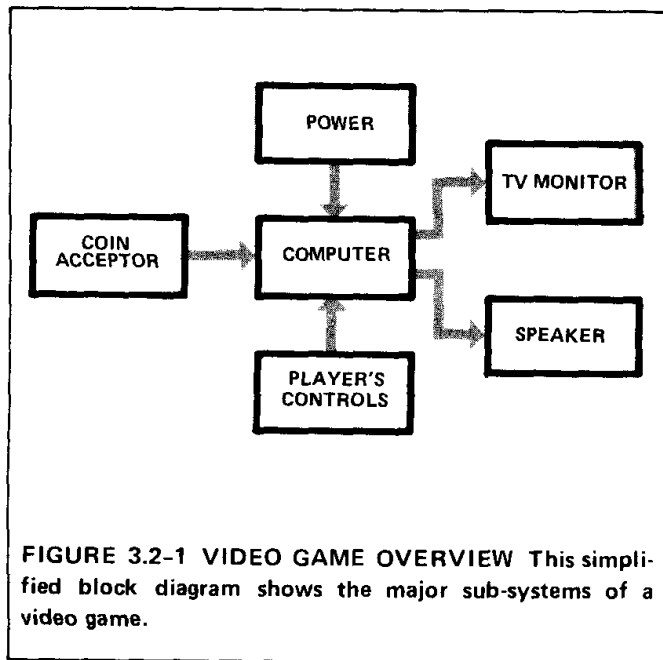
These similarities are a great timesaver for technicians because once you have become familiar with the two ways a sync circuit can be designed, you will feel comfortable with any sync circuit in any new game. The same applies for the rest of the circuitry as well. Although the circuit will always be different on the most specific level, generally all circuits with the same function will be found to be operationally similar.

### 3.2 A Simplified Overview

If you are new to video games altogether, you are probably completely confused by now about how all the various components and circuits are wired together to make a game. So far, we have mentioned a number of different circuits, but have supplied very little information to tie them together. So before getting into the specifics of how these circuits actually operate, we need to discuss the video game as a whole and only then break it down into its various sub-systems.

Figure 3.2-1 shows the basic sub-systems from which a game is constructed. On this very general level, the game consists of a cabinet in which are mounted the player's controls, a coin acceptor, a transformer, a printed circuit board computer, a TV monitor, a speaker and a wiring harness to connect the whole mess together electrically.

On a more specific level, these components have the following functions. The player's controls vary with the theme of the machine and usually involve some sort of eye-hand coordination task pitting



players against themselves or against the computer itself. The coin acceptor is a mechanical device which tests a deposited coin in a number of ways and operates a micro-switch if the coin is accepted. The power supply (at this point) consists only of a transformer which reduces the line voltage to a level more suitable to the on-board power supply components which further process this AC voltage into one or more carefully regulated DC supplies. As time passes, more and more games are found to have separate power supply PCBs which do all the processing required. The advantage of a separate supply is that the manufacturer can design a single power supply circuit which will serve a whole series of games and thereby eliminate the need to create a new supply design for each new game.

The printed circuit board computer for random logic games is a *dedicated* computer which generates all the logic functions necessary to form the various video displays and change them as necessary to form the various video displays and change them as necessary. By dedicated computer, we mean that the PCB has been specifically designed to operate one game and cannot be used for another game unless extensive internal modifications are made to the actual hardware. All contemporary games however do not use dedicated comput-

ers. The new microprocessor games are capable of being changed into a whole new game simply by changing a *software* or *firmware* package.

The computer receives a number of inputs and generates many types of signals, most of which are used internally only. On the input side, raw AC power and ground are connected to some of the edge connector pins while inputs from the player's controls, the coin acceptor and start switch are connected to others. Other PCB edge connector pins are used to output sync and video to the monitor, sound waveforms to the speakers and miscellaneous signals which vary from game to game and go out to control special features such as indicator lamps or LED displays and occasionally electromechanical devices such as "thumper" coils.

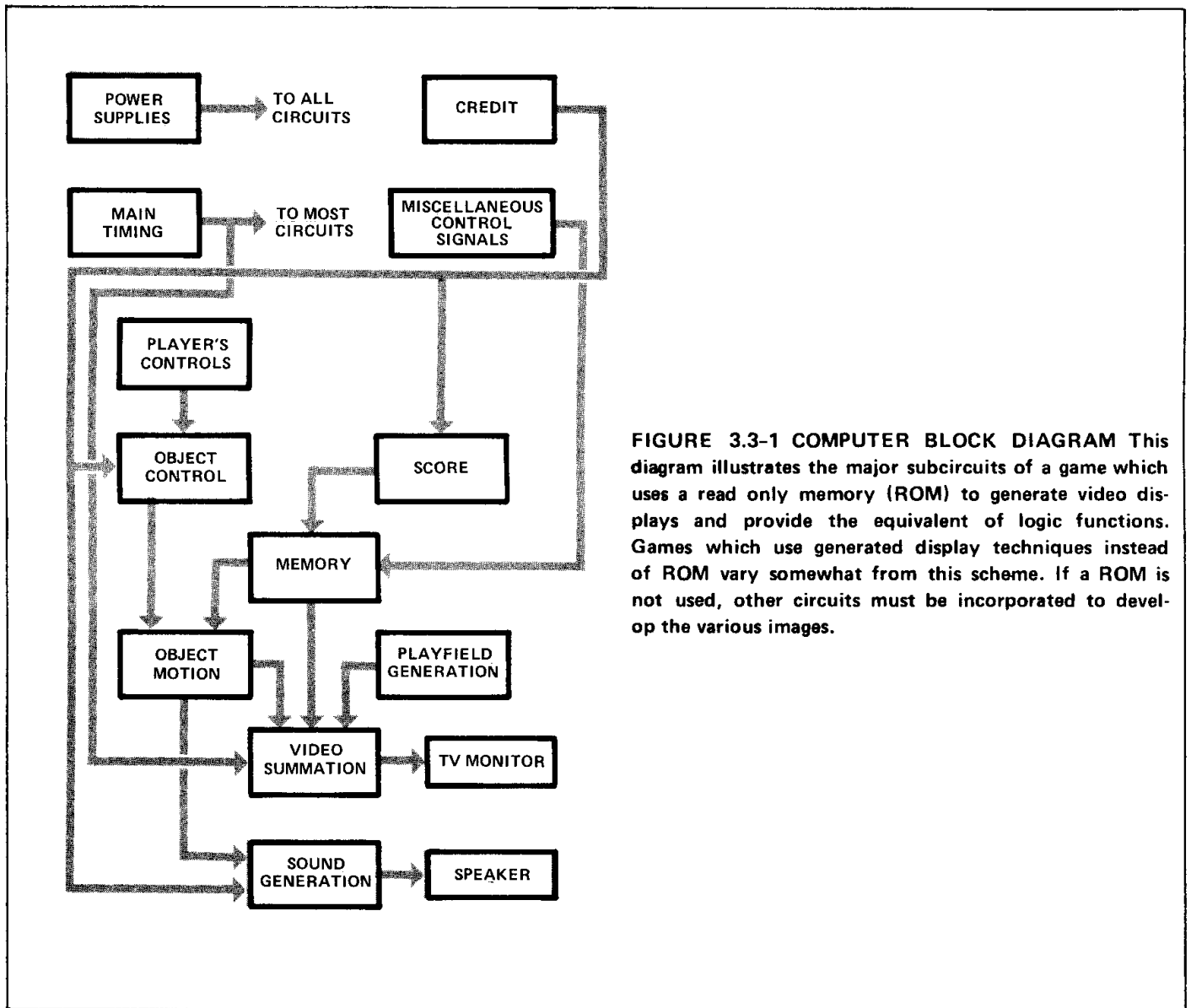
### 3.3 Functions & Relationships of Common Circuits

The following descriptions are simplified and streamlined for a great deal of depth here would only confuse the subject. Since this chapter is but a vehicle for the orientation to the general architecture of the system, we need to stress only the main functions of and significant interrelationships between circuits for successive chapters go into considerably greater depth about each type of circuit.

The typical configuration of a game using a *read-only memory* (ROM) is illustrated in the block diagram below. While not all games follow this general design, it is common to many games, especially contemporary ones.

### 3.4 Power Supplies

The function of any power supply is to generate one or more closely regulated DC voltages to power the integrated circuits and other components. The reduced AC voltages from the transformer secondaries are first rectified and then filtered by a large capacitor to produce one or more pulsating DC levels. These voltages are sent to other sub-circuits which fully regulate the waveforms so they are even voltages at the proper DC levels.



**FIGURE 3.3-1 COMPUTER BLOCK DIAGRAM** This diagram illustrates the major subcircuits of a game which uses a read only memory (ROM) to generate video displays and provide the equivalent of logic functions. Games which use generated display techniques instead of ROM vary somewhat from this scheme. If a ROM is not used, other circuits must be incorporated to develop the various images.

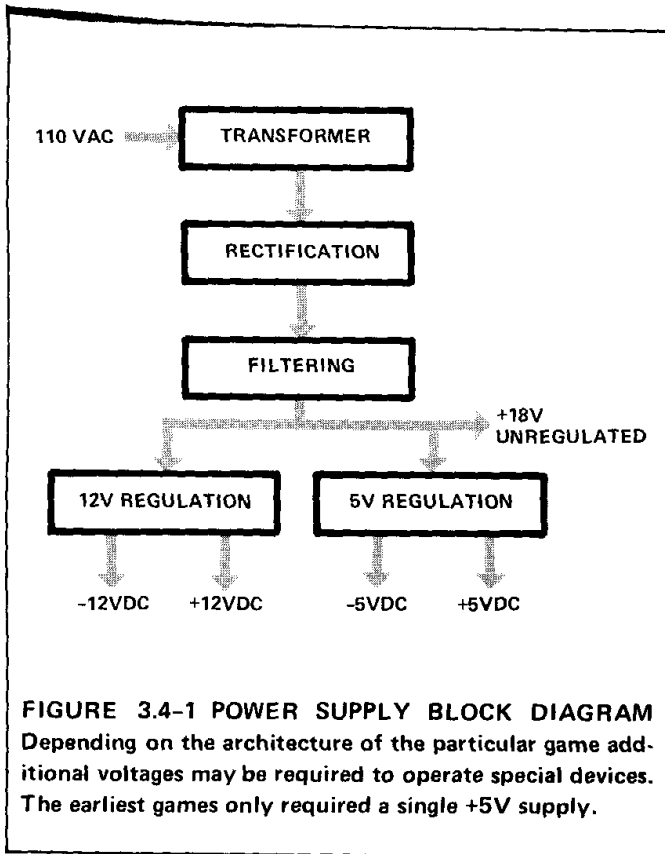
Generally, integrated voltage regulators are used for the +5 volt supply which greatly reduces the number of discretes which would otherwise be required. However, these regulators are good for only so much current so when an especially large amount of current is required, a more elaborate supply is custom designed using a pass transistor controlled by other circuitry which senses voltage and load and compensates for voltage drops due to load fluctuations.

### 3.5 Coin & Credit Circuitry

This circuit generally has a number of loosely related sub-circuits grouped together to make sure

the operator gets his share. The primary function, of course, is to detect the deposit of a coin and provide game credit. The signal COIN indicates a coin has been deposited and one of its functions is to enable the attract circuit. When the start switch is closed, the start circuit outputs a HI START signal which operates the attract circuit so a LO ATTRACT is produced. ATTRACT is generally the signal which allows or prevents the game's play mode function to begin. When not in play, ATTRACT returns HI, disables the game and starts the attract mode display to entice another player.

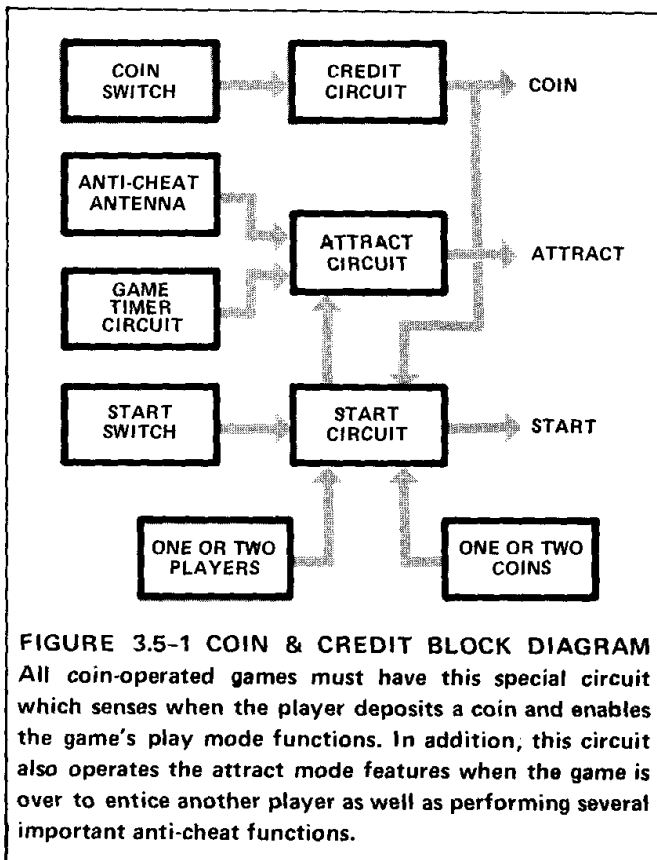
If the proper signal from either the antenna circuit or the game timer reaches the attract circuit, the



game will be instantly shut off. The antenna is used to sense the release of a static discharge which would otherwise allow players to obtain free games merely by inducing a static discharge into the coin circuit. The game timer is as its name implies – simply a circuit which counts time or score and notifies the attract circuit to shut the game off when the player has had his money's worth. Start sub-circuits are usually adjustable so one or two players may use the machine by paying either one or two quarters per play.

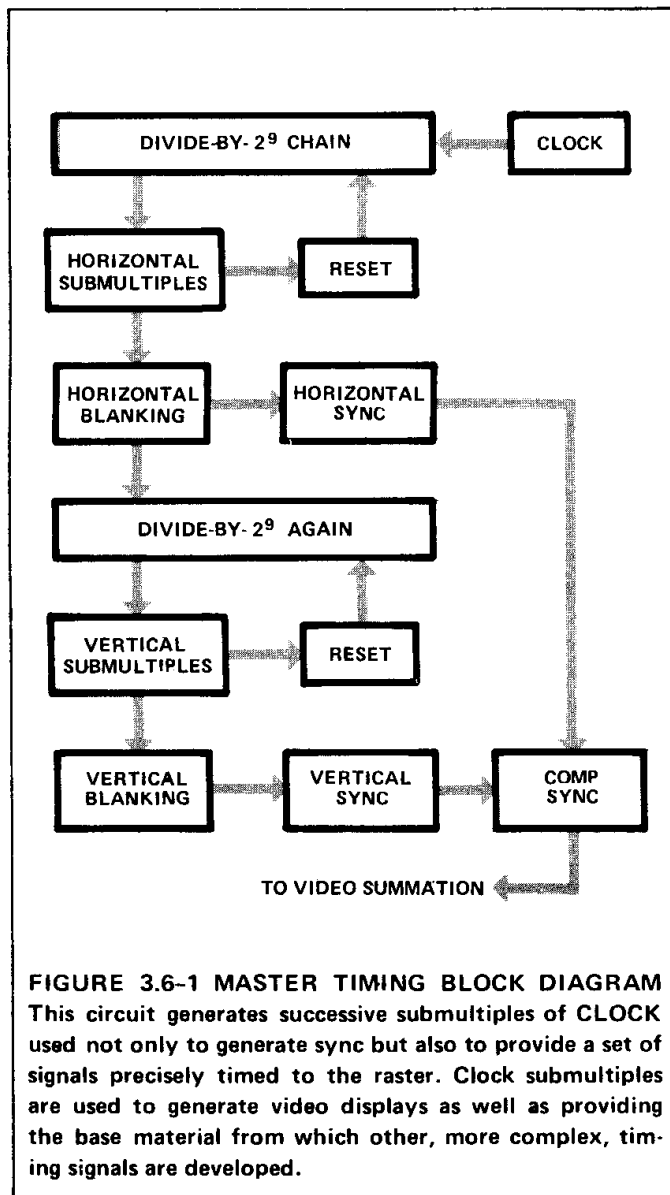
### 3.6 Master Timing

Sync circuitry is divisible into three general areas: the *Clock*, *Horizontal Sync* and *Vertical Sync*. The Clock is a crystal oscillator which generally produces a 14.318 MHz frequency divided in half to 7MHz to form CLOCK, the master timing signal for the entire computer. This signal is divided  $2^9$  times by the horizontal sync chain which consists of two four-bit counters and a flip-flop. Each bit of the counter chain successively divides CLOCK in half and outputs horizontal submultiples, some of which are added together to form a reset pulse which sets the entire chain back to zero when the electron beam has reached the end of a line. Other submultiples are added together to develop the horizontal blanking signal H BLANK which is used to load information into other circuits while the electron beam is being retraced. H BLANK is also used as the base signal which is further processed to form the actual H SYNC pulse to which the horizontal oscillator of the monitor is synchronized.



The H BLANK signal also provides the input to the vertical divide chain which again divides this frequency  $2^9$  times to produce various vertical submultiples. These submultiples are combined to develop vertical reset, blanking and sync signals which have the same basic functions as their horizontal counterparts. However, this time V RESET occurs when the electron beam has reached the end of the field and V BLANK occurs during the time of vertical retrace.

Finally, the horizontal and vertical sync signals are



combined into a composite signal (COMP SYNC) which is sent to the video summation network to be combined with COMP VIDEO (all the display information) before actually entering the monitor.

### 3.7 Motion Circuitry

Motion is generally created by a standard set of circuits known as *slipping counters* which run in parallel with the sync chains. The similarity between the sync and motion circuits should be readily apparent from the block diagrams and there is a very good reason for this parallel design. Although we are going to delve much more deeply into this subject in the motion chapter, here is a brief analysis of how it happens.

Picture the electron beam moving horizontally across the CRT, sweeping out raster lines. At the end of each line, an HRESET pulse occurs from the horizontal sync chain. Now let's say we have an image stored within memory which is enabled to appear only when it receives a signal from the motion circuitry. If the motion output occurs exactly in time with the horizontal reset signal, the object will be read out in the same place each frame and will appear to be stationary. However, if the motion circuit is slowed a little bit (i.e. one clock pulse), it will output its signal a little later than the horizontal reset pulse and the object will be created a bit to the right each frame so it will appear to move to the right.

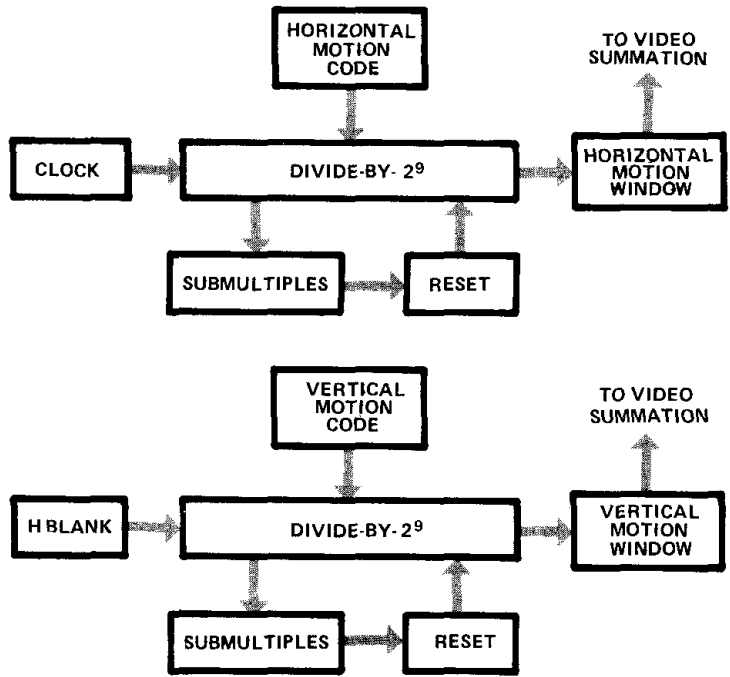
By slowing or speeding up the horizontal motion chain with respect to the sync chain, we can make the image move to the left or right. Controlling the vertical motion chain similarly causes up and down movement. When the outputs of both chains are combined, the image can be vectored off at almost any angle desired.

The chains are controlled by *motion code* outputs from the motion control circuits which are in turn affected by how the player sets his controls. These codes are loaded into the parallel inputs of presetable counters to change the point at which they reach terminal count and effect the time differential between the motion and sync chains.

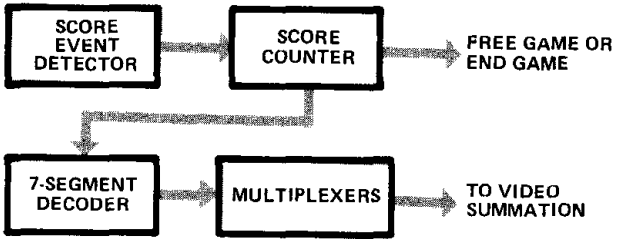
### 3.8 Score Circuits

Early games used the score display technique illustrated in Figure 3.8-1 where a score event detector senses when a player achieves the specified objective and increments a counter. The output number of the counter is used for the input to a BCD to 7-segment decoder which outputs the code necessary to turn on the proper segments of a 7-segment number generated on the monitor CRT by a multiplexer subcircuit. The score counter is usually wired either to provide a free game at a certain score or to end the game at the appropriate time.

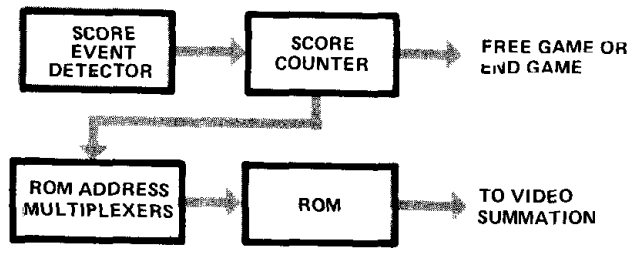
While this technique is still a valid one, most con-



**FIGURE 3.7-1 TYPICAL MOTION CIRCUITS** Separate circuits are required to operate both vertical and horizontal motion, however the most important characteristic to notice in this diagram is the striking similarity between the motion and master timing circuits.

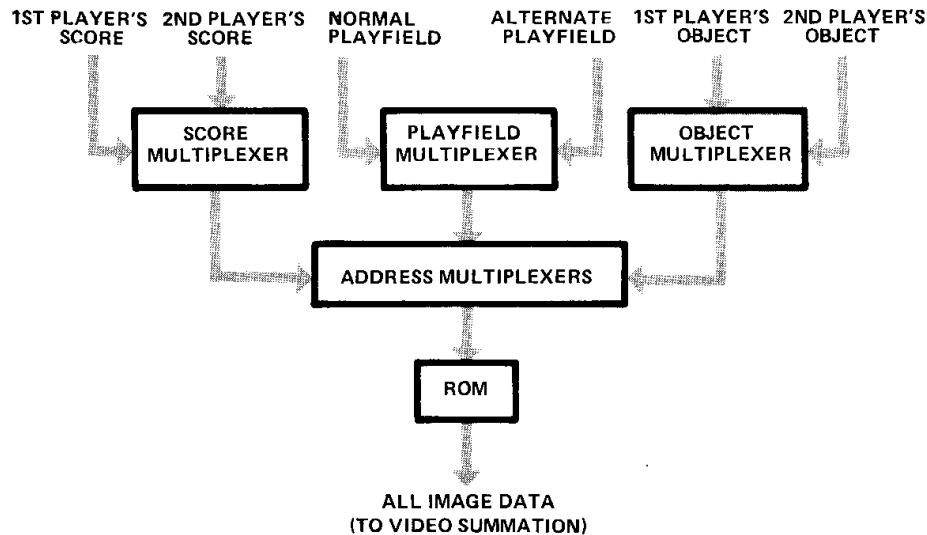


**FIGURE 3.8-1 7-SEGMENT SCORE DISPLAY** This type of score display circuit uses random logic to generate the score numerals and is seen primarily in earlier video games. The advantage of this configuration is that no special ROM need be programmed.



**FIGURE 3.8-2 STORED-NUMERAL SCORE DISPLAY** This type of numeral generator is quite common in contemporary games which need a ROM for a variety of functions. By storing the actual numeral information within memory, the circuitry can be simplified and the numerals themselves can be given interesting shapes without increasing circuit complexity.





**FIGURE 3.9-1 MEMORY CIRCUIT** This circuit contains the read only memory (ROM) used to provide image data and occasionally other types of information such as motion codes or the equivalent of logic functions.

temporary games use a read only memory (ROM) to store image information. In this case, the score counter operates a set of ROM address multiplexers which look up the locations of the stored numeral images in the ROM and read the data out. This information is then sent to the video summation network to be serialized prior to display.

Since this variety of game has a ROM and a ROM address circuit anyway, the designer can not only minimize circuitry by storing all the numbers in memory, but he can also produce much more appealing numeral shapes.

### 3.9 Memory Circuits

There are a number of different techniques using a ROM or ROMs to store image data depending on the theme of the machine and other considerations. However, the diagram above illustrates a fairly typical arrangement where a single 2048 by 8-bit (16K) ROM is used to store the data required for all the images.

In this case, a number of multiplexers are employed to initially multiplex both player's information together. The result from the first set of multiplexers is a set of signals which corresponds to the first player's score some of the time and the second player's score the rest of the time. The first player's object signal is similarly mixed with the second player's signal. However, the playfield data is not actually multiplexed together. The purpose of the playfield multiplexer is to provide a convenient point at which one of two different playfields may be selected. This way, when players become bored with a particular playfield lay-out and earnings begin to drop, the operator need only adjust a single switch to access a different set of playfield data from the ROM and cause a totally different playfield to be displayed on the CRT.

Once the two sets of signals are initially multiplexed together, they are entered into another set of multiplexers which is the actual sub-circuit that forms the binary address used to read out the data

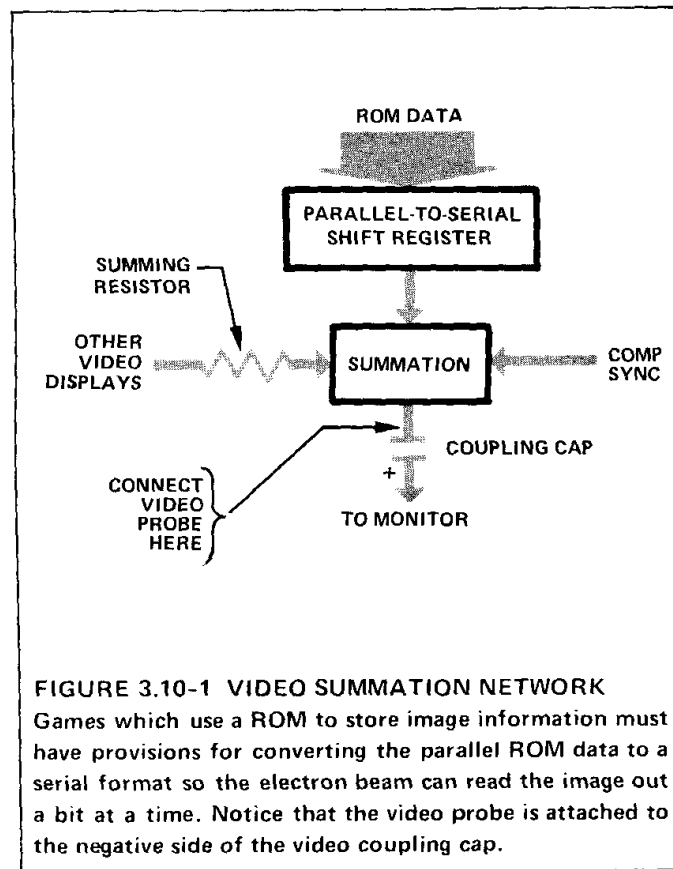
from the ROM. Often, ROMs are constructed where the memory is divided into *pages* and this becomes very handy when dealing with different types of information stored in a single ROM. Consequently, one type of address may be used to access the page on which the score numeral information appears while another address is used to access one of the two pages which store the play-field data.

In spite of the fact that we have drawn only a single address line entering the ROM, be aware that there are actually a number of parallel address lines (i.e. 11 lines for a 16K ROM). The states of these lines form a binary number which accesses a particular word or row of eight data bits from one of four internal pages of 512 words. All eight bits then appear *simultaneously* from the eight ROM outputs. These data are sent directly to the video summation network where they are converted to a serial format so they may be displayed a bit at a time on the CRT.

### 3.10 Video Summation Networks

The actual construction of a summation network will vary widely depending on the specific type of architecture used to implement a game, however all share the same basic function. The summation network is primarily used to *add or sum together* all the various video displays with the composite sync signal before the information is sent to the monitor. If a ROM is used in the game to store image data, its parallel outputs must be converted to serial form and this becomes an important summation network function in games using memory.

Notice that a *summing resistor* has been placed in the line between the "other video displays" and the summation network itself. The function of this resistor is to regulate the final brightness of the display by controlling the amplitude of that part of the final output signal. By using different values of resistors for different video displays, it is possible to obtain objects which are black, white or almost any shade of grey in between.



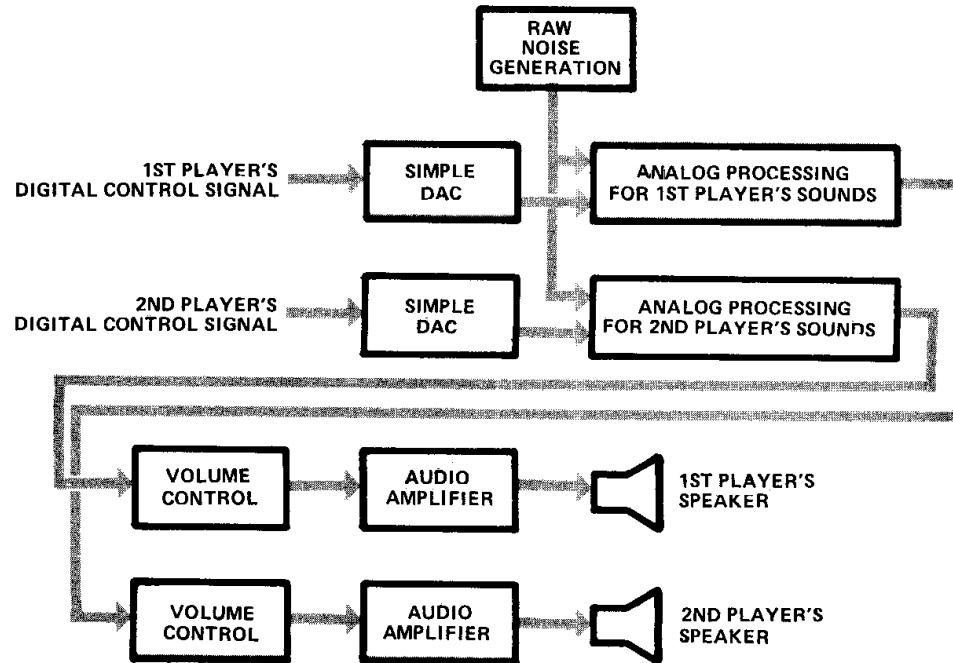
**FIGURE 3.10-1 VIDEO SUMMATION NETWORK**  
Games which use a ROM to store image information must have provisions for converting the parallel ROM data to a serial format so the electron beam can read the image out a bit at a time. Notice that the video probe is attached to the negative side of the video coupling cap.

The *coupling capacitor* in the final output line is also an important component because the video probe is attached to the *negative side* of this cap. Consequently, one of the first things to do when poking around in a new game is to locate the PCB position of this cap.

### 3.11 Sound Circuits

Sound circuits are one of several areas which show little specific similarity from game to game. This is a natural result of designers needing very different noises for play functions of games where the theme of the machines varies greatly. For example, a shooting game requires a much different sound circuit design than a driving game. In spite of this, there are still a number of functions which must be performed by any sound circuit regardless of its specific design.

Generally, a raw noise waveform is first generated by using the *zener characteristics* of a selected noise transistor. This random or *white noise* wave-



**FIGURE 3.11-1 TWO PLAYER SOUND CIRCUIT** The earliest video games (i.e. all paddle type games) use clock submultiples directly to generate the tones for the various functions. Contemporary games, however, need to produce more complex sound wave forms to simulate the sounds of diving airplanes, race cars, explosions, etc. The generation of these more complex, custom tailored waveforms require an analog circuit. But since the input to the circuit is digital in nature, a digital-to-analog conversion (DAC) must first be performed.

form is then sent to various other sub-circuits for processing into more refined waveforms which are identifiable with a particular real-world situation (i.e. a shot or airplane diving sound). However, the noise must also often change with how the player has set his controls or other parameters such as the level of score a player has reached. Since the format of these control inputs is generally a digital one, they must first be converted into an analog waveform and this is accomplished by a digital-to-analog converter which, in some cases may be nothing more than a single resistor.

trimpot where the operator may adjust the volume of the final sound output or balance the player's individual sound levels. Then this waveform is entered into an integrated audio amplifier which amplifies the sound waveform to a level high enough to drive the speaker coil.

Once the final sound waveform is developed and controlled, it is generally first passed through a

**4**

**power supplies**

## 4.1 Introduction

Depending on the architecture and requirements of the actual game, the power supply for the computer may be as simple as a single +5 volt supply using little more than an LM309K or it may be a much more complex system creating several voltages to power more esoteric componentry such as analog hybrids and audio amplifiers.

Digital integrated circuits are very picky about the power they use. The supply must be protected against line voltage fluctuations, load fluctuations and must be capable of delivering a constant level of voltage even when these conditions are present. Marginally designed power supplies are often the cause of many annoying problems which can be quite difficult to iron out, short of completely redesigning the supply. For example, even though a supply might output a nicely regulated voltage when under little load, it may not be capable of delivering an adequate amount of current when the total amount of circuitry in the system is increased. The result is likely to be a supply which drops out of regulation and this is manifested in video games by a nasty *hum bar* rolling up or down the monitor CRT.

## 4.2 The LM309K

The LM309K is an integrated voltage regulator

designed specifically for TTL devices and it has enormously simplified power supply design and increased cost-effectiveness of the supply for a single LM309 replaces several transistors, a zener diode, an operational amplifier and a handful of resistors and capacitors. The 309 is fabricated on a single silicon chip and is capable of delivering more than 1 amp if adequate heat sinking is provided.

## 4.3 A Simple +5 Supply

Figure 4.3-1 illustrates a typical +5 volt supply adequate to power a relatively small video game computer. The transformer first steps the 117 VAC line down to about 16.5 volts AC at the transformer secondaries (Figure 4.3-2). This sinusoidal waveform is then full-wave rectified by diodes D1 and D2 which average the 16.5 VAC to about 8 volts DC. The large 8000 $\mu$ F capacitor C1 (this is the very large cap usually found at the edge connector side of a game computer PCB) filters this waveform so the resulting pulsating DC has small peaks and valleys centered about the 8 volt level (Figure 4.3-2b). The pulsating DC waveform is then fully regulated to a constant +5 volts by the LM309 and decoupled by capacitor C2 to prevent load fluctuations from disturbing the operation of the regulator (Figure 4.3-2c).

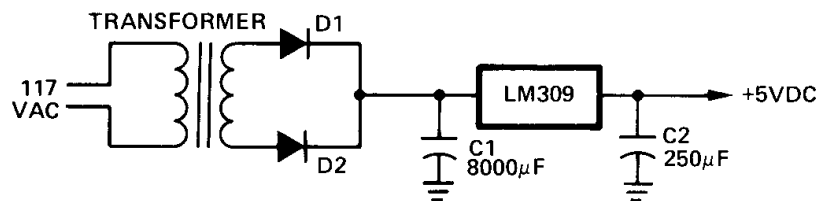


FIGURE 4.3-1 SIMPLE POWER SUPPLY Since early video games required only a single +5 volt supply, this simple circuit fulfilled the power requirements for the entire game. The use of an integrated voltage regulator (LM 309) has greatly simplified power supply design.

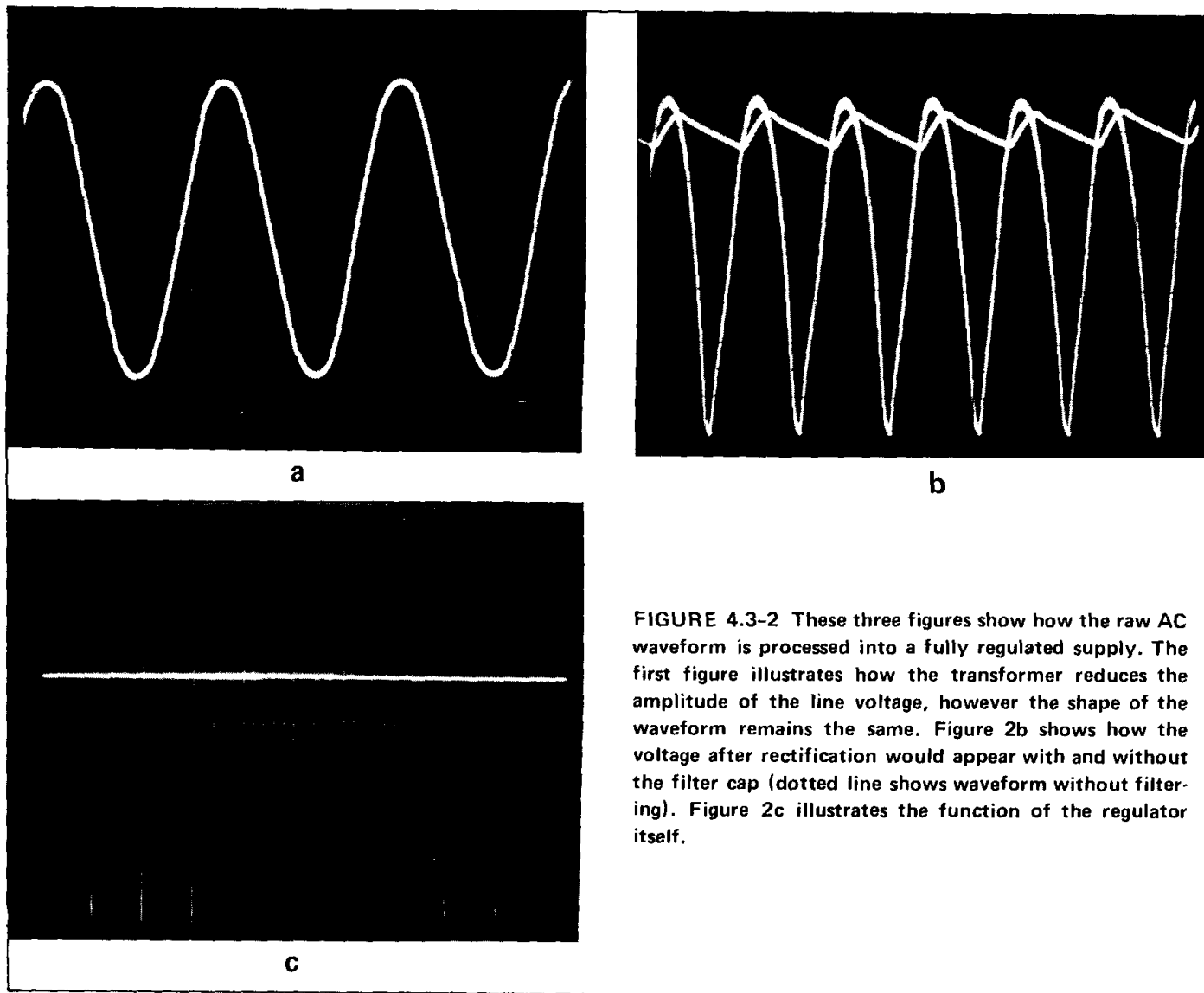


FIGURE 4.3-2 These three figures show how the raw AC waveform is processed into a fully regulated supply. The first figure illustrates how the transformer reduces the amplitude of the line voltage, however the shape of the waveform remains the same. Figure 2b shows how the voltage after rectification would appear with and without the filter cap (dotted line shows waveform without filtering). Figure 2c illustrates the function of the regulator itself.

#### 4.4 A More Complicated Supply

The simplicity of the foregoing supply is due to the use of the LM309 which eliminates the need for a lot of cumbersome circuitry. The next example produces a fully regulated voltage without the use of an integrated regulator and we have included this example to illustrate how many more components are required in this case.

The circuit in Figure 4.4-1 is divisible into two parts. The +5 volt section is nearly identical to the one in Figure 4.3-1 with the exception that a 7.5 Ohm, wire wound resistor (R1) has been wired across the input and output lines of the regulator. This quick fix expands the loading capacity of the circuit by 2/3 amp so that a larger number of ICs

may be powered.

The second portion of the power supply generates two voltages (+18V and -12V) but the most significant aspect to notice is that fact that an integrated regulator is not used. To create the -12 volt supply, the 25 VAC waveform from the transformer is full-wave rectified by diodes D1 and D2. The resulting waveform is negative with respect to ground because the cathodes of the diodes are wired to the transformer secondaries. This voltage is entered at the collector of pass transistor Q1 and is also passed through R2 and Zener diode Z1. The Zener establishes a *reference voltage* at the (+) input of the operational amplifier A1 which is used as an *error amplifier* in this circuit. The amplifier compares the reference voltage at the (+)

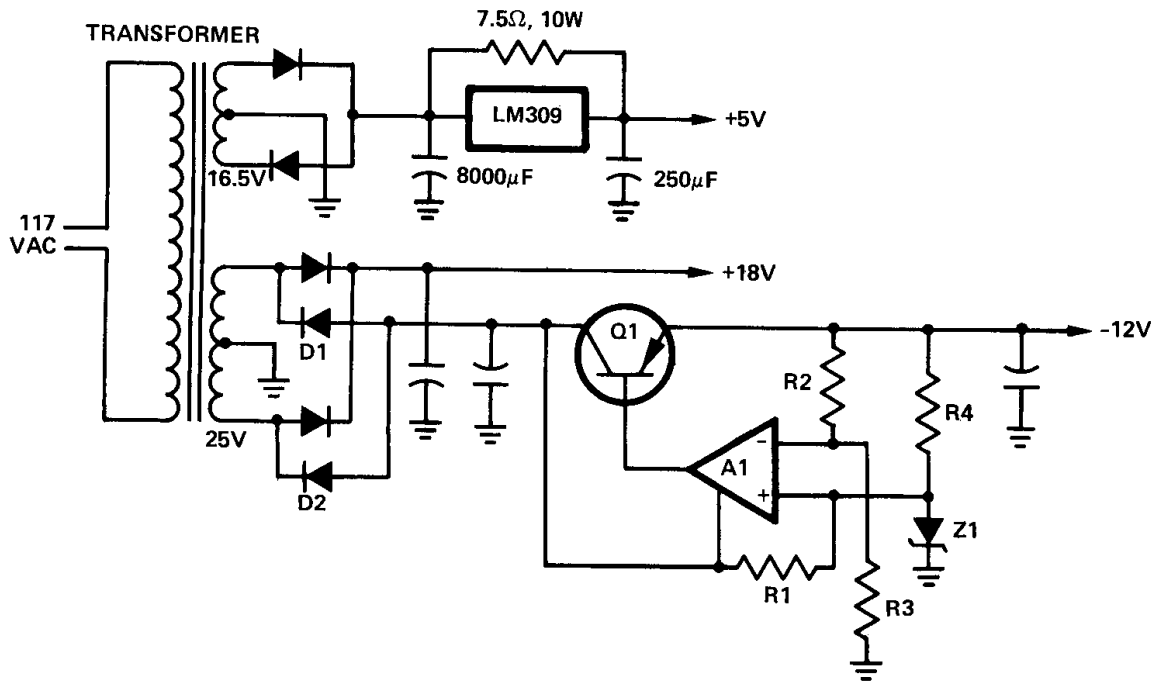


FIGURE 4.4-1 A MORE COMPLEX SUPPLY In addition to the normal +5V, this circuit also outputs regulated +5V, this circuit also outputs regulated -12V and an unregulated +18V supply. This is an interesting circuit for it demonstrates how much circuitry is required to produce a regulated supply when an integrated regulator cannot be used.

input with the actual voltage at the (-) input and uses the difference as a control for the pass transistor.

A voltage divider network constructed from R2 and R3 is connected to the emitter of Q1. When the output of Q1 is at the required -12 volt level, the junction of the network will be at -10 volts. Since this node is connected to the (-) input of the amplifier, the result is a zero error voltage. But since we are actually dealing with a pulsating DC waveform, the voltage is usually either above or below the required value. So, if the voltage from Q1 should fluctuate to -13 volts, the junction of R2 and R3 would increase to -10.8 volts and the base of Q1 will become more positive with respect to the emitter. This shuts Q1 off temporarily until the voltage returns to the correct level.

#### 4.5 Another Approach

This supply follows another common scheme wherein a pair of differential transistors sense voltage and load and drive a pass transistor through an amplifier transistor. Initially, the 6.3 VAC waveform from the transformer secondaries is full-wave rectified by CR20 and CR21 and filtered by C42 so the resulting waveform is pulsating about the 7 volt level. Q21 and Q24 form the differential pair which controls the emitter-follower amplifier transistor Q23 which the actual device that operates pass transistor Q22. In this circuit, Q20 performs the current limiting function.

The sub-section formed by CR24 and Q24 controls the voltage with respect to load. If the output voltage should rise above the +5 volt level, CR24 con-

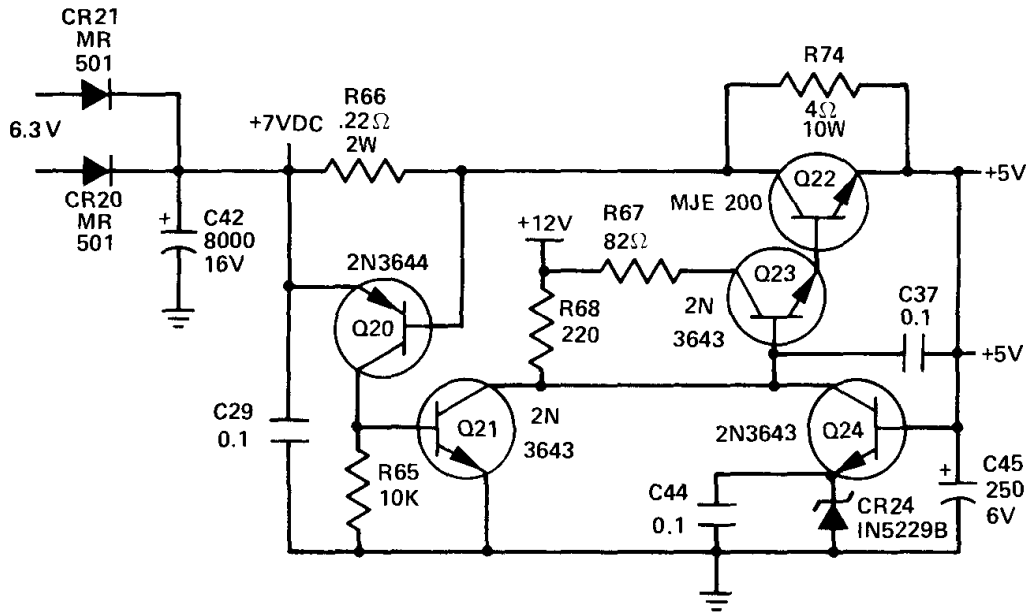


FIGURE 4.5-1 ANOTHER APPROACH This high current supply uses a pair of differential transistors to sense both voltage and load. In this case, the pass transistor is driven by an amplifier transistor.

ducts and Q24 starts to draw current from the base of Q23 thereby turning Q23 off which in turn shuts pass transistor Q22 off and reduces the voltage. Q20 performs the current limiting feature in conjunction with Q21. If more than 3 amps are drawn through R66, the base-emitter junction of Q20 is forward biased and Q20 then conducts which pulls up the base of Q21. This causes Q21 to conduct and it "robs" base current from Q23, shutting it off and thereby limiting current to a safe level.

#### 4.6 Ripple In The +5 Line

Occasionally, you may have seen a game where a wide dark horizontal bar rolls up or down the monitor CRT. This bar is known as a *bum bar* and may be produced by any one of a number of different failures. However, the most common cause is a *ripple* in the +5 volt supply.

The ripple causes modulation of the electron beam as it scans the CRT and produces a darkening of the affected lines. The width of the ripple controls how many of the lines are modulated. The ripples occur at a frequency which — due to the inherent weaknesses of electrical generating equipment and other factors — is constantly shifting out of phase with the frame rate of the monitor. When introduced into the video line of the monitor, the phase differential between the ripple and the monitor frame rate controls the rate at which the ripple rolls up or down the CRT.

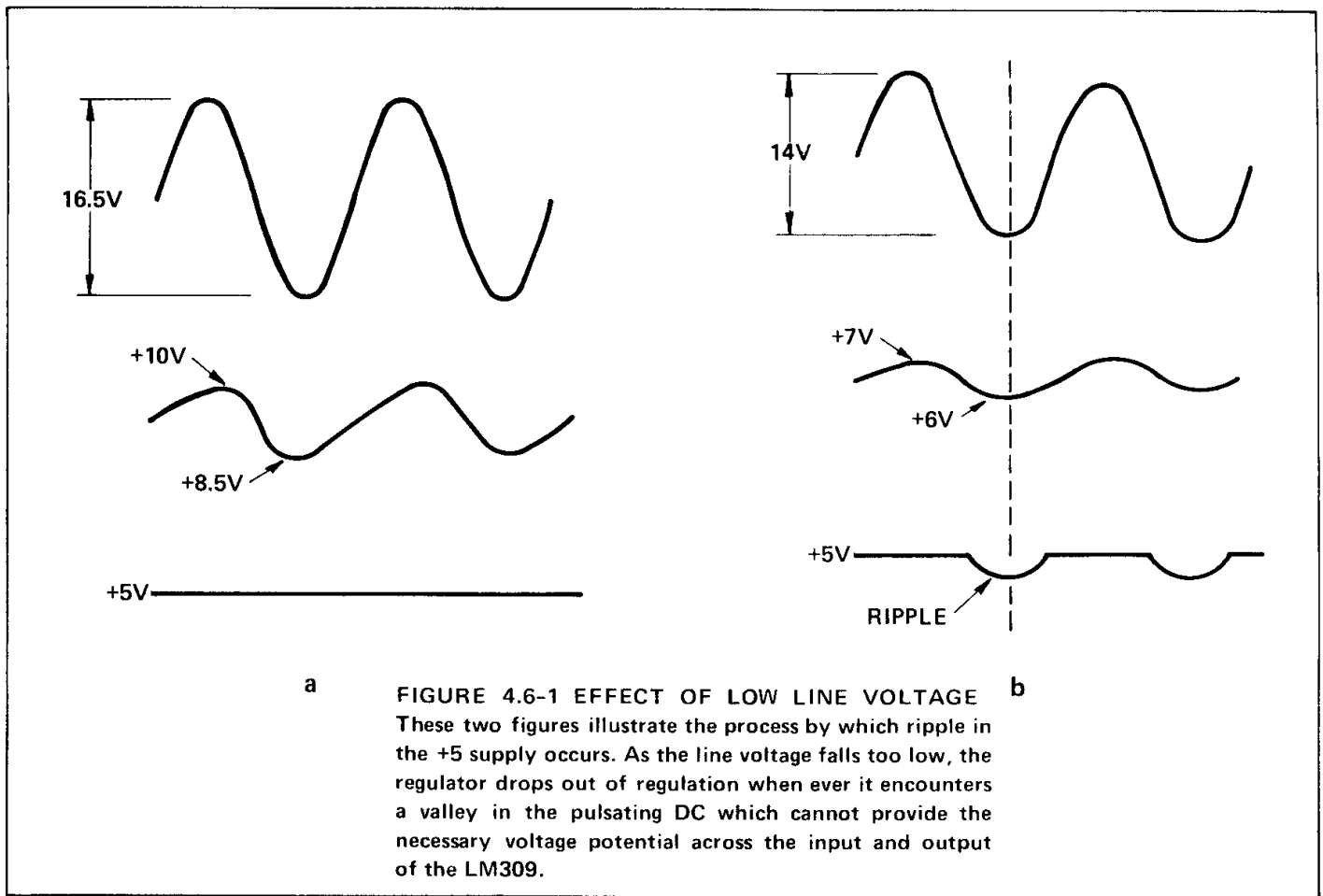
Ripple can be introduced into the video line at either of two places. Generally, the video output of the computer is tied HI to a pull-up. Also, since all the ICs of the computer are tied to the same +5 volt supply, ripple may be passed through each integrated circuit so even the video information itself contains the ripple.



Ripple may be caused by anyone of a number of related power supply deficiencies which include low line voltage, resistive connections at the transformer or even by overloading of the supply. To illustrate how a ripple develops, let's analyze what happens as the line voltage drops.

Normally, the 16.5 VAC waveform from the transformer secondary is full-wave rectified and filtered so the resulting DC waveform has peaks and valleys at about +10 volts and +8.5 volts respectively (Figure 4.6-1). But if the line voltage drops, the secondary drops to 14 volts or so, the peaks and valleys may appear at 7 and 6 volts (Figure 4.6-1b).

This might not appear particularly significant unless you consider that an LM309 *must* have at least 2 volts potential difference between the input and output of the regulator. The output of the regulator will always be at 5 volts as long as the lowest valley of the pulsating DC remains above the 7 volt level. But, after the line voltage has dropped enough, the valleys begin to appear below the 7 volts and minimum potential difference of 2 volts no longer exists. Therefore, each time a really low valley comes up, the regulator momentarily drops out of regulation and introduces a small ripple in the otherwise stable +5 volt line.



**FIGURE 4.6-1 EFFECT OF LOW LINE VOLTAGE**  
 These two figures illustrate the process by which ripple in the +5 supply occurs. As the line voltage falls too low, the regulator drops out of regulation when ever it encounters a valley in the pulsating DC which cannot provide the necessary voltage potential across the input and output of the LM309.



5

**master timing**

## 5.1 Introduction

It would be difficult to maintain that any one circuit of a game computer is more important than another for the game simply will not be operational unless all the major circuits are functioning correctly. In spite of this fact, the master timing circuitry which creates the conditions by which the game computer can operate the TV monitor in such a way that coherent images can be displayed. Since display is the primary function of any video game, the circuitry enabling this function takes on a greater significance.

Sync circuitry may also be one of the areas most difficult to fully understand because so many operations occur simultaneously and all operations are referenced to the operation of the monitor. The operation of both the monitor's scan and the rest of the computer is intimately tied in with the operation of master timing.

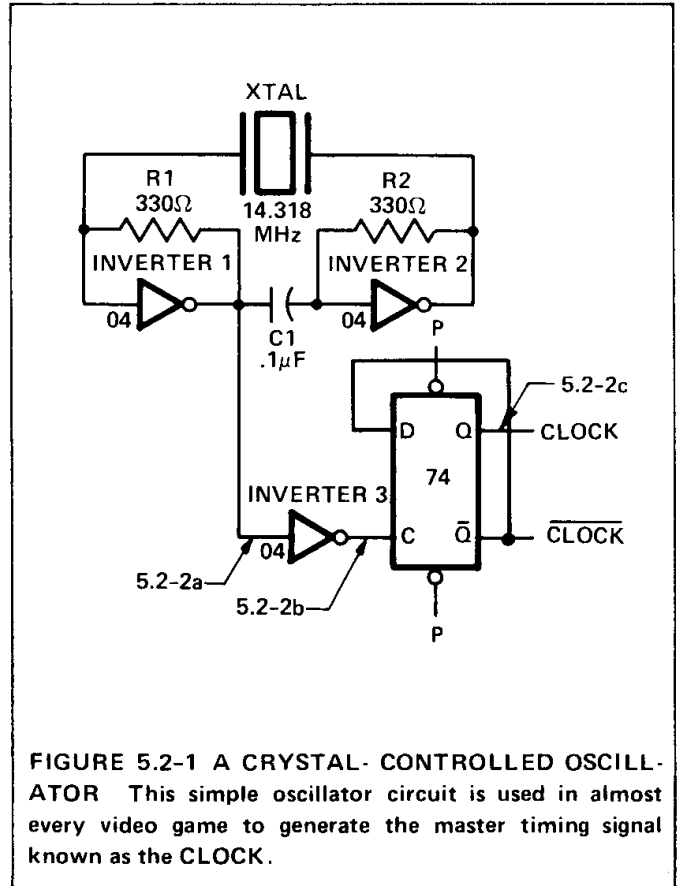
The primary reason for discussing the two types of raster scans in Chapter 2 was to build a basis for understanding the operation of the master timing circuitry, because unless you know the parameters around which a circuit is designed, its actual operation will never make sense. So, now that you basically understand how monitors function we can delve into the actual operation of the two types of circuits which generate the non-interlaced and interlaced versions of the raster scan. Sections 5.2 through 5.8 deal with the more common non-interlaced scan while sections 5.9 through 5.12 contain an analysis of a typical interlaced game.

As we pointed out in the introductory section on master timing (Section 3.6), there are a number of very important functions this group of circuits performs. First of all, it generates CLOCK, the master timing signal used everywhere in the computer. But more important, the sync sections develop sync signals used to synchronize the operations of the monitor and the computer as well generating clock submultiples which are also used in just about every corner of the PCB to develop images and provide accurate timing signals.

## 5.2 The Oscillator

It all starts with the oscillator which is one of those standard circuits that almost never changes from game to game. Although other systems may employ clock generators using other devices such as 555 timers, all video games use an actual quartz crystal oscillator because the crystal has the most stable operating characteristics and produces a signal with the greatest possible accuracy.

Figure 5.2-1 illustrates a typical oscillator using a 14.318 MHz crystal, however manufacturers have recently begun to use other frequencies as well for timing considerations and to avoid emitting problematic RF which would otherwise tend to interfere with certain communications networks.



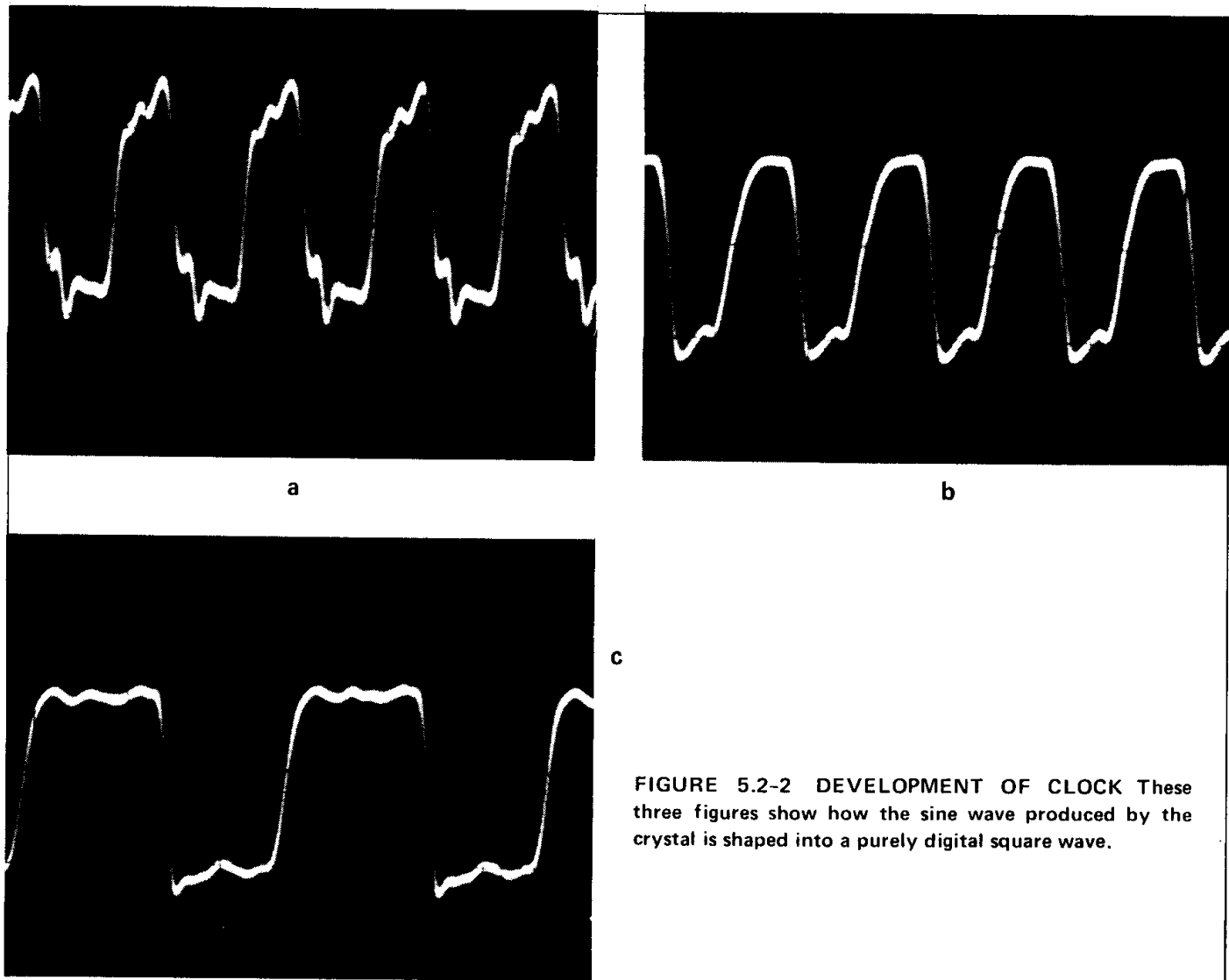
The operation of the oscillator might be easier to understand if you view the circuit with a capacitor substituted for the crystal. When the circuit is powered up, a positive voltage is produced by Inverter 2 creating a negative potential at the op-

posite plate of the crystal which acts like a capacitor. This negative potential is then inverted positive by Inverter 1 and the resulting positive spike is coupled through C1 to Inverter 2 where it is again inverted negative so that the circuit oscillates back and forth.

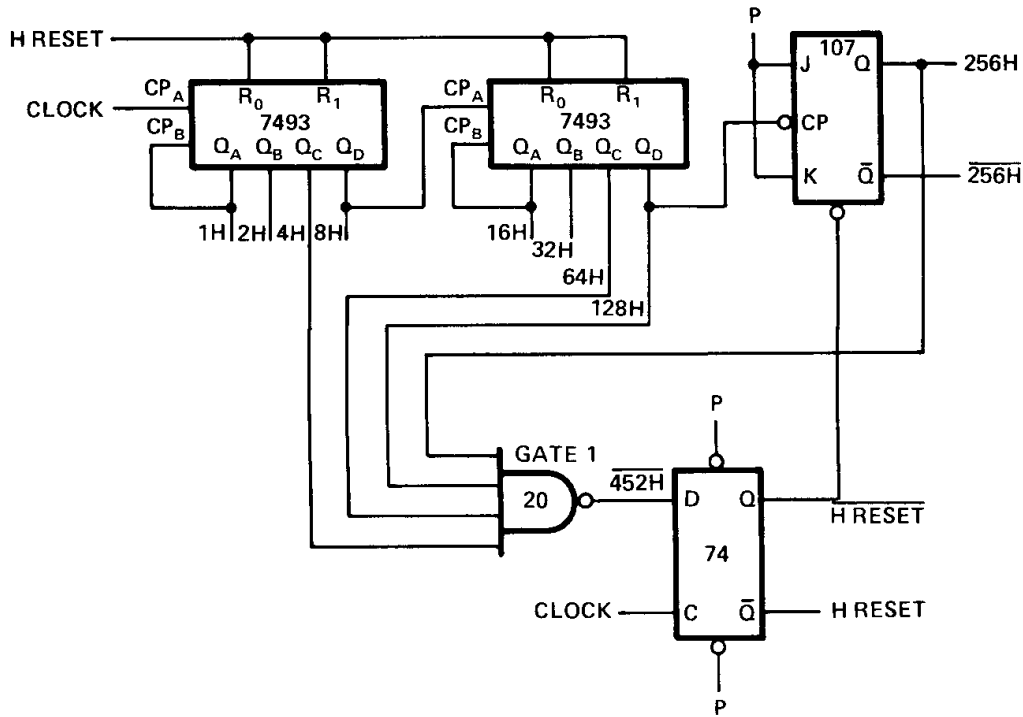
Due to the physical characteristics imparted to the crystal in the manufacturing process, it has a point at which its impedance is lowest causing it to *resonate* at a particular frequency which happens to be 14.318 MHz in this case. Since quartz has a very low coefficient of thermal expansion, quartz crystals are affected very little by temperature changes. Furthermore, quartz also responds well to supply voltage variations so that even when the voltage changes, the point of resonance remains fairly constant. The physical shape or *cut* given the crys-

tal when manufactured allows it a certain mass and molecular orientation which governs the temperature range in which the crystal is stable and also controls its resonant frequency.

Before it reaches Inverter 3, the output of the oscillator is an approximate *sine wave* (Figure 5.2-2a). Inverter 3 tends to *square up* this waveform slightly, however its main function is as a *buffer-amplifier* (Figure 5.2-2b). This inverted waveform is entered into the flip-flop which has two functions. First, it finishes squaring up the waveform so it has the *square wave* shape required by digital circuits (Figure 5.2-2c). Second, since the flip-flop is wired in the toggle mode, the input rate is divided in half to 7 MHz and this output is known as the **CLOCK**.



**FIGURE 5.2-2 DEVELOPMENT OF CLOCK** These three figures show how the sine wave produced by the crystal is shaped into a purely digital square wave.



**FIGURE 5.3-1 HORIZONTAL SYNC CHAIN** This counter chain divides each line of the raster into 452 points, each capable of being illuminated separately by the electron beam.

### 5.3 Horizontal Submultiples

The counter chain consisting of the two 7493 counters and the flip-flop in Figure 5.3-1 generates submultiples of CLOCK by a process of *successive division*. In all but interlaced sync chains, the clock must be divided a total of  $2^9$  times so two 4-bit counters are used to provide the first eight bits and a flip-flop is tacked on the end for the ninth bit. But before getting into how all this actually works, notice that both counters are wired as 4-bit devices by connecting the  $Q_A$  output back to the  $CP_B$  input and that the flip-flop is wired in the toggle mode. As long as the reset line ( $R_0$  and  $R_1$ ) to the counters is LO, they will count clock pulses.

On the falling edge of the first clock pulse, the  $Q_A$

output (1H) rises HI. On the falling edge of the second clock pulse,  $Q_A$  returns LO and  $Q_B$  (2H) rises HI and so on until  $Q_D$  (8H) rises HI. Since the  $Q_D$  output is connected to the clock input of the next counter, it counts 8H pulses and successively divides this count to produce four additional submultiples known as 16H, 32H, 64H and 128H (characterizations of these waveforms appear in Figure 1.2-1). The flip-flop is simply wired up as the ninth bit of the chain and it toggles when it sees the falling edge of the  $Q_D$  output of the second counter to produce the last submultiple, 256H.

The submultiples can be examined by a number of instruments. Unfortunately, the logic probe will reveal HI, LO and Pulsing where the only noticeable difference between outputs is a progressive

decrease in symmetry further down the line. However a scope will reveal waveforms which correspond exactly to the characterizations in Figure 1.23-2.

This brings us to another instrument which is very useful in probing these kinds of circuits: *the video probe*. A video probe is nothing more than a 4.7K resistor attached to a jumper which has an IC clip on one end and a test prod on the other. The resistor can be hidden in either the prod or the clip to keep it out of the way.

The video probe is used primarily for examining video signals but it can also be used as a *sound probe*. When used to probe for noise, the clip is attached to the input of the audio amplifier and the test prod can then be used to hear any signal fast enough to produce a tone. Generally, a noise probe is used to check sound circuits, but it can also be used to “hear” sync chains working because — as you move down the chain — the tones become successively lower at each division.

But the most valuable use of the video probe is in examining *video* signals used in the development of more processed signals where the developmental signals you need to check are not visible on the monitor CRT. But, if you connect the probe clip

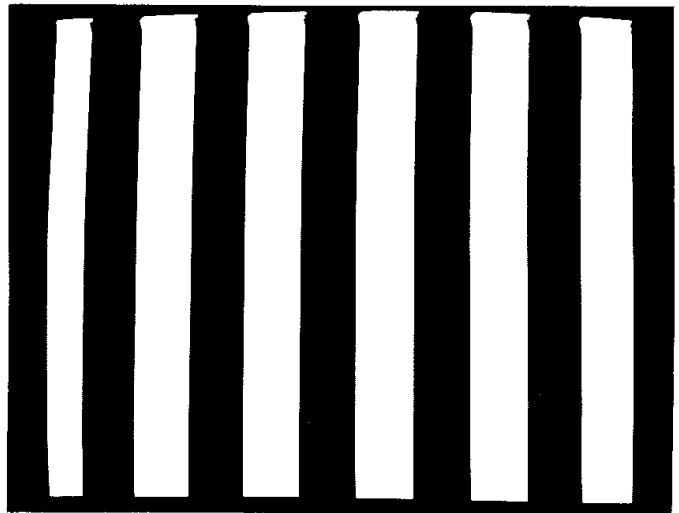


FIGURE 5.3-3 SIGNAL 32H Viewed with a video probe.

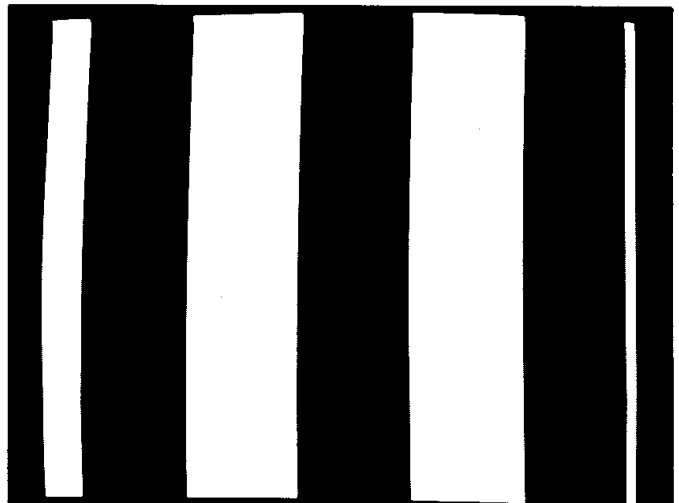


FIGURE 5.3-4 SIGNAL 64H Viewed with a video probe.

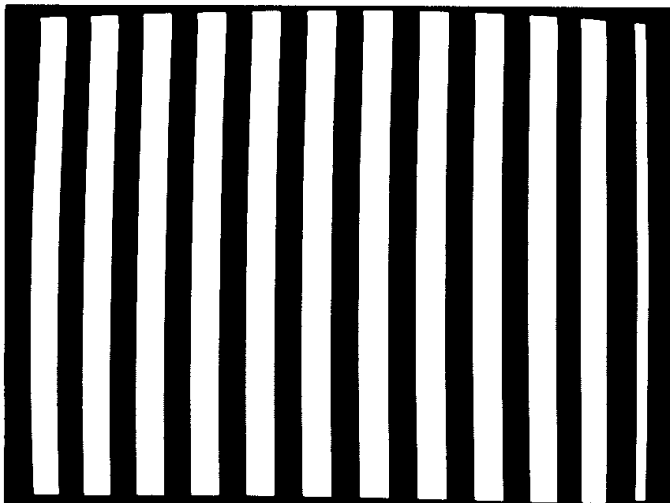


FIGURE 5.3-2 SIGNAL 16H Viewed with a video probe.

to the negative side of the coupling capacitor found in the video summation section (Figure 3.10-1), the signal present at any node you touch with the prod will be injected directly into the video output line. If the examined signal is of a video nature, meaning it is neither too fast nor too slow, it will be displayed as an identifiable shape on the monitor CRT.

For example, if we examined signals 16H, 32H and 64H in the circuit illustrated in Figure 5.3-1, the

patterns illustrated in Figures 5.3-2, 3 and 4 would be visible. In these photographs, the light areas correspond to the periods of time when the signal is HI and the dark areas represent LO periods. You can see that, as each signal is divided in half, the width of the bars doubles. The reason for this and the fact that *horizontal* signals produce *vertical* bars will be more apparent in a moment, but first let's get back to the horizontal sync chain.

#### 5.4 Horizontal Reset

The HRESET pulse is used in the development of the horizontal sync pulse but it has the equally important function of resetting the counter chain.

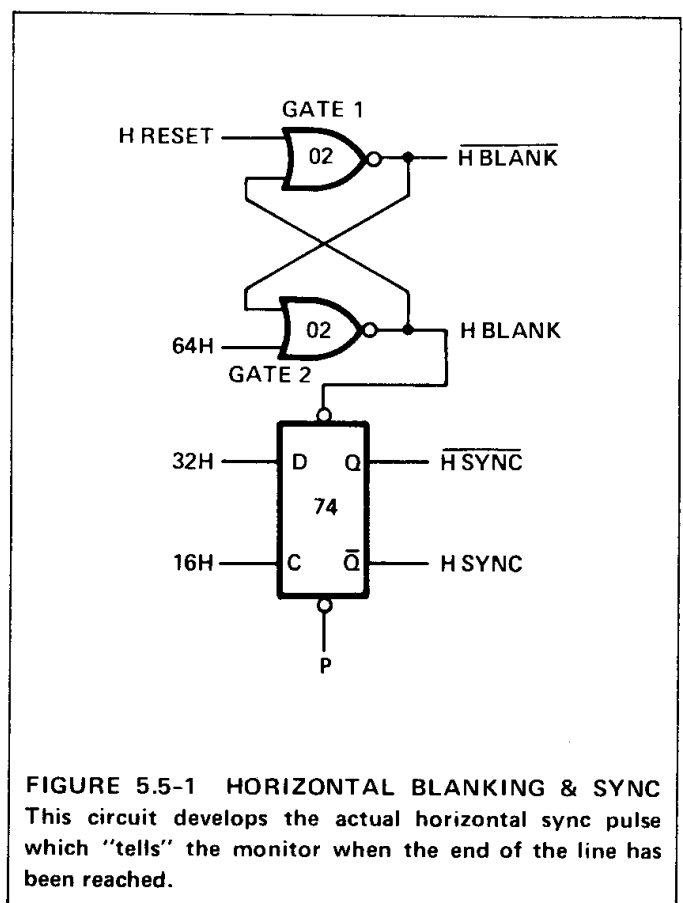
In any case,  $\overline{\text{HRESET}}$  is formed by the addition of 256H, 128H, 64H and 4H at Gate 1. Since the gate can open up only when all inputs are HI, the output of this NAND drops LO on the 452nd clock pulse (256 plus 128 plus 64 plus 4 equals 452) and stays LO until the least submultiple 4H drops LO four clock pulses later. The LO pulse from Gate 1 is clocked through the flip-flop by the next clock pulse (which actually occurs on the 453rd count) so that a LO-going  $\overline{\text{HRESET}}$  pulse occurs every 452 clock pulses. Since HRESET is connected to both reset pins of the counters and  $\overline{\text{HRESET}}$  is connected to the clear input of the ninth-bit flip-flop, the entire counter chain is reset back to zero on the 453rd clock pulse.

This reset business is very significant because we have now created a period of time extending from the beginning to the end of every raster line which is divided exactly into 452 separate clock pulses or *points* where each point is capable of being individually illuminated by the electron beam. As a matter of fact, any point along the line can now be described by adding up the appropriate clock submultiples, just like the reset pulse was generated. And, this is also the reason for the light and dark horizontal bars in the previous photographs. Take 64H for example. This signal is LO for the first 64 clock pulses of the line, so that portion is read out black by the electron beam. But 64H rises HI on the 64th clock pulse, so the next 64 points appear

light. When all the lines of the raster are stacked on top of one another, alternating light and dark columns result. In later chapters, we will see how images can be created by various manipulations of these submultiples.

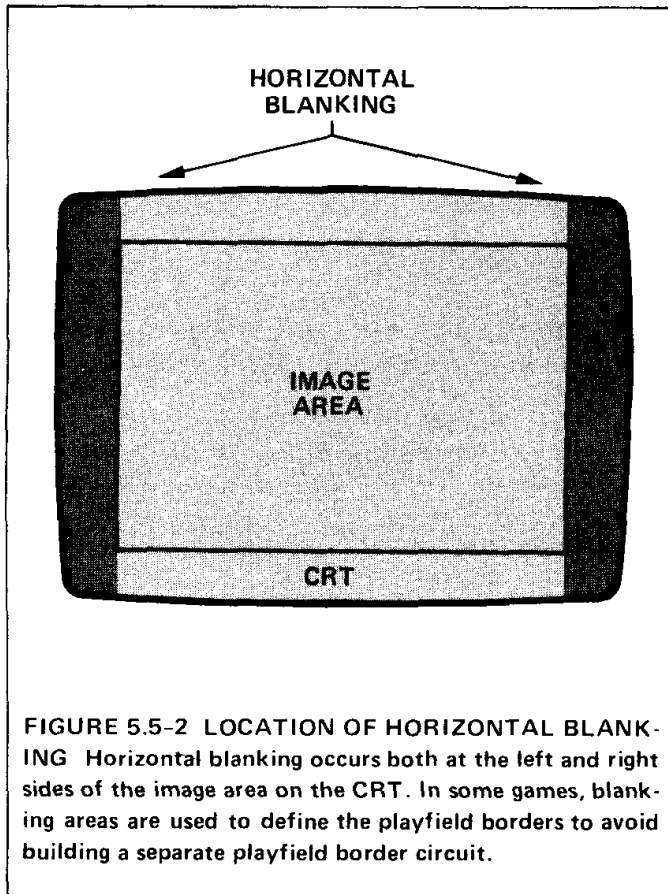
#### 5.5 Horizontal Blanking & Sync

There are many schemes for generating  $\overline{\text{HSYNC}}$ , but the one illustrated in Figure 5.5-1 is probably one of the simplest even though it develops a sync pulse slightly longer than usual. The circuit first generates HBLANK, a signal generally used in two places. First, it forms a period of time in which the sync pulse can occur, but it is also used in the motion circuitry of many games to provide a period in which new information can be entered into the horizontal motion circuit.



**FIGURE 5.5-1 HORIZONTAL BLANKING & SYNC**  
This circuit develops the actual horizontal sync pulse which "tells" the monitor when the end of the line has been reached.

HBLANK is generated by the R-S flip-flop constructed from Gates 1 and 2. The flip-flop is set at the count of 452 or 0 when HRESET pulses HI and is reset 64 clock pulses later when 64H rises

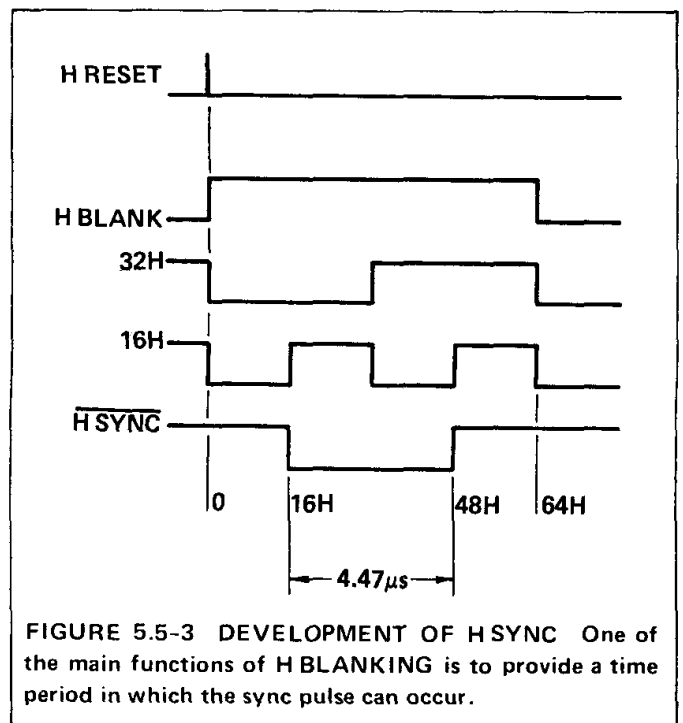


HI to produce a 64H wide HI pulse at the beginning of each line.

This pulse is connected to the preset input of the D-type flip-flop so it is enabled only during the time H BLANK is HI. When H BLANK is LO, the Q output is forced to a LO level and is disabled. At any rate, once enabled, the flip-flop is clocked by the next falling edge of 16H producing a LO from the Q output. At the next rising edge of 16H, 32H is HI so the Q output returns HI limiting the H SYNC to a pulse 32H wide which extends from 16 to 48. The electron beam is reset on the falling edge of H SYNC and only starts reading out the video after H SYNC returns HI. Consequently, the position of the actual image area on the CRT is determined by where on the line the sync pulse occurs.

## 5.6 Vertical Submultiples & Reset

The vertical sync chain in Figure 5.6-1 is almost identical to the horizontal version with two small



– but important – modifications. First of all, the chain is clocked by H RESET and since an H RESET pulse occurs with the termination of each horizontal line of the raster, the vertical chain counts lines rather than clock pulses. Also notice that the inputs to Gate 1 produce a pulse on the 261st H RESET pulse.

If we look at any of the vertical submultiples with a video probe, the display will contain alternating light and dark bars doubling in width at each successive division. But in this case, the bars are oriented horizontally (Figure 5.6-2) and the reason for this is quite similar to why the horizontal signals we looked at previously appeared vertically oriented.

Since the vertical chain counts lines, the signal 32V is LO for the first 32 lines down the CRT which causes the electron beam to read these lines out dark. But 32V rises HI on the 32nd count or 32nd line, so the next 32 lines appear light.

Now back to vertical reset. Since 256V, 4V and 1V and Nanded at Gate 1, the output of this gate drops LO on the 261st count into the chain or on the 261st line of the raster pattern. This LO is



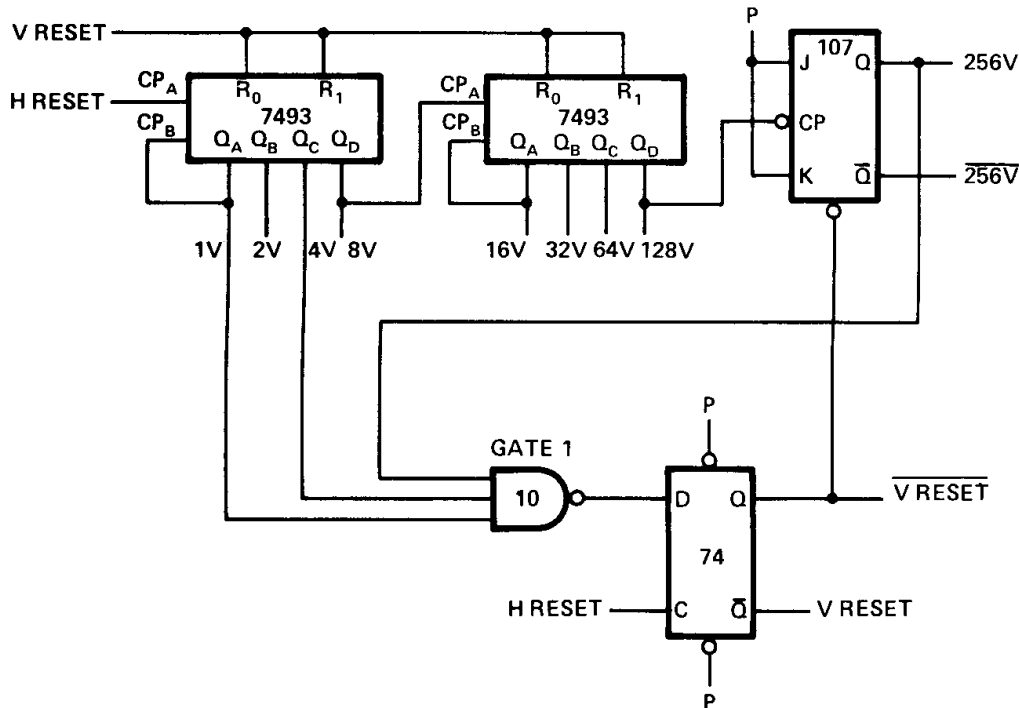


FIGURE 5.6-1 VERTICAL SYNC CHAIN This chain is virtually identical to the horizontal one, except that it counts lines per field rather than points per line.

clocked through the flip-flop by the next H RESET pulse, so the V RESET pulse actually occurs on the 262nd line and completely resets the counter chain at the end of the field.

### 5.7 Vertical Blanking & Sync

The V BLANK signal is produced in the same manner as H BLANK, except that it occurs from 0V to 16V since the R-S flip-flop is set by V RESET and reset 16 lines later by 16V.

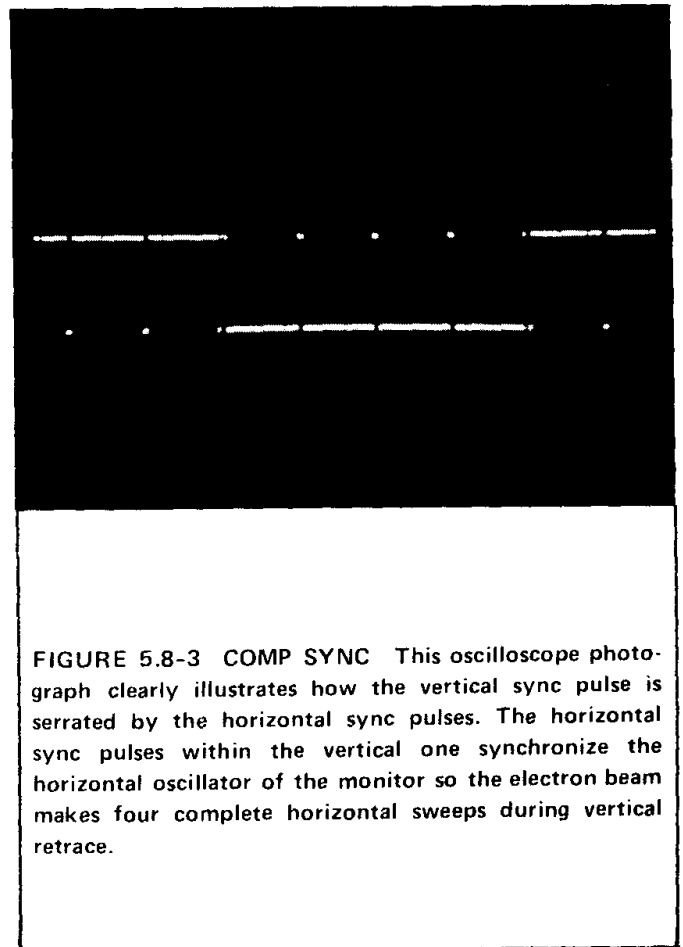
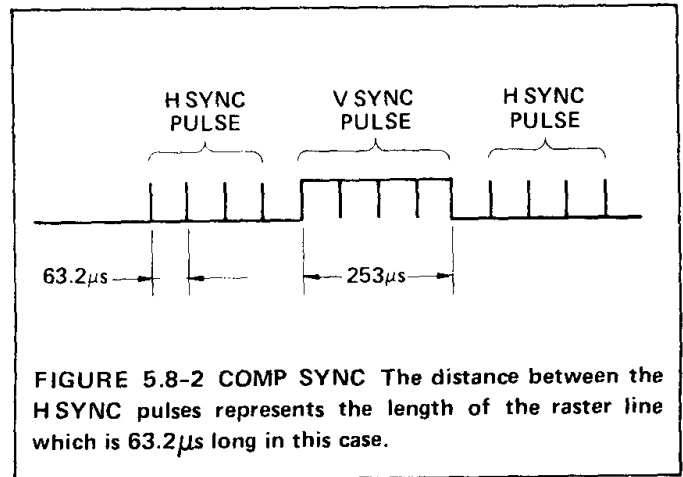
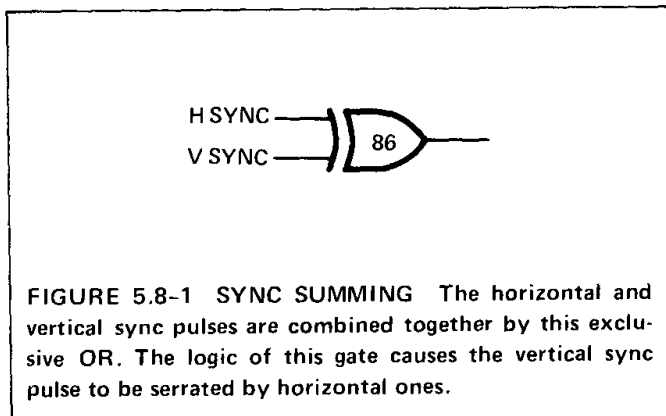
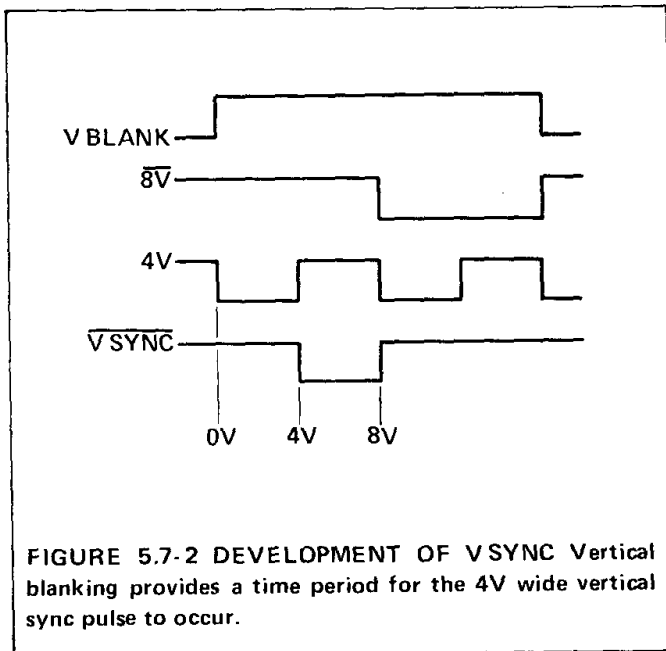
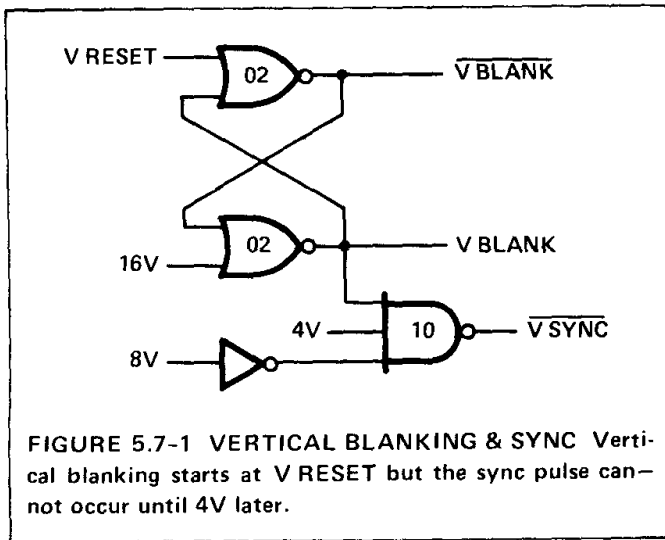
Vertical sync in this circuit is really simple. V BLANK, 4V and 8V are NANDed together to produce a LO-going pulse lasting for the HI duration of the least submultiple, 4V (between 4V and 8V).

### 5.8 Sync Summing

Both sync signals are exclusively ORed in Figure 5.8-1. Since an exclusive OR is used, the logic of



FIGURE 5.6-2 SIGNAL 32V Viewed with a video probe.



this gate causes the vertical sync pulse to be serrated by four horizontal sync pulses for the purpose of synchronizing the horizontal oscillator during vertical retrace. Consequently, the electron beam makes four complete sweeps horizontally while it is being retraced back to the top vertically (Figure 2.7-2).

Figure 5.8-3 shows how COMP SYNC actually appears on the oscilloscope.

## 5.9 An Interlaced Game

The remaining sections of this chapter are devoted to an analysis of the main timing circuitry of a fully interlaced game. When interlace is achieved using the technique described in the pages following, it is easy to spot even before the schematics are examined because this method invariably creates a fine vertical “jog” down the middle of the CRT where one can actually see that each line of the raster has been divided in half. This occurs because the halves of the raster lines do not exactly match up and, when they are all stacked on top of one another, a slight jog in the raster becomes apparent.

## 5.10 Horizontal Main Timing

The oscillator circuit of this game is constructed in the usual way, except that the final flip-flop has been omitted so the output frequency of CLOCK is *14 MHz*. This is necessary in interlaced games because each line contains twice as many separately addressable points (904 to be exact) which means the clock must count twice as fast.

One problem associated with such a high clock rate is that the familiar 7493 ripple-through counters normally used in sync chains cannot be used for an interlaced design since the outputs will contain spikes due to internal propagation delay if these counters were to be run at 14 MHz. Consequently, 9316 counters have been used in the fully synchronous divide chain which appears in Figure 5.10-1. Since the internal flip-flops of these counters are all wired to CLOCK, the outputs all change simultaneously thereby eliminating the “glitches” which would otherwise occur.

The operation of the horizontal divide chain is more or less normal except that another flip-flop has been tacked onto the end of the chain increasing the total length of the chain to 10 bits so a full count of 1024 could be reached. Normally, only

9 bits are needed because the greatest count for horizontal reset is never above 454 in a non-interlaced game and 9 bits allows a full count of 512 which is more than enough. But in this game, each horizontal line contains 904 clock pulses or points, so the length of the divide chain must be increased to accommodate the greater count.

Now we come to a more significant difference. To interlace the fields, *half lines* must be created by implementing a horizontal reset which occurs both in the center of the CRT and at the right hand edge. To generate HRESET, 256H, 128H, 64H and 2H are first ANDED at D13 to produce a LO-going pulse at 450.

This pulse is clocked through flip-flop K12 one clock pulse later so HRESET occurs at the count of 451. Notice that this signal is connected to the reset inputs of both counters and the first flip-flop, but that it is *not connected* to the second flip-flop. Therefore, HRESET clears both counters and the first flip-flop so they are reset to zero on the 451st count. Since the first half of C2 has been cleared, its Q output drops LO which toggles the second half of C2. Although the second half is actually toggled at 451, the signal does not become effective at B12 until the whole chain is reset which occurs on the next — or 452nd — clock pulse.

Since the signal 452 is LO in the left side of the CRT and HI in the right, this signal divides the screen in half creating the half-lines necessary to interlace the scan.

Figure 5.10-3 shows a numerical representation of the relationship between HRESET and 452H. The actual appearance of signal 452H is illustrated in Figure 5.10-4.

Another graphic illustration of the fact that the counter chain is reset in the middle of the CRT can be seen by examining the horizontal submultiple  $\overline{64H}$  with a video probe (Figure 5.10-5) where you can see two white vertical columns right next to each other and separated only by a very thin black line. The thin line represents the time when

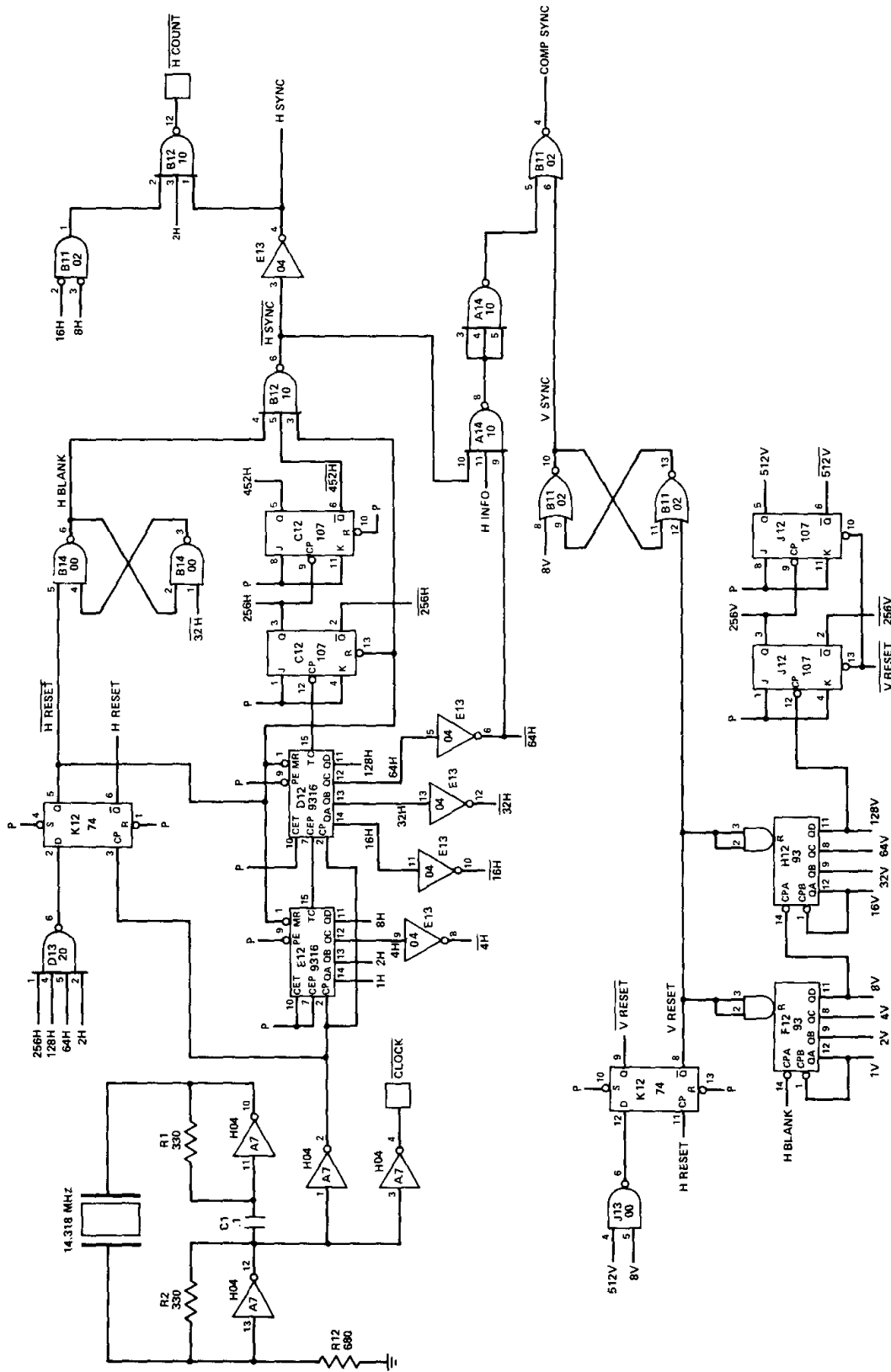


FIGURE 5.10-1 AN INTERLACED MAIN TIMING CIRCUIT In examining this circuit, you may notice several unusual items. For one thing, 9316 counters are used because of the high clock rate needed for the interlaced. Also, each chain contains ten bits to accommodate the greater count.

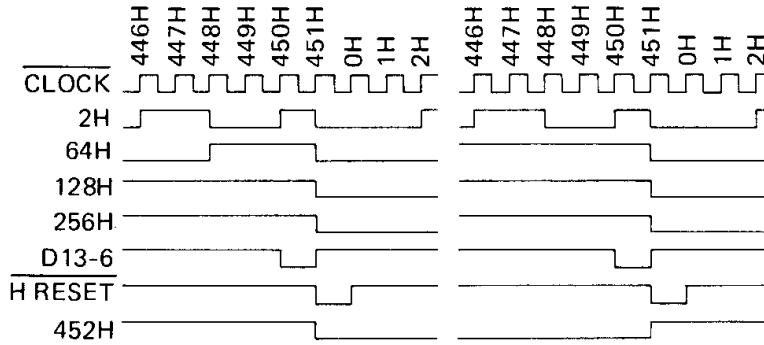


FIGURE 5.10-2 DEVELOPMENT OF H RESET AND 452H As you can see from this figure, the count is repeated twice per line since the lines must be divided in half to permit interlace.

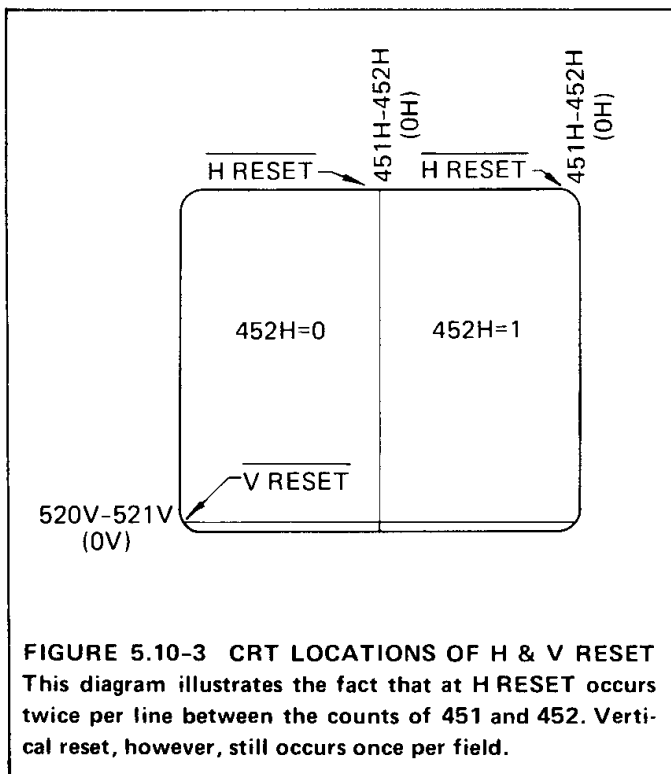


FIGURE 5.10-3 CRT LOCATIONS OF H & V RESET This diagram illustrates the fact that at H RESET occurs twice per line between the counts of 451 and 452. Vertical reset, however, still occurs once per field.

the signal has been reset. Also, notice that another reset occurs at the extreme right edge of the CRT where reset normally is found. Now, compare this picture with the video probe photograph of 64H in a non-interlaced game (Figure 5.2-4).

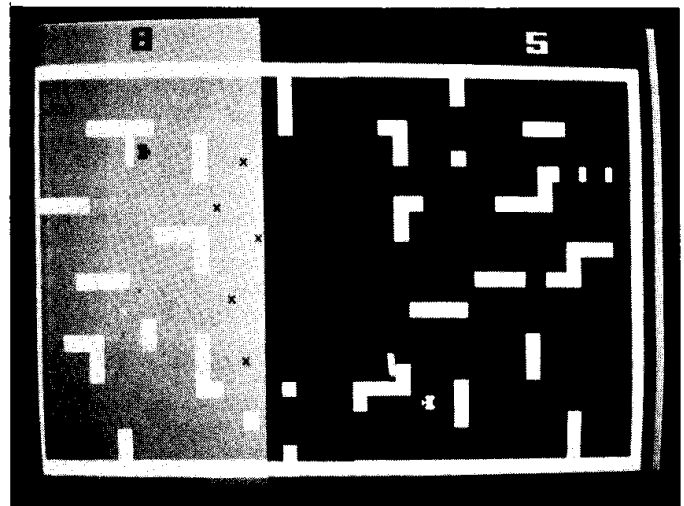


FIGURE 5.10-4 SIGNAL 452H This is the actual signal which divides the lines in half.

The process by which HBLANK, HSYNC and a miscellaneous control signal HCOUNT are developed is illustrated in Figure 5.10-6. The R-S flip-flop constructed from gates B14 is set by H RESET and reset 32H later to produce a blanking signal 32H wide between 0H and 32H.

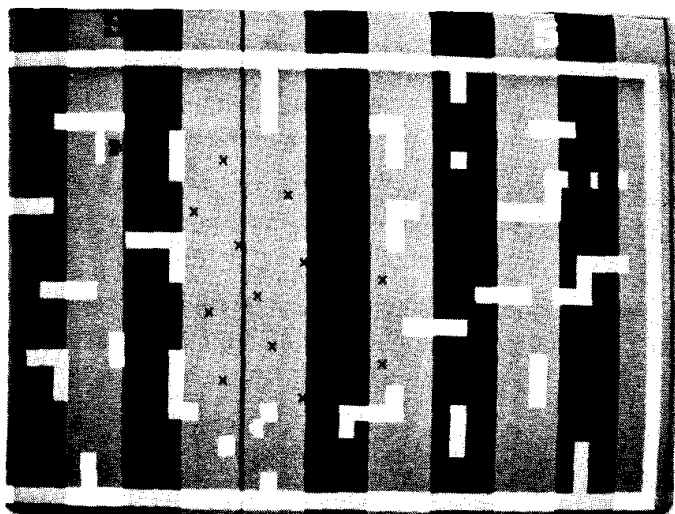


FIGURE 5.10-5 SIGNAL  $\overline{64H}$  This is a good example of a horizontal submultiple in an interlaced game. You should notice that the signal is reset in the approximate center of the CRT and this point is represented by the thin black line.

The photograph below shows how the signal H BLANK actually appears with a video probe. Notice that the signal occurs both in the center of the CRT and at the extreme right edge, where only part of the second H BLANK is visible.

Then, H BLANK,  $\overline{452H}$  and  $\overline{HRESET}$  are added together at B12 and the result is a LO-going pulse between 452H (0H) and 32H known as H SYNC, however this is not the actual signal to which the monitor is synchronized.  $\overline{HCOUNT}$  is developed by inverting H SYNC and adding it with the signal from B11-1 ( $\overline{16H-8H}$ ) and 2H to produce two LO-going pulses at the very beginning of each half-line.

The signal H INFO is used in the development of the actual horizontal sync pulse and the circuit which generates H INFO appears in Figure 5.10-8. H INFO is developed by the R-S flip-flop constructed from gates A13. This flip-flop is set when H SYNC rises HI and it is reset 128 clock pulses later when 128H rises HI to form a window 128H wide at the beginning of each line (Figure 5.10-9).

The resulting signal is gated with  $\overline{HSYNC}$  and 64H at A14 and then inverted to produce a HI-going pulse between 32H and 64H which is the actual signal to which the horizontal oscillator of the monitor is synchronized.

To clarify any confusion which might have arisen concerning the pulse named  $\overline{HSYNC}$  and the actual pulse to which the monitor is synchronized, we have included the photograph above which compares the two signals. Notice that these waveforms correspond precisely to their counterparts in Figure 5.10-9.

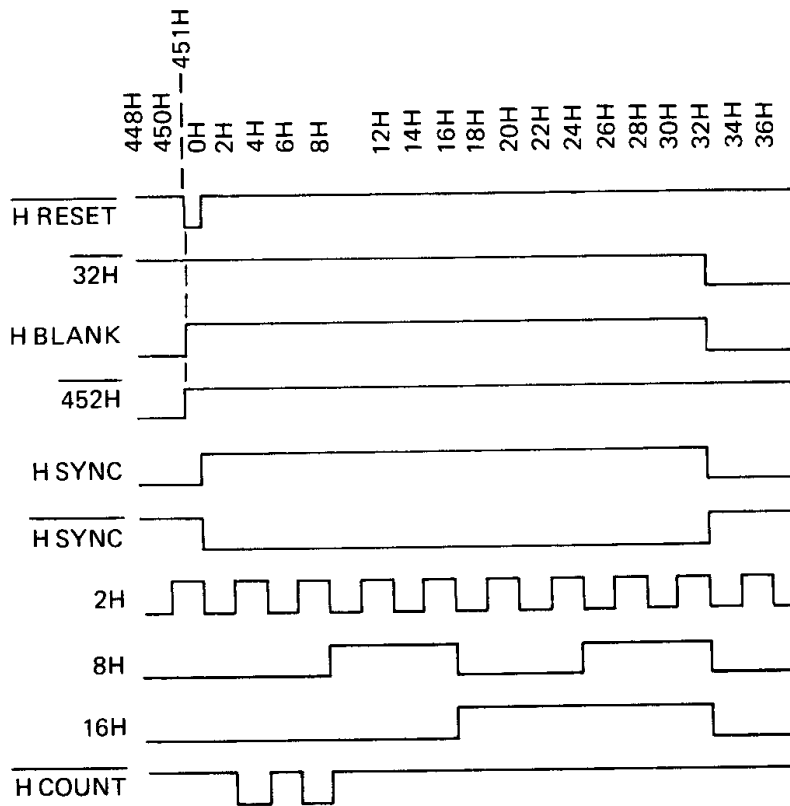
### 5.11 Vertical Main Timing

The vertical timing chain is virtually identical to the horizontal one except that it counts H BLANK pulses or half-lines rather than clock pulses. This means that — since each H BLANK pulse coincides with the termination of a half-line — the vertical timing chain counts lines per field and a vertical reset pulse coincides with the termination of each full field of half-lines (Figure 5.10-3).

This chain also contains 10 bits because it must count a total of 521 half-lines per field rather than the usual 262 full-lines found in most non-interlaced games. Also, since the vertical chain operates at a much slower speed than the horizontal, 7493 counters can be used. The two counters divide the frequency of H BLANK  $2^8$  times to produce the submultiples 1V through 128V. The ninth and tenth bits (256V and 512V) are supplied by two flip-flops and notice that both are reset by V RESET.

The figure above illustrates the vertical submultiple 64V and it appears much the same as the signal for a non-interlaced game.

To produce the  $\overline{VRESET}$  pulse at the required count of 521, 512V is gated with 8V to produce a LO-going pulse at 520 which is clocked through K12 by the next H RESET or on the 521st half-line. the reason for generating the vertical reset on an *odd number* is so that the two fields may be



**FIGURE 5.10-6 DEVELOPMENT OF H BLANK, H SYNC AND H COUNT** This diagram shows the relationships between most of the important horizontal timing signals.

interlaced. If an even number of half-lines were counted, the electron beam would always begin the new field in the upper left corner. By counting *one extra half-line* in between frames, the electron beam must begin the next field at the middle of the top edge of the CRT. Once it has finished this field, the next extra half-line forces the beam to begin again in the upper left corner.

On the 521st half-line, the vertical reset pulse occurs which resets the entire counter chain and the R-S flip-flop constructed from gates B11. This flip-flop is reset 8V later to develop a vertical sync pulse which extends from 521V (or 0V) to 8V.

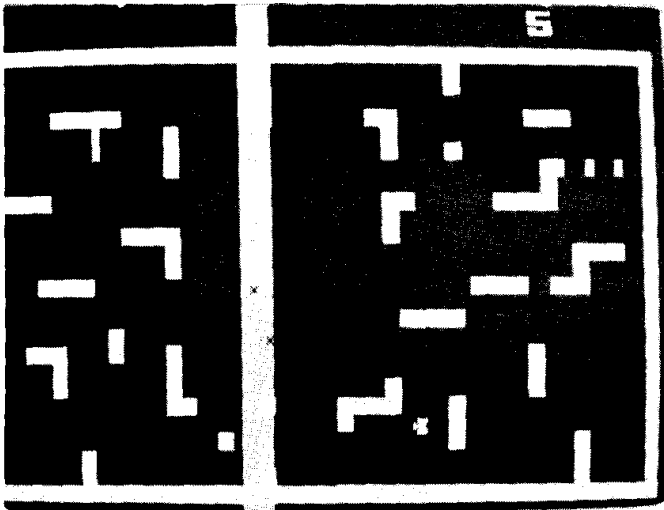
COMP SYNC is developed by gating the horizontal

and vertical sync pulses together at B11. Notice that a regular NOR gate is used to generate COMP SYNC rather than the usual exclusive OR.

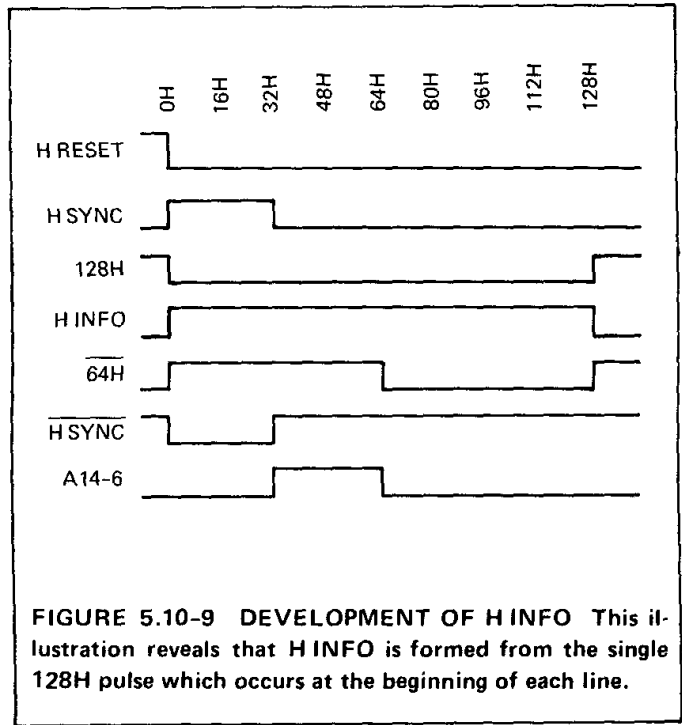
The photograph above graphically compares the COMP SYNC signals for the even and odd fields. You can see that the horizontal sync pulses contained within the vertical sync pulses are staggered.

## 5.12 V INFO

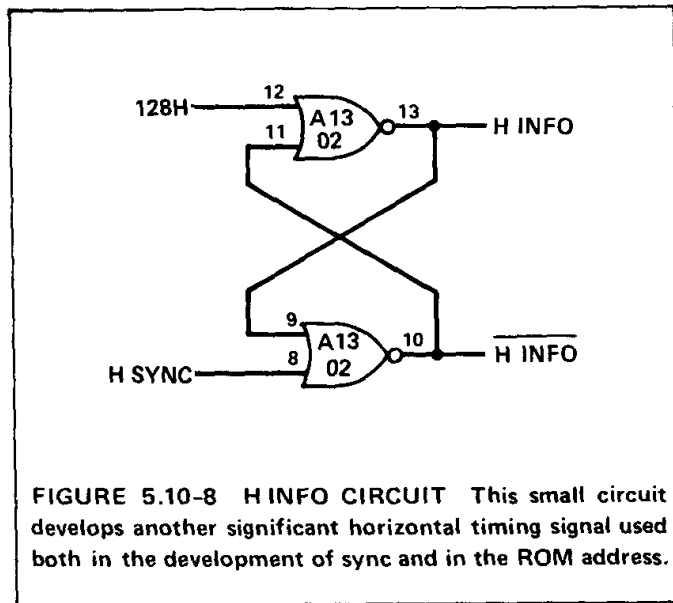
While the development of V INFO is not necessary to our discussion of interlaced sync, it will become important in a later chapter of this book for the section discussed in that chapter uses both H INFO and V INFO in the address of a read only memory.



**FIGURE 5.10-7 HBLANK** When viewed with a video probe, this signal is visible both in the center of the CRT and at the extreme right edge.

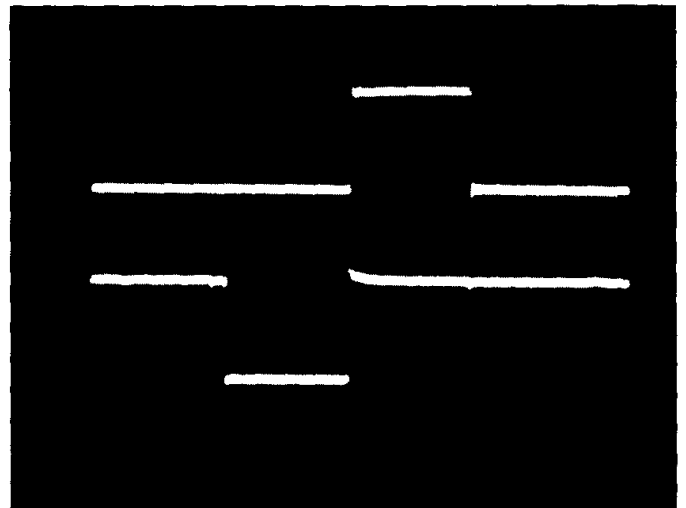


**FIGURE 5.10-9 DEVELOPMENT OF HINFO** This illustration reveals that HINFO is formed from the single 128H pulse which occurs at the beginning of each line.



**FIGURE 5.10-8 HINFO CIRCUIT** This small circuit develops another significant horizontal timing signal used both in the development of sync and in the ROM address.

The R-S flip-flop composed of gates J13 is set when 512V drops LO and is reset at 80V by the signal from J13-8 so that VINFO is HI between 512V (or 0V) and 80V.



**FIGURE 5.10-10 H SYNC** The actual pulse to which the monitor is synchronized (top) is compared with H SYNC (bottom).



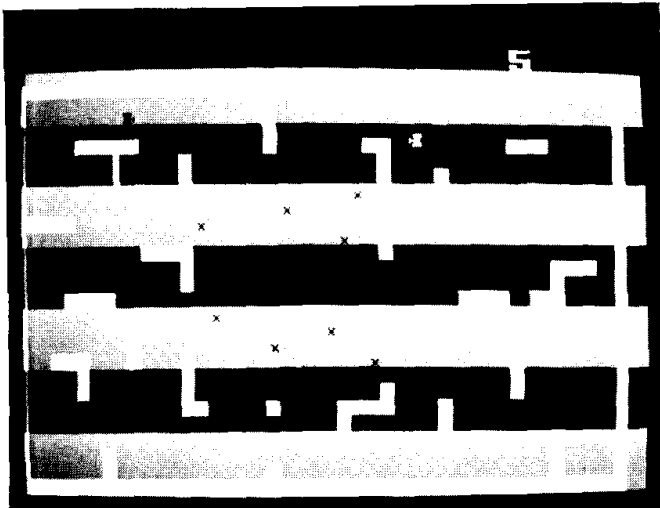


FIGURE 5.11-1 SIGNAL 64V This vertical submultiple appears very much as it would in a non-interlaced game.

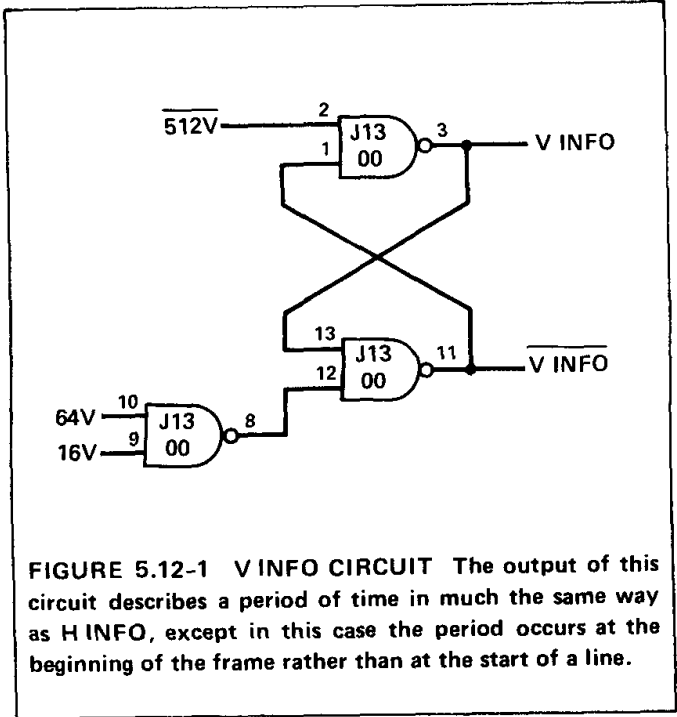


FIGURE 5.12-1 V INFO CIRCUIT The output of this circuit describes a period of time in much the same way as H INFO, except in this case the period occurs at the beginning of the frame rather than at the start of a line.

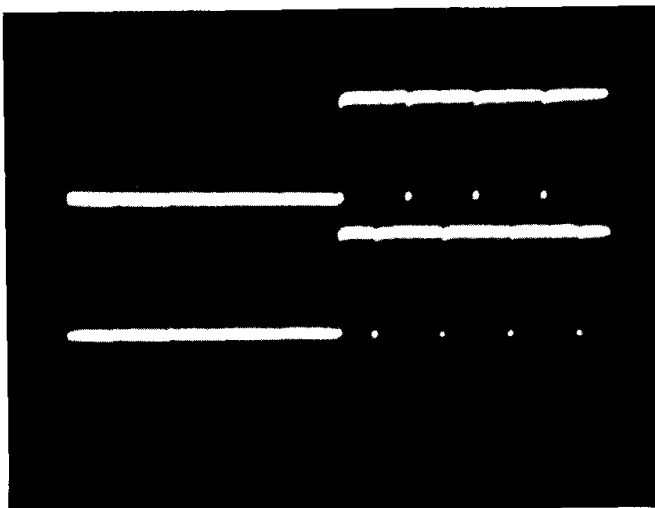


FIGURE 5.11-2 COMP SYNC The composite sync signal for the odd field (top) is compared with that of the even field (bottom).

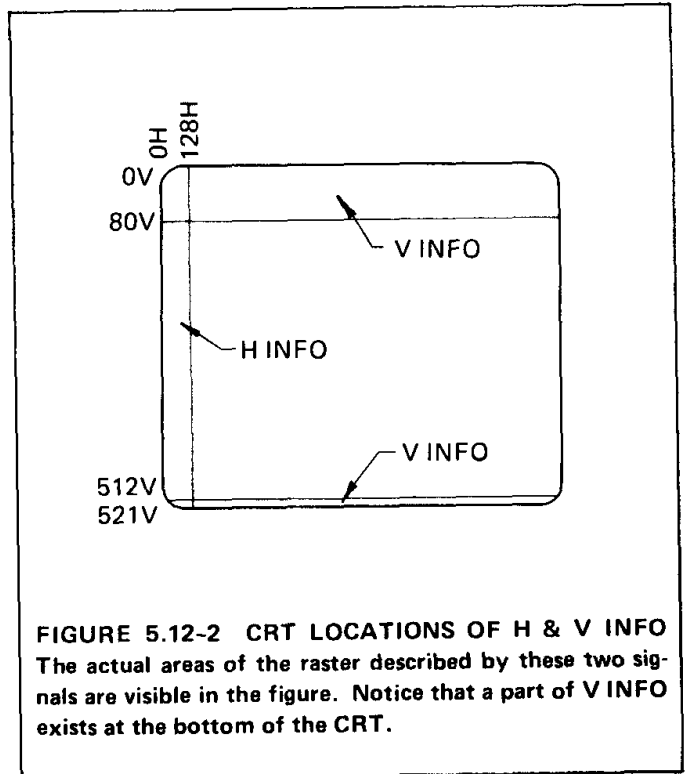


FIGURE 5.12-2 CRT LOCATIONS OF H & V INFO The actual areas of the raster described by these two signals are visible in the figure. Notice that a part of V INFO exists at the bottom of the CRT.



6

**motion**

## 6.1 Introduction

Players are enticed into putting their hard-earned quarters into a video game by the fact that they can exercise their own control over the images displayed on the CRT. Were these images static, players might watch the screen briefly, but they certainly would not be interested in paying to watch. However, when the player is allowed to actually direct the action on the CRT, a whole new area of interest is created.

Most of us were raised during the period when television gained tremendous popularity in this country and it is a fact that most homes currently have a TV set on for an average of six hours a day. This means that a lot of us have sat in front of the boob tube for a great many hours and are thoroughly addicted to that sinister, glowing device known as the cathode ray tube. But few of us have ever had the opportunity to exercise our own control over what happens on the screen; we sit passively and merely absorb what others control for us.

So it cannot be said that video games gained their popularity from the fact that they generate a display of some sort, because — if this were true — we could sit in front of regular TV and watch it for free. Then, what explains the fact that players line up to put their quarters into a video game for a mere two minutes of play?

Competition obviously has a lot to do with it because the most successful video games are also the most competitive. Other refinements such as complex controls, dynamic displays and the challenge of coordinating the proper eye-hand movements also have a great deal to do with maintaining player interest in a particular game, but these reasons do not sufficiently explain why the player was so eager to put his quarter in the machine in the first place.

The explanation of this remarkable phenomenon is simple: we just like to move images around on the tube. Like all other human endeavors, it allows

us a measure of control with which we can express our individuality and modify the environment around us. And, it is particularly pleasing for most people to be able to dominate a TV, a mysterious electronic device with which our lives are inevitably intertwined, yet a device which we have never been able to directly control before.

This being true, the design of video game motion circuits takes on a much greater significance even though the circuit itself may bear no greater degree of design difficulty than any other circuit because — without this circuit — the video game *per se* cannot exist.

The concept of using still pictures to generate a moving image has been around for a long time. Even before the moving picture was invented, the same effect was exploited by school boys who doodled pictures in the corners of their textbooks. As the pages were flipped, the image appeared to move.

Although CRT motion is produced using the same basic concept, several techniques have been employed to effect it. Regular TV receivers produce realistic motion by displaying successive frames wherein the image is slightly shifted each frame. Radar tracking systems are another example of motion achieved on a CRT, albeit slowly. In this case, a “blip” on the vector-scan monitor or CRT is repositioned each time the radar antenna makes a full sweep and it can be seen to move if only in a very slow and jerky manner. Obviously, the faster the radar sweeps, the more times the blip will be repositioned within a given time period and the smoother the motion will appear to be.

Given the raster-scan monitor, there is only one way motion can be created, but there are several ways it can be controlled. Motion can only be achieved by displaying successive frames where the position of the image is slightly changed each frame, but the movement of the image itself can be controlled by a multitude of techniques which include both analog and digital circuits. In terms

of analog techniques, the most widely known is one developed by Magnavox, Inc. which uses a technique conceptually similar to that of video game paddle movement. We have also seen a myriad of other designs as well and it would seem that the possible number of analog motion designs is almost limitless.

While analog motion designs can be successfully implemented, their use in a digitally-oriented video game computers is limited due to the inherent difficulty of interfacing the two types of circuits. However, even though digital techniques are much more commonly used, only two distinctively different types have been developed. The oldest of these two uses a full-blown computer programmed with a logical sequence of steps to take information from memory and display it at the proper time on the CRT. Prior to the advent of microcomputers, this technique was possible only with expensive minicomputers or even larger systems. While the expense of this technique could be justified for uses such as radar tracking displays and graphics terminals, it was far too high for production video games where the existing coin game standard of no more than 25¢ per play would not be capable of generating the income required to install a mini. Clearly another approach needed to be devised for video games whereby motion could be digitally generated and controlled by a simple and inexpensive circuit. Once the need for such a circuit was encountered, the solution became immediately apparent.

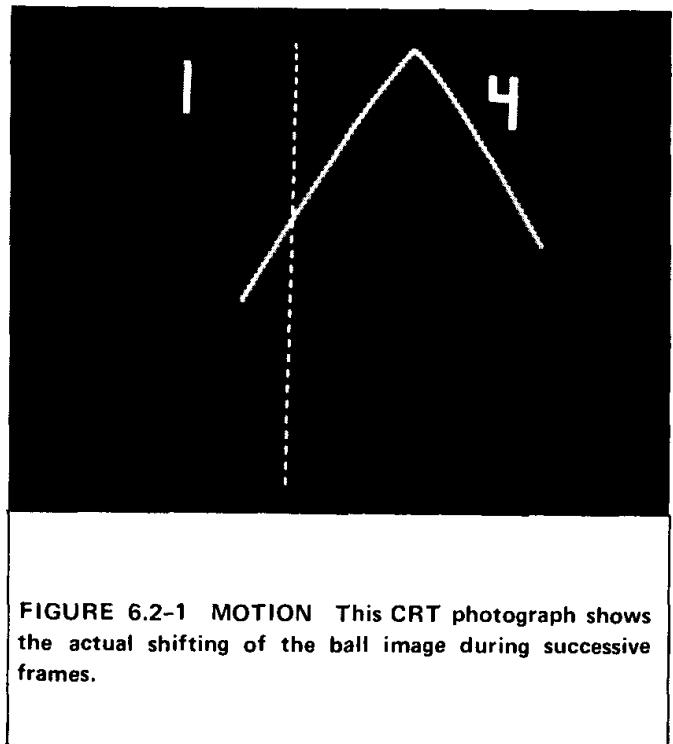
This technique involves building a dedicated, hard-wired circuit in a configuration which produces a dependable *algorithm* which causes the image to be shifted to a slightly different place each frame. Essentially, this is accomplished by running two sets of counter chains in parallel, a concept which has been around for quite some time but never applied before in this way.

## 6.2 The Illusion of Motion

The term "motion" when applied to CRT phenomena is actually a misnomer in that the image

never does move; it only *appears* to move. You may have never considered this while sitting in a movie theater, but the images do not actually move on the movie screen either. In fact, were it not for an interesting quirk of the human eye and the eagerness of the brain to misinterpret what the eye sees, even the movies would be perceived to be no more than a succession of still pictures rapidly flashed on and off.

As successive frames of the movie are projected on the screen, the eye perceives the positional shift as smooth motion because of a phenomenon known as *retinal after-image*. In other words, the last seen image is "remembered" for a short period of time and, as the position of the image is rapidly shifted, the images overlap and the brain is tricked into interpreting the result as continuous motion. When displayed on a TV screen, this effect is enhanced by the *persistence* of the phosphor coating on the inside of the CRT, which causes the image to linger even after the electron beam has moved on.

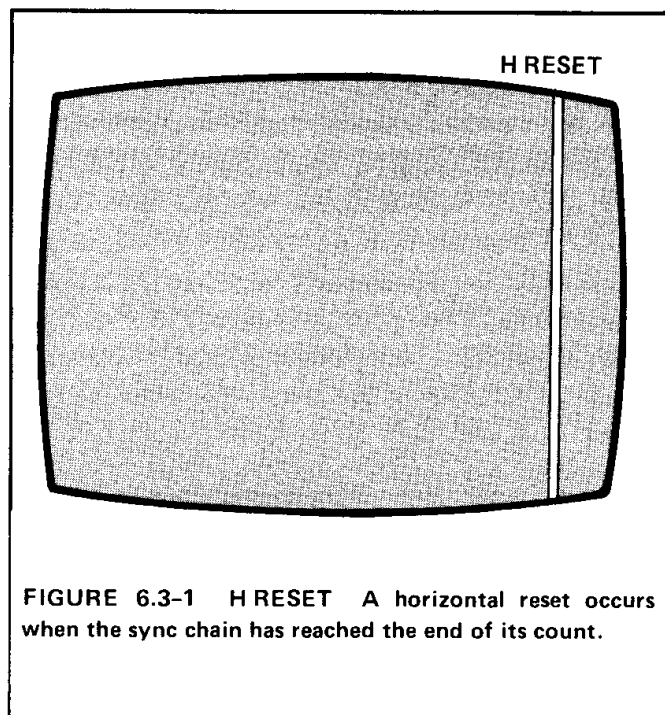


## 6.3 Vectored Motion

Although the motion concept itself is familiar

and relatively easy to comprehend, the circuitry required to control and display motion is a bit more complex since the operation of several circuits must be visualized simultaneously. Essentially, these circuits are constructed and operated just like sync circuits and are no more complex. The only difficult idea to grasp is the *interdependence* between the sync and motion circuits.

If we examine a horizontal sync chain output with a video probe, we will see some sort of display which contains a pattern of vertical lines or columns. This pattern cannot move because the points of light are displayed in exactly the same place each frame due to the precise and repetitive nature of the sync chain. Normally, a horizontal sync chain uses 7493 asynchronous counters which output an HRESET pulse every 454 (sometimes 452) clock pulses (see figure below).



**FIGURE 6.3-1 HRESET** A horizontal reset occurs when the sync chain has reached the end of its count.

Now let's say we construct a parallel horizontal counter chain which counts clock pulses just like the sync chain and develops a "reset" pulse at 454, only this time let's use 9316 synchronously presettable counters instead of the 7493s. And, instead of resetting these counters with the "reset" pulse, let's just display it on the CRT and only

reset the counters when they reach their terminal count.

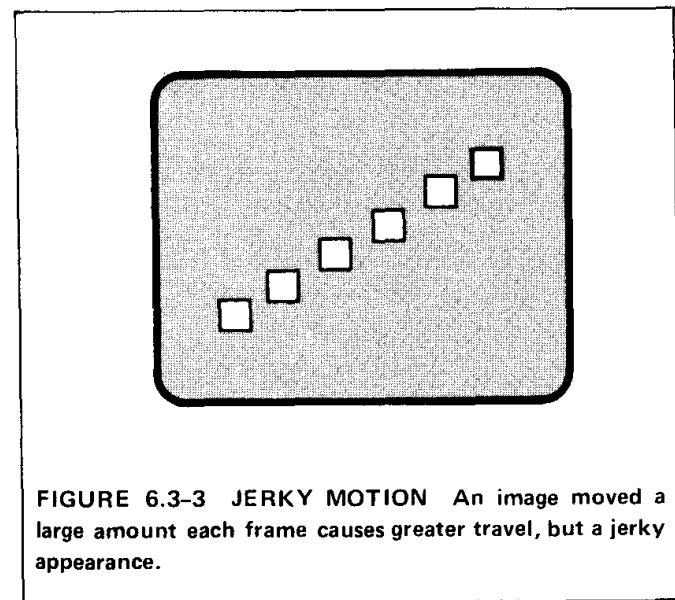
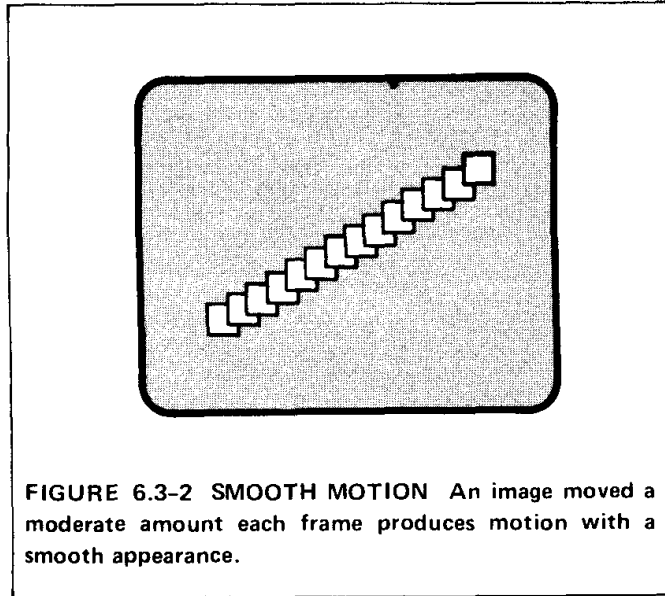
By inputting the right combination of 1s and 0s into the parallel inputs of the counters, we can force the chain to begin its count from any desired number within its range, however terminal count will *always* occur on the 454th clock pulse or count.

Now, if we begin this chain the same time we start the sync chain, both will reach their terminal counts simultaneously and if we examine TC from the presettable motion chain, it will appear in the same place as HRESET. However, if we enter a binary number into the parallel inputs of the motion chain in such a way that it delays the TC of this chain by one clock pulse with respect to the reset of the horizontal sync chain, TC of the motion chain will be displayed one clock pulse to the right of HRESET each frame and will therefore appear to move to the right.

The difference in time between when the motion and sync counters reach the end of their counts can be called the *sync/motion differential*. If this differential remains at zero, the object image will be created in the same place each frame and will appear motionless. However, as this differential is varied *up* or *down*, the point at which the motion chain reaches its terminal count will be *advanced* or *retarded* with respect to sync and the position of the image controlled will be shifted to the *left* or *right*. Because the TC of this type of motion circuit can be varied with respect to the sync outputs, this circuit is often called a *slipping counter chain*.

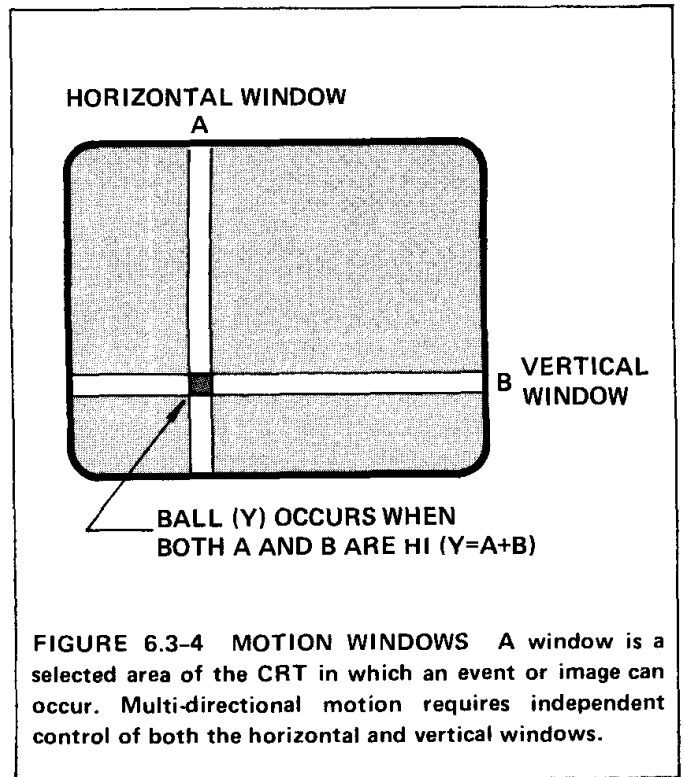
The speed or *velocity* of the moving image is controlled by the *shift rate/frame rate ratio*. In other words, if the position of the image is shifted one clock pulse each frame, it will appear to move much more quickly than if it were shifted only once every two or three frames. The same effect can be achieved by holding this ratio constant and varying the *amount* the image is shifted each frame. Obviously, a ball image that is shifted three

clock pulses each frame is going to appear to move much faster than if the same image were shifted only one clock pulse each frame. The total amount of shift, however, can only be carried to a certain point or the shifting process will be revealed as a series of jerky repositionings.



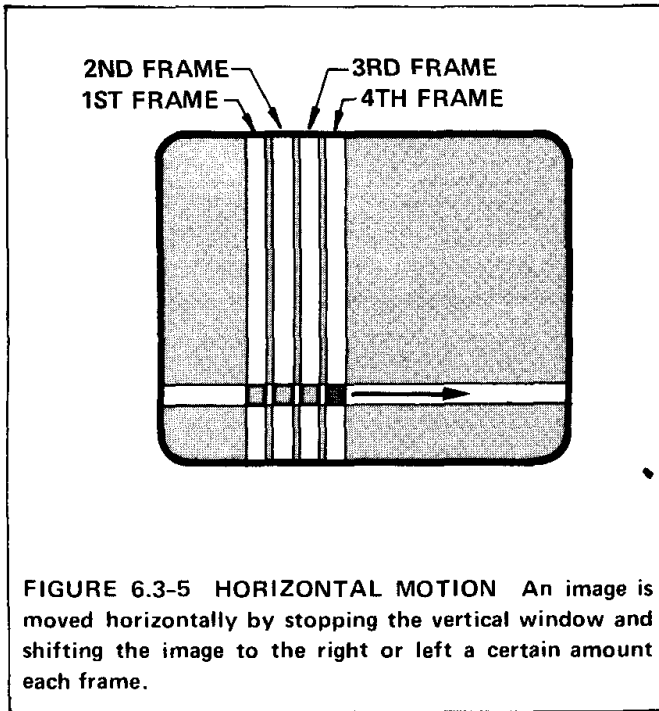
Although we will be delving into the subject of *windows* much deeper in the next chapter, let it suffice for now to say that a *window* is a selected area of the CRT in which an event or image can occur. A vertical window appears as a horizontal bar and is limited at the top and bottom; a hori-

zontal window looks like a vertical column and is bounded on the left and right sides. The ball image in Figure 6.3-4 occurs at the intersection of the two windows and it is bounded on all four sides.



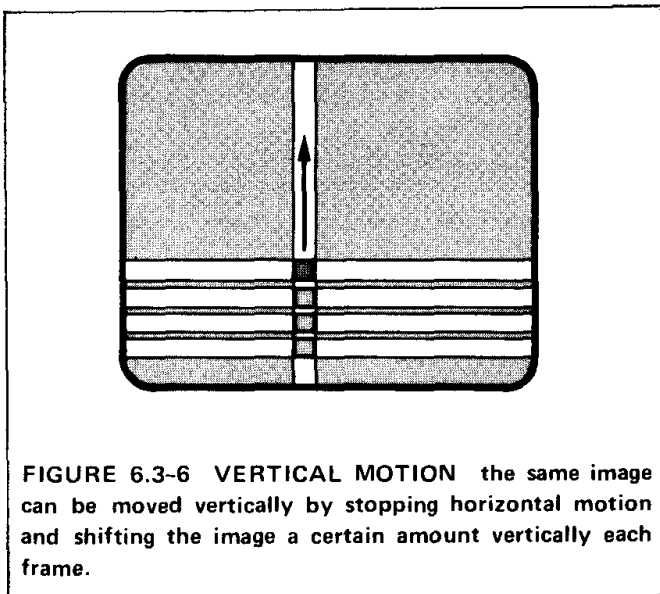
Moving displays are created in the same way as static displays, except their image windows are *moved*. The image contained within the window may be as simple as a square ball shape as in the figure above, or the square area determined by the intersection may be used to hold a more elaborate image such as a race car produced by another circuit. The *direction* of the moving image is controlled by shifting the horizontal and vertical windows *with respect to each other*.

Figure 6.3-5 shows how shifting the horizontal window affects the image created at the intersection. In this case, we have “frozen” the vertical window in one place so the effect of horizontal shifting can be more readily appreciated. You can see that if the horizontal window is retarded by one clock pulse per frame, the image will appear to move to the right.

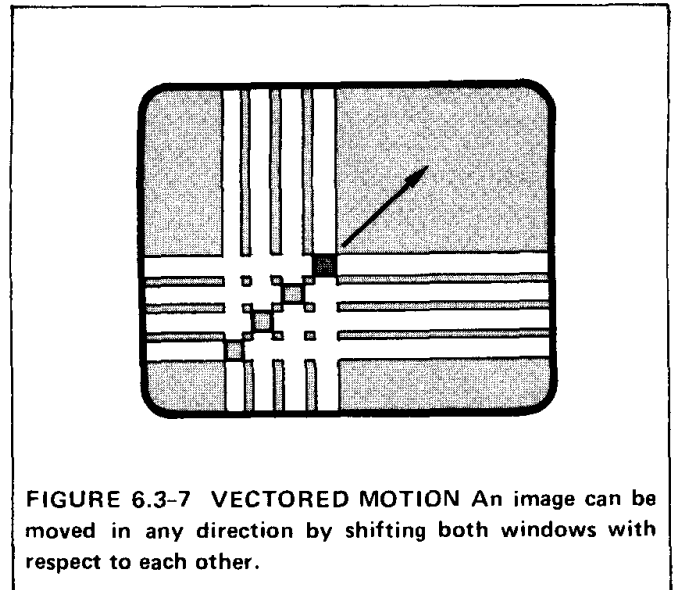


The same process can be used to move the image vertically. In Figure 6.3-6, we have stopped the horizontal window, however the vertical window is being advanced each frame so the image travels upward.

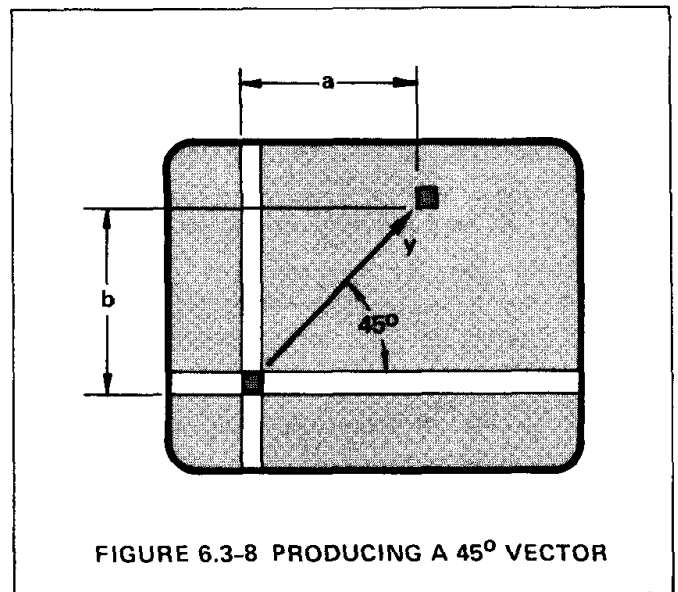
So far, we have generated motion in two directions only: up/down and left/right, but this is obviously not going to make a very exciting video game. To produce an image capable of being moved in *any*

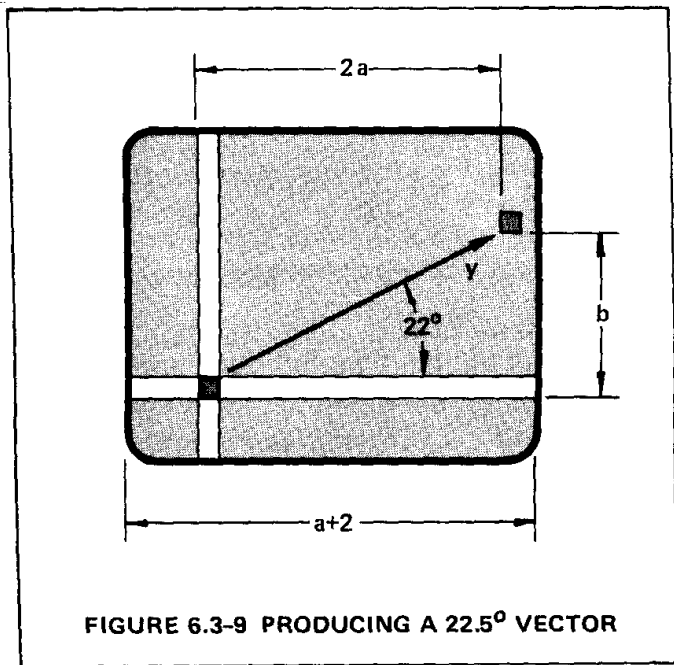


direction, all we need to do is vary *both* windows *simultaneously*. In Figure 6.3-7, you can see that if we move the image to the right *and* up one clock pulse per frame, it will appear to move along a  $45^\circ$  vector.

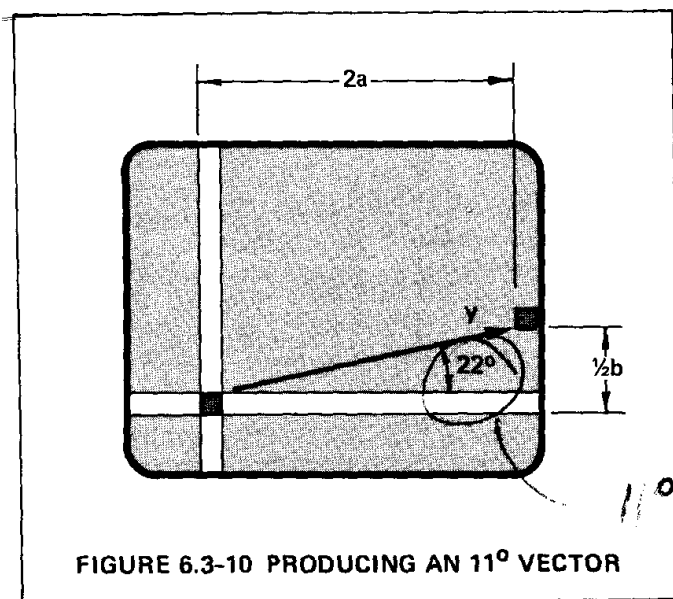


But what if we want the image to move at  $22.5^\circ$  instead of  $45^\circ$ ? Since equal vertical and horizontal shifting (Figure 6.3-8) causes a  $45^\circ$  vector, a  $22.5^\circ$  can be just as easily produced by shifting the image horizontally by *twice* the amount of the vertical shift (Figure 6.3-9).



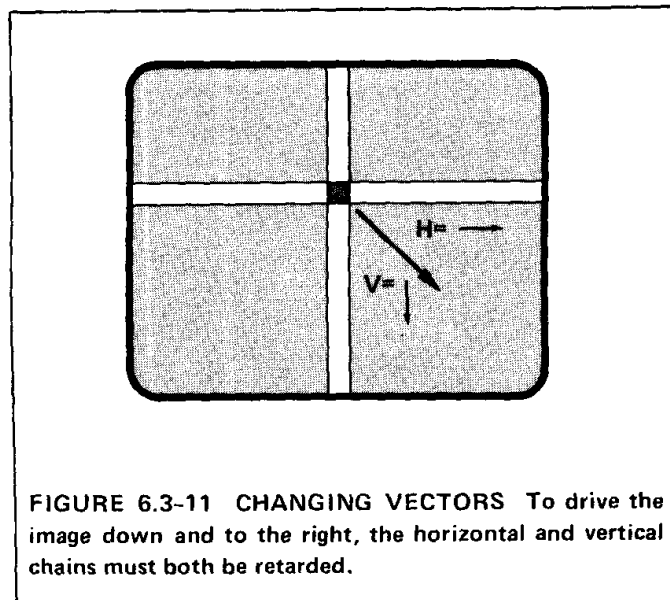


To carry this process even further, all we need to do is shift the image to the right *four* clock pulses horizontally to every *single* clock pulse vertically and the image will be forced to move along the vector as illustrated in the figure below.

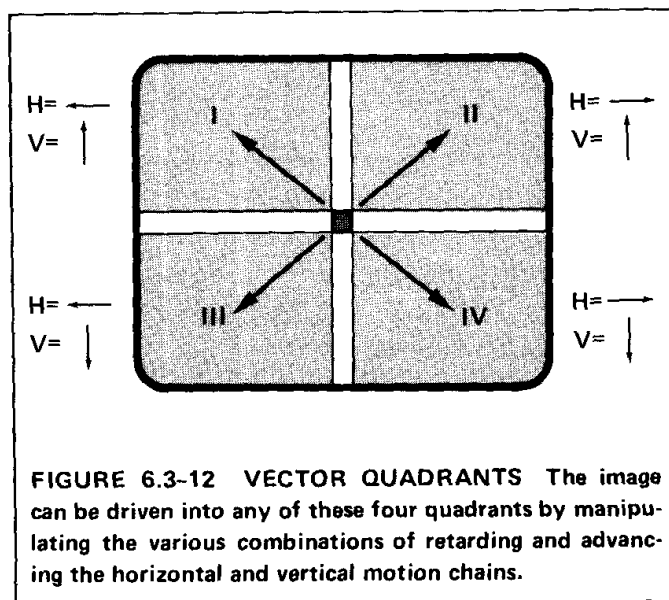


Using this same process, we can drive an image anywhere around on the CRT. For example, let's say we want our image to be moved *down* and to the right at an angle of 45°. In this case, we must shift the horizontal and vertical windows equal amounts to create a 45° vector, but to cause

*downward* motion, we must retard *both* the vertical and horizontal windows each frame as in Figure 6.3-11.



Likewise, we can drive our image into any one of the four quadrants illustrated in Figure 6.3-12 by varying when the horizontal and vertical windows are advanced or retarded. Quadrants I and II necessitate advancing the image vertically, but to make the image down into III and IV, the vertical motion chain must be retarded. Similarly, the left quadrants I and IV require that the horizontal window be advanced with respect to sync and the





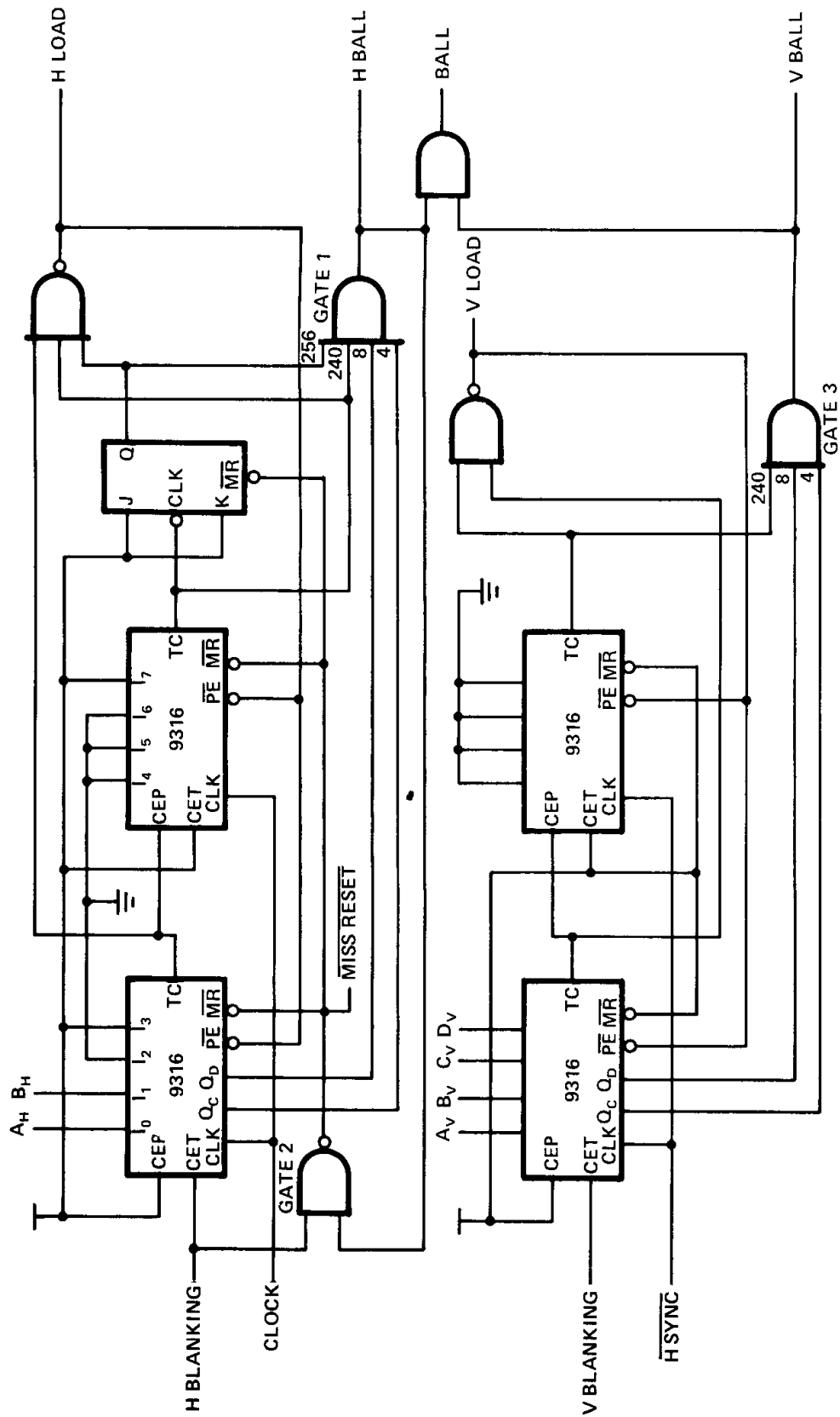


FIGURE 6.4-1 SLIPPING COUNTER MOTION Motion is achieved by varying the point at which these counter chains reach their terminal counts with respect to their sync chain counterparts. Notice that these chains can be loaded with a number at the LSBs. The different combinations of these binary numbers are known as motion codes.

right quadrants necessitate delaying the output of the horizontal motion chain with respect to the horizontal sync chain.

#### 6.4 A Typical Motion Circuit

Now that we have discussed the theory behind moving images, we can dive into a real motion circuit. As long as you remember that a motion chain is no more than a glorified presetable sync chain, the following information should be relatively easy to assimilate.

The implementation of motion actually involves two types of circuits. First, two *slipping counter motion chains* are needed to display both horizontal and vertical motion. Second, two more sub-circuits are required to generate the inputs to the slipping counters and these are usually known as the *motion control* circuits.

The horizontal and vertical chains are illustrated in Figure 6.4-1 and the striking similarities between these two sub-circuits and their sync counterparts in Figures 5.3-1 and 5.6-1 should be immediately obvious. The major difference is that there are *variable inputs* to the least significant bits of both motion chains.

The 9316 counters in the motion circuit above can be started from any desired number by entering the binary equivalent of that number at the parallel inputs while the parallel entry terminal (PE) is held LO. When  $\overline{PE}$ , CET and CEP are all HI, the counter will count clock pulses starting from the last number loaded. The counter outputs (pins 11, 12, 13 and 14) increment until the 15th count, at which time TC rises HI and clocks the next bit in the chain.

The various codes which affect the motion/sync differential are entered at the  $A_H$  and  $B_H$  inputs to the horizontal motion chain to change the motion window in the horizontal plane. The signals at vertical inputs  $A_V$ - $B_V$  control the positioning of the vertical window. Each chain has several types of codes which can be entered and, for hor-

izontal motion, these include (1) *left codes* which advance the chain, (2) *right codes* which retard the chain and (3) the *horizontal stop code* which time this chain with the horizontal sync chain and causes a complete cessation of motion. The codes for vertical motion are similar except, in this case, they are (1) *up*, (2) *down* and (3) *vertical stop*. All these codes are generated by the motion control circuits which change codes according to playfield events to affect both the speed and the direction of the image.

The example circuit in Figure 6.4-1 has been taken from the original paddle game, Pong, and — as you are probably already aware — the ball in this game changes *direction* after each hit by a paddle and after it encounters either the upper or lower playfield boundary (vertical blanking). The *speed* of the ball is increased after the twelfth volley to increase the difficulty of play to make the game more interesting for better players. The horizontal sync chain of this game is reset on the 455th clock pulse which means that if we want to stop motion, we must enter a number into the  $A_H$  and  $B_H$  inputs such that the motion chain also reaches TC at 455.

When the entire motion chain has reached TC, HLOAD drops LO causing the new motion code entered at the parallel inputs to be latched into the counter and this number immediately appears at the parallel outputs so that the chain begins its count from this point.  $A_H$  and  $B_H$  control the two least significant bits and these are the only two parallel inputs which can be changed since the rest of the inputs are all tied either HI or LO. If we substitute "X"s for the  $A_H$  and  $B_H$  inputs, the permanent or *base* code always present at the chain would appear as: X X 0 1 0 0 0 1. If  $A_H$  and  $B_H$  are both 0s, the actual code entered is 0 0 0 1 0 0 0 1 or decimal 136. However, since both  $A_H$  and  $B_H$  can be changed, there are four possible combinations permitting a preset number ranging from 136 to 139 (Figure 6.4-2).

Like horizontal sync, the motion chain also con-

CODE	A <sub>H</sub> B <sub>H</sub>		I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>
	I <sub>0</sub>	I <sub>1</sub>						
136	0	0	0	1	0	0	0	1
137	1	0	0	1	0	0	0	1
138	0	1	0	1	0	0	0	1
139	1	1	0	1	0	0	0	1

**FIGURE 6.4-2 HORIZONTAL MOTION CODES** By varying the two LSBs of the horizontal motion chain, a total of four different codes can be entered. The variable bits are indicated in grey.

tains nine bits permitting a full count of 512 as long as the ball is not missed. But if a player does not return the ball by hitting it with his paddle, the ball will pass through horizontal blanking and the LO from Gate 2 will reset the entire chain.

With this in mind, let's drop code 137 into the chain and see what happens. Once the number 1 0 0 1 0 0 0 1 is latched into the chain and the chain enabled to begin counting, it will begin from 137 and count up until it reaches terminal count 375 clock pulses later (512 minus 137 equals 375). Since horizontal reset occurs on the 455th count, a sync/motion differential of 375 counts would produce drastic shifting of the image and very unrealistic motion were it not for the fact that the counters are inhibited for an additional 80 clock pulses, by the presence of HBLANKING at the CEP pin of the first counter. This being true, the addition of 80 more counts to our previous figure of 375 equals 455, which synchronizes this chain with the horizontal reset of the sync chain and stops horizontal motion. Therefore, in this game, the code 137 is called the horizontal stop code.

But we are obviously not always going to drop the stop code into the chain or motion would not be possible. So, if we enter the binary equivalent of the decimal code 138 into the chain, the motion

chain will reach TC one clock pulse before the sync chain and the ball image will be shifted to the left by one clock pulse per frame. Similarly, the code 136 causes shifting to the right because the motion chain will advance once per frame, causing the electron beam to deposit the ball image one clock pulse later.

The addition of counts 256, 240, 8 and 4 at Gate 1 equals 508. Since 4 is the least submultiple here, the output of Gate 1 will rise HI at 508 and return LO 4 clock pulses later, or at 512. Since this process occurs once during each line of the raster, a 4H wide vertically-oriented horizontal window is generated which limits the width of the ball image to a thickness of 4H.

The vertical chain operates in a similar manner, however there are a few significant differences. First of all, this chain counts H SYNC pulses rather than points on the line. Also, only eight bits are required in the chain, since a trick is employed such that the largest number needed is only 256. Furthermore, there are four variable inputs to this circuit so a greater number of codes can be generated.

The signal which enters the new code (V LOAD) is derived by the addition of the TCs of both counters. The TC of the first counter rises HI between the 15th and the 0 counts and the second rises HI only between the 240th and 256th counts. This means VLOAD will drop LO only between the 255th and 256th counts. Since vertical reset occurs on the count of 262, a count of 256 would normally be insufficient to operate the circuit. Therefore, VBLANKING is used to disable the first counter for 16 clock pulses so the result of adding 16 to 256 equals 272, which is 10 more clock pulses than is required to stop motion vertically. So, we need a way to cause this point to occur at 262 and this is easily done by dropping the stop code into the chain. If we enter the code 10 into the chain, it will begin at 10 and reach TC 262 counts later which times this chain to vertical sync and stops motion vertically.

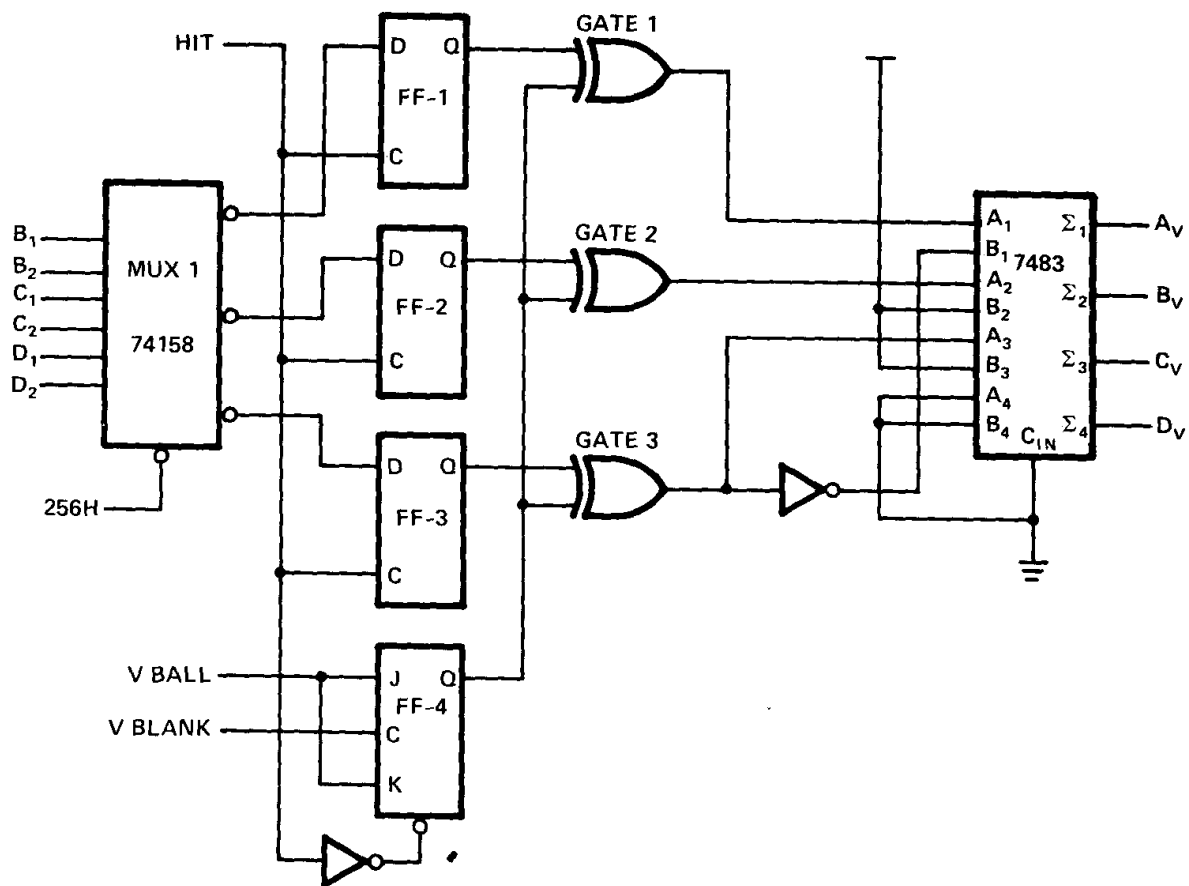
However, we could also enter codes 11, 12 or 13 which produce upward motion by advancing the count or we could drop in 7, 8 or 9 which will retard the chain and produce downward movement of the ball.

### 6.5 Vertical Motion Control

In reality, two separate circuits are needed to control both horizontal and vertical motion, but the vertical version is the only one analyzed here since it is quite a bit more interesting. This circuit de-

fects either a paddle hit or a playfield wall encounter and changes the motion code to produce a realistic bounce.

The circuit in Figure 6.5-1 has several interesting features, including a "selective inverter" constructed from the three exclusive ORs which produce inverted motion codes to give the complementary angle after a playfield wall encounter. The direction after a paddle hit is determined by the paddle segment inputs to MUX 1 and this information is temporarily stored in the first three



**FIGURE 6.5-1 VERTICAL VELOCITY** This circuit detects which part of the paddle was contacted by the ball and changes the motion code an appropriate amount. The circuit also detects an encounter between the ball and the playfield boundaries and produces the inverse of the motion code so the ball is bounced at a complementary angle.

flip-flops. The new motion code results when this information is clocked through the flip-flops by HIT and appears at the inputs to the exclusive ORs.

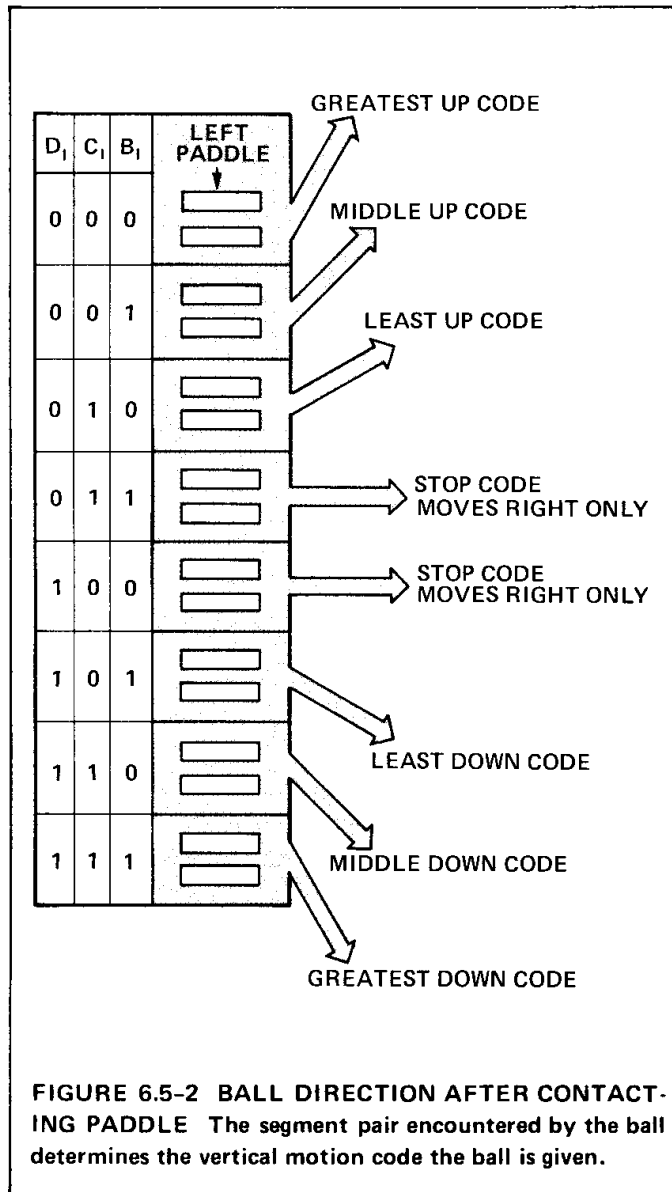
The paddle circuit in Chapter 9 generates a 16-segment paddle image where each pair of segments is associated with a binary number output by the paddle counter (Figure 6.5-2). Since there are two players, each with his own separate paddle count, both player's signals must first be multiplexed together so the circuit looks at one player's inputs and then the other's. Since 256H is LO during the left half of the CRT and HI in the right, the first player's paddle count ( $B_1$ ,  $C_1$  and  $D_1$ ) appears at the multiplexer outputs as the electron beam is

scanning the left half of the CRT. When the beam scans the right half of a line, 256H is HI and the second player's signals are enabled through.

If the ball strikes one of the three upper segments in Figure 6.5-2, this circuit computes a new motion code which produces upward appearing motion when interpreted by the vertical motion circuit. The higher up on the paddle the ball strikes, the greater the resulting code and the position of the ball is shifted a larger amount on the CRT to increase its angle. If the ball strikes any of the three lower segments, the appropriate down code appears at the multiplexer outputs to generate downward appearing motion. But if the ball strikes either of the two middle segments, the vertical stop code is output by this circuit so the ball can only travel across the screen horizontally. Since two pairs of segments are used for the stop code, the probability of a hit producing the stop code is twice that of any other single code. This is the reason straight horizontal volleys are so common when playing paddle games.

Let's say the ball has just bounced off the top playfield wall and the first (left) player moves into position to hit it. His paddle count always increments through a regular binary count which is always present at the multiplexer outputs whenever the electron beam is scanning the left half of the CRT. This count is inverted by internally-inverted multiplexer outputs so a hit on the lowest paddle segment (1 1 1) actually appears as 0 0 0 at the outputs. At any rate, when a hit is detected (by another circuit incidentally), a HI pulse on the HIT line clocks the existing paddle segment count through Flip-flops 1, 2 and 3.

The function of these flip-flops is to store this count until the next hit occurs so the ball is given a constant code between collisions and travels in a straight line. So, the information at the Q outputs remains stable and appears in the same form at the inputs to the adder as long as an encounter between the ball and the top or bottom playfield walls does not occur.



The adder is a device used to take a binary paddle count number and convert it to another binary number more usable by the motion circuit. All it does is take the number present at the A inputs, add it to the number at the B inputs and drop the result out the  $\Sigma$  outputs. Since the  $B_2$  and  $B_3$  inputs are tied HI and the  $B_4$  and carry-input ( $C_{IN}$ ) are held LO, a base number of 6 (0 1 1 0) is always added to the number at the A inputs if  $B_1$  is LO. But if the  $B_1$  input rises HI, the base number 7 is added with the A inputs. At any rate, when the A and B numbers are summed, the result is output codes which range from 7 to 13: three speeds down, three speeds up and one stop code.

TYPE OF CODE	VELOCITY CODE DECIMAL EQUIVALENT	PADDLE SEGMENT PAIRS	INPUTS TO ADDER							
			A <sub>4</sub> (TIED LO)	OUTPUTS FROM GATES			B <sub>4</sub> (TIED LO) B <sub>3</sub> (TIED HI) B <sub>2</sub> (TIED HI) B <sub>1</sub> (INVERSE OF A <sub>3</sub> )			
				A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>				
UP	13		0	1	1	1	0	1	1	0
	12		0	1	1	0	0	1	1	0
	11		0	1	0	1	0	1	1	0
STOP	10		0	1	0	0	0	1	1	0
	10		0	0	1	1	0	1	1	1
DOWN	9		0	0	1	0	0	1	1	1
	8		0	0	0	1	0	1	1	1
	7		0	0	0	0	0	1	1	1

FIGURE 6.5-3 MOTION CODE TRUTH TABLE This table compares the type of motion code and its decimal equivalent with the inputs to the adder (the base number and the paddle segment pair code).

Figure 6.5-3 illustrates the operation of the adder in this circuit. On the left, you can see the decimal motion code equivalents compared with the direc-

tion of the code. A set of paddle segment pairs is symbolically illustrated in the center and the paddle segment count is incorporated in the A input column just to the right. This column contains the paddle count with an extra 0 tacked on the end and inverted by the outputs of the multiplexer so that it goes from 0 1 1 1 down to 0 0 0 0.

You may be wondering by now how the complementary motion codes are produced when the ball collides with the top or bottom walls. This process is illustrated in Figure 6.5-4 which shows successive frames of the ball striking the upper playfield wall.

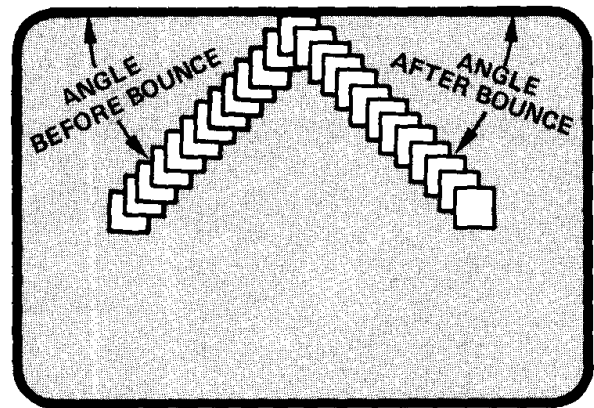


FIGURE 6.5-4 BOUNCE ANGLE The ball is given a complementary angle after contacting either the upper or lower playfield boundaries which are defined by V BLANKING.

Remember that the last angle is always stored by Flip-flops 1, 2 and 3 so this information is present at the inputs to the exclusive ORs. Normally, the information is simply passed through these gates since the Q output of Flip-flop 4 is normally LO. But when the vertical ball window runs into the upper playfield wall (V BLANKING), the HI at V BALL is clocked through the flip-flop and inverts the count from the exclusive ORs. HIT is inverted and clears the flip-flop so the Q output is LO and the true codes can result after a hit from a paddle occurs.



7

**creating images**

## 7.1 Introduction

The prime factor influencing the success of video games is the fact that players are fascinated by watching and being able to move glowing images around the cathode ray tube of the TV monitor. This being true, techniques for generating images are one of the most important single design considerations for successful games, so a great deal of research has been done to discover new and more efficient ways to generate and store image information.

The importance of display techniques has led to a number of interesting innovations and clever circuits, however most of the basic techniques have not been developed specifically for video games. Rather, these methods are based in other fields of digital electronics such as semiconductor memories and computer terminals where the display of characters is of the utmost importance and a great deal of research has led to many of the techniques we see today in video games. Although the basic concepts as applied to video games were not originated by game designers, their application in games has resulted in some highly original and interesting circuit designs.

Early games were, of course, the simplest and the techniques used were the most basic. In fact, the earliest games depended very little on sophisticated display techniques; they were successful merely because of the newness of the craze. However, even these games needed some display and engineers came up with a number of simple, but interesting techniques for generating basic images such as paddles and score numerals. But as players became a little jaded, designers began to search for ways to generate more complex images and it quickly became obvious that the old methods would be too unwieldy for the greater number of more detailed shapes required.

## 7.2 Generated Vs. Stored Displays

The techniques pioneered by the first games are still in use even in state-of-the-art games, however

their importance has drastically decreased. These techniques all centered around the *generated display* concept where a circuit using basic components such as gates and flip-flops produced an *algorithm* which resulted in a particular shape on the CRT. To create images using this technique, sync submultiples are combined in various ways to produce geometrically oriented shapes. While this technique was eminently successful, its limitations quickly became apparent for not only does this technique require many separate chips and a considerable amount of PCB real estate, but it is also quite limited in the amount of detail which can be conveniently produced and the flexibility with which a particular shape can be effected.

So designers began to explore ways with which they could produce specific and intricate shapes they had in mind while keeping circuits simple and minimizing the total number of ICs required for a particular game theme. The only other digital technique which appeared applicable was that of *stored displays* which depend on some sort of memory to retain the information needed to generate the images. The essential difference between this technique and the generated display is that stored information is read out only when needed — it is not constantly being produced.

The first games using stored displays depended on the *diode matrix* to hold the information. Although the diode matrix cannot conveniently hold a great deal of data, it does allow the designer a good measure of flexibility insofar as he can easily plot out and execute the shape he has in mind. But more important is the fact that the diode matrix can be built with ordinary parts lying around the lab and does not require the hassle and expense of programming a real memory. In the days of the early video games, only one type of read-only memory was available and this mask-programmed ROM required a considerable investment of money to pay for the initial mask charge to enter the specific data. This was pretty scary to designers, so they avoided ROMs like the plague.



But today, there are many new varieties of ROMs available which a designer can program in his own lab. This allows him to check for mistakes and have the results he needs inexpensively and quickly. When a final shape has been decided upon, he can then produce a limited run of PROMs so prototype games can be manufactured and tested to see if the design will be successful. If the prototype results are encouraging, the manufacturer will feel justified in having a mask cut so that a large number of relatively inexpensive memories can be programmed with his data.

The proliferation of integrated semiconductor memories has led to an entirely new video game architecture. Many, if not most, of today's games are virtually built around the ROMs they contain and it is not unusual to find that the ROM in a game contains all the object shapes, score numerals, motion codes and other instructions and data the computer might need.

### 7.3 Building A Generated Display

Sync submultiples can be gated together in an endless variety of ways to generate many types of images ranging from the very simple to more sophisticated shapes which appear quite complex. If the display requirements are relatively basic, SSI components are normally used but often more complex images can be derived using MSI devices such as counters and magnitude comparators without increasing (or maybe even decreasing) the total number of ICs required for the particular image. All generated display techniques depend heavily on the use of *windows* to specify certain areas of the CRT.

### 7.4 The Window Concept

The term *window* is one which has been coined to describe a *selected* area of the CRT. A window is created by using the appropriate sync submultiples to define the *limits* or *boundaries* of the desired area.

Figure 7.4-1 shows how the submultiple 64H

appears when displayed on the CRT with a video probe. As always, the dark areas represent the LO time periods of the signal and the light areas occur only when 64H is Hi.

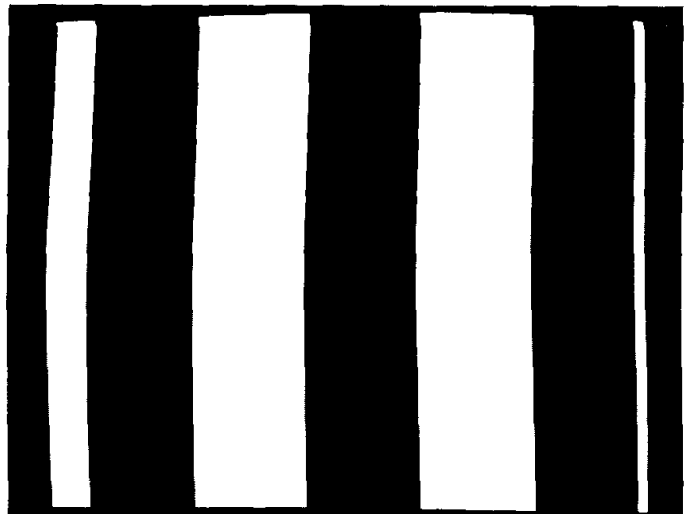


FIGURE 7.4-1 SIGNAL 64H The simplest way to achieve a video display is to couple a sync output to the video line.

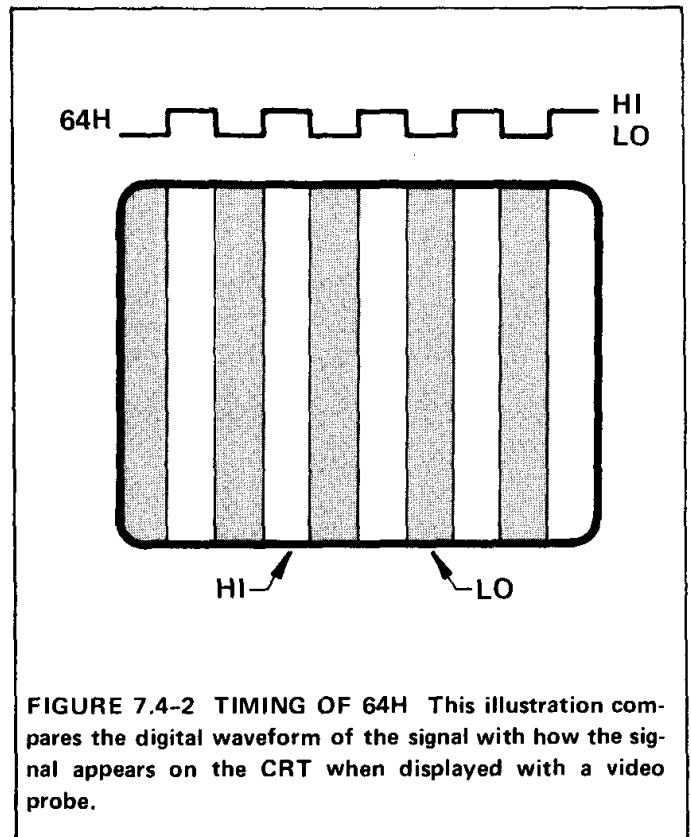
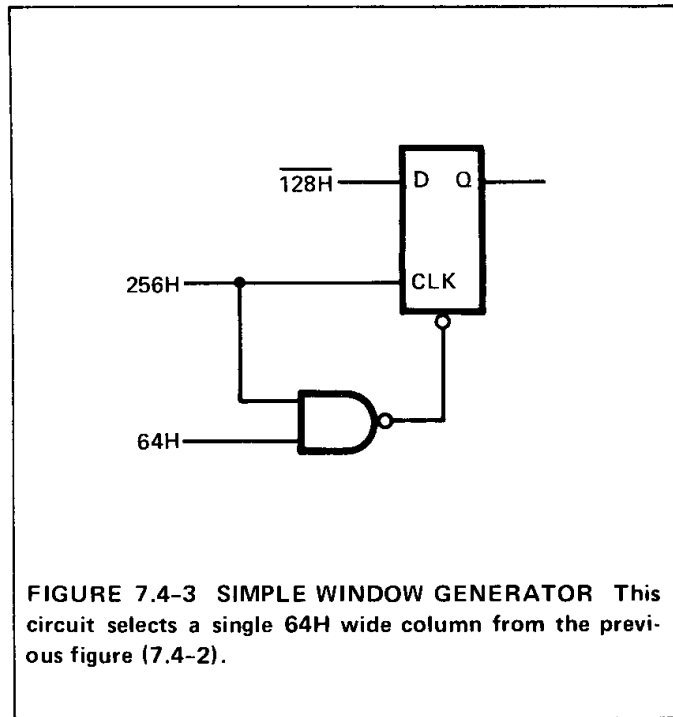


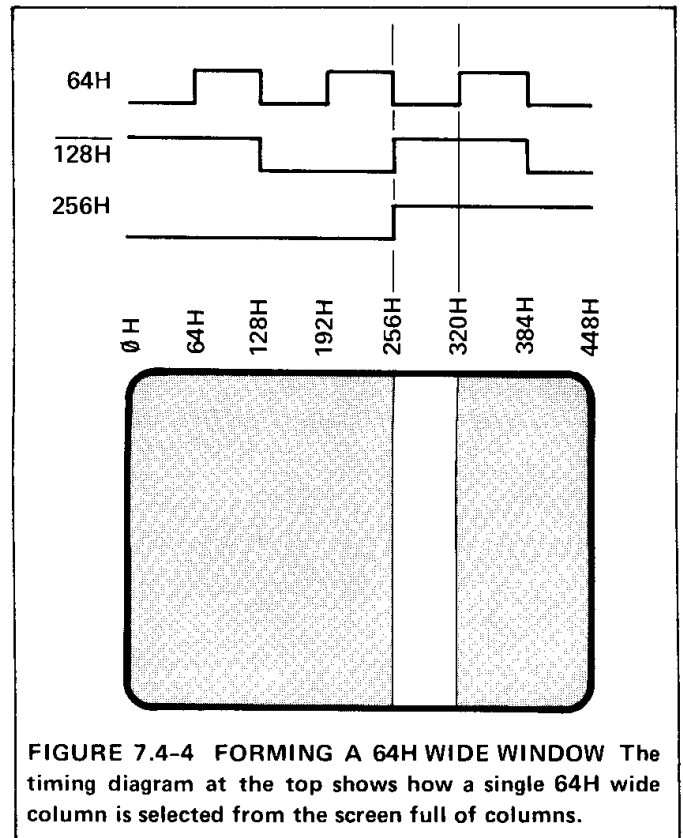
FIGURE 7.4-2 TIMING OF 64H This illustration compares the digital waveform of the signal with how the signal appears on the CRT when displayed with a video probe.

Let's bring this point home by drawing a symbolic representation of the 64H waveform compared to the light and dark areas on the CRT. The figure below shows how each LO period corresponds to a dark column.

In order to construct a window, we need to select two of the vertical lines from the signal 64H and use them to define the horizontal boundaries of the final window. So we build the small circuit in Figure 7.4-3.



Since 256H is LO in the left half of the CRT and HI in the right, we are effectively disabling the flip-flop in the left since it will wait for the transition before clocking  $\overline{128H}$  through. Consequently, any signal we get out of the flip-flop can only occur on the right side of the CRT. Therefore, when 256H rises HI,  $\overline{128H}$  is clocked out of the Q output and you can see from the waveform representations at the top of Figure 7.4-4 that  $\overline{128H}$  is also HI at this time. This clocks a HI from the Q output which lasts for a total of 64H (or 64 clock pulses) because after 64H, the output of the NAND gate clears the flip-flop and its Q output is forced LO. Since this process occurs on every line of the raster, the result is a single white vertical



column displayed slightly to the right of center. And in case you hadn't noticed, we have now specified the left and right boundaries of a window.

To form a complete window bounded on all four sides, we must go through a similar process using vertical submultiples to define a horizontal bar across the CRT. When the vertical column and the horizontal bar are ANDed together, the only area of the CRT which remains HI is the small square in the figure below.

## 7.5 An Actual Generated Display

The following is an interesting example for it typifies a contemporary technique using a generated display to create images which are positioned by ROM data rather than by generated location signals. In addition, the section also contains a clever bit of circuitry which controls which of the images are displayed.

The game which contains this circuit is a simulated combat game where players maneuver tanks around

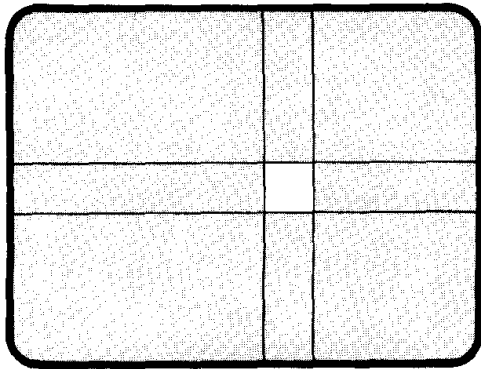


FIGURE 7.4-5 A COMPLETE WINDOW To form a complete window, the area must be defined on all four sides by combining both the horizontal and vertical windows.

a playfield while firing at each other. There are a number of white obstacles in the playfield (tank traps) which the players must drive around to get at each other and hitting one of these traps or obstacles stops the player's tank dead in its tracks. But there are also a large group of mines in the center of the playfield and if a player accidentally

contacts a mine, it blows up, taking the tank with it. The player who drove into the mine is penalized one point and is also immobilized for a short period of time which allows the opposing tank driver to maneuver into a better position.

Three separate sub-circuits are needed to fully control the mines. First, a *display circuit* generates the mine images which are confined to a particular area by a *minefield location circuit*. Finally, the exploded mines are removed and the remainder are correctly positioned by a *mine control circuit*.

The small circuit in Figure 7.5-1 contains all the circuitry required to fully develop a whole screen full of mine shapes, sixteen of which are later selected from this matrix to appear in locations determined by CRT coordinates stored within a ROM.

A basic checkerboard pattern is developed at gates F13 by the following process so that part of the checkerboard can be selected to produce the mine shapes. First, 4V and 2V are exclusively ORed with each other to generate a pattern of alternating light and dark horizontal bars (Figure 7.5-2a). This result is then exclusively ORed with a similar vertical pattern generated using horizontal submultiples 2H and 4H (Figure 7.5-2b) to develop the checkerboard pattern in Figure 7.5-2c.

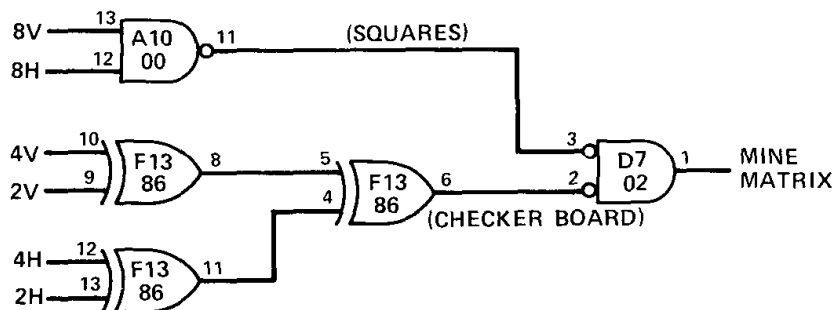
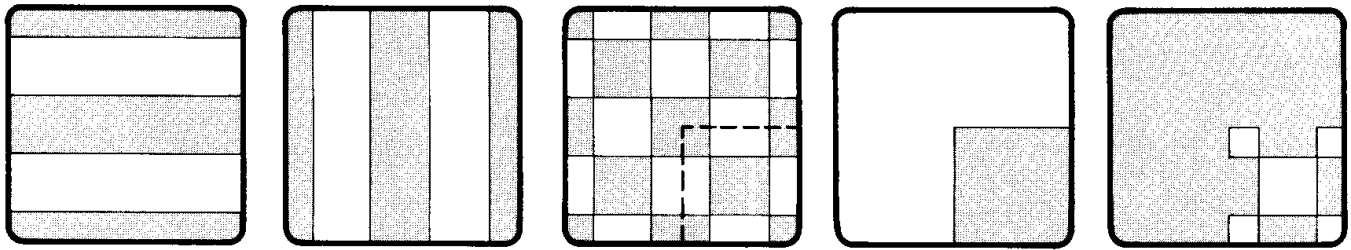


FIGURE 7.5-1 MINE MATRIX CIRCUIT This relatively simple circuit develops the mine shapes in Figure 7.5-3.



**FIGURE 7.5-2 MINE MATRIX DEVELOPMENT** This series of illustrations shows various clock submultiples combined in such a way as to eventually produce the desired shape.

A small portion of the checkerboard is selected by the result of ANDing 8V with 8H which produces a set of large squares (Figure 7.5-2d). When the squares are NANDed with the checkerboard at D7, part of the information in 7.5-2c is selected by the dark square in 7.5-2d to generate the actual mine shapes in Figure 7.5-2e.

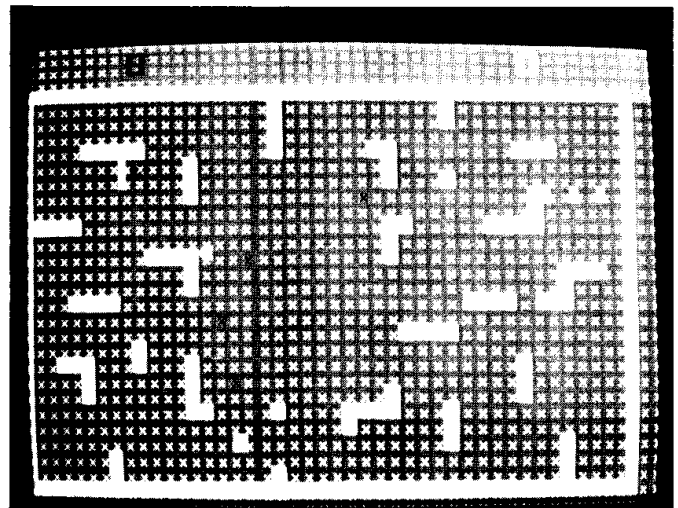
Figures 7.5-2a through 7.5-2e only show a small *cell* of the full display. In reality, the patterns shown in these figures cover the entire CRT and this is illustrated by the video probe picture, Figure 7.5-3.

Now we need to develop a general area within which the minefield can occur and this is also accomplished by using a generated or gated display technique. The circuit in Figure 7.5-4 supplies a large rectangular window in the center of the CRT into which the mine shapes are later injected.

You can see that all the inputs to this circuit are submultiples from the sync chains. The output of H13-6 drops LO at 160V and sets the R-S flip-flop composed of gates B14 which is reset by the signal from H13-8 when it drops LO at 416V. The output of the R-S is known as MEV for it supplies the vertical boundaries of the minefield which occur at 160V and 416V (but keep in mind these

vertical signals *appear* horizontal). MEH is a similar signal except horizontal in nature and it is HI between 384H and 596H (Figure 7.5-5). When the two signals are NANDed together and examined with a video probe, the result appears like Figure 7.5-6.

Now that the mine shapes have been generated,



**FIGURE 7.5-3 MINE MATRIX** Sixteen of these mine shapes are selected to appear in their proper places by location coordinates stored in memory.

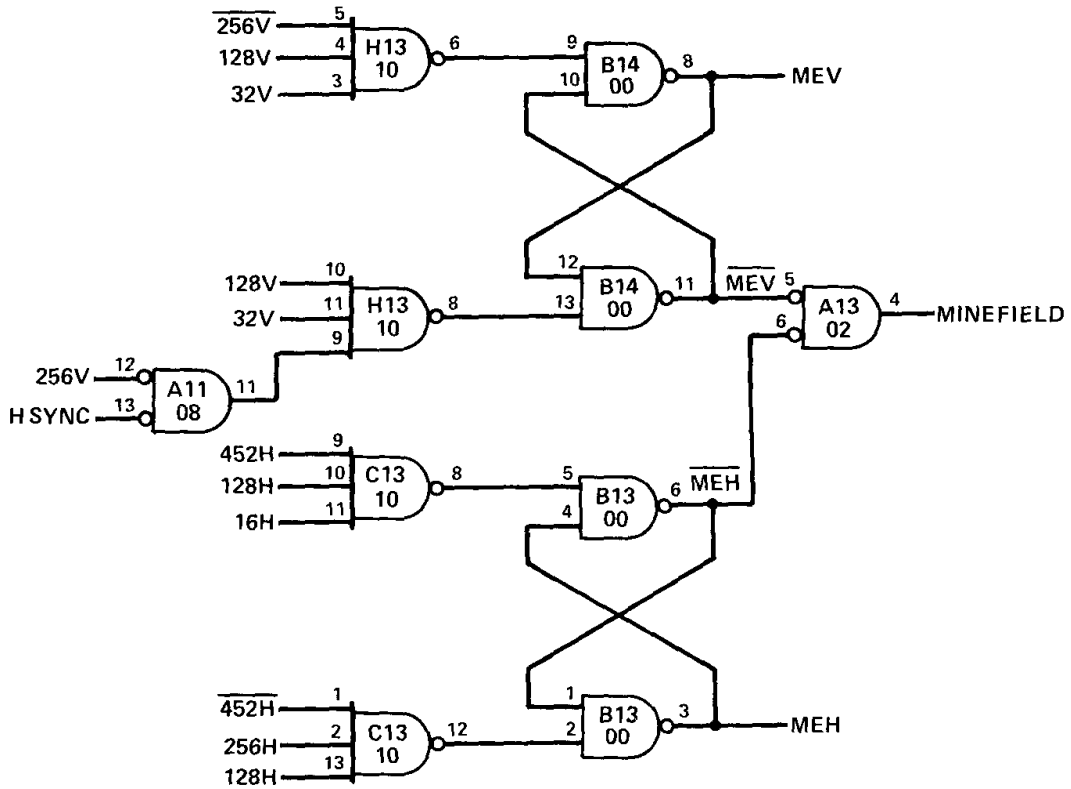


FIGURE 7.5-4 MINEFIELD CIRCUIT This section produces the large rectangular window in Figure 7.5-6.

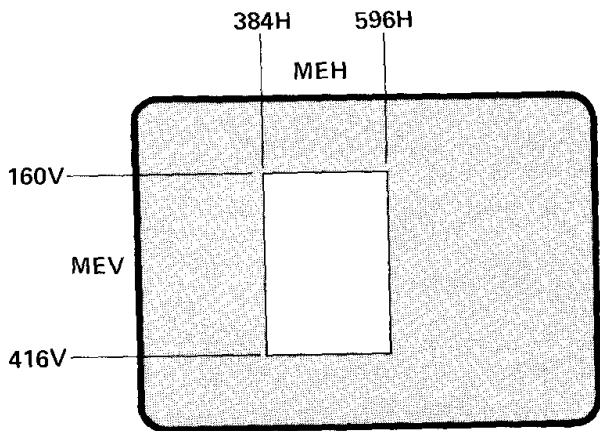


FIGURE 7.5-5 MINEFIELD WINDOW The sixteen individual mine shapes can only appear within the boundaries of this rectangular window.

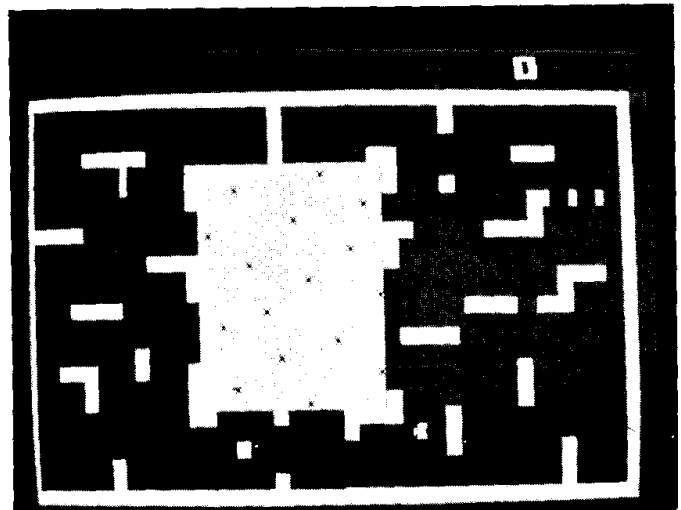
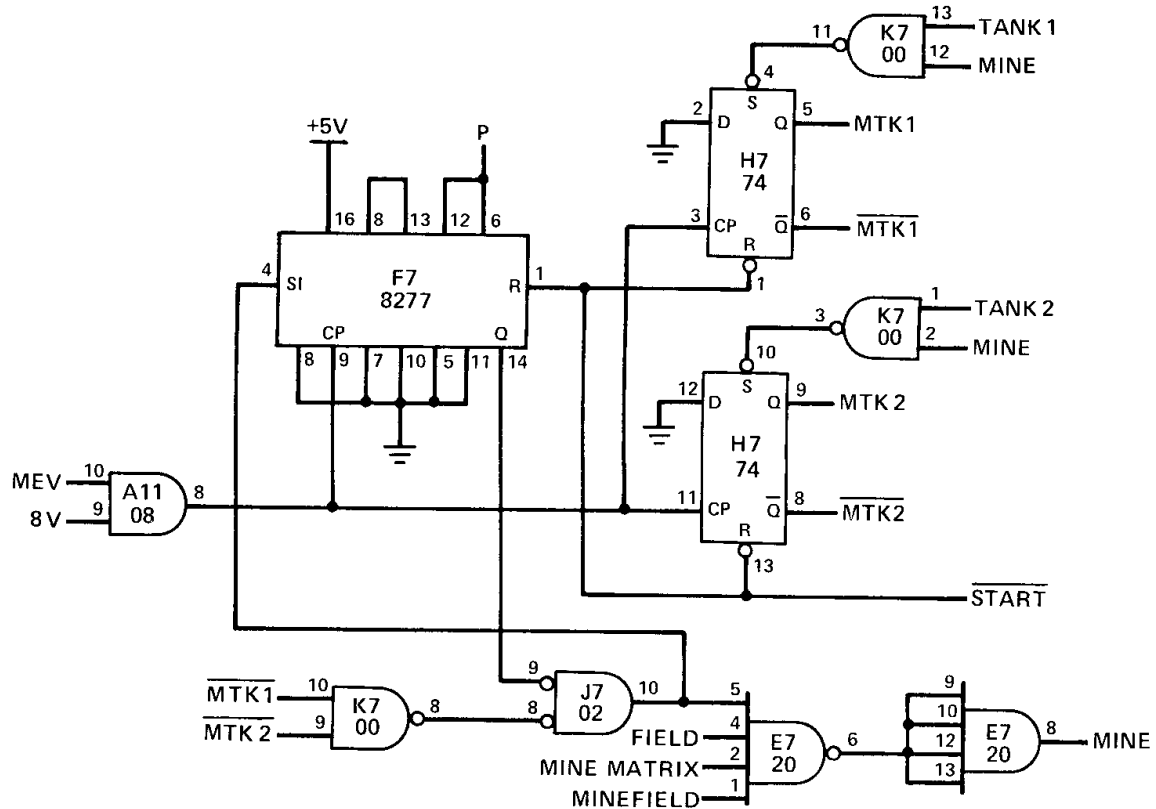


FIGURE 7.5-6 MINEFIELD This window is used to define the area in which the selected mines are to appear.

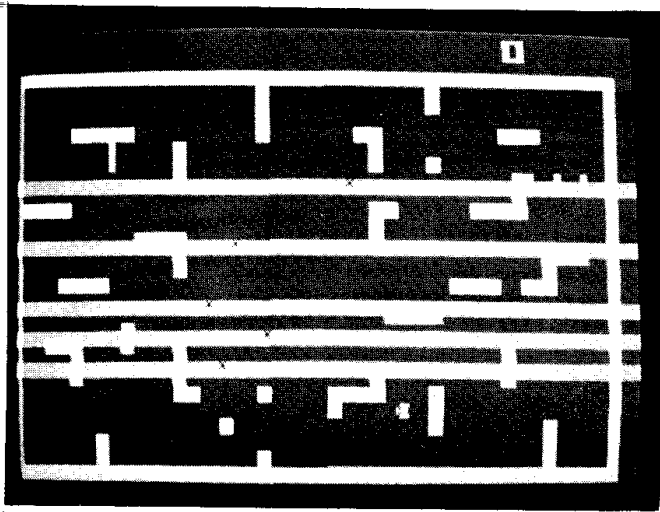


**FIGURE 7.5-7 MINE CONTROL CIRCUIT** The exploded mines are removed by this circuit which also positions the mines in their places.

provisions must be made to select only the sixteen mines we actually need as well as creating a way to subtract those mines which have been hit by a tank and exploded. The following circuit keeps track of the "live" mines, positions them in the right spots and outputs a pulse when a mine-tank collision occurs so the opposing player can be awarded a score point. The circuit operates by recirculating information through a *shift register*, a scheme used in several games for similar features. The mine information appears from the Q output of the shift register (pin 14) and passes through J7 *unless* a collision has occurred at which point the signal from K7-8 will shut off the expended mine image.

F7 is a 16-bit shift register where each bit corres-

ponds to one of the sixteen mines. When START pulses LO at the beginning of the game, zeros are loaded into the shift register which clears H7. If the first player drives his tank into a mine (TANK 1 encounters MINE at K7), a LO from K7-11 presets the upper half of H7 and  $\overline{MTK1}$  drops LO, indicating a collision has occurred. Since both  $\overline{MTK1}$  and  $\overline{MTK2}$  are connected at K7-9 and K7-10, a LO  $\overline{MTK1}$  produces a HI from K7-8 which disables the mine information from the shift register at J7 and the resulting LO is clocked into the serial input of F7, thereby shutting off that mine image when its turn comes up. On the next 8V pulse, and during the vertical mine window, the output of A11-8 clocks a HI from H7-6 and the shift register is allowed to continue to recirculate mine information. The output of J7-10 is

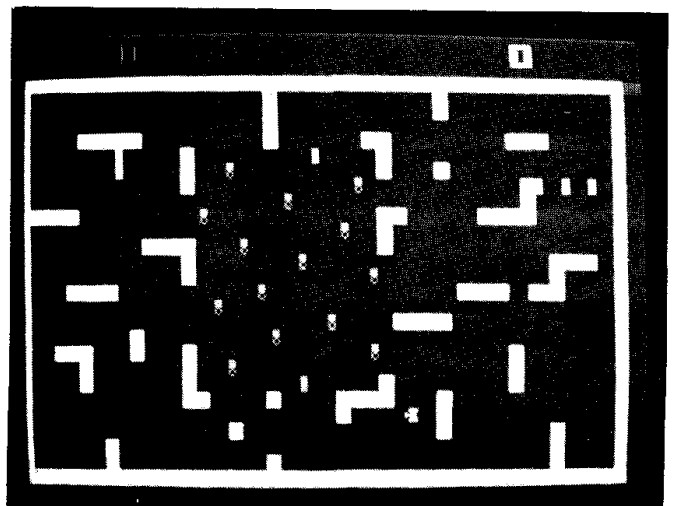


**FIGURE 7.5-8 MINE CONTROL** This signal controls which mines are blanked out when exploded. Notice that the only visible mines are covered by a white stripe. This stripe enables the mine to appear.

illustrated in Figure 7.5-8 where the horizontal white bars represent mines not yet exploded.

The selected mines from J7-10 are directed to appear in the appropriate places by the signals at NAND gate E7. MINE MATRIX is the mine shape signal which we have already discussed. MINE-FIELD is the large rectangular window generated by the circuit in Figure 7.5-4 and its inclusion here, while not immediately obvious, is quite necessary since the white tank traps coming from memory are HI and, if MINEFIELD were not included at this point, these traps would be filled in with mine shapes. For the same reason, no tank traps are placed within the minefield window.

The sixteen actual mines are also selected at gate E7 by the signal FIELD, which contains the mine location coordinates stored in the ROM. In the figure below, you can see small white rectangles over the mines where each rectangle represents one of the sixteen sets of position coordinates. Notice that although two of the mines are exploded, their position windows are still visible with the video probe.



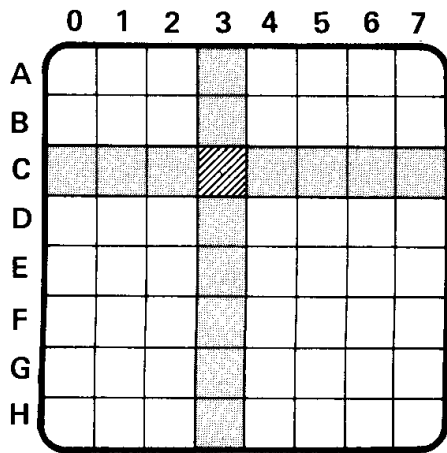
**FIGURE 7.5-9 FIELD** The mine position information stored in memory is shown here. Notice that each mine is covered by a white rectangle which represents the location information. Since two mines (very top and bottom) have been exploded their rectangles do not contain mine shapes.

## 7.6 Addressing Memory Locations

Using any kind of memory involves depositing the information in a certain location, the *address* of which is known and recorded so the information can be retrieved by looking up that address to find the location. The concept of address locations in memory is not a complicated matter, especially when you consider the fact that we do it every day without thinking about it.

An address can be either one or two dimensional. Your street address is a good example of a single dimensional address. For instance, if you live at 16 Main Street, the postman needs only to count 16 houses down your street to find the one you live in and deliver mail to you.

But let's say you get tired of finding all that junk mail in your mailbox, so you decide to get a post office box downtown and you rent P. O. Box number 3-C. A few days later, you want to find if you have received any letters, so you motor downtown, walk up to the post office counter and ask, "Are there any letters in 3-C?" The postman remembers your number and walks up to the wall



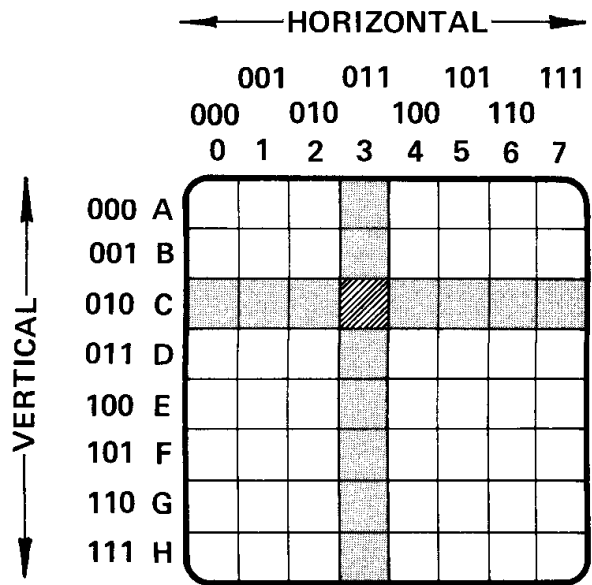
**FIGURE 7.6-1 POST OFFICE BOXES** The location of your post office box is at the intersection of the 3rd vertical column and the C row. The postman finds your letters by scanning across the top and down the side simultaneously until he finds the right box.

of boxes which look something like Figure 7.6-1.

In order to find your box numbered 3-C, he runs one hand down the third column and the other across the "C" row. Your box is at the intersection and — sure enough — there is a letter inside. So, he gives you the letter and you go on your way.

Now, all we have to do is call the letter the "data", and we have just created a post office "memory" system. The location of the box (3-C) is the address for that "data" and you can see that it makes no difference what kind of data is at the specified location for it could have also been a parcel and the postman would still have been required to hand it over to you. The address for the data contains two types of information of directions. The number "3" specifies the *horizontal* location of the column which contains your box and the letter "C" supplies the *vertical* coordinate of the box.

Although a postman can use decimal numbers and alphabetical letters to find the location of the box, a computer must use a binary code to deter-



**FIGURE 7.6-2 BINARY LOCATION OF BOX** A computer works the same way, except it uses binary counters to scan the memory. As each set of counters increments up, every box is addressed and the data contained within is read out.

mine locations. If we want to replace the postman with a computer, we must re-label the boxes according to the figure above.

All we have done in this figure is to assign a binary number to each row and column so the box we are looking for is at the intersection of column 010 and row 010. Row and column addresses in a computerized memory system are derived by using a set of binary counters which — in this case — would begin at 000 and end at 111.

### 7.7 A Diode Matrix Display

A simple memory circuit known as the *diode matrix* can be set up the same way we worked the post office analogy, except in this case the data we find in the box can only be either a HI or a LO rather than a letter or a parcel.

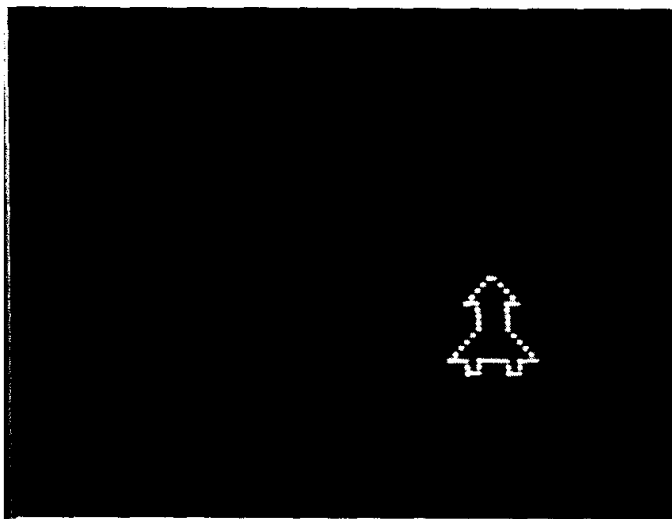
The diode matrix is an important and interesting type of circuit, not because it is frequently used, but for what it reveals about how all memory



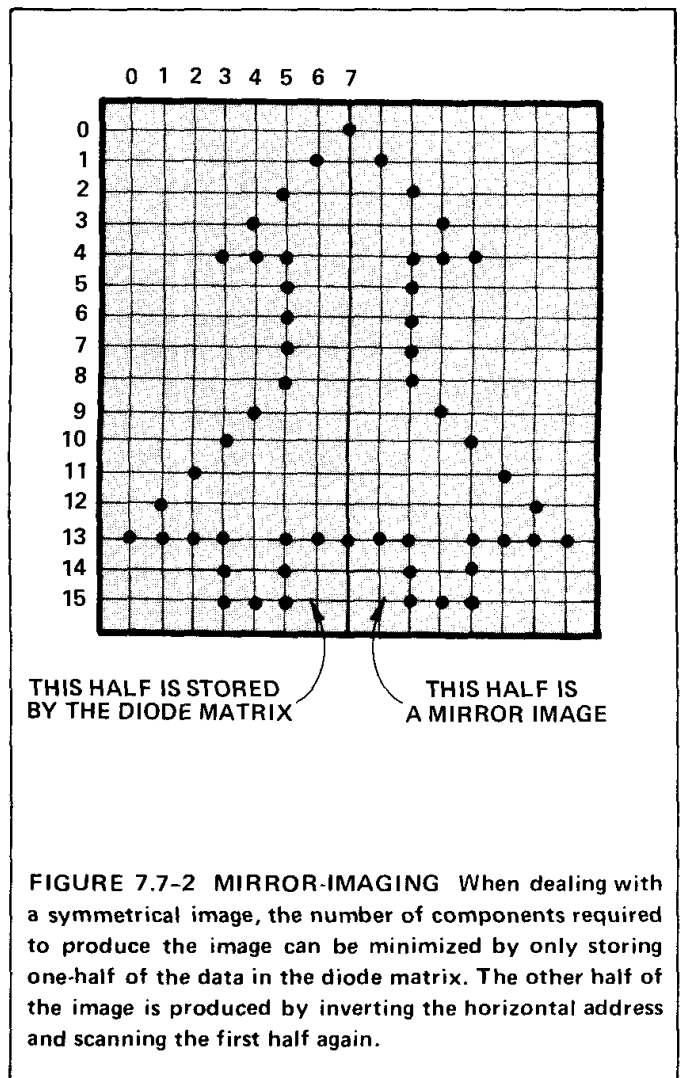
systems are constructed because the insides of an integrated ROM or RAM are constructed just like the address scheme in Figure 7.8-1. Once you have learned how a diode matrix works, you will never be confused about the operation of any other memory device because whether it is static, dynamic, read-only or read-and-write, the address technique for all memory systems is the same.

The diode matrix has been around a long time and the techniques implemented for this type of circuit laid the groundwork for later developments in the semiconductor industry. Essentially, an integrated ROM is no more than a whole bunch of "diodes" jammed into a very small package along with all the address decode circuitry required to look up the information.

Using a diode matrix (or any kind of ROM for that matter), a designer can place a pattern of dots anywhere on the CRT. The spacial relationship between the dots can easily be changed simply by moving the diodes around in the matrix to develop a new shape or to correct a mistake in the



**FIGURE 7.7-1 A DIODE MATRIX IMAGE** Any simple image such as the one above can be stored in a diode matrix — the earliest form of read-only memory. The memory is scanned and the data read out by two sets of counters. The vertical counter selects each line of the image while the horizontal one scans each point along the line.

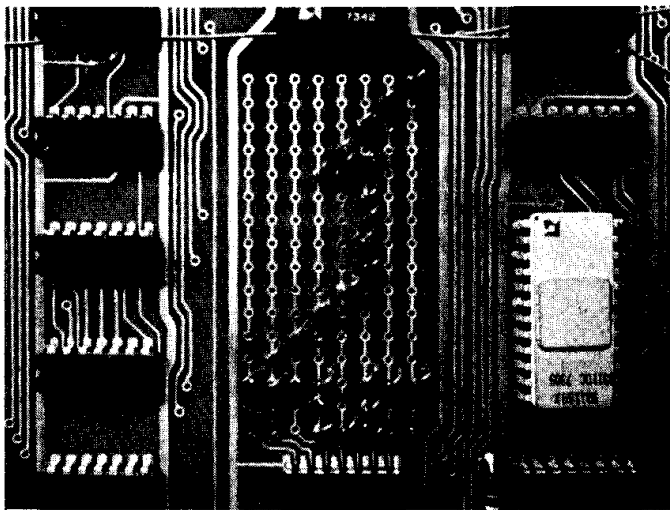


old one. If the designer wants a moving pattern of dots, the matrix can be addressed with a motion code count which will cause the dot pattern to move along with the motion/sync circuit differential.

To create the image in the photograph above, the designer sat down with a piece of graph paper, laid out a 16 by 16 grid and plugged in the points which would produce the shape he had in mind. Because of the fact that the image is symmetrical about the vertical axis, only one-half of the total number of points requiring illumination by the electron beam actually need to be laid out since the other half of the image can be produced by "mirror imaging" the first half. In this way, an 8 by 16 (or 128-bit) diode matrix can be made to supply all the information necessary to generate

a displayed image which occupies a 16 by 16 or 256-bit grid.

The photograph below shows how the matrix actually appears on an engineering prototype board and you can see that the physical layout of the components corresponds closely to the schematic circuit. The prototype board has all 128 possible locations drilled to accept a diode to allow the designer some flexibility in changing the design. This way, if he finds the initial shape unsatisfactory he needs only to unsolder and move a few diodes. Production boards for this game, however, only have the required holes drilled to minimize cost.



**FIGURE 7.7-3 ACTUAL DIODE MATRIX** This photograph shows how the diode matrix which generates the rocket image appears on one of the engineering prototype boards.

### 7.8 How It Works

The matrix is *scanned* vertically by the 9311 one-of-sixteen decoder and horizontally by the 9312 multiplexer. The inputs to the decoder are vertical submultiples and these signals represent a count which starts at 0 0 0 0 and ends at 1 1 1 1. Later we will see how the matrix can be scanned vertical-

ly by a motion code to make the image move up and down.

The multiplexer, on the other hand, is selected by a horizontal count which does some interesting things, as we shall see in a moment. Since this is an eight bit multiplexer, only three inputs are required to select any one of the inputs. But the most important thing to keep in mind when figuring out how the scan works, is that the vertical code changes much more slowly than the horizontal.

The diodes are inserted into their places because a connection needs to be made at each of these points, but the lines cannot simply be soldered together or a short will result since all the lines are pulled up to +5 volts by pull-up resistors.

The type of sub-circuit constructed from the three exclusive ORs and used to select the multiplexer is a common design employed whenever it is necessary to count up to a certain point and then reverse the count and go backwards down to the beginning. The count begins at 0 0 0, goes up to 1 1 1, repeats 1 1 1 and returns to 0 0 0. The process by which the type of count is achieved is illustrated in the truth table below.

The truth table is divisible into two sections: (1) the count from sync into the gates and (2) the count from the gates which selects the multiplexer. In the first section, the submultiples 1H, 2H, 4H and 8H form a regular binary count starting from 0 0 0 0 (decimal 0) and ending at 1 1 1 1 (decimal 7) and then descends back down to 0 0 0 0 again. The fact that we can generate a count which reverses itself like this allows us to generate a symmetrical image from a 128-bit matrix which would otherwise require a full 256 bits.

So here is what happens as the electron beam scans the CRT. During the first line of the raster, the vertical count into the decoder remains at 0 0 0 0 since 1V, 2V, 4V and 8V are all LO for the entire line. During the time the vertical sync chain remains at 0 0 0 0, the horizontal chain counts all

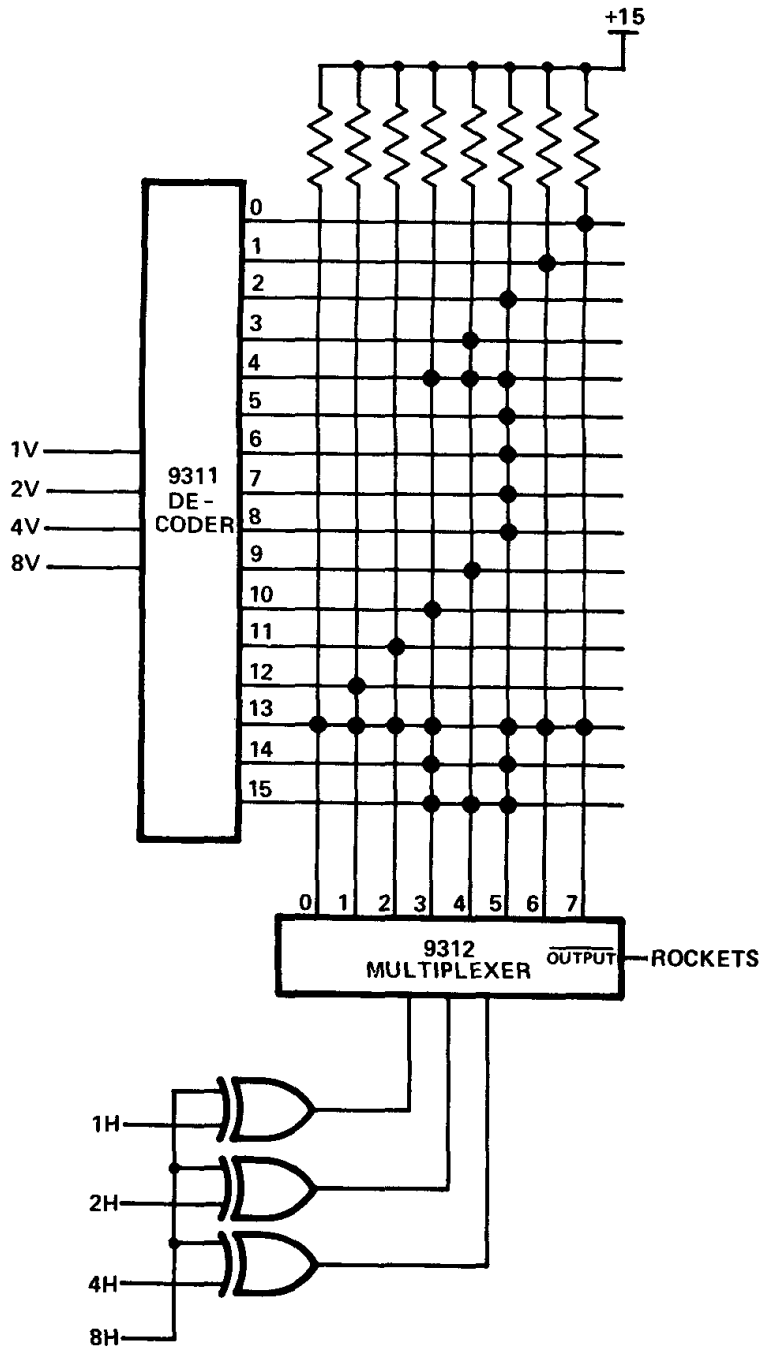


FIGURE 7.8-1 DIODE MATRIX CIRCUIT The matrix is scanned vertically by the decoder and horizontally by the multiplexer. As the decoder selects each line, the

multiplexer reads out all the data contained on that line. The exclusive OR gates at the bottom are responsible for inverting the horizontal scan and mirror-imaging the data.

	1H	2H	4H	8H		1H+8H	2H+8H	4H+8H
0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	1	0	0
2	0	1	0	0	2	0	1	0
3	1	1	0	0	3	1	1	0
4	0	0	1	0	4	0	0	1
5	1	0	1	0	5	1	0	1
6	0	1	1	0	6	0	1	1
7	1	1	1	0	7	1	1	1
8	0	0	0	1	7	1	1	1
9	1	0	0	1	6	0	1	1
10	0	1	0	1	5	1	0	1
11	1	1	0	1	4	0	0	1
12	0	0	1	1	3	1	1	0
13	1	0	1	1	2	0	1	0
14	0	1	1	1	1	1	0	0
15	1	1	1	1	0	0	0	0

FIGURE 7.8-2 COUNT INVERSION The table at the left shows a regular binary count starting at 0 0 0 0 and ending with 1 1 1 1. But, when this count is entered into the exclusive ORs (Figure 7.8-1) on the right, the outputs of the gates count from 0 0 0 to 1 1 1 and then back down to 0 0 0 again.

the way from 0 to 454. This means that the count at the decoder remains at zero, but the count at the multiplexer selects increments rapidly. The code at these selects starts at 0 0 0, increments to 1 1 1, repeats 1 1 1 and starts back down to 0 0 0 while the vertical address remains constant.

So, as the electron beam is scanning the first raster line, the code 0 0 0 selects the first output of the decoder which is connected to the top row of the matrix. Then, the code at the multiplexer starts counting up. The first multiplexer code (0 0 0) selects the leftmost column of the matrix and since there is no diode at the intersection of the top row and leftmost column, the pull-up causes the multiplexer to output a LO (it would be a HI except for the fact that we are using the inverted output of the multiplexer). The next code into the

multiplexer is 0 1 0 and there is again no diode at the intersection, so the multiplexer continues to be selected by the ascending code. But when the address reaches the 1 1 1 (7), a diode connection is encountered and a LO is pulled through the decoder causing the output of the multiplexer to rise HI at this time. This HI causes intensification of the electron beam and it puts a bright white dot in the first line of the raster.

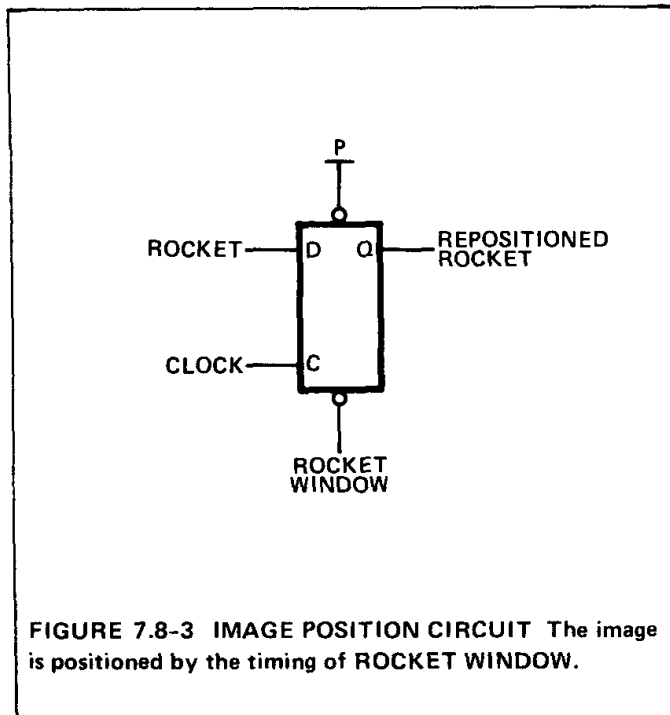
Since the count into the multiplexer reverses itself after the number 7, the multiplexer is again addressed by the select code 7 and since this looks at the rightmost line of the matrix *again*, another dot is deposited right next to the first one. But these two dots are so close together that they are perceived as a single point of light for there is absolutely no detectable separation between them. We have now just created the "nose cone" point of the rocket image.

On the second line of the raster, the vertical address is incremented by one count to 1 0 0 0 which enables the second row of the matrix. This code remains at 1 0 0 0 until the horizontal address has scanned the matrix from left to right and back down from right to left. So, while we are still on the second line, the multiplexer finds no diodes in the first six positions so it outputs a string of LOs. But at the seventh position, a diode connection is encountered, the line is pulled LO and the multiplexer outputs a HI causing the electron beam to deposit a white dot in the second line of the raster. The address of the multiplexer is still counting up and since it finds no diode in the eighth column, no dot is placed. At this point, the horizontal address flips over, the multiplexer finds no connection again at the eighth position so it decrements by one more count to the seventh position where it does find a diode which causes the electron beam to repeat the dot. But since the electron beam has now moved some distance, this dot appears further over to the right and the separation between the dots allows the eye to perceive them as two distinct points.

The address continues in this fashion until all the

vertically-numbered rows have each been scanned and all the diode connections on each row have been read out twice on the CRT. Since all the lines of the raster are stacked on top of one another, a full rocket image results.

We now have a nice rocket image on the CRT. But it is stuck up in the upper left corner of the CRT. To position the image somewhere else in the screen, all we need to do is build the small circuit in Figure 7.8-3.



Let's say that we have also constructed a Rocket Window circuit which produces a HI signal from 256H to 272H. Whenever this signal is LO, the Q output of the flip-flop will also be forced LO, disabling the image. But when ROCKET WINDOW rises HI at 256H, the rocket image signal at the D input will be clocked through and appear at the output.

Now we have a nice little rocket image stuck in the center of the CRT at the top of the screen. In order to entice the player into using the game, we need to make provisions for him to move the rocket. So we build a standard motion circuit which produces the normal count of 0 0 0 0 to

1 1 1 1 which begins and ends with a certain time differential with respect to when the sync count begins and ends. To enable vertical rocket motion, all we need to do is enter this motion count at the inputs of the decoder instead of the sync count 1V, 2V, 4V and 8V.

## 7.9 Integrated ROMs

Integrated ROMs are constructed and operated in much the same way as the diode matrix in the previous section, however the entire circuit is greatly reduced in size and placed on a single, *monolithic* chip. The degree of size reduction has enabled the device manufacturers to offer ROMs which contain a very large amount of information in a relatively inexpensive 16 to 24-pin DIP and it is this capability which makes ROMs so attractive to designers. ROMs which contain over 16K bits are quite common in video games which use a memory to generate all the video displays. Can you imagine the amount of circuitry that would be required to store 16K bits of data in a diode matrix? Just the cost of the diodes alone would be astronomical. For example, to program all 16,384 locations with diodes costing only \$0.05 each would cost over \$800 and require a printed circuit board of several hundred square inches! Now, compare this to an integrated 16K ROM which costs only \$15.00 and contains a chip only 250 mils on a side (about a ¼") and you can see why integrated ROMs are so widely used.

Although the primary usefulness of the integrated ROM in video games thus far has been to store *image data*, other uses are beginning to be seen as well. In the example circuit found later in this chapter, a ROM is used not only to store image information, but also for motion code data. Games using either integrated microprocessors or hard-wired processors also use ROMs to store image data, but in addition these games must also have a ROM — or part of a ROM — to store the *game program* which is a set of instructions that directs the processor to perform various operations. In other digital applications, ROMs are also used for *code conversion*, *character generation*, *look-up*

tables and to provide the *equivalent* of logic functions for control circuits.

All read-only memories are *fixed address* devices meaning the data is entered into a known, *pre-determined* location and can only be retrieved from that location by entering the correct address code. ROMs are also characterized by the fact that the data is essentially unchangeable once it is entered or *programmed* into the device. Although electrically *reprogrammable* ROMs are now available, reprogramming is a complicated and tedious task.

Read-and-write memories (RAMs), of course, are programmed continuously by the system which contains the device and — when used in a video game — the entire contents of the memory may be updated once per frame. RAMs are necessary in a system whenever a processor-type architecture is used. In spite of their greater programming flexibility, RAMs have several serious drawbacks which limits their usefulness in most systems. Not only are RAMs larger and more costly, but their contents are also *volatile*, meaning that whenever the power is turned off, the stored information is forever lost. Computers using RAMs must have emergency back-up systems to instantaneously provide power should a local power failure occur and they usually must also have a *low-power mode* where only enough power required to *refresh* the memory is used.

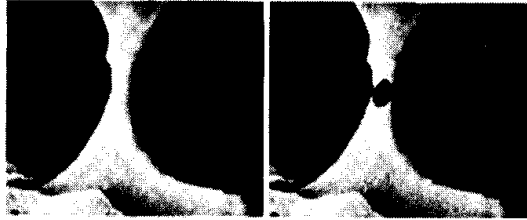
The last few years have seen tremendous proliferation in types of ROMs and RAMs as well as a great deal of expansion in the amount of data these devices can hold. ROMs are currently available using both bipolar and metal-oxide technologies, however MOS techniques are generally used for the larger ROMs since this technology is capable of greater size reduction. Depending on the particular application, one of two general types of ROMs can be used: *mask programmed* or *field programmed*. Mask programmed ROMs are relatively inexpensive and very reliable, however the initial mask cost can be quite high and the data sent to the manufacturer must be complete and

absolutely correct in every respect. Field programmable devices (PROMs) offer in-the-lab programming capability to the engineer and they may be either erasable or non-erasable. While there is no initial large mask charge for the field programmable devices, the actual device itself is more expensive and some additional expense is usually incurred in the programming process.

No matter what kind of ROM you plan to use, the first step is organizing your information so it can be entered into the device and — since the data in a mask programmed device is unchangeable — this information must be absolutely correct. The information is first assembled in the form of truth tables and then entered into punched tape or cards so it can be tested using a ROM simulator, since errors found at this stage are easily corrected. The verified information is then sent to the manufacturer of the devices in the format that company prefers.

The manufacturer feeds the cards or tape into a numerically-controlled mask cutting machine which uses the data to direct a stylus to cut precisely positioned holes in the rubylith mask. This custom mask is then used to selectively etch areas of the final metalization layer so that your 1s and 0s are correctly located. The primary advantage of this technique is that the resulting chips can be mass produced on an inexpensive basis. However, the mask charge is generally prohibitive for small runs and experimental work.

This brings us to *fusible link* ROMs. Once entered, the data in the ROM is also unchangeable, although un-blown bits can be programmed at a later date if necessary. The final layer of metalization in this type of ROM is provided with a *fuse* for each bit to be programmed. Depending on who makes it, the fuse may be either nickle-chrome or poly-silicon and the unprogrammed bits may be either HI or LO. During programming, the address for the bit to be blown is entered at the inputs and a special set of pulses is fed into the device outputs. Figure 7.9-1 shows a typical fuse before and after



**FIGURE 7.9-1 PROGRAMMING A PROM** These two photographs show a single fuse of a PROM before and after programming. A special set of pulses fed into the PROM's outputs "blow" the addressed bit.

the programming process.

The polysilicon process produces a more reliable device because the nichrome tends to "regrow" after the fuse has been blown, which may cause an intermittent or permanent reconnection. Also, when a polysilicon fuse is blown, it oxidizes completely and there is no possibility of conductive residue shorting out other circuit parts. In contrast, blowing a nichrome fuse does tend to produce conductive residue.

The fusible-link ROM offers a neat solution to situations where a designer needs to either test his program before having a mask cut or where only a few ROMs are needed for a limited production run. But once a bit has been programmed, the process can never be reversed. If reprogrammability is required, the designer may either blow a new fusible-link ROM or he may go to an entirely different type of device known as the *electrically reprogrammable PROM*.

These PROMs are ideally suited for applications where bit-pattern experimentation and non-volatility are important considerations. The typical PROM as this variety is provided with a *transparent* quartz lid which allows the user to expose the chip to ultraviolet light and erase the existing bit pattern so that an entirely new pattern can be written into the device.

The storage medium for the data is a silicon-gate MOS transistor where the gate of the transistor is left floating and completely surrounded by oxide. The device is manufactured so the floating gate has no charge and all the transistors are in the off or non-conducting state. However, avalanching the transistor produces electrons with sufficient energy to bridge the thin gate oxide layer which charges the gate negatively and, with a sufficient negative gate charge, the transistor will enter the conducting state.

Once the gate is charged, considerable energy is required to give the electrons sufficient velocity to escape the gate. Sunlight and ordinary illumination are insufficient and thermal excitation associated with normal operating temperatures is also far too small. X-rays and stray radiation may discharge the gate, but only in quantities far in excess of those fatal to humans. However, special short-wave ultraviolet radiation very effectively erases the device by sufficiently exciting the trapped electrons. Therefore, to erase the contents of the PROM, all one needs to do is hold an ultraviolet light source above the device for a few minutes. After completely erasing the existing bit pattern, a new set of data may be entered.

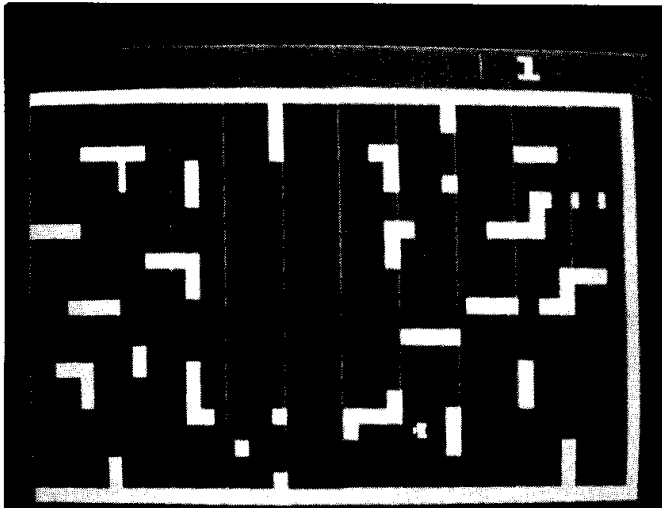
### 7.10 An Integrated ROM Circuit

The ROM in Figure 7.10-6 not only stores all the information necessary to generate two complete playfields, 64 different score numerals and the various "tank" image aspects for this game, but also all the motion codes necessary to produce realistic motion for each tank aspect. This particular ROM is a 2048-word by 8-bit device (16,384 bits in all) organized into four 512-word *pages* of data where each page stores a completely different type of information.

The different pages of data are accessed by the *highest order address* formed by the two MSBs ( $A^9$  and  $A^{10}$ ) of the ROM address. For example, when the page access code 0 1 is entered at the two most significant address bits, the score data block is accessed. Once the correct page has been

accessed, the *lower order addresses* read out the specific information to generate the desired shapes.

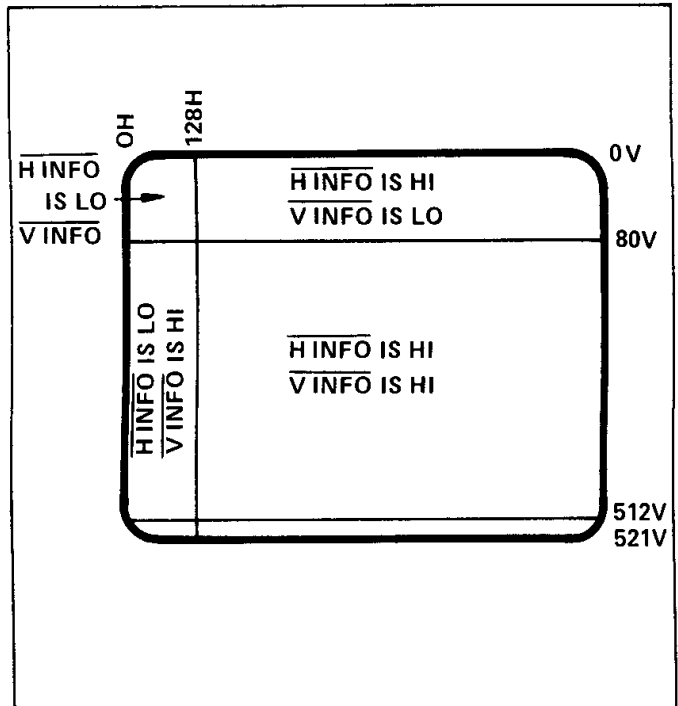
The specific lower order addresses are selected by the states of  $\overline{\text{HINFO}}$  and  $\overline{\text{VINFO}}$  at multiplexers D9, E9, J9, H9 and F9 according to the scheme presented in Figures 7.10-1 and 7.10-2. The development and operation of  $\overline{\text{HINFO}}$  and  $\overline{\text{VINFO}}$  have already been discussed in sections 5.10 and 5.12 of the interlaced sync analysis in preparation for this chapter and their presence here is quite important.



**FIGURE 7.10-1 A STORED PLAYFIELD** All the images in this photograph are stored within an integrated ROM. Varieties of integrated read-only-memories allow designers to store a relatively large amount of data on an inexpensive basis.

Since the A and B selects of the multiplexers are controlled by  $\overline{\text{HINFO}}$  and  $\overline{\text{VINFO}}$  respectively, these signals determine which input instructions are allowed through the multiplexers. Figure 7.10-1 illustrates the states of  $\overline{\text{HINFO}}$  and  $\overline{\text{VINFO}}$  in relationship to the raster timing. In other words, when  $\overline{\text{HINFO}}$  is HI and  $\overline{\text{VINFO}}$  is LO, the multiplexers enable the score signal inputs through the outputs to address the ROM for the score numeral image data.

Therefore, whenever the electron beam is in the *score data area* of Figure 7.10-3, the multiplexers are addressing the ROM for the score numeral

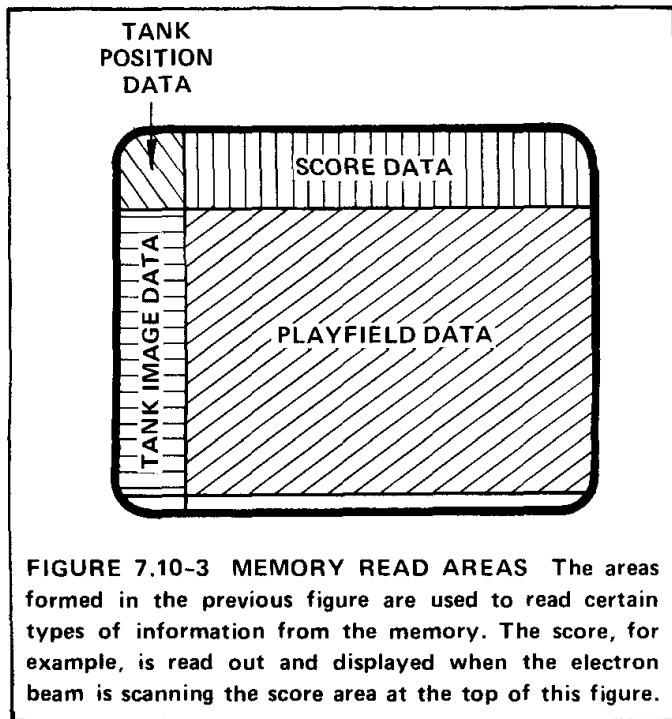


**FIGURE 7.10-2 H & V INFO** The states of these two signals are used to divide the CRT into a number of selected areas or windows.

information and, since this data is entered directly into the video summation network, the score numerals all appear within this rectangle. Likewise, the playfield is read out as the electron beam is scanning the large *playfield data area* and this is also the area in which the playfield actually appears. Although the *tank position data* and the *tank image data* are read out between 0H and 128H, this information is not displayed in this area of the CRT for two reasons: (1) the area between 0H and 128H is not visible on the CRT and (2) the tank position information is used for motion control rather than display purposes and the appearance of the tank image data is controlled by the motion window so that it can appear anywhere within the playfield area.

Another way to achieve a slightly different view of the address formation process is with a truth table such as the one in the figure above. This table shows the type of information read from the ROM compared with the multiplexer selects and the actual ROM address. For example, if the A and B selects are HI and LO respectively, the score



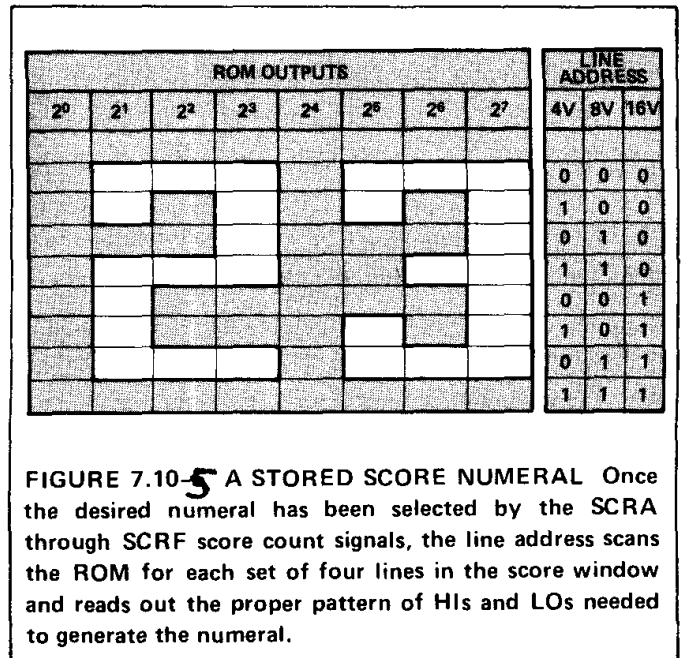


**FIGURE 7.10-3 MEMORY READ AREAS** The areas formed in the previous figure are used to read certain types of information from the memory. The score, for example, is read out and displayed when the electron beam is scanning the score area at the top of this figure.

memory page (0 1) is accessed while the binary score count addresses the ROM for the numeral image data.

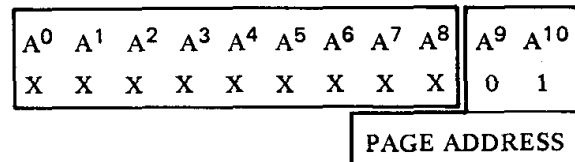
Now that we have described the general address procedure, we can turn our attention to the specific operation of this circuit during each of its four data read modes.

**SCORE DATA:** During the score window area in Figure 7.10-3, the A and B selects of all the multi-



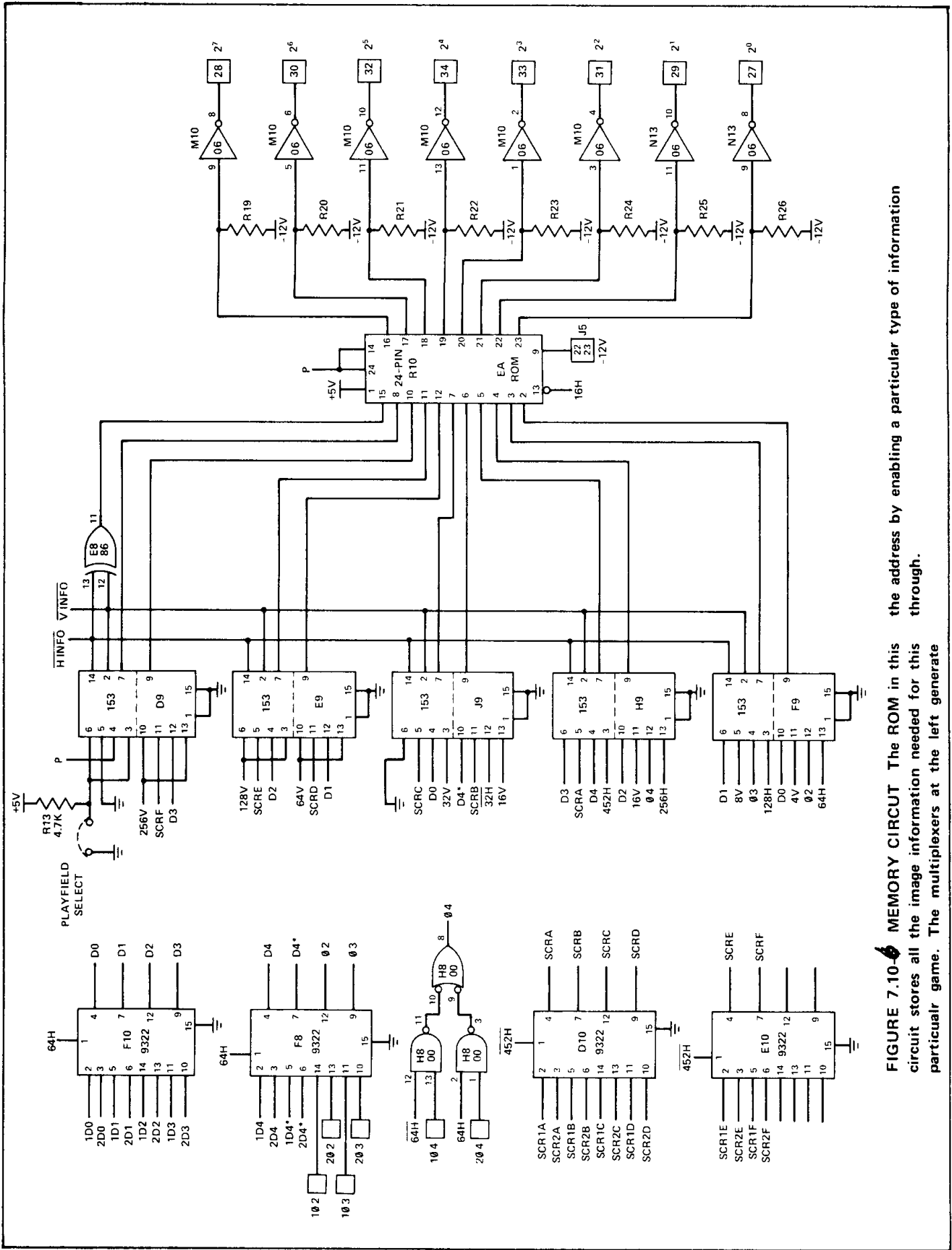
**FIGURE 7.10-5 A STORED SCORE NUMERAL** Once the desired numeral has been selected by the SCRA through SCRF score count signals, the line address scans the ROM for each set of four lines in the score window and reads out the proper pattern of HIs and LOs needed to generate the numeral.

plexers are HI and LO respectively. Since these are all dual four-to-one devices, the binary select code 1 0 enables the signals at input pins 5 and 11 to appear at output pins 7 and 9 of each multiplexer. Since HINFO is HI at this time, E8-11 is also HI when the LO at D9-5 is selected to appear at D9-7. This produces a page address code of 0 1 at the A<sup>9</sup> and A<sup>10</sup> address inputs, so the address at this point is:



INFORMATION	MUX SELECTS		ROM ADDRESS INPUTS										
	A	B	A <sup>0</sup>	A <sup>1</sup>	A <sup>2</sup>	A <sup>3</sup>	A <sup>4</sup>	A <sup>5</sup>	A <sup>6</sup>	A <sup>7</sup>	A <sup>8</sup>	A <sup>9</sup>	A <sup>10</sup>
POSITION	0	0	D0	D1	D2	D3	D4*	0	64V	128V	256V	1 or 0	0
SCORE	1	0	4V	8V	16V	SCRA	SCRE	SCRC	SCRD	SCRE	SCRF	0	1
TANK IMAGES	0	1	02	03	04	D4	32H	D0	D1	D2	D3	1	1
PLAYFIELD	1	1	64H	128H	256H	452H	16V	32V	64V	128V	256V	1 or 0	0

**FIGURE 7.10-4 ROM ADDRESS TABLE** This figure illustrates the process by which data is read from memory. The type of data (left) is selected by the multiplexer selects and read out by the signals at the ROM address inputs.



**FIGURE 7.10 MEMORY CIRCUIT** The ROM in this circuit stores all the image information needed for this particular game. The multiplexers at the left generate the address by enabling a particular type of information through.

Since the page address contains the two MSBs, it will remain in this configuration the entire time the remaining nine address bits go through their counting operations. And now that the ROM is "opened" to this page, it is ready to receive the actual numeral address.

The next six MSBs of the address contain the actual binary equivalent of the player's score. For example, let's say the player has accumulated a total of 23 points. The counters of the score storage circuit will express this by outputting the binary equivalent 1 1 1 0 1 0 at the SCRA through SCRF output pins. So, while the electron beam is within the score window, the address will look like:

A <sup>0</sup>	A <sup>1</sup>	A <sup>2</sup>	A <sup>3</sup>	A <sup>4</sup>	A <sup>5</sup>	A <sup>6</sup>	A <sup>7</sup>	A <sup>8</sup>	A <sup>9</sup>	A <sup>10</sup>
X	X	X	1	1	1	0	1	0	X	X
			PLAYER'S SCORE							

This part of the address specifies an area of the ROM which contains the pattern of HIs and LOs that will generate the number 23 on the CRT. We have now selected the score page of the ROM and the internal location of the score image data, but the question still remains of how the electron beam actually displays this numeral.

The actual numeral is read out by the three least significant address bits in a process similar to the vertical address scheme used in the diode matrix example (Section 7.8). These three LSBs are controlled by vertical submultiples 4V, 8V and 16V which go through a regular binary count beginning at 0 0 0 and ending at 1 1 1 so that the ROM is addressed for each raster line of the score window. Consequently, the address during the first four lines of the score is:

A <sup>0</sup>	A <sup>1</sup>	A <sup>2</sup>	A <sup>3</sup>	A <sup>4</sup>	A <sup>5</sup>	A <sup>6</sup>	A <sup>7</sup>	A <sup>8</sup>	A <sup>9</sup>	A <sup>10</sup>
0	0	0	X	X	X	X	X	X	X	X
LINE ADDRESS										

Now that we have filled in the three LSBs, we have a complete address of

A <sup>0</sup>	A <sup>1</sup>	A <sup>2</sup>	A <sup>3</sup>	A <sup>4</sup>	A <sup>5</sup>	A <sup>6</sup>	A <sup>7</sup>	A <sup>8</sup>	A <sup>9</sup>	A <sup>10</sup>
0	0	0	1	1	1	0	1	0	0	1

which reads a single line of memory so that an eight-bit word is dumped out the parallel ROM outputs (2<sup>0</sup> through 2<sup>7</sup>) which contain the required pattern of HIs and LOs necessary to generate the first row (0 0 0) of the numeral in Figure 7.10-5. Since the LSB is 4V, the electron beam scans out this pattern for a total of four lines down the CRT at which point the code at the three LSBs of the address is incremented to 1 0 0 for the new address

A <sup>0</sup>	A <sup>1</sup>	A <sup>2</sup>	A <sup>3</sup>	A <sup>4</sup>	A <sup>5</sup>	A <sup>6</sup>	A <sup>7</sup>	A <sup>8</sup>	A <sup>9</sup>	A <sup>10</sup>
1	0	0	1	1	1	0	1	0	0	1

which reads out the second row (1 0 0) of the score numeral. This continues until all eight rows of the numeral have been read out and, since the electron beam stacks all lines directly on top of one another, a fully developed numeral results.

In case you are wondering, both the width of each bit and the position of the entire number is controlled by the video summation circuit which we will get to in a while. This circuit is necessary because the ROM information appears in a parallel format, so a shift register must be employed to make this information acceptable for a serial display.

Before moving on to the other read modes, we would like to point out the efficiency of the score data page for it is characteristic of optimum use of available memory area. In this game, there are 64 different scores possible and since each score occupies an 8-bit by 8-bit matrix — or 64 bits per numeral — a total of 4096 bits are needed to store all the numbers from 0 to 63. The page itself holds 512 8-bit words for a total of 4096 bits also, so the full set of scores exactly fills the entire page.

POSITION DATA: The ROM is addressed for this information when the electron beam is in the small square area in the upper left corner of Figure 7.10-3. So the new motion code can be loaded

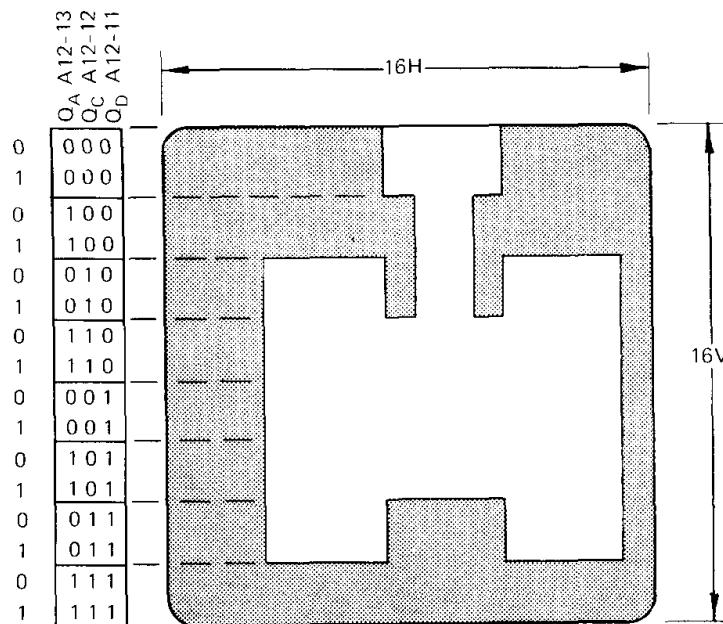
before the electron beam begins the new frame. During this time, the position data page is accessed and the first four address bits read out the contents of that page. These address bits are counter outputs which represent the current rotational aspect of the tank images, of which there are 32. Therefore, each rotational aspect reads out a different code which is utilized in the motion circuitry to generate realistic motion for the moving tanks. If the tank is moving and being turned simultaneously, these changing codes cause the image to travel in a curve which appears smooth and realistic.

**PLAYFIELD DATA:** The information necessary to generate the playfield border and the white "tank traps" inside is read from the ROM while the electron beam is scanning the large area in Figure 7.10-3 where both HINFO and VINFO are HI. Two completely different playfields are stored in the ROM and are selected by moving the jumper wire connected to D9-6. If D9-6 is

grounded LO, data page 0 0 will be accessed. The alternate playfield can be retrieved by tying D9-6 HI which will access page 0 1. Due to the construction of the address multiplex system, adjusting D9-6 for the playfields also affects the address for the position data, hence the position information must be duplicated in both areas.

Once the desired data block is chosen, the lesser order address at inputs A<sup>0</sup> through A<sup>8</sup> read the specific data needed for both the playfield and the mine locations. Horizontal inputs 64H, 128H 256H and 452H address the ROM for the horizontal data while 16V, 32V, 128V and 256V retrieve the vertical positions. Since 64H is the fastest signal, the address changes every 64 clock pulses and an 8-bit word is read out.

**TANK IMAGE DATA:** The ROM is addressed for this information in the thin vertical strip in the left of Figure 7.10-3 so the resulting data can be loaded prior to the scanning of each line.



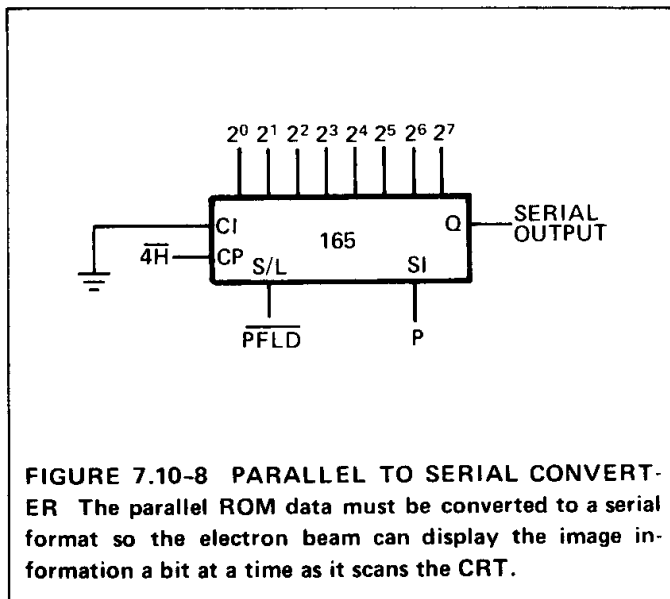
**FIGURE 7.10-7 A TANK ASPECT** The tank images are read from memory in much the same way as the score numerals. As the line count increments, all the data need-

ed to generate the line of the image appears simultaneously at the parallel ROM outputs.

A higher order address is formed by signals D0, D1, D2 and D3 which selects one of the 32 tank aspects in much the same way as the score count addressed the ROM for the proper numeral in the score mode. For example, when the rotational code at D0-D4 is 0 0 0 0, the tank aspect selected faces upward as in Figure 7.10-7.

Since the tank has to be capable of movement, motion counter outputs  $\emptyset 2$ ,  $\emptyset 3$  and  $\emptyset 4$  are used for the line address. However, notice that since the  $Q_B$  output of the counter has been used for the LSB of the line address, the same data is read out for two raster lines which "stretches" the image vertically in much the same way each bit of the score numeral is stretched to a total of four lines.

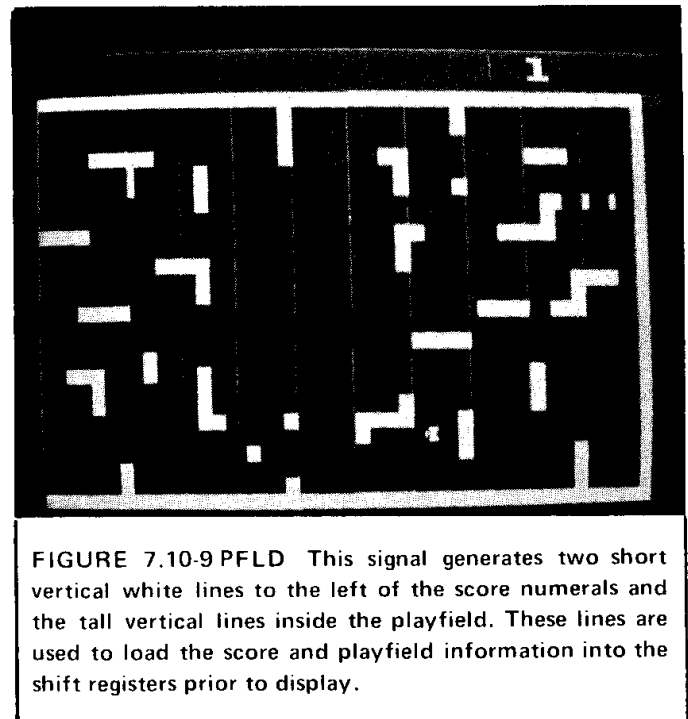
So as the count at  $\emptyset 2$ ,  $\emptyset 3$  and  $\emptyset 4$  increments all the way from 0 0 0 to 1 1 1, each pair of lines in the tank window enters a new address into the ROM and it outputs an 8-bit word containing the pattern of HIs and LOs needed to generate that section of the image. Since the time differential between when this motion code count starts can be changed with respect to sync, the direction of the image read out can also be controlled.



As we have mentioned before, provisions need to be made to convert the parallel ROM data into a serial data stream so the electron beam can illumin-

ate successive points one at a time. This conversion is easily done by a shift register and a typical configuration appears in the figure below.

The parallel score and playfield data ( $2^0$  through  $2^7$ ) is loaded into the shift register when the load signal  $\overline{PFLD}$  pulses LO. The data is shifted out serially by 4H after  $\overline{PFLD}$  returns HI. This signal is developed by another circuit and the output is illustrated in Figure 7.10-9.  $\overline{PFLD}$  contains two short vertical lines to the left of each player's score and a number of taller lines inside the playfield.



Since the shift register is clocked by 4H, the minimum width of any object is 4 clock pulses, since each data bit will appear for this amount of time and, hopefully, this and the inclusion of  $\overline{PFLD}$  should explain a few missing details. For one thing, the locations of the score load signals positions the score numbers in their respective areas. Also, if you will remember, each bit of the score numeral was stretched to 4 lines vertically. By clocking the shift register with 4H, each bit is stretched a corresponding amount horizontally.

8

**score circuitry**

## 8.1 Introduction

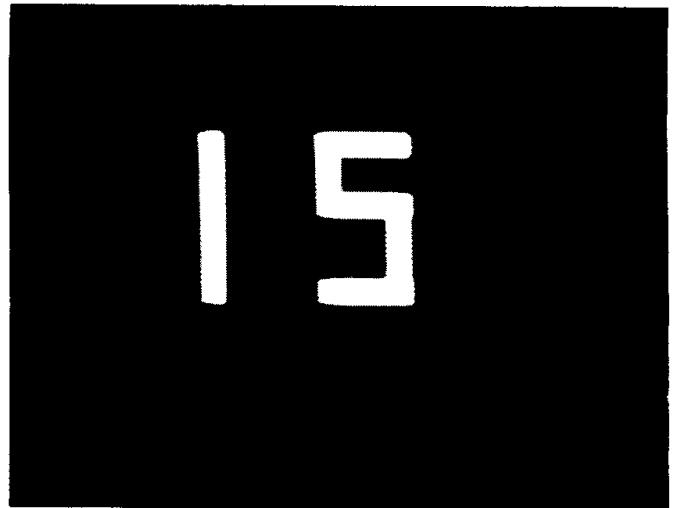
Although score circuitry is one of those areas which has seen a great deal of evolution since the introduction of video games, contemporary games show much similarity of function and even a certain amount of actual design similarity, particularly with regard to score storage circuits.

Any score circuit has two primary functions: (1) It must be able to count the player's score-producing events and make this information available to other circuits so the proper score numerals can be displayed and (2) the circuit must be able to generate the actual numbers on the CRT. The first function is performed by a sub-circuit usually known as *score storage*, however the actual generation of the images is produced by another section, generally named *score display*.

By and large, score storage circuits are very similar because the function rarely changes. The typical configuration is a set of counters which are asynchronously clocked by the score producing events and thereby keep a running tally of the player's score. These counters output a binary number representing the player's score and this code is used mainly by the score display circuit to generate the proper numerals. However, depending on the theme of the machine, the score storage outputs may be used in other places as well. For example, games in which the players are given a certain number of points to play (i.e. most paddle games) rather than time units use the score storage outputs to generate an end game signal when one of the players has won the predetermined number of points.

Score display circuits show much less similarity from game to game, not because this circuit is required to perform different functions, but because the general architecture of games has evolved from designs which use gated display techniques into image generation schemes where the actual information is held within ROM. In the earlier video game designs, score display circuits were invariably constructed using a 7-segment generator

capable of displaying all digits from 0 to 9 merely by turning the proper segments on and off. At first, this version was quite bulky until designers found ways of replacing a bunch of cumbersome gates with a much more streamlined multiplexed 7-segment display. Although this variety of score circuit eventually came to be quite efficient, it nevertheless still produced a set of numerals which add a very crude appearance.

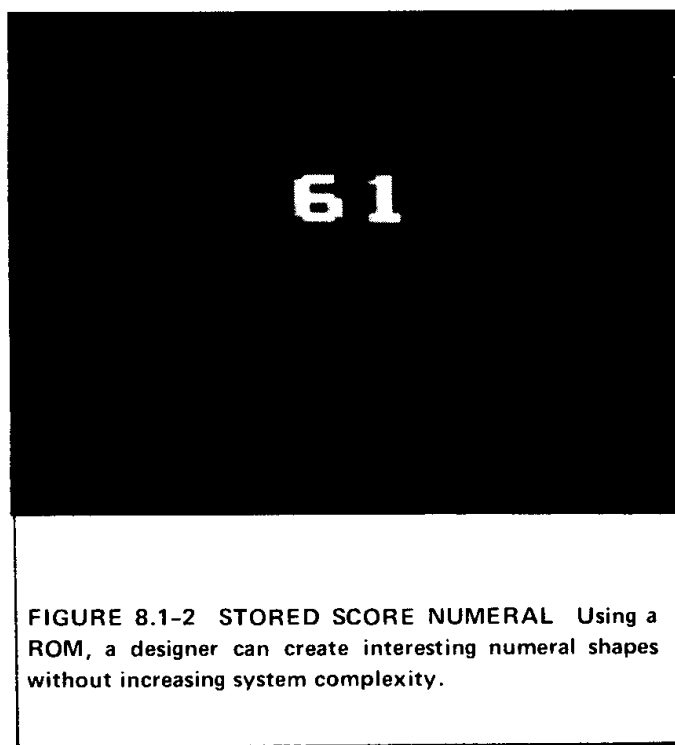


**FIGURE 8.1-1 7-SEGMENT SCORE NUMERAL** This is an example of a generated score display where different numerals are created by turning on the various segments of a 7-segment display. This type of display is attractive to designers because it does not require programming a ROM.

But as the paddle game craze began to die, manufacturers realized they would need to design games which produced more interesting effects to entice players bored with the same old paddle action packaged in different cabinets. Unfortunately, the more complicated images required for the new generation of games could not be conveniently generated the same way the simple square ball was generated in paddle games. This factor necessitated the use of memories which could conveniently store complex shapes containing considerable detail. So designers began to switch over to game architectures which used ROMs and it immediately became obvious that efficiency could be increased

by storing the score numerals in the ROM along with all the other images. Since the ROM address circuitry and the other provisions the use of a ROM necessitates had to be included anyway, storing score numerals meant only that a slightly larger ROM had to be programmed. And this is no big deal because even the most complex of score numeral shapes require no more than 64 bits of memory (an 8 by 8 matrix), meaning the entire set of numerals from 0 to 9 requires far less than 1K of memory.

But the ease with which score numerals are stored within memory is only part of the reason so many designers are currently using this technique. Because of the fact that the ROM essentially presents the designer with a blank matrix to work with, he can sit down with a piece of graph paper and simply plot out the shapes he desires with the result that more pleasing numeral shapes can be quickly programmed into the ROM.



In Figures 8.1-1 and 2, you can appreciate the aesthetic difference between *generated* score numerals (Figure 8.1-1) and *stored* displays (Figure 8.1-2). The ROM is able to produce a more pleas-

ing shape because a greater amount of information can be conveniently stored within it.

## 8.2 A 7-Segment Score Display

The following example incorporates the storage and display functions into a single circuit which we have tried to make as simple as possible so it will be easy to understand. The circuit is divisible into three sub-sections for convenience of analysis: (1) the *score storage* section which consists of the two 7490 counters, (2) the *score window* generator built from the two flip-flops and associated gating and (3) the actual *display* circuitry which contains three multiplexers and a 7-segment decoder.

Constructing a seven segment type display boils down to generating a series of progressively smaller windows all nested within each other where the largest positions the full number and successive divisions yield *digit windows* separating one numeral from the next and seven *segment windows* within each digit window. Displaying a particular number then becomes a relatively simple matter of turning on the proper set of segment windows. In the following circuit analysis, the score window circuit positions the whole number and the storage circuit tells the display section which set of segments to light up.

## 8.3 Score Storage

Most score circuits must be able to count both ones and tens digits which necessitates a counter for each place. In the example circuit below, Counter 1 keeps track of the ones digits and Counter 2 tallies up the tens place. Both counters are asynchronous 7490 divide-by-ten devices which are the BCD counterparts of the familiar 7493s.

At the beginning of a new game, both counters are reset by the SCORE RESET signal which is generally produced in the start section of the credit circuitry. This reset pulse sets the counters back to zero so they are ready to accept the new information. Each time a score-producing event



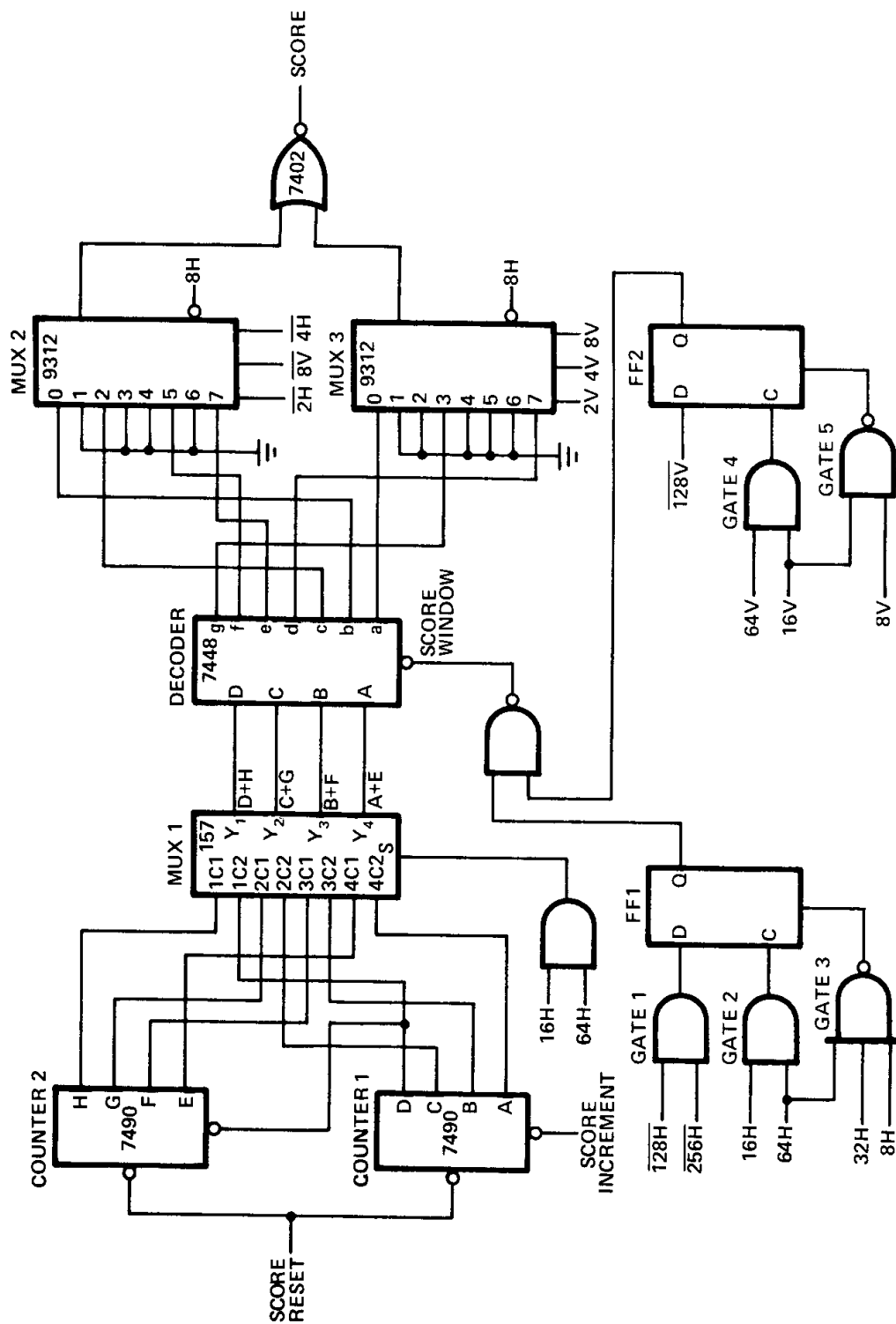


FIGURE 8.3-1 A TYPICAL 7-SEGMENT SCORE CIRCUIT. This circuit operates by using counters to keep track of the player's score and by generating sets of progressively smaller windows. The smallest set contains the seven segment-windows which are turned on and off by the counter outputs.

occurs on the playfield (i.e. the opposing player misses the ball), SCORE INCREMENT pulses LO and increments the ones counter, which can count a total of ten pulses before it resets itself back to zero again.

But at the start of the game and before the first increment pulse, the A, B, C and D outputs of the counter read 0 0 0 0 because the counter has just been reset. At the first increment pulse, the output code reads 1 0 0 0 or decimal 1.

On the second increment, the code reads 0 1 0 0 or decimal 2 and so on until the ninth count at which time the outputs reflect 1 0 0 1. On the next – or tenth – count, the  $Q_D$  output drops LO along with all the other outputs and, since the D output is connected to the clock input of the tens counter, Counter 2 is incremented by the LO pulse. On the next – or eleventh – count, the tens counter outputs remain at 1 0 0 0, but the ones counter reflect 1 0 0 0 where the addition of these two numbers equals decimal 11.

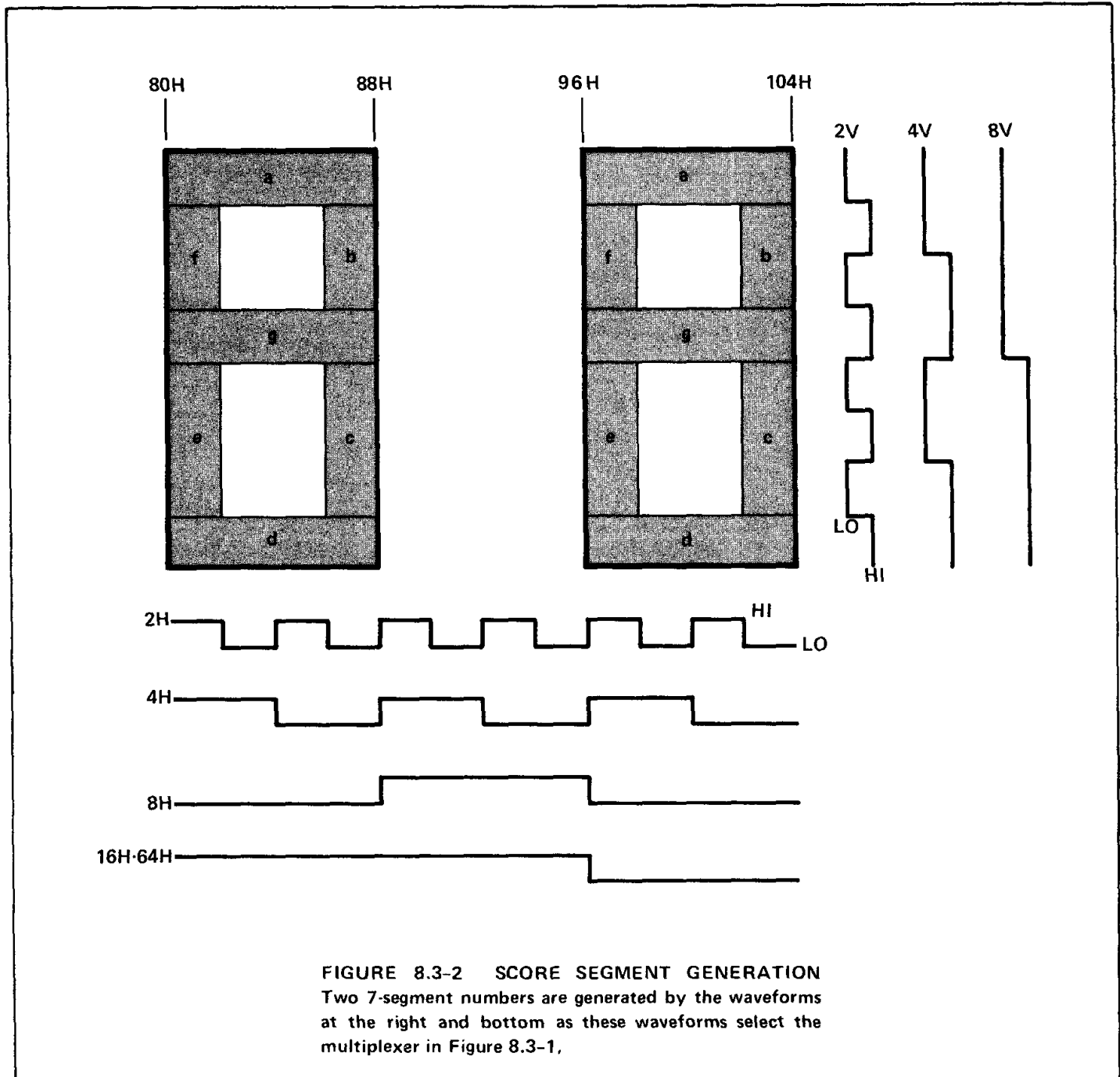


FIGURE 8.3-2 SCORE SEGMENT GENERATION  
Two 7-segment numbers are generated by the waveforms at the right and bottom as these waveforms select the multiplexer in Figure 8.3-1,

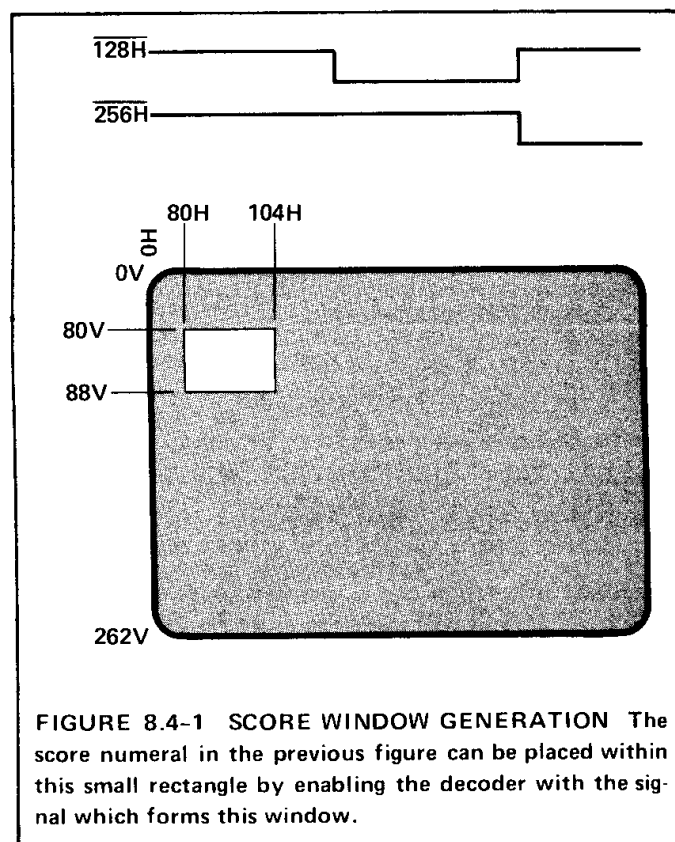
The outputs from both counters become the inputs to Multiplexer 1, the sole function of which is to select the ones or tens number for display. Consequently, Mux 1 is selected by the signal 16H-64H which distinguishes the ones numbers from the tens numbers. If you examine the waveform in Figure 8.3-2 labelled 16H-64H, you can see it is HI during the time the ones digit occurs and that it drops LO at the beginning of the tens digit. When the electron beam is scanning the ones place, this signal is HI and outputs A, B, C and D from the ones counter appear at the multiplexer outputs. During the tens place, 16H-64H is LO, so the E, F, G and H outputs from Counter 2 appear at the multiplexer outputs. These outputs then become the inputs to the 7-segment decoder which takes the binary score number and converts it to a 7-segment format so the proper segment of the display can be illuminated (see Section 1.36 for more details on the operation of the 7448). But before getting into any more detail about how the actual numbers are displayed, let's discuss the function and operation of the score window sub-circuit.

#### 8.4 Score Window

By enabling the decoder with the output from the score window circuit, we can position the numerals anywhere on the CRT. Let's say we want the numerals to appear in the upper left corner of the CRT (Figure 8.4-1). So, we construct a circuit which produces a SCORE WINDOW signal that's LO (because the decoder enable is active LO) from 80H to 104H horizontally and from 80V to 88V vertically. This signal fully describes the rectangle in Figure 8.4-1.

In the score window portion of the schematic, Flip-flop 1 produces the horizontal boundaries and the window is limited vertically by Flip-flop 2. The addition of 16H and 64H at Gate 2 clocks a HI from the first flip-flop at 80H producing the left boundary. Signals  $\overline{128H}$  and  $\overline{256H}$  are entered at Gate 1 so the horizontal boundaries of the window can only occur where these signals are both HI or, in other words, in the leftmost quarter of the CRT. This HI is cleared 24H later by the out-

put of Gate 3 when it rises HI at 104H and creates the right boundary since the window drops LO at this time.



The vertical boundaries are formed similarly by using vertical submultiples. In this case, the vertical dimensions can only appear in the top half of the CRT or when 128V is LO and the actual boundaries are produced by the addition of 64V and 16V (80V) at Gate 4 and the addition of 80V and 8V (88V) at Gate 5. The resulting SCORE WINDOW signal fully defines the limits of the rectangular score window area.

#### 8.5 Score Display

The actual segments of the score display are created by multiplexers 2 and 3 which generate the horizontal and vertical coordinates of each segment window.

Notice that Mux 3 in the schematic is selected by 2V, 4V and 8V. Now, compare these selects with the representation of their waveforms to the side

of the 7-segment numbers in Figure 8.3-2. You can see that — in order to generate the “a” segment — 2V, 4V and 8V are all LO so Mux 3 is enabled at this time. The combination of the LOs at all three signals defines the vertical boundaries of a thin stripe across the CRT superimposed exactly on top of the “a” segment. The states of the selects 2H, 8V and 4H at Mux 2 determine the horizontal boundaries of this stripe and fully define the “a” segment window.

To display the numeral 8, all the segments — including the “a” segment — need to be turned on. When the player has reached this score, the counter outputs 0 0 0 1 to the decoder which enables HIs from all the segment outputs. If the “a” decoder output is HI, the electron beam is intensified during the “a” window produced by the multiplexers and that segment appears white on the CRT.

In order to space the two numerals a distance of 8H apart, Mux 2 and Mux 3 are enabled by 8H. While 8H is LO, the multiplexers are enabled and can generate a number. But after this number is finished, 8H returns HI, disabling the multiplexers so nothing can happen for an equal amount of time until 8H returns LO again.

# 9

## **paddle generation and control**

## 9.1 Introduction

By virtue of the fact that the first successful video game was a paddle game, an entire generation of paddle games was spawned. The first of these were mere copies which even incorporated mistakes found in the original. As the industry grew, manufacturers brought out a number of relatively imaginative paddle game variations where teams of players could compete on soccer or hockey playfields, but these really boiled down to just adding more of the same kind of paddles. Then came *multi-directional* paddles which enabled the player to move anywhere within his playing area and finally paddle circuits were built with all the above features and with the added attraction of *size selector* buttons.

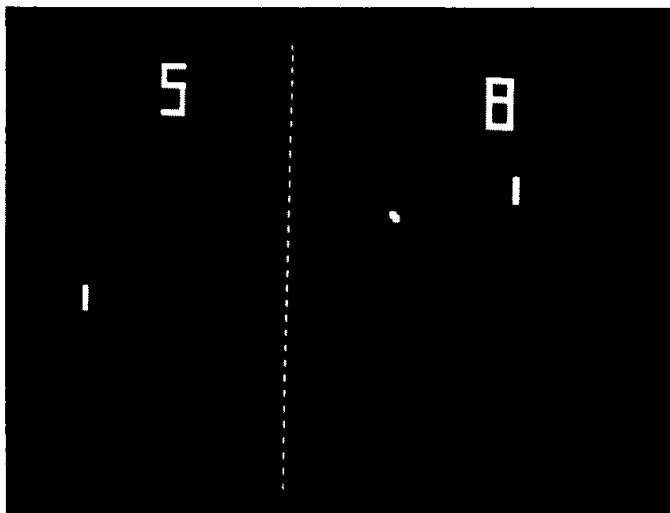


FIGURE 9.1-1 EARLY PADDLE GAME The incredible success of this paddle game spawned an entire generation of imitations.

In spite of all these modifications, a single type of paddle circuit remains the standard against which almost all others are designed. In this type of circuit, the horizontal position of the paddle is fixed in one place and it travels smoothly in the vertical plane as the player rotates his control. The overall design of this circuit essentially revolves around the type of player control used. Since the desired effect was to have the paddle move smoothly as

the player turned a knob, some means had to be devised to interface the player's potentiometer to the digital circuitry. This was handily solved by using a *linear circuit*, the 555 timer.

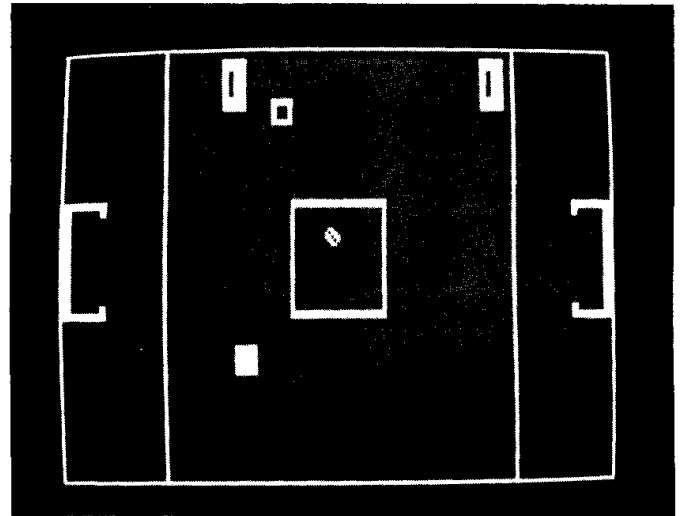


FIGURE 9.1-2 ADVANCED PADDLE GAME An imaginative designer gives new life to an old theme. Not only are these paddles capable of multi-directional movement, but they also have built-in "inertia" which makes this game quite challenging.

## 9.2 The 555 Timer

The 555 timer illustrated in Figure 9.2-1 is connected in the *monostable mode*, meaning it outputs a single pulse when the enabling event occurs. In the *astable mode*, the timer can be used to generate a series of timed pulses, accurate enough to be used for a system clock in some cases.

Mode selection is determined by the way the control pins are connected and pulse timing is controlled by an externally mounted RC network.

The timer has two internal voltage comparators which compare the threshold (pin 6) and the trigger (pin 2) inputs. As the values of these voltages change, the comparators set and reset an internal flip-flop which charges and discharges the externally mounted capacitor. The resulting signal is internally amplified to a level high enough to drive

a number of TTL loads.

The trigger input signal 256V is used to limit the lower end of paddle travel. Since 256V is HI after the 256th line, the paddle image is prevented from being generated after this point. This has created an interesting problem because the raster actually extends to 262V. This means that the *ball* can reach the 262nd line, but the player can only move his *paddle* as far as the 256th line, thereby creating a hole or *blind spot* through which the ball may freely pass.

The timer's threshold level is controlled by an adjustable 50K trim pot which is placed in this line to allow correct positioning of the top end of paddle travel.

The player's control consists of a 5K pot mounted to the external control panel of the game and it is connected to the VCO input of the timer through a 470 Ohm resistor. The potential at the VCO input determines the amount of time required to charge capacitor C1 and varying this charge time causes the paddle image to be generated sooner or later with respect to the raster timing. The result is a paddle image which is displayed further up or down the CRT as the player rotates his control.

If you look closely at the photograph (Figure 9.2-2) of an actual paddle image, you can see it consists of 16 short horizontal lines produced by the 7493 counter in Figure 9.2-1. The timer output (pin 3) is connected to both reset inputs of

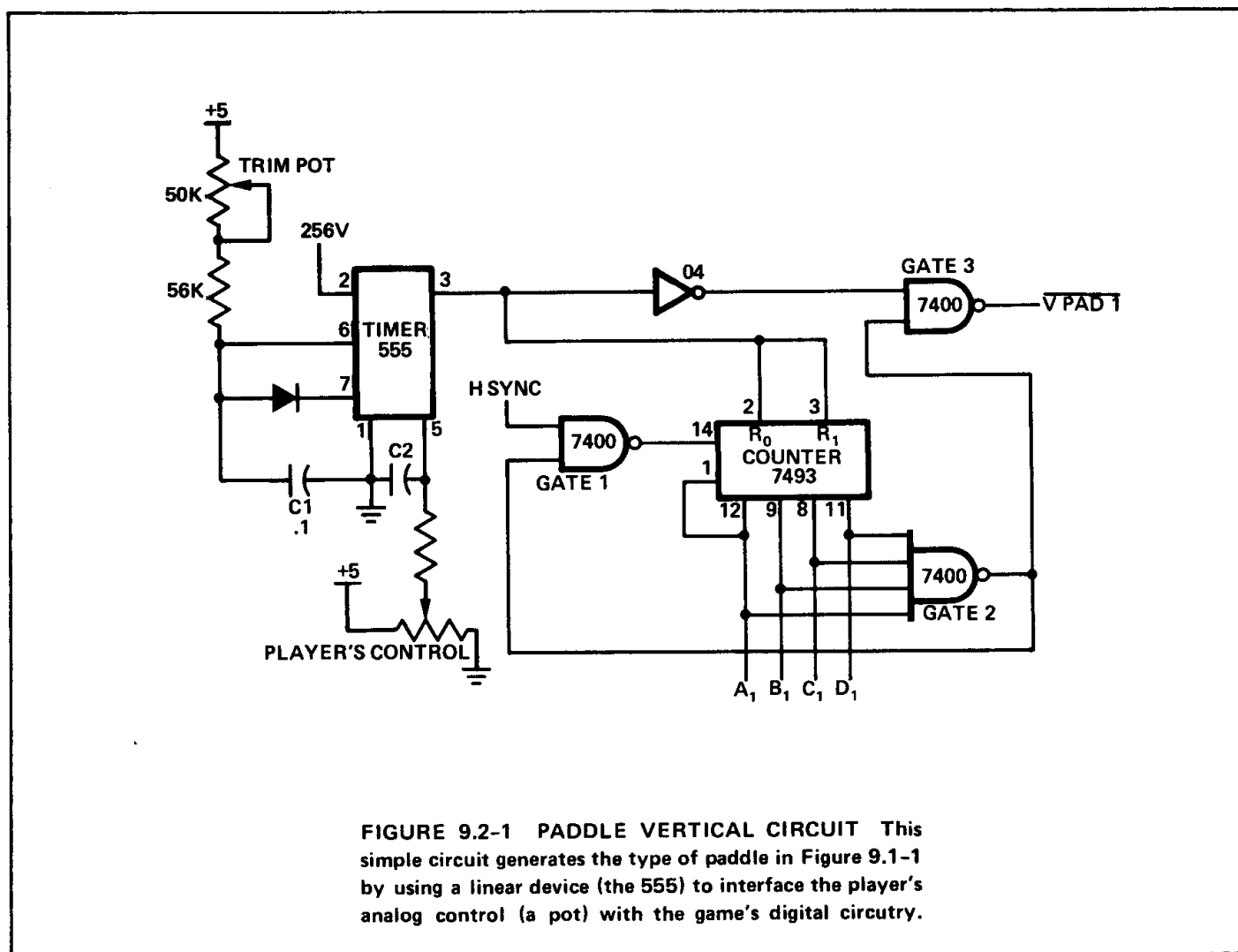


FIGURE 9.2-1 PADDLE VERTICAL CIRCUIT This simple circuit generates the type of paddle in Figure 9.1-1 by using a linear device (the 555) to interface the player's analog control (a pot) with the game's digital circuitry.

the counter and, until C1 is adequately charged, the HI from the timer keeps the counter in the reset condition so that all its outputs are forced LO. However, when the cap becomes adequately charged, the timer outputs a LO pulse which removes the reset condition from the counter and since all its outputs are LO, Gate 2 produces a HI which is connected back to Gate 1 and enables H SYNC pulses through to the clock input of the counter. Since this is a 4-bit counter, it counts 16 H SYNC pulses thereby limiting the length of the paddle to 16 segments. When the counter has reached a full count of 1 1 1 1 at its outputs, the output of Gate 2 drops LO and shuts off any further H SYNC pulses at Gate 1. An inverter has been placed in the line between the timer and Gate 3, so the LO pulse from the timer is inverted and enables the paddle segment information through this gate.

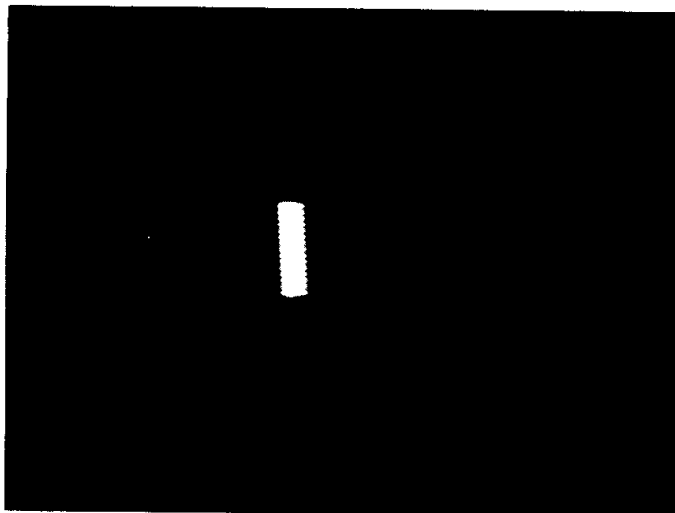


FIGURE 9.2-2 ACTUAL PADDLE DISPLAY This enlarged photograph of an actual paddle image clearly shows the sixteen short horizontal segments used to generate the paddle image.

Figure 9.2-3 compares the paddle segments with their binary equivalents as produced at the counter. Since the angle of the ball must be changed according to which paddle segment it contacts, this count is used in the motion circuitry to control vertical ball velocity. However, to simplify matters, the sixteen segments are grouped in eight

pairs of two simply by dropping the LSB of the count (the A<sub>1</sub> output). If you cover up the A<sub>1</sub> column, you can see that the first two segments now have the same number (0 0 0). So, if the ball contacts *either* of these segments, it will be given the same code.

		A <sub>1</sub>	B <sub>1</sub>	C <sub>1</sub>	D <sub>1</sub>
█	0	0	0	0	0
█	1	1	0	0	0
█	2	0	1	0	0
█	3	1	1	0	0
█	4	0	0	1	0
█	5	1	0	1	0
█	6	0	1	1	0
█	7	1	1	1	0
█	8	0	0	0	1
█	9	1	0	0	1
█	10	0	1	0	1
█	11	1	1	0	1
█	12	0	0	1	1
█	13	1	0	1	1
█	14	0	1	1	1
█	15	1	1	1	1

FIGURE 9.2-3 BINARY PADDLE SEGMENT CODES Each segment of the paddle image is assigned a binary code which is used in the game's motion circuitry to develop the ball motion code.

### 9.3 Player Selectable Paddle Sizes

This effect is very easy to achieve because by placing on-off switches in the A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub> and D<sub>1</sub> lines to Gate 2, any paddle size from 0 to 16 segments may be selected. For example, let's say we are expert players and desire a smaller paddle of 4 segments to increase the difficulty of the game. All we need to do in this case is to disconnect the C<sub>1</sub> and D<sub>1</sub> inputs to Gate 3 by opening their switches which leaves only the four counts from 0 0 (0) to 1 1 (3).

### 9.4 Paddle Size and Summing

Had we taken signal  $\overline{VPAD\ 1}$  and injected it into the video line, the resulting paddle segments would extend all the way across the CRT because they have not yet been limited in the horizontal dimen-



sion. So a separate circuit is needed to limit paddle width as well as positioning the first and second player's paddles in their proper places.

First of all, notice that all the inputs to Gates 2 and 3 must be LO before these gates will be enabled. The paddle signal itself is always LO when the paddle is being generated, so we do not have to worry about it. But 256H and the output from Gate 1 change in such a way as to limit the width of the paddles and position them correctly.

The game begins when the players deposit their coin which causes  $\overline{\text{ATTRACT}}$  to drop LO and enable the flip-flop. Since 128H is connected to one input of Gate 1 and to the D input of the flip-flop, there will be two HIs at this gate after 128H rises HI. But since the flip-flop is clocked by 4H, the Q output will drop LO 4 clock pulses later, limiting the LO output of Gate 1 to pulse 4H wide occurring after the rising edge of 128H.

The LO from Gate 1 limits the paddle to the proper width, however both paddles still need to be positioned by signals 256H and  $\overline{256H}$  at Gates 2 and 3. During the time the electron beam is scanning the left side of the CRT, 256H is LO which enables a 4 clock pulse wide chunk of  $\overline{\text{VPAD 1}}$

through Gate 2. Since 128H rises HI in the middle of the first player's court, his paddle appears for a 4 clock pulse wide time in the center of the left hand side. Simultaneously,  $\overline{256H}$  at Gate 3 is HI, so the second player's paddle is prevented from being displayed in the first player's side.

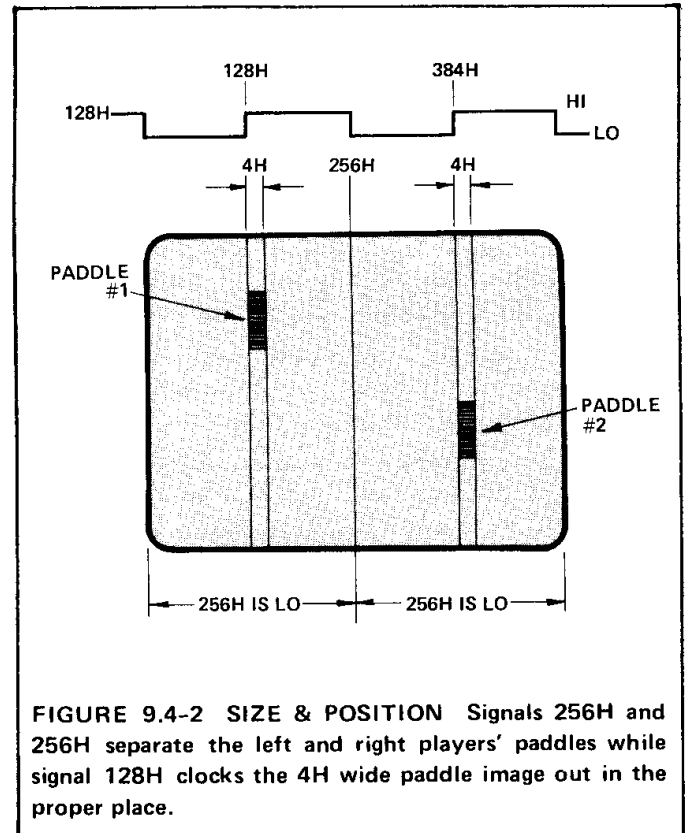


FIGURE 9.4-2 SIZE & POSITION Signals 256H and  $\overline{256H}$  separate the left and right players' paddles while signal 128H clocks the 4H wide paddle image out in the proper place.

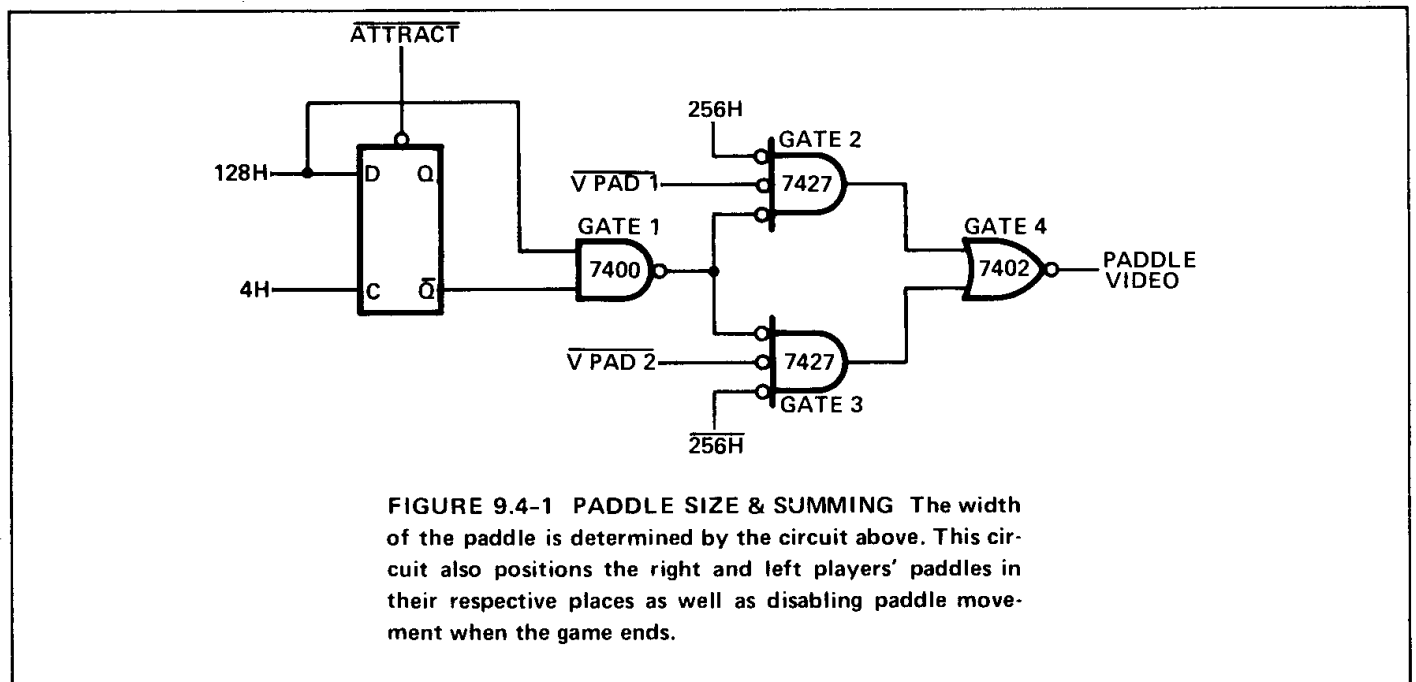


FIGURE 9.4-1 PADDLE SIZE & SUMMING The width of the paddle is determined by the circuit above. This circuit also positions the right and left players' paddles in their respective places as well as disabling paddle movement when the game ends.

## SUMMARY

In the preceding chapters we have seen many examples of digital video technology. Most chapters have contrasted a simple technology and circuitry to the more complex which blends Phase I and Phase II Video Games. What is the major difference anyway? If you have studied the subject matter of this text, your conclusion might be obscure in that most Phase II games employ circuitry utilized in Phase I types. Here is one example of the overlap in circuitry. We have seen 2 or 3 types of sync. generation; two of raster scan and one of interlace scan. But had you noticed the similarity between unloading a diode matrix and a memory. Casual observation demonstrates the need for support electronics when preprogrammed memory is used such as a ROM. However when creating objects from the discrete arrays of circuitry we only need to gate our signals when they are to be called out and used. There is quite a difference between creating a 4H x 4V ball with 11 chips and a car or ship with 4 chips. Anyway, the significant difference is the type of display generated. Typically, we say the Phase I games are the Paddle types and the Phase II are the Display types. Because we see tremendous overlap of circuitry, we have put together this textbook, the first of a series of general education books which directly apply themselves to our occupation, keeping the DAMN things running. You will find more in-depth material on each of the general topics, but nobody applies the information like KUSH N' STUFF AMUSEMENT ELECTRONICS. We are the answer to the industry's needs for well-documented information.

For those of you with specific game problems and the desire to service your own equipment, the following publications are available:

### Video Game Data Books

Vol. I	Pin Pong	Atari Inc.
Vol. II	Wheels I & II	Midway Manfg. Co.
Vol. III	Tank	Kee Games
Vol. IV	Gunfight	Midway Manfg. Co.
Vol. V	Seawolf	Midway Manfg. Co.
Vol. VI	8080 CPU	Midway Manfg. Co.
Vol. VII	Ramtek/Volly	6 game issue
Vol. VIII	Exidy/Chicago Coin	(unreleased)

### Video Game Logic Textbooks

Vol. I	Phase I	Video Game Logic
Vol. II	Phase II	Video Game Logic (unreleased)
Vol. III	Phase III	Video Game Logic (unreleased)

### Video Game Supplements

Vol. IIIB	Tankers/Biplane	Fun Games
Vol. A	Seminar Book	Kurz-Kasch
Vol. B	Video Game Logic Reference Book	Kush N' Stuff