TABLE OF CONTENTS

TABLE OF CONTENTS

(continued)

TABLE OF CONTENTS

(continued)

INTRODUCTION

The Atari Video Interface Circuit (VIC) is controlled by a Microprocessor (MPU) through a series of instructions contained in a computer program. The program is stored in a Read Only Memory (ROM) and is executed according to a set of predefined algorithms and flowcharts developed by the programmer. The MPU also requires some Random Access Memory (RAM) in order to execute the program and to store temporary results of the various routines executed. Also required is a Peripheral Interface Circuit (PIC) to communicate with the external controls (i.e., switches, potentiometers, etc.).

The VIC is an object-oriented device in that it will directly generate an object which can be moved under the control of the MPU. The object is not generated through the use of screen RAM, but rather from an object generator to which we supply the object shape (font) as needed for the generation of the object on the video screen. The object has 8 Bits of Horizontal Font Data. The horizontal placement of the object is made by the VIC. However, we can move the object left or right from the position by controlling the Movement section of the VIC. Vertical positioning is under the direct control of the MPU. The Objects can be programmed to three different Horizontal sizes and can be repeated one or two times after the original Object.

The VIC generates a non-universal object called a Projectile. The Projectiles can be programmed to only 4 different Horizontal sizes. The Projectiles have all of the motion capabilities of the Objects. The Projectiles can also be programmed to repeat one or two times after the original Projectile.

The VIC also controls a non-universal object generator called the Border, which has all the motion capabilities of the Objects and Projectiles, but which is like the Projectiles and does not have the font capability of the Objects. The Border can only make 4 different shapes horizontally like the Projectiles versus 255 for an Object. The Border also cannot be repeated horizontally as the Objects and Projectiles can.

The Foreground appears as a type of screen RAM on a one-line basis, in that it cannot be moved horizontally and covers the entire horizontal visible area of the video screen.

The Sound Section works under the control of the MPU to generate complex sounds. The Sound section generates 10 different sound wave shapes that are used as building blocks for complex sounds. The MPU can control the frequency and amplitude for each of the two Sound channels.

The Vic has a Concidence detection circuit which checks for coincidence between each of the Objects, Projectiles, Border, and Foreground.

There is also a set of Input circuits that can be used as digital inputs and for performing an Analog to Digital conversion for use with the Player controls.

The VIC has a Color Generator circuit that generates all of the signals needed to produce a color picture on a standard NTSC Color Television.

FEATURES

* 2 General Purpose Objects
* 3 Dedicated Objects
* Object Duplication (2 Programmable Repeat Objects)
* Object Size and Movement Under Microprocessor Control
* Programmable Object Priority
* 280 nsec Object Resolution
* Programmable Foreground
* Programmable Foreground Repeat or Mirroring
* 128 Programmable Colors
* Programmable Vertical Sync and Vetical Blanking Timing
* 2 Programmable Sound Generators
* 4 Analog Potentiometer Inputs
* 2 Digital Inputs (Edge Sensitive Programmable)
* 4 Displayable Colors per Horizontal Scan Line
* 4 Chip Select Lines for Address Decoding
* 40 Pin Dual-In-Line Package
* Page Zero Microprocessor Operation

SYSTEM DESCRIPTION

The Atari Video Interface Circuit (VIC) is designed for use in a Microprocessor controlled Video Game System. It provides all the game circuits on a single N-Channel MOS LSI Integrated Circuit. Circuits are provided for both analog and digital player inputs, Foreground, moving Objects, Projectiles, Border, and audio signals.

The Microprocessor used in the system is the 6507. The ROMS used are the 2316 (16K) or 2332 (32K). The PIC is a 6532. The VIC is a custom circuit.

The game is Microprocessor controlled and each game definition is stored as a program in the ROM. The ROM contains the game rules, the score font, the object font, the background font or algorithm, and the sound algorithms. Each ROM can contain more than one game or variations of a game depending on game complexity.

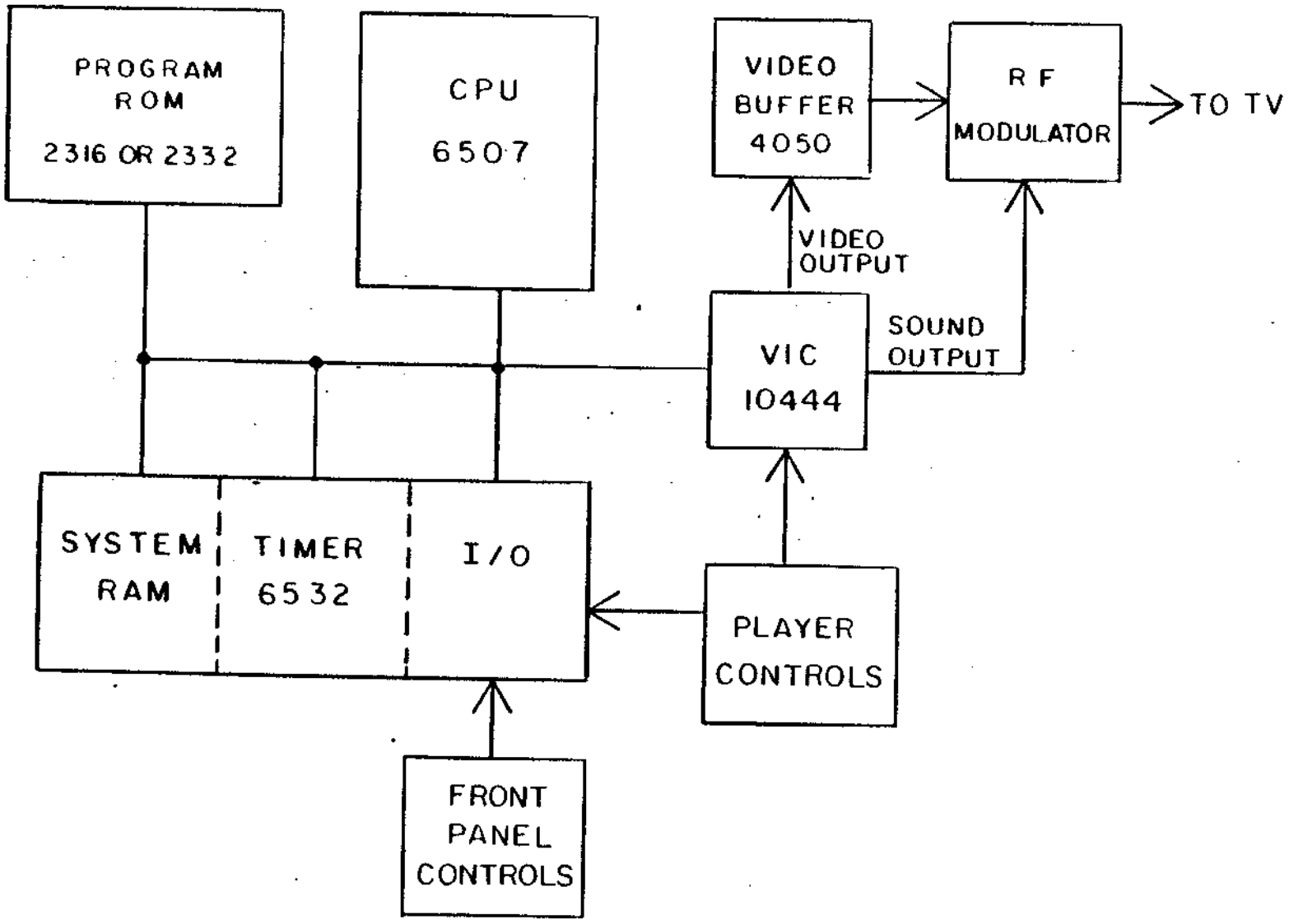
The VIC develops Composite Video with Color Burst according to the NTSC video standard and uses a 3.579545 MHZ oscillator frequency. The display is non-interlaced.

A block diagram of the system is shown in Figure 1. The Microprocessor reads the stored program in the ROM and controls the generation of the video output by the VIC.

The VIC generates the Horizontal Sync, Horizontal Blanking, Color Burst, and Video Signals which contain the Color and Luminence signals. The Microprocessor keeps track of the number of horizontal lines scanned and controls the VIC to generate the Vertical Blanking and Sync. The VIC generates the sounds under control of the Microprocessor, which can control the frequency, shape, and amplitude.

The Microprocessor uses the PIC and VIC as the I/O interface for the player controllers (digital), player potentiometers (analog), and the front panel controls (digital). The system scratchpad RAM and Stack are in a 128 byte RAM in the PIC which also contains a programmable 8 bit timer.

The 128 byte RAM in the PIC is shared between the MPU stack and the system scratchpad RAM. The RAM is address transparent between $0080 thru $00FF and $0180 thru $01FF. That is to say that the same RAM appears in both of these address ranges simultaneously and care must be exercised not to let the stack alter the scratchpad RAM being used by the program.



ATARI VIDEO GAME BLOCK DIAGRAM

FIGURE 1

REV A 7-23-81

VIC DESCRIPTION

The VIC is a Bus oriented device. The Microprocessor address and Data Busses enter the VIC and access the major functional areas. With the Address Bus the Microprocessor selects the area it desires to communicate with. The information is presented or received from the selected area on the Data Bus.

An external oscillator provides the 3.58 MHZ clock frequency for the VIC. THe VIC then divides it by 3 to generate the clock for the Microprocessor.

The VIC internally generates the Horizontal Sync, Blanking, and Color Burst and generates 4 displayable colors per line (Object A/Projectile A, Object B/Projectile B, Border/Foreground, and Background). Each color is independently programmable, and each one has a priority over the other. Some of the priorities can be altered by the Microprocessor. There are 128 Programmable colors for each of the above.

There are two fully programmable Objects (Object A and Object B). Each of these Objects is 8 bits wide and is fully programmable which means that there are 255 visible combinations of display for each Object. There are two Object Font Registers for each Object. Each Object Font Register contains one Byte (8 bits) of data. The direction in which the data is scanned can be reversed. The horizontal size of the bits in the Object display

can be programmed to be 1, 2, or 4 units wide. Each of the Objects can be programmed to be repeated one or two times after the original and at different intervals from the original Object.

There are two dedicated objects (Projectile A and Projectile B) that are related to the main Objects (Object A and Object B). These Projectiles can only be 1, 2, 4, or 8 units wide. Projectile A can be programmed to track the horizontal movement of Object A. Projectile B can be programmed to track the horizontal movement of Object B. The Projectiles can be repeated horizontally in conjunction with the associated Object. Projectile A is repeated exactly as Object A and Projectile B is repeated exactly as Object B.

There is also a dedicated object (Border) that can be moved or used as a ball, Border, or center line. The Border can only be 1, 2, 4, or 8 units wide. The Border cannot be repeated.

The Foreground is a 20 bit memory that can be displayed in one of four methods horizontally. Each bit is 4 units wide and the active Foreground area is 160 units wide. The Foreground memory acts as a register. The register is 2-1/2 bytes of RAM (20 bits). The Foreground Register is displayed two times each line (40 bits/line).

An object can be made to appear over or under the Foreground and Border and can appear over or under another object based on a

priority system. Object A, Object B, Projectile A, and Projectile B can be programmed as higher or lower priority than the Foreground and Border.

Object A and Projectile A are always a higher priority than Object B and Projectile B.

The horizontal movement of each Object (A and B), Projectile (A and B), and the Border is controlled by the Microprocessor. Each of these 5 can be moved left or right relative to the Horizontal Reference Counter. Movement can be from -7 to +8 horizontal units per horizontal line sweep.

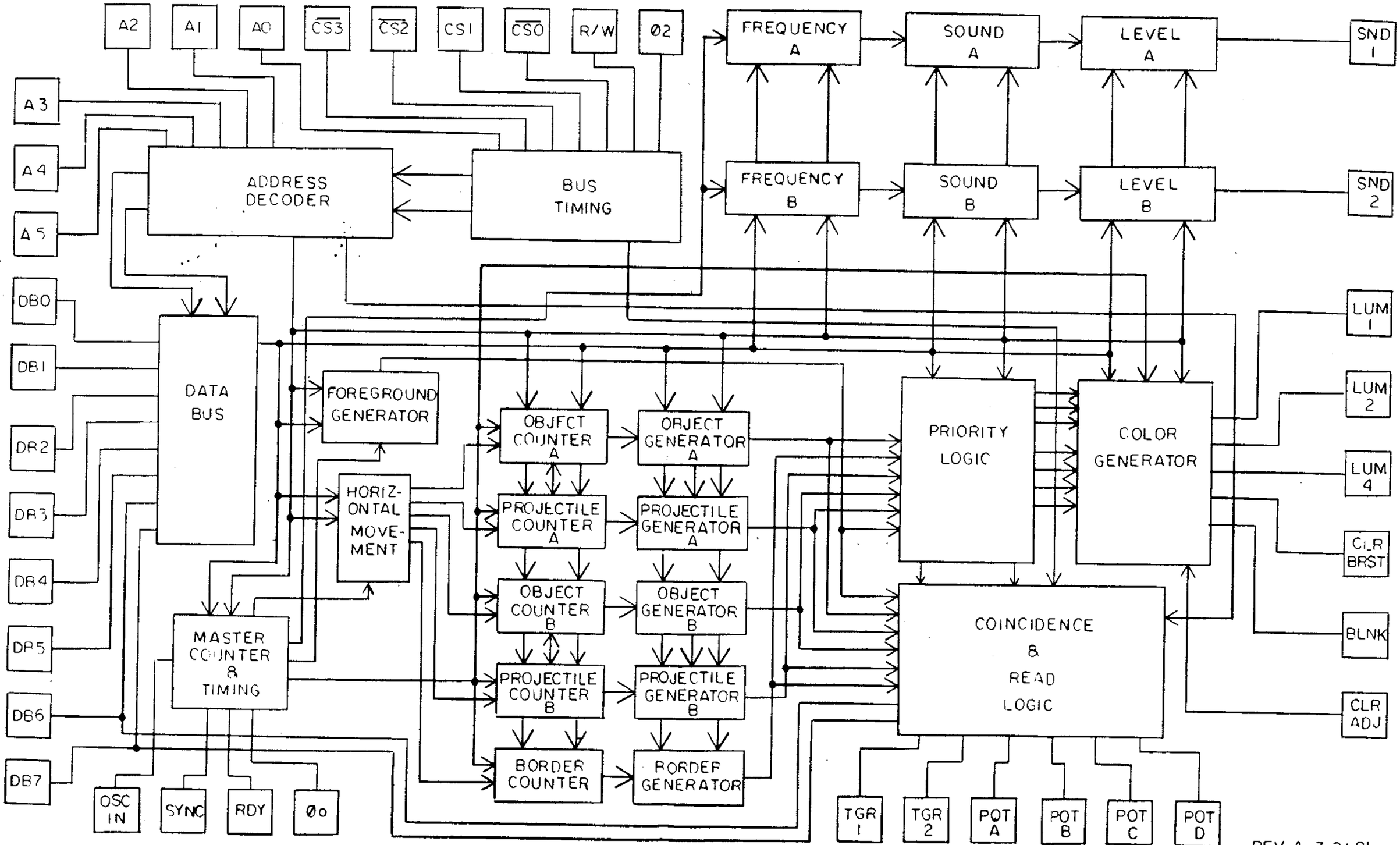
The VIC has a coincidence detection circuit that indicates when any two Objects, Projectiles, Foreground, Border, or any combination of the above are coincidental with each other. A set of 15 comparators compares each object against the other and stores the result in a set of registers which can be read and reset by the Microprocessor.

The VIC has 4 analog inputs with Schmitt triggers for accurate repeat detection of the player potentiometer setting. The analog inputs use a resistor-capacitor circuit for the time constant; the resistor is a potentiometer that is controlled by the player. Each input has a progammable discharge transistor that can be turned on to discharge the capacitor in the RC timing circuit.

There are also 2 Trigger inputs which can be used as latching Input Ports. Whenever one of these inputs goes to a "0" level, this transition is stored in a latch that can be reset under software control. The latching mode is programmable and when turned off the data at the Trigger inputs is passed directly to the MPU when read.

The VIC contains two Sound genertors each of the Sound generators are connected to one of the two Sound Output pins.

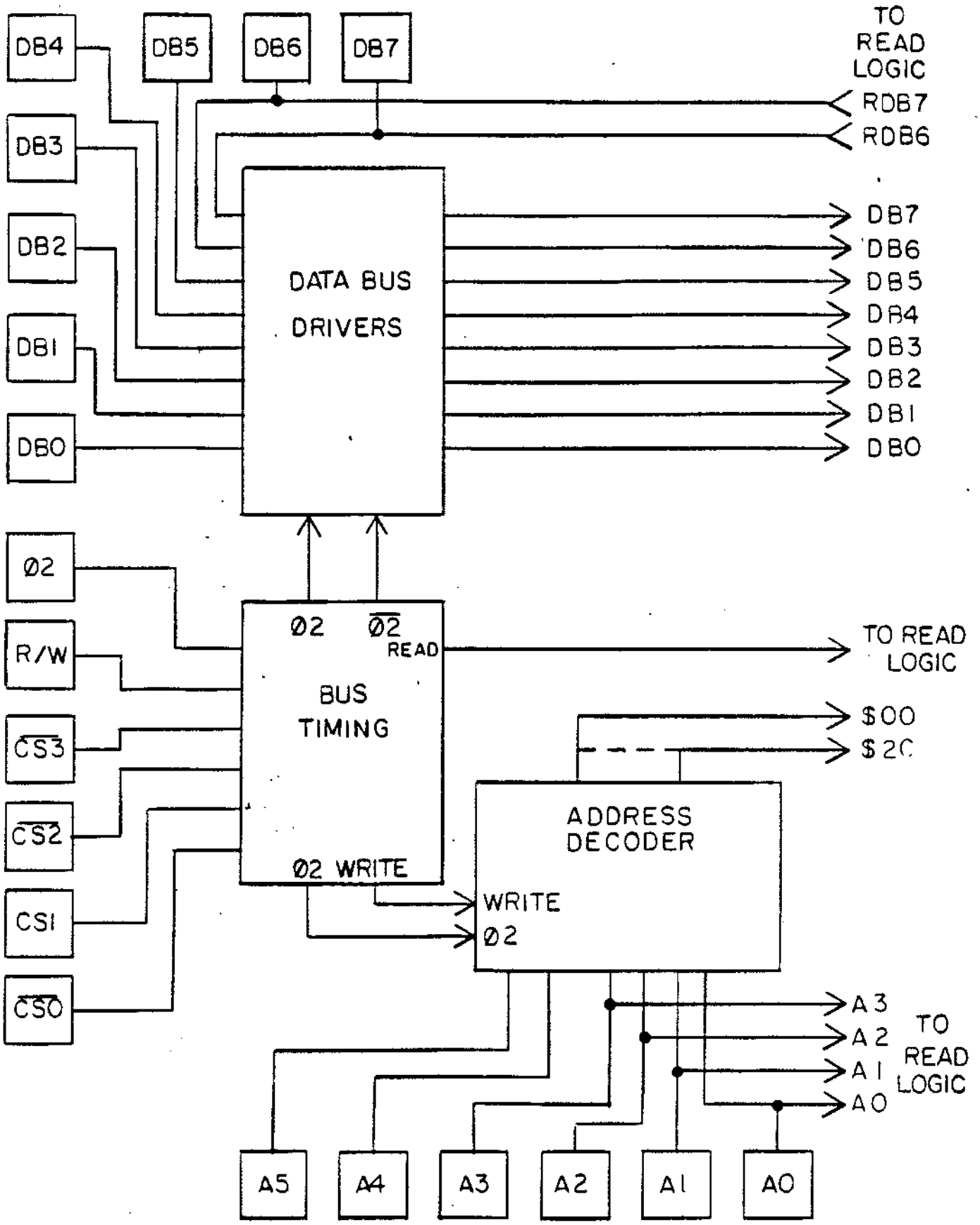
The Sound Generator consists of a Programmable Divider that divides the horizontal sweep frequency, a Sound Generator, and a programmable Output Driver. The Divider can be programmed to divide the 15.7 KHZ horizontal sweep frequency by 1 to 32. The Sound Circuit is programmable for 10 different sounds and tones. This circuit can produce a series of sounds from a simple tone to a complex random noise. The Output Driver can be programmed for 15 different output levels.



REV A 7-21-81

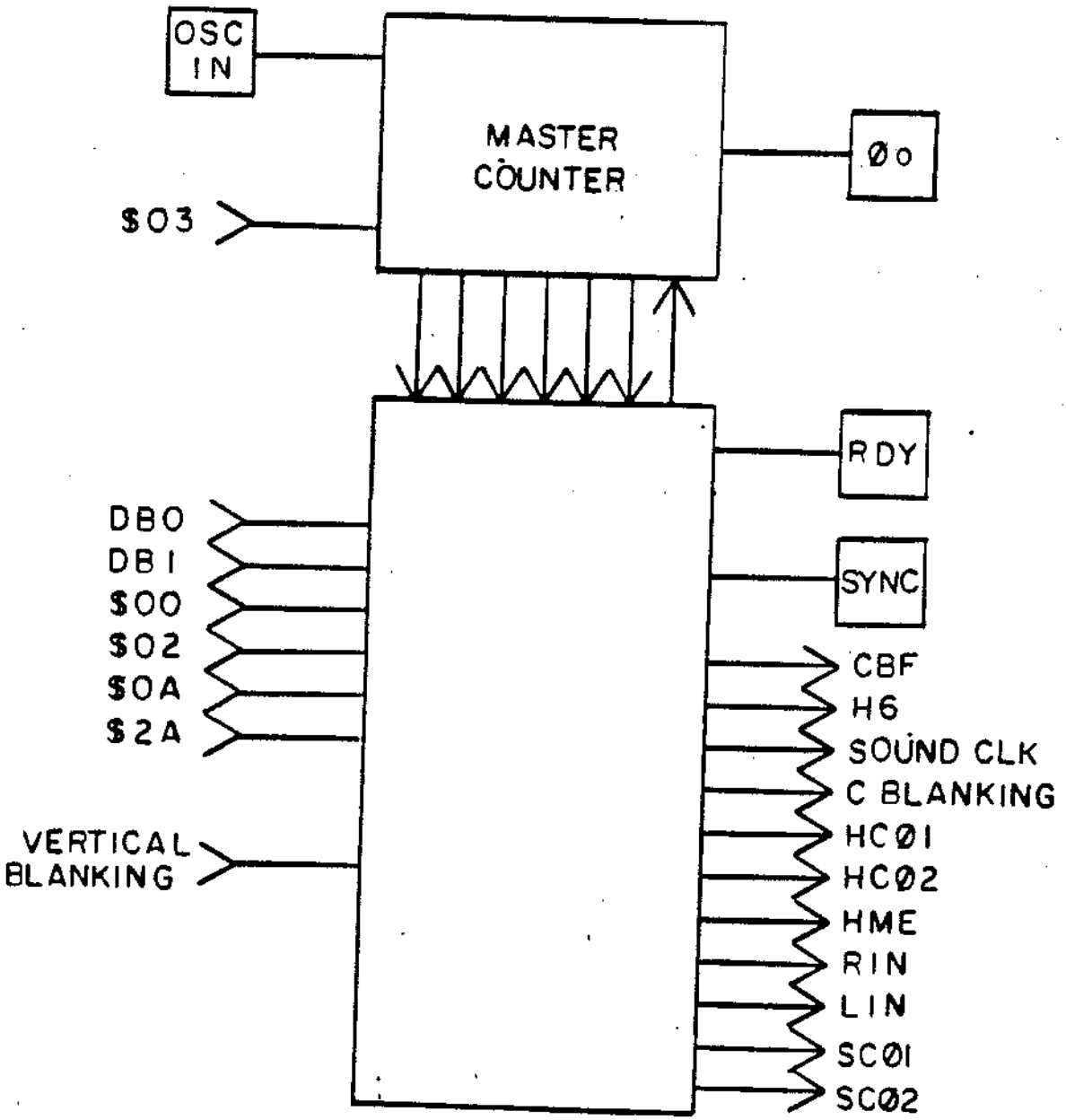
VIC BLOCK DIAGRAM

FIGURE 2



REV B 4/27/81

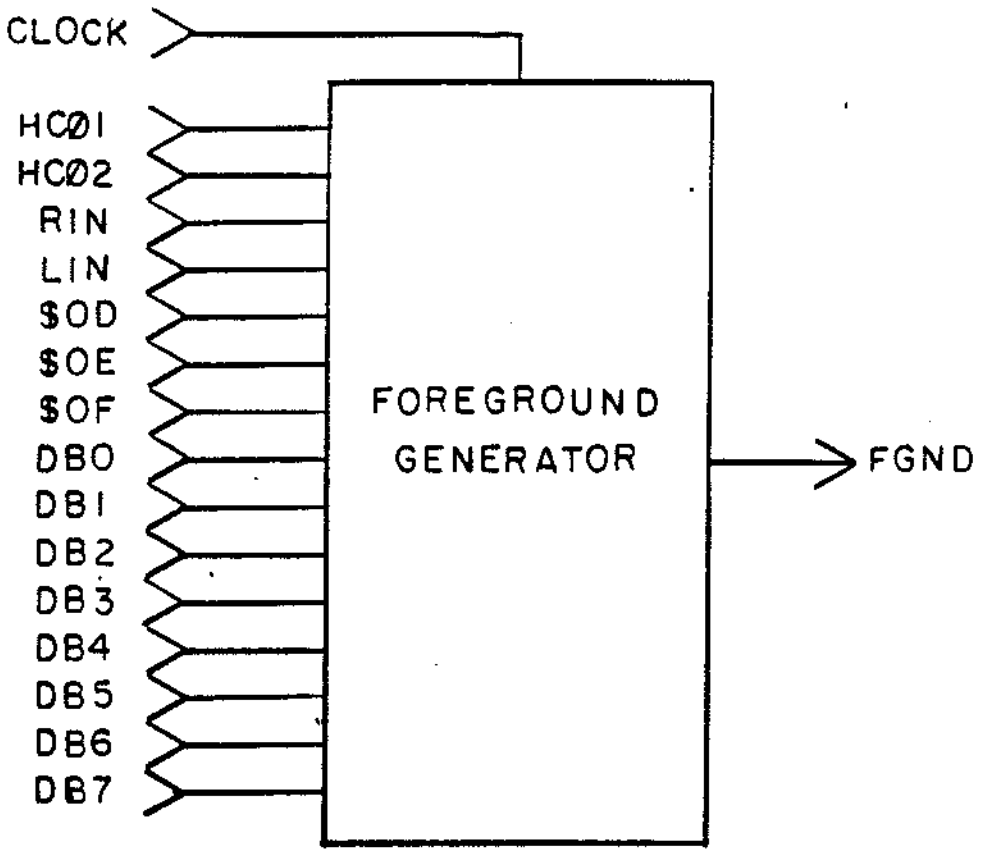
FIGURE 3



MASTER COUNTER LOGIC

REV A 4/29/81

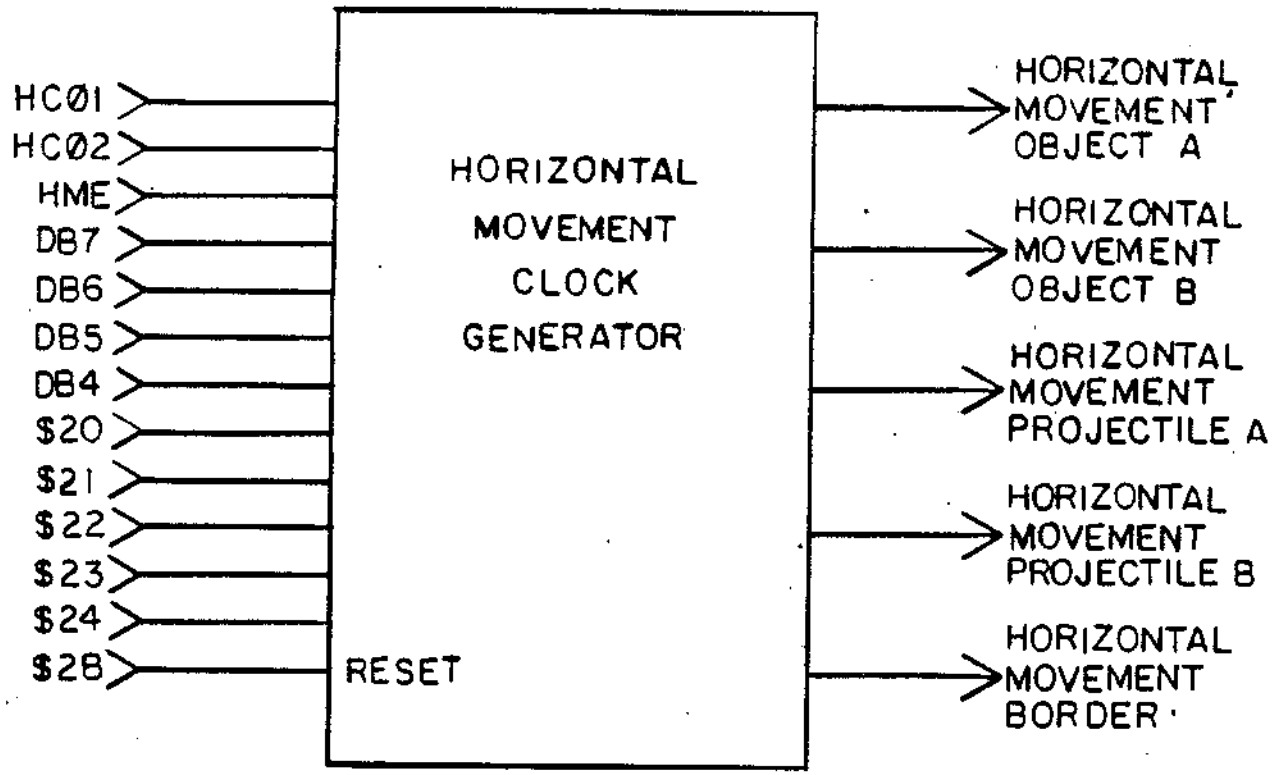
FIGURE 4



FOREGROUND GENERATOR
LOGIC

REV A 4/29/81

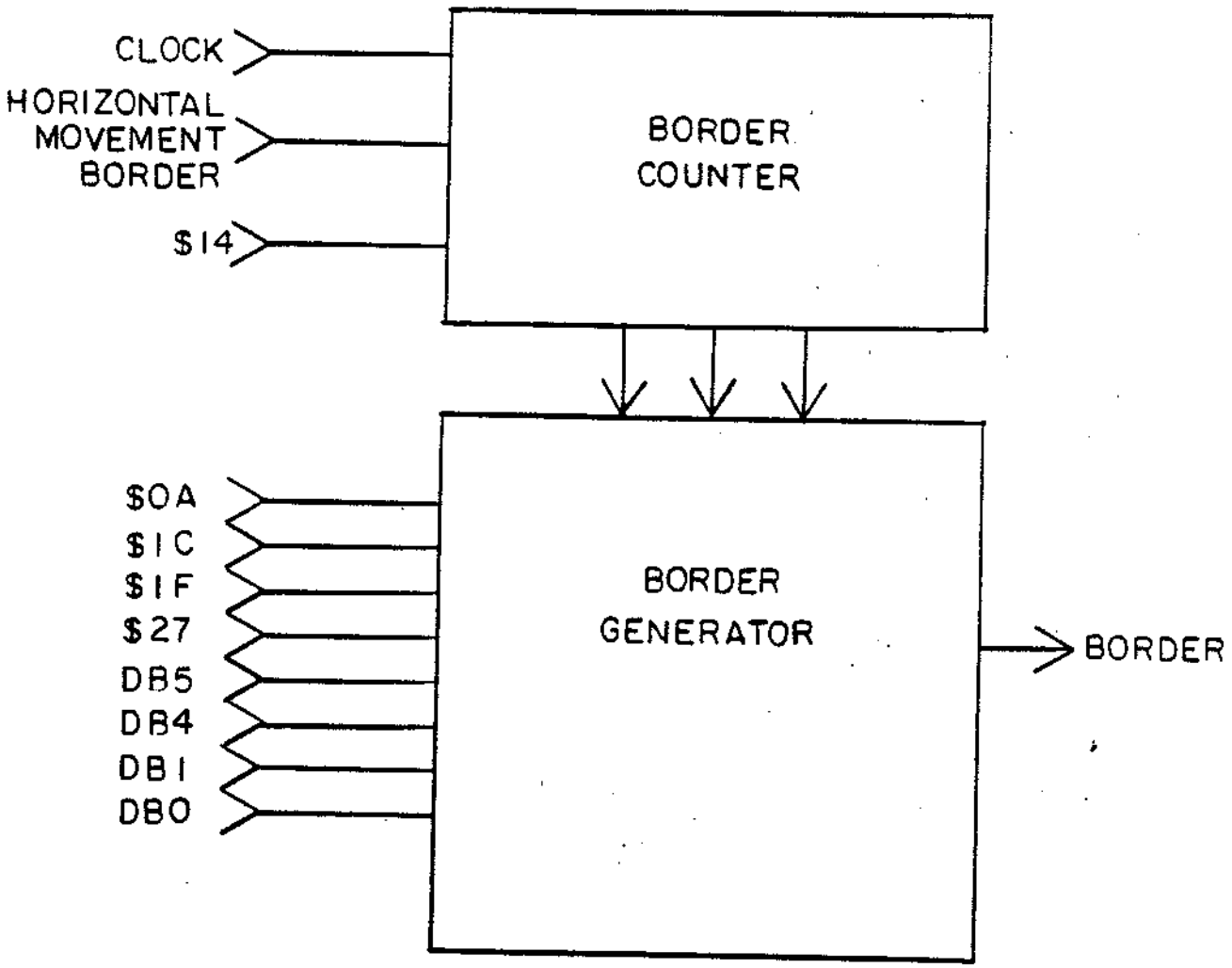
FIGURE 5



HORIZONTAL MOVEMENT

REV A 4/29/81

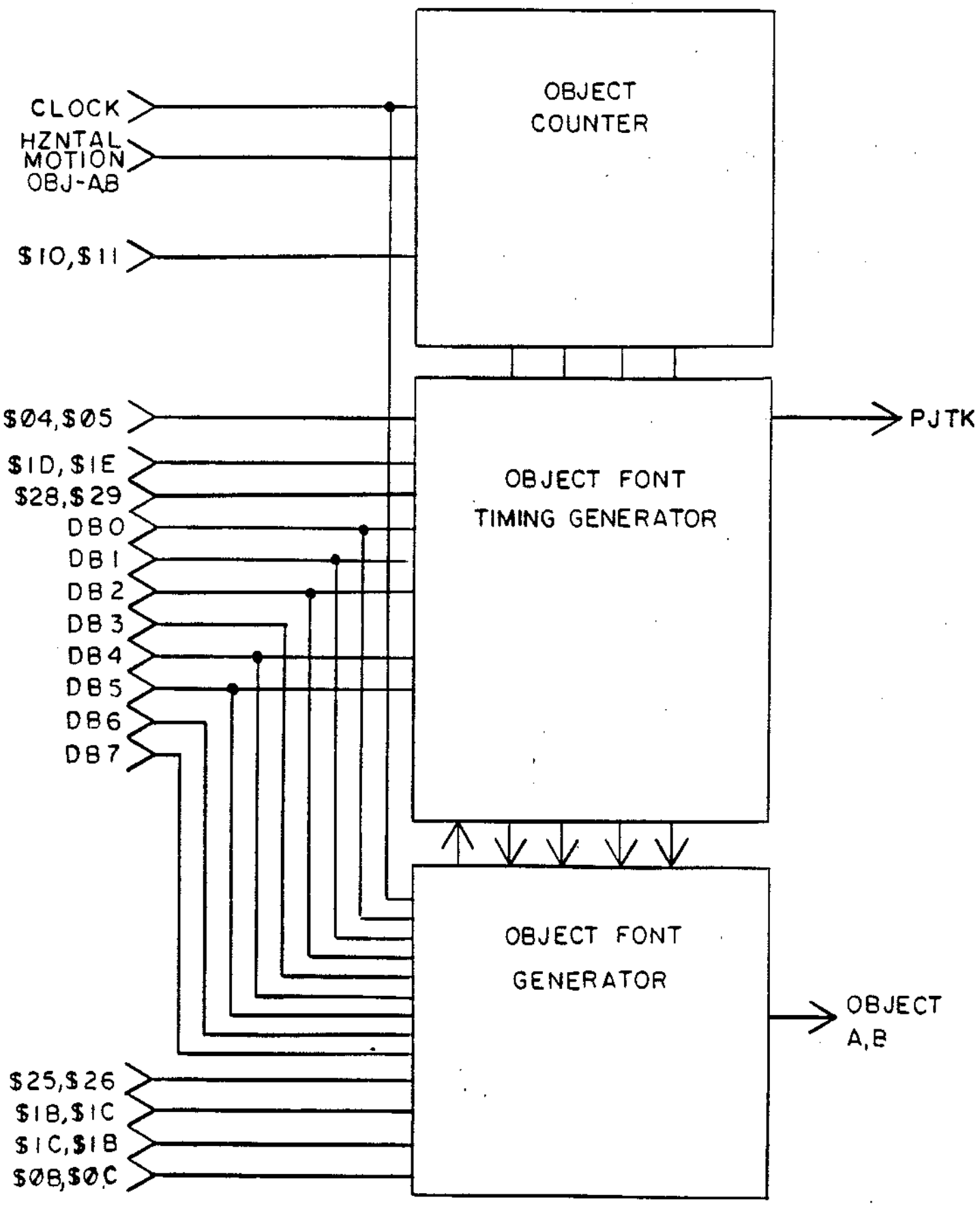
FIGURE 6



BORDER

REV A 4/28/81

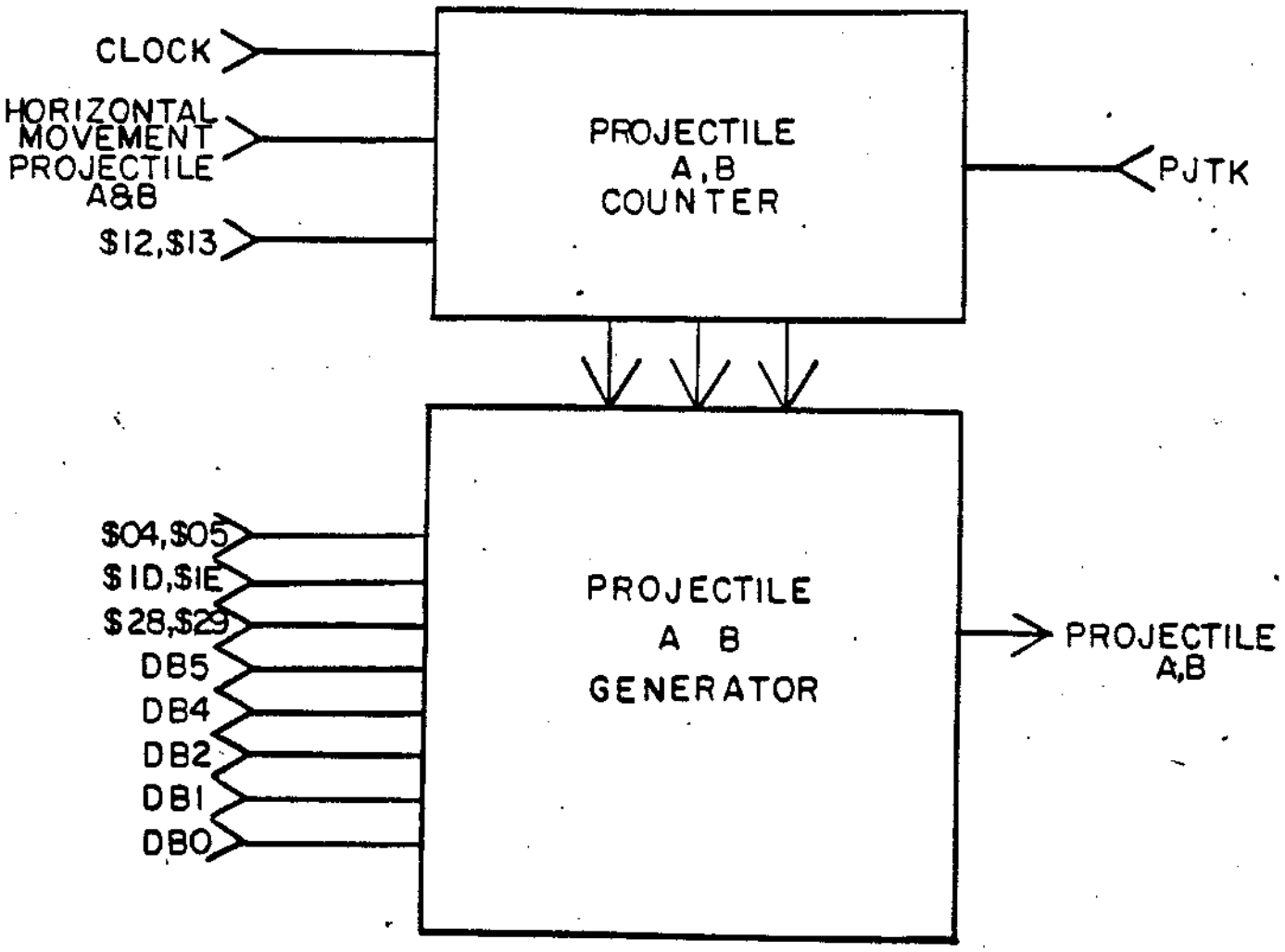
FIGURE 7



OBJECT GENERATOR LOGIC

REV A 7-21-81

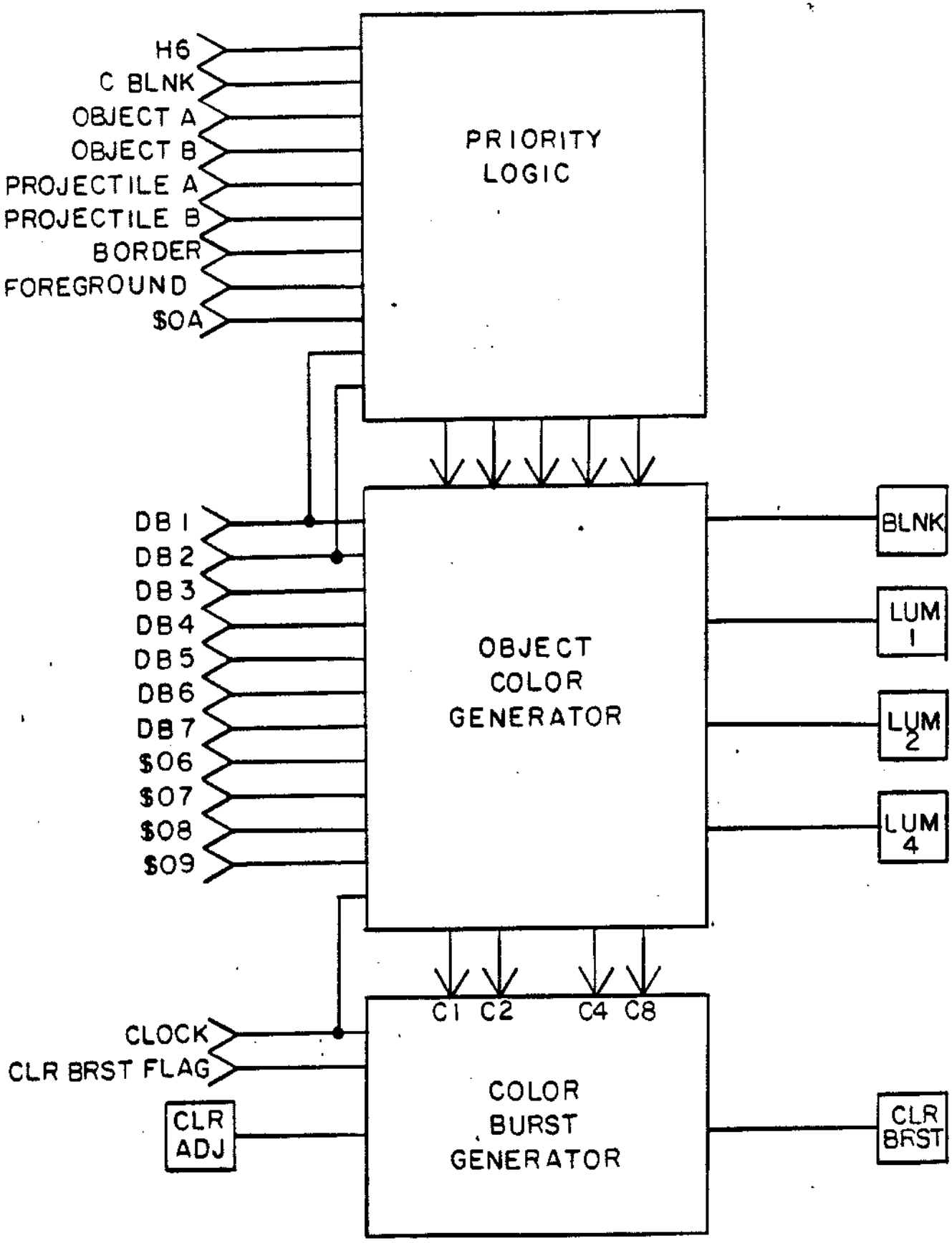
FIGURE 8



PROJECTILE A,B

REV A
7/31/81

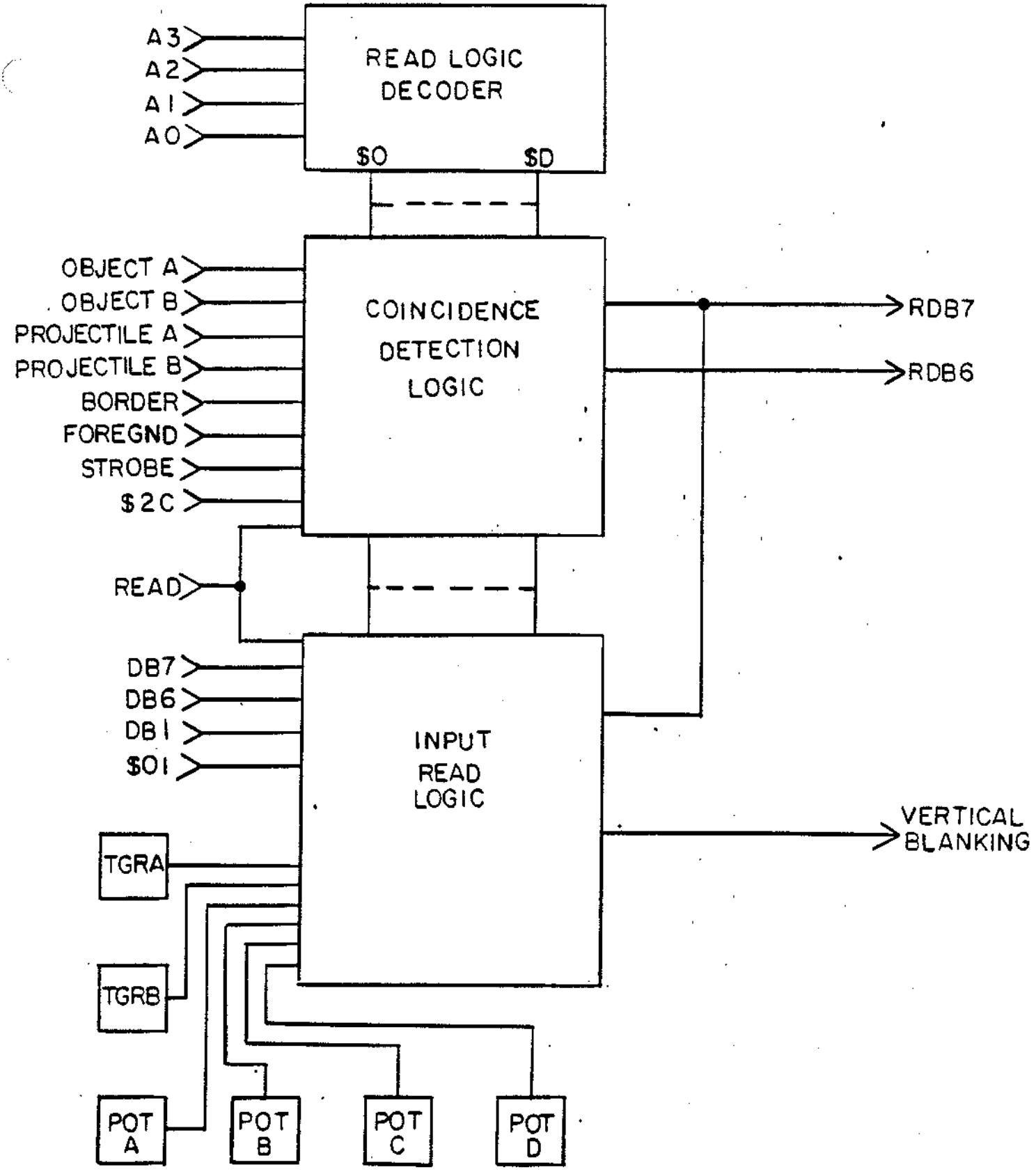
FIGURE 9



COLOR LOGIC

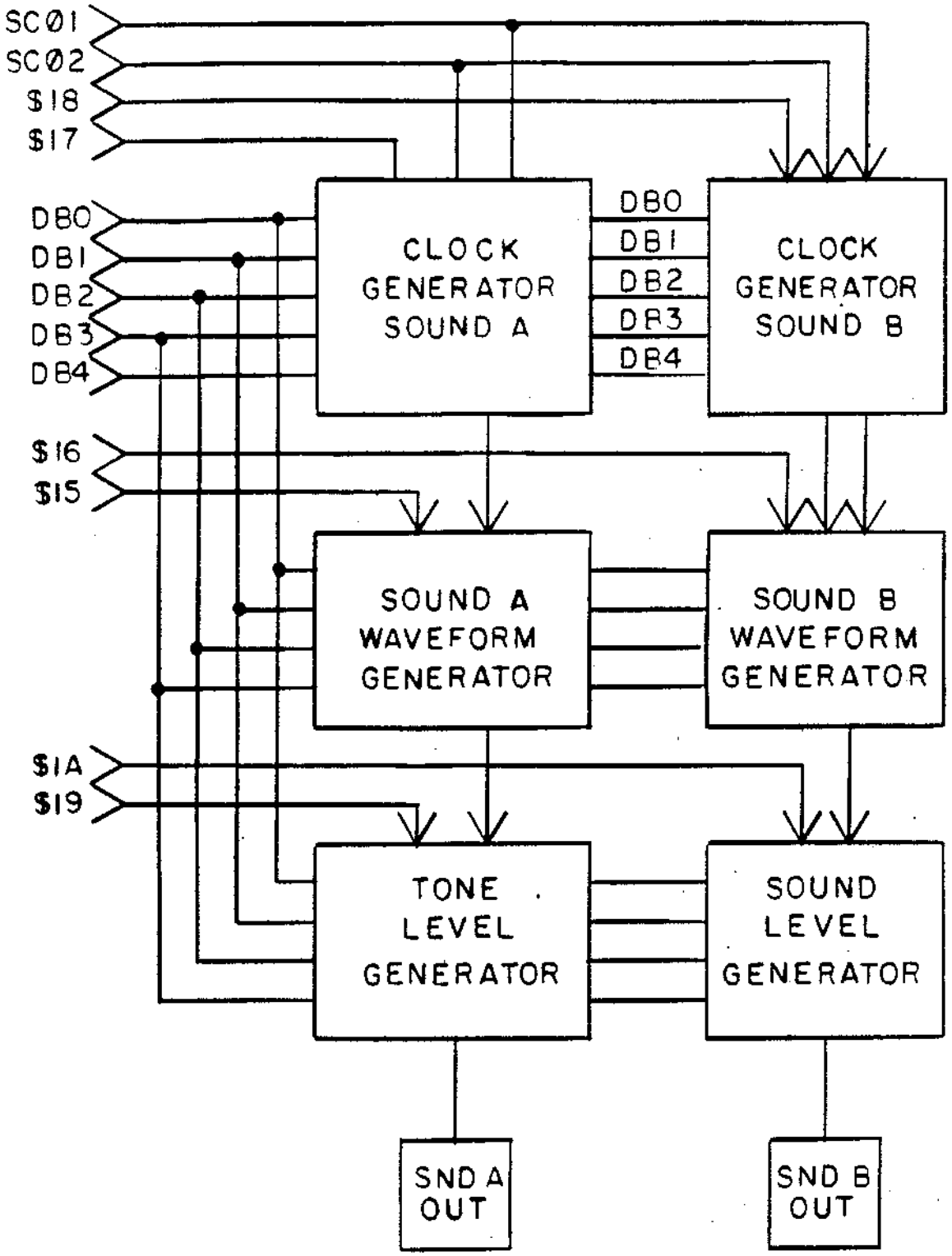
REV A 4/28/81

FIGURE 10



REV A 4/28/81

FIGURE 11



SOUND
GENERATORS

REV A
4/30/80

FIGURE 12

VIC THEORY OF CONTROL

The VIC is an Object oriented video circuit that is controlled by the Microprocessor. The VIC automatically generates the Horizontal scan for the Video System but the Microprocessor provides the Vertical scan for each frame.

The Microprocessor controls the Vertical Blanking and the Vertical Sync. The Microprocessor keeps track of the Horizontal line scan count and the Vertical position of the objects. At the correct Vertical position the Microprocessor enables the objects and supplies the font to the object generator one line at a time until the objects have been fully displayed.

The VIC keeps track of the horizontal positon of each object and retains that horizontal positon until directed to move the object left or right by the Microprocessor. There are 160 distinct horizontal locations for an object.

Horizontal motion of an object is the movement left or right at a constant rate (i.e., 1 horizontal unit per frame, 2 horizontal units per frame, or 1 horizontal unit every other frame).

Vertical motion of an object is the movement up or down at a constant rate (i.e., 1 horizontal line per frame, 2 horizontal lines per frame, or 1 horizontal line every other frame).

Diagonal movement of an object is the combination of horizontal and vertical motion of an object at the same time. The apparent direction of movement of an object can be determined by using vector arithmetic. Motion in any one plane can be defined as being either positive (+) or negative (-).

The two directions of motions are divided into two planes; one called X, for horizontal motion, and one called Y, for vertical motion. These two planes of motion will intersect each other at a 90 degree angle and will form the basis of an X-Y graph on which can be shown vectored motion. Vertical motion towards the top of the screen will be positive (+Y) and towards the bottom of the screen will be negative (-Y). Horizontal motion towards the right will be positive (+X) and towards the left will be negative (-X). The X-Y graph is then broken into four quadrants and each of the quadrants is numbered from 1 to 4. The first quadrant is in the upper right hand corner and is represented by both +X and +Y motion, the second quadrant is the lower right quadrant and is the +X and -Y motion. The third quadrant is the lower left quadrant and is the -X and -Y motion, the fourth quadrant is the upper left quadrant and is -X and +Y motion.

The horizontal movement range provided by the VIC is -7 to +8 horizontal units each time the motion is enabled. The motion logic (HMENB) is normally enabled only one time per frame or less. The maximum amount of horizontal motion normally used is less than 7 units per frame.

For the purpose of this example which should cover most applications the vertical motion will be limited from -8 to +8 horizontal lines per frame. As seen in figure 13 the amount of the motion can be shown as a vector using the value of horizontal motion and vertical motion on the graph in figure 14 and the distance from the intersection of the X and Y axis to the value of motion is the relative speed of the object.

In figure 15 vector A shows a moderate rate of motion in the first quadrant, the object motion shown by vector B is in the opposite direction at half of the rate of vector A. Both of these examples show movement where the number of horizonatal units of motion are equal to the number of horizontal lines of vertical motion. Vector C shows an example of a motion vector where X an Y are not equal.

Figure 14 can be used to calculate the angles of motion that can be generated on the screen using the VIC. It should be noted that the aspect ratio of the Video Screen is 4:3 and the actual angle seen will vary with this relationship. By making copies of figure 14 it will be possible to actually draw each vector and examine the rate of movement and the angle.

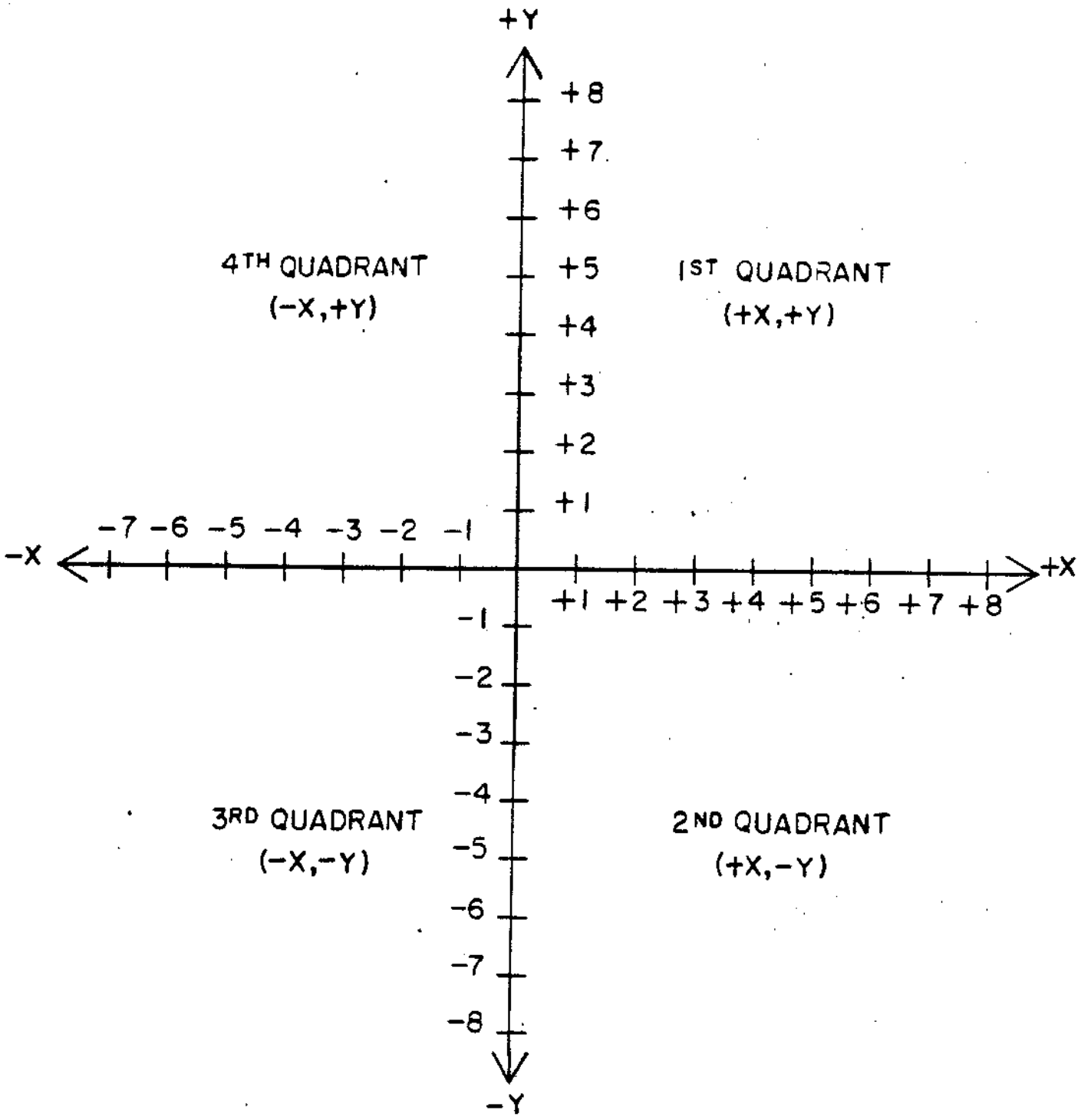


FIGURE 13

REV A 2-11-80

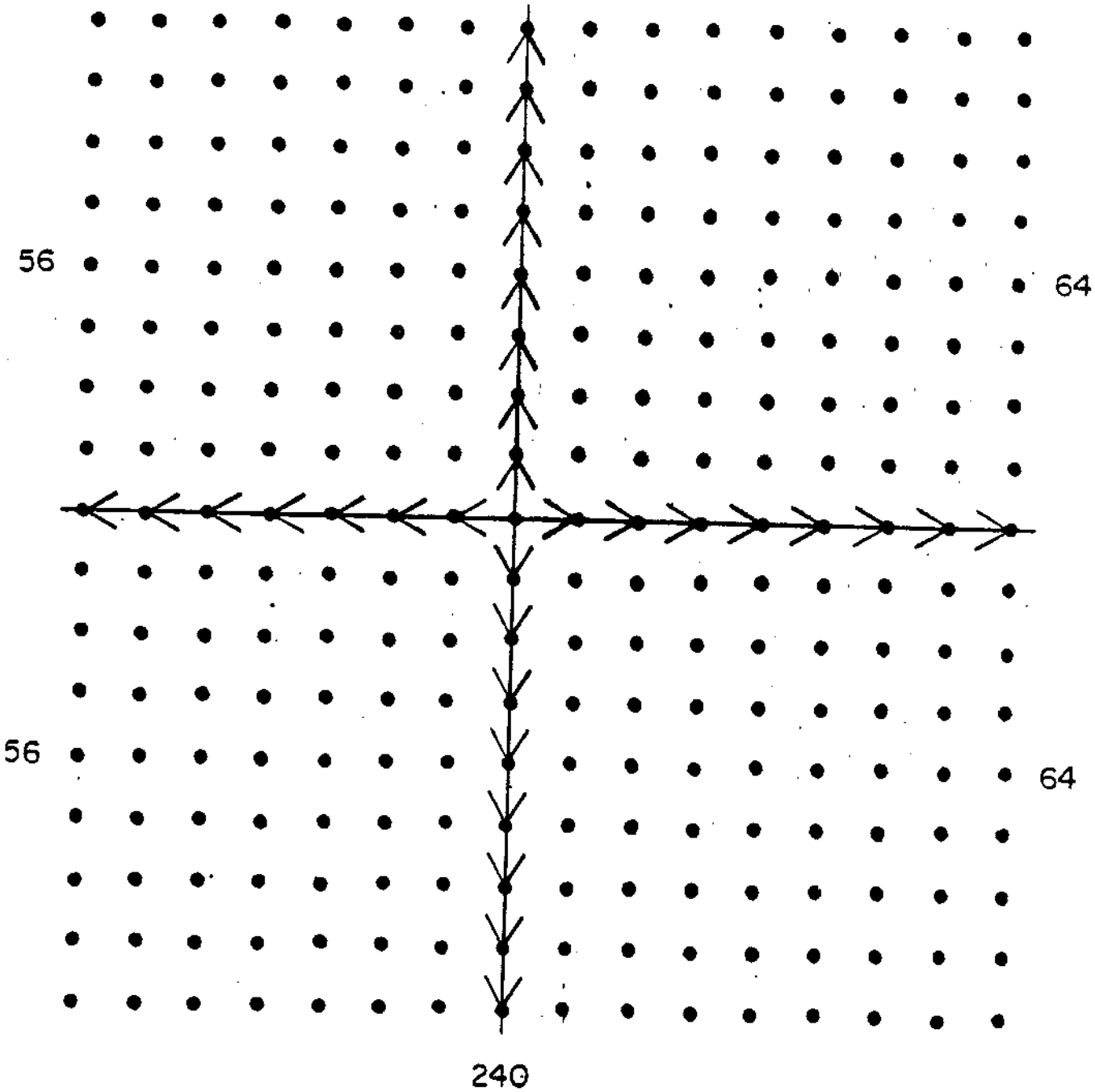


FIGURE 14 REV A 2-11-80

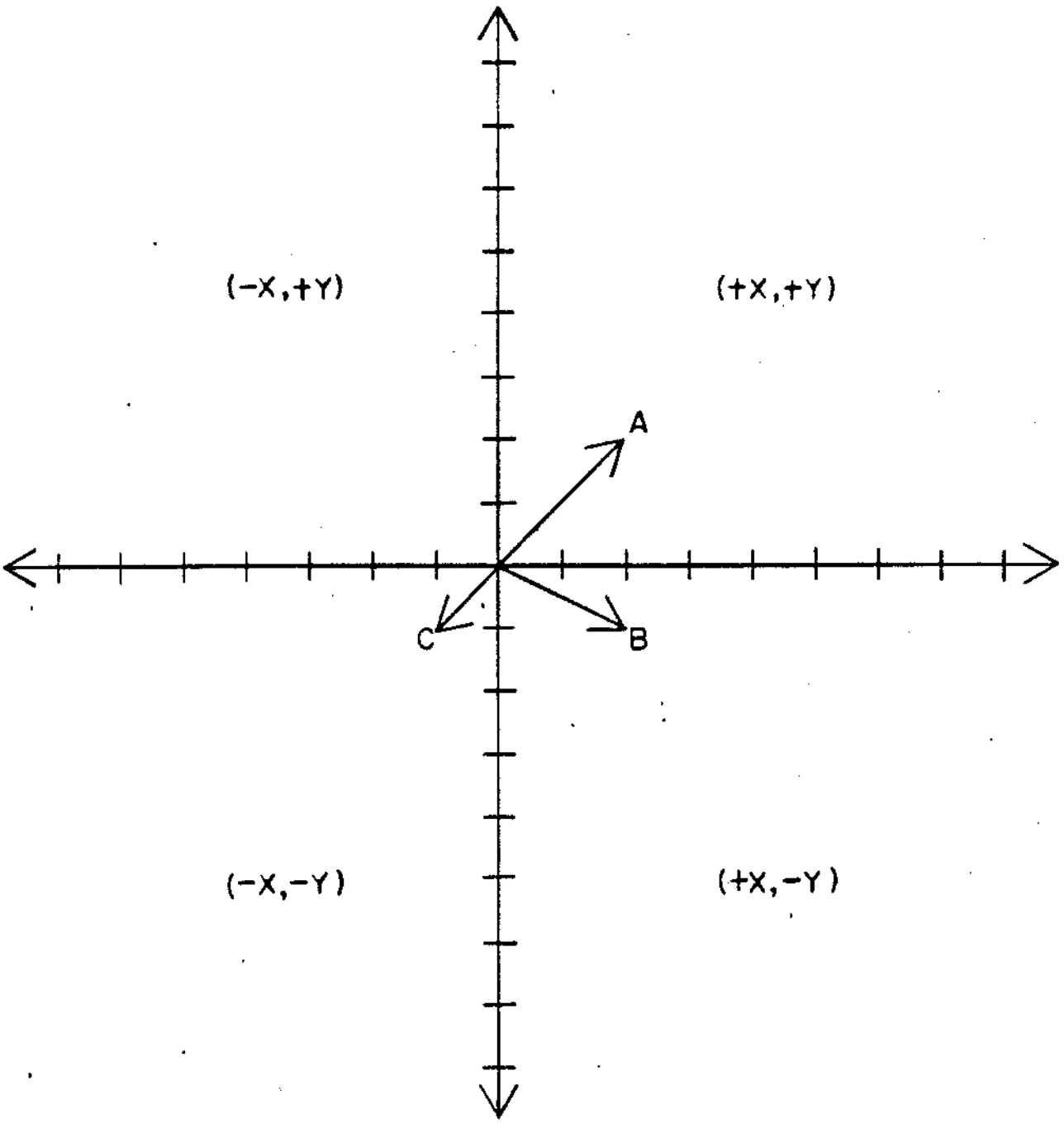


FIGURE 15 REV A 2-11-80

SYNC & BLANKING

The VIC generates a frame of video information under the direct control of the MPU. The VIC will automatically generate the Horizontal Blanking, Horizontal Synchronization, and Color Burst signals. The MPU must either directly or indirectly keep track of the number of horizontal lines that have been scanned. At the correct time, based on the number of lines scanned, the MPU will write to the VIC and turn ON the Vertical Blanking. Then, according to the timing requirement of the system, the MPU will wait the correct number of lines and turn on the Vertical Sync. The MPU will again wait for the correct period of time and then turn off the Vertical Sync, and again at the proper time, turn off the Vertical Blanking and begin a new frame of video information. Below is a list of nominal or commonly used timing specifications for the VIC vertical scan format:

Blanking Period Total	Sync Period
21 lines	3 lines

Start of Blanking to Sync	Sync to Blanking End
3 lines	15 lines

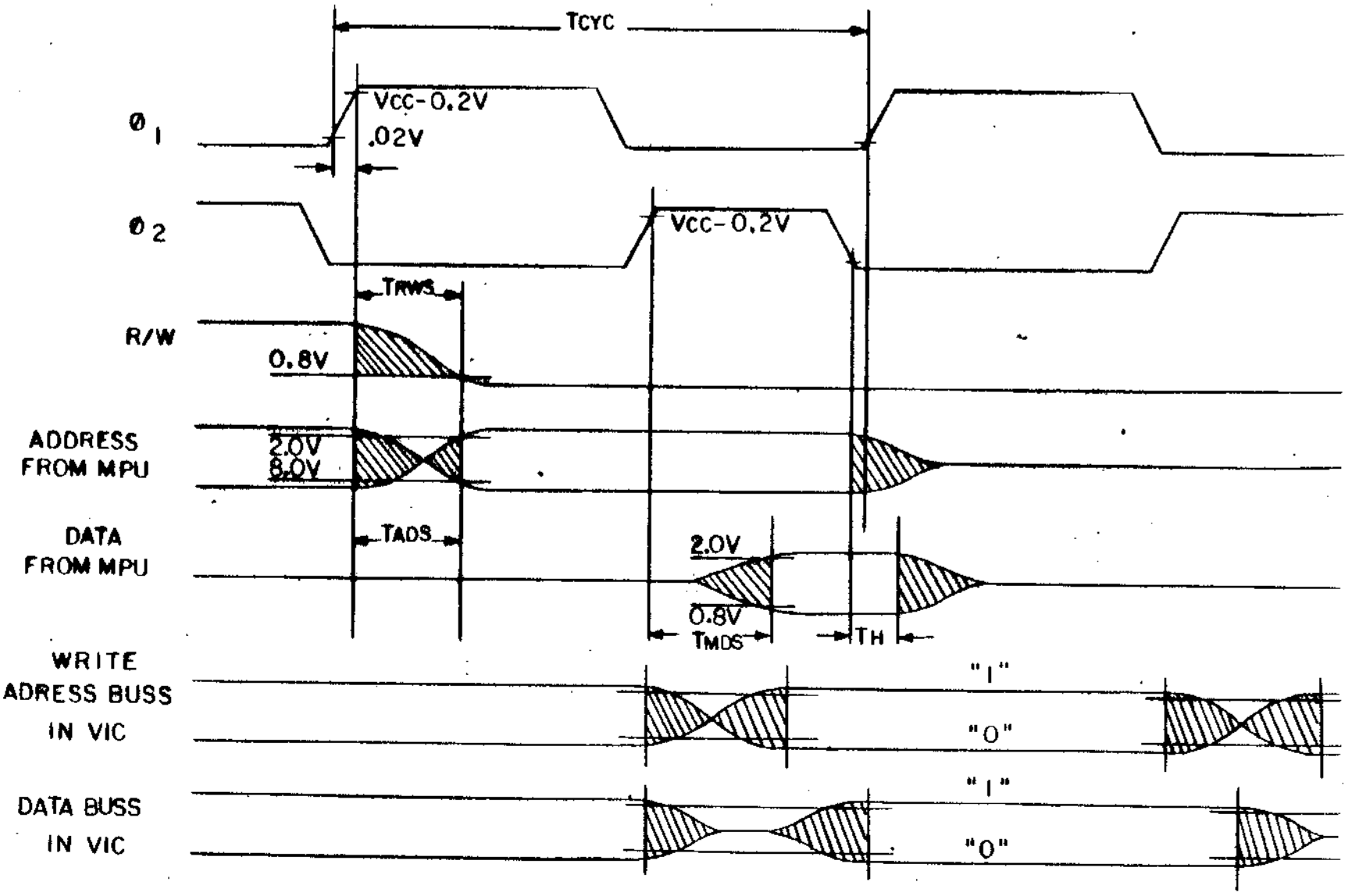
For a NTSC Standard television there will be 262 lines per frame. The Vertical Sync period should always be 3 lines but number of visible lines added to the number of Vertical Blanked lines should add up to 259.

BUS STRUCTURE

The Microcomputer System is organized around two primary Busses and one secondary Bus. The two primary Busses each consist of a set of parallel paths which can be used to transfer binary information between the devices in a Microcomputer System. These two Busses are the Address Bus and the Data Bus. The third Bus is a control oriented bus consisting of the Microcomputer System Clock (Phase 2), the READ/WRITE Control Line, the RESET Line, and the RDY (Ready) Line.

The timing of all Data transfers are controlled by the System Clock. The clock that we use as a reference is Phase 2 (Ö2). When Ö2 is low the Address Lines change, and when Ö2 is high the Data Lines change. The VIC accepts the Data and Address information when Ö2 is high and executes the Instruction or stores the Data as Ö2 goes low during the next clock cycle. The timing of the signals on the Address Bus, Data Bus, and R/W line are shown in figure 16 and figure 17.

Data on the data lines and address lines will be referenced in hexadecimal (base 16) form. Each hexadecimal digit represents 4 binary bits. Data on the Data Bus is represented as 2 hexadecimal digits (8 bits) and data on the Address Bus is represented as a two digit (page 0 only) or four digit (16 bits) number. All hexadecimal numbers are preceded by a "$".

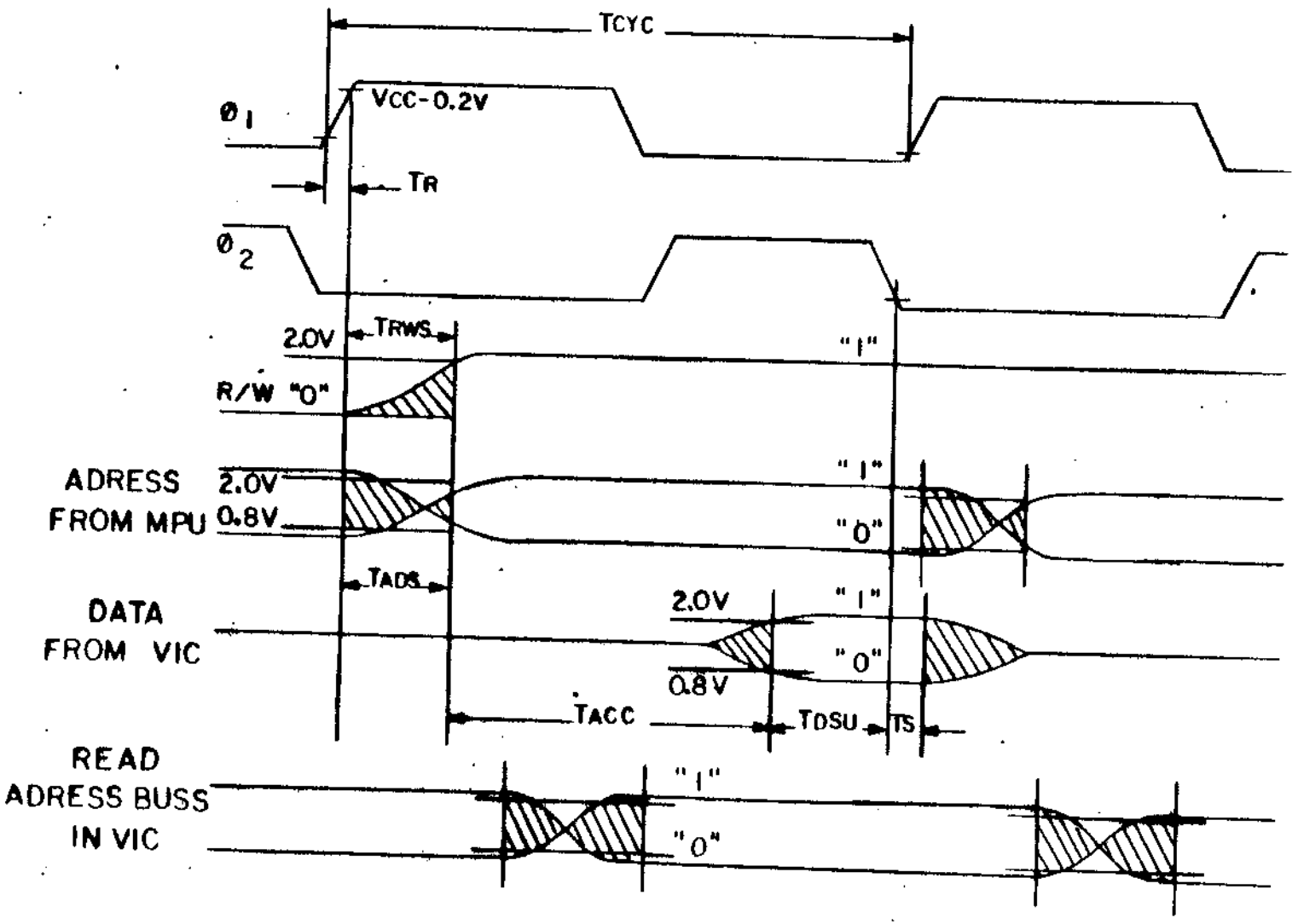


TIMING FOR WRITING TO VIC

FIGURE 16

REV A
1-24-80

-31-



REV A
1-24-80

TIMING FOR READING DATA FROM VIC

FIGURE 17

-32-

ADDRESS BUS

The Address Bus is used to transfer the Address generated by the Microprocessor to the Address Inputs of the Memory, the Peripheral Interface Circuit, and the VIC. The Microprocessor is the only source of Addresses in the System, so this Bus is referred to as unidirectional. The Address Bus consists of 13 lines. This allows the Microprocessor to access (READ or WRITE) up to a total of 8,192 different address locations.

The 8,192 address locations are broken up into pages; there are 256 address locations in each page and 32 pages. The addresses are expressed in hexadecimal (base 16) form. One page is expressed as having an address range of $00 to $FF (256 locations). The pages are also numbered in the same manner and are labeled from $00 to $FF. The entire address range (8,192 locations) is numbered from $0000 to $0FFF and $F000 to $FFFF.

There are 2 Pages of the Addresses that have very special importance in the Microcomputer System. These are Page 0 ($0000 to $00FF) and Page 1 ($0100 to $01FF). Page 0 has special importance because the Microprocessor has a special addressing mode in which it can easily operate on the Addresses between $0000 and $00FF. Page 1 has special importance because the Microprocessor requires RAM Memory in the top of Page 1 ($0100 to $01FF) for the Microprocessor Stack.

The VIC is located in Page 0. The VIC has 45 addresses and Page 0 has 256 addresses. The VIC occupies addresses $00 thru $2C, and $40 thru $6C transparently.

The VIC is memory mapped and each of the 45 addresses has a special function. Some of the address locations are control registers or data registers and are controlled by the data stored in these registers. Others are controlled only by addressing a particular address during a write cycle.

DATA BUS

The Data Bus is used by the Microprocessor to communicate with the VIC. The Microprocessor also uses the Data Bus to communicate with the other circuits in the Microcomputer system. The Data Bus is an 8 bit bi-directional data path between the Microprocessor, the VIC, the Program ROM, and the PIC. The Data Bus lines transfer Data from the Microprocessor to the VIC during the Write operation, and transfers Data from the VIC into the Microprocessor during the Read operation. All Data and most Instructions are transmitted to the VIC on the Data Bus.

The direction of the Data transfer is controlled by the READ/WRITE (R/W) Line on the Microprocessor. This line performs the Write Enable function when it is low ("0" state). When it is in this state the Data on the Data Bus will be received by the VIC and stored in the appropriate Data Registers corresponding to the address on the Address Bus.

When the Read Enable functions is being executed on the VIC by the Microprocessor, the VIC outputs Data onto the Data Bus. The VIC only uses 2 of the 8 Data Lines for outputing information, unlike most other Peripheral Devices which use all 8 Data Lines. DB6 and DB7 are the Data Lines used by the VIC to communicate information to the Microprocessor. When the Microprocessor performs the BIT Instruction on the VIC the Data on DB6 and DB7

are set directly into the V Flag Register and the N Flag Register of the Microprocessor. The Microprocessor can then perform one of the following Instructions based on the status of the V or N Flag: BVC, BVS, BPL, or BMI.

SYSTEM MEMORY MAP

The followng Memory Map outlines the System Memory for the Microcomputer System:

$F800-$FFFF	PROGRAM ROM HIGH (16K = 2K X 8)
$F000-$F7FF	PROGRAM ROM LOW (32K = 4K X 8)
$0280-$02FF	I/O INTERFACE & TIMER (PIC)
$0180-$01FF	STACK RAM (PAGE 1) (PIC)
$0080-$00FF	RAM (PAGE 0) (PIC)
$0000-$002C	VIC

VIC MEMORY MAP

The following Memory Map shows the Address locations of the VIC functions in Page 0 from $00 thru $2C:

LABEL		FUNCTION
$00	VSYNC	Vertical Sync Register
$01	VBLNK	Vertical Blanking and Input Control Register
$02	HSYNC	RDY (READY) Line Set
$03	CRHOR	Horizontal Counter Reset
$04	OBASR	Object A and Projectile A Size and Repeats Register
$05	OBBSR	Objects B and Projectile A Size and Repeats Register
$06	CLROA	Object A and Projectile A Color Register
$07	CLROB	Object B and Projectile B Color Register
$08	CLRFB	Foreground and Border Color Register
$09	CLRBG	Background Color Register
$0A	BDRSZ	Border Size and Foreground Control Register
$0B	OBADR	Object A Scan Direction Register
$0C	OBBDR	Object B Scan Direction Register
$0D	FGNDA	Foreground Font Register A
$0E	FGNDB	Foreground Font Register B
$0F	FGNDC	Foreground Font Register C

LABEL		FUNCTION
$10	CROBA	Object A Counter Reset
$11	CROBB	Object B Counter Reset
$12	CRPJA	Projectile A Counter Reset
$13	CRPJB	Projectile B Counter Reset
$14	CRBDR	Border Counter Reset
$15	SNDAS	Sound A Select Register
$16	SNDBS	Sound B Select Register
$17	FREQA	Sound A Frequency Select Register
$18	FREQB	Sound B Frequency Select Register
$19	LVLSA	Sound A Level Register
$1A	LVLSB	Sound B Level Register
$1B	OBJAF	Object A Font Register
$1C	OBJBF	Object B Font Register
$1D	PJAEB	Projectile A Enable Register
$1E	PJBEB	Projectile B Enable Register
$1F	BDREB	Border Enable Register
$20	HMOBA	Object A Horizontal Movement Register
$21	HMOBB	Object B Horizontal Movement Register
$22	HMPJA	Projectile A Horizontal Movement Register
$23	HMPJB	Projectile B Horizontal Movement Register
$24	HMBDR	Border Horizontal Movement Register
$25	OBARS	Object A Font Select Register
$26	OBBRS	Object B Font Select Register
$27	BDRES	Border Enable Select Register

LABEL		FUNCTION
$28	PJATK	Projectile A Tracking and Disable Register
$29	PJBTK	Projectile B Tracking and Disable Register
$2A	HMENB	Horizontal Movement Enable
$2B	HMRST	Horizontal Movement Registers Reset
$2C	OCRST	Coincidence Logic Latch Reset

PIC MEMORY MAP

$0280	I/O PORT A (PLAYER CONTROLS)
$0281	I/O PORT A DATA DIRECTION REGISTER
$0282	I/O PORT B (FRONT PANEL CONTROLS)
$0283	I/O PORT B DATA DIRECTION REGISTER
$0284	TIMER (READ)
$0285	TIMER FLAG (READ)
$0294	Ö2 divided by 1 (WRITE)
$0295	Ö2 divided by 8 (WRITE)
$0296	Ö2 divided by 64 (WRITE)
$0297	Ö2 divided by 1024 (WRITE)

$0280 (READ) and $0281 (WRITE)

PA0	RIGHT	CONTROLLER	PIN 1
PA1	RIGHT	CONTROLLER	PIN 2
PA2	RIGHT	CONTROLLER	PIN 3
PA3	RIGHT	CONTROLLER	PIN 4
PA4	LEFT	CONTROLLER	PIN 1
PA5	LEFT	CONTROLLER	PIN 2
PA6	LEFT	CONTROLLER	PIN 3
PA7	LEFT	CONTROLLER	PIN 4

$0282 (READ) and $0283 (WRITE)

PB0	GAME RESET	"1" = UP (NORMAL) "0" = DOWN (DEPRESSED)
PB1	GAME SELECT	"1" = UP (NORMAL) "0" = DOWN (DEPRESSED)
PB2	NO CONNECTION	
PB3	TV TYPE	"1" = COLOR "0" = B/W
PB4	NO CONNECTION	
PB5	NO CONNECTION	
PB6	LEFT DIFFICULTY	"1" = A (UP) "0" = B (DOWN)
PB7	RIGHT DIFFICULTY	"1" = A (UP) "0" = B (DOWN)

SYNCHRONIZATION OF MPU TO VIC

The Microcomputer program that is used to generate the Video Display does not have a direct method of determining the condition of the horizontal scan of the VIC. The Microprocessor must have some method of counting the number of Horizontal Lines that have been scanned as well as a way of determining when the Horizontal Blanking is on.

The Microprocessor can be synchronized to the VIC Horizontal Scan by using a closed loop type of control circuit. The 6500 series Microprocessor has the ability to be halted during a READ Cycle by pulling the RDY (Ready) Input to the Microprocessor to the low ("0") state. The VIC has an output that controls the RDY Line and the Microprocessor controls the VIC forming a closed loop control circuit.

After the Microprocessor has executed the portion of the program for a given Horizontal Scan and it has used less than the maximum number of Microprocessor Cycles for that Horizontal Scan we must halt the Microprocessor until the beginning of the next Horiztonal Scan. In this manner we use up the previously unused Microprocessor Clock Cycles by being in a wait mode.

The Wait Function is initiated by executing a WRITE to HYSNC ($02) using the STA, or STX, or STY Instruction. When the HSYNC Command is executed, the VIC will set the RDY Line to the Wait condition, then as the Horizontal Scan ends, the VIC will automatically release the Microprocessor from the Wait Mode and normal program execution will resume.

VIC FUNCTIONS

This Chapter describes in detail the function of every Address and Data Bit that can be programmed in the VIC. There are drawings with many of the descriptions so that the programmer may see what is actually happening to the VIC as these Functions are executed.

The programmer should exercise caution when using certain Commands or Control Registers as some of these Commands and Control Registers are Video Scan Time sensitive. When a Function is Time sensitive, the description will explain when the Function can be executed. Execution of a Command or a WRITE Cycle to a Control Register will be described in Microprocessor Clock Cycles. There are 76 Microprocessor Clock Cycles in each Horizontal Scan.

The programmer is advised to utilize the Page 0 Address capability of the 6500 series Microprocessor when reading from or writing to the VIC. This will save one Byte in the ROM each time a READ or WRITE is programmed and will save one Microprocessor Clock Cycle each time a READ or WRITE is executed. Each of the Functions of the VIC are grouped together so that Indexed Addressing can be used efficiently when programming the VIC. The use of the Offset makes the transferring of Data from ROM to the VIC very software efficient.

VERITCAL SYNC CONTROL REGISTER

The VSYNC ($00) Vertical Sync Control Register is used to enable the Vertical Sync Function. This Register uses DB1.

The VIC automatically generates the Horizontal Sync Signal and the Color Burst. To generate the Vertical Sync, the Horizontal Sync signal is inverted. When DB1 is a "0", the normal Horizontal Sync signal is generated. When DB1 is a "1", the Horizontal Sync signal is inverted and the Color Burst is Disabled.

An example program that generates the Vertical Sync is shown in VBLNK ($01).

Figure 18, on the following page, shows the Horizontal Sync, Horizontal Blanking, Color Burst Flag, and Composite Video Signals automatically generated by the VIC when the Vertical Sync and Vertical Blanking are off.

ATARI NTSC SYNC

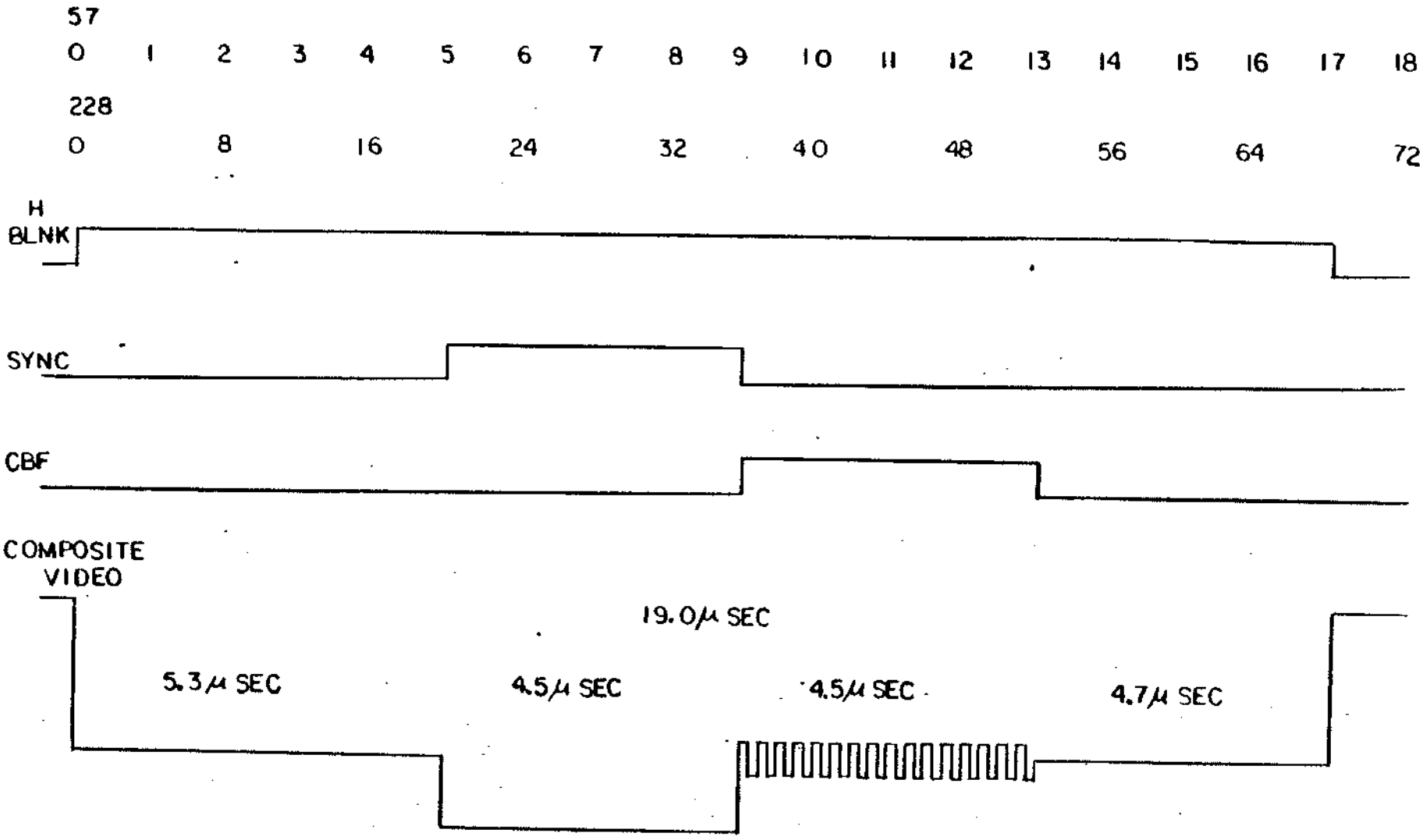


FIGURE 18

REV A 7-21-81

-47-

VERTICAL BLANKING

The VBLNK ($01) Vertical Blanking Control Register is a multiple function Register. It is used to control the Vertical Blanking, to control the Trigger Input Latches, and to control the Analog Input Capacitor Discharge Transistors. The Vertical Blanking Register uses DB1, DB6, and DB7.

The Vertical Blanking is controlled by using VBLNK ($01) Control Register. The Blanking is controlled by DB1. When DB1 is a "0", the Vertical Blanking is OFF. When DB1 is a "1", the Vertical Blanking is ON. The program to generate Vertical Blanking and Vertical Sync would look similar to the following sample program:

```
LDA #%XX****1*          ; Data to turn on Blanking
STA HSYNC               ; Wait for beginning of next line
STA VBLNK               ; Turn on Blanking
```

At this point, a program would be executed to wait until the Sync is to be turned on. Also, subroutines could be performed during part of this waiting period.

```
LDA #$02                ; Data to turn on Sync
STA HSYNC               ; Wait for beginning of next line
STA VSYNC               ; Turn on Sync
```

```
STA HSYNC          ; Wait for end of 1st line of Sync
STA HSYNC          ; Wait for end of 2nd line of Sync
LDA #$00           ; Data to turn off Sync
STA HSYNC          ; Wait for end of 3rd line of Sync
STA VSYNC          ; Turn off Sync
```

Again, a program would be executed to wait for the end of the blanking period. Also, subroutines could be performed during part of this waiting period.

```
LDA #%XX****0*     ; Data to turn off Blanking
STA HSYNC          ; Wait for beginning of next line
STA VBLNK          ; Turn off Blanking
```

At this point, the program would begin executing the next frame of video to be displayed.

NOTE: X = Bit is used to perform another function, not related to this example.

* = Bit not used, can be either 0 or 1.

The VBLNK Control Register also controls a portion of the Read logic. DB6 controls the data storage latches on pins 35 and 36. DB7 controls the capacitor discharge transistors on pins 37 thru 40.

Pins 35 and 36 are used as digital inputs. These pins can be programmed to be either level sensitive or edge sensitive. When DB6 is set to a "0", the logic state on the Input Pin will be passed directly to the Data Bus when a Read is performed on the address associated with that particular input. When DB6 is set to a "1", an internal latch is enabled. Once enabled, any negative input transition will set the latch to the logic "0" state and will remain set until it is cleared. The internal latch can only be cleared by setting DB6 to a "0" while the Input Pin is a logic "1".

Pins 37 thru 40 are dual purpose pins which can be used as A/D converter inputs or as digital Inputs. When used in the A/D converter mode, a capacitor is connected between the input pin and ground. A potentiometer and fixed resistor are connected between the pin and the positive power supply. The RC time constant must be selected to allow the capacitor to charge above the trigger level of the Input. The discharge transistors are turned on by setting DB7 to a "1". When the discharge transistor is ON, the charge stored in the capacitor is released to ground and restores the voltage across the capacitor to nearly 0. When DB7 is set to a "0", the discharge transistor is turned off and appears as an open circuit. The voltage across the capacitor will begin to increase at the rate determined by the following equations:

$$DV = (DT \times I) / C$$

$$I = (V / R)$$

DV is change in voltage, DT change in time, C value of capacitor, V voltage across potentiometer, and R value of potentiometer plus the value of the fixed resistor.

When the voltage across the capacitor exceeds the trip voltage of the Schmitt Trigger Input, the logic state of the pin will change in the Read Logic. The time that is required to exceed the trip voltage is used to determine the relative setting of the potentiometer; that is, the higher the value of the potentiometer, the longer it will take to exceed the trip voltage. We can determine the time it takes by periodically reading the status of the pin through the Read Logic. Each time we read the pin, we can increment a register, and when we detect the logic state change of the pin, the contents of the register will indicate the relative time that was required to reach the trip point.

When pins 37 thru 40 are used as digital Inputs, the Discharge Transistors are not used.

RDY LINE CONTROL

The HYSNC ($02) Command is used to synchronize the Horizontal Blanking and Horizontal scan of the VIC to the Microprocessor. This is a Command level Instruction which is executed only by writing to the Address. There are not any Data Bits associated with this Command, therefore the Data on the Data Bus during the WRITE Cycle will not affect in any way the execution of this Command. HSYNC can be executed using the STA, STX and STY Instructions.

This Command is used to halt the program execution until the beginning of the next Horizontal Sweep. At the beginning of the next Horizontal Sweep the Microprocessor will resume program execution. The beginning of a Horizontal Sweep is defined as the point where the Horizontal Blanking (HBLNK) line goes low and starts the Blanking period. Twenty-two (22) Microprocessor Clock Cycles can be executed during the Blanking period.

The programmer must be careful to exercise caution when writing Data to the VIC; which will cause a change to the visible area of the screen; when the Horizontal Blanking is off. Writing to the Font or Control sections of the Objects, Projectiles, Border, or Foreground should be done during the Horizontal and Vertical Blanking time or at a time that the Object is not visible.

The VIC executes the HSYNC Command by setting the RDY (READY) line to the low state ("0") and halting the Microprocessor. The VIC then returns the RDY line to the high state ("1") just before the Horizontal Blanking turns on again. When the RDY line has returned to the high state the program execution will then resume in a normal manner.

There is one Microprocessor Clock Cycle during which the HSYNC Command cannot be executed. It is located just before the Horizontal Blanking turns on. When the WRITE to HYSNC occurs during the very last Microprocessor Cycle, it will be ignored. This occurs when the number of Microprocessor Clock Cycles for the Instructions used during the Horizontal Scan are equal to the total number of Microprocessor Clock Cycles used in one or more Horizontal Scans, and the last Instruction is a HSYNC Command.

The following figures show the timing of the HSYNC Command and the RDY Line Control. Figure 19 shows the use of HSYNC during the middle of a Horizontal Scan and the action of the VIC to the RDY Line. Figure 19 also shows the release of the RDY by the the VIC. Figure 20 shows the relationship between Öo and the Horizontal Blanking Line and the synchronization of the Öo Line to the Horizontal Blanking. Figure 21 shows the critical WRITE areas at the beginning and end of a Horizontal Scan.

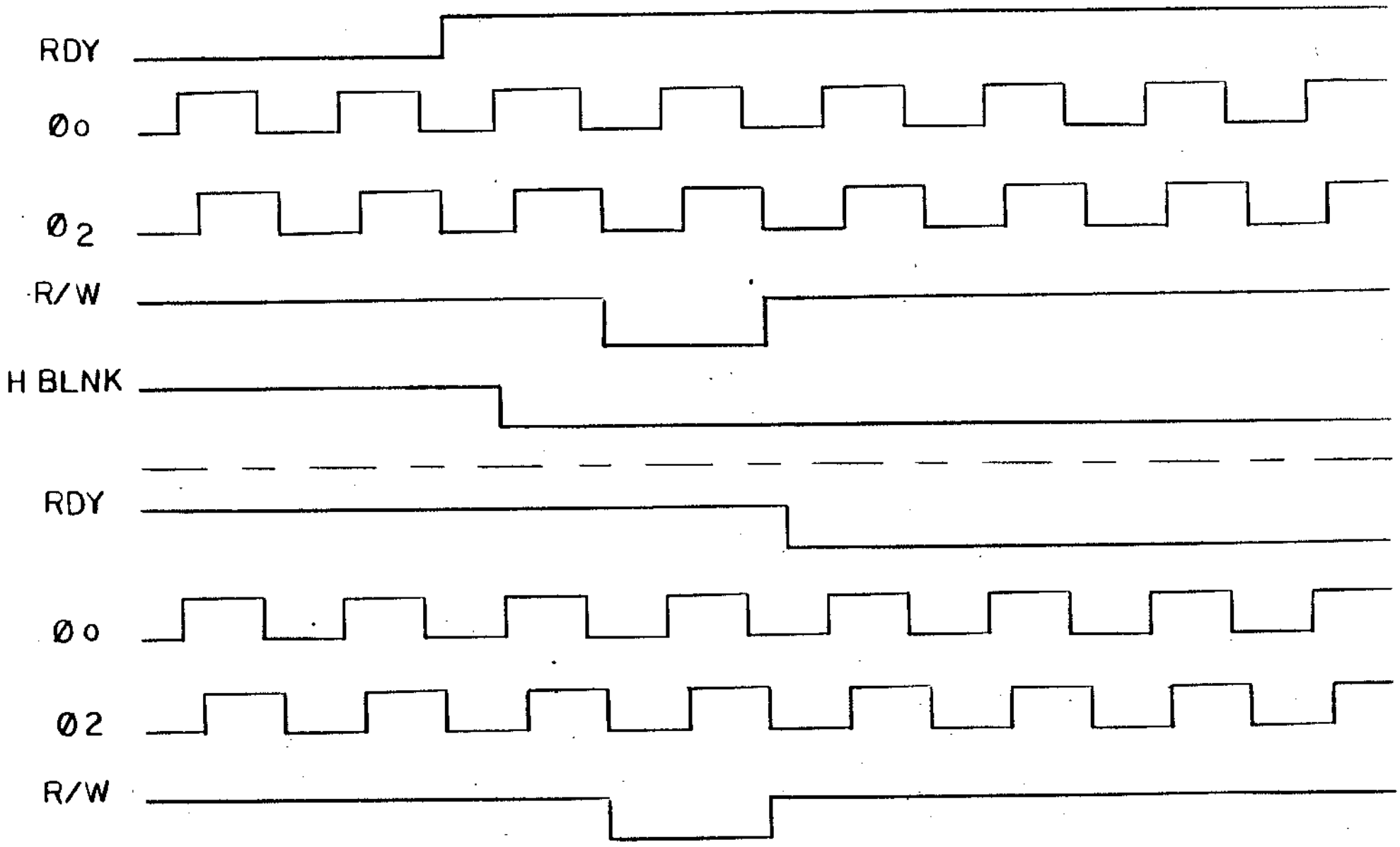


FIGURE 19

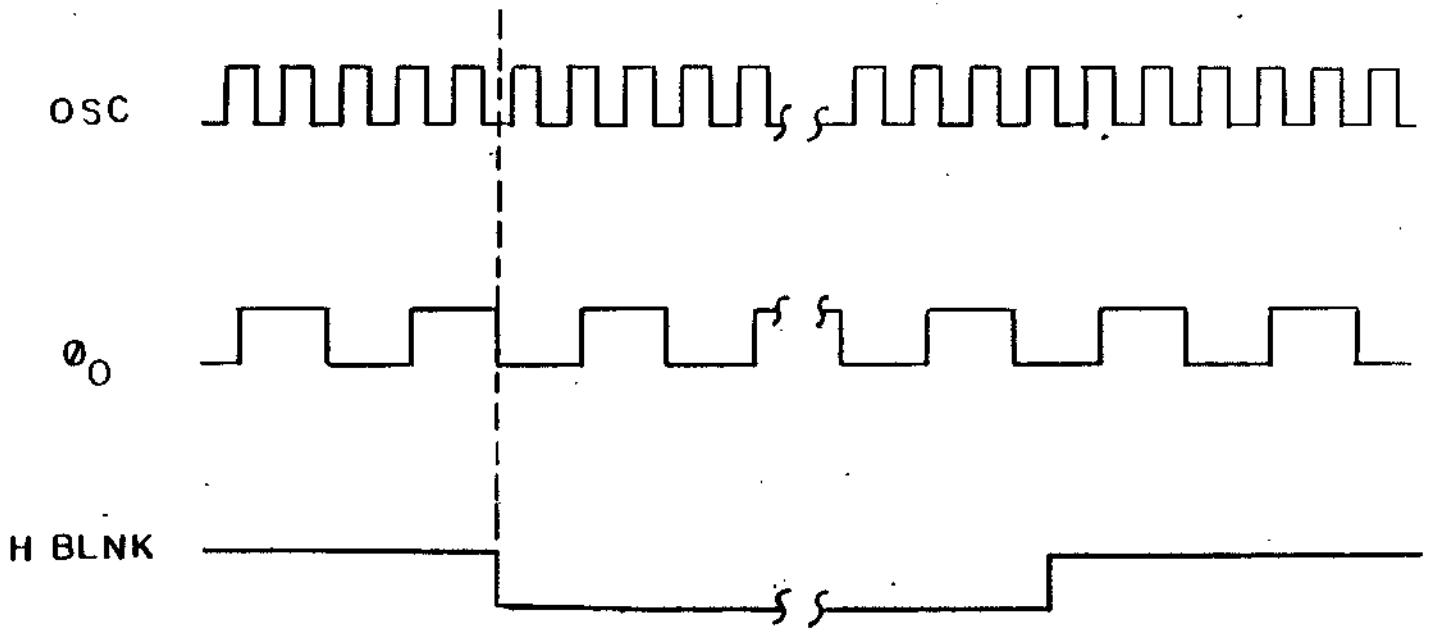
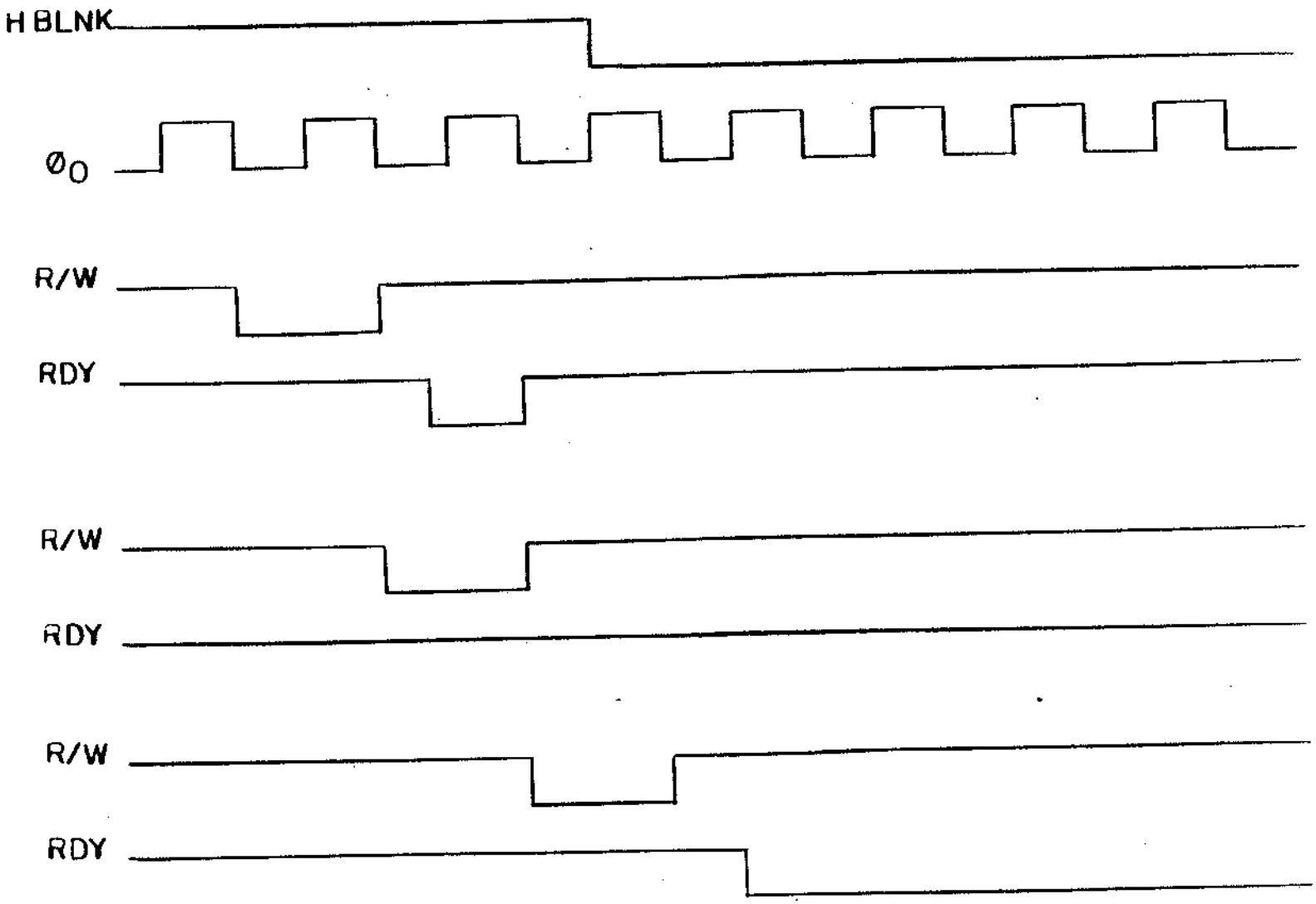


FIGURE 20



REV A 1-10-80

FIGURE 21

-56-

HORIZONTAL COUNTER RESETS

The Horizontal Counter Reset Commands are used to Reset the Master Timebase Counter, the Object Counters, the Projectile Counters, and the Border Counter. The Commands and their respective counters are as follows:

CRHOR	($03)	Master Timebase Counter
CROBA	($10)	Object A Timebase Counter Reset
CROBB	($11)	Object B Timebase Counter Reset
CRPJA	($12)	Projectile A Timebase Counter Reset
CRPJB	($13)	Projectile B Timebase Counter Reset
CRBDR	($14)	Border Timebase Counter Reset

Each one of these is a Command Level Instruction which is executed only by writing to the Address. There are not any Data Bits associated with these Commands, therefore the Data on the Data Bus during the WRITE Cycle will not affect in any way the execution of these Commands. These Commands can be executed using the STA, STX, and STY Instructions.

When the CRHOR Command is issued, the Master Timebase Counter for the VIC is reset to its starting state which is the beginning of the Horizontal Blanking.

When the CROBA, CROBB, CRPJA, CRPJB, and the CRBDR Commands are executed, the Object being written to will be reset to the left side of the screen if the command is executed during the Horizontal Blanking period.

During the execution of a game program it may be desired to move an Object, Projectile or the Border to a known location. This may be done by using the Horizontal Counter Reset for the Object, which will place the Object in a known location and then move the object to the desired location. The Horizontal Counter Reset Commands should normally be executed during the Horizontal Blanking Period. There are 23 Microprocessor Clock Cycles during the Horizontal Blanking when the Reset Command can be executed.

The following is a sample program in which all of the Objects are initialized to their reset location.

```
STA HSYNC
STA CROBA
STA CROBB
STA CRPJA
STA CRPJB
STA CRBDR
```

Figure 22 shows the location of the Objects and the Border after the Horizontal Counter Reset Commands have been issued to the VIC during the Horizontal Blanking Period.

OBJECT POSITION AFTER HORIZONTAL
COUNTER RESET

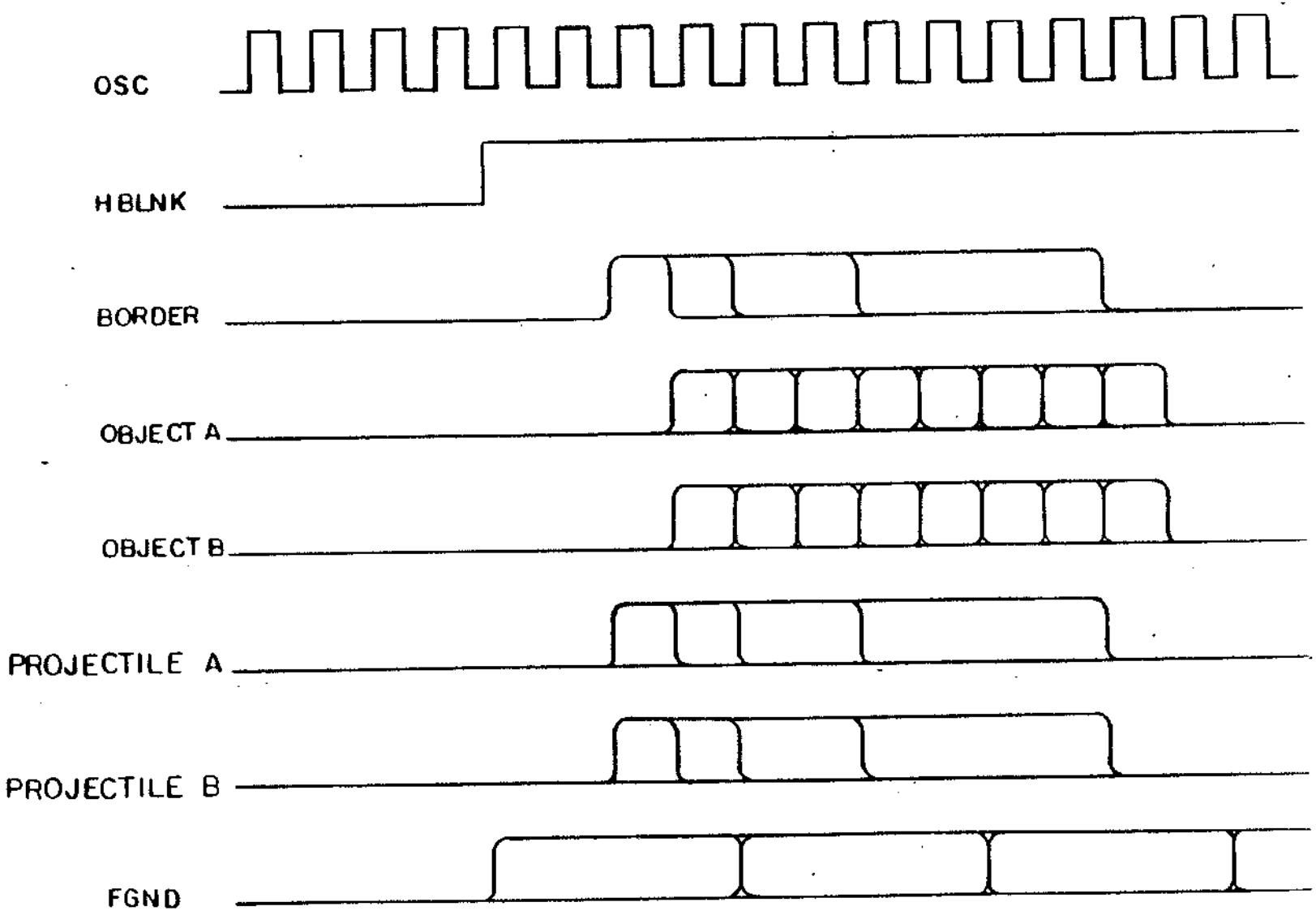


FIGURE 22

OBJECT AND PROJECTILE SIZE AND REPEATS

The OBASR ($04) and OBBSR ($05) registers are used to control the size of each bit of the Objects and the size of the Projectiles. These registers also control the repeat Objects and Projectiles. These registers use DBO, DB1, DB2, DB4, and DB5.

These registers actually perform two separate functions. The first function is the size of the Object and how many times the Object and Projectile will be displayed in one horizontal scan. The second independent function is the horizontal size of the Projectile.

DB0, DB1, and DB2 are used to control the Object size and the number of times the Object and Projectile will be repeated. The Object size is the number of VIC Clock Cycles for each Bit in the Object Font. The width of a Bit can be programmed to be 1, 2, or 4 VIC Clock Cycles wide. When the size of an object is programmed to the 2 or 4 VLC Clock Cycle Mode, the location of the origin of the left most Bit (DB7 or DB0) will remain in exactly the same location as when the 1 VIC Clock Cycle Mode is selected. However, the location of all of the other Bits will be shifted to the right. In addition to controlling the size of an Object, the number of times that an Object and its associated Projectile (Object A and Projectile A, Object B and Projectile B) will be repeated to the right of the main Object or Projectile are also

controlled. An Object and Projectile can be displayed up to 3 times horizontally. The repeat Objects and Projectiles are the same color as the main Objects. The repeat Objects and Projectiles are spaced 16, 32, or 64 VIC Clock Cycles after the beginning of the main Object or Projectile. The location of the repeat Object and Projectile is a function of the size of the main Object. The repeat Objects are the same horizontal size as the main Object.

The following table shows the different combinations of Object size and repeats:

DB2	DB1	DB0	# of VIC Clock Cycles Per Bit	# of Repeat Objects and Projectiles and Location	
0	0	0	1	0	N/A
0	0	1	1	1 repeat	16 apart
0	1	0	1	1 repeat	32 apart
0	1	1	1	2 repeat	16 apart
1	0	0	1	1 repeat	64 apart
1	0	1	2	0	N/A
1	1	0	1	2 repeat	32 apart
1	1	1	4	0	N/A

DB4, and DB5 control the size of the Projectile independently of the size of the Object and the number of repeats of the Objects and Projectiles. The size of the Projectile is the number of VIC

Clock Cycles. The Projectiles can be programmed to be 1, 2, 4, or 8 VIC Clock Cycles wide. Each of the repeat Projectiles will also be the same size as the main Projectile. The left edge of the projectile will remain in exactly the same horizontal location regardless of the size of the Projectile. Only the right edge of the Projectile will move as the size is altered. Figure 22 on page 60 shows the 4 different sizes of the Projectiles.

The following table shows the size and width of the Projectiles in VIC Clock Cycles and NSEC:

DB5	DB4	# of VIC Clock Cycles	Width in NSEC
0	0	1	280
0	1	2	560
1	0	4	1120
1	1	8	2240

COLOR CONTROL REGISTERS

The Color Control Registers are used to select the Color of each of the Objects, the Projectiles, the Border, the Foreground and the Backgournd. These Control Registers use DB1 thru DB7. The Registers are as follows:

CLROA	($06)	Object A and Projectile A Color Control Register
CLROB	($07)	Object B and Projectile B Color Control Register
CLRFB	($08)	Foreground and Border Color Control Register
CLRBG	($09)	Background Color Control Register

The Color Control Registers use DB1 thru DB7 to Control the Color. DB1, DB2, and DB3 are used to control the Color Intensity and DB4, DB5, DB6, and DB7 are used to control the Color Phase. There are 128 color combinations.

The color of an Object does not affect any of the Coincidence Detection or Priority Circuits in the VIC. The actual colors displayed on the video screen when two Objects overlap will be the color of the Object with the higher priority. An Object with a Black color will still have coincidence detected with other Objects when they overlap each other.

Figure 23 shows the phases selected by the VIC for each of the 15 colors and the Color Burst Reference. The phases used to generate each color can be looked up in the color table below.

The following two tables show how each Object can be programmed to the 128 different Colors by combining the high Data Bits (Hue) and the low Data Bits (Luminance):

DATA	HUE	APPROXIMATE PHASE (in degrees)
$0X	GREY	NONE
$1X	YELLOW-ORANGE	0
$2X	ORANGE	24
$3X	RED-ORANGE	48
$4X	RED	72
$5X	PINK	96
$6X	MAGENTA	120
$7X	PURPLE	144
$8X	RED-BLUE	168
$9X	BLUE	192
$AX	BLUE	216
$BX	LIGHT BLUE	240
$CX	CYAN	264
$DX	BLUE-GREEN	288
$EX	GREEN	312
$FX	YELLOW-GREEN	336
	COLOR BURST	0

DATA	LUMINANCE LEVEL
$X0 or $X1	13%
$X2 or $X3	25%
$X4 or $X5	38%
$X6 or $X7	50%
$X8 or $X9	63%
$XA or $XB	75%
$XE or $XF	100%

The data in these two tables is combined together to generate the 128 possible colors.

COLOR PHASE OUTPUT

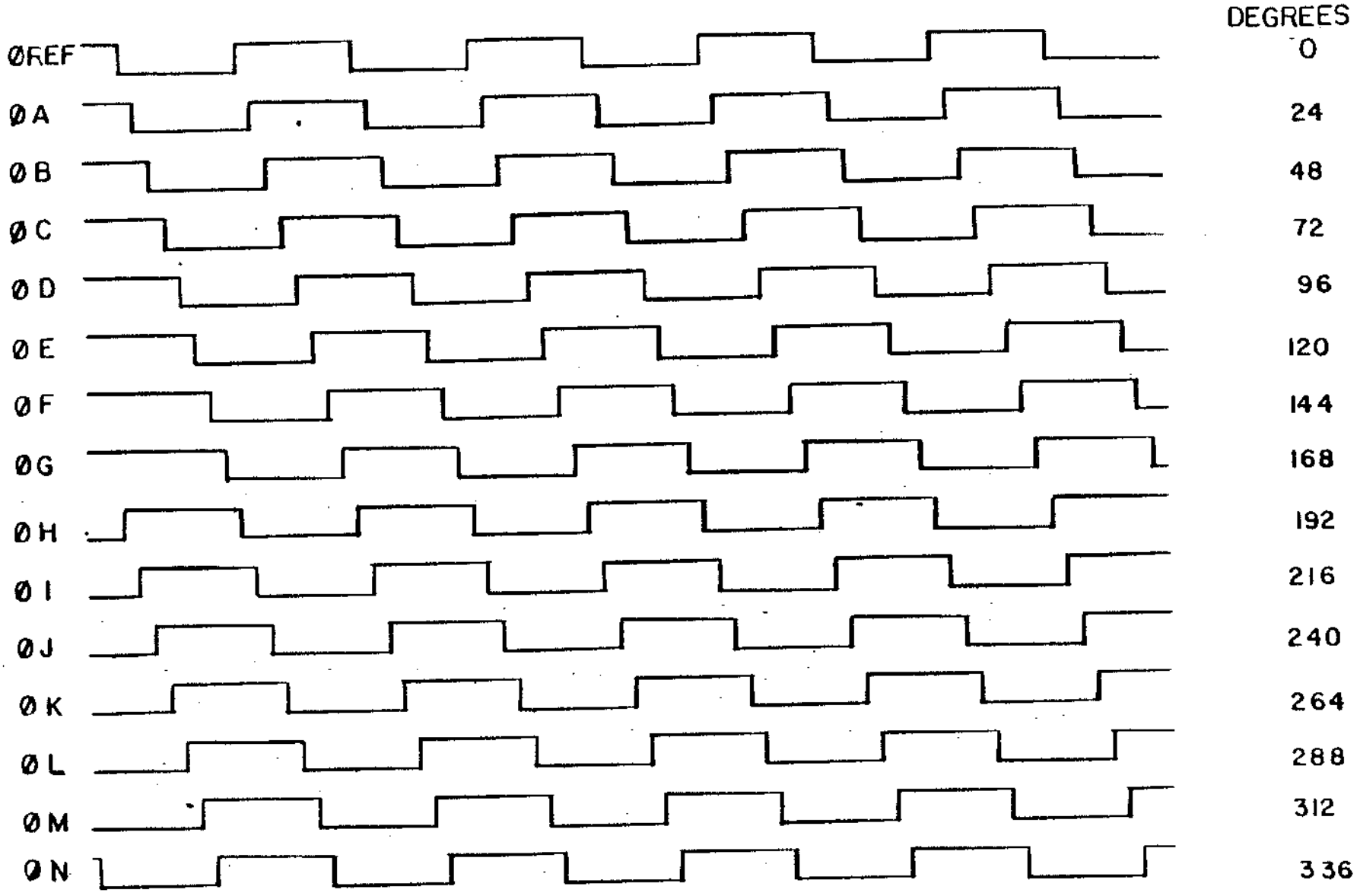


FIGURE 23

BORDER SIZE AND FOREGROUND CONTROL REGISTERS

The BDRSZ ($0A) Control Register is a multiple function Register. It controls the horizontal size of the Border as well as the Scan Control for the Foreground, the Color format for the Foreground, and the Priority of the Border and Foreground in relation to the Objects and Projectiles. This Control Register uses DB0, DB1, DB2, DB4, and DB5.

DB0 and DB1 are used to control the Foreground Display Mode. There are 4 different display modes. DB0 controls the scan sequence of the Foreground. DB1 controls the Color of the Foreground during the scan sequence. The Foreground is a 20 bit register (see Section 3.9 FOREGROUND FONT REGISTERS) that is scanned 2 times per horizontal line. Each Bit is 4 VIC Clock Cycles wide. The first scan of the Foreground is fixed and cannot be altered. The second scan is controlled by DB0. When DB0 is programmed to a "0", the second scan of the Foreground register is exactly the same as the first scan. When DB0 is programmed to a "1" the second scan of the Foreground is reversed. That is, the last Bit displayed during the first scan will be the first Bit displayed during the second scan and the first Bit displayed during the first scan will be the last Bit displayed during the second scan creating a mirror image of the first scan.

When DB1 is programmed to a "0", the color of the Foreground will be as programmed by the CLRFB ($08) Register for the entire scan. When DB1 is programmed to a "1", the color of the first scan (left half of the screen) will be the color of Object A and Projectile A, CLROA ($06). The color of the second scan of the Foreground Font Register (the right half of the screen), will be the color of Object B and Projectile B, CLROB ($07). These 4 modes of display of the Foreground are shown in figure 24.

DB2 controls the Priority of the Border and Foreground in relation to the Objects and Projectiles. There are 2 different Priority modes of display. The normal priority, DB2 programmed to a "0", is Object A and Projectile A, Object B and Projectile B, Border and Foreground, then the Background Color. The priority relationship between the Objects and Projectiles cannot be altered, however, the priority relationship of the Border and Foreground can be altered to the Objects and Projectiles. When DB2 is programmed to a "1", the second Priority mode is selected. In the second mode the Priorities are: Border and Foreground, Object A and Projectile A, Object B and Projectile B, then the Background Color.

DB4 and DB5 control the Border Size. The size of the Border is the number of VIC Clock Cycles. The Border can be programmed to be 1, 2, 4, or 8 VIC Clock Cycles wide. The left edge of the

Border will remain in exactly the same location regardless of the size of the Border. Only the right edge of the Border will move as the size is altered. Figure 22 on page 60 shows the 4 different sizes of the Border.

The following table shows the size and width of the Border in VIC Clock Cycles on NSEC:

DB5	DB4	# of VIC Clock Cycles	Width in NSEC
0	0	1	280
0	1	2	560
1	0	4	1120
1	1	8	2240

FOREGROUND DISPLAY

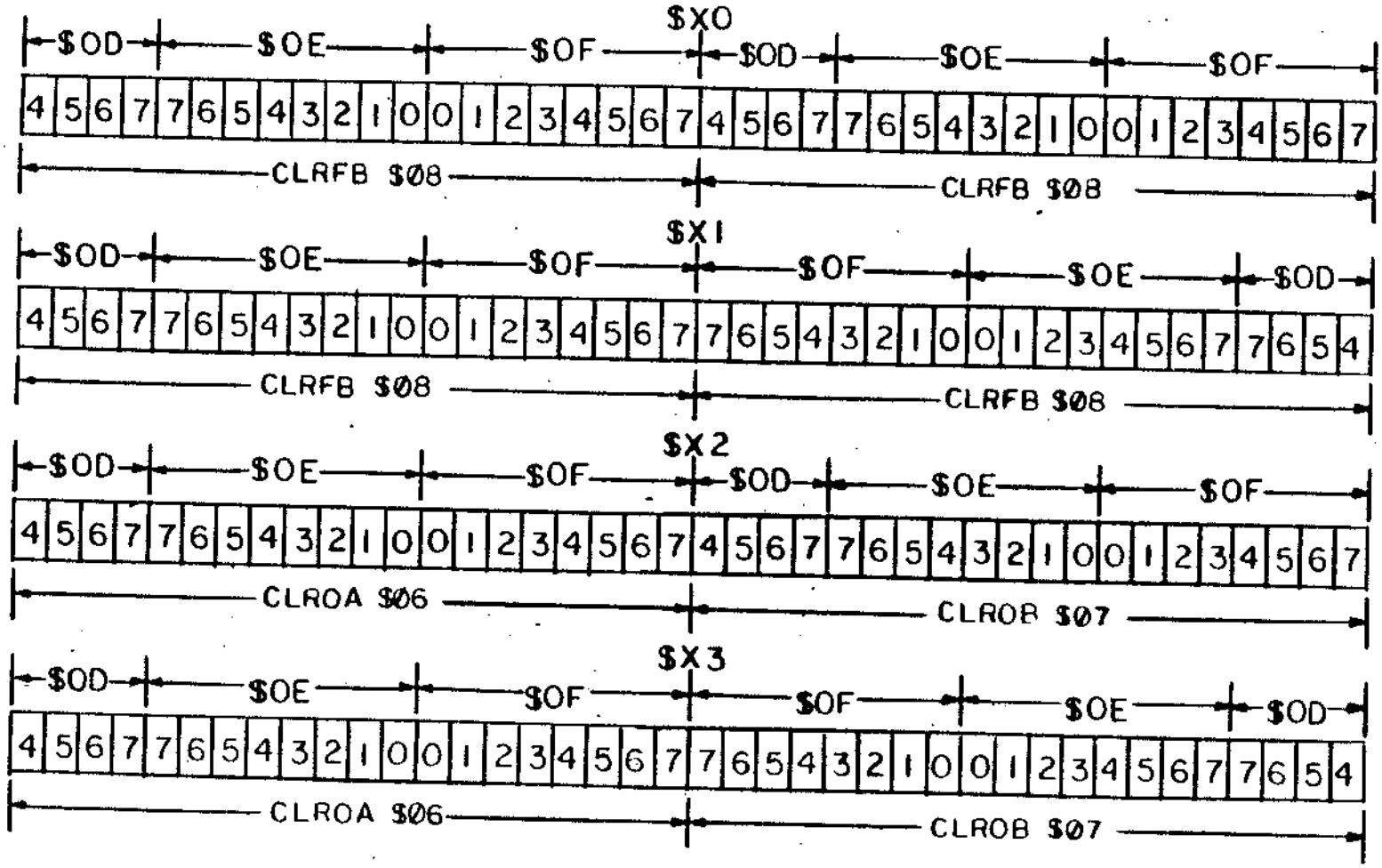


FIGURE 24

REV A 4/27/81

-71-

OBJECT SCAN DIRECTION CONTROL REGISTERS

The Object Scan Direction Control Registers are used to control the order that the Data in the Object Font Registers is displayed. The Registers use DB3. The Registers are as follows:

OBADR	($0B)	Object A Scan Direction
OBBDR	($0C)	Object B Scan Direction

DB3 Controls which direction the Object Font is scanned for the Video Display. When DB3 is a "0", the Font Data is scanned from left to right (DB7 to DB0). When DB3 is a "1", the Font Data is scanned from right to left (DB0 to DB7).

The following table shows the action of DB3 for the OBADR and OBBDR Control Registers:

DB3	FONT SCAN DIRECTION
0	Scan Sequence is DB7-DB6-DB5-DB4-DB3-DB2-DB1-DB0
1	Scan Sequence is DB0-DB1-DB2-DB3-DB4-DB5-DB6-DB7

FOREGROUND FONT REGISTERS

The Foreground Font Registers are used to store the Display Font Data for the Foreground. The Font Registers are as follows:

FGNDA	($0D)	Foreground A Font Register
FGNDB	($0E)	Foreground B Font Register
FGNDC	($0F)	Foreground C Font Register

The Foreground Font Register FGNDA uses DB4 thru DB7 and Foreground Font Register FGNDB and FGNDC use DB0 thru DB7 to store the Foreground Font Data. Each Font Data Bit is four VIC Clock Cycles wide when it is displayed.

The Foreground appears to be a RAM Memory Block of 20 Bits that can be displayed in one of four different Horizontal Display Modes. The active Foreground area is 160 VIC Clock Cycles wide. Each Data Bit in the Foreground is 4 VIC Clock Cycles Wide regardless of the Scan Mode selected. The Foreground Registers are displayed 2 times each Horizontal Scan.

The Foreground is two and a half Bytes of RAM (20 Bits). The Foreground Display is a combination of displaying the Foreground Register two times. The four methods of displaying the Foreground are controlled by the BDRSZ ($0A) Control Register. The programmer

should read the BDRSZ description for a complete explanation of the Foreground display methods. Figure 24 on page 71 shows the location of each of the Data Bits of the Foreground. Figure 24 also shows all four of the different modes of Foreground display.

SOUND SELECTION REGISTERS

The SNDAS ($15) and SNDBS ($16) Sound Selection Registers are used to select the Waveform shape for Sound Generator A and Sound Generator B. The Selection Registers use DB0, DB1, DB2, and DB3.

There are 10 different Sound Waveforms and an OFF condition that can be selected by the Sound Selection Registers. Of the 16 possible Sound Selections, 2 are the OFF condition and 4 are duplicates of the 10 different Sound Waveforms.

The 10 Sound Waveforms range from a simple Square Wave to a complex Waveform. The Waveforms are composed of from 1 to 12 different frequency components. The Sounds generated by the Waveforms range from a simple Tone to White Noise.

The Frequence at which the waveform will begin to repeat the wave shape is referred to as the Divider Ratio. The number of different Frequencies in a Waveform is referred to as the Frequency Spectrum.

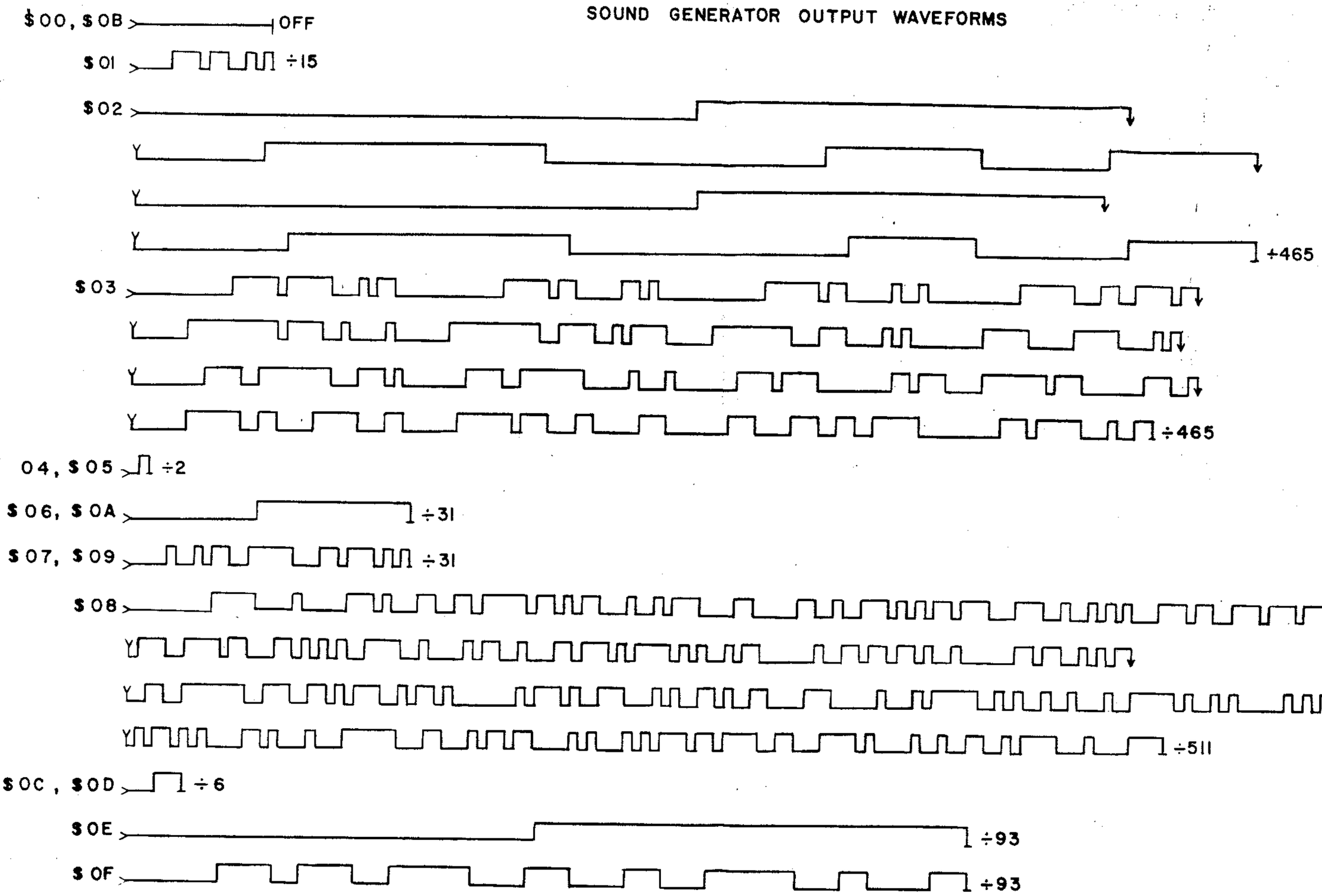
The actual Frequency at which the VIC Sound Waveform will begin to repeat is a function of the Sound Waveform selected and the Frequency selected for the Sound Waveform. This Frequency can be calculated by looking up the frequency in the table in the Sound

Frequence description and divide that frequency by the Divider Ratio and then multiply by 2. The following table shows the Divider Ratio and the Frequency Spectrum of each of the Sound Waveforms:

DATA	DIVIDER RATIO	FREQUENCY SPECTRUM
$X0	OFF	0
$X1	15	4
$X2	465	5
$X3	465	12
$X4	2	1
$X5	2	1
$X6	31	2
$X7	31	5
$X8	511	9
$X9	31	5
$XA	31	2
$XB	OFF	0
$XC	6	1
$XD	6	1
$XE	93	2
$XF	93	7

Figure 25 shows the actual Wave Shape of the 10 different Waveforms generated by the Sound Generator Circuit in the VIC.

SOUND GENERATOR OUTPUT WAVEFORMS



REV. B
8/14/81

FIGURE 25

SOUND FREQUENCY SELECTION REGISTERS

The FREQA ($17) and FREQB ($18) Sound Frequency Selection Registers are used to select the Clock Frequency for Sound Generator A and Sound Generator B, respectively. The Selection Registers use DB0, DB1, DB2, DB3, and DB4.

There are 32 different frequencies that can be programmed into the Sound Clock Generator Circuit. The Sound Frequency Selection Register is used to select the 32 frequencies. The 32 frequencies are the Horizontal Sweep Frequency doubled and then divided by an integer from 1 to 32. The Horizontal Sweep Frequency is 15,700 Hz.

The Integer Divider Factor for the Sound Frequency Generator can be calculated by converting the Hexadecimal Data stored in the Frequency Selection Registers to Decimal and adding 1 to the Decimal value. The Hexadecimal value must be in the range of $00 to $1F. For example, $00 is a Divider Factor of 1, $10 is a Divider Factor of 17, and $1F is a Divider Factor of 32.

The following tables show all of the possible Frequencies that can be programmed into the VIC Sound Frequency Selection Registers. The Frequencies listed below are the actual Output Frequency with the Sound Generator programmed to the highest possible frequency ($04, a divided by 2 Tone). The following

table lists the Hexadecimal Data programmed into the Sound Frequency Selection Register and the Sound Output Frequencies (in Hertz):

DATA	DIVIDER	FREQUENCY	DATA	DIVIDER	FREQUENCY
$00	1	15699.76	$10	17	923.52
$01	2	7849.88	$11	18	872.21
$02	3	5233.25	$12	19	826.30
$03	4	3924.94	$13	20	784.99
$04	5	3139.95	$14	21	747.61
$05	6	2616.63	$15	22	713.63
$06	7	2242.82	$16	23	682.60
$07	8	1962.47	$17	24	654.16
$08	9	1744.42	$18	25	627.99
$09	10	1569.98	$19	26	603.84
$0A	11	1427.25	$1A	27	581.47
$0B	12	1308.31	$1B	28	560.71
$0C	13	1207.67	$1C	29	541.37
$0D	14	1121.41	$1D	30	523.33
$0E	15	1046.65	$1E	31	506.44
$0F	16	981.23	$1F	32	490.62

SOUND OUTPUT LEVEL REGISTERS

The LVLSA ($19) and LVLSB ($1A) Sound Output Level Control Registers are used to Control the amplitude of the Audio Outputs of the Sound Generator Circuits. These Registers use DB0, DB1, DB2, and DB3.

There are two Audio Output Pins on the VIC. Sound Generator A is connected to Pin 13 and Sound Generator B is connected to Pin 12. Each of the Sound Outputs can be programmed independently of each other to 15 different Output Levels. When both of the Sound Outputs are programmed ON at the same time, up to 30 different Output Levels can be generated. The two Sound Output Pins are connected together so that the two Sounds are mixed together into one Output signal.

The Outputs from the Sound Generators act like a Binary weighted Current Source to V-. The Outputs are ratio'd 1:2:4:8. DB0 controls the lowest amplitude device and DB3 controls the highest amplitude device.

When both Sound Generators are ON at the same time the sounds being created will be mixed together so that more than one sound can be made at the same time. The Sound Generators can also be used as a type of envelope control on each other.

The following table shows the relative Output Current Levels for each of the Sound Outputs:

DATA	OUTPUT LEVEL
$X0	0.0%
$X1	6.7%
$X2	13.3%
$X3	20.0%
$X4	26.7%
$X5	33.3%
$X6	40.0%
$X7	46.7%
$X8	53.3%
$X9	60.0%
$XA	66.7%
$XB	73.3%
$XC	80.0%
$XD	86.7%
$XE	93.3%
$XF	100.0%

OBJECT FONT REGISTERS

The Object Font Registers are used to store the Object Font Data for the VIC. The Object Font Registers are as follows:

OBJAF	($1B)	Object A Font Register
OBJBF	($1C)	Object B Font Register

The Object Font Registers use DB0 thru DB7 to store the Object Font Data. There are two Font Data Registers at each Object Address. One of the Object Font Data Registers is the On-Line Register and the second Object Font Data Register is the Delayed Register. The programmer should read the Object Scan Direction and Register Select description for a complete explanation of the use of the On-Line Register and the Delayed Register.

The programmer should also read the following descriptions of Control Registers that control the display of the Object Font Data:

Object and Projectile Size and Repeats

Object Scan Direction Control Registers

Object Font Register Selects

Projectile Tracking and Disable Registers

PROJECTILE ENABLE CONTROL REGISTERS

The PJAEB ($1D) and PJBEB ($1E) Control Registers are used to Enable each of the Projectiles A and B. These Control Registers use DB1.

PJAEB and PJBEB allows the programmer to use a single Address location to Enable a Projectile.

PJAEB controls one of the Enable Functions for Projectile A. The other Enable is controlled by the Projectile A Tracking Register PJATK ($28).

PJBEB controls one of the Enable Functions for Projectile B. The other Enable is controlled by the Projectile B Tracking Register PJBTK ($29).

DB1 controls the Enable Function. When DB1 is programmed to a "1", the Projectile is Enabled if the Projectile Tracking Function PJATK or PJBTK ($28 or $29) is programmed to a "0". When DB1 is programmed to a "0", the Projectile is Disabled.

BORDER ENABLE REGISTERS

The BDREB ($1F) Border Enable Registers are used to enable the Border. The Border Enable Registers use DB1.

There are two Border Enable Registers. One of the Border Enable Registers is the On-Line Enable Register and the other Border Enable Register is the Delayed Enable Register. The On-Line Enable Register is connected directly to the VIC Internal Data Bus as are most of the Data Registers in the VIC. The Delayed Enable Register is delayed by a WRITE Cycle to OBJBF ($1C) (Object B Font Register).

When the On-Line Border Enable Register is selected and the Data in the On-Line Register is a "0", the Border will be Disabled. When the data in the On-Line Register is a "1" the Border will be Enabled.

When the Delayed Border Enable Register is selected and the Data in the Delayed Register is a "0", the Border will be Disabled. When the Data in the Delayed Register is a "1", the Border will be Enabled.

The following table shows the movement of Data thru the Border Enable Registers:

DATA WRITE	ON-LINE REGISTER	DELAYED REGISTER
- - - -	0	0
BDREB ($1F)	1	0
OBJBF ($1C)	1	1
BDREB ($1F)	0	1
OBJBF ($1C)	0	0
BDREB ($1F)	1	0
BDREB ($1F)	0	0

The following table shows the Border Enable Registers and when the Border is Enabled:

BORDER	BDRES ($27) REGISTER	ON-LINE REGISTER	DELAYED REGISTER
OFF	0	0	0
OFF	0	0	1
ON	0	1	0
ON	0	1	1
OFF	1	0	0
ON	1	0	1
OFF	1	1	0
ON	1	1	1

HORIZONTAL MOVEMENT CONTROL REGISTERS

The Horizontal Movement Control Registers are used to select the amount of Horizontal Movement for the Objects A, and B, Projectiles A, and B and the Border. These Control Registers use DB4 thru DB7. The Registers are as follows:

HMOBA	($20)	Object A Horizontal Movement Control Register
HMOBB	($21)	Object B Horizontal Movement Control Register
HMPJA	($22)	Projectile A Horizontal Movement Control Register
HMPJB	($23)	Projectile B Horizontal Movement Control Register
HMBDR	($24)	Border Horizontal Movement Control Register

The Horizontal Movement Control Registers use DB4, DB5, DB6 and DB7 to control the Horizontal Movement. DB7 controls the direction of movement and DB4, DB5 and DB6 control the magnitude of the movement. The movement range is -7 to +8 VIC Clock Cycles.

The Horizontal Movement of an Object, Projectile, or the Border is performed by writing the amount of movement desired to the appropriate Movement Control Register and then executing the HMENB Command. The Horizontal Movement of the Objects, Projectiles, or the Border can be set to 0 by writing the Data "$0X" to the Movement Control Register. The five Horizontal Movement Registers can all be reset to 0 movement by executing the HMOVR Command.

When the Movement Register is set to 0 and the HMENB Command is executed, the Object will remain stationary. When the Movement Register is set to -1 and the HMENB Command is executed, the origin of the Object will be moved one VIC Clock Cycle to the left one the screen. When the Movement Register is set to +1 and the HMENB Command is executed, the origin of the Object will be moved on VIC Clock Cycle towards the right on the screen.

The Object Size does not affect the operation of the Horizontal Movement Circuit. The number of VIC Clock Cycles that an Object must move to get from one side of the screen to the other side of the screen is 160.

When an Object is moved to the far left or right side of the screen and then is moved beyond the edge, the Object will begin to wrap around the screen. Wrap around occurs as the Object moves across the Horizontal Blanking. When wrap around occurs, the left side of the Object will be displayed on the right side of the screen and the right side of the Object will be displayed on the left side of the screen.

Horizontal wrap around can be eliminated by setting the Border to be 8 VIC Clock Cycles wide and setting its color to the same color as the Background, thus making it invisible. The Border is then set to a higher priority than the Objects and

Projectiles that are to be prevented from wraping around and placed with 4 VIC Clock Cycles on each side of the screen. When an Object or Projectile comes into coincidence with the Border, it will disappear behind the Border and coincidence can be checked between the Object or the Projectile and the Border and the Object or Projectile can be prevented from wrapping around to the other side.

The direction of horizontal movement of an Object, Projectile, or the Border can be reversed at the same rate of movement by taking the Two's-Complement of the Data in the Horizontal Movement Control Register. The following is an example of a program that could be used to reverse the horizontal movement of an Object, Projectile, or the Border:

```
LDA MOTION          ; LOAD OLD MOVEMENT RATE
AND #$FO            ; REMOVE LOW NIBBLE
EOR #$FO            ; INVERT HIGH NIBBLE
CLC                 ; CLEAR CARRY
ADC #$10            ; ADD 1
STA HMXXX           ; STORE NEW MOVEMENT RATE
```

The following table shows the relative movement of an Object, Projectile or the Border from the original position in VIC Clock Cycles:

	DB7	DB6	DB5	DB4	MOVEMENT
$0X	0	0	0	0	0
$1X	0	0	0	1	-1
$2X	0	0	1	0	-2
$3X	0	0	1	1	-3
$4X	0	1	0	0	-4
$5X	0	1	0	1	-5
$6X	0	1	1	0	-6
$7X	0	1	1	1	-7
$8X	1	0	0	0	+8
$9X	1	0	0	1	+7
$AX	1	0	1	0	+6
$BX	1	0	1	1	+5
$CX	1	1	0	0	+4
$DX	1	1	0	1	+3
$EX	1	1	1	0	+2
$FX	1	1	1	1	+1

OBJECT FONT REGISTER SELECT

The Object Font Register Select Control Registers are used to control which of the two Object Font Registers are to be displayed. The Registers use DB0. The Registers are as follows:

OBARS	($25)	Object A Font Register Select
OBBRS	($26)	Object B Font Register Select

DBO selects the Object Font Register to be displayed. When DB0 is a "0", the On-Line Register is selected. When DBO is a "1", the Delayed Register is Selected. The On-Line Register is connected directly to the VIC internal Data Bus as are most of the Data Registers in the VIC. The Delayed Register is delayed by a WRITE Cycle to the other Object Font Register.

The Object Font Registers are paired together. When Object Font Data is written to Object A, the Font Data stored in the On-Line Register of Object B will be transferred to the Delayed Register of Object B. When Object Font Data is written to Object B, the Font Data stored in the On-Line Register of Object A will be transferred to the Delayed Register of Object A.

The following table shows the action of DB0 for the OBARS and OBBRS Control Registers:

DBO	REGISTER SELECT
0	Selects ON-LINE Register
1	Selects DELAYED Register

The following table shows the movement of Font Data through the Registers:

OBJECT WRITE	ON-LINE REGISTER A	DELAYED REGISTER A	ON-LINE REGISTER B	DELAYED REGISTER B
OBJAF	FONT 1	- - -	- - -	- - -
OBJBF	FONT 1	FONT 1	FONT 2	- - -
OBJAF	FONT 3	FONT 1	FONT 2	FONT 2
OBJBF	FONT 3	FONT 3	FONT 4	FONT 2
OBJAF	FONT 5	FONT 3	FONT 4	FONT 4

BORDER ENABLE SELECT REGISTER

The BDRES ($27) Border Enable Select Register is used to select one of two Enable Registers for the Border. The Register uses DB0.

There are two Enable Registers for the Border. One of the Enable Registers is the On-Line Register and the other Enable Register is the Delayed Register. When DB0 is "0", the On-Line Enable Register will Enable the Border. When DB0 is "1", the Delayed Enable Register will Enable the Border.

The On-Line Enable Register is used to Enable or Disable the Border directly. The Delayed Enable Register can be used to Enable or Disable the border indirectly while Writing Object Font Data to Object B. This allows the programmer to set up the Enable Data in the On-Line Register and then transfer the actual Border Enable when the next WRITE Cycle to OBJBF ($1C) is executed. The Programmer is directed to read the BDREB ($1F) (Border Enable Registers) description for more information concerning this Register (BDRES).

PROJECTILE A AND B TRACKING AND DISABLE REGISTERS

The Projectile Tracking and Disable Control Registers are used to Disable each of the Projectiles and to control the Tracking Function for each of the Projectiles. These Control Registers use DB1.

PJATK	($28)	Projectile A Tracking and Disable Register
PJBTK	($29)	Projectile B Tracking and Disable Register

PJATK and PJBTK allows the programmer to use a single Address location to Disable a Projectile and to Enable the Tracking Function.

PJATK controls one of the Disable Functions for Projectile A and it Enables the Tracking Function. The Tracking function allows the Programmer to force Projectile A to track Object A horizontally regardless of Projectile A's previous position. Projectile A cannot be moved horizontally away from Object A until the Tracking Function has been turned OFF. When the Tracking Function is Enabled, Projectile A is Disabled and will follow Object A if it is moved.

PJBTK controls one of the Disable Functions for Projectile B and it Enables the Tracking Function. The Tracking Function allows the programmer to force Projectile B to track Object B horizontally regardless of Projectile B's previous position. Projectile B cannot be moved horizontally away from Object B until

the Tracking Function has been turned OFF. When the Tracking Function is Enabled, Projectile B is Disabled and will follow Object B if it is moved.

Figures 26 thru 28 show the position relationship between Object A and Projectile A and the relationship between Object B and Projectile B when the Tracking Function is Enabled for each of the different sizes of Object A and Object B.

DB1 controls the Tracking Function. When DB1 is progammed to a "0", the Tracking Function is OFF and if the Projectile is Enabled (see PJAEB and PJBEB) it will be Enabled. When DB1 is programmed to a "1", the Tracking Function is turned ON and the Projectile will be Disabled.

OBJECT AND
PROJECTILE RELATIONSHIP WITH
TRACKING FUNCTION ENABLED
1X OBJECT SIZE

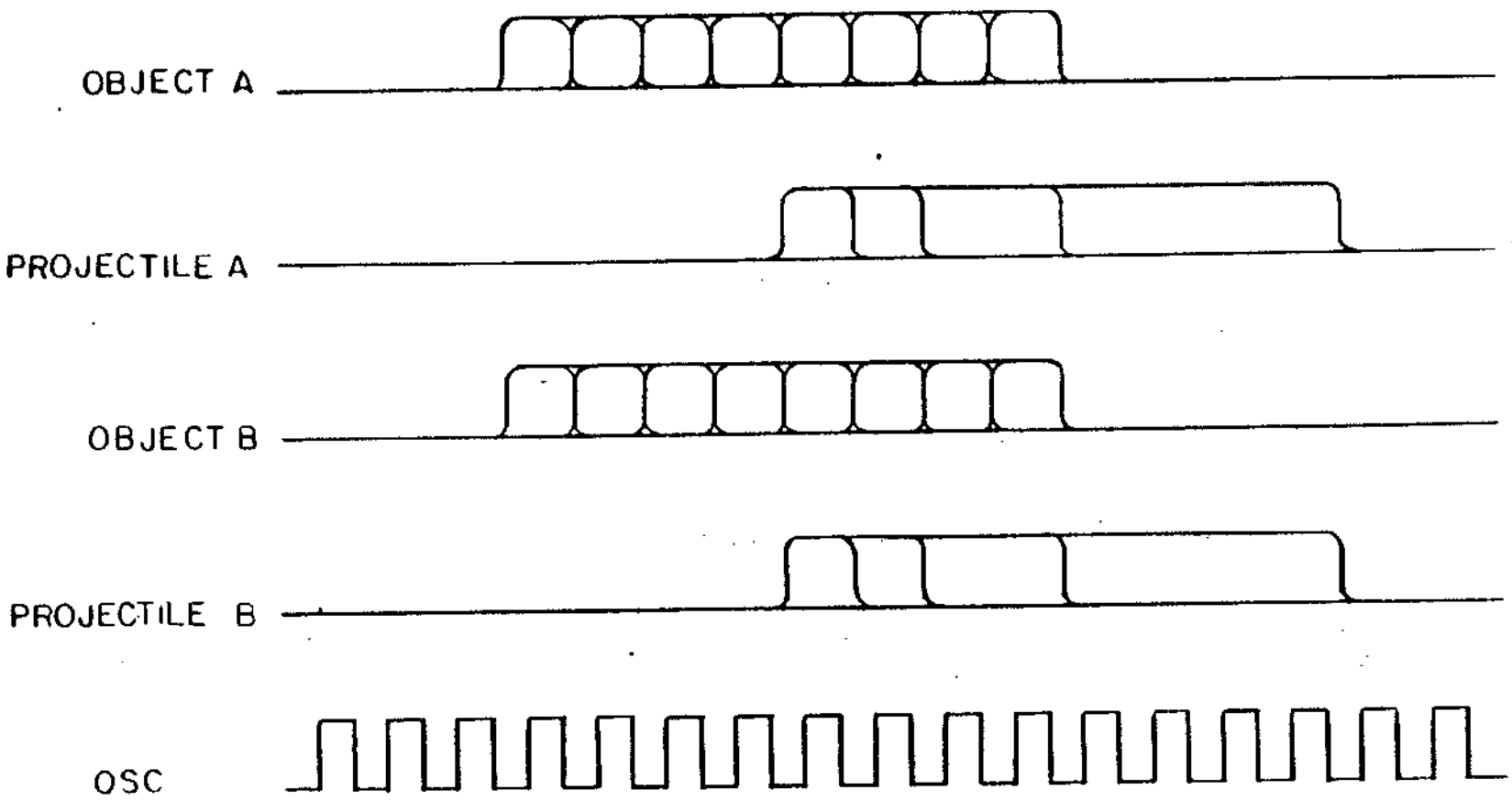


FIGURE 26

OBJECT AND
PROJECTILE RELATIONSHIP WITH
TRACKING FUNCTION ENABLED
2X OBJECT SIZE

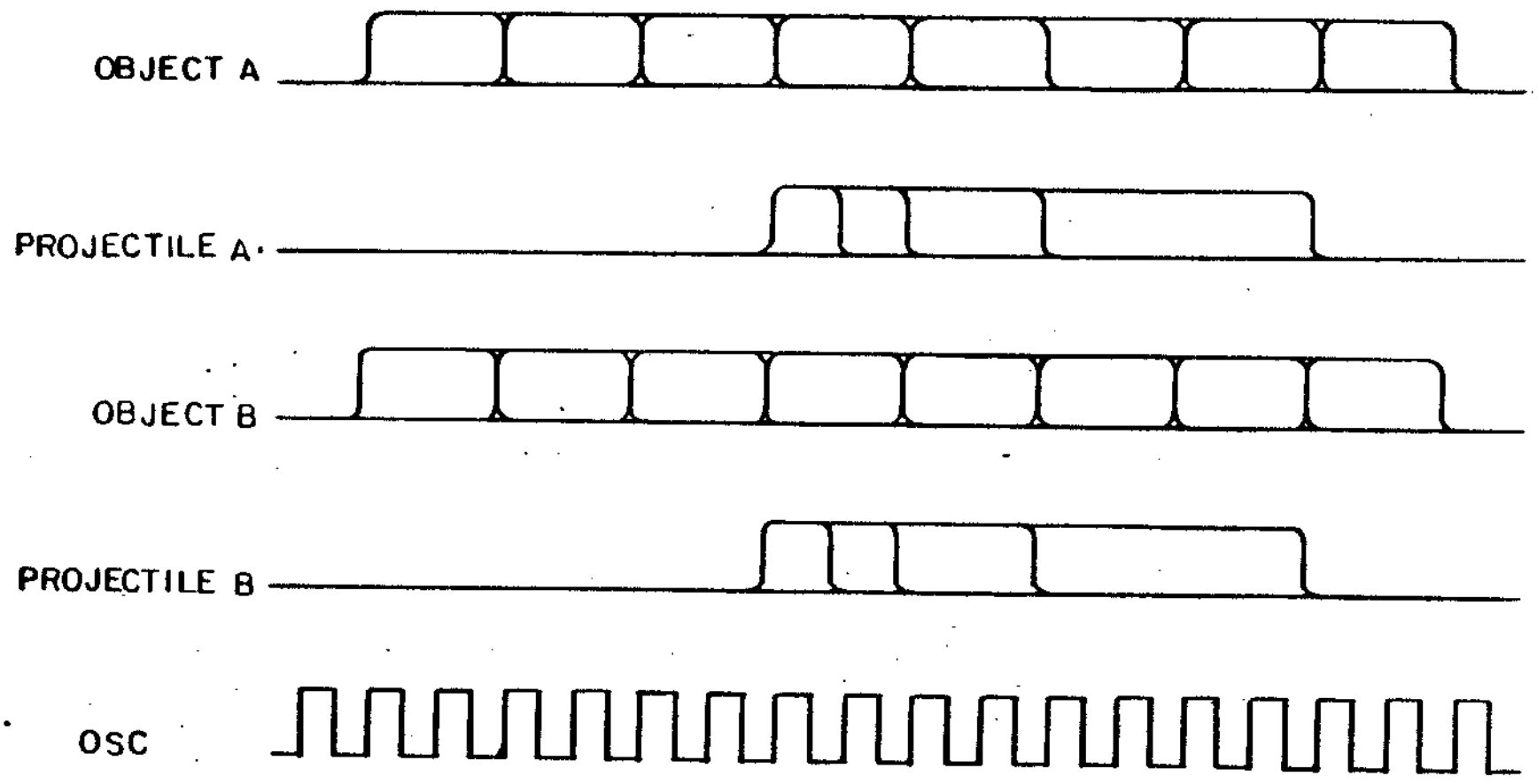


FIGURE 27

REV A
7/30/81

-96-

OBJECT AND
PROJECTILE RELATIONSHIP WITH
TRACKING FUNCTION ENABLED
4X OBJECT SIZE

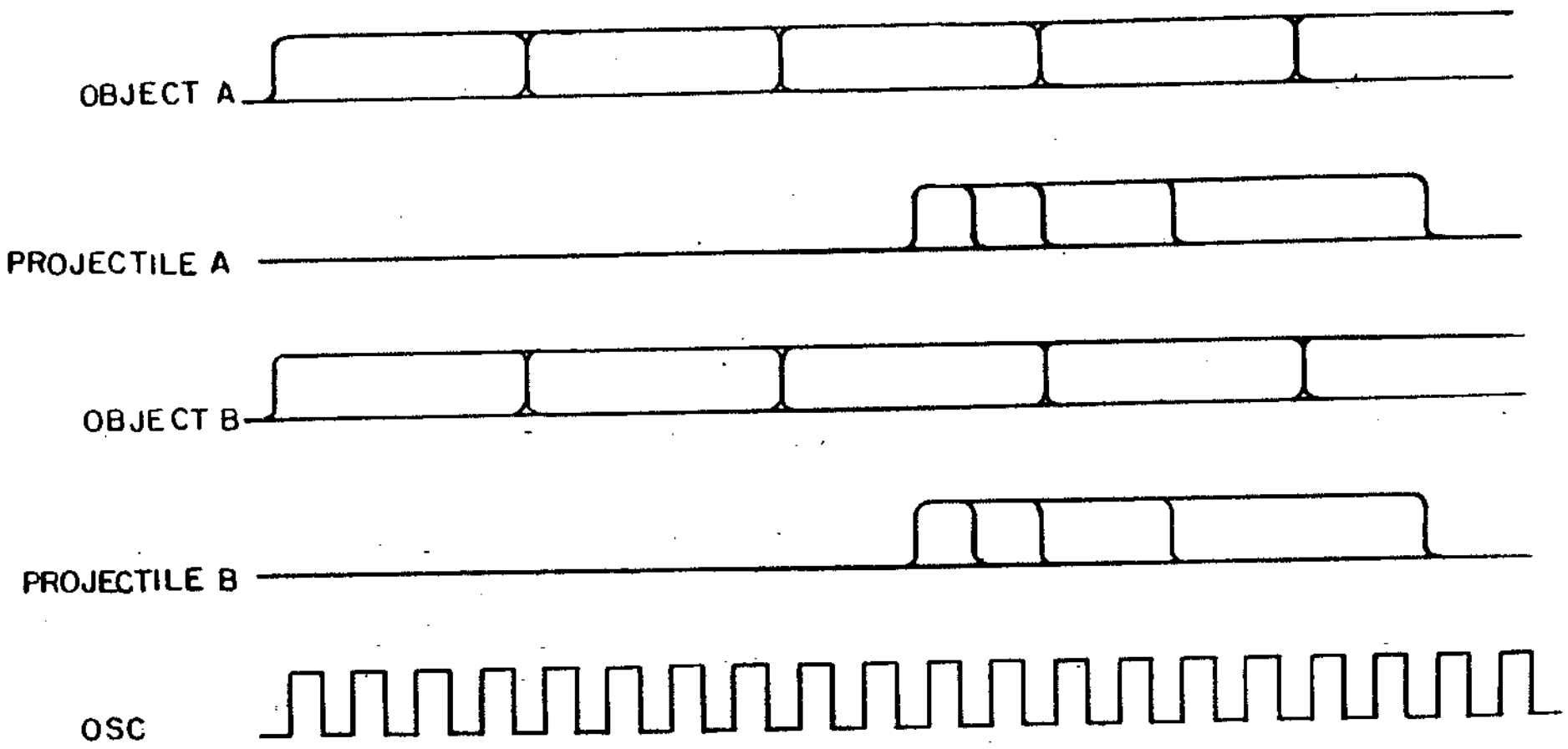


FIGURE 28

REV A
7/30/80

-97-

HORIZONTAL MOVEMENT ENABLE

The HMENB ($2A) Command is used to enable the movement of Object A, Object B, Projectile A, Projectile B, and the Border. This is a Command Level Instruction which is executed only by writing to the Address. There are not any Data Bits associated with this Command, therefore, the Data on the Data Bus during the WRITE Cycle will not affect in any way the execution of this Command. HMENB can be executed using the STA, STX, and STY Instructions.

HMENB is a scan oriented Command which must be executed immediately following a HSYNC Command. Due to the nature of this Command, it must be executed during the Horizontal Blanking period and is affected by the Microprocessor Clock (Ö2). Twenty-six (26) Microprocessor Clock Cycles must pass before the HMENB Command has finished executing.

Immediately after the HMENB Command has been issued to the VIC, there are certain Commands and Registers that must not be executed or written to. These include HMENB, CRHOR, CRBDR, CROBA, CROBB, CRPJA, CRPJB, HMRST, HMBDR, HMOBA, HMOBB, HMPJA, and HMPJB. If any of these are executed or written to, the Movement Command will not be executed correctly, and the Objects may move to a different location than expected or desired. After the

Horizontal Blanking has been turned off, normal operation is resumed and we may utilize these commands and registers normally again.

When the HMENB Command is executed, the Horizontal Blanking Period is extended for 8 OSCILLATOR clock periods.

HORIZONTAL MOVEMENT REGISTERS RESET

The HMRST ($2B) Command is used to reset the Horizontal Movement Registers in the VIC. This is a Command Level Instruction which is executed only by writing to the Address. There are not any Data Bits associated with this Command, therefore the Data on the Data Bus during the WRITE Cycle will not affect in any way the execution of this Command. HMRST can be executed using the STA, STX, and STY Instructions.

This Command resets all 5 Horizontal Movement Registers in the Movement Circuit. When these Registers are reset the motion for Object A, Object B, Projectile A, Projectile B, and the Border is set to 0 motion. This one Command is the equivalent of writing the Data "$0X" to HMOBA ($20), HMOBB ($21), HMPJA ($22), HMPJB ($23), and HMBDR ($24).

The Horizontal Movement Registers will remain in the reset condition ("0") until Movement Data is written into the Movement Registers. This Command can be used when movement of one or more Objects is to be executed and the other objects are to remain stationary. By writing to HMOVR then writing the motion Data only to the Objects that are to be moved several Microprocessor Clock Cycles and ROM Bytes can be saved in the program.

OBJECT COINCIDENCE REGISTER RESET

The OCRST ($2C) Command is used to reset the Coincidence Detection Registers in the Read Logic Section of the VIC. This is a Command Level Instruction which is executed only by writing to the Address. There are not any Data Bits associated with this Command, therefore the Data on the Data Bus during the WRITE Cycle will not affect in any way the execution of this Command. OCRST can be executed using the STA, STX, and STY Instuctions.

This Command resets all 15 Coincidence Detection Registers in the Read Logic Circuit. When these Registers are reset and then read by the Microprocessor the Data at the Output of the VIC will be a "0". When Coincidence for a Register is detected the Register will be set. After the Register has been set and is read by the Microprocessor the Data at the Output of the VIC will be a "1".

The Coincidence Detection Registers will remain in the reset conditions ("0") until Coincidence for a Register is detected. Once Coincidence for a Register has been detected and the Register is set ("1") it will retain the Coincidence Detection indefinitely or until the OCRST Command is executed. The only method that can be used to reset the Coincidence Detection Registers is to execute the OCRST Command.

The Coincidence Detection Registers are normally read once per frame during the Vertical Blanking period using the BIT Instruction. After having read the Register and immediately prior to turning the Vertical Blanking OFF the OCRST Command is executed. The OCRST Command can be executed at any time, however, it erases any Coincidence Detection during or prior to the execution of the OCRST Command.

COINCIDENCE AND INPUT REGISTERS

The Coincidence Registers are used to store the results of the comparison of the Object Coincidence Detection Circut. The Input Registers are used to store and transmit the status of the Potentiometer and Schmitt Trigger Inputs. These Addresses are READ by the Microprocessor on DB6 and DB7.

The Coincidence Detection Circuit provides Object to Object, Object to Projectile, Object to Foreground, Object to Border, Projectile to Projectile, Projectile to Foreground, Projectile to Border, and Border to Foreground Coincidence. Coincidence between any of the Objects, Projectiles, Border, or Foreground is stored in a set of Registers that can be Read by the Microprocessor. These Registers can only be cleared by writing to the OCRST ($2C) (Object Coincidence Register Reset) Address. There are 15 Registers that can be Read to check for Coincidence between the Objects, Projectiles, Border, and Foreground. The status of the Registers is Output to the Microprocessor on DB6 and DB7. This allows the Microprocessor to use the BIT Test Instruction on the particular Address to be tested followed by the BMI or BPL Branch Instructions to test DB7, and the BVS or BVC Branch Instructions to test DB6.

The four Potentiometer Inputs and the two Schmitt Trigger Inputs can also be Read by the Microprocessor in the same manner

as the Coincidence Detection Circuit. These Inputs are all Output on DB7. This allows the Microprocessor to use the BIT Test Instruction followed by the BMI or BPL Branch Instructions to test DB7.

The two Schmitt Trigger Inputs have a Register to store the results of any Input transition to the "0" level. This allows these Inputs to be used as Edge Detection Inputs. The Edge Detection will be stored in the Register until cleared by the Microprocessor. See the VBLNK ($01) (VERTICAL BLANKING CONTROL REGISTER) description.

The following table lists all of the READ Addresses for the VIC and the Functions that can be Read at each Address:

READ ADDRESS	DB6	DB7
$X0	OBJECT A / PROJECTILE A	OBJECT B / PROJECTILE A
$X1	OBJECT B / PROJECTILE B	OBJECT A / PROJECTILE B
$X2	OBJECT A / BORDER	OBJECT A / FOREGROUND
$X3	OBJECT B / BORDER	OBJECT B / FOREGROUND
$X4	PROJECTILE A / BORDER	PROJECTILE A / FOREGROUND
$X5	PROJECTILE B / BORDER	PROJECTILE B / FOREGROUND
$X6		BORDER / FOREGROUND
$X7	PROJECTILE A / PROJECTILE B	OBJECT A / OBJECT B
$X8		LEFT CONTROLLER PIN 5 POTENTIOMETER (VIC PIN 40)
$X9		LEFT CONTROLLER PIN 9 POTENTIOMETER (VIC PIN 39)

READ ADDRESS	DB6	DB7
$XA		RIGHT CONTROLLER PIN 5 POTENTIOMETER (VIC PIN 38)
$XB		RIGHT CONTROLLER PIN 9 POTENTIOMETER (VIC PIN 37)
$XC		LEFT CONTROLLER PIN 6 SCHMITT TRIGGER (VIC PIN 36)
$XD		RIGHT CONTROLLER PIN 6 SCHMITT TRIGGER (VIC PIN 35)

SYSTEM I/O

The Atari Video Game System I/O is located on the front panel of the console and on the back of the console. The front panel I/O consists of the five switches that are labeled TV Type, Left Difficulty, Right Difficulty, Game Select, and Game Reset. The I/O on the back of the console is interfaced through two 9 pin connectors labeled Right Controller and Left Controller. There are four different types of Controllers that can be connected to the Game System. These are the Joystick Controllers, the Paddle Controllers, the Keyboard Controllers, and the Steering Controllers.

The five front panel controls are connected to the 6532 Port B. The position of these switches can be read at address $0282. Each switch is represented by one bit of data. Three of the data bits are not used (DB2, DB4 and DB5). The "Game Reset" switch is DB0, the "Game Select" switch is DB1, the "TV Type" switch is DB3, the "Left Difficulty" switch is DB6, and the "Right Difficulty" switch is DB7. See page 42 for an explanation of the switch position and the logic state of each of the data bits as they are read from the 6532.

One note of caution should be made about using the Data Direction Register for Port B at address $0283. When the power is first turned on, a power-up reset pulse is applied to the 6532 reset pin. When the reset is activated, the Data Direction

Registers are set to the logic "0" state which in turn causes all pins on both Port A and Port B to become inputs. When the pins of the Ports are inputs, a pullup resistor to the positive power supply is connected to each pin. Each of the front panel control switches is connected between one of the Port B pins and the electrical ground of the Game System. When a switch is in the open position, the voltage at the associated pin will be a logic "1". When a switch is in the closed position the input pin will be connected to ground resulting in a logic "0".

If a bit of the Port B Data Direction Register is programmed to a logic "1", the associated pin becomes an output. When a Port B pin is an output and a read of address $0282 is executed, the data that will be read is from the Output Register for Port B not the logic state of the Port B pin, resulting in incorrect data. To prevent this problem the Data Direction Register at address $0283 should not be programmed to anything other than the data $00 which is the same as the power on reset condition.

The two connectors on the back of the Game System are connected to both the 6532 and the Video Interface Circuit. Four of the pins on each connector are connected to Port A of the 6532, three of the pins on each connector are connected to the Video Interface Circuit, and two pins on each connector are each connected to the power supplies. The connections of each of these 9 pin connectors are shown in the following table:

Left Controller		Right Controller	
Connector Pin #		Connector Pin #	
L1	PA4 (6532 PIN 12)	R1	PA0 (6532 PIN 8)
L2	PA5 (6532 PIN 13)	R2	PA1 (6532 PIN 9)
L3	PA6 (6532 PIN 14)	R3	PA2 (6532 PIN 10)
L4	PA7 (6532 PIN 15)	R4	PA3 (6532 PIN 11)
L5	VIC PIN 40	R5	VIC PIN 38
L6	VIC PIN 36	R6	VIC PIN 35
L7	+5V	R7	+5V
L8	GROUND	R8	GROUND
L9	VIC PIN 39	R9	VIC PIN 37

Please note that PA0 is Port A of the 6532 and is read or written to at address $0280 and is DB0. The same is true for each of the other pins through to PA7 which is DB7 of address $0280.

Controller Connector Pins R1, R2, R3, R4, L1, L2, L3, and L4 are each connected to 0.001 ufd capacitor, which is connected to ground, and the 6532 Port A which can be individually programmed to be either an input or an output by using the Data Direction Register for Port A at address $0281 which controls PA0 thru PA7. For more information, the reader is directed to the Data Sheet for the 6532.

Controller Connector Pins R5, R9, L5, and L9 are connected through a 1.8K resistor to the VIC Potentiometer Inputs on pins 37

thru 40. There is a 0.068 ufd capacitor connected between ground and each of the Potentiometer Inputs. The four Potentiometer Inputs are inputs only, however, there are two different input modes. One of the modes is the Potentiometer mode and the other is the Digital mode. The use of these inputs in these two modes is explained in more detail in the description of the Vertical Blanking Control Register ($01) beginning on page 48.

Controller Connector Pins R6 and L6 are each connected to pins 35 and 36 of the VIC through a CMOS Buffer. Pins R6 and L6 each have a 10K resistor connected between the pin and the positive power supply (+5V). There is also a 220 pfd capacitor between the pin and ground. The CMOS Buffer (4050) is non-inverting so the same logic state at the Controller Connector will appear at the VIC Input pin. The input logic levels will be as defined in the Data Sheet for the CMOS Buffer (4050). The Schmitt Trigger input of the VIC cannot be used as it is being driven by the CMOS Buffer Output. These two inputs have two different modes of operation which are explained in more detail in the description of the Vertical Blanking Control Register ($01) beginning on page 48.

JOYSTICK CONTROLLERS

The Joystick Controllers consist of an X-Y joystick and a push button. These activate 5 separate switches. The push button activates a normally open SPST momentary switch. The X-Y joystick can activate one or two of four normally open SPST momentary switches. The five switches are connected between the ground pin (Pin 8) and one of the input pins.

The push button is connected between Pin 8 and Pin 6. When the switch is open the logic state will be a logic "1". When the button is depressed the logic state will be a logic "0".

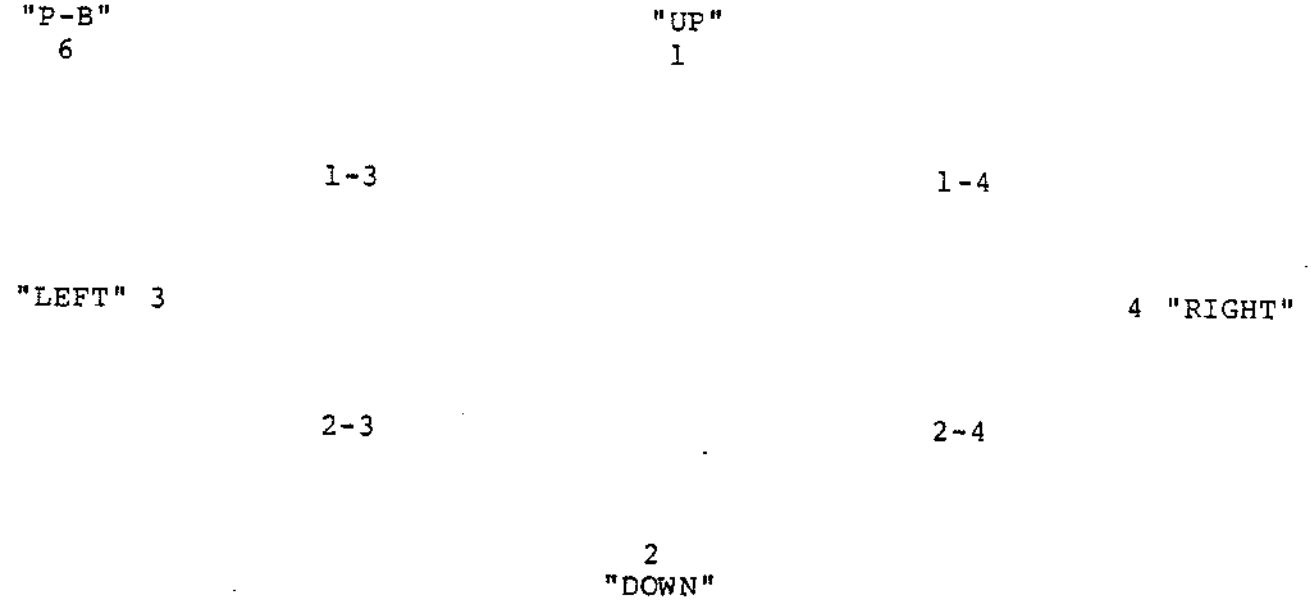
With the push button (P-B) in the upper left corner we can define the location of the four switches for the joystick. The four switches are positioned 90 degrees from each other. When looking down at the controller, the position straight up is defined as the "forward" or "up" position. The position 180 degrees from that position is defined as the "reverse" or "down" position. The position 90 degrees from the "up" position to the right will be called the "right" position and the position 180 degrees from there will be the "left" position. One of each of the four X-Y switches is associated with each of these positions.

The "up" position is associated with the switch connected between Pin 1 and Pin 8 (Ground). The "down" position is associated with the switch connected between Pin 2 and Pin 8

(Ground). The "left" position is associated with the switch connected between Pin 3 and Pin 8 (Ground). The "right" position is associated with the switch connected between Pin 4 and Pin 8 (Ground).

When the joystick is moved into one of these positions the switch associated with that position will be closed connecting the input pin associated with that switch to ground. With the Joystick Controller connected to the connector on the Video Game System the associated pin of the 6532 Port A will be grounded. When grounded, and with the pin programmed to the Input Mode the data at that pin will be a logic "0". When none of the pins are grounded the 6532 will read the data $FF. Each of the Joystick Controllers is associated with the high or low nibble of address $0280. The low nibble is associated with the Right Controller and high nibble is associated with the Left Controller.

If the joystick is moved to a position midway between an adjacent pair of switches, both switches will be activiated. This creates 8 possible positions for the joystick, plus the center position with all of the switches open. These positions and the pins associated are shown below in their relative position when they are activated:



The following table shows the position of the joystick, the switches that will be closed and the data for the nibble associated with each of the 9 possible combinations:

Joystick Position	Pins Connected to Pin 8	Data
Center	NONE	$F
Up	1	$E
Down	2	$D
Left	3	$B
Right	4	$7
Upper Right	1, 4	$6
Lower Right	2, 4	$5
Lower Left	2, 3	$9
Upper Left	1, 3	$A

The push button is read through the VIC at address $0C or $0D on DB7. The push button for the Joystick Controller connected in

the "Left Controller" position is read at address $0C, and the "Right Controller" is read at address $0D. When the push button is depressed and the switch is closed, the data read from the VIC will be a logic "0". When the push button is not being pushed and the latching mode for the VIC input is not selected, the data read from the VIC will be a logic "1".

PADDLE CONTROLLERS

The Paddle Controllers consist of two separate controllers attached to a single connector. The Video Game System can accept up to four controllers attached to two connectors. Each controller contains a 1.0 Megaohm linear taper potentiometer and a normally open SPST momentary switch. The potentiometer is mounted in the center of the controller with a knob attached. The push button is mounted on the left side of the controller. When the knob is rotated fully clockwise the minimum resistance value is selected. As the knob is rotated in the counterclockwise direction the value of the resistance selected will increase until the maximum value is selected when the knob reaches the full counterclockwise position.

The first controller has one side of the switch connected to Pin 4 and the other side connected to Pin 8 (Ground). The potentiometer has one side connected to Pin 7 (+5V) and the wiper arm connected to Pin 5. The second controller has one side of the switch connected to Pin 3 and the other side connected to Pin 8 (Ground). The potentiometer has one side connected to Pin 7 (+5V) and the wiper arm connected to Pin 9.

The push button switches are read from the 6532 Port A at address $0280 on data bits DB2, DB3, DB6, and DB7. The potentiometers are read through the VIC from the four Potentiometer Inputs.

The position setting of the Potentiometers is determined by measuring the time constant of the RC network created by the connection of the capacitor on the input pin of the VIC to the +5V power supply through the 1.8K resistor and the potentiometer. Before the time constant can be measured, the capacitor must be discharged. This is done by the VIC. The discharge transistors are turned on at the beginning of the Vertical Blanking Period which discharges the capacitors. At the end of the Vertical Blanking Period the discharge transistors are turned off and the capacitors begin to charge up towards the +5V power supply level. The lower the resistance setting of the potentiometer the faster the capacitor will charge up.

The Potentiometer Inputs of the VIC are level sensitive inputs and when the input voltage exceeds the trip point, the logic state at this input will change from a logic "0" to a logic "1". The software program tests the potentiometer input status by reading the appropriate address of the VIC once per horizontal line until the logic level shift is detected. For more information about the use of the Potentiometer Inputs see the description of the Vertical Blanking Control Registers ($01) on page 48 and the description of the Coincidence and Input Registers on page 103.

KEYBOARD CONTROLLERS

The Keyboard Controller consists of a keyboard with 12 normally open SPST momentary switches. The keyboard is layed out exactly like a standard push button telephone. The push buttons are labeled "1", "2", "3", "4", "5", "6", "7", "8", "9". "0", "*", and "#". Each push button activates a switch that is connected between 2 of the pins on the Controller Connector. The switches are wired in a X-Y matrix. The matrix is 4 rows by 3 columns.

The top row contains the switches "1", "2", and "3"; and one side of each of these switches is connected to Pin 1. The second row contains the switches "4", "5", and "6"; and one side of each of these switches is connected to Pin 2. The third row contains the switches "7", "8", and "9"; and one side of each of these switches is connected to Pin 3. The bottom row contains the switches "*", "0", and "#"; and one side of each of these switches is connected to Pin 4.

The left column contains the switches "1", "4", "7", and "*"; and the other side of these switches is connected to Pin 5. The middle column contains the switches "2", "5", "8", and "0"; and the other side of each of these switches is connected to Pin 9. The right column contains the switches "3", "6", "9", and "#"; and the other side of each of these switches is connected to Pin 6. Pin 6 on the Left Controller and Right Controller connectors on the Video Game System have an internal pull up resistor to the +5V

power supply. Pins 5 and 9 do not have a pull up so a 4.7K resistor is connected between Pin 5 and Pin 7 (+5V), and between Pin 9 and Pin 7 (+5V) inside the Keyboard Controller.

The Keyboard Controllers require that the keyboard be scanned in order to determine what switch is depressed if any. This is done by using Pins 5, 9, and 6 which are used as inputs to the Video Game System. Pins 5 and 9 will be used in the digital mode. Pins 1, 2, 3, and 4 will be used as outputs from the Video Game System. This requires that the Data Direction Register for Port A of the 6532 at address $0281 be programmed to the output mode. Depending on which connector the Keyboard Controller is connected to, the four bits associated with Pins 1, 2, 3, and 4 are programmed to the logic "0" state one at a time. The data at the input of Pins 5, 9, and 6 is examined to see if any of these pins are at the logic "0" state.

When all of the switches are open (normal state) Pins 5, 9, and 6 will all be at the logic "1" state regardless of the logic state of Pins 1, 2, 3, and 4. If Pins 1, 2, 3, and 4 are all programmed to the logic "1" state and any of the switches are closed, the Pins 5, 9, and 6 will remain at a logic "1". However, if one of the Pins 1, 2, 3, and 4 is programmed to a logic "0" and one of the switches in the row associated with that pin is closed, the pin associated with the column the closed switch is in will be pulled to the logic "0" state. The software can then determine, by matching up the row and column, which button was closed. If

two or more switches are closed simultaneously then incorrect results may be derived unless the software is written to account for this possibility.

The following table shows the pins that will be connected for each of the buttons when they are pressed:

BUTTON	ROW PIN NO.	COLUMN PIN NO.
1	1	5
2	1	9
3	1	6
4	2	5
5	2	9
6	2	6
7	3	5
8	3	9
9	3	6
0	4	9
*	4	5
#	4	6

STEERING CONTROLLERS

The Steering Controllers consist of a shaft encoder and a push button. The push button activates a normally open SPST momentary switch. The shaft encoder generates a two bit Gray Code. The push button switch and the shaft encoder are connected between the ground pin (Pin 8) and one of the connector pins.

The push button switch is connected between Pin 6 and Pin 8 (Ground). When the switch is open the logic state will be a logic "1". When the push button is depressed the logic state will be a logic "0".

The shaft encoder is attached to a knob. The knob can be rotated 360 degrees continuously. There are 16 separate positions of the shaft in the 360 degress of rotation. The shaft encoding is broken into four quadrants. Each of the quadrants is a duplicate of the others only rotated in 90 degree increments. Each of the different shaft codes is located 22.5 degrees from each other. There are only four different codes repeated four times as the shaft is rotated the full 360 degrees.

The shaft encoder contains what appears to be two SPST switches. One switch is connected between Pin 1 and Pin 8 (Ground). The second switch is connected between Pin 2 and Pin 8 (Ground). When the switches are open the logic state will be a logic "1". When the switches are closed the logic state will be a logic "0".

The shaft encoder is used more for determining rotation of the knob rather than absolute positioning of the knob. The Gray code allows the programmer to determine in which direction the knob is being rotated by comparing the previous code from the encoder to a new code. Since only one bit of data will change at a time as the shaft is rotated, erroneous information about the rotation of the shaft is eliminated. The software that interprets the data from the shaft encoder can initialize on any position of the shaft and give rotation results as it reads the data from the encoder.

The following table shows the 16 different positions as the shaft is rotated both clockwise and counterclockwise. For this example the starting position will have the data "00", however, the starting data can be any of the four possible combinations. The starting angle is assumed to be 0 degrees with clockwise rotation being positive increments of angle of rotation. The data shown at Pin 1 and Pin 2 is the logic state as it would be read by the 6532 Port A at address $0280 on data bits DB0 and DB1 or on data bits DB4 and DB5.

Clockwise Rotation Angle In Degrees	Pin 1	Pin 2	Counterclockwise Rotation Angle In Degrees	Pin 1	Pin 2
0.0	0	0	0.0	0	0
+22.5	0	1	-22.5	1	0
+45.0	1	1	-45.0	1	1
+67.5	1	0	-67.5	0	1
+90.0	0	0	-90.0	0	0
+112.5	0	1	-112.5	1	0
+135.0	1	1	-135.0	1	1
+157.5	1	0	-157.5	0	1
+180.0	0	0	-180.0	0	0
+202.5	0	1	-202.5	1	0
+225.0	1	1	-225.0	1	1
+247.5	1	0	-247.5	0	1
+270.0	0	0	-270.0	0	0
+292.5	0	1	-292.5	1	0
+315.0	1	1	-315.0	1	1
+337.5	1	0	-337.5	0	1
+360.0	0	0	-360.0	0	0

FRONT PANEL CONTROL SWITCHES

The Front Panel Control Switches consist of three SPST switches and two normally open SPST momentary switches.

The two momentary switches are the "Game Select" and "Game Reset" switches. These switches are connected between ground and pins on Port B of the 6532. When these switches are in the normal position the data is a logic "1". When each of these switches are depressed the logic state for each switch will be a logic "0".

The three other switches are the "TV Type", the "Left Difficulty", and the "Right Difficulty" switches. These switches are also connected between ground and pins on Port B of the 6532. When these switches are in the up position the data is a logic "1". This is the "Color" position for the "TV Type" switch and position "A" for the "Difficulty" switches. When these switches are in the down position the logic state for each switch will be a logic "0". This is the "B-W" position for the "TV Type" switch and the "B" position for the "Difficulty" switches.

The connection locations for each of these switches is shown in the following table:

$0282 Port B	Switch
PB0	"Game Reset"
PB1	"Game Select"
PB2	No Connection
PB3	"TV Type"
PB4	No Connection
PB5	No Connection
PB6	"Left Difficulty"
PB7	"Right Difficulty"

For more information about using the Front Panel Control Switches see the description of the System I/O on page 106.

PROGRAMMING

This Chapter deals with the basic programming requirements to make a picture and eventually a game on the Atari Video Game System.

A frame of a Video picture requires the following functions be performed:

1. Vertical Blanking ON.
2. Perform Game Rule Program.
3. Vertical Sync ON.
4. Vertical Sync OFF.
5. Perform Game Rule Program.
6. Vertical Blanking OFF.
7. Display Video Picture.

   Loop back to step 1.

There are certain timing requirements for each of the above operations. Each frame must last exactly 262 lines of horizontal scan in order to meet the NTSC Television timing requirements. The NTSC Television standard translates to the following periods specified in number of horizontal scan lines:

Vertical Blanking (Total)	21 lines
Vertical Blanking before Vertical Sync	3 lines
Vertical Sync	3 lines

Vertical Blanking after Vertical Sync	15 lines
Video ON (Visible Picture)	241 lines

The above specifications are for standard video pictures which includes overscan both above and below the visible area on the television screen. In practice we modify the Blanking and visible area line counts. However, the total number of lines must remain 262. The normal maximum number of lines of visible area is 224 lines or less. Each of the lines removed from the visible area is added to the Vertical Blanking either before, after or both before and after the Vertical Sync. As the number of lines of Vertical Blanking after Vertical Sync are increased, a black area at the top of the screen will move further down from the top. As the number of lines of Vertical Blanking before Vertical Sync are increased, a black area at the bottom of the screen will move further up from the bottom.

The Vertical Blanking Period must be an exact number of lines with the Vertical Sync located during the Blanking period in exactly the same location each and every frame. During the Vertical Blanking Period the game rule programs and algorithms are normally executed. It would be an excessive burden to have these programs also keep track of the number of lines being scanned while executing, so we need an alternate method to count the lines. Contained in the 6532 is an Interval Timer that can be programmed to a time interval that is equal to the number of lines to be counted.

The Interval Timer can be programmed to time various intervals from 1 to 262,144 microprocessor clock cycles. The Interval Timer has a pre-scaler that pre-divides the microprocessor clock by 1, 8, 64, or 1024. The divided clock then drives a counter that can be programmed from 1 to 256. The total interval timed is the value of the pre-scaler multiplied by the value programmed into the counter.

There are 76 microprocessors in each horizontal scan line. The following table shows the number of microprocessor clock cycles in various line counts for the Vertical Blanking Period. These values are computed by multiplying the number of lines by 76 then subtracting 38 (half a line period) then dividing by 8 (pre-scaler factor) and finally taking the integer value.

Line #	MPU Clocks	Pre-Scaler	Counter
1	76	8	4
2	152	8	14
3	228	8	23
4	304	8	33
5	380	8	42
6	456	8	52
7	532	8	61
8	608	8	71
9	684	9	80
10	760	9	90
11	836	8	99
12	912	8	109
13	988	8	118
14	1064	8	128
15	1140	8	137
16	1216	8	147
17	1292	8	156
18	1368	8	166
19	1444	9	175
20	1520	8	185
21	1596	8	194
22	1672	8	204
23	1748	8	213
24	1824	8	223
25	1900	8	232

A program using the timer to count 15 lines would look like the following:

```
        STA $02             ; HSYNC
        LDA #137            ; 15 line interval
        STA $0295           ; Start Timer

    Perform program here.

LOOP    LDA $0284           ; Read Timer Value
        BNE LOOP            ; Loop until Timer = 0
```

If we were using this program to do a Vertical Blanking Period of 21 lines, the program would be as follows:

```
        LDA #$02            ; Data to turn on Blanking
                            ; Other Data Bits may be set also
        STA $02             ; HSYNC
        STA $01             ; VBLNK Turn on Blanking
        LDA #194            ; 21 line interval
        STA $0295           ; Start Timer

    Perform program here

LOOP    LDA $0284           ; Read Timer Value
        BNE LOOP            ; Loop until Timer = 0
        STA $02             ; HSYNC
        STA $01             ; VBLNK
```

This program does not account for the Vertical Sync that must be inserted into the Vertical Blanking Period. In reality, if we were doing the NTSC Standard Sync, the timer could be programmed for 3 lines of Blanking, then 3 lines of Sync, then 15 more lines of Blanking. The Blanking is not turned off during the Vertical Sync and remains on for all of the 21 lines. For additional information see the description of the Vertical Blanking ($01) on page 48.

The program that executes the actual creation of the video picture must also keep track of the number of lines being scanned during the execution of the program to make sure that every frame always has a total of 262 lines.

One of the biggest problems encountered in the generation of the video picture is the horizontal placement of the Objects, Projectiles, and the Border. In the section, Horizontal Counter Resets (page 57), the description of the Counter Resets describe using CROBA, CROBB, CRPJA, CRPJB, and CRBDR to reset the counter and therefore the location of each of these objects. These reset commands can be executed outside of the Horizontal Blanking in order to place an object horizontally anywhere on the screen. However, extreme care must be exercised to perform this object placement. By using a subroutine that always executes the write to the object during exactly the same microprocessor clock each time, an object may be placed in the same location each time the

subroutine is executed. The following subroutine is such a program:

```
        STA $02
LOOP    DEY                     ; HSYNC
        BPL LOOP
      * STA $001X
                                ; X
```

*Note this instruction cannot be direct[illegible] must be done in the following manner; [illegible] assembler will try to convert STA $001X to [illegible] not want to do in this program.

This program is exactly timed so that the [illegible] Register does a course positioning of the res[illegible] takes 5 microprocessor clock cycles or 15 VIC clock cycles to loop 1 time. Each time Y is decremented or incremented the location of the reset and thus the object is moved 15 VIC clock cycl[illegible]. This then places the object on 15 VIC clock cycle increments. We can now use the Horizontal Movement Section of the VIC to fine position the object. This section has a movement range of -7 to +8 VIC clock cycles which is sufficient to fine position the object. By adding to the above subroutines, we can make a single subroutine to position an object anywhere in one of the 160 horizontal locations. If we call the subroutine and pass a value in 'Y" (Course Position) and a value in "A" (Fine Position) to the

subroutine, we can execute the following program which will position the object anywhere on the screen horizontally:

```
        STA $02              ; HSYNC
LOOP    DEY
        BPL LOOP
        .BYT $8D, $1X, $00   ; STA $001X Course Position
        STA HMZZZ            ; ZZZ = Object, Projectile or Border
        STA $02              ; HSYNC
        STA $2X              ; X = 0, 1, 2, 3, or 4
        STA $2A              ; HMENB Fine Position
        RTS
```

This program only positions one object, so we can modify the program to allow the flexibility of using the same program to position any of the objects. By also passing a value in "X" to determine which object we are positioning ("X" = 0, 1, 2, 3, or 4) we can use the following universal program:

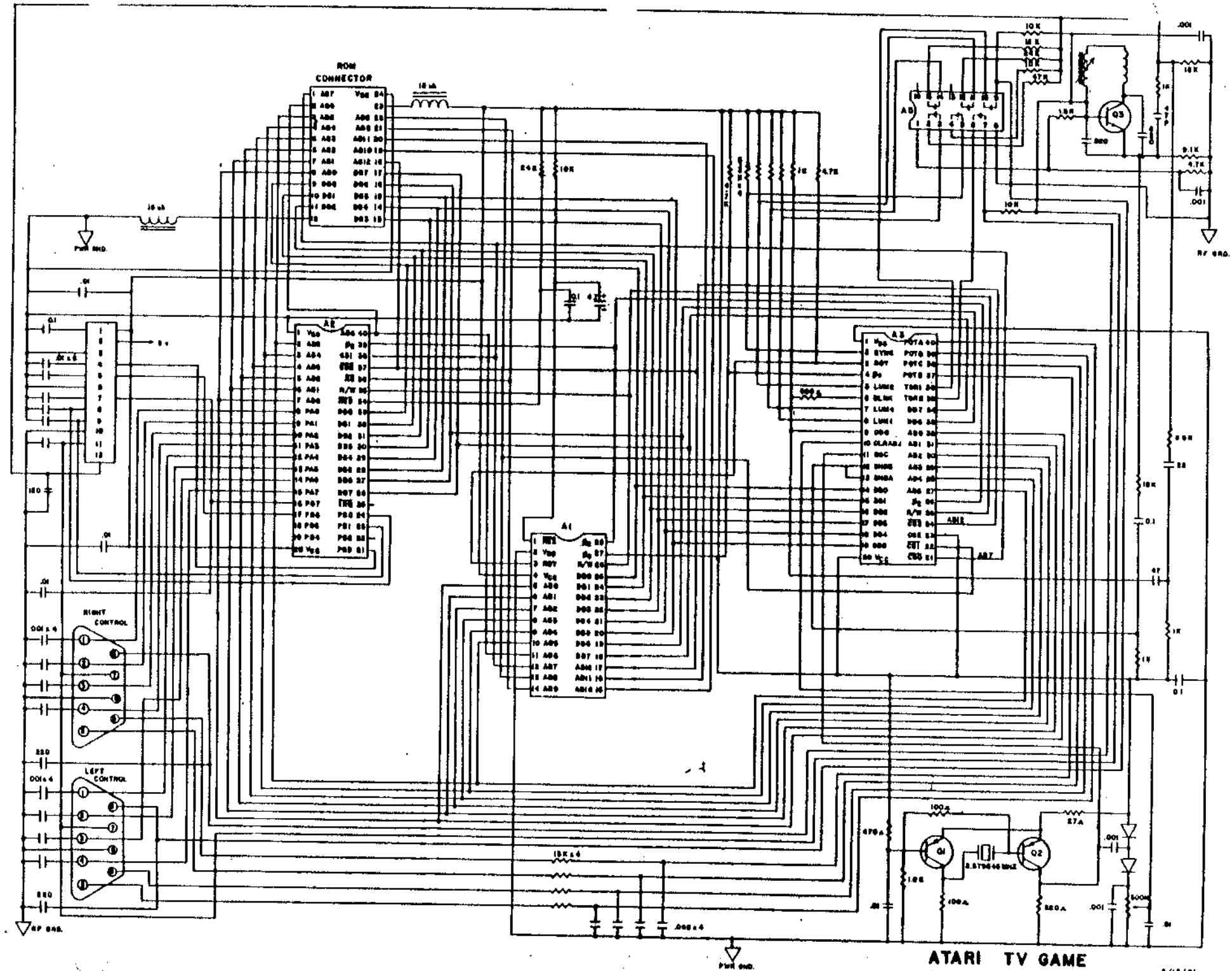
```
        STA $02              ; HSYNC
LOOP    DEY
        BPL LOOP
        STA $10,X            ; CROBA,X
        STA $02              ; HSYNC
        STA $20,X            ; HMOBA,X
        STA $2A              ; HMENB
        RTS
```

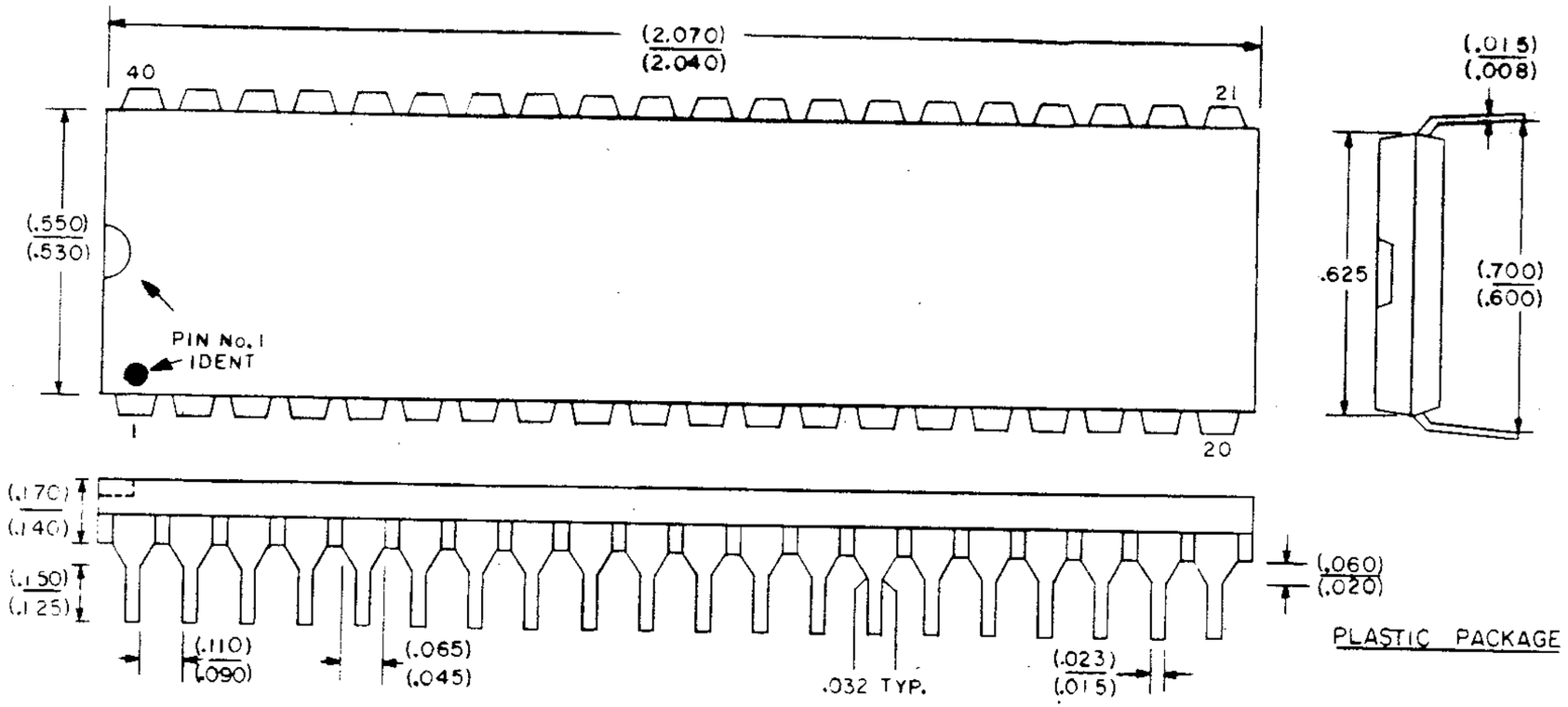
There are limitations to the values in "A", "Y", and "X". e horizontal motion "A" may not use the value #$80, "Y" must be in the range of #$02 to #$0C, and X must be 0, 1, 2, 3, or 4. What we need now is a program to generate the value of "A" and "Y". The following subroutine is such a program in that you pass a value in "A" to the program and it returns a value in "Y" for the course position of the object and a value in "A" for the fine position of the object. The initial value passed to the program is in the range of 0 to 159 for one of the 160 possible horizontal locations.

```
CLC
ADC #$2E
TAY
AND #$0F
STA TEMP
TYA
LSR A
LSR A
LSR A
LSR A
TAY
CLC
ADC TEMP
CMP #$0F
BCC SB
SBC #$0F
INY
```

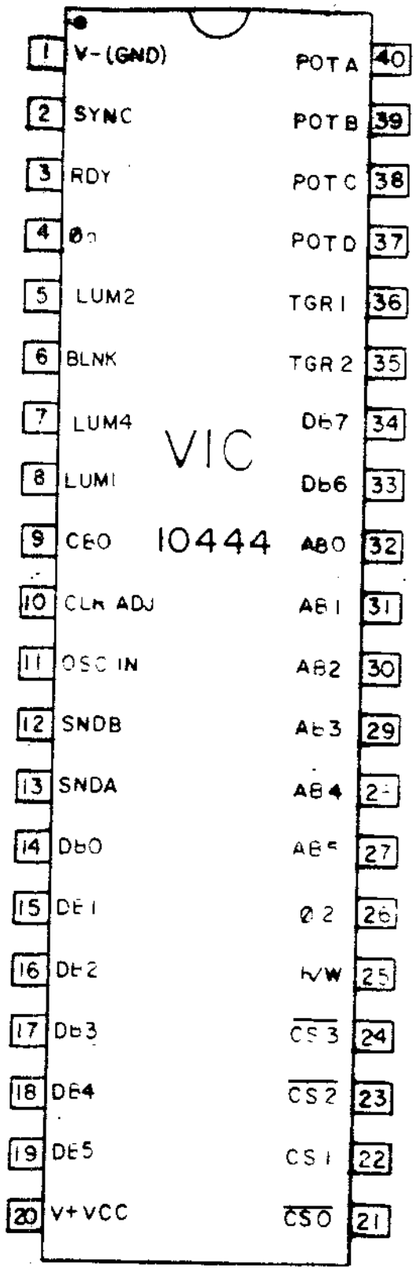
```
SB      EOR #$07
        ASL A
        ASL A
        ASL A
        ASL A
        RTS
```

Note: This program is from an Activision Game Program.


ROM CONNECTOR
RIGHT CONTROL
LEFT CONTROL
A1
A2
A3
A5
Q1
Q2
Q3
ATARI TV GAME
MAIN PC BOARD
9/18/81



PLASTIC PACKAGE



PACKAGE DRAWING AND PINOUT

REV. A
8/16/81