List of Figures

1. Introduction

1.1 Preface

Rainbow is a "video graphics interpreter." That is, it interprets a description (in memory) of full-screen color images. Just as a microprocessor interprets a language (its instruction set), so Rainbow interprets a language, which is described in this document. And just as a microprocessor's program tells it what actions to take, so Rainbow is told, in its language, what images to generate.

The design of Rainbow was begun to provide for the personal computer design team a high performance video system on silicon. Rainbow was to use current technology to create a display capability having more colors, more resolution, and more flexibility than any other graphics system in the home computer market.

Rainbow was designed to be simple to use. The principal expression of this goal is Rainbow's uniform representation. For example, while Antic, the graphics chip in current ATARI Home Computers, uses three different graphics entities (playfield, player, and missile), Rainbow uses only one: the object. Simplicity was also the reason for separating parameters. For example, while an Antic mode specifies color resolution and spatial resolution together, these parameters are independently settable in Rainbow.

1.2 Key features

The principal features of Rainbow are as follows:

- High-resolution screen definition (640 pixels x 480 pixels)
- 256 colors displayable at a time without CPU intervention
- Access to one megabyte of memory over a sixteen bit bus
- Simple, uniform graphical primitives: objects
- Dynamic memory refresh control
- Autonomous DMA fetching of all parameters and data
- Modular hardware allowing expansion of capability
- Relaxed system timing constraints (via line buffers)
- RGB or NTSC/PAL video interface capability
- CPU timing independent of video clocks
- Minimal CPU intervention
- Programmed line interrupt
- Bitmap and runcoded graphic data types
- Transparency
- Independent horizontal and vertical magnification

1.3 Overview

1.3.1 Function

Rainbow translates structured, program-oriented digital representations of graphics into the time-sequenced signals necessary to drive a raster-scanned CRT. It produces interlaced digital video outputs which can be converted to RGB or NTSC/PAL video signals.

1.3.2 Object Based

The Rainbow video graphics system is object-based. All display activity is in movable, variable-size, objects. An object is like a sprite, but more general; it resembles a combination of the Antic entities called playfield and players. Objects have no size limit. The horizontal display complexity is limited to the 24 object processors supplied. Each object processor holds a parameter block and pointers to the object representation in memory. Object processors are reused vertically through automatic parameter linking. There can be 24 objects on a line and 11,520 objects on a screen.

1.3.3 Screen Organization

The screen is organized as an array of square pixels. The standard resolution in Rainbow for NTSC displays is 640 horizontal pixels by 480 vertical pixels. All graphics are represented in screen pixels of these dimensions. Coarser resolution modes are available for use where the picture tube quality or memory limitations prevent effective use of the full resolution available.

1.3.4 Color Palette

Rainbow can produce up to 256 colors at a time without CPU intervention. Each color is represented in sixteen bits; the interpretation of those bits is defined outside the Rainbow chip set. Normal practice would assign a bit field to each of the red, green, and blue color components.

1.3.5 Communication with Memory

Rainbow communicates with memory over an asynchronous, sixteen-bit bus as bus master. Rainbow generates the signals necessary to refresh dynamic memories. The system bus needn't run at a multiple of the video clock rate.

1.3.6 Communication with CPU

The system CPU will set up the object descriptions for Rainbow to interpret and will notify Rainbow by writing initial parameter pointers to it over the bus. Rainbow can interrupt the central processor to indicate error conditions such as "Line Incomplete" and the non-error condition "Programmed Line."

2. Functional description

2.1 Objects

A full-screen Rainbow display is composed of objects and a background. An object is displayed by the action of an object processor interpreting a parameter block. An object processor is a hardware processing element like a microprocessor. A parameter block is a complex instruction to an object processor directing it to produce an object. A parameter block references a window within a picture. A picture holds the pixel data that makes up the context of the object. Rainbow translates memory-represented graphics into screen-displayed graphics. It can depict "two and a half" dimensions; that is, with Rainbow, many two-dimensional objects can move independently, and they can overlay and obscure each other.

Rainbow can process 24 objects on each scan line, one for each object processor. Each object processor is reusable vertically; it reloads itself at the end of a window with the parameters for its next object without CPU intervention.

2.2 Pictures and windows

The source in memory of graphical information, called a "picture," is a sampled pixel map (rather than, for instance, vectors or polygons). The picture can be any size, even much larger than the video display screen. A portion of a picture, called the "window," is selected for display by an object processor at any one time. The window can be the entire picture, or only a piece of it, and it can be located anywhere on the picture (that is, not necessarily at the upper left corner). The object processor fetches source pixel data from the selected window and transforms it into a stream of video (that is, screen pixel) data to be sent to the display picture tube. The data in a source pixel represents not an absolute color but a selection from the color map. The object processor fetches and interprets source pixel data for an object as specified by a parameter block.

2.3 Picture data formats

Pictures are coded in one of two ways, "bitmap" and "runcode." A picture in either is a two-dimensional array of bit fields corresponding to pixels in raster order, left-to-right across each scan line and top-to-bottom, representing a rectangular area. In a bitmap picture, each bit field represents the color of a single source pixel. The bit fields are one, two, four, or eight bits wide; the number of bits determines the number of color choices per pixel.

In a runcode picture each bit field represents a contiguous horizontal sequence (a "run") of pixels of one color. Each runcode specifies both a color to use and the length of the run of pixels of that color. A runcode is always two eight-bit subfields, one specifying color and one specifying run length. Runcodes save memory space and accessing time for objects with large areas of solid color.

Both bitmap and runcode data are packed on word (sixteen-bit) boundaries. The object processor unpacks the memory words and uses the resulting unpacked data to select a color from the color map.

2.4 Object priority

When objects overlap in screen position, one of them will be displayed in the area of overlap, obscuring the others. In such a case, the choice is determined by a fixed priority order among the object processors. There is a single-color background of lower priority than any object processor.

If priority among objects needs reordering, object parameters can be swapped among the object processors by swapping the parameter block pointers in memory; moving or copying the contents of the blocks is not required.

2.4.1 Transparency

Defining areas of an object where lower priority objects can show through is sometimes desirable. This feature is called "transparency". Pixel bit fields that are zero can be interpreted as transparent pixels; this interpretation is selected by a parameter in each object processor. If transparency is not selected, every pixel bit field value represents a color.

Figure 2-1 shows an example of transparency in which two objects partially overlap. Object processor 5 displays a window from picture I and object processor 6 displays a higher priority window with transparency from picture II. At point A, no object is active, so background displays. At point B, object 5 is active and so it displays pixels from the window in picture I. At point C, object 6 becomes active and it obscures object 5, since object 6 has higher priority. At point D, however, object 6 begins reading pixel data indicating transparency. Object 6 then yields priority to others. In this case, object 5 has the next lower priority, and so its pixel data displays.

2.5 Object parameters

A parameter block is the instruction to display an object. Each object's parameter block specifes the object's display representation in memory, the object's location on the screen, its color and its spatial properties. Parameter blocks are linked in a list. The parameters are: 1) origin, 2) stride, 3) X and Y, 4) width and length, 5) scale factors, 6) depth, 7) color index, 8) transparency, 9) coding, and 10) link. The relation of these parameters to the displayed result is shown in Figure 2-2.

2.5.1 Origin

Origin is the bit-field address of the upper left-hand corner of a window. This value is made up of a twenty-bit word address and a four-bit pixel address (these may be considered as a 21-bit byte address and a three-bit pixel address, if convenient).

2.5.2 Stride

Stride is the number of words of memory between data for a pixel and data for the pixel to be displayed directly below it. In other words, stride is the width in words of the picture from which the object takes pixel data. For example, if origin points to the top left corner of the picture, then origin plus stride points to the start of the second line in the picture.

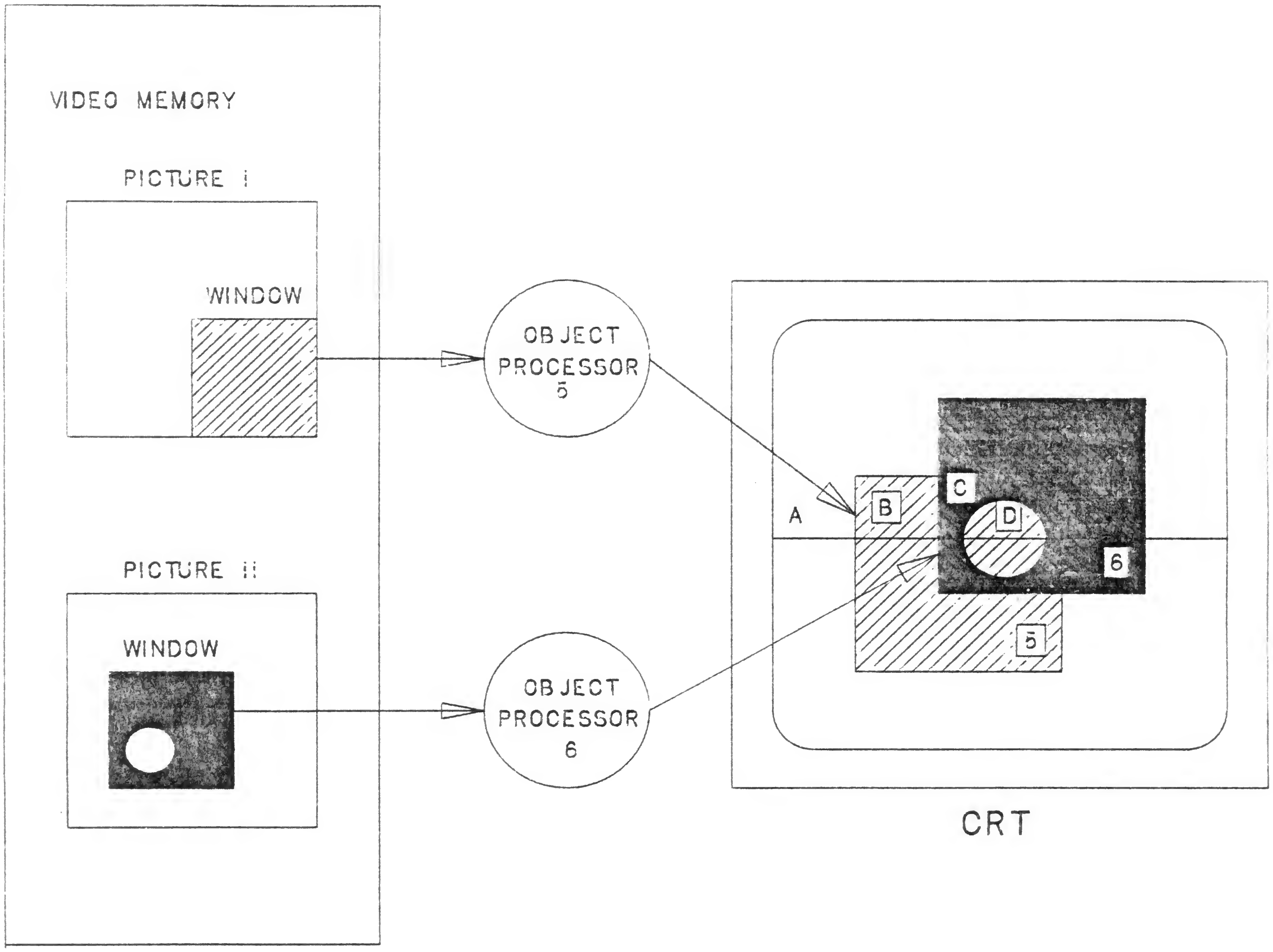


Figure 2-1: Transparency example

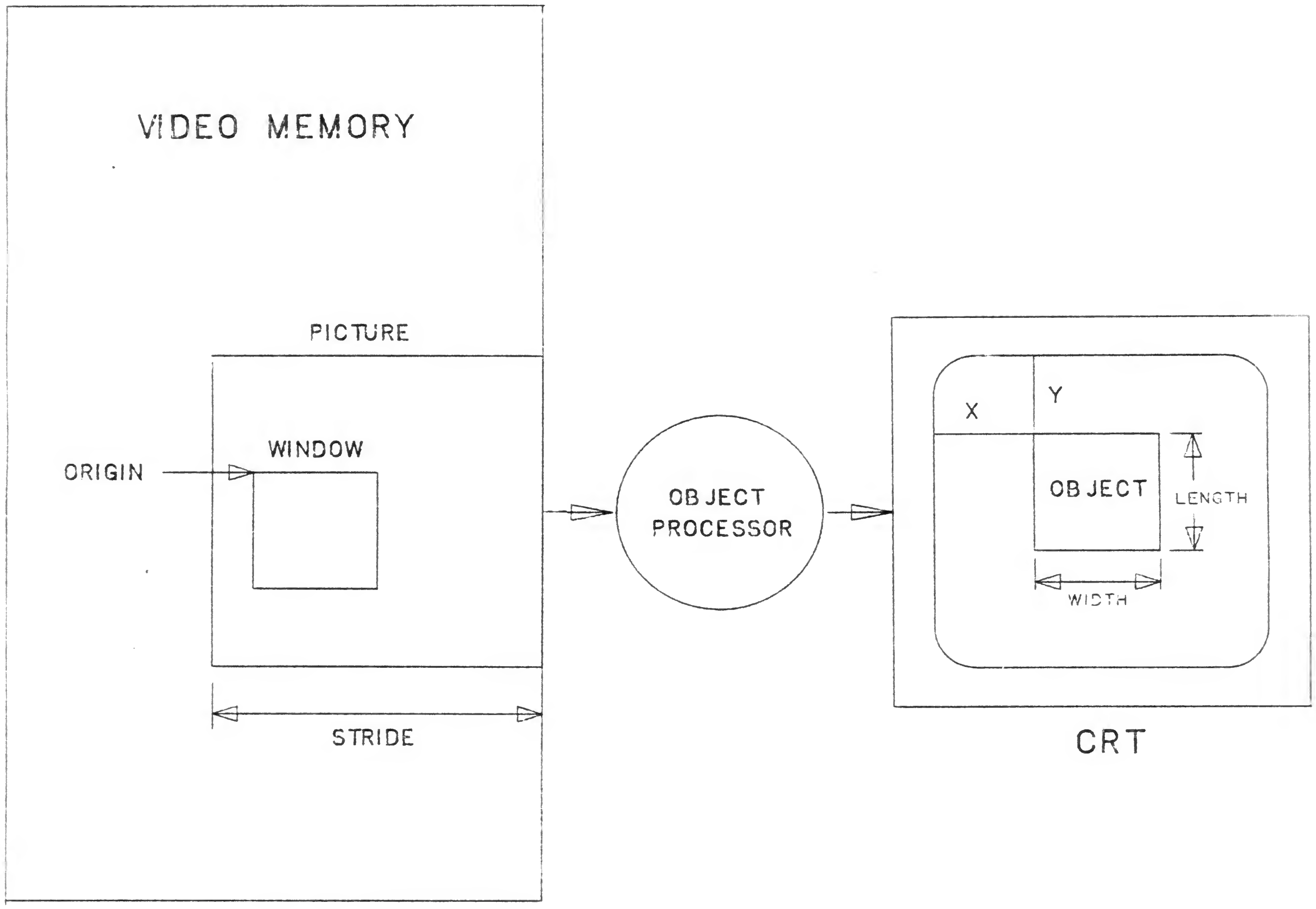


Figure 2-2: Picture, window, and object

2.5.3 X and Y

X and Y specify the position of the object on the screen. The X parameter is the number of screen pixels the left edge of the object is offset from the left side of the screen. The Y parameter is the number of scan lines the top line of the object is offset from the top of the screen.

If X is specified greater than the last pixel on a line (639 for NTSC format), or Y greater than the last visible line (usually 479), the object will not appear at all.

2.5.4 Width and Length

Width and Length specify the rectangular dimensions of the object as it appears on the screen. Width is the horizontal size of the object, measured in screen pixels. Length is the vertical size of the object, measured in screen lines.

If Width or Length is zero, the object will not appear (of course).

If X + Width or Y + Length imply that part of the object would be off the screen, that part is simply ignored (not fetched or displayed).

2.5.5 Scale.X and Scale.Y

Scale factors magnify an object in the X and Y directions. Each source pixel will appear Scale.X + 1 times in succession. Each line of source pixels will appear Scale.Y + 1 times. Scale.X is ignored for runcoded objects.

Magnification is achieved by repeating each pixel, or each line, the number of times given by the scale factor. Implementing this effect in the Y direction with interlaced scanning requires that each line be repeated one-half Scale.Y times in each field. For odd scales, "half" must average out; for instance, with Scale.Y+1 = 5, lines will be repeated alternately two times and three times on successive fields. The result is five repetitions of each line in each complete frame.

2.5.6 Depth

Depth specifies the number of bits that make up each source pixel in bitmap coding--one, two, four, or eight.

2.5.7 Color Index

The Color Index points to the start in the color map of the range of colors available to the object. All colors are named relative to this one; that is, Color Index is added to the pixel data to give the effective address in the color map.

2.5.8 Transparency

Transparency indicates whether the "transparent" color is available to this object. If transparency is set and the pixel data is zero, the object processor will not display a color, but instead lets a lower priority object or background display.

2.5.9 Coding

Coding indicates the format of source pixel data: bitmap (as described by Depth) if clear, runcode if set.

2.5.10 Link

Link is the absolute address of the parameter block for the next object displayed by this object processor. The address of the first parameter block, called the "root," for this object processor (for display in each frame) is written directly to an object processor by the CPU. If the link or root in an object processor is zero, no further objects will be interpreted by that processor until a new root is written; in other words, the object processor is not reused vertically. A parameter block is automatically loaded either when a non-zero root is written or at the end of display of the current object's window.

2.6 Parameter block layout

Each parameter block begins on a word boundary. All numeric parameters are unsigned integers.

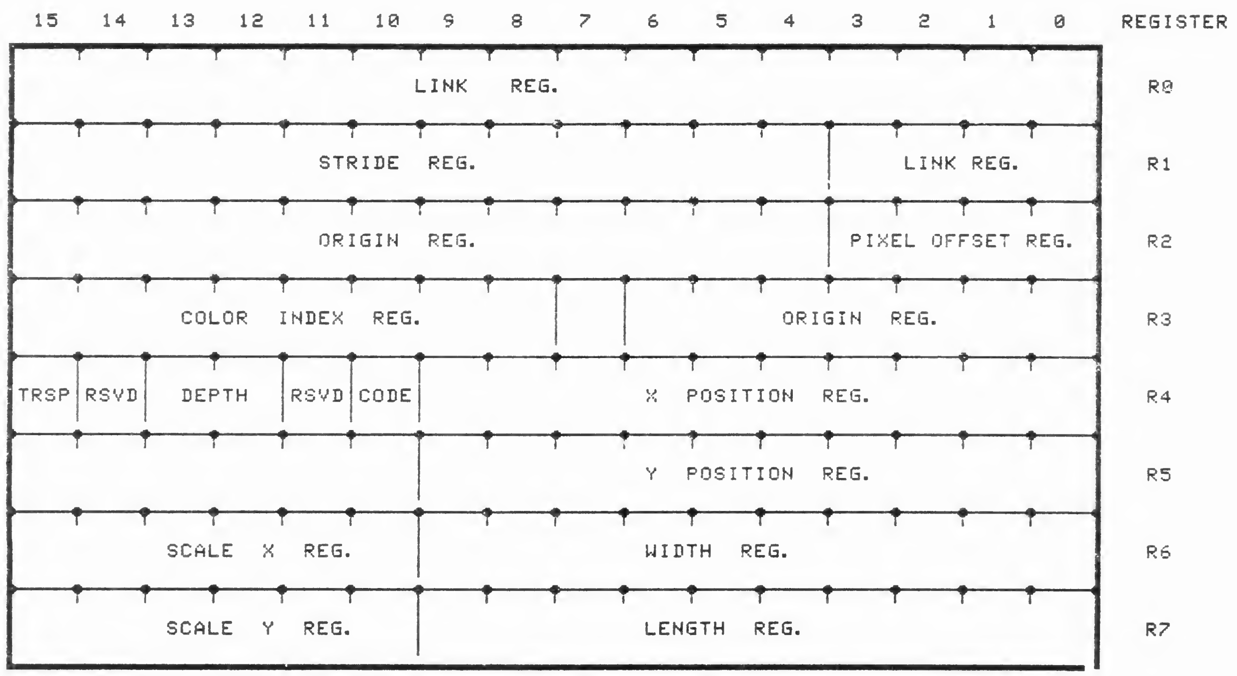


Figure 2-3: Parameter block layout

Origin	Bit pointer to the upper left-hand corner of the window. (24 bits)
Stride	Distance in words from a pixel in the picture to the pixel directly below it. (12 bits)

X	Horizontal position of object's left edge, measured in screen pixels from left of screen. (10 bits)
Y	Vertical position of object's top line, measured in screen lines from top of screen. (10 bits)
Width	Horizontal size of object, measured in screen pixels. (10 bits)
Length	Vertical size of object, measured in screen pixels (lines). (10 bits)
Scale.X	Horizontal scale magnification factor. Each pixel will appear Scale.X + 1 times. (6 bits)
Scale.Y	Vertical scale magnification factor. Each line of source pixels will appear Scale.Y + 1 times. Interlaced scan is taken into account. (6 bits)
Depth	Selects number of bits used to represent each source pixel. The relationship between depth, pixels per word, and the number of different colors that a pixel could assume is shown in Figure 2-4. (2 bits)

Depth	Pixels/word	Bits/pixel	Colors
3	2	8	256
2	4	4	16
1	8	2	4
0	16	1	2

Figure 2-4: Depth encoding

Color Index	Index in the color map of the color to use for pixel value zero. (8 bits)
Transparency	Selects whether pixel data with zero value means transparent. If set, yield priority. If cleared, generate color given by Color

	Index. (1 bit)
Coding	Selects coding format of source pixel data: cleared for bitmap-style data (as described by Depth), set for runcoded data. (1 bit)
Link	A pointer to the next object parameter block in memory (a byte address). The object processor reloads parameters upon completion of the display of the current object. (20 bits)

2.7 Pixel data formats

Pixel data--the actual contents of pictures--is stored in two-dimensional arrays of sixteen-bit words. Each word of a picture array has the same format, determined by the coding and depth parameters as shown in figure 2-5. Within a word, the pixel at the LSB end comes first (leftmost) on the video line.

2.8 Color map

Colors to be generated by Rainbow are stored in a 256-location memory called the color map. Thus a Rainbow screenful can have up to 256 distinct colors, or more if the processor reloads the color map within a field. The color map memory is external to the Rainbow custom chip set.

2.8.1 Color representation

Each location contains up to sixteen bits of color information, so the total range of accessible colors is 65536. In one possible arrangement, four bits specify the level of each of red, green, and blue intensity. Shades of gray are generated by equal values for each color, giving sixteen gray levels (including black and white levels).

Other encodings of color values are possible. The choice of color and flag encodings may be made independently for each system or product that uses Rainbow.

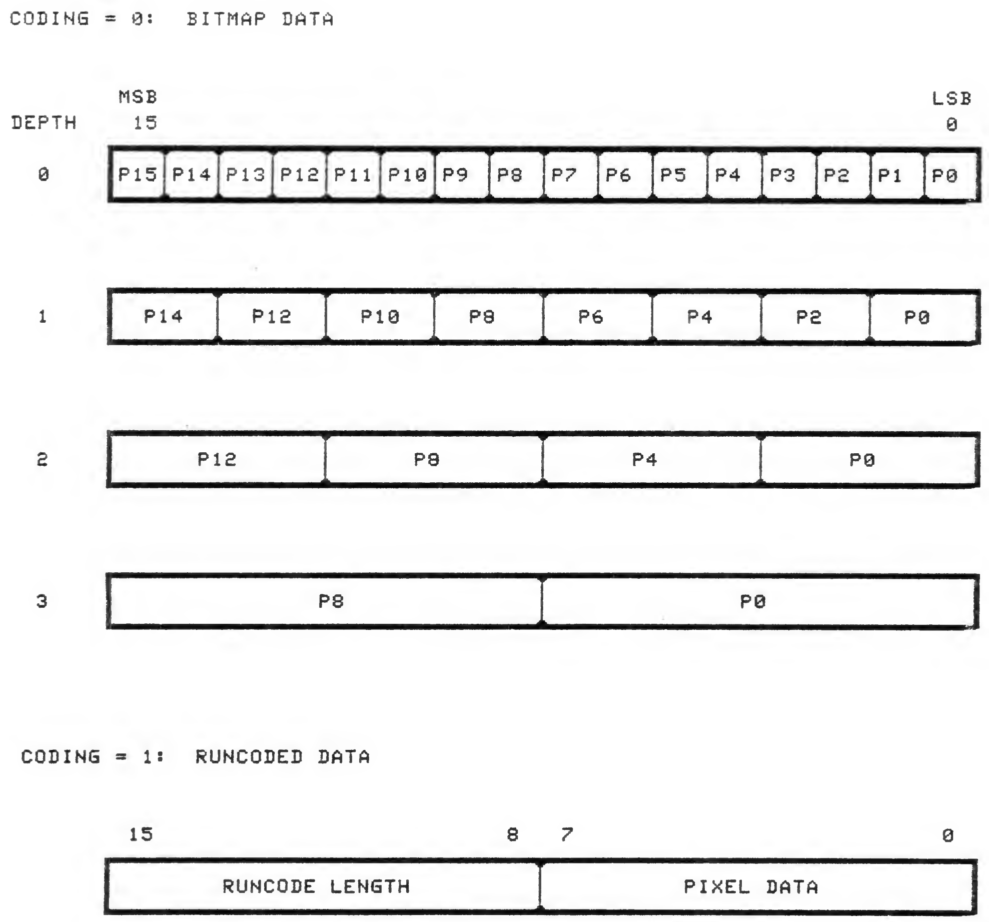

CODING = 0: BITMAP DATA
MSB
LSB
DEPTH
15
0
0
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
1
P14 P12 P10 P8 P6 P4 P2 P0
2
P12 P8 P4 P0
3
P8 P0
CODING = 1: RUNCODED DATA
15
8
7
0
RUNCODE LENGTH
PIXEL DATA

Figure 2-5: Formats of picture data arrays

2.8.2 Flags

Besides the color values, the color map may hold bits in each location for flags. One of the flag bits is to be used by external circuitry to enable external video data, so that Rainbow-generated and external images (from a video disk, for instance) can be combined on a pixel-by-pixel basis. Other flags may be used to enable external texture-generation signals or smoothing filters.

2.9 Global control modes and values

2.9.1 Background color

Rainbow generates a background color if no object displays. The background is the color addressed by the background register.

2.9.2 Interrupts

Rainbow generates interrupts for the conditions Programmed Line, Buffer Incomplete, and Bad Address. A program can get an interrupt each field by setting the Programmed Line Interrupt for the last line of the field. Individual interrupts are enabled by setting bits in the interrupt status/mask register. Reading the status/mask register returns a byte with bits set corresponding to interrupts that occurred. The interrupt status/mask register bit allocations are shown in Figure 2-6.

```
9   8   7   6   5   4   3   2   1   0
|___|   |   |   |   |___|   |   |   |_ Programmed Line Mask
 N/A    |   |   |    N/A    |   |_____ Buffer Incomplete Mask
        |   |   |           |_________ Bad Address Mask
        |   |   |
        |   |   |_____________________ Programmed Line Status
        |   |_________________________ Buffer Incomplete Status
        |_____________________________ Bad Address Status
```

Figure 2-6: Interrupt mask register bit allocations

Most sixteen-bit CPUs accept an eight-bit vector during interrupt acknowledgement. This vector directs the CPU to the appropriate interrupt service routine. The interrupt vector is created by logically ORing an interrupt vector register with a value representing the type of interrupt. Rainbow generates one of three different vectors depending on the value in the interrupt vector register. Bit zero of the interrupt vector is always set to zero.

The value of the interrupt type ORed with the vector register is:

Programmed Line Interrupt:	0
Incomplete Interrupt:	2
Bad Address Interrupt:	4

2.9.2.1 Programmed Line Interrupt

The main CPU can be interrupted on an arbitrary line number, which helps synchronize the CPU with video rate. The CPU is interrupted every field at a specific line number. Line number zero is the first line displayed at the top of the screen; the last displayed line is 479. Even numbered lines occur in the even field, and odd numbered lines in the odd field. The line number for this interrupt is a ten-bit read/write register.

Because of interlace, if the line is even-numbered, the Programmed Line Interrupt occurs on the specified line in the even field and on the next line in the odd field; if the line is odd-numbered, the Programmed Line Interrupt occurs on the specified line in the odd field and on the next line in the even field.

The Programmed Line interrupt will occur at the beginning of the specified line. It can be used as a "field interrupt," by setting it for line 480. This will interrupt after the last displayed line of every field.

Programmed Line interrupt may be set for a line number greater than the last visible line, and will then occur within the vertical blanking interval. Any line number up to the last line of the frame, which will be a blanked line, may be used. NTSC-format signals have 525 lines per frame, so to get an interrupt immediately before the start of each field one can set Programmed Line Interrupt for line number 523.

2.9.2.2 Incomplete Interrupt

Rainbow will generate an interrupt whenever it is unable to finish generating a line of color data in the time available for that line.

2.9.2.3 Bad Address Interrupt

Rainbow will generate an interrupt whenever it attempts a memory read cycle that is not completed in a reasonable amount of time. (Reasonableness of time is determined by timing hardware outside of Rainbow itself.) This will usually be caused by an erroneous Origin or Link value. The object processor responsible for the unanswered memory cycle will be shut down, regardless of interrupt masking, until restarted by a root write sequence.

2.9.3 Non-Interlace

Rainbow will generate interlaced video signals for full vertical resolution on normal (NTSC and monitor) displays. There is provision for use in non-interlaced systems: if the Non-Interlace bit is set, Rainbow generates even fields only. That is, every field is identical (rather than alternating between even and odd), and is what would have been generated for the even fields in interlace mode. Rainbow will operate initially and by default in interlace mode (that is, until Non-Interlace is explicitly set).

2.9.4 Current Line value

Rainbow maintains a read-only location that gives the current line number when read. The line number is even on even fields, odd on odd fields, and continues counting through the blanking time (that is, it is reset only when a field is about to begin). Current Line has several software uses: it tells how many line times remain in the blanking time (or any other state), its low bit distinguishes even from odd fields, and it can be used to determine safety of link changing.

2.9.5 Margin value

As an adjunct to the Incomplete Interrupt, Rainbow keeps a read-only location that tells how much time (measured in pixel clocks) was left when the previous line buffer was completed. This can be used as advance warning of impending incomplete interrupts, or as a measure of the complexity of an image. This location will be zero following a line which causes or would cause an Incomplete Interrupt.

2.9.6 Video timing and control

Rainbow supports software-adjustable video signal timing. Each video line has the number of pixel clocks of active video given by the Horizontal Line Size parameter. The actual line length is constrained to be a multiple of four pixels. The horizontal front porch interval is given by the Horizontal Front Porch parameter. The back porch is a minimum of thirty-two pixel clocks long; the length in excess of this is given by Horizontal Back Porch Extension.

Each field has at least 192 lines of active video; more may be set via the Vertical Active Video Extension parameter. The length of the vertical front porch in line times is given by the Vertical Front Porch parameter. The length of the vertical back porch is given by the Vertical Back Porch parameter.

Use of these paramters is somewhat arcane, and it is expected that each product that incorporates Rainbow will establish a set of parameters to be used for that product. Thus game or application programmers will never have to deal with them.

2.10 Addresses of Rainbow registers

Rainbow occupies 544 bytes of bus address space plus object-processor root locations. The color map resides in the first 512 bytes and contains the actual color information. The next sixteen word addresses are reserved for global registers. Root locations for object processors begin at location 544 with the lowest-priority object processor and extend as far as needed. With 24 object processors, the highest-priority one will have root addresses 636-638.

ACCESSIBILITY	9	8	7	6	5	4	3	2	1	0	BYTE ADDRESS
R/W			BACKGROUND								512~513
R/W			INTERRUPT MASK								514~515
R			INTERRUPT STATUS								516~517
R/W	PROGRAMMED LINE INTERRUPT										518~519
R	CURRENT LINE										520~521
R	MARGIN VALUE										522~523
W			HORIZONTAL ACTIVE VIDEO EXTENSION								524~525
W	HORIZONTAL LINE SIZE										526~527
W					HOR. BACK PORCH EXTENSION						528~529
W	NRF	NIN	VERT. ACTIVE VIDEO EXTENSION								530~531
W					VERT. FRONT PORCH						532~533
W					VERT. BACK PORCH						534~535
R			RANDOM NUMBER GENERATOR								536~537

** R/W = READ/WRITEABLE

** R = READABLE

** W = WRITEABLE

Figure 2-7: Rainbow register addresses

3. Hardware implementation

The Rainbow video system functions are partitioned into two IC designs. The first IC is the "Gold" chip which provides circuitry for the video timing, color map, interrupts, background logic and line buffers. The second IC is the "Silver" chip which contains circuitry to read pixel data from memory, process the information and send the data to the Gold chip. Each Silver chip contains twelve Object Processors. A full Rainbow system will include one Gold and two Silver chips. An application of Rainbow is shown in Figure 3-1.

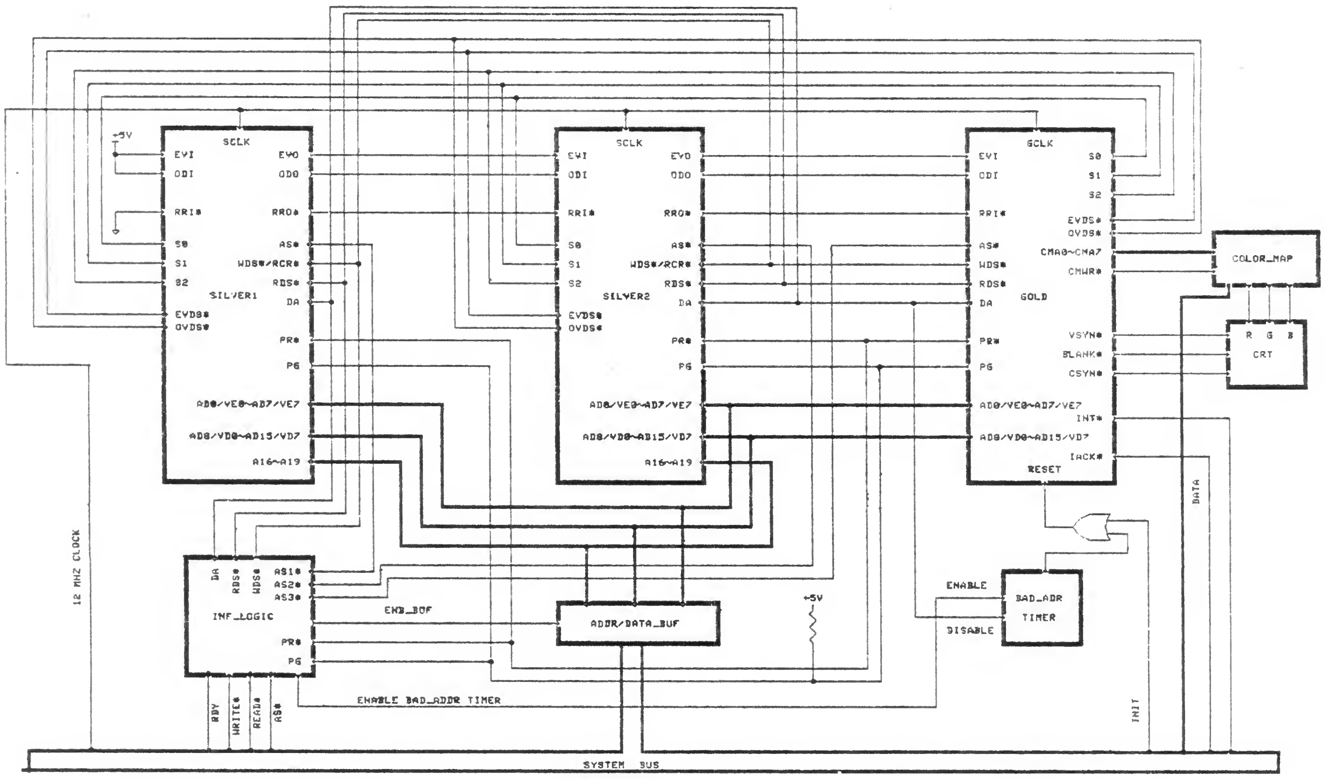
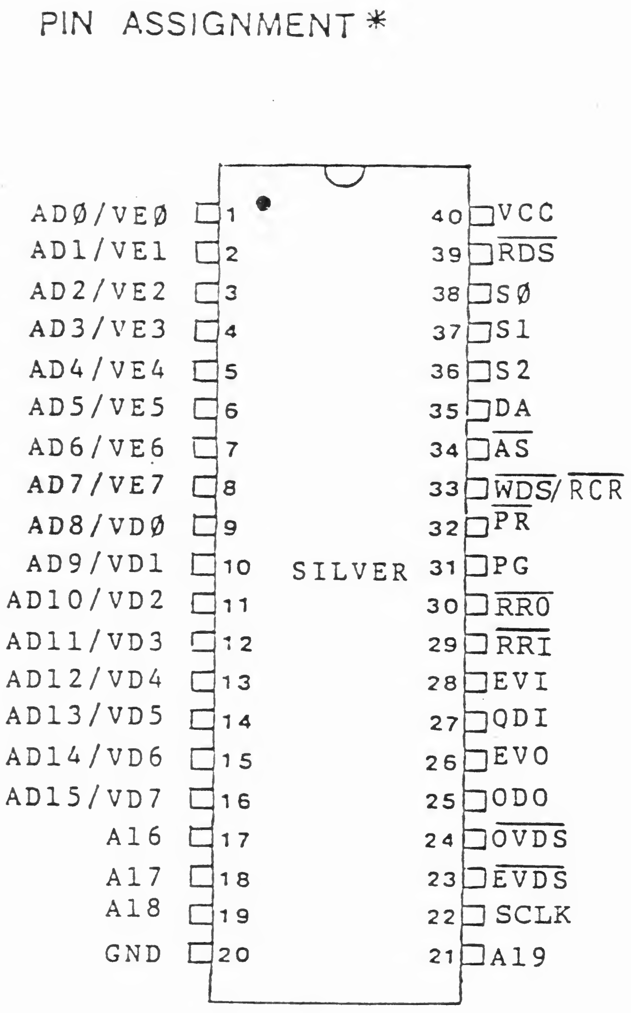


Figure 3-1: Rainbow system connection



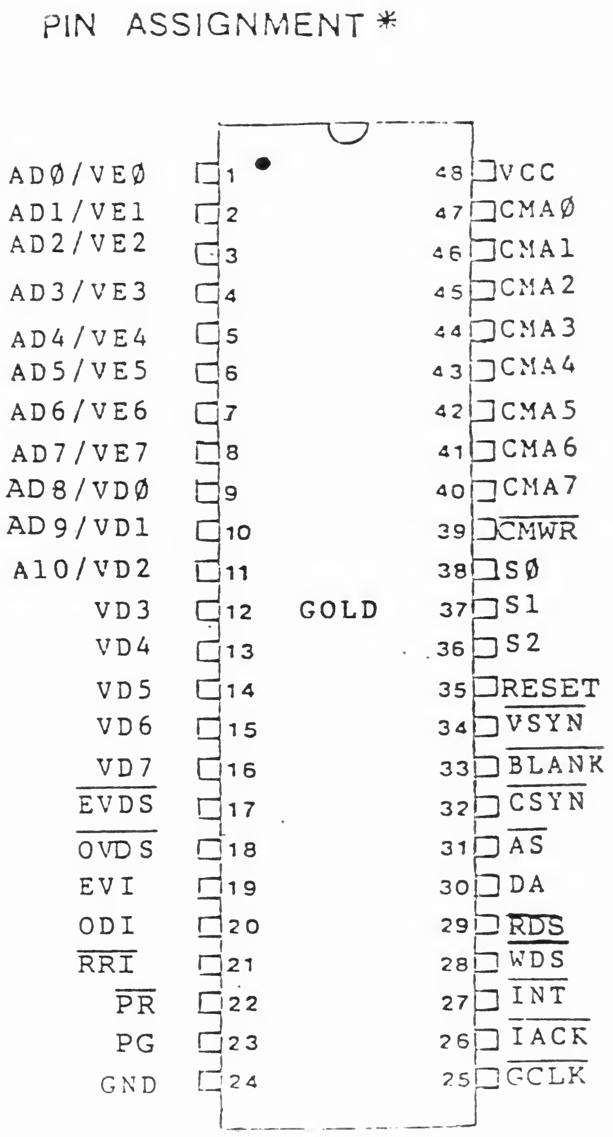


Figure 3-2: Rainbow pinouts

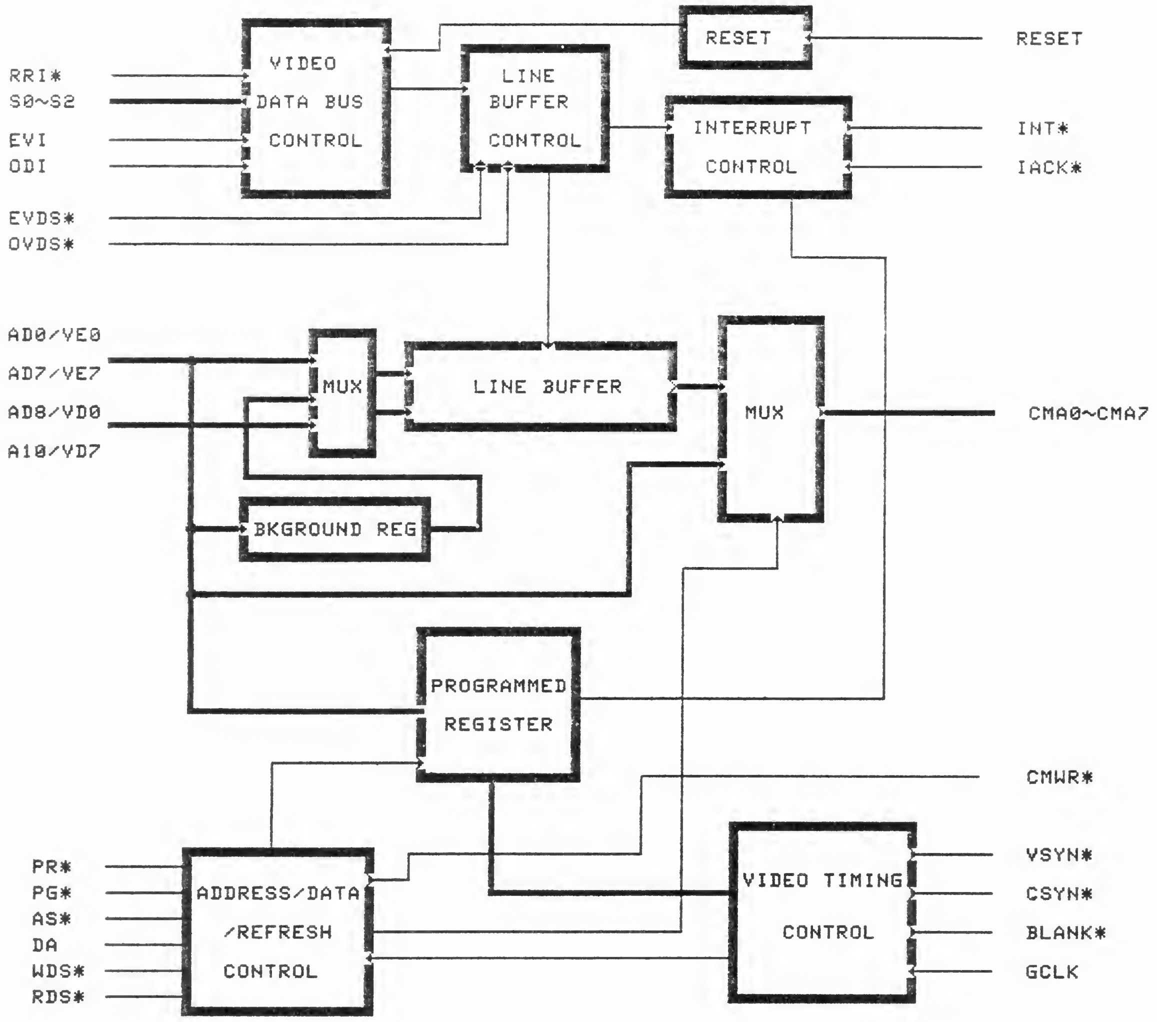


Figure 3-3: Gold chip block diagram

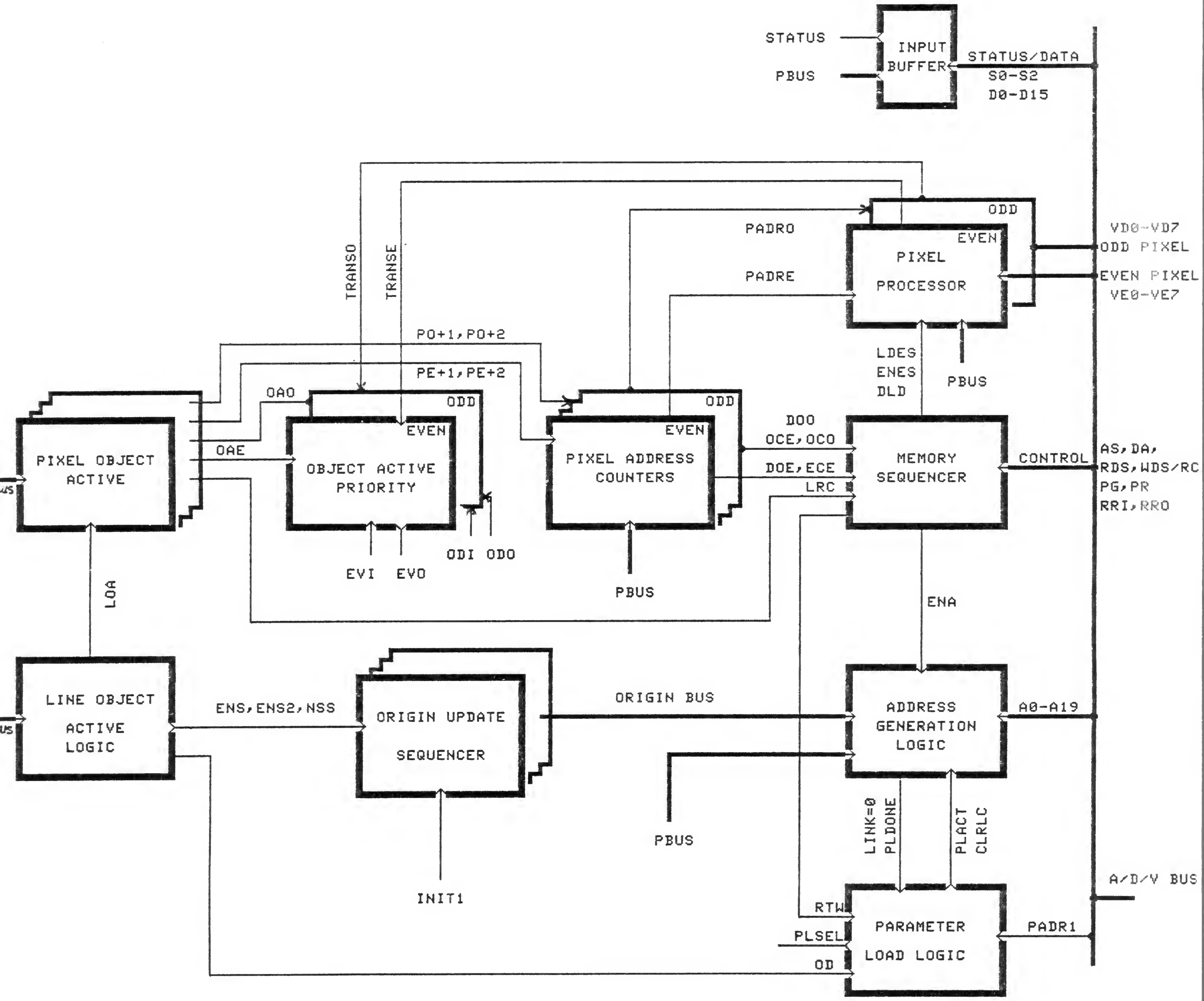
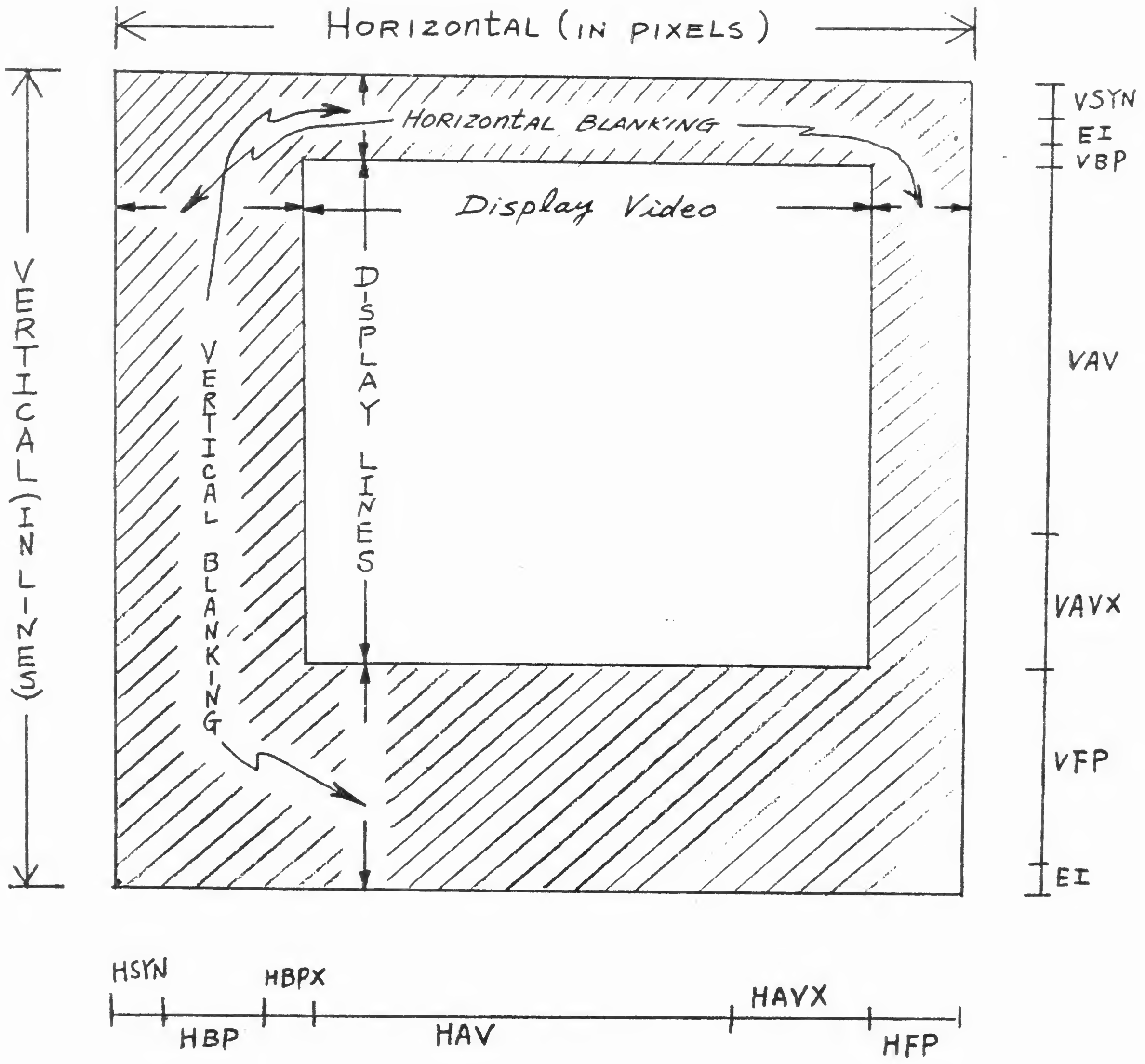
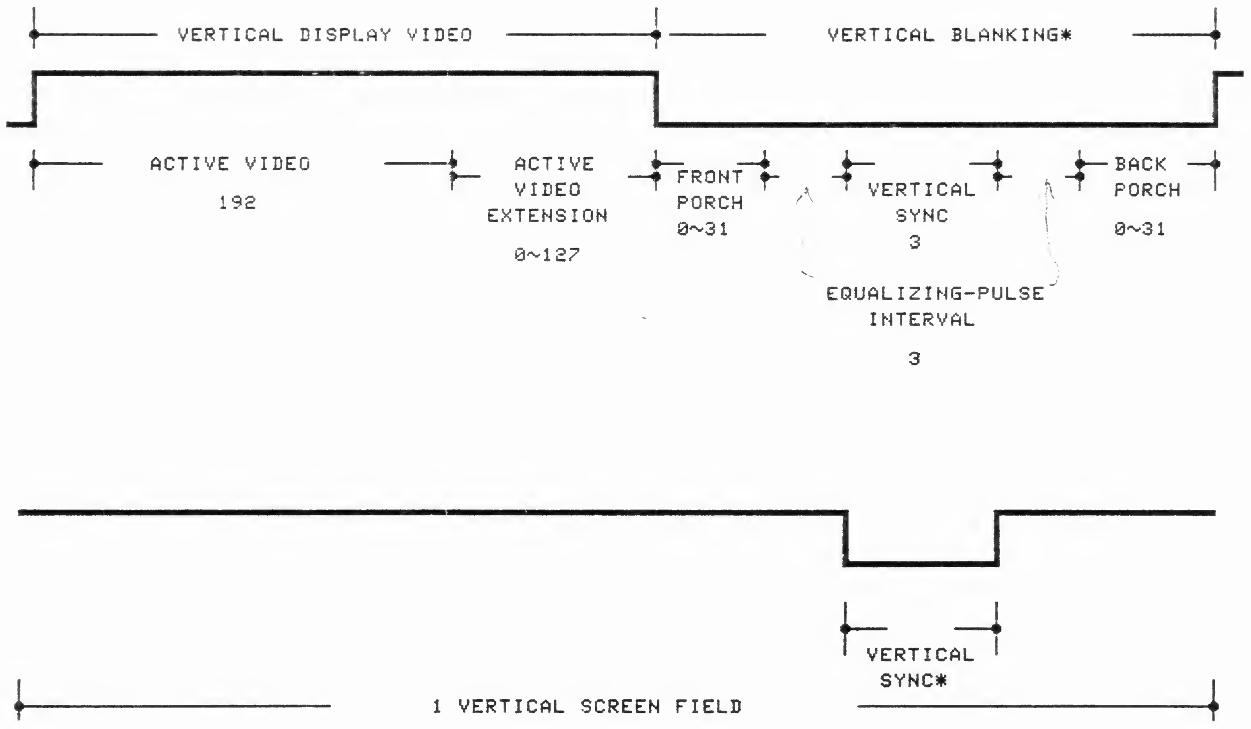


Figure 3-4: Silver chip block diagram



** Here notations HAVX, HLS, HBPX, VAVX, VFP and VBP for programmable registers indicate which part of picture they control.

Figure 3-5: Screen format



** THE NUMBERS SHOWN IN THIS FIGURE IS FOR EACH FIELD AND NUMBERS IN PROGRAMMED REGISTER IS FOR A FRAME.

Figure 3-6: Vertical video timing

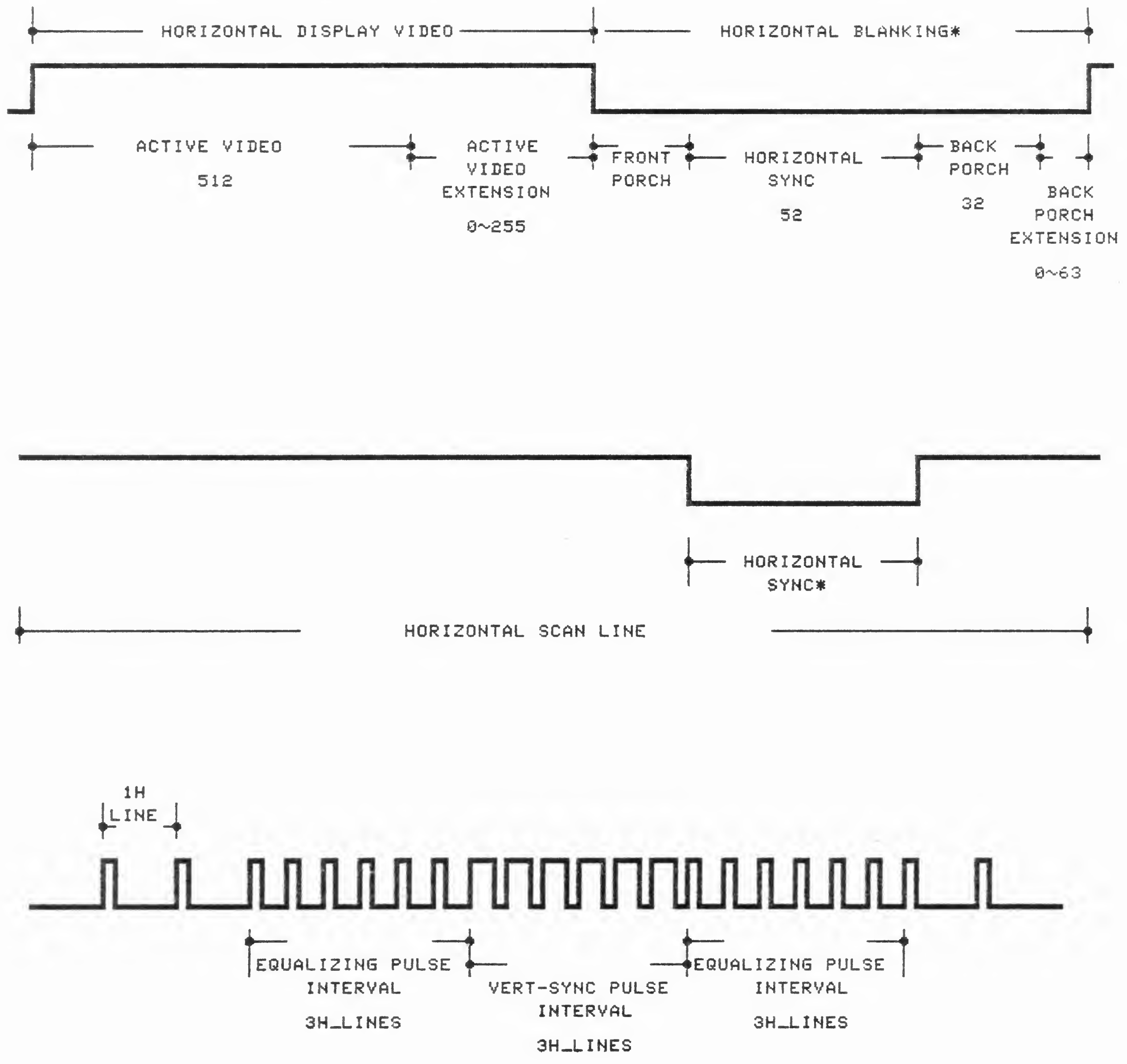


Figure 3-7: Horizontal video timing

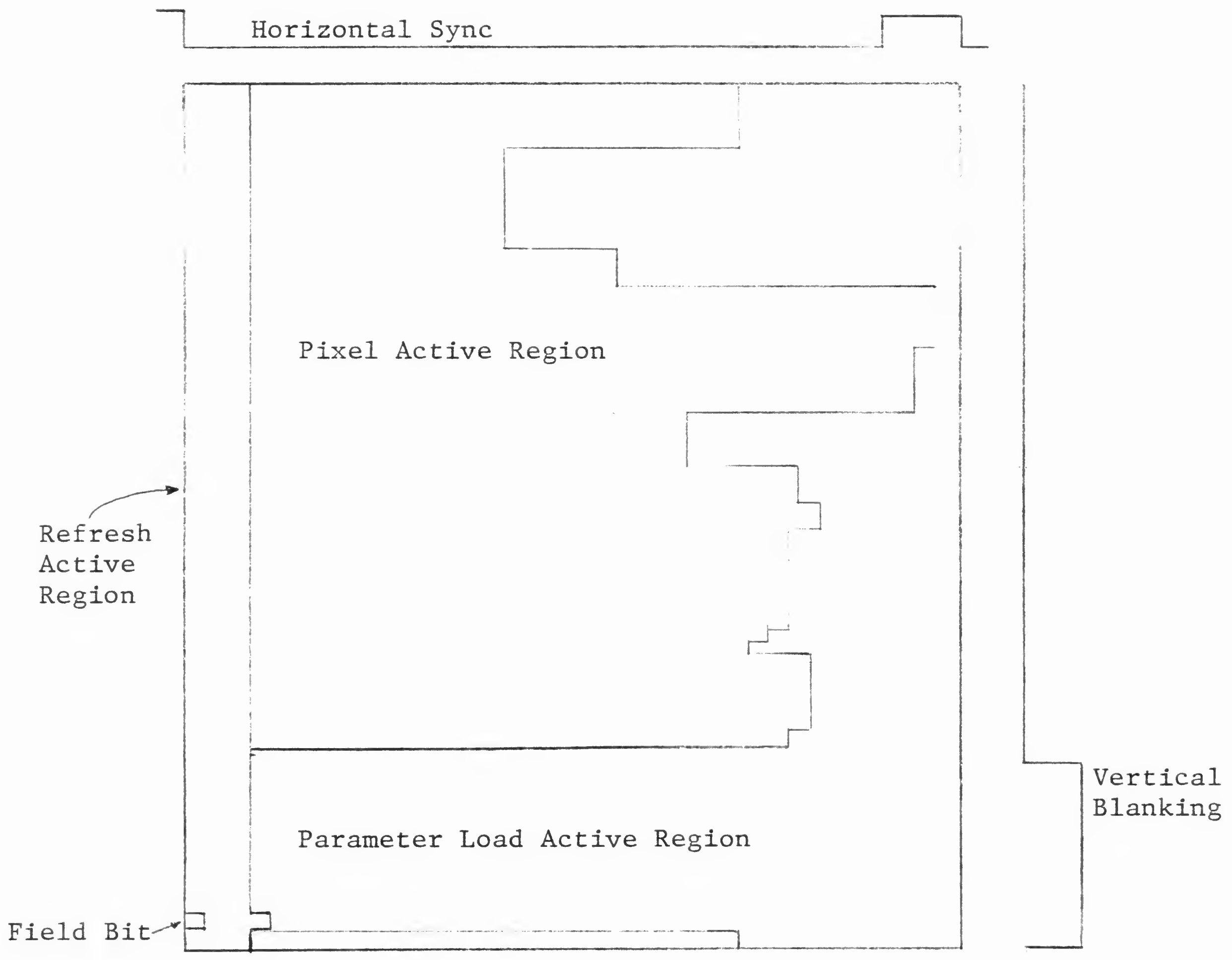


Figure 3-8: Rainbow status timing

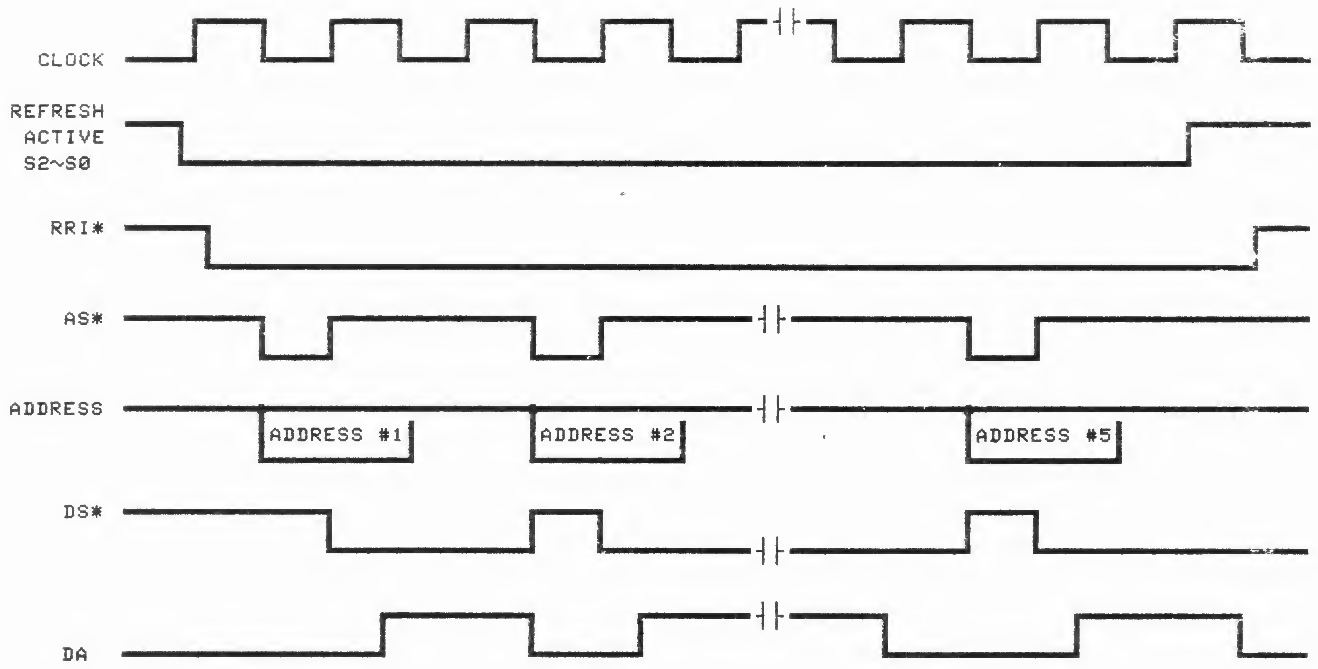


Figure 3-9: Refresh cycle

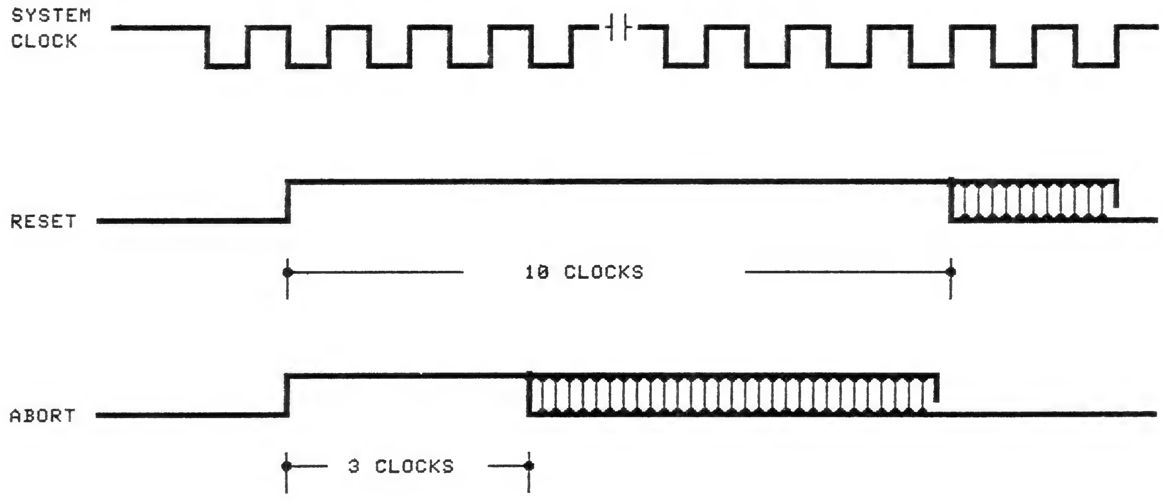


Figure 3-10: Reset sequences

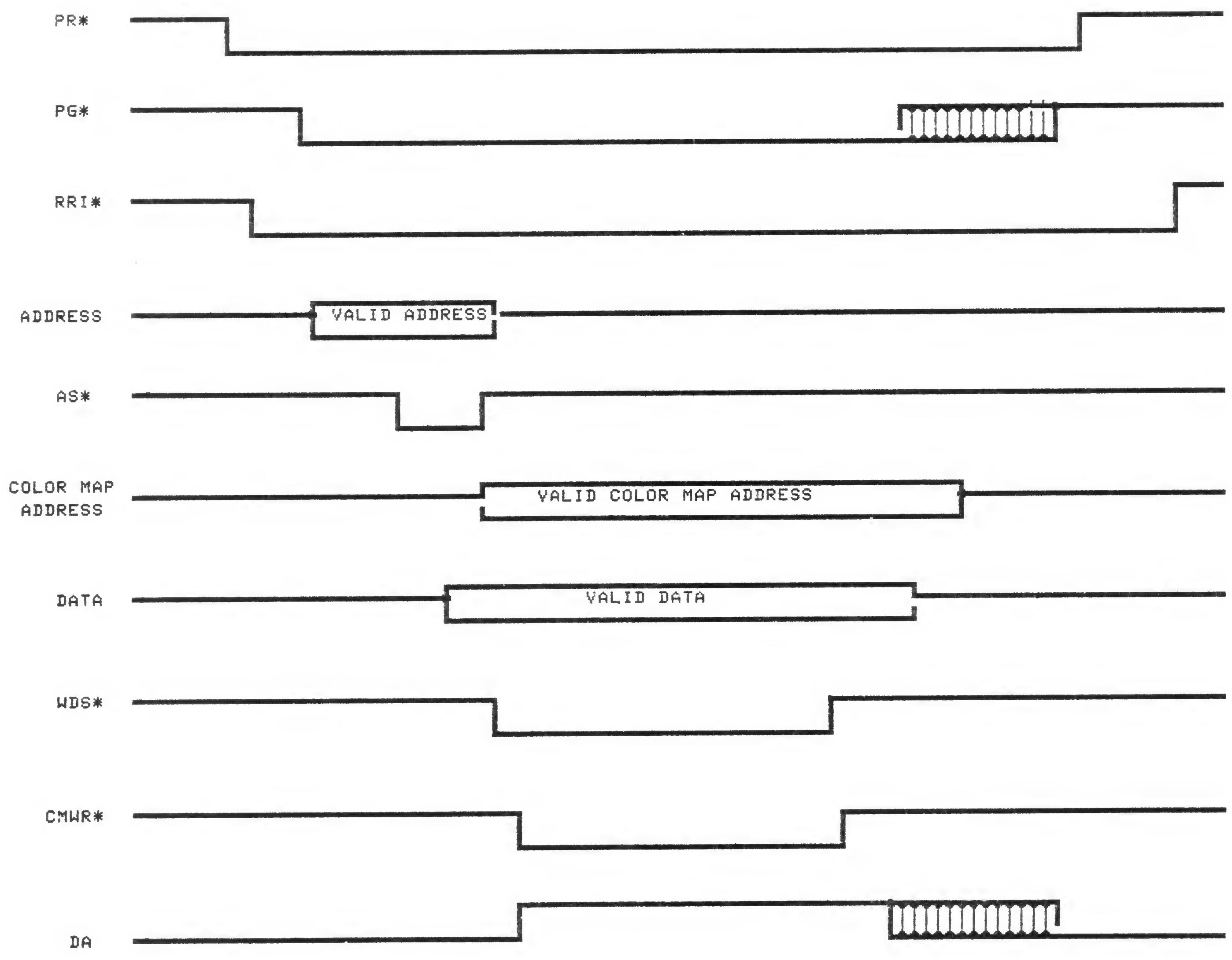


Figure 3-11: Color map write

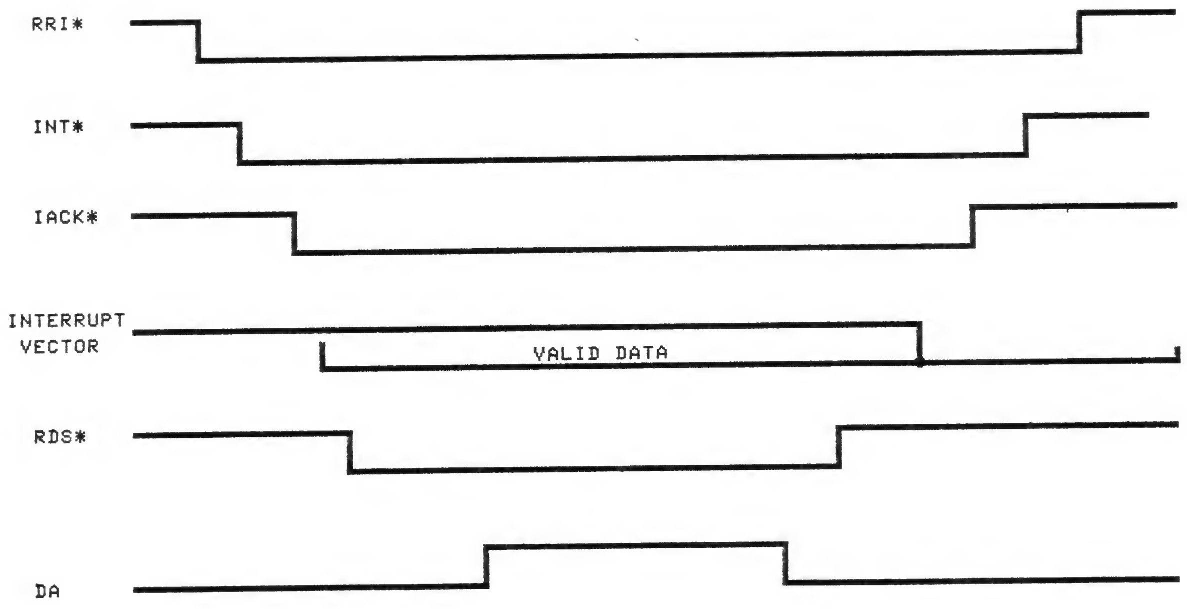


Figure 3-12: Interrupt timing

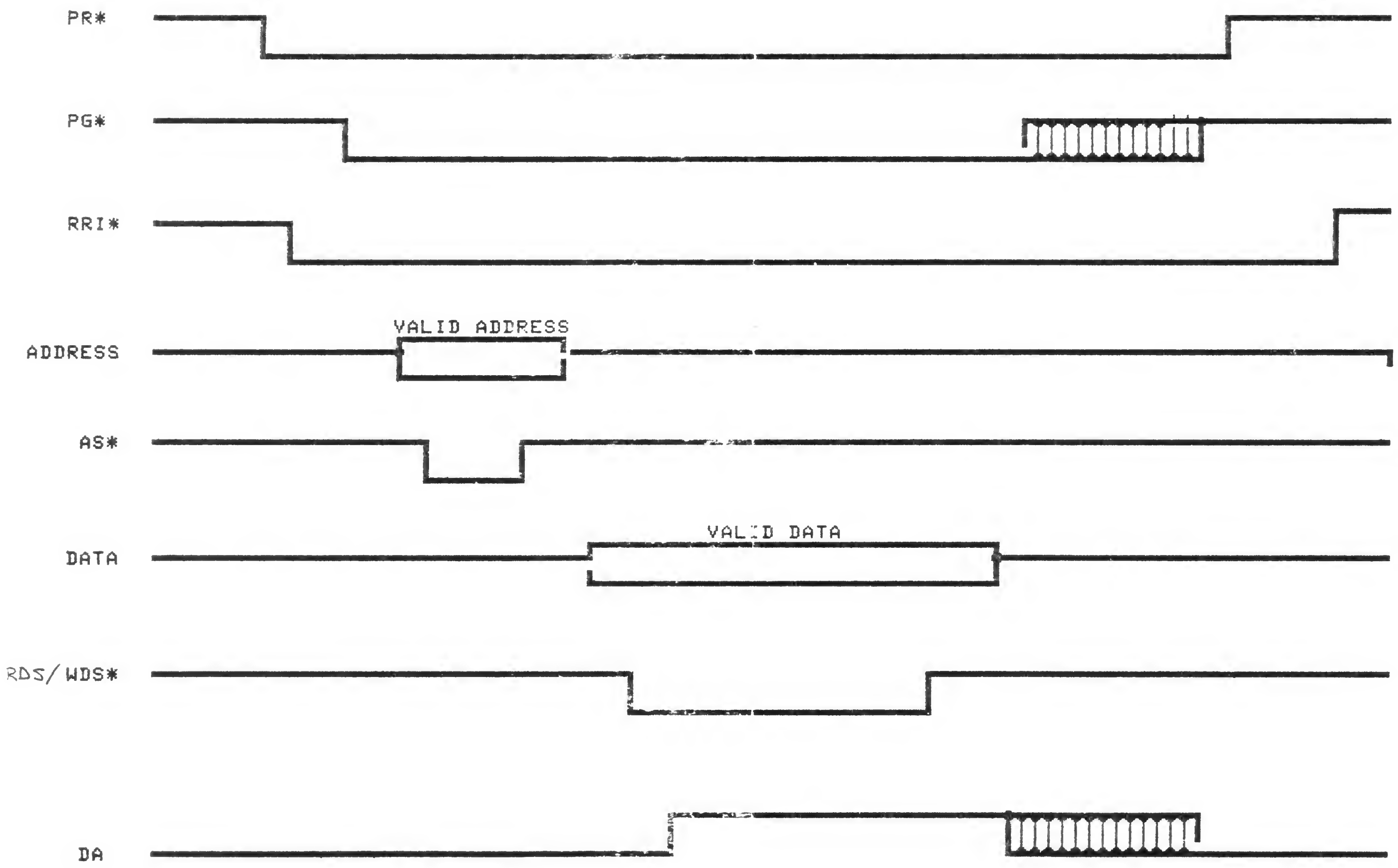


Figure 3-13: Read/write cycle

4. Performance of Rainbow

The graphic performance of Rainbow, as measured by the number, size, resolution, and color range of objects it can display, is sensitively dependent on the effective speed of processing pixels. This in turn depends on the effective performance of the external memory holding Rainbow's instructions and data, as well as on the Rainbow internal cycle time.

Rainbow will operate with a 12 Mhz clock, and will be capable of computing (in pipeline fashion) two pixels in each 83.3 ns clock period, if the data for these two pixels is already on-chip. Transparent objects also require one extra Rainbow clock each to resolve. If all required data has not already been fetched, Rainbow must access memory before delivering the pair of pixels.

4.1 Speed and available cycles

Because screen resolution, color resolution, transparency, and number of objects active are independently determined, the Rainbow hardware does not guarantee that every representable screenful is in fact displayable. The chief limitation is memory bandwidth: the number of bits that must be fetched to complete the description of a display screen. Some illustrative (and perhaps typical) cases follow.

Assume that system memory cycles in 280 ns, that system memory access takes 120 ns from request, and that Rainbow requires two clocks to do a memory fetch cycle. We know that a line time in NTSC is 63.5 us.

Just generating 640 pixels of background color (requiring no fetches), plus refreshing memory, will consume 27 us, leaving 36.5 us to fetch any object pixel data that will be used on the line. This is about 130 full cycles, or 304 access times (if we don't have to wait for precharge after each cycle).

Generating a full line from a "playfield" object takes:

Depth	cycles	time (us)	
3	320	53.1	(too long)
2	160	26.6	
1	80	13.3	
0	40	6.6	

Each skill-and-action player/missile of 40 pixel's width in full resolution will require:

Depth	cycles	time (us)
3	20	3.3
2	10	1.7
1	5	0.83
0	3	0.42

[This draft printed Wednesday 30 November 83 11:32; from SPEC.MSS last edited 30 November 1983 at 11:30.]