The RAINBOW Silver Chip Specifications

George Wang

Atari Semiconductor Group

Steve Saunders & Robert Alkire

Atari Sunnyvale Research Laboratory

January 23, 1984.

TABLE OF CONTENTS

TABLE OF CONTENTS (Con't)

TABLE OF CONTENTS (Con't)

TABLE OF ILLUSTRATIONS Page Number

TABLE OF ILLUSTRATIONS(con't) Page Number

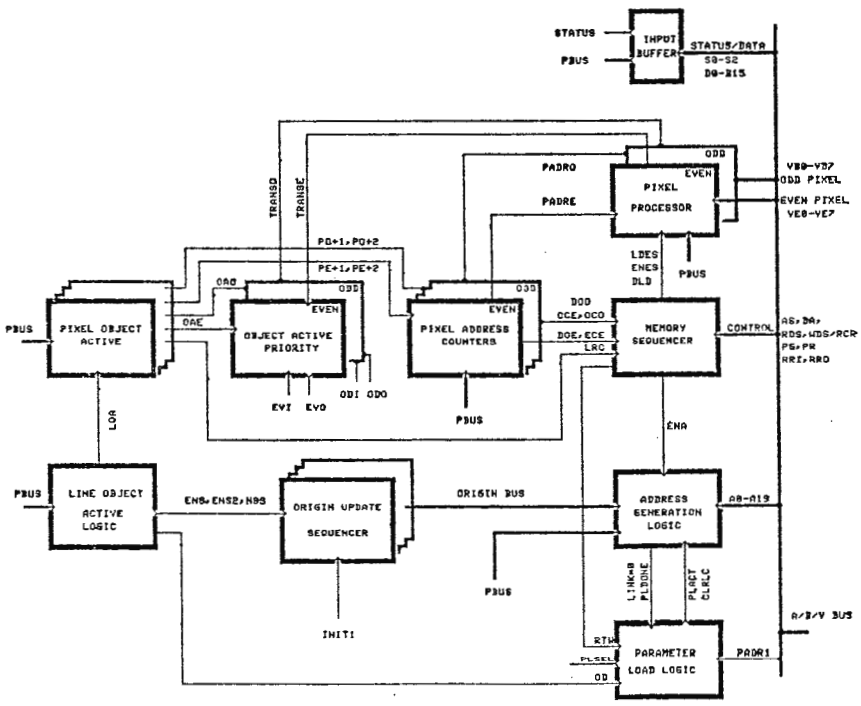
1. GENERAL DESCRIPTION

The Rainbow-Silver chip contains eight Object processors. Each Object processor is responsible for requesting information from video memory, processing it and delivering it line by line to the Gold chip. All Object processors are functionally identical with only different display priorities. To perform the graphics display capability it requires that the Rainbow-Silver chip works with the Rainbow-Gold chip together.

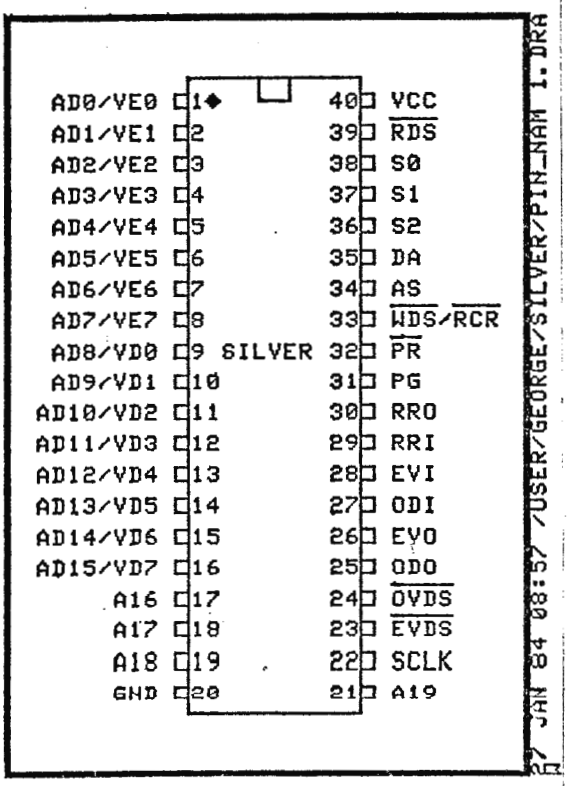
2. FEATURES

@ ~~Implementing~~ CMOS technology.
@ Access to 1M bytes of memory over 16 bit bus.
@ On-chip Direct Memory access functions.
@ Object Transparency.
@ Modular hardware allows easy expansion.
@ Independent Video/CPU clocks
@ Independent Horizontal and Vertical magnification.
@ Minimal CPU intervention.
@ Simple programming model for parameter and display data.

3. BLOCK DIAGRAM



4. PIN ASSIGNMENT



PIN DESCRIPTION

Pin Name	Type	Pin#	Function
AD0/VE0 AD7/VE7	I/O	1-8	These lines constitute the time-multiplexed Address bus A0 to A7, Data bus D0 to D7 and Video Even pixel bus VE0 to VE7. The Address bus and Data bus are valid inputs when the Processor Grant pin PG is LOW. The CPU uses these lines to to write data into the Silver chip. While the status pins S0 to S2 indicate Pixel Active or Link Load Active conditions, the Address bus will be used as outputs to fetch pixel data or parameter block into the Silver chip. Video Even pixel bus lines VE0 to VE7 are valid outputs when the Status lies S0 to S2 indicate video pixel condition being active.
AD8/VD0 AD15/VD7	I/O	9-16	These lines constitute the time multiplexed Address bus A8 to A15 Data bus D8 to D15 and Video Odd Pixel bus VD0 to VD7. The function and validility of the Address bus and Data bus are described in the above section. Video Odd pixel bus lines are valid outputs when the Status pins S0 to S2 indicate video pixel condiction being active.
A16-A18 A19	I/O	17-19 21	These Address bus are inputs when the Processor Grant pin PG is LOW. While the Status input pins S0 to S2 indicate Pixel active or Link Load active conditions, these Address bus will be used as outputs.
GND	I	20	Ground.

Pin Description(con't)

Pin Name	Type	Pin#	Function
SCLK	I	22	Silver Chip Clock; the clock provides the basic timing for the Silver chip.
EVDS	I/O	23	Even pixel video data strobe; if this Silver gets the even pixel priority EVI HIGH and the even pixel is active in this chip, then EVDS is asserted to LOW whenever this even pixel is processed and ready to be strobed. In addition, both EVDS and OVDS together provide a signal internally to inform the chip that it can proceed to process next pair of pixel. In most time, this is an input. Only when the EvI input is high and the priority is not passed to the next Silver chip, then this pin become an output.
OVDS	I/O	24	Odd pixel video data strobe; if this Silver gets the even pixel priority ODI HIGH and the even pixel is active in this chip, then OVDS is asserted to LOW whenever this odd pixel is processed and ready to be strobed. In addition, both EVDS and OVDS together provide a signal internally to inform the chip that it can proceed to process next pair of pixel. In most time, this is an input. Only when the ODI input is high and the priority is not passed to the next Silver chip, then this pin become an output.
ODO	O	25	Odd pixel priority out; when this output is HIGH, the Silver chip passes its odd pixel priority to the next chip which has an input ODI connected to this output.

Pin Description(con't)

Pin Name	Type	Pin#	Function
EVO	O	26	Even pixel priority out; when this output is HIGH, the Silver chip passes its Even pixel priority to the next chip which has an input EVI connected to this output.
ODI	I	27	Odd pixel priority in; when this input is HIGH,it means the Silver gets priority to process the odd pixel if any object is active in the chip. Otherwise, it will pass the priority to the next chip through the ODO pin.
EVI	I	28	Even pixel priority in; when this input is HIGH, it means the Silver chip gets the priority to process the even pixel if any object is active in the chip. Otherwise, it will pass its priority to the next chip through the EVO pin.
RRI	I	29	Video bus Release In(includes both odd and even pixel bus); the HIGH level of this input indicates the Video bus is released by the high priority Silver chip and this chip can access memory system through the Video bus.
RRO	O	30	Video bus Release Out(includes both odd and even pixel bus); the HIGH level of this output indicates the the video bus is released by this chip and granted it to the next chip whose input pin RRI connected to this output.

Pin Description(con't)

Pin Name	Type	Pin#	Function
PG	O	31	Processor Grant; when the Processor Request PR is LOW and this Silver chip is not using the Video bus, then the Silver chip can set this output HIGH to grant this bus to the CPU.
PR	I	32	Processor Request; when this input is LOW, it indicates the CPU wants to access the Programmable registers in the Silver chip.
WDS/RCR	I/O	33	Write Data Strobe/Run-length Coding Request; CPU may use this input to to write data into the Silver chip such as reading data to the Parameter Block. However,this input is used as a Write Data Strobe only when both Processor Request PR and Processor Grant are LOW. Otherwise, this input is used for Run-length Coding Request. When the Silver chip does not have priority to access the system bus but need to fetch new Run-length coding data, it can assert this output to LOW to inform the higher priority Silver to pass its priority to the lower priority Silver chip when it finish using of system bus.
AS	I/O	34	Address Strobe; This signal is used as an input to indicate that there is a valid address on the Address bus. When the Silver chip performs memory fetch, it is used as an output to strobe the valid address on the system bus.

Pin Description(con't)

Pin Name	Type	Pin#	Function
DA	I/O	35	Data Acknowledge; it is used as an output to inform CPU that data transfer is completed. It is used as an input to indicate that a memory read cycle is finished and valid data is on the Data bus.
S2-S0	O	36-38	These status lines from the Gold chip provide infromation to the Silver chip as follows:
RDS	I/O	39	Read Data Strobe; the Silver chip will use this pin as output to read video data or parameter block from memory into itself. CPU may use this input to read data from the Silver chip such as data in the Link register. However,this input is used as a Read Data Strobe by CPU only when both Processor Request PR and Processor Grant PG are LOW. Otherwise, this pin is an output most times.
VCC	I	40	VCC is the +5V power supply.

S2	S1	S0	Description
0	0	0	Refresh Active
0	0	1	No operation
0	1	0	Abort memory cycle
0	1	1	Reset
1	0	0	Top of screen in Even field
1	0	1	Top of screen in Odd field
1	1	0	Pixel Active
1	1	1	Link Load Active

6. FUNCTIONAL DESCRIPTIONS

6.1 Overview

The Rainbow video graphics system is object based. All display activity is described in movable, variable-size, multimode objects. An object is like a sprite, but more general, like a combination of Antic's playfield and players. Objects are not limited in size, and can be reused in vertical sequence. Thus the display horizontal complexity is limited to the number of hardware objects supplied in the Silver chip(8 object processors are contained in the chip now), but is not limited vertically.

The screen is organized as an array of square pixels. The standard resolution for NTSC display is 640 horizontal by 480 vertical. All graphics and text fonts are represented in term of pixels of this size. The reduced space-resolution modes may turn out to be far more often used, but this fine grained description is chosen as a standard that should cover all our "standard-video" needs, on NTSC, PAL and RGB monitors, for some time.

The video system can produce up to 256 different colors from a total palette of 4096(16 levels of gray).

The Rainbow chip set performs the function of translating from digital representations of graphics into the time-sequenced signals necesary to drived a raster-scanned CRT. It produces interlaced RGB digital video outputs. Three four bit D/A converters are needed to interface the chip set to RGB monitors. Rainbow can be used with RGB to NTSC/PAL conversion devices, e.g. National LM1886 & LM1889 chip set, to produce composite video signals.

The video system communicates with memory over an asynchronous 16 bit bus as bus master. The chip set can take advantage of page mode feature in dynamic memories to improve memory access speed and increase the bus bandwidth. For dynamic memory refresh requirements, Rainbow will assert refresh request and row address counter.

Each object processor in the Silver chip contains a parameter block and a pointer to video memory of the object representation. The video system can cause interrupts to main CPU to indicate error condition such as "Line Incompleted" and the NON-error condition "Programmed Line".

6.2 General Terminology

Why not use This as General description on Pg7

Rainbow's domain of competence is translation of memory-represented graphics into screen-displayed graphics. The graphical space is "2 1/2" Dimension, i.e. many 2-D objects that move independently and overlay and obscure each other. The memory representation (called a picture) is a sampled pixel map (not, for instance, vectors or ploygons). The data in a source pixel represents not an absolute color but a selection from a color map. A portion of the picture, called the "window", is selected for displayed at any one time. The picture can be of arbitary size which may be much larger than the screen. The window can be the entire picture, or only a piece of it, located anywhere on the picture (e.g. not necessarily the upper left corner). Pixel data from the selected window is fetched and transformed by the "Object processor" into a stream of video data to be sent to the CRT. The fetching and interpretation of pixel data for an object is under the control of a "parameter block".

6.2.1 Picture data formats

The pixel data, in memory, are packed on word (16 bit) boundaries. The object procesor must unpack the memory words and use the resulting unpacked data to select a color from the color map.

Pictures are coded in one of two ways, "bitmap" and "runcode". Bitmap representation is the conventional one-to-one specification of pixel colors, while runcode representation can save much space and time for certain kinds of images.

INCONSISTANT STATEMENT

A bitmap picture is an array of bitfields, each one representing the color of a single pixel. Bitmap pixels are in raster scan order, left to right then top to bottom; an array represents a rectangular area.

A runcode picture is an array of bitfields, each one representing the color of a contiguous horizontal sequence (a "run") of pixels. Each runcode specifies both a color to use and the length of the run of pixels that are that color.

In Rainbow, bitmap pixels are either 1, 2, 4, or 8 bits wide; runcodes are 16 bits consisting of 8 bits color and 8 bits runlength.

6.2.2 Object priorities

A full screen Rainbow display is composed of objects. Rainbow will be capable of processing as many as eight objects on the same scan line as there are object processor modules included in the hardware configuration of the Silver chip. Object Processors can be vertically reused, and, will in fact, reload itself at the end of a window with the parameter list for its next object without CPU intervention.

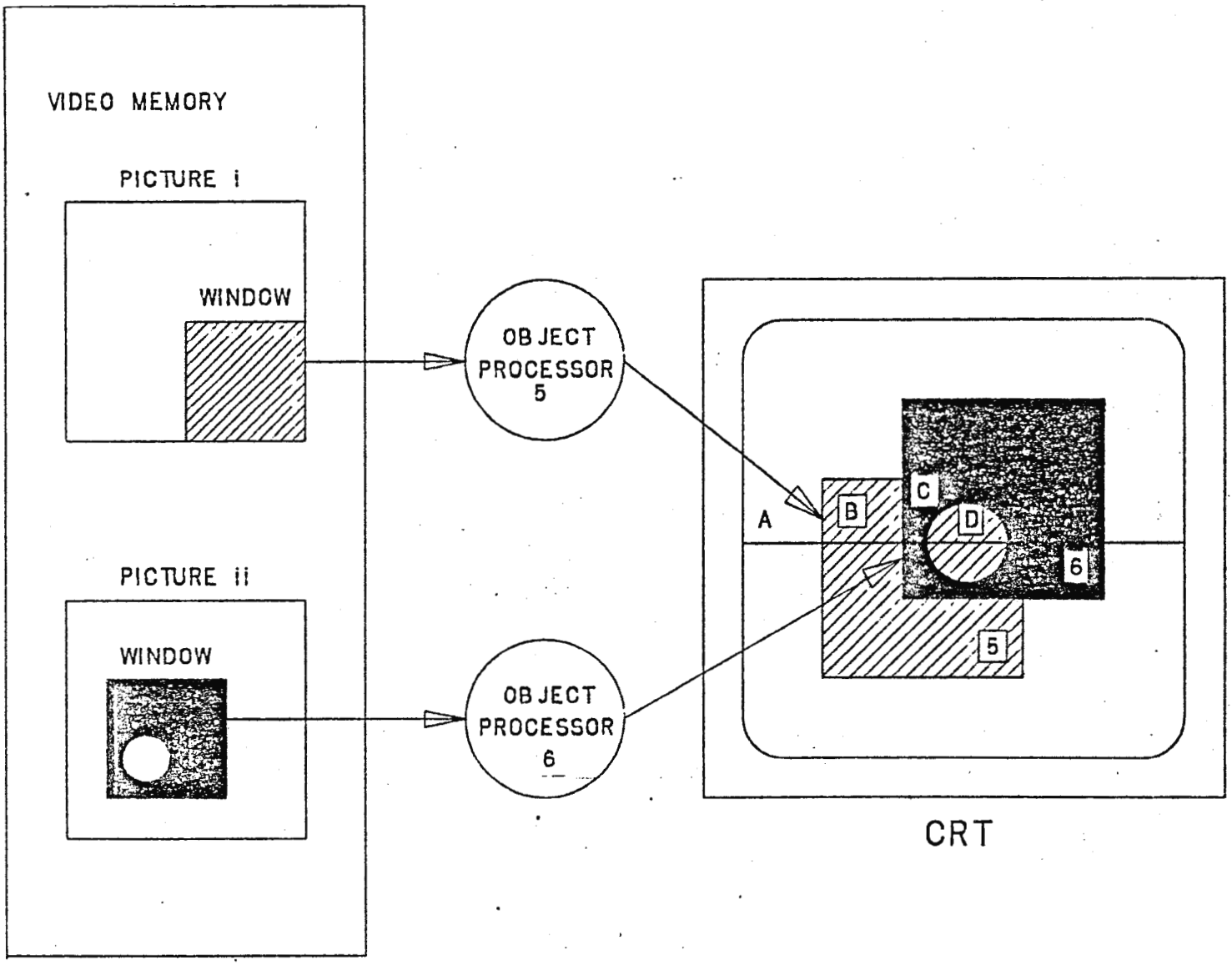
Typically one object will serve as the background analogous to ANTIC's "playfield", and others will be smaller moving players, missles, trees/faces, text areas, or whatever is required. If the screen is split into upper and lower sections, the background as well as the other objects may be reloaded at the boundary. The background will usually be the lowest priority object. Priority is in a fixed sequence among object processors. If priority among objects needs to be rearranged, the object parameters can be swapped between the object processors. This is accomplished by swapping the parameter block pointers in memory. Moving or copying the contents of the blocks is not required.

6.2.3 Object transparency

It is sometimes desirable to put "holes" into objects thus allowing objects normally obscured to appear through the holes. This feature is called transparency. Each object processor has a transparent parameter which can redefine the interpretation of the pixel data value so that instead of displaying a color, a hole is made in its place.

An example of transparency is shown in Figure 1 where two objects are partially overlapping. Object processor #2 is displaying a window from picture II and object processor #1 is displaying a higher priority image with transparency from picture II. At point A, no object is active so background is displayed. At point B, Object 2 becomes active and is displayed. At point C, Object 1 becomes active and will obscure Object 2 since it is of higher priority. At point D, Object 1 begins reading pixel data which indicates "transparent".

Figure 1. An example of Object transparency.



Object 1 then yields priority to others. In this case Object 2, which is the next lower priority object, begins displaying its own pixel data.

6.2.4 Object parameters

The parameters associated with the object's display representation in memory are Origin, Stride,X and Y, Pixel offset, Color index, Width, Length, coding, Depth Transparency, Scale X and Scale Y.

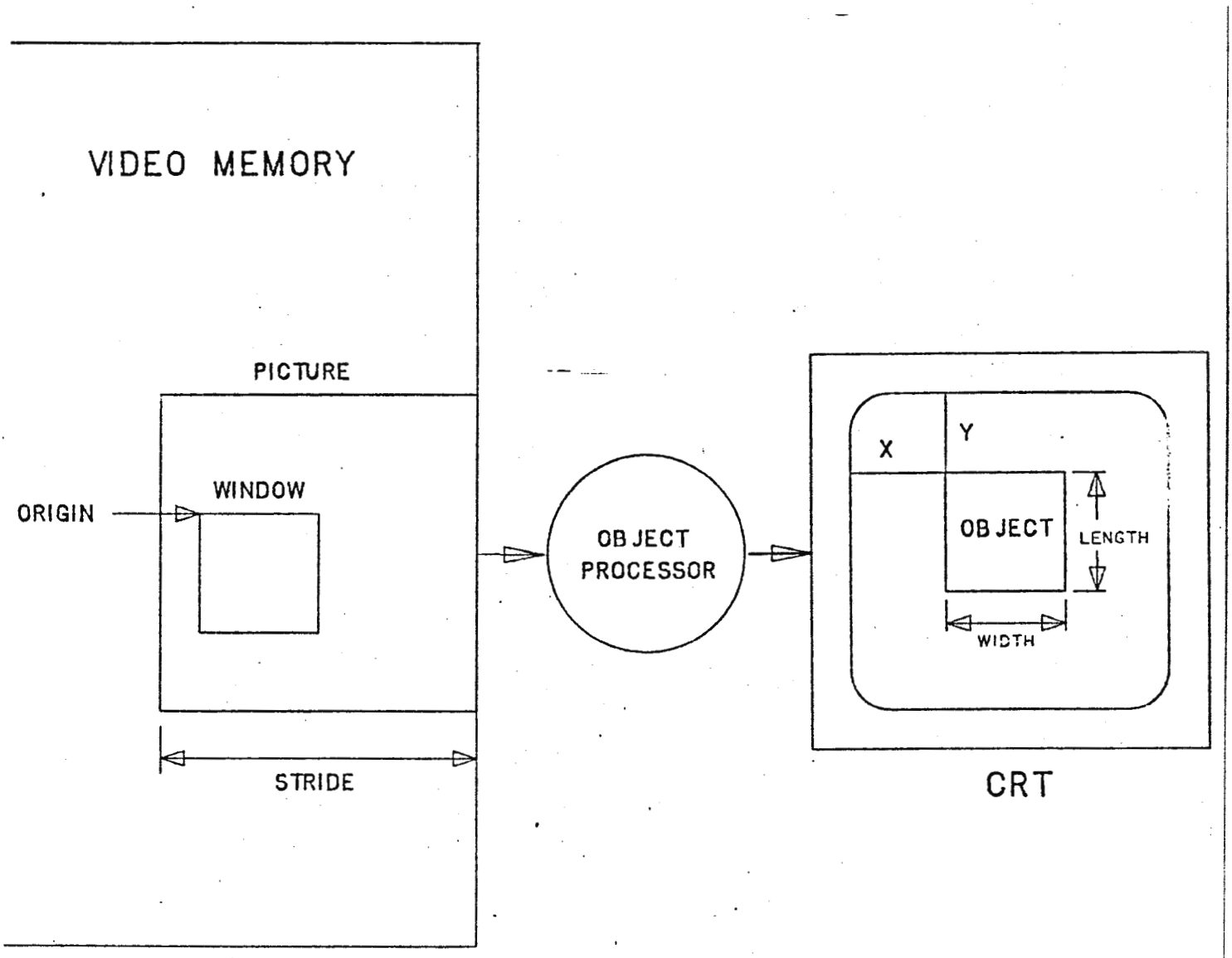
Origin is a bit of field address of upper left hand corner of a window. The Origin value is a 20 bits address (A0 = 0) pointed to a memory word and pixel offset value gives the exact starting position of the pixel in this word. Stride is the number of words of memory between data for a pixel and data for the pixel to be displayed directly below it. Stride's value can be said to be the width of the object's picture in words. 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 shows an example of the Origin and Stride.

Width and Length describe the rectangular dimensions of the object. The width is the horizontal size of the Window measured in scren pixels. Length is the vertical size of the Window measured in screen lines. Depth specifies the number of bits that make up each source pixel in bitmap coding, but not used in runlength coding.

The position of the object on the screen is given by the X and Y parameters. The X parameter is the number of screen pixels from the left edge of the screen to the left edge of the object. The Y parameter is the number of the scan lines From the top of the screen to the top of the object. X and Y are unsigned integers.

The object's spatial resolution can be modified by scale factors in the X and Y directions. The scale X parameter means to repeat each pixel in the object by an amount from 0 to 63. Th scale Y parameter means to repeat each scan line of the object by an amount from 0 to 63. The magnification is achieved by repeating each

Figure 2. Functions of origin, stride, X and Y parameters.



pixel or each line, the number of times given by the scale factor. Here 0 value means no scaling. Implementing this effect in the Y direction with interlaced scan requires thateach line be repeated half as many times in each field. For scale Y = 3 (odd number case), each , line will be displayed 4 times (repeat 3 times plus itself) in a frame. In other words, it will each line twice in each field. For scale Y = 4 (even number case), each line will be displayed 5 times. In other words, each line will be repeated 2 times and 3 times on any one field.

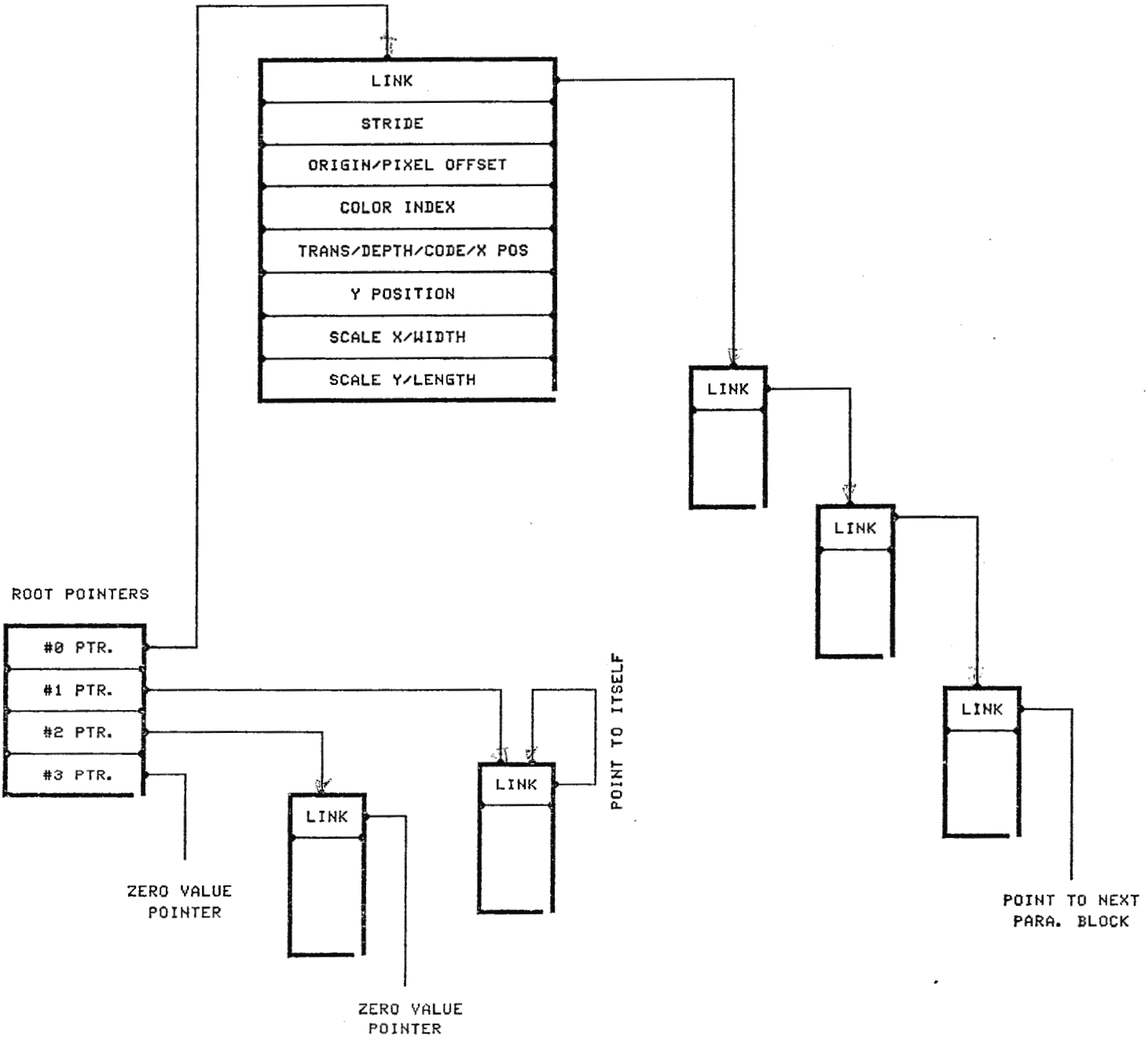
The pixel data, after it is unpacked, does not represent a predefined color, but serve as a displacement to the color index parameter, which address the color map. The color which finally appeared at the corresponding spot on the CRT screen is defined by the color map contents at that address.

The one exception is the treatment of pixel data with value zero. If the Transparency is set and the pixel data is zero, the object processor will not display a color but allow a lower priority object or background to display.

Link is the absolute address of the beginning of the next parameter block to be displayed. Root is the address of the start of the first parameter block and represents the only value written to an object processsor by the CPU. Both Link and Root value share the same register called Link register. If Link or Root of an object processor is zero, no further objects will be interpreted by that processor until a new root is written. The parameters are completely reloaded either when a non-zero Root is written or at end of display of the current object's window.

To CPU the Link parameter is the only addressable register in the Silver chip. CPU writes Root value into the Link register and the Silver chip uses this value to bring the whole Parameter block in including the next link value. Figure 3 will show the functions of the Link register.

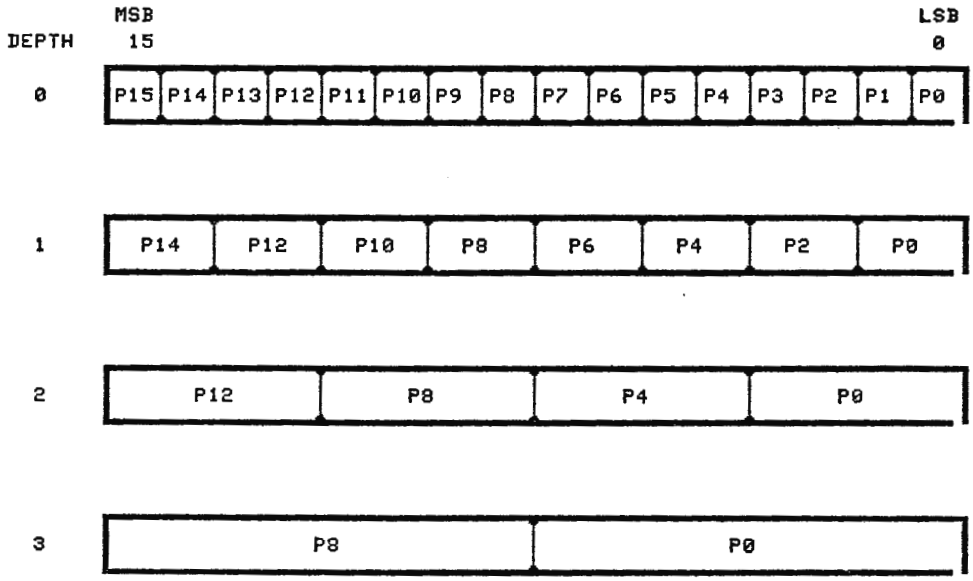
Figure 3. Function of Link register(assume 4 Object Processors in here).



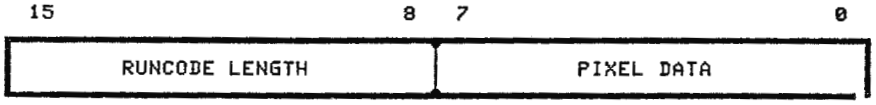
6.2.5 Pixel data format

Pixel data -- the actual contents of pictures -- is stored in 2-dimensional arrays of 16-bit words. Each word of a Picture array has the same format, determined by the coding and Depth parameters as follows:

CODING = 0: BITMAP DATA



CODING = 1: RUNCODED DATA



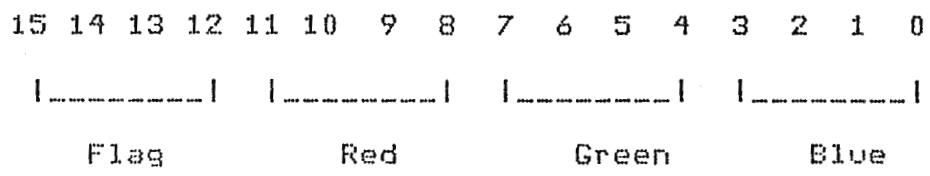
6.2.6 Color map

Colors to be generated by rainbow are stored in a 256-location memory called the "Color Map" which is a separate chip in the Rainbow system. Thus a Rainbow screenful can have up to 256 distinct colors (unless the CPU reload the color map within a field). Each location contains twelve bits of color infromation so that the total range of the accessible colors is 4096. Four bits data in the color map specify the level of each of red, green and blue. Shades of gray will be generated by equal values for each color, giving 16 gray levels (including black and white levels).

The color map holds, besides the color values, 4 bits in each location for flags. One of the flag bits is to be used by external circuitry to be 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 bais. Other flags may be used to enable external texture-generation signals or smoothing filters.

Other encodings of color values are posssible, and Rainbow does not prevent their use. The choice of color and flag encoding will be made by systems designer, will be based on the relative virtues of each encoding (performance, ease of manipulation, and compatibility).

The color map bit allocations are:



6.3 General Block Description

The silver chip contains several functional blocks such as pixel object active, line object active, object active priority, origin update sequencer, pixel address counters, pixel processor, memory sequencer, address generation logic, parameter load logic and input buffer. These blocks perform object active detection, priority solution, direct memory fetching, pixel processing, parameter block loading and provide a pair of even and odd pixels to the Gold chip. Figure 4 shows functioanal blocks of the Silver chip.

The line object active logic is responsible for determining if any of the objects are active anywhere within the scan line on a line by line basis. It use Y, scale Y and Length parameters to find if an object is scale and active in a line. The Y parameter determines when the top most line is active and scale Y parameter determines how many times a line should be repeated and Length parameter determines the number of lines after and including the active top line that the object is active.

The Pixel Object Active is responsible for determining if any of objects are pixel active while the object is Line active in this scan line. It uses X, scale X, Width and Runcode parameters to find if any object is runcode or bitmap, scale and pixel active. The X parameter determines when the right most pixel of an object is active. The scale X parameter determines how many times a pixel needs to be repeated before it is changed to a new pixel. The Width parameter determines the number of pixels after and including the right most active pixel. The Runcode parameter determines whether pixel data are stored in bitmap or runcode format, and how many times that pixel should be repeated. This logic circuit is designed for one per each Object processor.

The Object Active Priority is responsible for determining the priority of an object active if there are more than one object in the same pixel position. Based on the external inputs EVI and ODI, and the internal inputs Pixel Object Active signal, the Object Active Priority logic decides whether to generate an highest priority object active internally or pass the priority to the next cascade Silver chip through the output pins EVO and ODO. There are two circuits; one for even pixel and the other for odd pixel.

The Origin Update Sequencer is responsible for updating the origin which is used as a pointer to a memory word of the left-most pixels of the object. It will be updated by adding one stride or two stride value to origin on a line to line base. This logic circuit is designed for one per each Object processor.

The Pixel Address Counter is responsible for detecting if any any pixel needs to be fetched and determining whether a pair of new pixels or just a new odd pixel has to be fetched. This logic circuit is designed for one per each Object processor.

The Pixel Processor is responsible for processing pixel data, adding the color index, detecting the transparency and strobing the even and odd pixel data to the Gold chip. Based on the value of Depth and pixel offset parameter, each pixel data will be extracted from a word (16 bits) data, then add to the Color Index(8 bits) to creat the real pixel data. The Pixel Processor processes even and odd pixel separately.

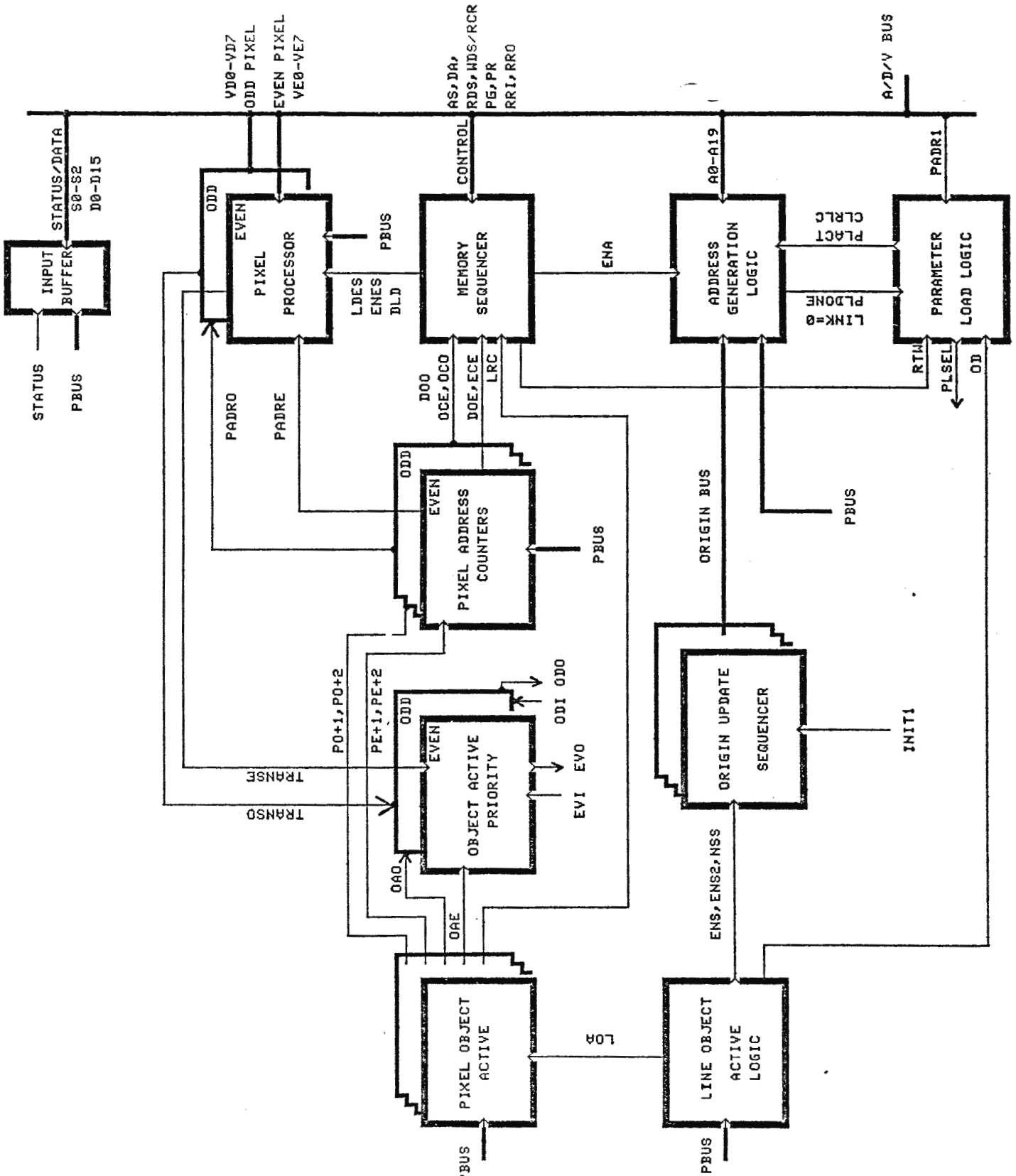
The Memory Sequencer is responsible for detecting the procesor request from CPU and granting bus to CPU whenever the Addresss/ Data/Video bus is not used by the chip. In addtion, the Memory Sequencer takes care of timings of control signals such as read, write and strobe, etc..

The Addresss Generation logic is responsible for generating address for pixel memory fetching and parameter block load. It takes origin data from Origin update sequencer as a base address to fetch pixel data. It also uses the Link parameter as the base address to perform the parameter link load.

The Parameter Load Logic is responsible for controling the sequence of parameter block loading if there any Object Processor needs to load new parameter block. The parameter load logic will determine the priority if more than one Object Processor request to load its parameter block.

The Input Buffer cotains the I/O buffer section and status decoder. Data from CPU will pass here to the internal bus lines. S0 to S2 will be decoded here to generate internal control signal.

Figure 4 Principal functional block of Silver chip.



6.4 Line object active logic

The line object active logic is responsible for determining if any of the objects are active anywhere within the scan line on a line by line basis. It contains Y, Scale Y and Length parameters to find if an object is active within a line and to be scaled in the display. The Y parameter determines when the top most line of an object is active, the length parameter determines the number of lines after and including the active top line that the object is active and the Scale Y parameter determines how many times that the same line has to be repeated in the display. Figure 5 shows the detailed block diagram of the Line Object Active Logic.

Each object is processed sequentially for line object active state (LOA) for the next line to have its pixel data processed. That is, for a line to be displayed, the pixel data must be processed on the line before it is actually displayed and the line and the line object active (LOA) must be determined on the line before the pixel data is processed.

6.4.1 Y parameter logic

The Y parameter is the vertical position of an object representing a distance in lines from the top of the screen to the top of the object. The line count sequence from the top of the screen down in a frame is 0,1,2,....479. The even field contains the even numbered line counts 0,2,4...478 and the odd field contains the odd numbered counts 1,3,5...479. The goal is to find the top most line (topline) of an object in both fields. If Y is even, topline occurs at line Y in even field and line Y+1 in the odd field. Then if Y is odd, topline occurs at line Y in the odd field and line Y+1 in the even field.

This concept also can be expressed in software programming:

```
If Oddy = Oddfield  then topline := Y = linecount<frame>
                    else topline := Y+1 = linecount<frame>
```

To avoid adding 1 to Y in hardware, we substract one from both sides of the quality resulting in

```
If Oddy = Oddfield  then topline := Y = linecount<frame>
                    else topline := Y = linecount<frame>-1
```

Linecount-1 is the same as the linecount for the previous line. Therefore, the state of Y = linecount-1 is the same as the state of Y = linecount for the previous line or last(Y = linecount). This is represented as

```
If Oddy = Oddfield then topline := Y = linecount<frame>
                   else topline := Y = last(Y = linecount<frame>)
```

This works if linecount is lines to a frame; however, the line counter we are using in hardware is for lines to a field. The line counter's count sequence is 0,1,2,...239 for both even and odd fields. A line to a field count can be derived from line counter by setting line counter's output to be the upper 8 bits of the frame count and the oddfield status is derived from the least significant bit.

To develop a algorithm that works with the lines to a field linr counter, we examine each case that Y is even number or odd number case.

For the cases when Y is even the following holds true: (OddY means Yis Odd number).

```
If not OddY then
      If Oddfield then topline := Y+1 = linecount<frame>
                   else topline := Y = linecount<frame>
```

When Y is even, then Y+1 is odd. In other words, the least significant bit (LSB) is changed from a 0 to 1. The LSB of Y or Y+1 would be compared to the LSB of linecount which, in this case, is oddfield. When oddfield, both the LSB of linecount and Y+1 are 1. Also when not oddfield, both the LSB of linecount and Y+1 are 0. Since this is true, the algorithm for even Y case can ignore the LSB and just compare the line counter to the upper 8 bits <1..8>. Therefore,

```
If not oddY then topline := Y<1..8> = linecount<field>
```

Also if Y is odd and an oddfield is being processed then LSB of Y is 1 and equal to the LSB of linecount<frame>. Then the following is true:

```
If OddY and Oddfield
            then topline := Y<1..8> = linecount<field>
```

If Y is odd and an even field is being processed, then topline is Y+1 = linecount. Adding 1 to an odd Y results in more bits modified than just the LSB. Therefore, we use the method described above to avoid an actual hardware addition.

```
If OddY and not Oddfield
        then topline := Last(Y<1..8> = linecount<field>)
```

Combining the three preceding algorithm we have:

```
If not Oddy or Oddfield
            then topline := Y<1..8> = linecount<field>
            else topline := Last(Y<1..8> = linecount<field>
```

The hardware used to find topline is the Y parameter register array, line counter, Y/line equality comparator, Ylag register and a portion of the line object active decoder. The line object active decoder is AND-OR-NOT combinatorial or PLA style logic. The Y parameter register array is a10 bit by 12 object array with data inputs from PBus lines 0 to 9. Data is strobed in by parameter load pulse P1d5. Bit 0 of the output or OddY goes directly into the LOA decoder. Output bits 1-9 are compared by a equality comparator to the output bits 0..8 of the line counter. This performs the operation of Y<1..9> = linecount<field>.

The output (TL) of the equality comparator goes directly into the LOA decoder and into the input of a staye register YLAG. The output (DTL) of the state register, selected by LOSEL, goes to the LOA decoder which performs the function of Last(Y<1..9> = linecount<field>). The YLAG register is loaded immediately after the LOA state has been loaded on an object by object basis.

Part of the LOAdecoder is used to detrmine topline. The decoder will select TL or DTL to be topline based on the status inputs of the OddY or EvenF signals. (EvenF is negated oddfield).

The line counter is an 9 bit counter which is cleared by top of screen (topscreen) and incremented right away after the LOA sequence has completed by signal Newline. Topscreen, EvenF and Oddfield are provided from the status decoder which decodes status S0 to S2 from the Gold chip. All these signals inform the Silver chip as to the current field and that the object processing for the field should begin.

Truth table of Topline is:

EvenF	OddY	TL	DTL	Topline
1	1	X	1	1
1	1	X	0	0
0	X	1	X	1
0	X	0	X	0
X	0	1	X	1
X	0	0	X	0

6.4.2 Length parameter logic

The length parameter represents the number of lines in a frame that an object is active starting at the Y position. The length extends from 0 (no lines displayed) to 479 lines (Object active on all lines). Length, like Y, is in lines to a frame rather than lines to a field. For even numbered lengths, the line to a field count would be the length parameter divided by 2. The same method is true for odd numbered lengths, except 1 must be added to the length for one of the fields. The field for which length is incremented is determined by the state of the LSB of length or Oddfield equal to OddY. This can be defined as:

```
If topline
   then  if Oddlen and (OddY = Oddfield)
            then  length.cnt := length/2 + 1
            else  length.cnt := length/2
   else  length.cnt := length.cnt - 1
```

or

```
If topline
   then
     length.cnt := length/2 + [Oddlen and (OddY = Oddfield)]
   else
     length.cnt := length.cnt - 1;
```

In hardware, the Length/2 is obtained from bits 1 to 9 of the length counter. The addition to length/2 is accomplished by comparing the length counter to one less than what would be normally compared to. This "procrastinates" the addition to the length counter into the length counter's terminating count. The length procrastination bit (LPB) is then the Length/2's increment or [Oddlen AND (OddY = Oddfield)].

The function of the length counter is to indicate how many lines an object is to remain active and when the last line of an object is active so that an automatic parameter reload sequence can occur. A special case arises for length's of 1 and 0. There may be no lines active in the field and for this case, the automatic parameter reload is done on the top line whether the object is active or not. The hardwaresignal that controls the automatic parameter reload is an ouput from the line object active decoder called "object done" (OD). Object Done occurs when length/2 = LPB or when Length/2 = 0 and LPB = 1 and topline is active.

The line object active (LOA) status goes to pixel object active to control whether an object will be displayed on the following line. The LOA status is held for one line time and is maintained as state for the line object active decoder. An object can only be active if either if either the last line was active or it is topline. If the previous line was activeor it is topline then the object is active when Length > 1 or Length/2 = LPB.

6.4.3 Scale Y logic

Scale Y, Origin and Stride parameters control the mechanism for the pixel address to the beginning of the next line of the object.

The Scale Y parameter "magnifies" an object in the Y (vertical) direction. Magnification is achieved by repeating each line scale y times. In other words, each line of source pixels will appear scale y + 1 times in succession. Scale Y is a 6 bit long parameter. The maximum number for scale y is 63.

The implementation of scale Y is somewhat complicated by interlaced scanning. An even scale (scale y +1 = 2, 4, 6...) requires that each line be repeated one-half scale y + 1 times in each field. For odd scales (scale y +1 = 1, 3, 5...), One-half scale y + 1 times must be averaged such that onefield will receive an extra repeat line.

For example, with scale y +1 = 3, lines will be repeated alternatively two times on one field and once on the following field. The result is three repetitions of each line in a complete frame. If Y (vertical position) is 0, then for the even field, origin is repeated by the sequence 2, 1, 2, 1,... and for the odd field the sequence is 1, 2, 1, 2,... Note that for y=1, the sequence is 1, 2, 1, 2,... for even field and 2, 1, 2, 1,... for the odd field.

For Y = 0, stride = 1 and Origin = 100:

line #	Current Origin Even	Odd	Frame
0	100		100
1		100	100
2	100		100
3		101	101
4	101		101
5		101	101
6	102		102
7		102	102
8	102		102

For Y=1, Stride = 1 and Origin = 100:

line #	Current Origin Even	Odd	Frame
0			
1		100	100
2	100		100
3		100	100
4	101		101
5		101	101
6	101		101
7		102	102
8	102		102

Let the sequence 1, 2, 1, 2,... have notation (1,2)* The repeat sequences can be defined in more general terms as:

scale y+1	y	field	sequence
odd	even	even	(scale y/2 +1, scale y/2)*
odd	even	odd	(scale y/2, scale y/2 +1)*
odd	odd	even	(scale y/2, scale y/2 +1)*
odd	odd	odd	(scale y/2 +1, scale y/2)*
even	x	x	((scale y+1)/2)*

x - don't care

If the repeats are controlled by a counter, then the counter woule be loaded with scale y/2, decremented every line and the terminal count would be 1 for scale y/2 and 0 for scale y+1 /2. The repeat sequence can be expressed in terms of the terminal count:

scale y+1	y	field	terminal count	initial term. count
odd	even	even	(0,1)*	0
odd	even	odd	(1,0)*	1
odd	odd	even	(1,0)*	1
odd	odd	odd	(0,1)*	0
even	x	x	0	0

x - don't care codition

The logical equivalent for initial terminal count is:

Oddscale AND (Oddy XOR Oddfield)

Subsequently the repeat sequence (0,1)* or (1,0)* is the state sequence of :

New_terminal_count := Oddcale AND NOT last_terminal_count

When the repeat counter reaches terminal count, the Origin is added to stride, terminal count state is sequenced and the counter is reloaded with scale y/2.

An exception to this sequence occurs with scale y =0 (no repeat). In the below example, the origin sequence for Y = 0 case, the even field is 100, 102, 104... and 101, 103, 105... for the odd field. For Y = 1, the sequence are transposed, with the even field sequence 101, 103, 105,... and the odd field sequence 100, 102, 104,... The difference between the origin within which sequence is two times stride. Adding two times stride to origin causes the Pixel Address Word counter to skip a line for interlaced scanning.

For Y = 0, stride = 1
and Origin = 100:

line #	Current Origin Even	Odd	Frame
0	100		100
1		101	101
2	102		102
3		103	103
4	104		104

For Y=1, Stride = 1
and Origin = 100:

line #	Current Origin Even	Odd	Frame
0			
1		100	100
2	101		101
3		102	102
4	103		103
5		104	104

The pixel address line update hardware is made up of three sections; scale y, origin/stride and part of the line object active logic.

The object_active_on_line and the top_line_of_object are determined by the line objcet active logic and are called LOA and Topline in the logic, respectively. Their mechanism is discussed in the Line Object Active section. The terminal count logic uses the line object active decoder since much of the input stimulus, such as oddscale, oddy and oddfield is common between the line object active and terminal count logic. Terminal count state is held in a 1 bit register. The terminal count output (tc) from the line object active decoder goes to pixel data data pointer update decoder and to the input of the terminal state register. The output of the state register goes back into the line object active decoder as last terminal count (last tc). Extracting the terminal count portion from the pixel address line update algorithm we get:

```
If(top_line_of_object)
  then terminal_count := oddscale AND (oddy XOR oddfield);
  else terminal_count := oddscale AND NOT last_terminal_count;
```

From this algorithm, we can then define the portion pertaining to the terminal count with the following table and notations:

Evenf: evenfield (negated oddfield from Status lines)
tl, dtl: top_line_of_object (topline & delayed topline)
oddy: LSB of Y parameter register
oddscale: LSB of scale Y parameter register
tc: terminal count(alias short,extra) input
last tc: last terminal count, output of tc state_register.

Terminal count table

Input:						Output:
evenf	oddy	tl	dtl	last tc	oddscale	tc
X	X	X	X	X	0-	0
0	0	1	X	X	1	1
0	1	1	X	X	1	0
1	0	1	X	X	1	0
1	1	X	1	X	1	1
0	X	0	X	0	1	1
X	0	0	X	0	1	1
1	1	X	0	0	1	1
0	X	0	X	1	1	0
X	0	0	X	1	1	0
1	1	X	0	1	1	0

The scale y logic consist of a 6 bit scale y parameter register , a 5 bit counter and the pixel data pointer update decoder. The scale y parameter register is loaded by the parameter data load signal Pld7 from data bits 10 through 15. The last significant bit, oddscale, goes to the line object active decoder for terminal count and to the pixel data pointer update decoder for part of scale y =0. The5 most significant bits of the scale y register represent the value (scale y/2), and go to the data inputs of scale y counter. The 5 bits are also compared to zero for rest of the scale y = 0 condition. The scale y counter is a 5 bit binary counter loaded by signal ldsy and decremented by sy-1 signal, both signals from the pixel data pointer update decoder. The scale y counter must compare to terminal count, which will either be a one or a zero. This is done by comparing the upper 4 bits of the counter to zero (sc1=0) and if zero, then the least significant bit (sc=0) will compare to bitwise to terminal count.

```
Scale y counter = 0   :=  sc1=0 AND NOT sc0
Scale y counter = 1   ;=  sc1=0 AND sc0
```

The pixel data pointer update decoder contains the decision making process to update the scale y counter and terminal count and control the origin/stride update circuit. The inputs are: object_active_on_line (LOA), top_line_of_Object (topline) and tc from the line object active logic, scale y = 0 and scale y counter = 0 or 1.

The outputs that control the scale y counter are load counter (ldsy) and decrement counter (sc-1) and next next terminal count state (ntc) which causes terminal count to pass to the next state. The output for the Origin/stride logic are: enable Origin + stride (ENS), and enable Origin + (2 * stride) (EN2S). The algorithm that represents the function of the pixel data pointer decoder can be expressed in the following:

```
If(object_active_on_line)
   If(top_line_of_object)
      then
        next_terminal_count;
        if(scale y = 0)
          then
            if(terminal_count =
               then do nothing
               else enable origin + stride;
          else load_scale_y_counter;

      else
        If(scale y = 0)
          then enable origin + (2 * stride)
          else
          If [(scale_y_counter = 0 and terminal_count = 0)
              OR
              (scale_y_counter = 1 and terminal_count = 1)
          then
            enable origin + stride;
            next_terminal_count;
            load scale_y_counter;
          else decrement scale_y_counter;
```

This then is translated to the following truth table with notations:

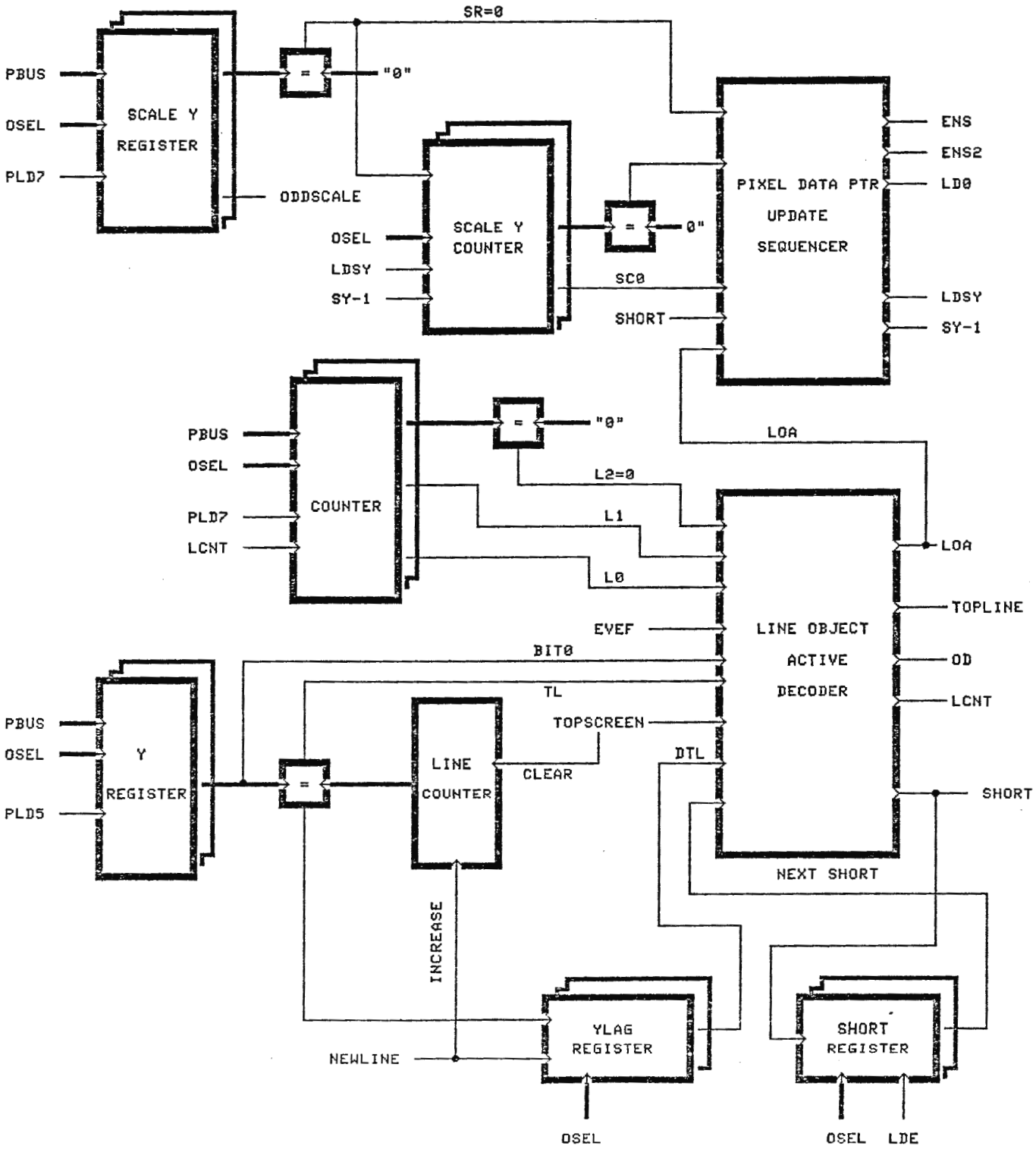
LOA-- object_active_on_line (line object active)
Topline-- top_line_of_object
sr=0-- bits 1 to 5 = 0 of scale y register and not oddscale
tc-- terminal count from line object active
sc1=0-- bits 1 to 4 = 0 of scale y counter
sc0-- bit 0 of scale y counter
ENS-- enable origin + stride
EN2S-- enable origin + (2 * stride)
ntc-- next terminal count
ldsy-- load scale y counter
sc-1-- decrement scale y counter

Pixel data pointer update truth table

Input:

loa	topline	sr=0	tc	sc1=0	sc0	ens	en2s	ntc	ldsy	sc-1
0	X	X	X	X	X	0	0	0	0	0
1	1	1	1	X	X	0	0	1	0	0
1	1	1	0	X	X	1	0	1	0	0
1	1	0	X	X	X	0	0	1	1	0
1	0	1	X	X	X	0	1	0	0	0
1	0	0	0	1	0	1	0	1	1	0
1	0	0	1	1	0	0	0	0	0	1
1	0	0	0	1	1	0	0	0	0	1
1	0	0	1	1	1	1	0	1	1	0
1	0	0	X	0	X	0	0	0	0	1

Figure 5 Principal functional block of Line Object Active logic.



6.5 Pixel object active block

The Pixel object active block is the first stage of the three stage pipeline archicture. This stage is responsible for determining within a video line if and when an object is active, update the source pixel address for bit mapped data, and control the fetching of run length encoded data. There is a pixel object active circuit foreach each Object processor. It uses the X, Width, Scale X and Coding parameters to creat the outputs: Object active to the second stage of pipline Object Active Priority block, Pixel address increment to the Pixel Address counter and run length encoded data fetch signals to the Memory Sequencer. Each Object processor's active state, address and run length encoded fetch information must be determined independently and at pixel colock speeds; therefore, the pixel object active logic must be replicated for each object processor. Basically, this block contains X parameter logic, Width parameter logic, Scale X logic and the Run length logic. Figure 6 shows the principal functional block diagram of the Pixel Object Active logic.

6.5.1 Parameters of Pixel Object active

The X parameter specifices the position of the object on the screen in the horizontal direction. The parameter represents the number of screen pixels from the left of the screen to the left edge of the object. If X is specified greater than the offset the pixel on the right side of the screen, the object will not appear at all. The range of the X parameter is from 0 to 1023.

Width is the horizontal size of an object, measured in scren pixels. If Width is zero, the object will not be displayed. If "Width plus X" imply that part of the object is off the right side of the screen, that part is not processed or displayed. The range of the Width parameter is from 0 to 1023.

The Scale X factor magnifies an object in the horizontal direction. Each source pixel will appear Scale X +1 times in succession. Magnification is achieved by repeating each pixel the number of times given by the Scale X factor. Scale X is ignored for run length encoded data. The range of the Scale X parameter is from 0 to 63.

The coding parameter selects coding format of the source pixel data, cleared for bitmap-atyle data (further described by Depth parameter) or set for run length encoded data (runcoded).

6.5.2 Dipels

Each horizontal line scans at a period of 63.4 usecs. For 640 pixels, we need to average about 99 nsecs per pixel. This includes fetching, processing (bit field extraction and color index add) and transfer to line buffer. It should be painfully clear that there are a number of serious bottle necks that affect the performance of Rainbow (for that matther any other IC's attempting similar performance). These botlenecks are the memory bandwidth (the memory speed times number of parallel bits), process speed of Rainbow and transfer speed of pixel data to the line buffer (in the Gold chip). There two archtectural techniqures that can improve the process and line buffer transfer bottle necks: one is to pipline the process stages of the silver chip and the second is to use multiprocessing. The technique of piplining the process stages will be in detail in another section. Multiprocessing is implemented by processing two pixels simultaneously. The pixel object active logic must be able to deal with pairs of pixels.

A pair of pixels is called a "Dipel". A dipel is made up of an even pixel and an odd pixel. The even pixel is the left most and first displayed followed by the odd pixel on the right. (the first pixel of a line is always the even pixel, the second is the odd pixel, the third one is even again and so on so forth). The dipel position is defined as the positon the current dipel in the horizontal line that Rainbow is processing.

The notation for a dipel is: [*|*]
even|odd
where * represents an active pixel.

6.5.3 Functions of the Pixel object active logic

The POA logic for each object processor must determine, for both the even and odd pixels in a dipel, if an object is active. It produces object active even OAE and object active odd OAD signals which og to their respective Object Active priority block. Each priority circuit determines which Object processor has the highest priority. The highest priority processor then goes about processing and transfering the pixel data to the line buffer.

The source pixel data is pointed to by a pixel bitfield address. In order to progress from one source pixel to the next, we must increment the pixel address. Conversely, to repeat a pixel we leave the pixel address unchanged. The Scale X parameter tells how many times a pixel address to be used before incrementing. If the coding parameter is cleared to bit mapped mode, the the POA logicupdates the bit field addresses of a dipel by providing the pixel address counters with even and odd increment signals PO+1, PO+2, PE+1 and PE+2. The pixel address counters must be held current at all times and and updated whether the object is being dispalyed or not. The pixel address counters are incremented by one or two bit fields or not at all.

If the Coding parameter is set to run length encoded mode, then the POA logicsends the memory sequencer run code fetch control signals. The logic will determine whether run coded data is needed for the even or odd pixel and sends load run code even (IRCE) or load run code odd signals (ICRO) to the sequencer.

In a runcode picture each bit field represents a contiguous horizontal sequence or run of pixels of one color. Each runcodes specifices both a color to use and thelength of the run of pixels of that color. The run coded data is fetched whether the object is displayed or not, in order to keep the run length current.

6.5.4 Object active

An object is defined as being active within the rectangular region bounded by pixel position >=X and pixel position < X+Width in the horizontal direction and by line position >= Y and line position < Y+Length in the vertical direction. Rainbow uses the line object active (LOA) logic to get active on a line state. The LOA logic is described in detail in the last section 6.4. Suffice it to say that the POA logic with an active on a line state. The rectangular region is measured by pixel increments and requires conversion to dipel increeements for Rainbow.

The region bounded in the horizontal direction in dipel increments is dipel position >= floor(X/2) and dipel position < celing((X+Width)/2). The object is active for both even and odd pixels within the region and inactive for the dipel position outside it, except for the following two exceptions. If X is odd and dipel position is equal to floor(X/2), then the object just becomes active for the odd pixel only. If X+Width is odd and the dipel position is equal to Ceiling((X+Width)/2), then the object goes inactive for the odd pixel only. Having only one pixel active in a dipel creates special cases in the start and end of the object's active region for pixel address increment and the runcoded fetch mechanism.

6.5.5 Pixel address increment

If the object is active for both even and odd pixels of the dipel, then the pixel address increment is determined by Scale X being zero, one or greater than one. If Scale X is 0 (no pixels are repeated), then the increment fo rboth even and odd pixel addresses is by two fields. That is, for the even pixel to advance to the next even pixel requires skipping past the odd pixel. If the Scale X is one, where each pixel is repeated once, the increment is by one bitfield. Scale X greater than one requires a counter mechanism which is initially loaded with Scale X and counted down for each pixel. When the counter reaches zero or terminal count, it is reloaded with Scale X. The pixel address is not incremented when terminal count is encountered in a dipel. If terminal count occurs in the odd dipel, then both the even and odd pixel address counters are incremented by one. If terminal count occurs during an even pixel,

then the previous odd pixel address required incrementing by a bitfield and the current even pixel address requires incrementing by a bitfield.

For example, here are three examples for Scale X of 0, 1, and 2 for 5 dipels. Assume that the dipel address begins at 0 and the bitfield width is 1. The address for each pixel is directly under the '*'. Th ebitfield increment for both even and od pixel addresses is between the curren dipel and the next dipel. The scale X count is shown for in the case of Scale X of 2.

```
            [*|*]  [*|*]  [*|*]  [*|*]  [*|*] Dipel
Scale X=0    0 1    2 3    4 5    6 7    8 9  Pixel address
               2 2    2 2    2 2    2 2      even/odd increments

            [*|*]  [*|*]  [*|*]  [*|*]  [*|*] Dipel
Scale X=1    0 0    1 1    2 2    3 3    4 4  Pixel address
               1 1    1 1    1 1    1 1      even/odd increments

            [*|*]  [*|*]  [*|*]  [*|*]  [*|*] Dipel
Scale X=2    0 0    0 1    1 1    2 2    2 3  Pixel address
               0 1    1 0    1 1    0 1      even/odd increments
```

The case when the object just becomes active for only the odd pixel in a dipel requires exceptions to update the pixel address for the starting dipel. For Scale X equals to 0, the even pixel address is incremented by one bitfield and the pixel address is incremented by two bitfields. For Scale X euqals to 1, only the od pixel address is incremented by one bitfield. Neither pixel address is incremented for Scale X greater than 1.

For the start dipel with only the odd pixel active are three examples for Scale X of 0, 1 and 2 for 2 dipels. Again assume that the pixel address begin at 0 and the bitfield width is 1.

```
            [ |*]  [*|*]  Dipel
Scale X=0    0 0    1 2   Pixel Address
               1 2         even/odd increment

            [ |*]  [*|*]  Dipel
Scale X=1    0 0    0 1   Pixel Address
               0 1         even/odd increment
```

```
            [ |*]    [*|*]  Dipel
Scale X=2    0 0      0 0   Pixel Address
             x 2      1 0   Scale X count
                0 0        even/odd increment
```

The Scale X counter must decrement by dipels not pixels. Terminal count then must be interpolated from the value of the Scale X counter to determine where the terminal count occurs on the even or odd pixel. Now if the Scale X counter is zero, the terminal count occurs on the odd pixel. Also if the Scale Counter is a 2, it follows that Scale X counter will be 0 in the even pixel of the next dipel. This is useful for the case of Scale X > 1, in that a Scale X counter of 2 indicates that the odd pixel address should be incremented.

The Scale X counter must be reloaded when terminal count is reached; however now that the counter is decremented by dipels an added complication enters into the picture. If terminal count occurs during the odd pixel, then the counter need only to be reloaded from Scale X to be ready for the next dipel; however, if terminal count occurs during the even pixel, then not only does the counter need reloading but the odd pixel must be accounted for to be ready for next pixel. This is normally done by loading the counter and decrementing it which requires two cycles. Unfortunately, we cannot afford to use two cycles, so instead we procrastinate the decrement of Scale X counter into the terminal count comparison by holding the fact that a decrement is pending in a register. The register is called "Lag". Lag is set whatever terminal count occurs on an even pixel and causes the next terminal count tobe compared to a value one more than it normally would. (i.e. instead of comparing for terminal counts of 0, 1 or 2 it would compare to 1, 2 or 3 respectively).

An example of how Lag and Scale X work is described as follows. Scale X is 2 and the pixel address begins at zero with 1bit per pixel. The logic is initialized by loading the Scale X counter with Scale x (=2) and resetting Lag. A terminal count of 2 causes POA to increment odd pixel address by one. Moving to dipel 2 causes the Scale X counter to be decremented by 2. This results in a terminal count of 0 (occuring on the even pixel) which the even pixel address to be incremented by one. When going to dipel 3, we would have had to load the Scale X counter to 2 and decrement to 1. Instead, we set Lag

and load the Scale X counter with 2. Since Lag is set when going to dipel 4, the terminal count of 2 is interpreted as a terminal count of 1 or ending on an odd pixel. Both even and odd pixel addresses are incremented, Lag is reset and the Scale X counter is reloaded.

	1	2	3	4	5	
	[*|*]	[*|*]	[*|*]	[*|*]	[*|*]	Dipel
Scale X=2	0 0	0 1	1 1	2 2	2 3	Pixel address
	2	0	2	2	0	Scale X counter
	0	0	1	0	0	Lag
		0 1	1 0	1 1	0 1	even/odd increments

6.5.6 Run length encoded data fetch mechanism

The fetch mechanism for run length encoded data is very silmiliar to that of the pixel address increment for bit mapped data. Both can require "runs" of repeated source pixels. The difference is that instead of incrementing this pixel address, we fetch next runcode. Since the similarity is strong, we can use the Scale X counter to double as the run length counter and still use terminal count and Lag logic with the same method as before. Now, instead of loading the Scale X counter from the Scale X parameter when a terminal count occurs, we tell the memory sequencer to load the Scale X/run length counter from memory. The nature of terminal count is somewhat different for run length encode data than bit mapped data since a terminal count for run length encode data requires fetching for the current dipel rather than incrementing a pixel address for the next dipel.

If the current length counter is 0, then the run ended on the even pixel and must fetch for the odd pixel. If the runlength counter is -1, the run ended in the last dipel's odd pixel and requires a fetch for the even pixel and a check for the possibility of an odd pixel fetch (if run length for even pixel was 0). For a run length of -1, the logic must decrement the run length counter and test for -1 requiring two cycles. Again we cannot afford the two cycles and use a similar technique to the Lag concept to predict when an even fetch will occur. That is, when the run length counter is 1, then it is predicted that the next dipel run length counter wil be -1 and will require a even pixel fetch.

We then use a register called FE (fetch even) to hold the state of predicted even pixel fetch. FE is initialized set so that the first dipel will fetch the first runcode fo rthe even pixel when the object become active. An exception to this is if the first dipel only has odd pixel active, then only the odd pixel is fetched for.

The following table illustrates thedecision making process for runcode fetching.

Dipel Run Code state Transition Table

Current State	Next state
FE	Do even fetch, Do even pixel run code state
ScaleX.cnt = Lag	Do odd fetch, Do odd pixel run code state
ScaleX.cnt = 1+Lag	Set FE, Reset Lag
ScaleX.cnt = 1+Lag	Decrement ScaleX.cnt

Odd Pixel Run Code State Transition Table

Current State	Next state
ScaleX.cnt = 0	Set FE, Reset Lag
ScaleX.cnt > 0	Set Lag

Even Pixel Run Code State Transition Table

Current State	Next state
ScaleX.cnt = 0	Do odd fetch, Do odd pixel run code state
ScaleX.cnt = 1	Set FE, Reset Lag
ScaleX.cnt > 1	Decrement ScaleX.cnt

An example of the run length fecth mechanism is described as follows. FE is initialized set, which cause a fecth for the first active dipel. The fetch results in a run length of 1 satisfying the pixel data requirements for the dipel, but setting FE because the next dipel's terminal count would have been -1. When dipel two is processed, because FE is set, LRCE is set causing a fetch fpr the even pixel.

Since the run length is grater than 1, thelogic progesses to the next dipel. In dipel 3, terminal count occurs on the even pixel, indicating a fetch is required for the odd pixel. LRCO is set and the fetch occurs giving a run length of 1. Lag is set to compensate for thefetch occuring on the odd pixel just as it would for thebitmapped pixel address increment. At dipel 4, because lag is set, the terminal count of 1 is interpreted as a terminal count of 0 resulting in a fetch for the odd pixel. Since the run length of the odd pixel is 0, FE is set again.

The runlengths used forthis example are (1,2,1,0)

1	2	3	4	
[*|*]	[*|*]	[*|*]	[*|*]	Dipel
0 0	1 1	2 2	2 3	Runcode address
1	2	0 1	0 0	Scale X counter
0	0	0 1	0 0	Lag
1	0	0 1	0 1	FE

6.5.7 Pixel object active decision tree

This part is basically a decoder which takes inputs from the X register logic, Scale X logic and width logic to generate the object active signals, set or reset Lag signals and pixel address increment signals. The whole decision tree can be implemented by the PLA circuitry. Figure 7 shows the diagram of the pixel object active decision tree.

Figure 6 Principal functional block diagram of Pixel Object Active Logic.

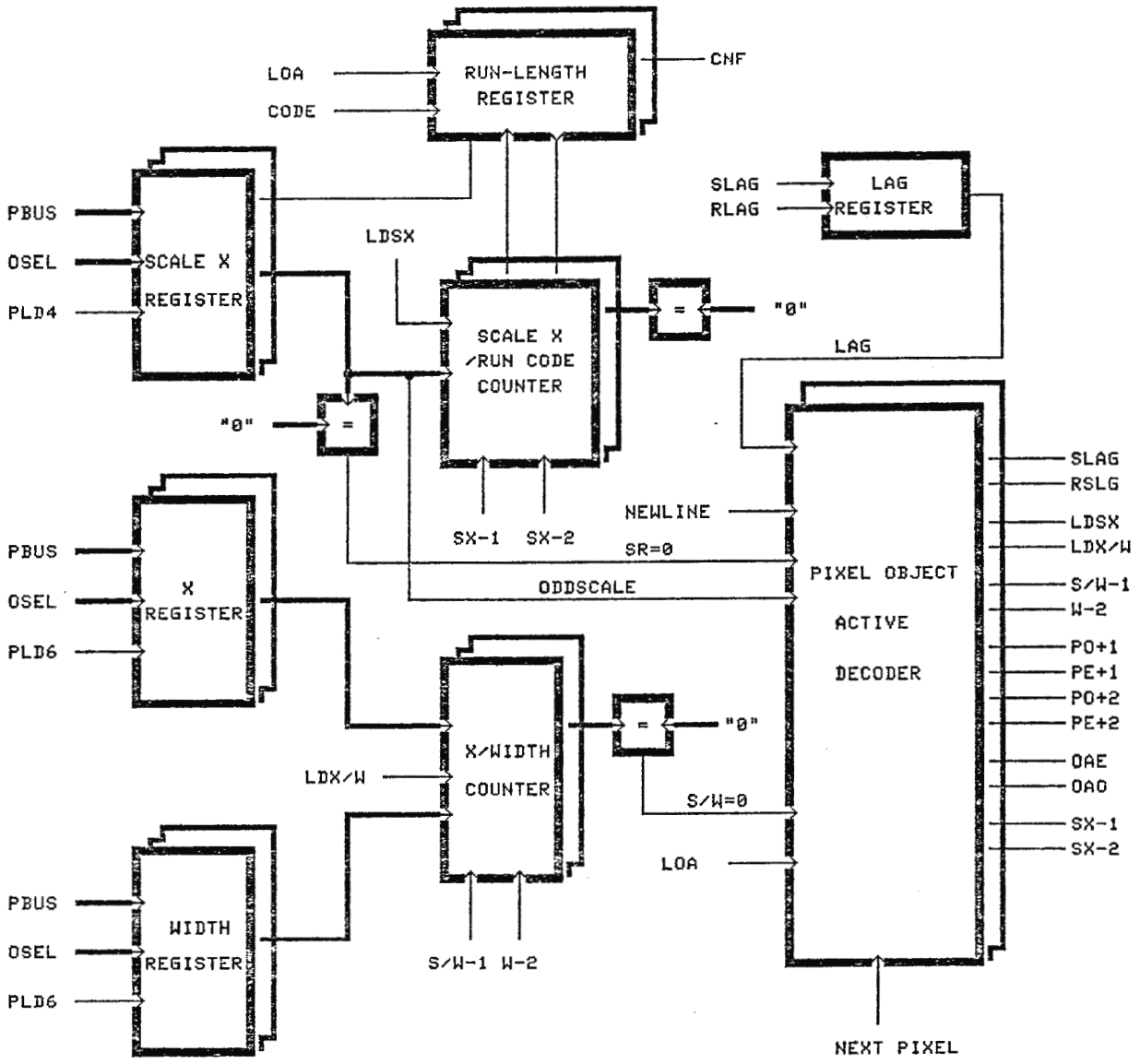
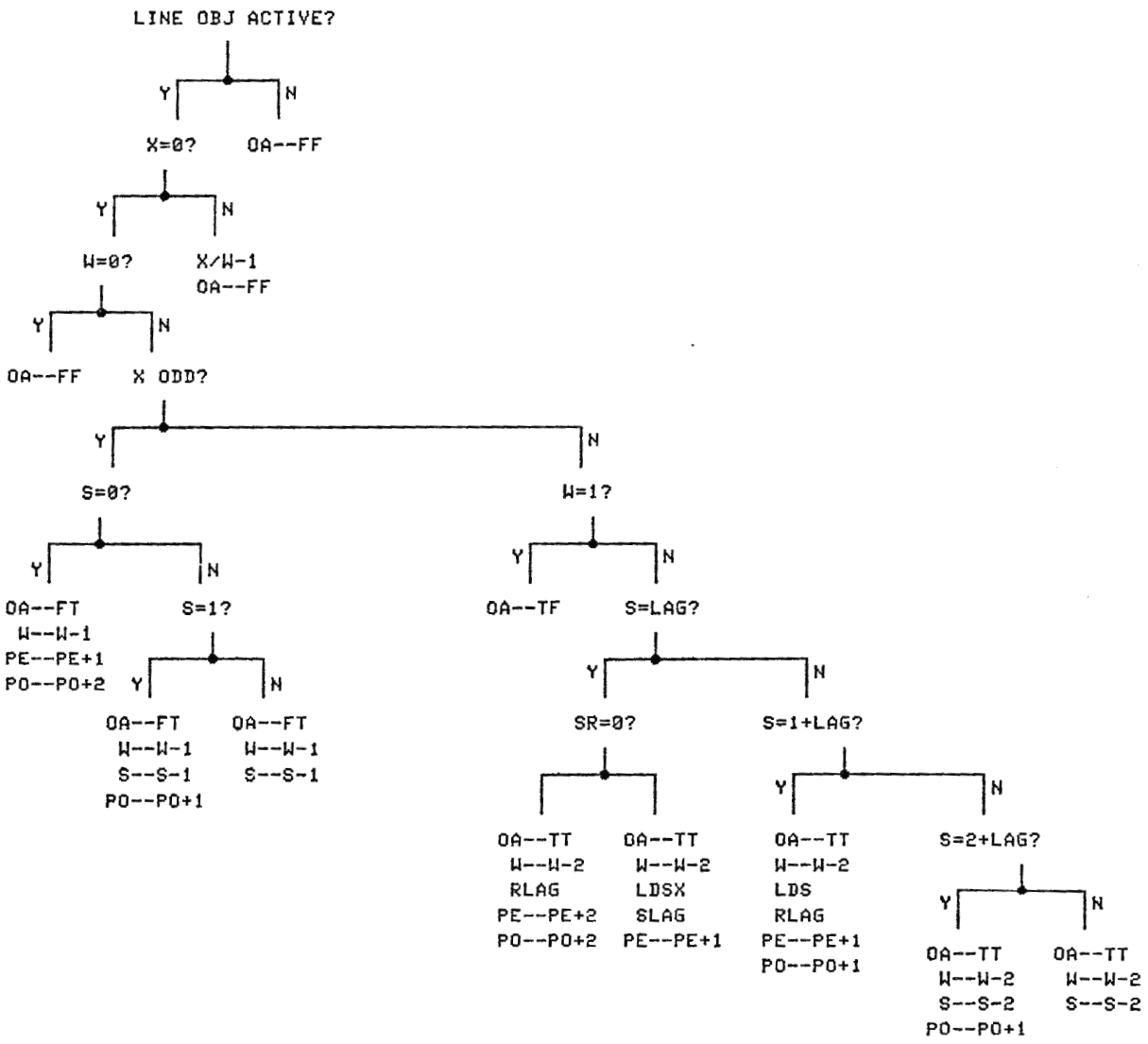


Figure 7. The diagram of pixel object active decision tree.



6.6 Object active priority block

The Object active priority block is the second stage of the three stage pipeline architecture. This stage is responsible for determining the highest priority Object among active Objects which come from the result of the first stage. If pixel data of the highest priority object is transparent, then the next highest priority active object will be determined within that cycle. Because the pipeline structure, this stage will be frozen when the third stage is performing a memory fetch. In other words, the object active priority logic will not proceed to process next pair of pixels when the present two pixels in the Pixel Processor Logic are obsolete and the Silver chip need to fetch pixel data to update its pixel data register. Figure 8 provides the principal functional block diagram of the Object Active Priority logic

6.6.1 Input port

The input to this block comes from the first stage of pipeline--the Pixel Object active which determines Object Active among Object Processors. These object active information will be loaded into this Object Active Priority block during phase-one clock. The loading will be completed before the third stage (pixel processor) detects the transparency of previous pair pixels. Once The pipeline is frozen, the Object Active information will not be loaded in.

6.6.2 Output port

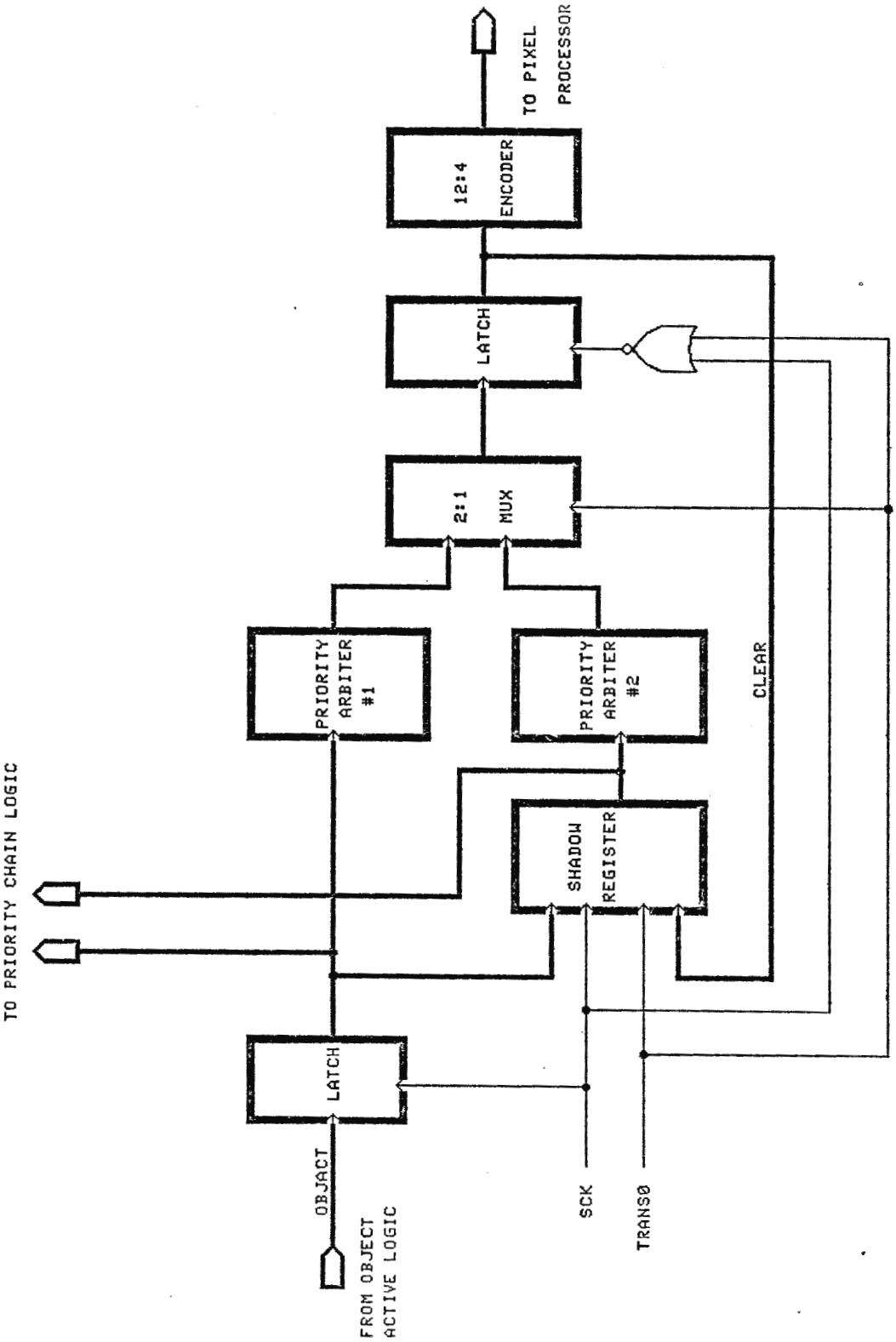
The output from this block provides Object Select signals indicated the highest priority object. Among these signals only one is active at one time which will be used to select the correct pixel data register in un the Pixel Processor block. The Object Select information is provided during the phase-two clock of this pipeline stage.

6.6.3 Priority determination

The highest priority object is determined through the Priority Arbiter 1 in the Non-transparent case and through the Shadow register and Priority Arbiter 2 in pixel transparent case. The input object active will be arbitrated during the Phase-one clock and sent out during the phase-two clock. Meanwhile, the input will be latched into the Shadow register during the same phase-two clock. Due to the pipeline archture, however, the highest priority object active stored in the Shadow register will be cleared in the phase-one clock and the Pixel Processor will feedback the pixel transparency condition in the phase-two clock. If pixels in the present highest priority object is transparent, the next highest priority object will be determined by data in in the Shadow register and Priority Arbiter 2.

The Object Active Priority block is designed for both even and odd pixel object. The present Silver chip will pass its pixel priority to the next Silver chip if there no object active in the input port latch or all active object has transparent pixels in the chip. The silver chip uses Even Priority Out EVO and Odd Priority Out ODO pins to pass the Priority.

Figure 8 Principal functional block diagram of Object Active Priority.



6.7 Pixel address counter block

The silver chip fetches 16 bit words from memory to the pixel processor. The data is temporarily stored in the 16 bit data register for each object. The register holds 2 to 16 bitfields (2 bitfields means eight bits for each bitfield and 16 bitfields means 1 bit for each bitfield) while the bit splitter selects the pixel data bit field for use. The data registers act as a bit field "data cache" holding groups of bit fields. The Pixel Address Counter block is responsible for maintaining the necessary pixel data in the data registers. When data is requires, the pixel Address counter will signal the Memory Sequence to read the next pixel data word from memory. Figure 9 shows the principal functional block diagram of the Pixel Address Counter.

6.7.1 pixel address counter logic

The Silver chip always maintains 23 bit address information of each object's pixel value. This address information contains word address portion and pixel offset address portion. The word address portion is a 19 bit value which initially assumes the values of the Origin parameter. This value is provided by the Origin Update Sequencer and updated by the Addresss Generation Logic Block. The Pixel Offset portion which is stored in the Pixel Address counter is a 4 bit address, that is, it point to the bit position in a 16 bit word position where the least significant bit is bit 0 and the most significant bit is bit 15. The Pixel Address counters are loaded with the contents of the Pixel Offset parameters at the beginning of every active line. There are two pixel pixel address counter for each object, for the even and odd pixels in a dipel (a pair of pixels).

The current pixel address are updated by adding either the designed 'bit field' or the combined 'bit field' width of two pixels to the pixel address counter and letting the odd pixel address counter increment the pixel word address counter (in the Addresss Generation Logic block). The pixel object active logic determines whether the pixel addresss counters are incremected by one 'bit field' width or two. Since pixels are processed pairs at a time as 'dipels', the pixel address counter must keep current 'bit field' addresses of both the even and odd pixels.

For example, the distance between even pixel of the last dipel and the even pixel of the current dipel is two pixels if the Scale X parameter is zero. If scale X is grater than zero, then the pixel address counter need not be updated until the pixel is repeated scale x times and then is incremented by one 'bit field'. The depth parameter determines what the 'bit field' width is and it is depth and increment signals Pe+1, Po+1, Pe+2 or Po+2 from the Pixel object active logic that ultimately determine the actual increment of the pixel address counters. The Pixel Address counters can beincremented by 1, 2, 4, 8, or 16. When the coding parameter is set for runlength encoded pixel data, the Pixel Address counter are bypassed and the Pixel word address counters are incremented by one for each fetch. This is the same as incrementing the pixel addresss counter by 16. Ths following table illustrates the address counter increments:

Depth	Increment Signal	Pixel address
	counter increment	
0	Pe/o + 1	1
0	Pe/o + 2	2
1	Pe/o + 1	2
1	Pe/o + 2	4
2	Pe/0 + 1	4
2	Pe/o + 2	8
3	Pe/o + 1	8
3	Pe/o + 2	16
x	runlength encoded	16

The 4 bit output of the pixel address counter associated with the object to be displayed is passed to the bit splitter uses these output Padr 0 to Padr 3 to select which bit field to use.

Each of the Pixel address counter produce a carry out of the 4th counter stage. The "carry" is active for only one state. For the even pixel address counter, the carry (EC) is selected by the even pixel object select (OSELe) to be passed to the even pixel fetch decoder (the carry signal is ECe). For the odd pixel address counter, the carry (OC) is selected by the even and the odd pixel object active select (OSELe & OSELo) to be passed to the even pixel fetch decoder (The carry is OCe selected by OSELe) and to the odd pixel fetch decoder (The carry is OCo selected by the OSELo).

The Odd carry (OC) for each object also sets the data obsolete register associated with all except the select object. The odd carry also serves as a increment for the Pixel Word Addres counter for each object.

6.7.2 Data obsolete register

The data obsolete register is used by the pixel fetch decoder to determine if the data in the pixel data register is current. It is a single bit register, one for each object, which is set by a odd carry for all object not selected and reset by a pixel data load for a selected object. Data in the Pixel Data register becomes obsolete when the pixel address overflows from the bit to the word boundary. It becomes current when the word is fetch from memory. The output of the data obsolete bits are selected to pass to the even pixel fetch decoder (as DOe) by the even pixel object active select (OSELe) and to the odd pixel data fetch decoder (as DOo) by the odd pixel object active select OSELo).

6.7.3 Pixel fetch decoders

The even and odd pixel fetch decoders translate the carry, data obsolete and selects signals into meaningful state stimulus for the memory cycle sequencer. There are 3 basic types of bit mapped pixel data fetching; these fetch for even pixel data (De), fetch for odd pixel data (Do) and shadow even pixel data and fetch for odd pixel data (S). Besides the basic types of fetches there are combinations of the basic types; these are fetch both even and odd pixel data (for different type objectin a dipel) and fetch even pixel data, then shadow even pixel data while fetching odd pixel data. The action of shadow even pixel data occurs when the object is active for both pixels in the dipel and bit fields for the pixels are divided by word boundaries. In a shadow fetch, the bit field for the even pixel is stored into a temporary buffer (called Even shadow pixel register in the Pixel processor logic) while the odd pixel is fetched.

The Even pixel fetch decoder translates the carries of both even and odd pixel address counters and data obsolete (selected by OSELe) into the shadow (S) and even pixel fetch (De) states. The odd pixel fetch decoder translates the odd pixel adress counter carry and data obsolete (selected by OSELo) into the Odd pixel fetch (DO) state. Also, a common input to both the even and odd pixel fetch decoders is the Data Same OBject Signal (DSO). The Data Same Object signal is the result comparing for equality the equality the object selects for even and odd pixels. The function of the data same object signal is to determine whether the fetch is a shadow fetch or a even pixel data/odd pixel data fetch (DeDo). The following truth table illustrates the relationship between the input signals and output states of the even and odd pixel fetch decoders:

Even pixel fetch decision truth table

Input:				Output:		
DSO	ECe	OCe	DOe	De	S	Condition
1	X	0	0	0	0	A
1	X	0	1	1	0	A
1	0	1	0	0	1	D
1	0	1	1	1	1	D
1	1	1	X	1	0	B
0	X	X	1	1	0	Data obsolete
0	1	1	0	1	0	B
0	0	1	0	0	0	D
0	X	0	0	0	0	A(D clean up)

Odd pixel fetch decision truth table

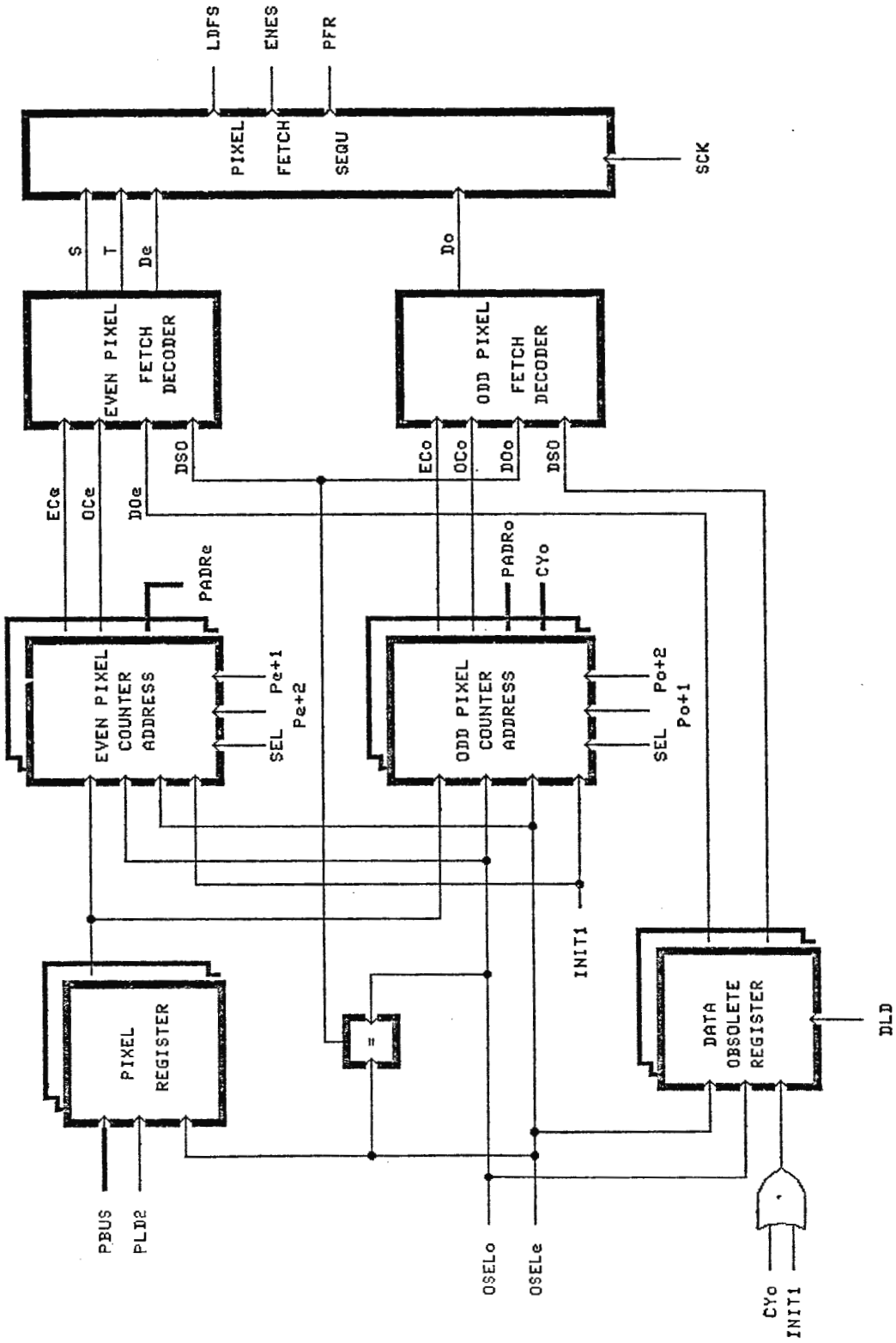
DSO	OCo	DOo	Do	Condition
1	X	X	0	N/A
0	X	1	1	Data obsolete
0	1	0	1	B or D
0	0	0	0	A

The information from carries used by the pixel fetch translation table indicates when word boundaries cross the current dipel. If word boundary cross before (B condition) the current dipel, both the even and odd carries will be set. If the word boundary occurs during (D condition) the current dipel then only the odd carry will be set. If the word boundary occurs after (A condition) the current dipel then no carry will be set.

The following is true for cases wher the pixels in the dipel belongs to the same object. If the word boundary occurs before the current dipel, a fetch for even pixel will satisfy fetch requirements for both pixels. If the boundary occurs during the current pixel, then the even pixel data must be stored in the temporary buffer (shadowed) and pixel data register is used for odd pixel fetched. If the data in the pixel data buffer is obsolete for the current dipel, then the boundary occurs after the current dipel, the data need not be fetched unless the data obsolete bit is set.

If the dipel contains pixels from different objects then the following rules hold. If data obsolete is true for either even or odd, then the respective pixel data must be fetched. Since the objects are different and pixel data is stored into different registers, then there is no need to shadow the data. If the data obsolete is not true, then for the even pixel, a fetch is required for only a boundary condition before thhe current dipel (since data is valid for the even pixel for boundaries during and after the current dipel) and for an odd pixel, a fetch is required for boundary conditions before and during the current dipel.

Figure 7 Principal functional block diagram of the Pixel address counter.



6.8 Pixel Processor block

The Pixel Processor block is the third stage of the three stage pipeline architecture. This stage is responsible for processing the pixel data from memory to the Gold chips line buffer. The packed pixel data is loaded into the pixel data register during a memory fetch and the data is then unpacked by the bit splitter intobit fields defined by the Depth parameter. The unpacked pixel data is added to the contents of the color index register and passed to the A/D/V (address/data/video) bus. At the same time that the pixel data is being summed with color index register, the data is also tested for zero value. If data value is zero and transparency bit is set, then the pixel processor will provide data transparent infomation to wherever is needed.

There are two sets of bits splitter, color index adder and transparency logic to process two pixels (Even and Odd) simultaneously. The pixel data register, Depth register, Color Index register and Transparency register contains two sets of output buffers. The output buffer are selected by two separate object select address (four bits address). These lines are OSELe (Even pixel select) and OSELo (odd pixel select). Figure 11 shows the principal functional block of the Pixel Processor LOgic.

6.8.1 Pixel data & Even shadow registers

The Pixel Data Register is storaged for packed pixel data. It is organized as 16 bits ina register and one register for each Object. The 16 bits data lines connect to the internal Parameter data bus (PBus). Data is loaded into Register when OSELe address lines select the register to be written into, data is valid on the data input lines and Data Load signal (Dld) is asserted. There are two sets output buffers; one set, selected by OSELe, is the even pixel data output and the other set, selected by OSELo, is odd pixel data output. The pixel data outputsbits 0-7 are multiplexed with outputs bits 8-15 in the output stages of the Pixel Data Register. This provides first step of "bit split" from 16 bit word to 8 bit byte. Pixel address bit 3 (PADRx 3) signal select upper byte (bits 8-15) when HIGH and the lower byte (bits 0-7) when LOW.

The purpose of the Even Shadow Register is used to temporarily save pixel data which is already in the Pixel Data Register but will be overwritten soon by a new Odd pixel Data whenever a memory fetch is necesaary.

The Even shadow Register output bits 0-7 are additionally multiplexed with even pixel data output bits 0-7 and 8-15 and this goes to the input f even pixel bit splitter. The Enable Even Shadow (ENES) signal selects the output of the Even Shadow Register when asserted by the Memory Sequencer during T and S mode pixel data fetches. If the ENES is not asserted, then either the upper or lower byte byte of the even pixel data Register outputs pass to the Even bit splitter. The Odd data output byte goes to the input of Odd bit splitter.

The Even Shadow Register is loaded from the Even pixel data register output by the Even Shadow strobe signal (LDES). LEDS is asserted by the memory Sequencer during a T or S mode pixel data fetches.

6.8.2 Bit splitter

There are two bit splitter which are usee for extracting bit fields from the 16 bit word of the Pixel Data register. The number of bits in the extracted bit field is controlled by the Depth parameter. The lower order 4 bits of the pixel counter (PADRo3-PADRo0 and PADRe3-PADRe0) From the Pixel Address Counter block controls the position of bit field within the packed pixel data word. The extracted bit field is justified to the least significant bits and balance of bits which are make a full byte out of the bit field are set to zero. The resulting 8 bit field output from the bit splitter goes to the input of the Color Index summer.

The first stage of the bit splitter is an octal 3 to 1 MUX (Multiplexer) for the even pixel and an octal 2 to 1 MUX on the odd side. The function of the first stage , as described in the above section), is to reduce the 16 bit packed word into an 8 bit value selected by PADRx 3 (PADRx stands for PADRo and PADRe). The subsequent 8 bit value goes to the second stage which will break the 8 bit value into the appropriate bit fields. An 8 to 1 MUX selected by PADRx 2 to PADRx 0

is the unpacked bit 0. A 4 to 1 MUX selected by PADRx 1 and PADRx 2 is the unpacked bit 1. Two 2 to 1 MUXes selected by PADRx 2 is the Unpacked bit 2 and 3. Unpacked bits 4-7 are from the first stage bits 4-7 If depth is 0, 1 or 2, the unpacked bit 4-7 are gated to zero. If depth is 0 or 1, then unpacked bits 2 and 3 are zeroed. For depth 0 the unpacked bit 1 is zeroed.

Figure 11 shows the design structure of the Bit splitter.

6.8.3 Depth register

The Depth Register is a 2 bits wide register and one register for each Object. Its data input are from PBus bits 14-15. The registers are loaded during a parameter sequence when data and OSELe are stable and parameter load 4 (Pld4) signal is asserted. The Depth Register has two separate output buffer which are selected by OSELe and OSELo. The output selected by OSELe goes to the even bit splitter and even pixel address counter, and the output selected by OSELo goes to the odd bit splitter and odd pixel address counter.

6.8.4 Color index logic

The unpacked pixel data is added to an eight bit constant called color index. The resultant sum is passed to the A/D/V (Addresss/Data/Video) bus by way of Video Data Storbe signals. The video data is then loaded into the the Gold chip's line buffer whose output is used to address a 256 entry Color Map. The color index parameter can be thought as a base pointer to the Color Map chip.

The Color Index Register is organized as an 8 bit wide register and one register is for each Object. Its data input are from PBus bits 8-15. The register are loaded during a parameter load sequence when data and OSELe are stable and the parameter load 3 (Pld3) signal is asserted. The color index Register has two separate output buffer which are selected by OSELo and OSELe. The output selected by OSELe goes to the even pixel color index summer and the output selected by OSELo goes to the odd pixel color index summer.

6.8.5 Transparency logic

Transparency selects whether a pixel data with a value of zero is interpreted as relinquishing display priority to a low priority object or as a zero offset to the color index into the color map. The transparency parameter serves as a mask to the test for zero of bit splitters' output. If a transparency is encountered and the transparent parameter is set, then the Object Active Priority block will delete the current object from being active and rearbitrate for next highest priority object.

The Transparency Register is a single bit register and one fro each register. Its data input is from PBus bit 15. The registers are loaded during a parameter load sequence when data and OSELe are stable and the parameter load 3 (Pld3) signal is asserted. The Transparency Register has two separate output buffers which will be selected by OSELo and OSELe. The output selected by OSELe and OSELo are AND gated with the corresponding pixel data equal to zero comparator and the resultant flags the even and/or odd Object active Priority block that a transparent pixel(s) occured.

Figure 10 Principal functional block of the Pixel Processor logic.

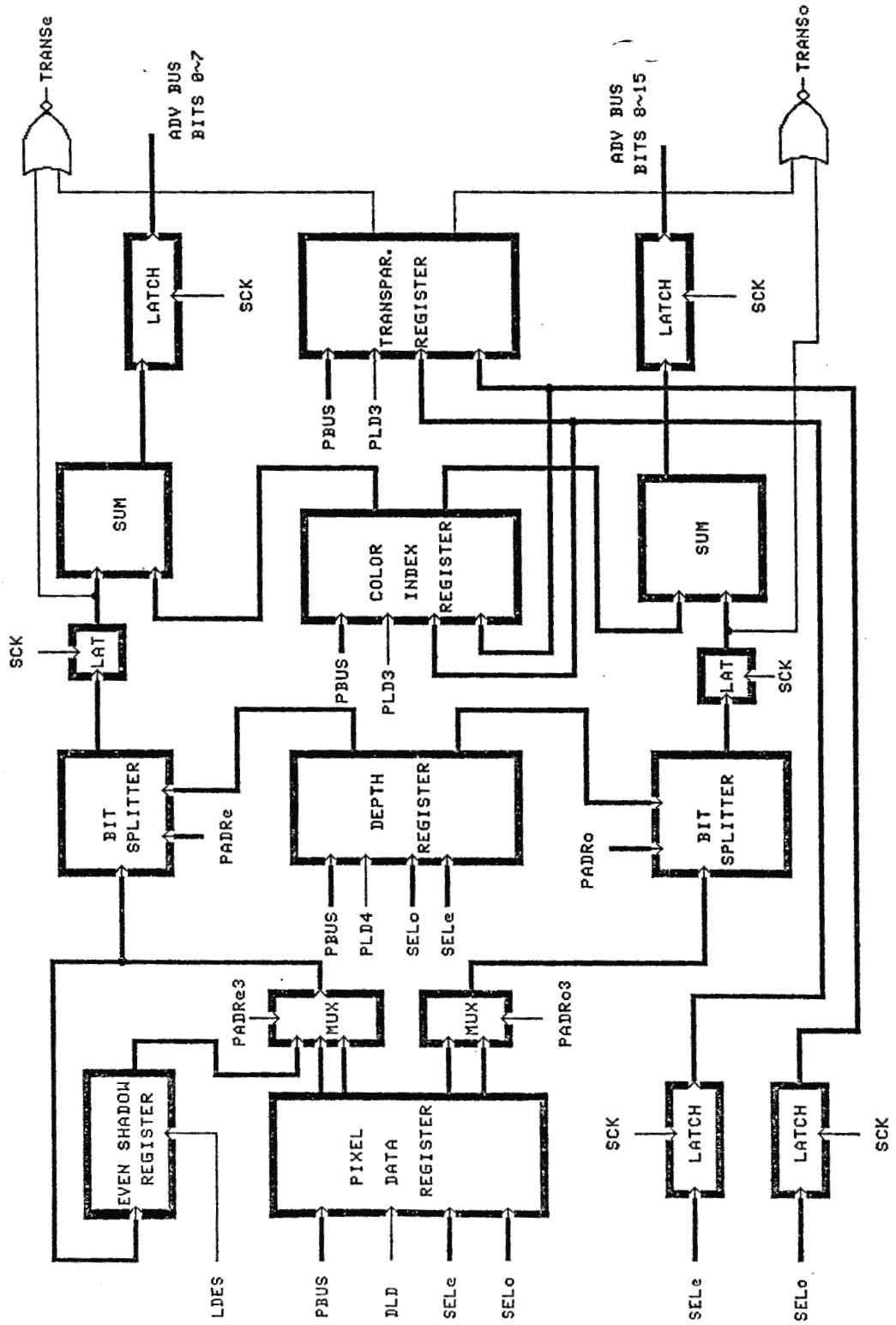
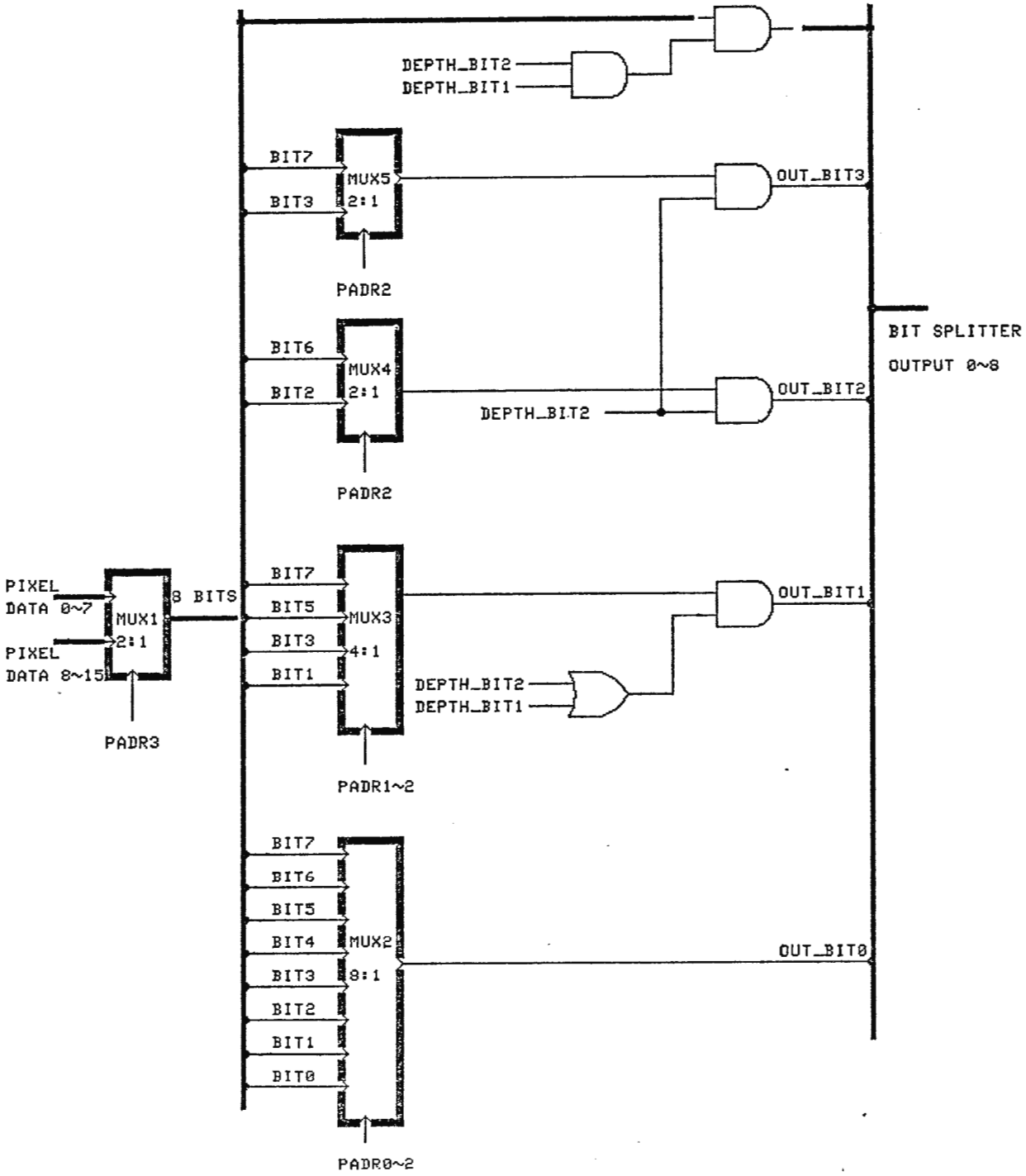


Figure 11. The architecture of Bit Splitter.



6.9 memory sequencer block

The communication between the CPU and the Silver chip includes the "Root-write" and the "Root-read" initiated by CPU to write or read Root in the Silver chip, the "Parameter load" and "Pixel data load" initiated by the Silver chip to read the Parameters block or pixel data from the system memory. The function of the memory sequencer is to generate appropriate sequence of control signals to guide the data transfer between the CPU and the Silver chip to avoid any contention. Also the memory sequencer is responsible for arbitrating the three bus chaining. Figure 12 shows the principal functional block diagram of this memory sequencer logic.

6.9.1 Interfacing control sequence

When CPU wants to perform a "Root-read" or "Root-write" to the Silver chip, CPU will present the appropriate address to the external "Interface_Logic" circuit which will generate the Processor Request signal to all Silver and Gold chips. Every Silver and Gold chip will assert Processor Grant to High to indicate it is not using the the Address/Data/Video bus. After received the PG High, the Interface_Logic will release the Address Strobe AS and Write (or Read) Data Strobe signals to the Silver or Gold chip (only one) which CPU wants to address. Then the addresed chip will assert the Data Acknowledge DA signal to the CPU. Figure 13 provides the timing diagram of the "Root-write"/"Root-read" sequence.

The data transfer between the system memory and the Silver chip consist of Parameters block and pixel data. The parameter load memory cycle is initiated by the parameter load request state machine. Eight parameterwords of a particular object will be loaded from the system memory into the Silver chip in one or several horizontal scanning line time. When the Status indicates Link Load Active, A/D/V bus is available and there is a need to fetch new parameters block, the Silver chip will assert the Address Stobe AS and Read Data Strobe RDS to LOW in order. Once Data Acknowledge DA is High from the Memory system Silver chip will latch data at the rising edge od the Read Data Strobe. Figure 14 provides the timing diagram of the Parameter load sequence.

The Pixel data fetch can be classified into four cases, namely, the DE, the DO, the S and the Runcode case. As they were described in the Pixel Address Counter section, the DE case means that the pixel data to be fetched is the for the even pixel of the current dipel. Similarly, the DO case is for fetching the odd pixel data. When both pixels in the current dipel are from the same object and the odd pixel to be fetched, thsi is the S case. In the S case, the valid data for the even pixel would be saved in the Shadow register less it should be destroyed by the new data. For Runcode case, the length of code runs down to zero for the present pixel and Silver chip has to fetch a new byte of Runcode for the next pixel. When status indicates Pixel Active, the A/D/V bus is available and silver needs to fetch new dipel, the Address Strobe AS and Read Data Strobe RDS signals will be asserted in order. The appropriate addresss will be provided by the Address Generation LOgic block. Then the memory system will provide Data Acknowledge DA signal to infor the Silver chip that data on bus line is Valid. Figure 15 provide the timing diagram of pixel fetch sequence.

To avoid the contention of the mentioned memory cycle, priorites are assigned to these memory cycles. The highest priority is assigned to the CPU Root-write/ Root-read operations. The Root-write or Root-read can happen during the pixel active period or the parameter load period. However, the Root-write or Root-read cycle will not start until the current pixel data fetch or parameter block fecth is completed. There is no contention problem between the pixel data fetch and parameter load memory cycles. Since both memory cycles only occur when the Status line indicates the appropriate status.

6.9.2 Video bus chain

The memory sequencer includes three chaining control signals to resolve the arbitration problem, namely, the RRI/RRO chain, EVI/EVO chain and ODI/ODO chain. Furthermore, two broadcasting type networks, the Even video data strobe EVDS and Odd Video Data Strobe networks are included in the Memory Sequencer block.

Two priority chain EVI/EVO and ODI/ODO have the same kind of functions. If EVI input is High, the chip gets priority for Even pixel. If EVI input is LOw, no matter which Object Processor is active in the chip for even pixel the Pixel Processor will not process the Even Pixel. For the EVI is High case, if none of Object processor is active in the chip or Even pixel data is transparent for all active Object, then rhe Silver chip will pass priority to the next chip through the EVO pin.

The RRI/RRO chain resolves the bus contention of accessing the Address/data/video bus. If the chip get RRI input High, it does mean that the chip obtain the right to use the A/D/V bus to access memory system. Otherwise, the chip can not access the memory. Several conditions to release the bus mastership by asserting High in the RRO pin includes:

1) Runcode request from the next priority chip, but present chip does not need to fetch data.
   (RRI is High; EVI, ODI are High; EVO, ODO are Low; RCR is Low)

2) Bus is available and both Priority are released to to the next chip.
   (RRI is High, EVO and ODO are High)

3) Bus is available; either priority chain is released to the next chip and no need to fetch the pixel data.
   (RRI is High; EVI (ODI) is High, EVO (ODO) is High)

4) Bus is available; either priority chain is released to the next chip and present pixel data fetch is completed.
   (RRI is High; EVI (ODI) is High, EVO (ODO) is High; but wait for present memory fetch is finished)

5) Bus is available; when link load active status is is off and refresh active is on.

From the above conditions, we can conclude that input RRI must be High first, then either or both priority chain has been passed to the nxt priority chip or the low priority chip generates runcode request, then the chip can release the RRO chain to the next chip.

The Video Data Strobe network includes two networks; one for Even data strobe and the other for Odd data strobe network. Each has pin (EVDS and OVDS) to connect to the network. What we called the broadcast network is because all pins are connected together, only one pin is a output to inform video data is being strobed into the line buffer in the Gold chip and the rest are inputs to receive the this video data storbe signal at one time.

The Video data strobe network has close relation with the priority chain. Only the chip has the priority in but not pass priority out, then the chip will swap video data strobe pin as output and generate a pulse to inform the rest chip in the network. Once the EVI (or ODI) is High and any Object Processor (at least one) is active in the chip, then the chip will process the even (or odd) pixel. If the processed pixel is not transparent, then the chip will assert pin EVDS to Low. Once an asserted EVDS is received by the chip with input EVDS, all the puxel offset counter will be incremented.

Figure 12. Principal functional block diagram of the Memory sequencer.

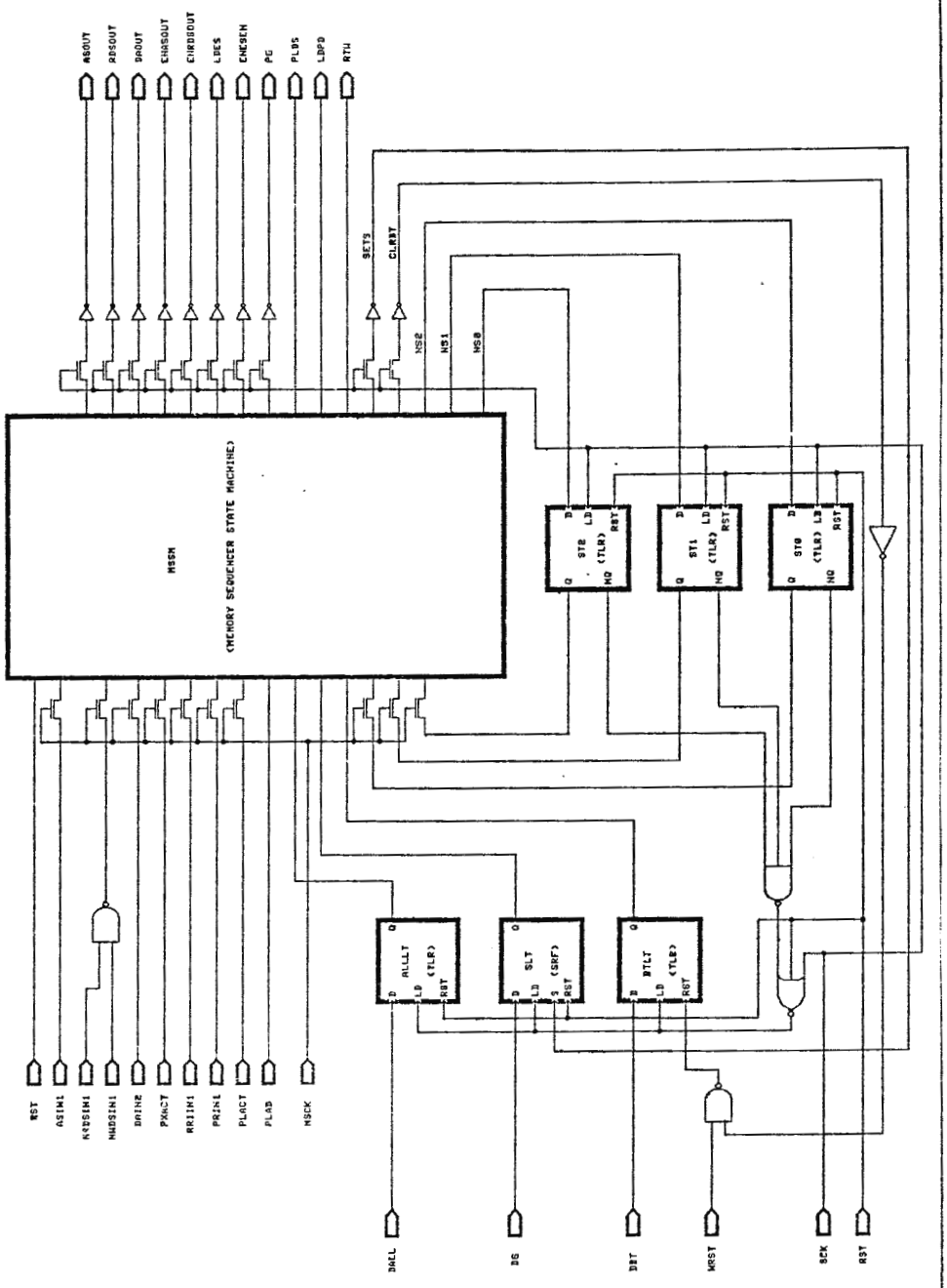
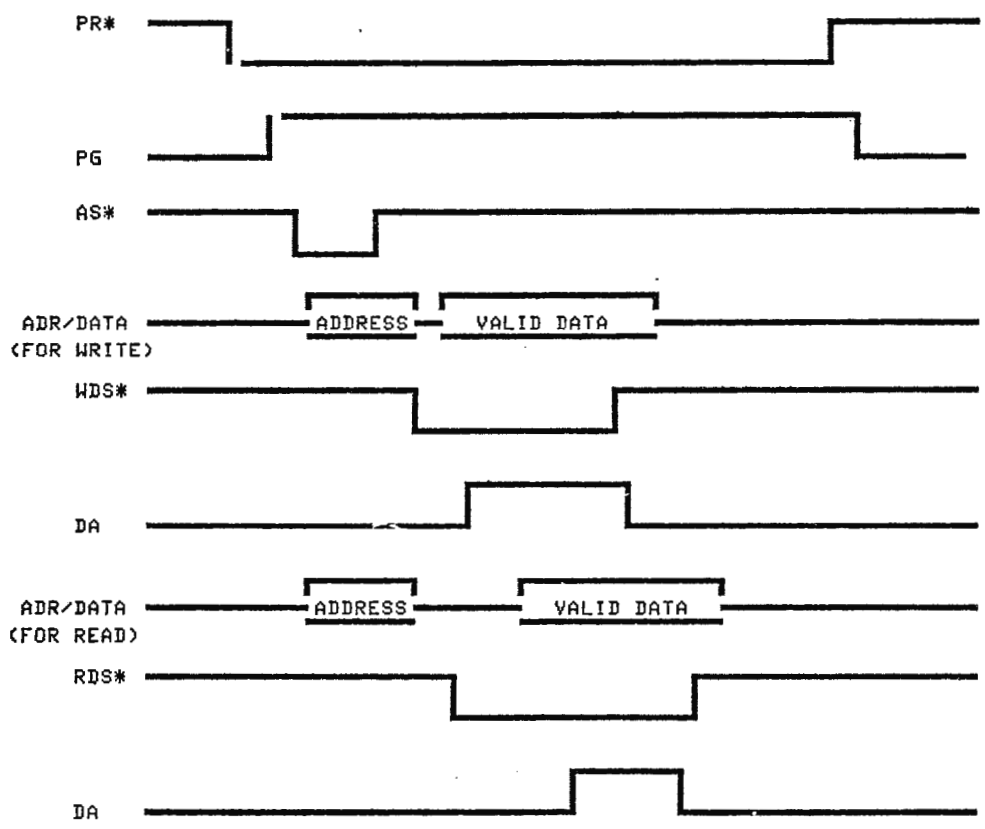
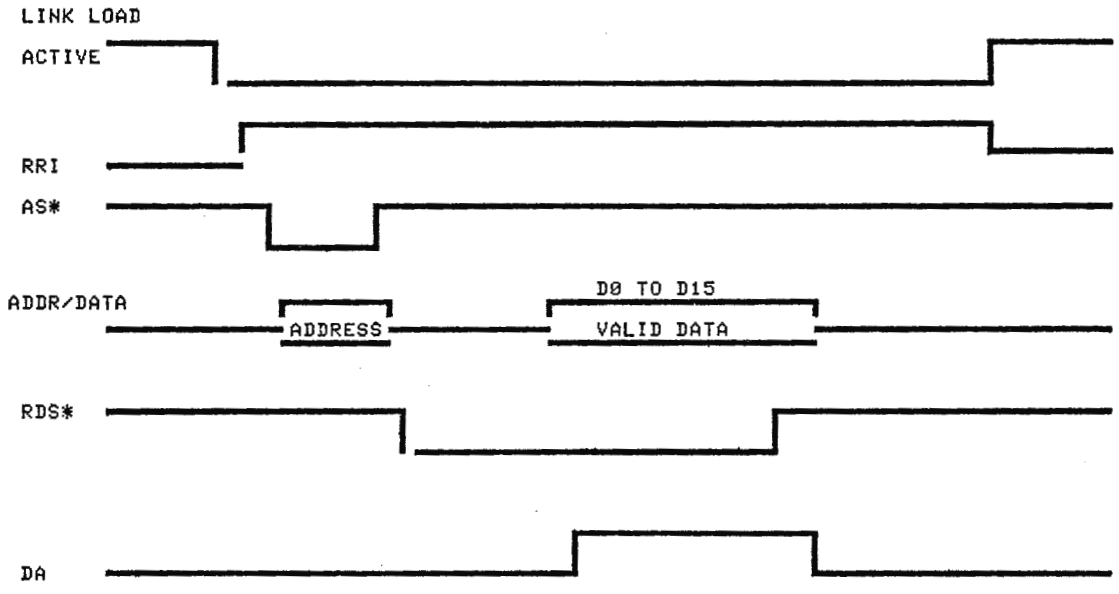


Figure 13. Timing diagram of the "Root/read" or "Root/write" memory cycle.



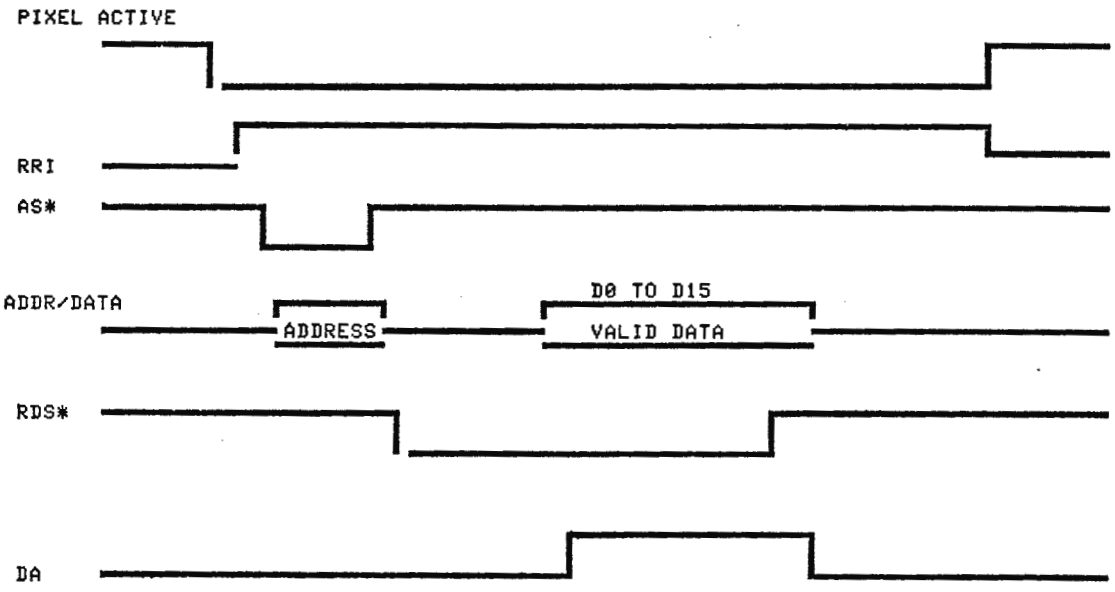
* THIS TIMING DIAGRAM IS FOR FIRST HALF ROOT.
THE SECOND HALF ROOT READ/WRITE JUST REPEATS THE SAME TIMGS.

Figure 14. Timing diagram of Parameter block load cycle.



* THIS TIMING DIAGRAM IS FOR FIRST WORD OF PARAMETER LOADING.
THE OTHER SEVEN WORDS HAS THE SAME TIMING DIAGRAM.

Figure 15. Timing diagram of Pixel fetch memory cycle.



6.10 Origin update sequencer block

The Origin Update Sequencer block is a very simple logic circuit block. Its main purpose is to provide the update origin information to the Address Generation Logic block as the starting address for fetching a pixel data of a new scan line. This block contains two registers: Origin and Stride registers and a one bit adder. Figure 16 shows the principal functional block diagram of the Origin update Sequencer.

6.10.1 Origin and Stride logic

The Origin register is a 20 bits register containing the Origin parameter which is a pointer to the upper left hand corner of the window. This register will loaded by the Memory sequencer through the Parameter Bus in two consecutive words. The low word (from bit 4 to bit 15) is loaded by signal Pld2 and upper word (from bit 0 to 6) is loaded by signal Pld3.

The Stride register is a 12 bit register containing the Stride parameter which is thedistance in words from a pixel in the picture to the pixel directly below it. This register is loaded by the Memory sequencer in a word fetch. Only from bit 4 to bit 15 of the word is loaded by signal Pld1 through the Parameter bus.

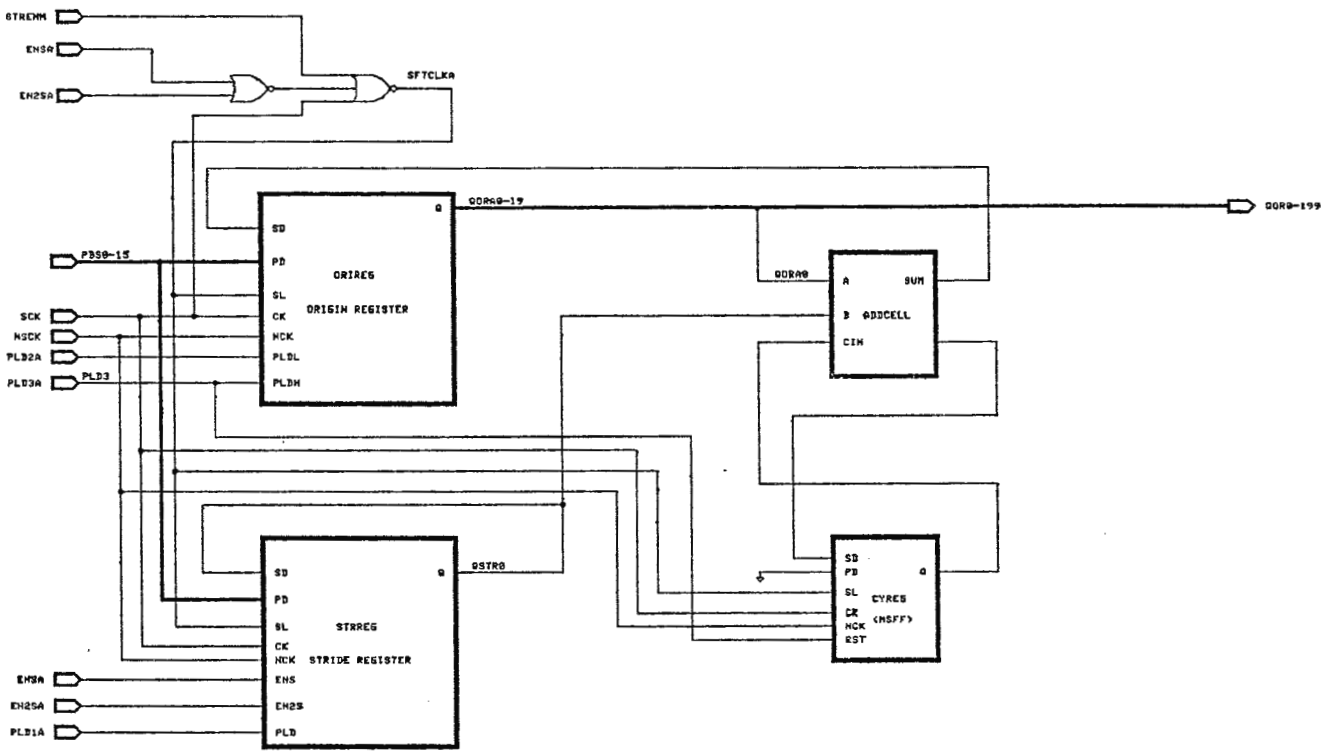
At the beginning of each line the Origin Update block will receive a signal called INIT1 which will trigger the contents of Origin and Stride are added together bit by bit serially through a one-bit adder. Whether Origin will be added one Stride or two times Stride depends on the inputs control signal ENS or EN2S which comes from the Line object Active block.

In the case of adding one stride, every bit in the Origin register is added the corresponding bit in the the Stride register one by from the least significant bit 0. whenever a carry occurs in the add operation, the carry will store in the Carry register and carry to the next bit addition. The adding operation will repeat 20 times for 20 bits.

In the case of ENS2, the content of the Stride register will be shift one bit position (which is the same as 2 times stride), then added to the Origin bit by bit.

Since the updated Origin is used as the starting address for Silver chip to fetch a new line, so the addition is needed to be done one line ahead. Although the adding operation is done bit by bit, the operation has a full scan line time to be completed.

Figure 16 Functional block diagram of the Origin Update Sequencer.



6.11 Address generation logic block

The responsibility of the Address Generation logic block is to set up the system bus for the information exchange between the Rainbow chip and the CPU as well as between the Rainbow chip and the memory. The CPU has to write into the Rainbow Silver chip a Root address for the Silver chip to load Parameter block for a particular object.

The address presented on the system bus by the Silver chip comes from three sources. The first source of the address is from the Origin update sequencer block which provides the memory address for the first pixel of a line. The second source of address comes from the Pixel Word Counter which points to the memory address for the next pixel word of the current line. The third source of address is from the Link register which is used as a pointerto load a set new parameters for a new object.

The Address Generation logic block contains various register and counter for holding parameter and generating new address. It contains Link register, Link counterand Pixel Word counter. Figure 17 shows the principal functional block diagram of the Address generation logic.

6.11.1 Link register & Link counter

Link register is a 20 bits register containing the Link parameter which is a pointer to the next object parameter block in memory. This parameter will be loaded into the Link register by the Memory Sequencer in two consecutive words. The low word of the Link parameter will be loaded by the signal Pld0 and the high word (actually only 4 least significant bit in this word belongs to Link will be loaded by signal Pld1. The content of the Link register is loaded onto the bus after every Object procesor is done the display. Every Object Processor has its own Link register.

The Link counter is a 20 bits counter which keeps the memory address for fetch the next parameter. When status is the link load active and a request from the Parameter Load logic block to load a new parameter, the link parameter will be loaded into the Link counter as the memory address for fetching

a new paprmeter. Each Object processor will need eight times accessing memory to complete a whole parameter block fetching. The link counter is counted up one word for one memory fetch.

6.11.2 Pixel word counter logic

The Pixel Word Counter is a 24 bits counter which consists of two parts, namely, the upper 20-bit Pixel address counter and the lower 4-bit pixel offset counter. The 20 bit pixel address counter keeps the memory address of the next memory word fetch in the current line. The 4-bit offset counter contains the offset position of the current pixel within a word.

The 4-bit counter will be able to increment by various number depending on the Depth Parameter. The incremental value will be 1, 2, 4 or 8, if the Depth value is 0, 1, 2 or 3, respectively.

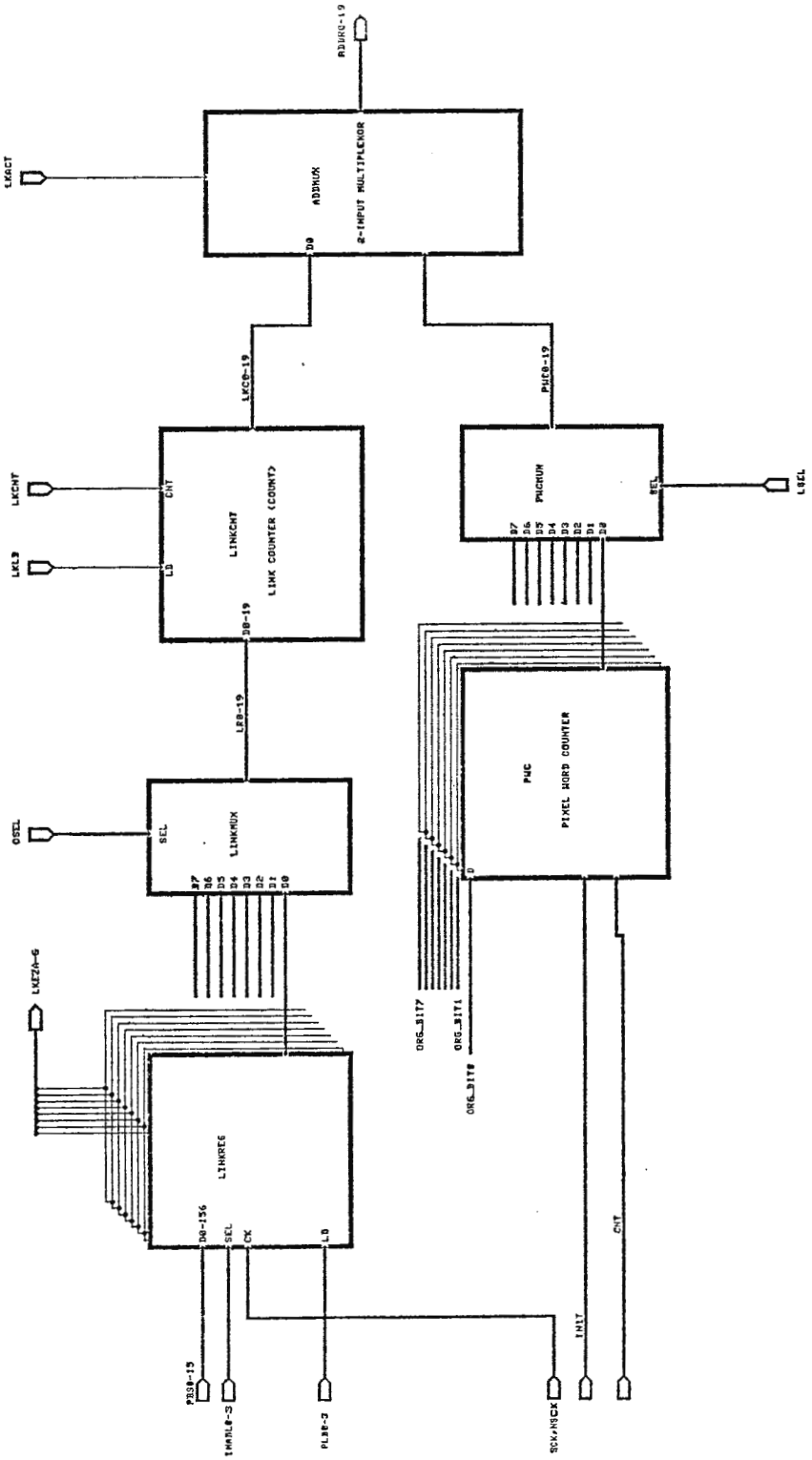
Instead of implementing the counter straightforwardly by adding different value, an alternative implement is adopted to save hardware circuits. Initially, the Pixel offset counter will be loaded with a value which is the Pixel offset value divided by 1, 2, 4 or 8 depending on the the value of Depth. The "division" of the pixel offset value is simply a right shift operation.

Each time a Next Pixel signal is asserted, the Pixel offset counter is incremented by one. If the current word boundary is reached, a Carry out signal will be generated. This Carry signal will serve as a signal to increment the 20-bit Pixel word counter and an indication for a new memory word fetch if more pixel are needed to be displayed for the san>me object.

The long 20-bit pixel address counter should be able to count as fast as possible for every word fetch to minimizing the gates propagation time. Instead of incrementing the Addresss counter right after the 4-bit offset counter reaches the word boundary, the increment of the Pixel Address counter will be overlapped with the data fetch operation. Once the system bus is available, a internal address strobe signa will latch the content of the Pixel Addresss counter into the address latch. On the rising edge

of the following system clock, the 20-bit pixel address counter will start to increment. This incrementing operation has to be finished in one and a half clock so that a new address can be ready for fetching the next pixel word. In terms of hardware, the counter is designed as a parallel counter in a 7-bit group and a ripple counter in 3 groups.

Figure 17. Functional block diagram of the Address generation logic block.



6.12 Parameter load logic block

During a scan line time, there are three statuses giving the Rainbow system what would be doing during that status period. One of the status is the Link Load Active status whcih indicates that the Silver chip can perform the parameters block fetch in this period. The parameter load logic block is responsible for taking the request from each Object Processor for parameter loading, encoding the priority and generate the proper parameter load sequence.

The Parameter load logic block consists of a Parameter load Request state machine, Parameter load Sequencer state machine, priority encoder and other combinational circuits. Figure 18 shows the principal functional block diagram of thw Parameter load logic.

6.12.1 Parameter load request logic

The parameter load request logic is basically a state machine. This state machine contains three states, namely, the Quiescent state, Root half written state and Parameter load request state. When is power on reset, the chip is in the Quiescent state. CPU will write the first half Root (a word data) into the Silver chip. On the conditions of both Root Write RTW and A1 address bit are Low, the state machine enters the Root half written state. Then CPU writes the second half Root into the chip. On the conditions of Root Write RTW, A1 address bit are High and the LInk register just being written is not zero (LINK = 0 not true), the state machine will enter the Parameter load request state. In the parameter load request state, it will generate Parameter load request signals PLR to the priority encoder and AONL to the Parameter load Sequencer state machine. Once the parameter load is operation is done, the state machine goes back to the Quiescent state.

When a Object Processor has just completed the display of a object, the Parameter load request state machine will receive a object done OD signal for this particular processor. If the content of Link register of this Object Processor is zero and object done OD signal is created, then the state machine will enter from the Quiescent state to the prameter load requesent

state. after the parameter load operation is completed (the PLDONE signal is generated), the state machine is back to the Quiescent state. However, inthe Parameter load state, if CPU writes a new Root to that Object processor(both RTW and A1 LOw conditions occures), then the state machine will abandon the current parameter fetch and enter the Root half written state. The parameter load state machine will use the new Root to catch new parameter block.

Either in the Parameter load request state or the Root half written state, if a RESET occurs or a bad address address abort occurs, the state machine will back to the Quiescent state.

The truth table of this state machine is listed in in the following and the state machine diagram is in in the Figure 19.

Input:								Pres. state		next state	
ABORT	RESET	LINK=0	PLDONE	OD	RTW	A1		PLR	PLH	PLR	PLH
1	X	X	X	X	X	X		X	X	0	0
X	1	X	X	X	X	X		X	X	0	0
0	0	X	X	X	1	0		0	0	0	1
0	0	0	X	1	0	X		0	0	1	0
0	0	1	X	X	1	1		0	1	0	0
0	0	0	X	X	1	1		0	1	1	0
0	0	X	1	X	X	X		0	0	0	0
0	0	X	0	X	1	0		1	0	0	1

RTW : Root Write Status
A1 : Root address bit 1 (RADR1)
LINK=0 : Current Link pointer addressed by OSEL is zero
OD : Object Done [Object is currently on the last line of display]
PLDONE : Parameter Load Done
ABORT : Status from Gold chip aborting current parameter fech
RESET : Status from Gold chip indicating it has Reset and inform the Silver to do the same.
PLH : Parameter Load Halt [Object has received half root comand]
PLR : Parameter Load Request [Object has either completed display or a non-zero root write]

6.12.2 Parameter load sequencer logic

The Parameter load sequencer logic is basically a state machine. This state machine contains four states, namely, state 0-No object requires load, state 1-initialization for parameter load, state 2-parameter load currently active and state 3-parameter load currently suspended.

When the chip is power on reset, the chip will be in the state 0. No output is active in this state.

When a parameter load request signal AONL comes in, the state machine changes to the state 1. The state machine generates a LDLOR signal to load contents from the priority encoder to the Loading Object register. Also a signal LDLC goes to the Link Address Counter to clear content. Meanwhile, if the status line is Link Load Active, the state machine enter the state 2. If the status is not the Link Load active, then the state machine enters the state 3.

In the state 3, it turns off the signals LDLOR and LDLC. Once the status changes to the Link load active, the state machine will swap to the state 2.

In the state 2, it turns off the signals LDLOR and LDLC. But it turn on the signal PLACT to inform the memory Sequencer to start a parameter fetch cycle.

In the state 2, if status changes from the Link load active to the Refresh active, then the state machine will swap to the state 3. If the whole parameter block has been completed, the state machine will change to the state 0. In other words, Once eight word memory fetches are completed in the Memory Sequencer, then a signal PLDONE from the Memory Sequencer will be sent to the Parameter load logic block to change to state 0.

In the state 2, if the state machine receives a PLHALT which means a half root written occurs, the state machine will change from state 2 to state 0. This change indicates that contents in the link register is going to be changed to a new one. Therefore, no need to proceed to fetch parameter by using old link parameter.

The truth table of this state machine is listed as follows and the state diagram is shown in the Figure 20.

Input:					OUTPUT:						
					Pres. state		next state				
RESET	ANOL	PLST	PLDONE	PLHALT	R0	R1	R0	R1	PLACT	LDLOR	LDLC
1	X	X	X	X	X	X	0	0	0	0	0
0	1	X	X	X	0	0	0	1	0	1	1
0	X	0	X	X	0	1	1	1	0	0	0
0	X	1	X	X	0	1	1	0	1	0	0
0	X	1	X	X	1	1	1	0	1	0	0
0	X	0	0	0	1	0	1	1	0	1	0
0	X	X	1	X	1	0	0	0	0	0	0
0	X	X	X	1	1	0	1	0	0	0	0

State 0 (R0=0,R1=0): No object requires load
State 1 (R0=0,R1=1): Initialization for parameter load
State 2 (R0=1,R1=0): Parameter load currenrly active
State 3 (R0=1,R1=1): Parameter load currently suspended

AONL: Any object needs load
PLST: Parameter load active status
PLDONE: Parameter load done
PLHALT: Parameter load halted [Half root written for currently active object]
PLACT: Parameter load active
LDLOR: Load content from priority encoder to the loading object register.
LDLC: Clear link address counter.

6.12.3 Combinational logic

The priority encoder is used to solve the priority among those eight objects' requests if there are more than one objects request the parameter load. The Loading object register is a latch register to store the highest priority object's parameter request.

Figure 18. Principal functional block diagram of the Parameter load sequencer.

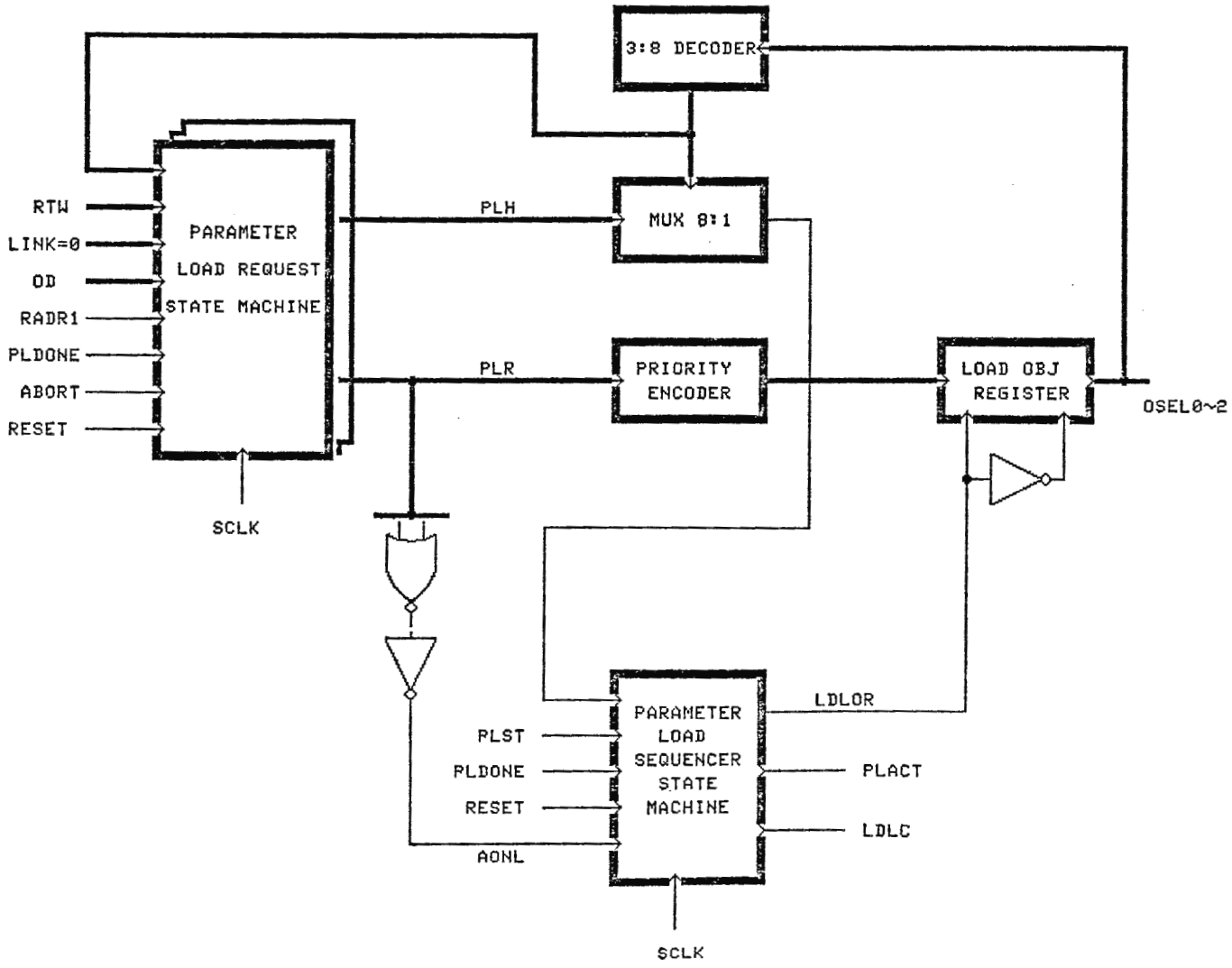


Figure 19. The state machine diagram of the Parameter load request.

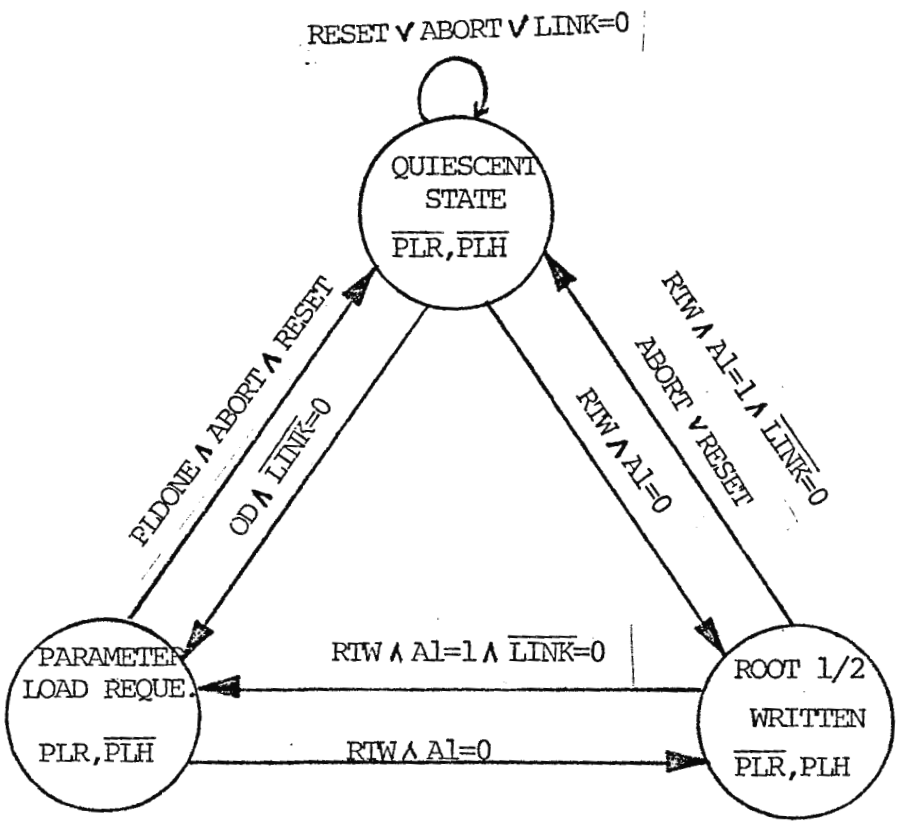
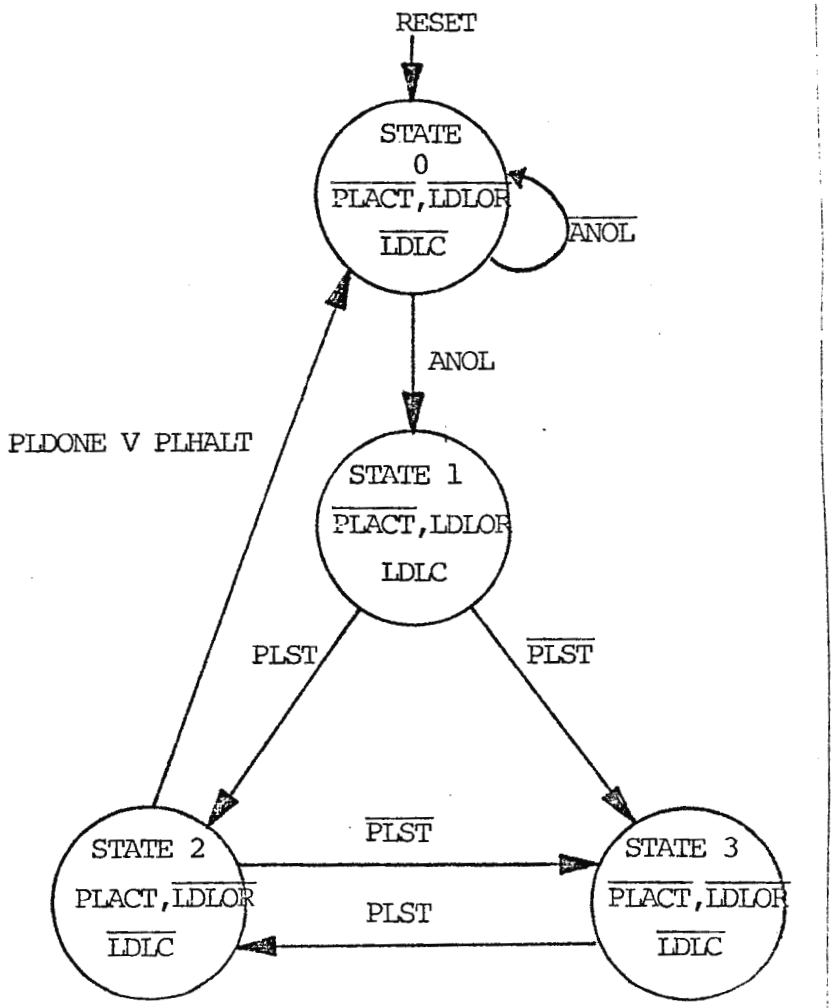


Figure 20. The state machine diagram of the Parameter load sequencer.



6.13 Data and status bus interface logic

The data and status bus interface logic is responsible for data bus input and output buffer control, and decodes the status lines form the Gold chip. This is a very simple logic block which is shown in the figure 21.

6.13.1 Data bus interface logic

The data transfer operations on the Data bus include 1) CPU writes data into the registers of the Silver chip (i.e. Root write), 2) CPU reads data from the registers of the Silver chip (Root read), 3) Silver chip read pixel data from the system memory, 4) Silver chip read parameter block from the system memory.

Due to the Prechargeing technology using for the design of the internal bus, PBUS, data transfer on this bus can only happen during phase 2 of internal clock. For a write operation, a latch is enabled by the WDS input and requried to save the data provided by the CPU. At this moment, the input buffer is opened. For a read operation, data stored in the registers have to be transfered to the data latch during RDS active period. Of course, the output buffer is enabled at this time.

6.13.2 Status line outputs

The Gold chip has three status line outputs to provide information to the Silver chips. These three lines are S0, S1 and S2 which can be decoded into eight different conditions. By using this information, the Silver chip can perform some actions such as fetching pixels or loading parameters in the correct sequence.

The following table gives the decoding information of three status line outputs:

S2	S1	S0	Status Description
0	0	0	Refresh Active
0	0	1	No operation
0	1	0	Abort memory cycle
0	1	1	Reset
1	0	0	Top of screen in Even field
1	0	1	Top of screen in Odd field
1	1	0	Pixel Active
1	1	1	Link Load Active

In general, the whole scanning time of a horizontal scan line can be divided into three active intervals. They are "Refresh Active", "Pixel Active" and "Link Load Active".

The "Refresh Active" means the memory system is refreshed during this interval. This Refresh active interval starts right at the falling edge of the Horizontal Sync. The duration of this interval is approximate 10 Gold clocks cycles. During this interval, the Gold chip provides memory refresh address and refresh initiating signal AS (Address Strobe) to the external memory refresh circuit. Then the memory refresh circuit will generate Row Address Strobe RAS and respond with an DAData Acknowledge. The Gold chip will provide five memory addresses sequentially during the entire Refresh Active.

The "Pixel Active" means that the Silver chip can access the memory system and fetch the pixel data. This pixel Active interval starts right after the Refresh Active. The duration of this interval is not fixed and depends on the complexity of the object. In other words, the whole duration is counted on how much time spending on fetching the pixel (the number of memory cycles). Therefore, the Gold chip has pixel counter to calculate time of this interval. At the beginning of this interval

During the video pixel loading period (i.e. from one scan line before starting the display to one line before ending the displya on CRT), the correct sequence of status generated by the Gold chip in order are Refresh Active, Pixel Active and Link Load Active. When loading of Line buffer is incomplete, then status only occurs Refresh Active and Pixel Active.

During noloading of video pixel period (i.e. a whole field period minus the pixel loadine period), the correct sequence in order is Refresh Active , then Link Load Active. However, at two scan lines before data actually displayed on CRT, the Gold chip inserts the Top of Screen in (Even or Odd) field status between Refresh and Link Load Active statuses.

When the Gold chip receives either bad address or Reset signal from outisde, it will generate corresponding status at any time during a scan line.

Figure 22 will show the status timings of the Rainbow system.

Figure 21. principal functional block diagram of the data/status bus interface.

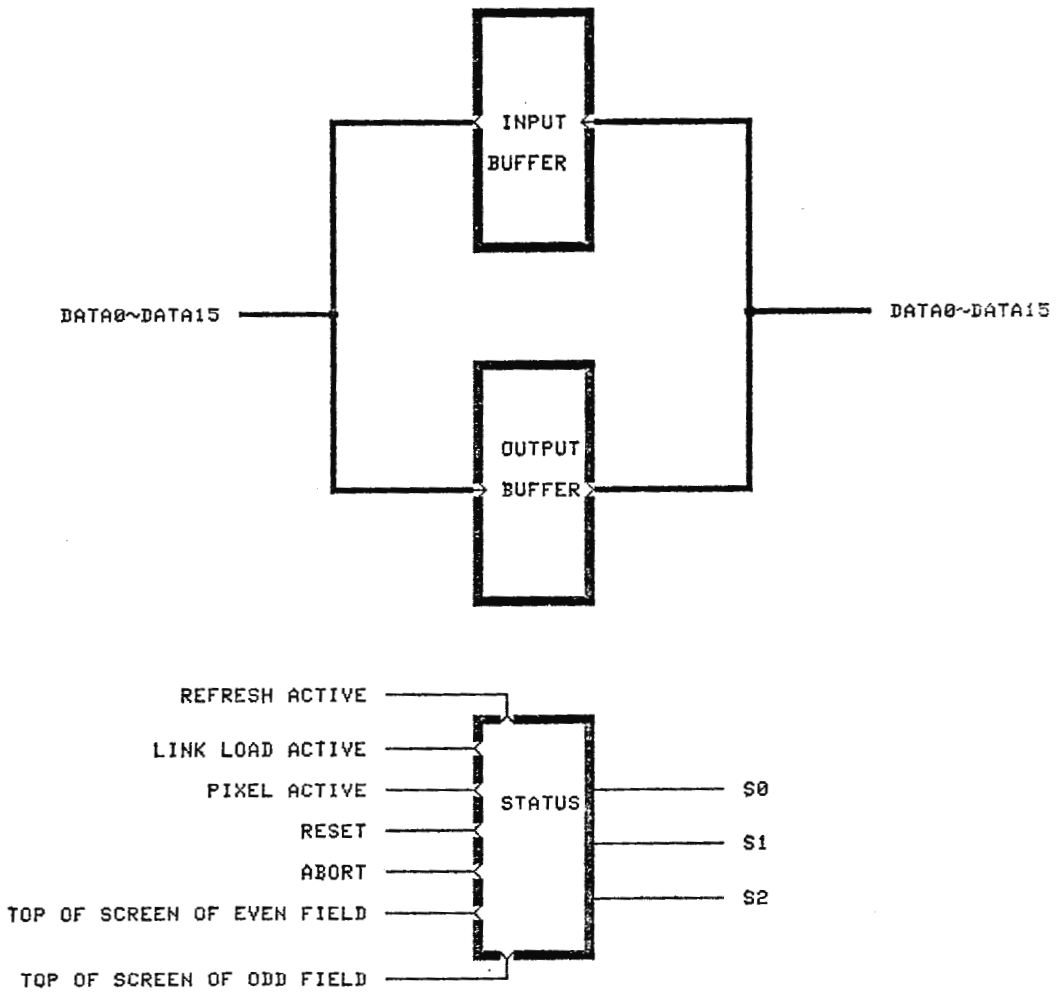
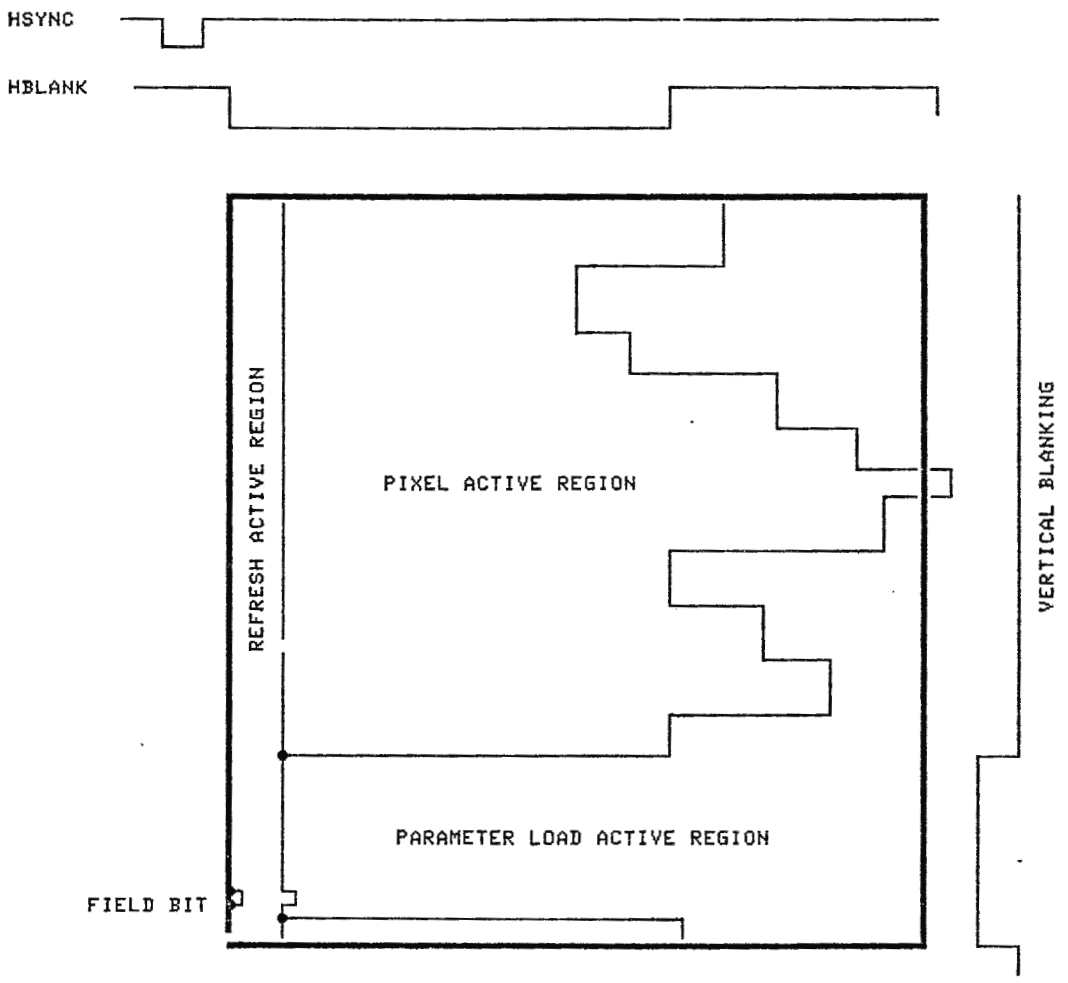


Figure 22. Raibow Status timings diagram.



6.14 Rainbow system configuration

The minimal Rainbow system is made from two chips Silver and Gold. The Silver chip contains circuitry to read pixel data from memory, process the information and then send the data to ghe Gold chip. The Gold chip provides circuitry forthe video timing, interrupt, programmable registers and line buffer. In addition, several logic circuits are needed by Rainbow system to interface with system bus and CPU.

Figure 23 shows the relationship of the Silver chip and Gold chip with the Color Map, CRT and System bus.

The "Interface Logic" in the figure 24 is a logic circuit that takes care of System Bus request and grant for the Rainbow system.

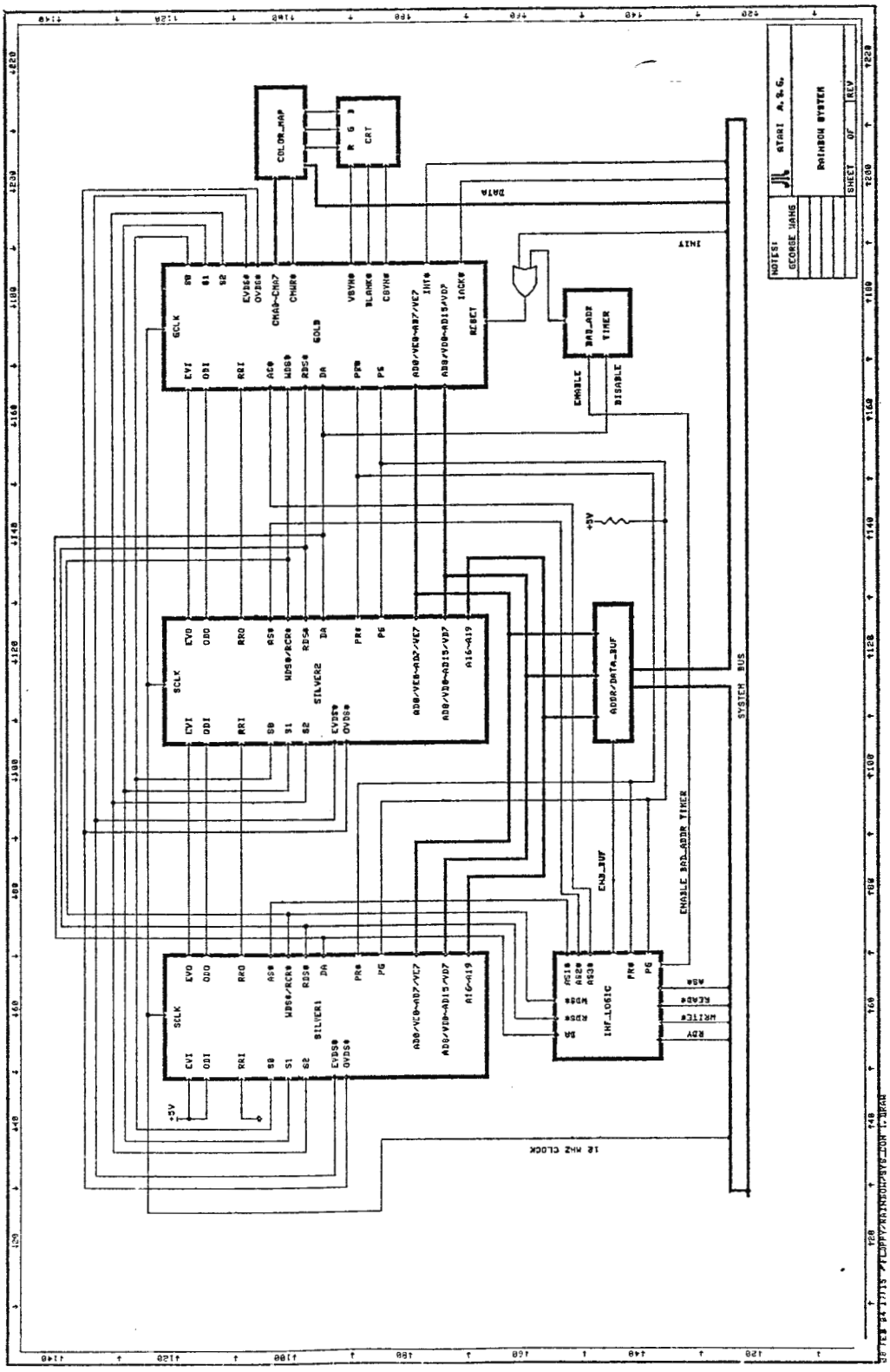
The "Address/Data buffer" is used for interface between the address/data lines of System Bus and address/data lines of Rainbow system.

The "Bad Address Timer" is a timer to detect a bad address memory access. This timer can be designed to use Address Strobe AS to start counting and Data Acknowledge DA to stop counting. If the counting exceeds a certain value (which is designed by user), then the timer will generate a signal to the Gold chip.

The "Bus Arbiter" takes access request from CPU and generate Processor Request to Rainbow system. Also it performs Bus arbitration between CPU and Rainbow system.

How to implement these Logic circuit depends on what kind of system that the Rainbow system wii fit in.

Figure 23. Rainbow system configuration.



7. Parameter Register organization

The object's parameter block contains information specifying the object's display representation in memory, the object's location on the screen, its color and its spatial properties. Parameters are linked in a list structure. Each parameter begins on a word boundary.

To CPU the Link register is the only addressable parameter in the Silver chip. The rest of parameters in the Parameter block are only accessed internally.

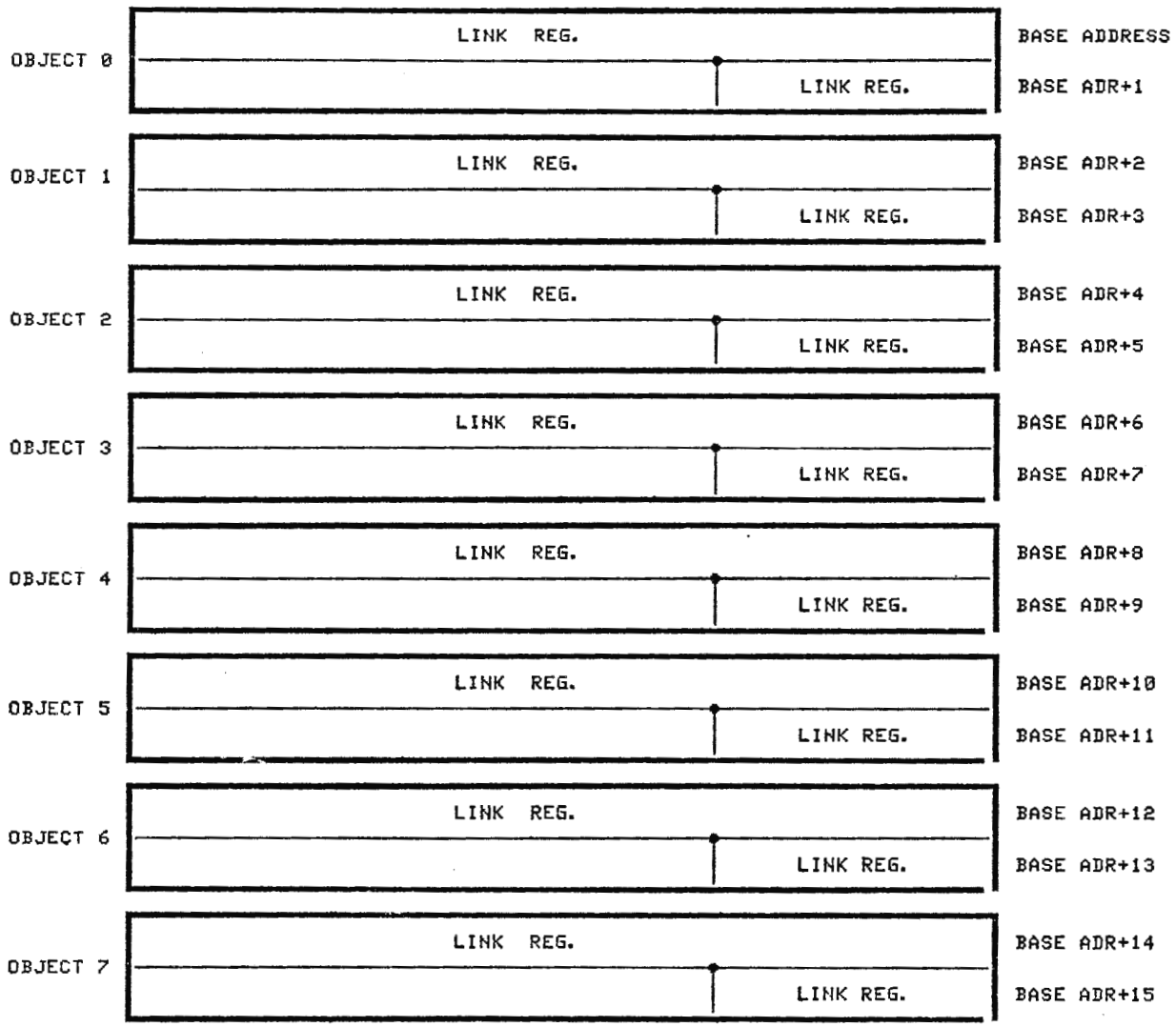
7.1 Addressable registers

Link is the absolute address of the beginning of the next Parameter block to be displayed. Root is the address of the start of the first parameter block and represents the only value written to an object processor by the CPU. Both Link and Root value share the same register called link register. The only difference between the Link and Root is the Root points to the start position of the first parametr block , but the Link points to the next parameter block for the same Object processor. If the Link or Root of an Object processor is zero, no further object will be interpreted by that Object procesor until a new root is written. The parameter are completely reloaded either a non-zero Root is written or at end of display of the current object7s window.

Each Link parameter is a 20 bit information which occupies two word memory. The first 16 bits locate in the lower word and the four most significant bits are in the higher word. Since there are eight Object processors in the Silver chip, each object processor has its own Link register. Each Link register is assigned two consecutive addresses (the lower value is for the lower word and the higher value is for the higher word).

Object processor 0 has highest priority and the Object processor 7 has the lowest priority. The addresses assigned to the Link register of the Object Processor 0 is the "Base address" and the "Base address + 1". The address for the Link register of the Object processor 1 are "Base address + 2" and "Base address + 3", and so on so forth. Here the "Base address" is the address selected by the System Designer for decoding to select the Silver chip. Figure 24 shows the layout o f these Link register.

Figure 24. Link registers and its assigned address.



7.2 Parameter block layout

A general view of the parameter block is shown in the Figure 25. To CPU, only the Link parameter register of the Silver is addressable and is described in the last section. To Gold chip, this parameter block is stored in external memory and will be fetched into the internal registers by eight consecutive fetches (word addresses). The Root address is the starting address of the parameter block and will be written into the Gold chip by CPU during the initialization and/or the vertical blanking time. Bit description of each parameter is described as follows:

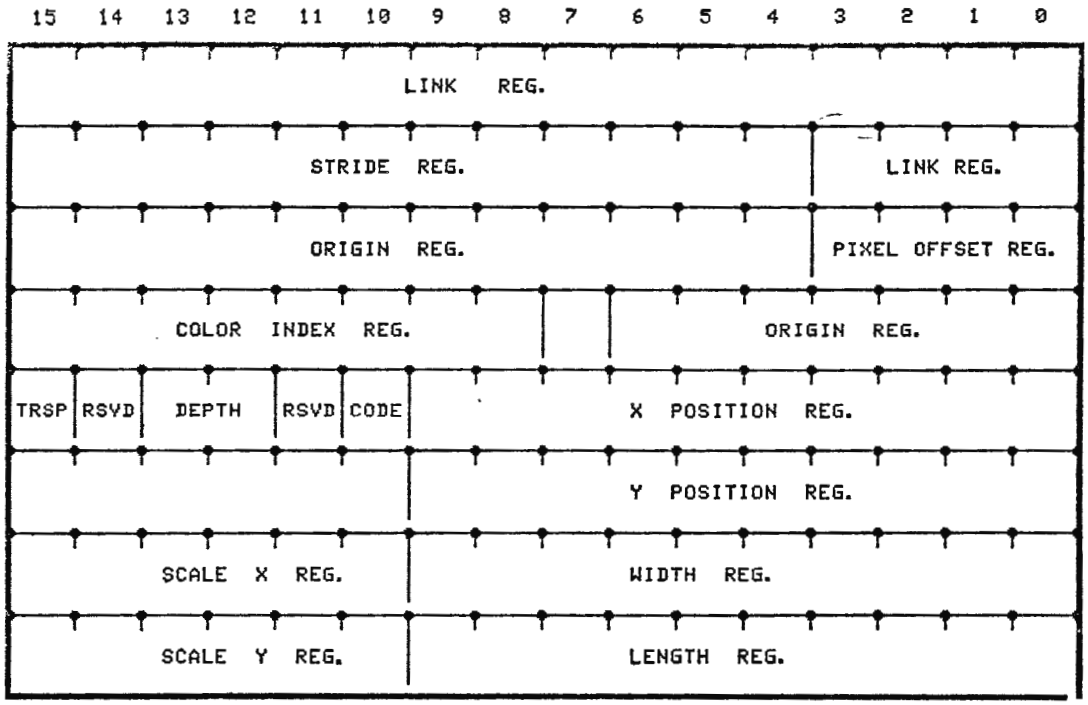
Link parameter

The Link parameter is a pointer to the next object parameter block in memory (word pointer, not a byte address). The Object processor reloads parameters upon the completion of the display of the current object (20 bits). This parameter is stored in the memory word address as Root addresss, Root + 2.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LK15	LK14	LK13	LK12	LK11	LK10	LK9	LK8	LK7	LK6	LK5	LK4	LK3	LK2	LK1	LK0
												LK19	LK18	LK17	LK16

Bit	Mnemonic	Description
15-0	LK15-LK0	Bits from LK15 to LK0 are the lower word locating in the internal register R0 and bits from LK19 to LK16 are higher word locating in the register R1.

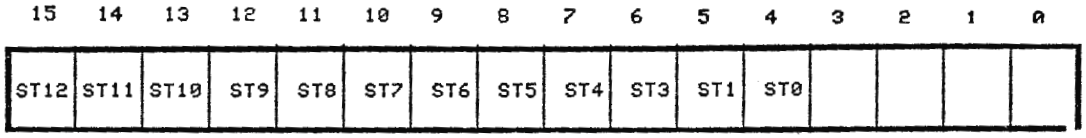
Figure 25. The layout of the parameter block.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	REGISTER
LK15	LK14	LK13	LK12	LK11	LK10	LK9	LK8	LK7	LK6	LK5	LK4	LK3	LK2	LK1	LK0	R0
ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0	LK19	LK18	LK17	LK16	R1
OR12	OR11	OR10	OR9	OR8	OR7	OR6	OR5	OR4	OR3	OR2	OR1	P3	P2	P1	P0	R2
CI7	CI6	CI5	CI4	CI3	CI2	CI1	CI0		OR19	OR18	OR17	OR16	OR15	OR14	OR13	R3
T0	RSVD	D1	D0	RSVD	C0	X9	X8	X7	X6	X5	X4	X3	X2	X1	X0	R4
						Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	R5
SX5	SX4	SX3	SX2	SX1	SX0	W9	W8	W7	W6	W5	W4	W3	W2	W1	W0	R6
SY5	SY4	SY3	SY2	SY1	SY0	L9	L8	L7	L6	L5	L4	L3	L2	L1	L0	R7

Stride parameter

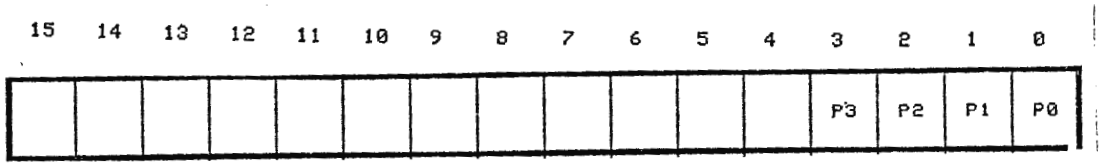
The stride parameter is the distance in words from a pixel in the picture to the pixel directly below it. This is a 12 bits parameter which is stored in the memory word address: Root + 2.



Bit	Mnemonic	Description
15-4	ST11-ST0	Stride bits; ST11 is the most significant bit and ST0 is the least significant bits. These bits are located in the internal register R1.

Pixel Offset parameter

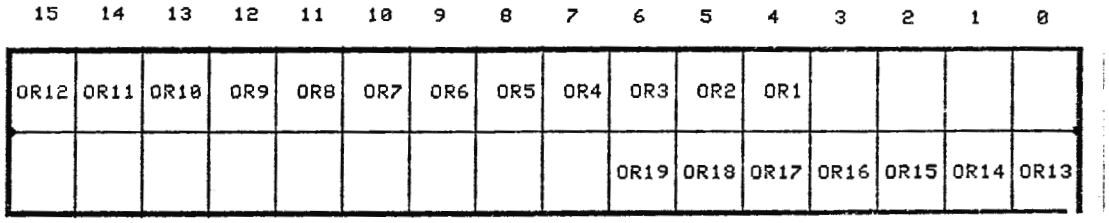
It is a parameter to tell the exact position of the first pixel in the memory word which is addressed by the Origin parameter. This parameter has close relation with the Depth parameter. For nstances, if the Depth value is 0, the pixel offset value could be any number from 0 to 15. If the Depth value is 1, the pixel offset value could be 0, 2,... or 14. If the Depth value is 2, then the pixel offset value could be 0, 4, 8 or 12. If the Depth value is 3, the noly two values 0 or 8 will be the pixel offset value. This parameter contains four bits which is stored in the memory word address: Root + 4.



Bit	Mnemonic	Description
3-0	P3-P0	Pixel offset bit; P3 is the most significant bit and P0 is the least significant one. These bits are stored in the internal register R2.

Origin parameter

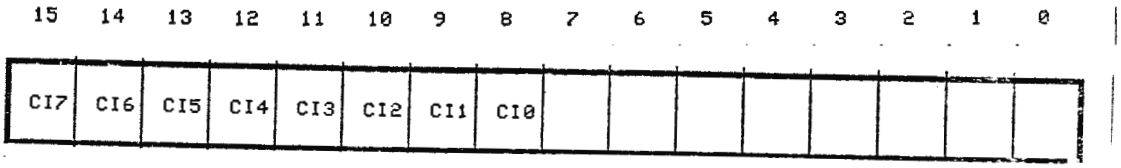
It is a pointer to the upper left hand corner of the window. This value is made up of a 20 bit address "word pointer" which is stored in word addresses: Root + 4 and Root + 6. Since it is a word pointer, the least significant bit is zero. Therefore, only 19 bits are needed in the Parameter block.



Bit	Mnemonic	Description
15-4	OR12-OR1	Origin bits; bits from OR12 to OR1 are lower part of this 19 bits value and bits from OR19 to OR13 are higher part of this Origin parameter.

Color index parameter

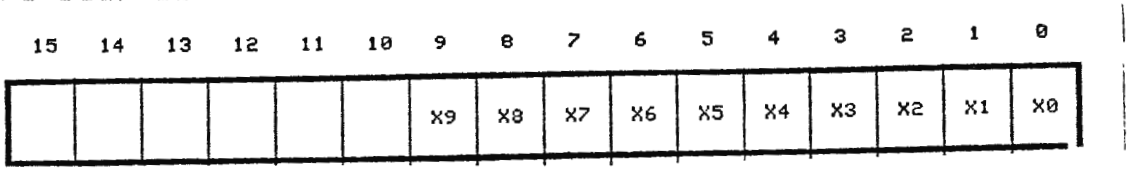
This index is used as a base pointer which is added to the pixel data to give the effective address in the color map. Color index effectively points to a range of adjacent colors in the color map that are available to the object. This is a eight bits parameter which is stored in the memory word address: Root + 6.



Bit	Mnemonic	Description
15-8	CI7-CI0	Color Index bits; CI7 is the most significant bit and CI0 is the least significant bit.

X parameter

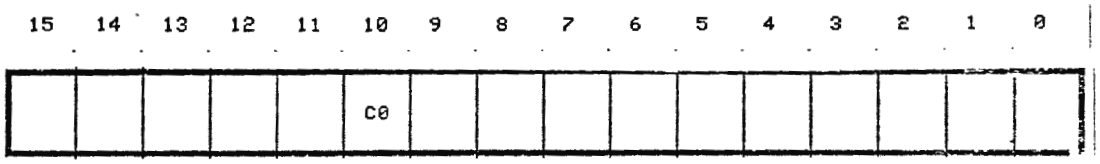
It gives horizontal position of a object's left edge. It is measured in screen pixels from left of the sreen, this parameter contains 10 bits value which is stored in memory word address: Root +8.



Bit	Mnemonic	Description
9-0	X9-X0	X position bit; X9 is the most significant bit and X0 is the least significant bit. These bits are stored in the internal register R4.

Code parameter

This parameter selects coding format of source format of source pixel data: "bitmap" style data (as described by Depth) or run-length coded data. Code parameter contains one bit value which is stored in the word memory address: Root + 8.



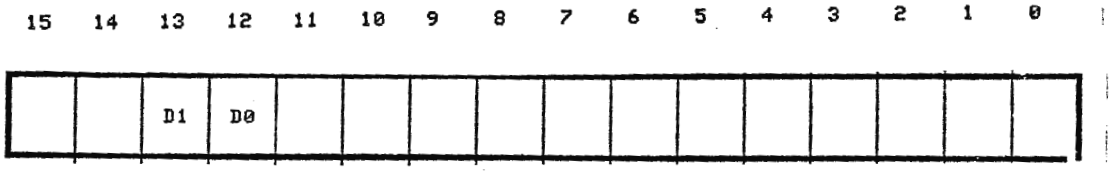
Bit	Mnemonic	Description
10	C0	Coding bit; C0 = 0 means the pixel data is stored in the bitmap format, C0 = 1 means the Pixel data is stored in the run-length format.

Depth parameter

It selects number of bits used to resprent each source pixel. The relationship betwen Depth, pixels per word and the number of different colors that a pixel could assume is:

Depth	Pixel/word	bits/pixel	colors
3	2	8	256
2	4	4	16
1	8	2	4
0	16	1	2

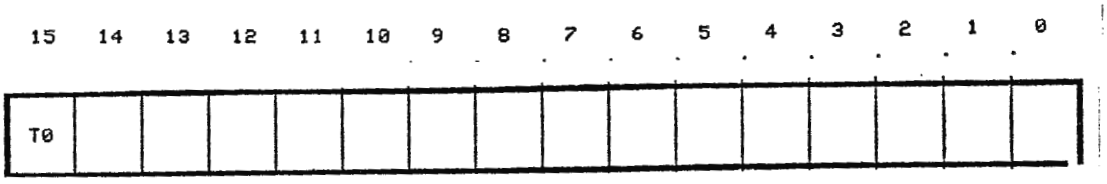
This parameter contains two bits value which is stored in the memory word address: Root + 8.



Bit	Mnemonic	Description
13-12	D1-D0	Depth bits; D1 is the most significant bit and D0 is the least significant bit. These bits are stored in the internal register R4.

Transparent parameter

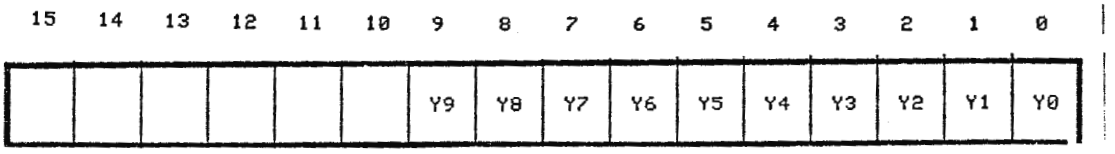
It selects what the pixel data with a zero value means. If transparent, the current color is to be taken from lower priority objects or background. If not transparent, the color index provides the address into the color map. This parameter contains one bit value which is stored in the memory word address: Root + 8.



Bit	Mnemonic	Description
15	T0	Transparent bit; T0 = 0 means no transparent pixel in this object, T0 = 1 means the display object has transparent pixel.

Y parameter

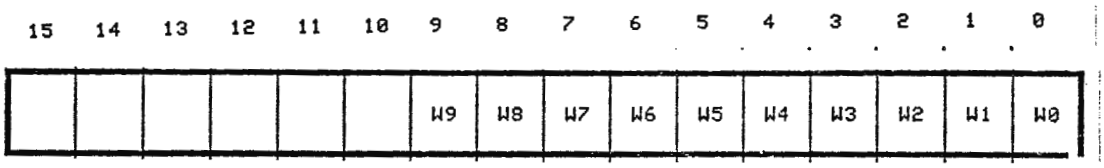
It gives vertical position of object's top line which is measured in screen lines from top od the screen. This parameter contains ten bits value which is stored in the memory word addresss: Root +10.



Bit	Mnemonic	Description
9-0	Y9-Y0	Y position bits; Y9 is the most significant bit and Y0 is the least significant bit. These bits are stored in the internal register R5.

Width parameter

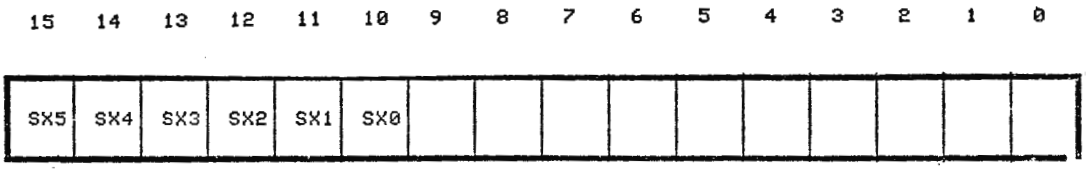
This parameter specifies the horizontal size of a window of a displayed object. If no scaling, this value is equal to the horizontal size of an actual displayed window which might not be equal to the horizontal size of a object times scale factor. This parameter contains 10 bits value which is stored in the memory word address: Root +12.



Bit	Mnemonic	Description
9-0	W9-W0	Width bits; W9 is the most significant bit and W0 is the least significant bit. these bits are stored in the internal register R6.

Scale X parameter

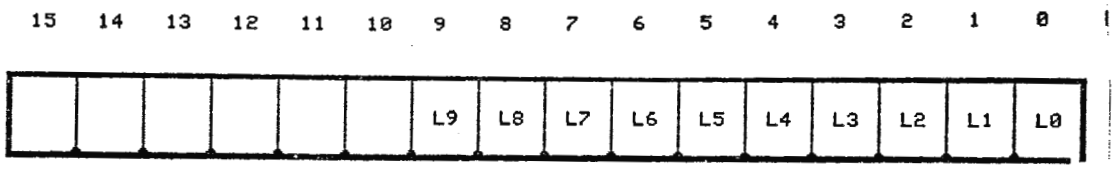
This parameter specifies the horizontal scale magnification factor of a object. Each pixel of source pixels will appear Scale X +1 times (i.e. value 0 means full horizontal resolution). It contains 6 bits which is the memory word address: Root + 12 and the maximum value is 63.



Bit	Mnemonic	Description
15-10	SX5-SX0	Scale X bits; SX5 is the most significant bit and SX0 is the least significant bit.

Length parameter

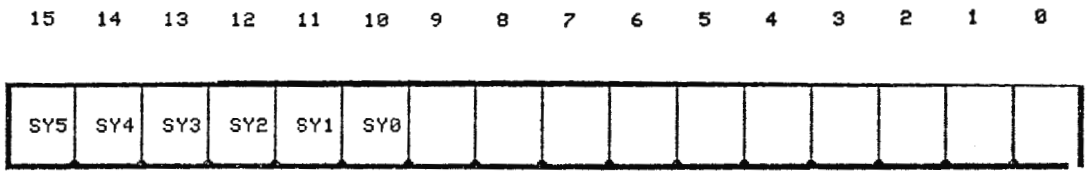
This parameter specifies the vertical size of a window of a displayed object. If no scaling, this value is equal to the vertical size of a object. If scaling, this number is the length of an actual display window but might not be value of scale factor times vertical length of a object. This is a ten bits value which is stored in the memory word address: Root + 14.



Bit	Mnemonic	Description
9-0	L9-L0	Length bits; L9 is the most significant bit and L0 is the least significant bit. Thses bits are stored in the internal register R7.

Scale parameter

This parameter specifices vertical scale magnification factor. Each line of source pixels will appear Scale Y + 1 times (i.e. value 0 means full vertical resolution). Interlaced scan is taken into account. This is a six bits value which is stored in the memory address: Root + 14.



Bit	Mnemonic	Description
15-10	SY5-SY0	Scale Y bits; SY5 is the most significant bit and SY0 is the least significant bit. These bits are stored in the internal register R7.

8. MAXIMUM RATINGS

Storage Temperature	-65 to +150 C
Ambient Temperature under Bias	0 to +70 C
Voltage at Pin releative to Ground	-0.5 to +7 C
Power Dissipation	750 mW

note: beyond the maximum ratings useful life may be impaired.

9. CAPACITANCES

Ambient Temperature Parameters: T_A=25 C; V_{cc}=GND=0V

Symbol	Parameter	Min	Max	Units	Test Cond.
C_{in}	Input Capacitance		10	pF	
C_{out}	Output Capacitance		15	pF	