4.0 Display Controller and Clocks

4.1 Overview

This chapter describes the Dandelion display controller. It is located on the high speed I/O (HSIO) board. Only the Display hardware is covered. The minimum microcode requirements are given.

4.2 Display Functions

The Dandelion large format display has the following parameters:

* 10" high by 12.8" wide bit map display.
* Separate Video, Horizontal and Vertical sync signals.
* Visible area = 808 lines x 1024 bits.
* Refresh rate = 38.7 frames/second (one frame every 25.8 ms)
* Memory used (808+16)*64 = 52,736 words in low 64k bank (16 lines for cursor).
* Border area = 26 lines at top, 26 lines at bottom, 32 bits at each side. Contents of user-settable register is repeated to form border pattern. Size of top and bottom borders set by microcode.
* Total frame (visible + non-visible) = 897 lines x 1088 bits.

The display hardware supports the scrolling of windows on the screen. These windows and cursors may be moved or scrolled vertically without actually moving bits in memory. Horizontal displacement requires the memory images to be moved.

Memory refresh is also performed by the display microcode.

4.3 Display Controller Hardware

The display controller uses a partitioned, two-port memory to reduce the loss of processor bandwidth while the display is running. The display controller blocks processor access to the low 64K memory bank *only* when it is acquiring data bits during an active horizontal line. The processor has complete access to the low bank at all other times (i.e. during one click in each round while the picture is being displayed, while the beam is turned off (blanking) and while the border is being displayed). When not being used by the display hardware, the low memory bank is identical in performance to the high banks. The display hardware cannot access the higher banks of memory and has no effect on processor access to these banks.

The following functions are performed by the display controller hardware/microcode.

1. Read data from memory and shift out blocks of 1024 bits.

2. Provide horizontal sync, vertical sync, and blanking signals.

3. Perform memory refresh.

Some versions of display microcode will automatically display a 16x16 cursor given its position. Others support smooth (continuous) scrolling of display windows. The hardware is constructed to support these features but does not supply them directly.

4.4 Partitioning Functions Between Hardware and Microcode

The tasks required of the display controller span a wide range of times (shifting bits, reading words, providing blanking and sync signals and composing fields and frames). It is important to minimize the amount of hardware used for any individual Dandelion controller while not requiring an excessive amount of the processor for a single I/O function. For the display controller, a horizontal line period (28.8 uS) was chosen as the dividing point between functions implemented in hardware and microcode. Memory accesses, parallel to serial conversion, and horizontal sync generation are done in hardware. Line counting, vertical sync, cursor insertion, scrolling support and memory refresh are handled with microcode. The hardware is capable of displaying only a single horizontal line. The microcode assembles the lines necessary to make a coherent picture.

4.5 Microcode - Hardware Interface

Display microcode uses three registers to control the display hardware. They are described below and summarized in the next figure. Use of this interface to operate the display will be described in the next section. The following terms appear in the discussion.

Line Segment - A subset of a horizontal line in which the displayed words come from contiguous memory locations. A line segment can be between 1 and 64 words long. The line segments which comprise a horizontal line must total 64 words in length. Each entry in the control FIFO (First-In-First-Out buffer) described below specifies one line segment.

Window - A rectangular region on the display made up of line segments on successive scan lines. The boundaries of the windows considered here are horizontal or vertical. The hardware does not preclude windows of arbitrary shape.

Cursor - This is a special case of window which is 16 scan lines high and two words wide. Contained in this region is a 16x16 array which is bit aligned. The remaining area in the two word wide area not covered by the 16x16 array is typically loaded with those bits from the main bit map over which the cursor is placed. The resulting image shows a 16x16 bit-aligned cursor.

Control Register

This register contains 7 bits which control the display operation.

> On - This bit enables requests to the processor for service during the display click. These requests begin at the end of every horizontal line and end when disabled by the display microcode. This bit does not affect memory accesses nor does it cause picture or border to be displayed. Its only function is to allow the processor to execute display-task microcode.
>
> Blank (Bk) - Setting this bit always causes the video beam to be turned off. No memory accesses will occur when this bit is set. Typically, the blanking bit will be set during vertical retrace.
>
> Picture (Pic) - Setting this bit will cause memory accesses unless Blank is also set (in which case there is no picture and there are no memory accesses). The contents of the control fifo is used to specify which locations are accessed and displayed. If both Pic and Bk are cleared, the border pattern will be displayed for all bits within a line and no memory accesses will occur. This is done to create the top and bottom picture borders.
>
> Invert (Inv) - Setting this bit causes the video signal to the monitor to be inverted. All areas of the screen (border and picture) shown while this bit is set will be inverted.

Odd (OD) - Setting this bit indicates to the controller that the odd field of a frame is being scanned. This is used by the controller to determine whether vertical sync pulse should start and stop at the beginning or middle of a line. It starts at the middle for an odd line. This bit should not be changed during a vertical sync pulse, since changing it during the sync pulse would cause the end of the sync pulse to occur at a different location in a line from where it started. Neglecting this could cause interlace problems on monitors triggered on the trailing edge of vertical sync. (Most of our monitors are triggered on the leading edge of vertical sync.)

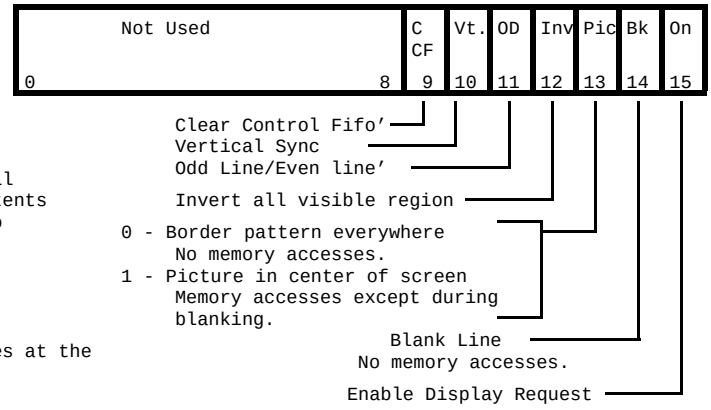
Vertical (Vt.) - Vertical sync pulse line goes low when this bit is set. The exact time of the transition relative to a horizontal line time is determined by the odd bit.

Clear Control Fifo' (CCF) - When set to zero, this bit causes the contents of the control fifo to be declared invalid. The bits are not actually set to zero but the fifo is declared to be empty. Normally this bit is kept set to one. The Control Fifo should be cleared during each vertical retrace as a safety measure.

Dandelion Display Controller Registers

Control Register (on X Bus)

DCtl may be written in any cycle.



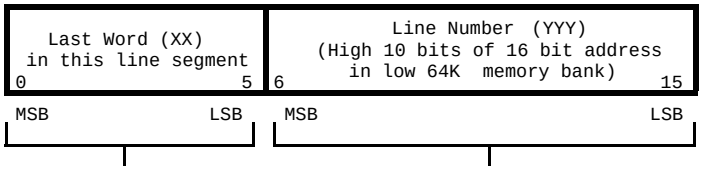
Register controls display operation.

It is cleared to 0 by IOPReset when system is powered on. When cleared, the display will receive only horizontal sync, video will be the contents of border register at power-up, and there will be no display requests to the processor. There will be no vertical synchronization or blanking.

Vertical sync is a strobed version of the vertical bit in the control register. To produce interlaced scan, vertical sync is strobed and changes at the beginning of the line for even lines and middle of the line for odd lines, as specified by bit 11.

Control Fifo Register (on Y Bus)

DCtlFifo may be written in any cycle.



One word is written into Fifo for each cycle in which DCtlFifo is asserted. Used to specify a location (line segment) in memory from which to retrieve data for display. The low 10 bits specify the line number, and the high 6 bits specify the Last Word to be read before changing line segments. If this Last Word is not 63 (end of line) then the next Fifo entry is used to identify the next line number from which a group of words will be taken. Note that the low 6 bits (word #) used for the address is incremented from 0 to 63 in each line. The control fifo only permits selecting the line number and the location along the line at which a transition is made from one line to another. Thus, as viewed on the monitor screen, this mechanism facilitates vertical movement of images, but not horizontal movement, since low 6 bits of address come from word counter. One control word is loaded into the fifo for each continuous segment of words in a horizontal line. Thus, a normal line with no cursor or window will have one control word. A line with a cursor in the middle will have 3 control words. While the fifo size is 16 words, no more than 10 entries should be for a single line. The last control word for a line should specify word 63 (decimal). The controller will "wrap around" to the next scan line in a field if necessary to advance the word number specified in a control fifo entry. Control words can be loaded into the Fifo any time before the line in which they are used. Care must be taken not to insert extra control words, and lose sync. Clearing control fifo during vertical sync will purge the fifo. The control Fifo is cleared by setting bit 9 of control register to 0.

Selects location (word #) along horizontal line after which display will jump to a new line in memory from which it reads data.

Horizontal line starts reading data at

YYY	00

in main memory,

and continues reading until location

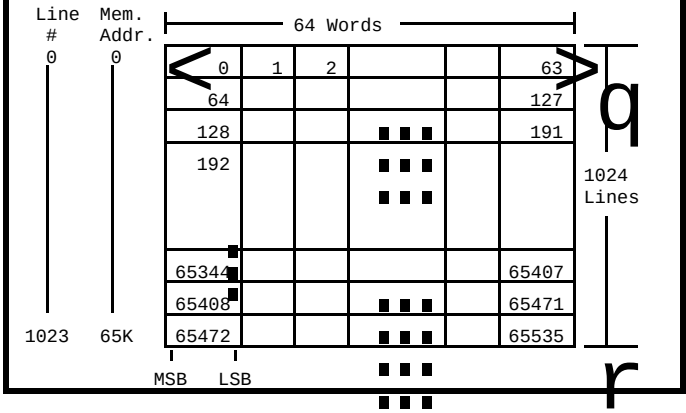
YYY	XX

AFTER which it advances to the next control Fifo location specifying the next line segment:

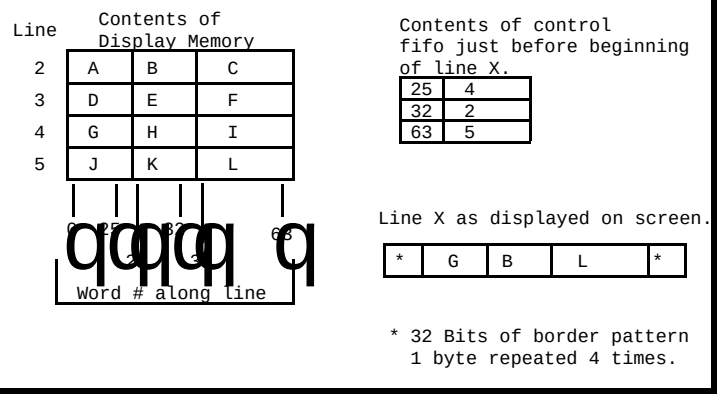
YYY'	XX+1

on the same output line.

Display Memory (Low 64K bank of physical address space)

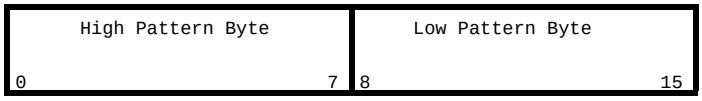


Example of Control Fifo Use



Border Pattern Register (on Y Bus)

DBorder may be written in any cycle.



Border pattern is repeated on every line during the first and last 32 bits scanned. The high and low patterns are used on alternate pairs of lines. Lines are numbered starting from 0 at the top of the screen. The border pattern will be repeated all across a horizontal line if bits 13 (Pic) and 14 (Blank) of the control register are both 0.

High pattern is repeated on lines 4n+2, 4n+3 where n is an integer.

Low pattern is repeated on lines 4n, 4n+1 where n is an integer.

Control Fifo Register

This register contains two fields; *last word* and *line number* which are used to specify a line segment.

> *Last word* is used to specify the number of the last word position (relative to lines aligned on 64 word boundaries in memory) to be used for a given line segment. *Last word* is the high 6 bits of the control fifo entry, and typically remains constant for a given window. The *last word* field of the control fifo entry for the last segment in a horizontal line must be 63.
>
> *Line number* is used to calculate the memory addresses in which bits for the displayed line segment are found. The controller hardware maintains a 6 bit counter for addressing words within a horizontal line. This counter always counts from 0 througn 63 as the line is displayed. The low 6 bits of the current memory address are the controller's 6 bit count. The high 10 memory address bits come from the *line number*. When the controller's count matches the *last word* from the current control fifo entry, the next fifo word containing the next *line number* is fetched. Using this mechanism, the user can define the mapping between memory address and screen position. Note the low 6 bits of memory address are not involved the mapping; they always specify the word's horizontal position on the screen. The display hardware supports only vertical displacements. For example, word 0000 in memory may be shown on any line of the display but must always be the first word in the line. This line number is typically incremented by 2 (because of interlacing) for successive lines within a window.

Border Pattern Register

This register contains the two border pattern bytes. Only one of the bytes is used in any given scan line. The low border byte is used during lines 4n, and 4n+1 (lines 0,1,4,5,8,9...) and the high border byte is used during lines 4n+2 and 4n+3 (lines 2,3,6,7,10,11....). The proper byte is repeated 4 times at the beginning and end of each horizontal line to form the side borders. If the Picture and Blank control bits are both off, the byte is also used to fill the picture area. The top and bottom borders are created in this fashion. The border pattern register need only be loaded once.

4.6 Using the Controller

In the Dandelion architecture, the processor is shared among a number of microcode *tasks*. One of these is a high level language emulator; the others control I/O devices. The processor is used in round-robin fashion by the tasks. Each I/O task is assigned one or more *clicks* in the processor *round*. There are five clicks per round. A task may perform one main memory access in parallel with three microinstructions in a click. The display is assigned click number 4 of each round. Clicks not used by their assigned I/O tasks are available to the emulator.

Each round takes 2.055 uS to execute. There are exactly 14 rounds per horizontal scan line (the processor clocks are derived from the display clock so there is no skew). Thus the display microcoder must ensure that any action scheduled to take place in one scan line can be done in 14 clicks.

This section outlines the actions the microcode must take to get an image in the low 64K of memory shown on the display. The following figure shows what is loaded into each register during the various parts of a frame. Note that the only differences between the "odd" and "even" fields of a frame are the setting of the odd field bit in the control register, the line offset used when loading the control fifo and the length of the vertical sync pulse.

Note also that the parameters for line n must be loaded during line n-1. For example, parameters for the first picture line are loaded during the last border line at the top of the screen. Assuming the microcode used to set the control and control fifo registers runs once per scan line; the proper order is:

Second to last Top Border line: Send a 41'X to DCtl (display line of border).

Last Top Border line: Send a 41'X to DCtl, load DCtlFifo with parameters for first line of screen.

First Picture Line: Send 45'X to DCtl (display picture) and load DCtlFifo with parameters for second picture line.

Register Loading Sequence to Get Bit Map on Display

# Lines	Function	Control Register Loaded only once per function during first line	Control Fifo Loaded once per line in even and odd fields.
1	End Vert Sync, Blk	3	—
13	Top Border	41_{16}	—
404	Even Field	45_{16}	Last Word = 63 Line Number = even #'s
13	Bottom Border	41_{16}	—
18	Start Odd V Sync	73_{16}	—
1	End Vert Sync, Blk	3	—
13	Top Border	41_{16}	—
404	Odd Field	45_{16}	Last Word = 63 Line Number = odd #'s
13	Bottom Border	41_{16}	—
17	Start Even V Sync	63_{16}	—

897 lines total 448.5 lines/field

To add a cursor, the control fifo is loaded with 2 or 3 segments per line for a run of 8 lines in each field. This is shown in the next figure. Showing a window in addition to the cursor requires more segments per line. For both the cursor and the window, some computation must be made once per frame to determine the control fifo entries. In addition, the cursor bitmap must be updated each time the cursor moves.

Register Loading Sequence to Get Bit Map with Cursor

# Lines	Function	Control Register Loaded only once per function during first line	Control Fifo Loaded once per line in even and odd fields.
1	End Vert Sync, Blk	3	———
13	Top Border	41_{16}	———
x	Even Field	45_{16}	Last Word = 63 Line Number = even #'s
8	Even Cursor	45_{16}	3 Seg. for cursor
396-x	Even Field	45_{16}	Last Word = 63 Line Number = even #'s
13	Bottom Border	41_{16}	———
18	Start Odd V Sync	73_{16}	———
1	End Vert Sync, Blk	3	———
13	Top Border	41_{16}	———
y	Odd Field	45_{16}	Last Word = 63 Line Number = odd #'s
8	Odd Cursor	45_{16}	3 Seg. for cursor
396-y	Odd Field	45_{16}	Last Word = 63 Line Number = odd #'s
13	Bottom Border	41_{16}	———
17	Start Even V Sync	63_{16}	———

897 lines total 448.5 lines/field

During Cursor, 2 or 3 control fifo entries per line are used.

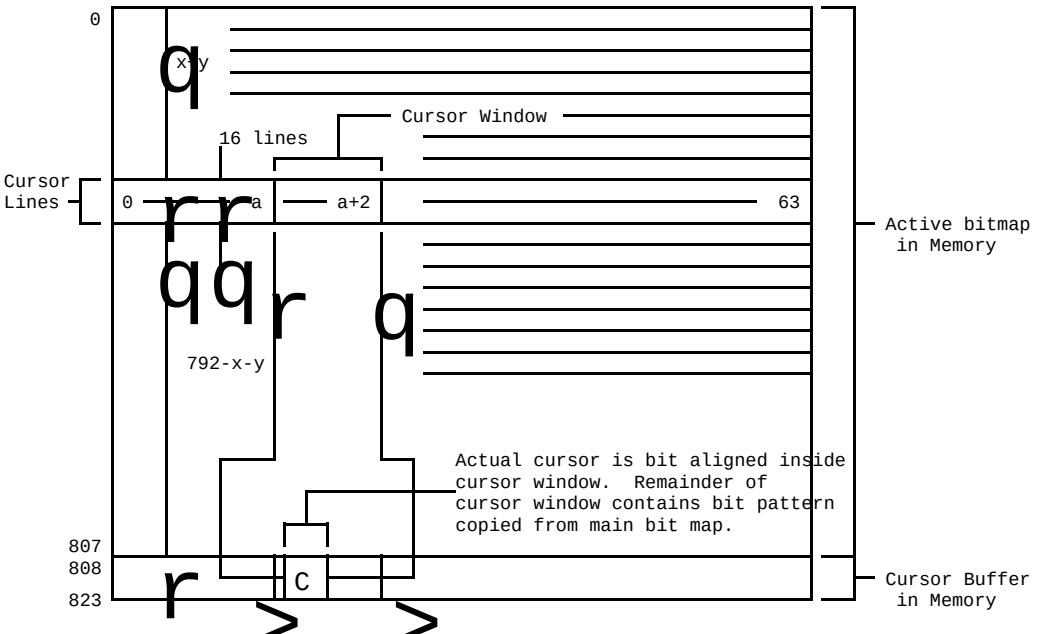
Last Word	Line Number
a	x+y + 2n
b	808 + 2n
63	x+y + 2n

a	x+y+1 + 2n
b	809 + 2n
63	x+y+1 + 2n

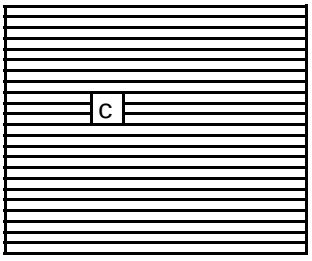
n is line increment within cursor ranging from 0 to 7 for successive lines in a field.

The size of the bitmap required may be reduced if more border pattern is shown or lines from memory are shown more than once on the screen.

Memory



Display Screen



Actual cursor is bit aligned inside cursor window. Remainder of cursor window contains bit pattern copied from main bit map.

During Cursor lines, each line is divided into 3 segments. The middle segment comes from the cursor bitmap.

Segment 1 Word 0 to a
Segment 2 Word a+1 to a+2
Segment 3 Word a+3 to 63

Horizontal cursor position is set by horizontal position of cursor image in Cursor Buffer.
Vertical cursor position is set by line number at which the image is displayed.

4.7 Display Hardware Implementation

Display Controller (Horizontal line generator)

The display hardware handles only those functions which repeat on a horizontal line basis. If the processor had provided these functions, a great deal of its bandwidth would have been required for a fairly simple, repetitive task. Similar hardware for counting lines and controlling windows is not used because the processor bandwidth required for these tasks is available.

Horizontal Events

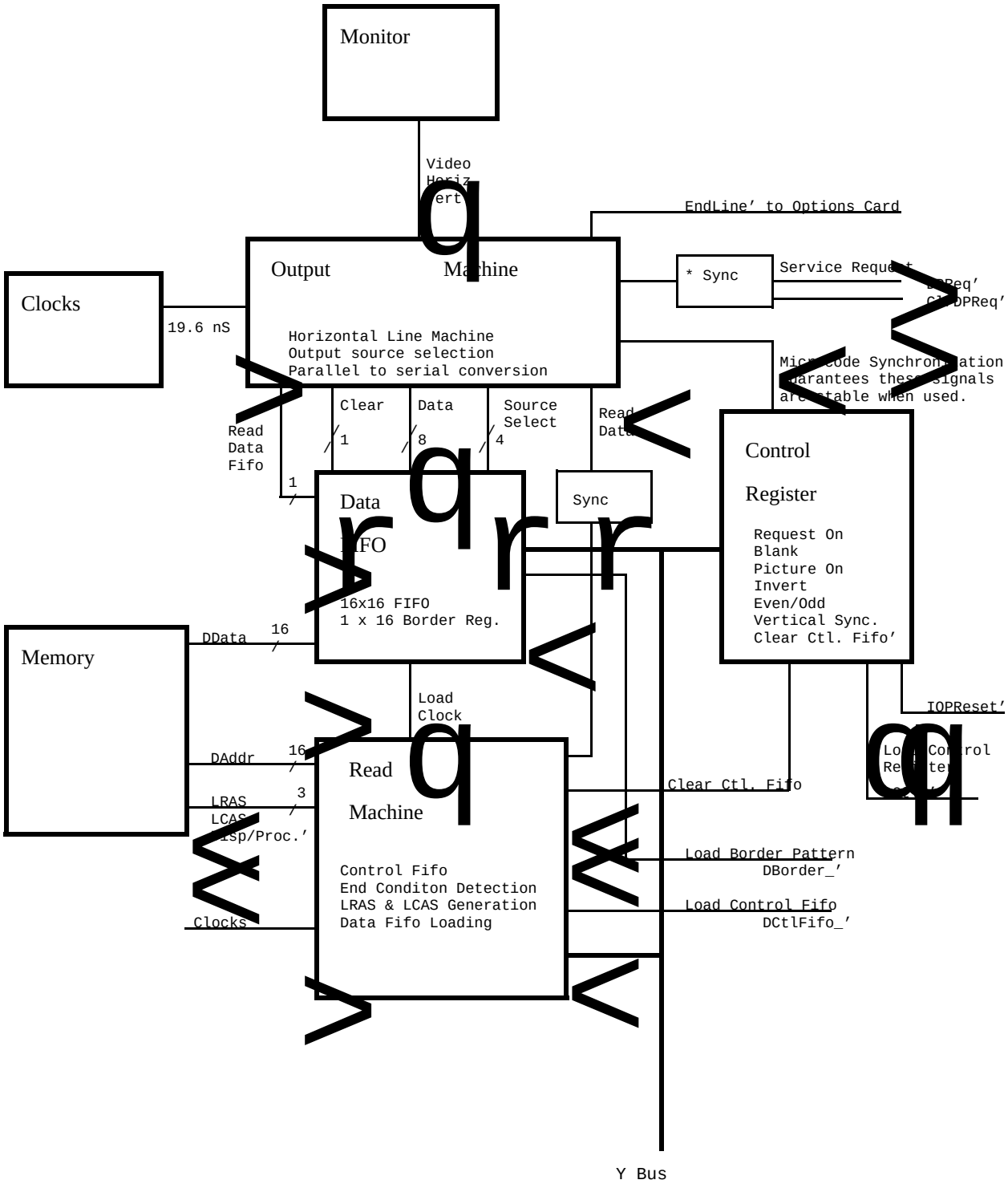
A horizontal line contains 32 bits of border pattern on the left, 1024 bits of picture, 32 bits of right border, and 382 bits of blanking. A horizontal sync pulse starts 8 bits after blanking starts and ends 8 bits before blanking ends.

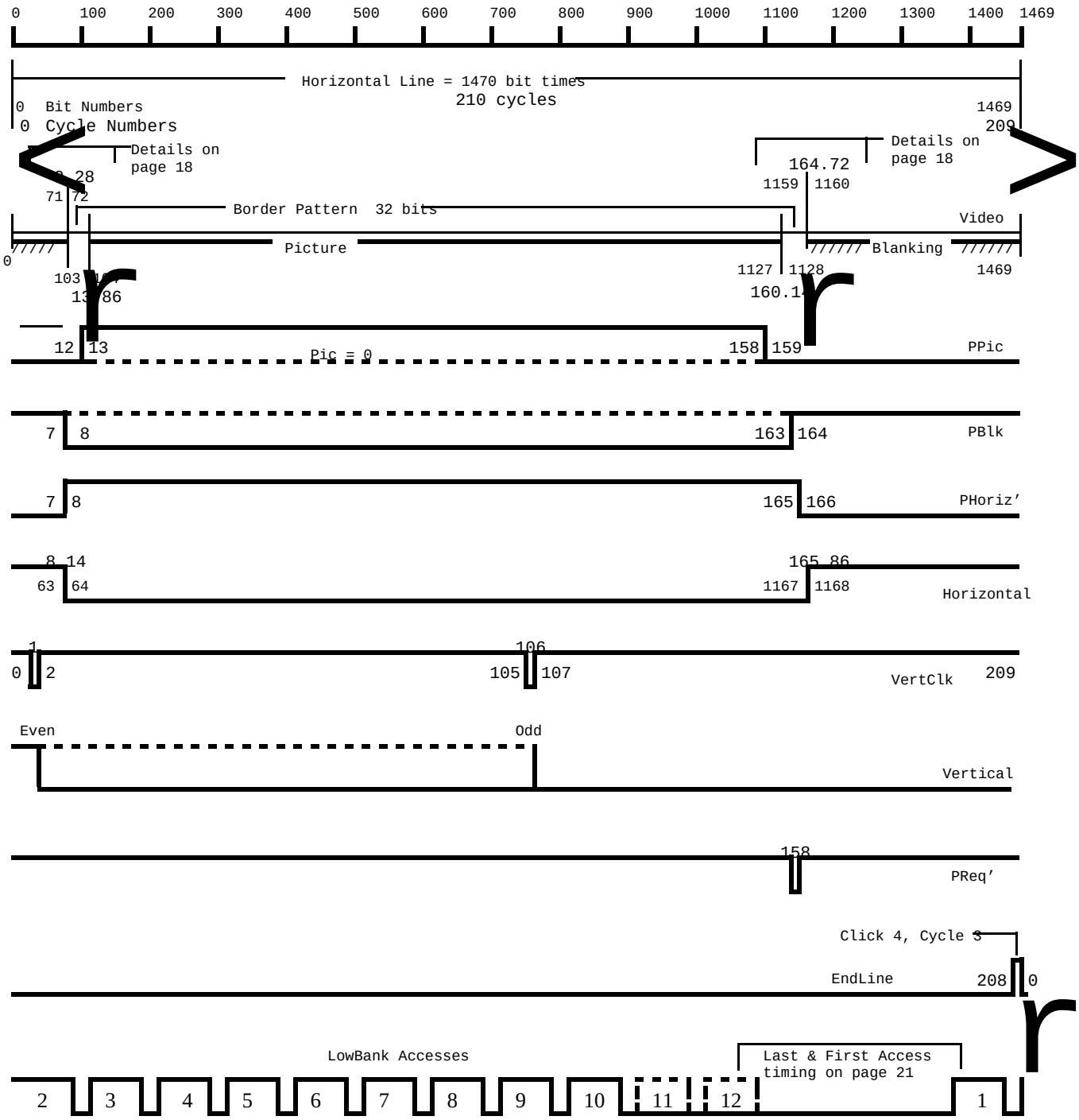
Vertical Events

A frame consists of an even and an odd field, each of which contains 13 lines of top border pattern, 404 lines of picture, 13 lines of bottom border, and 18.5 (18 lines in one field and 19 lines in other) lines of blanking during which vertical retrace takes place. The section covering controller use further describes vertical events. No further mention of vertical events will appear in this section.

The next two figures show a functional block diagram of the display controller and output machine timing diagrams. It has three principal parts; the output machine, a data fifo, and the read machine. Associated with the output machine is the control register. The border pattern register is associated with the data fifo. The read machine contains the control fifo and associated control fifo register, end condition logic to terminate memory accesses at the end of a round and at the end of a line, and the LRAS and LCAS memory clock generation. Following are descriptions of these sections.

Display System Functional Block Diagram

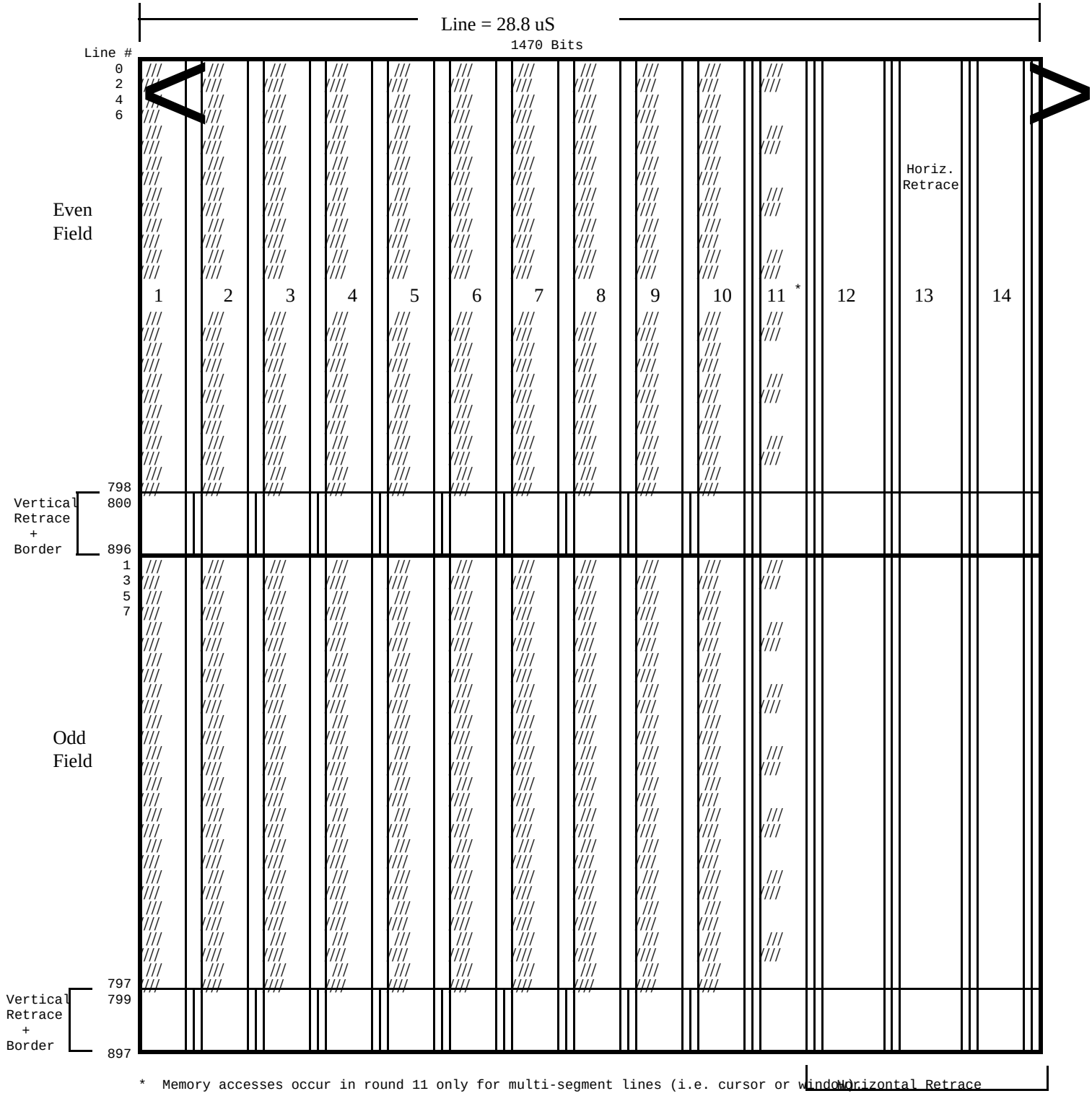




There are low bank accesses during rounds 1-10 for a line with 1 segment. Some amount of round 11 is used for lines with 2-10 segments. While not recommended, round 12 is available for lines with more than 10 segments. Care must be taken here since the data fifo can go empty in this round if more than 10 segments are specified. Words must be strobed into DataFifo at least 330 ns before it is read out. At beginning of line, DataFifo is filled to 13 words at the end of round 2 .

3 cycles/ click	210 cycles/ line	Bit Time = 19.59 nS	Cycle Time = 137.14 nS
5 clicks/ round	7 bits/ cycle	Bit clock = 51.04 MHz	Round Time = 2.0572 uS
14 rounds/line	1470 Bit times/line		Line Time = 28.8 uS

Figure x. Timing For Output Machine



1 Frame = Even Field + Odd Field

Field = 399 Active Lines + 32 Border Pattern Lines + 17.5 Blank LInes (retrace)

Low 64K bank
Memory used by display

Processor use during display
click 5 by display & memory refresh.

Figure x. Resource Use During Display Frame

Output Machine and Control Register

The output machine will be described in terms of the actions that take place during the output of a horizontal line. Each horizontal line starts with 32 bits of border pattern, followed by 1024 bits of data from memory, followed by 32 bits of border pattern, and ending with a blanking period during which the horizontal retrace takes place. This sequence repeats every horizontal line and is shown in the output machine timing diagram.

The output machine is controlled by a prom state machine with 210 states (1 state per machine cycle). It is cycled through these states by the display prom counter, which is a part of the system clock. The timing diagram shows the outputs of the prom register marked with asterisks. There are two time references in this figure. There are 1470 bits per line and they are marked on the top of the figure. There are 210 cycles of 7 bit times each, which are labeled in italics.

Starting at bit position 0 (look at line labeled video), the F16 counter in the output machine has just been resynchronized to 0 by the EndLine and Tick7' pulse. The output shift register and blanking register are strobed at the beginning of bit 8 and every 8th bit thereafter during a horizontal line. Thus, shift register loading, blanking, and unblanking are done on byte intervals.

The first 32 bits of a line come from the border pattern register. The selection of the high or low byte is done by a flip-flop which is toggled every horizontal line in a field. This produces the signal BPBS (border pattern byte select). This signal is always reset by a vertical sync pulse so the first line of a field comes from the low byte of the border pattern register. (More specifically, it is the first line after the trailing edge of the vertical sync pulse. Note that since the first few lines of a field are often blanked, it may not correspond to the first visible line.)

On the first cycle boundary after the 4th border byte is loaded, PPic goes to a logic 1, such that the next byte loaded comes from the high byte of the data fifo. Byte selection is performed by the high bit of the bit counter. The fifo is clocked after the high byte is loaded. This process continues until all 64 words have been loaded into the shift register and shifted out. While the low byte of the 64th word is being shifted out, PPic goes low so that the next byte to go out comes from the border register. While the 4th border byte is being shifted out, PBlk comes on so that blanking starts on the next byte boundary. Blanking continues until the end of bit 71 (after the counter wraps around), after which the next horizontal line starts with the border pattern again.

The horizontal sync pulse starts 8 bit times after blanking starts and ends 8 bit times before blanking ends. Both horizontal and vertical sync signals pass through a low pass filter which increase the rise and fall times to approximately 100 nS. This helps reduce high-frequency radiation from the cable going to the monitor.

Data Fifo

A 16 word data fifo provides buffering and solves the problem of synchronization between the memory system and the output machine. (While both memory and output machine run from the same clock, the largest common period is the 19.6 nS clock period which is too fine to be of any value.) Data is strobed into the holding register and fifo with DCAS' and DCASDly', respectively, both of which come from the read machine. Words are read out of the fifo with the signal ReadDataFifo, which comes from the output machine. The outputs of both the data fifo and border register are multiplexed onto a byte wide tri-state bus, then through TTL-ECL converter to the parallel input of the output shift register in the output machine. Selection of the appropriate output byte is done by the output machine. The output machine controls the read machine such that the fifo never overflows or underflows during a line.

Read Machine

The read machine does memory accesses during the first 4 clicks of a round. It always starts at the

beginning of a round and continues to either the end of the round or the last word of a line has been accessed, whichever occurs first. Reads in a round are initiated by a signal, PD/P, from the output machine. The read machine will determine the mix of full and page mode accesses necessary and do the maximum number of memory accesses possible within a round. The low 6 bits of the memory address always count from 0 to 63. The high 10 bits (line number) are specified in the control fifo entry. The last word to be used from a given line is also specified in the control fifo entry (6 bits) and is used to advance to the next fifo entry when that word number is reached. The three parts of the read machine (control fifo, end condition logic, and LRAS, LCAS generation) are described in the following paragraphs.

Control Fifo

The control fifo contains 16 entries. Each entry identifies a line segment using 10 bits to specify the line number and 6 bits to specify the last word in the segment. The control fifo is loaded from the Y-Bus, unloaded by a signal from the end condition logic, and cleared by a bit from the control register. The microcode must take care to load only the entries for one scan line per horizontal line wakeup on the average. The control fifo should be cleared once per vertical field to eliminate the effects of noise and assure its state at the beginning of a field.

Word Counter & End Condition Logic

The word counter counts from 0 to 63, synchronous with the memory accesses used to fill the data fifo. The output of this counter is compared with the 6 bit last word field of the current control fifo entry. When they are equal, the control fifo is advanced to the next entry. There is also logic to determine when a full (RAS and CAS) memory reference should take place. A full reference must take place whenever one of the RAS bits at the memory chips changes. This can occur on the first reference in a round, when the control fifo is advanced, and on every 8th memory reference due to the arrangement of bits in the memory system.

The number of accesses in a round depends on the number of full (293 nS) and page mode (215 nS) accesses that occur. A maximum of 5 full accesses, 4 full and 2 page accesses, or 1 full and 6 page accesses can occur. Thus the total number of accesses can range from 5 to 7. A prom state machine looks at the combination of accesses and drops the signal EndRndRead' during the last access of a round. The accesses in a round can end early if word 63 is reached. The signal InhibitRead also becomes true after word 63, locking out any further reads, independent of PD/P signal from output machine, until is reset by the signal ClrDataFifo' from the output machine. Details of the state machine and other logic timing are in the Clock and Display drawing package.

LRAS-LCAS Generation

The signals LRAS and LCAS are the clocks for the low bank of the memory system. These signals are identical to RAS and CAS for processor memory references (411 nS cycles), but have a faster full cycle time (293 nS) and a page mode cycle (215 nS) when the display is using the low bank (indicated by Disp/Proc' in the high state). In all cases, CAS follows RAS by 49 nS. Both of these generators are simple state machines using one counter and discrete logic for decoding. They have a 19.6 nS cycle time.

4.8 Clock Generation

The CP cycle clock (137.14 nS) is derived by dividing the display's bit clock by seven. The next figure shows the relationships between the clocks generated on the HSIO card.

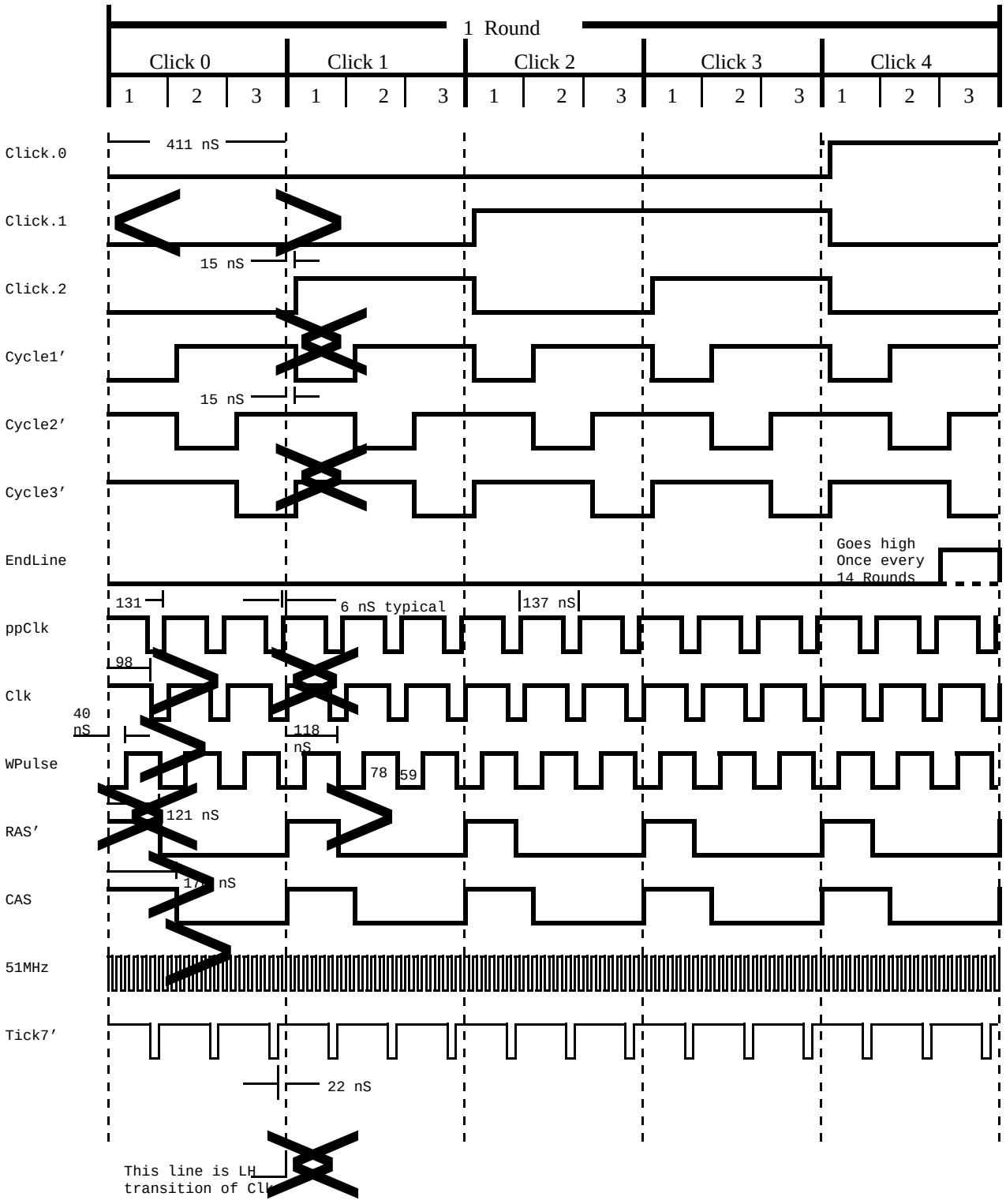


Figure x. System Clocks (Backplane Timing)

5.0 Disk Controllers

Two types of rigid disks can be controlled by the Dandelion. Section 5.1 discusses the Shugart disk controller located on the HSIO card. Section 5.2 describes the Trident disk controller, which is found on the HSIO-L card.

5.1 Shugart Disk Controller

5.1.1 Overview

This chapter is concerned with the Dandelion's controller for the Shugart SA4000 and SA1000 type disks. It identifies the major components of the system and their connections. It is assumed that the reader will have read the *SA4000 Fixed Disk Drive OEM Manual* and *SA1000 Fixed Disk Drive OEM Manual* from Shugart. This chapter is concerned with the function of the disk controller, not of the disk drive.

There are four major blocks in the Dandelion Disk Controller . They are the Input Conditioning, Output Conditioning, Processor Interface and Serializer/DeSerializer circuits. Disk read data, disk clocks and reference clocks arrive via the Input Conditioning circuits, as do disk status lines. The disk control lines, disk write data and write clocks are sent via the Output Conditioning circuits. The Processor Interface generates microcode service requests, detects the overrun condition and passes data, status and commands along the X-Bus. Disk data is converted from 16 bit parallel words to a serial data stream and back in the Serializer/DeSerializer.

5.1.2 Constraints

Cost

The Dandelion is intended to be a relatively low cost workstation. To this end, the hardware it contains should be minimized. This leads to low manufacturing, testing and service costs. The guiding principle of the controller's design has been that only functions which occur too quickly for microcode to handle or require hardware buffering are implemented in the controller. For example, step pulses may be sent relatively slowly, so the step line is toggled by having the microcode send control words in which the step line is alternately set and reset.

Another result of the cost constraint is that one controller board should serve to control both the SA1000 and the SA4000 drives. It is able to support drives with 2 to 32 heads. The effort required to change the board from an SA1000 configuration to an SA4000 configuration is small. In fact, it is limited to unplugging a set of SA1000 cables and plugging in a set of SA4000 cables.

Disk Format

The disk is divided into cylinders. Each cylinder represents a distinct position of the read/write heads. Each cylinder is divided into tracks, one per read/write head. The SA1004 drive has 4 heads, the SA4008 has 8 and the SA4104 has 16. Each track is divided into sectors. There are 28 sectors per track on the SA4x00 type drives, 16 sectors per track on SA1000 type drives. Each sector is divided into three fields, Header, Label and Data. The Header field is used to specify the sector's physical position on the disk (cylinder, head and sector numbers), the Label specifies the page's position in the file system and the Data field holds the actual data. Each field is broken into 4 areas. A pattern of all zeros is followed by a synchronization word or address mark, the field's data and a word of CRC checksum. The length of the synchronization pattern is 7 words on both

drive types. A synchronization word of all ones is used to define the first word boundry on the SA4000 drive. An address mark serves a similar purpose on the SA1000 drive. The Header field contains 2 words of data, the Label field 12 words and the Data field 256 words. The CRC checksum word following the data area of each field is used to implement an error detecting code.

The controller hardware does not preclude other disk formats. It is designed to read, write or verify an individual field of a sector. The length of each field, the number of fields per sector and the number of sectors per track is set by the microcode. There is a restriction on the number of sectors on SA4000 type disks. The SeekComplete signal on those disks is sent before the heads have really settled so the controller adds a delay of 29 sector pulses before passing it on. Thus SA4000 type disks should have no more than 28 sectors per track (the 29 sectors pulses is intended to delay at least 20 mS) or should be prepared to add some sort of extra delay in microcode.

One of the constraints on the design is that it must be possible to read, write or verify each field in every sector of a cylinder at the rate of one revolution per track. This means that in addition to the raw data rate constraint, the inter-field, inter-sector and inter-track setup required by the hardware must be minimized. A design which requires a great deal of setup between sectors or fields may not be acceptable. It should be possible to perform almost any combination of operations on the fields of a sector. An exception to this rule is that when a write is performed to one field, further fields of that sector must either also be written or are assumed to be lost. The microcode must also be capable of aborting operations on later fields based on the results of operations on earlier ones. For example, if the Header and Label fields of a sector are to be verified before the Data field is written, the Data write should be aborted if either the Header or Label verify operations fail.

The SA1000 drive does not contain a data separator, the SA4000 drive does contain one. The controller board sends and recieves MFM (Modified Frequency Modulation) encoded data to and from the SA1000 drive and NRZ (Non Return to Zero) data to and from the SA4000. The SA1000 data rate is 4.27 MBits/Sec (234 ns/bit). The SA4000 data rate is faster at 7.14 MBits/Sec (140 ns/bit). The SA1000 data rate is governed by a clock in the Dandelion, the SA4000 data rate is set by drive itself.

Function Allocation

The most complex operation on a field is verify. It requires that each bit be checked against a template from memory, a CRC checksum be maintained, a memory address updated and a word count decremented. Four pieces of information must be maintained, an address, a word count, the data to be verified and some sort of checksum. While it would be possible to combine the address and word count by requiring all field templates to begin (or end) on page or nibble boundries, this is not generally acceptable. The designer has been unable to find an encoding scheme which makes it possible to combine the data to be compared and the checksum. These seem to be the only remotely workable combinations. Hence all four quantities must be kept independently.

The four quantities must be divided between the two R registers in the processor and registers in the controller. The lack of U register speed precludes their use. One must spend an entire click to update one U register (read it, change it, then store it), yet the microcode is only allowed one click per word transferred. Due to the main memory addressing scheme, the address must reside in one of the R registers. This leaves the other R register available for either the checksum, data to be verified or the word count.

Were the R register to be used for the checksum, the hardware would contain the word count and the data to be verified. This scheme would have the advantage of substituting a simple counter for a more complex CRC chip. However, the microcode would have to both read the disk data to maintain the checksum and send memory data to the controller to be verified. This scheme has latency difficulties. The disk controller and processor use different, unsynchronized clocks. After sending a Service Request, the controller expects an interval of random, but bounded, length will pass before microcode reads or writes the proper buffer. The Service Request is sent so that the

controller will have the buffer ready before the minimum service time and will not require it again before the maximum service time. As seen from the processor side, there is a window during which each Service Request must be served. If the service takes place too soon, the buffer may not be ready; if it is too late, the controller may have used the buffer again. In the case of the SA4x00 type disks, the service window is barely one cycle wide. The Service Request is sent so this is cycle 2 during Read operations and cycle 3 during Write and Verify operations. Sending and receiving data in one click would require 2 cycles, hence a 2 cycle service window. This is reason the microcode cannot maintain the checksum while the controller does data verification.

It would be possible to compute the checksum and maintain the word count in the controller while doing the address and verification in microcode. Unfortunately, the microcode would be messy and the status of an operation would be partially in microcode, partially in hardware. The controller as designed allocates the address and the word count to microcode and the data and checksum to hardware.

5.1.3 Microcode - Hardware Interface

The controller has been designed with the idea of minimizing the amount of hardware used. As much functionality as possible has been left in the microcode and software. This results in fairly simple controller hardware.

Many of the lines used to control the disk are set directly by microcode and are ignored by the controller. For example, the Step and Direction lines controlling the position of the disk's read/write heads are merely bits in the control register that are relayed directly to the drive. The same is true for many of the status signals returned by the drive, they are read and interpreted only by the microcode or software.

The controller contains one word of buffering for write and verify operations and one word for read operations. As explained above, the Dandelion architecture allows the designer to calculate the minimum and maximum latencies between a service request and the processor's response to ensure an overrun never occurs in normal operation. If the disk microcode stops servicing the hardware, the overrun flag is set and write operations are disabled to restrict the amount of random data written on the disk.

This section will begin with an overview of the status, control and data registers then proceed with a detailed description of each.

Control Register

This 16 bit register receives its inputs from the X-Bus, sending them to both the disk drive and to the controller. It is reset by IOPReset'. The control bits are arranged so that when reset, the controller and disk are dormant. It is expected that IOPReset' will be held active while power to the machine is being turned on or off.

Status and Test Registers

Three types of 16 bit quantities may be read from the controller. One is data from the disk, the second is the status of the current disk operation, the third is a group of test points on the disk and display controllers. The first will be discussed below under Read Data Register. The second two are independently sent to the X bus. The operation status is composed of some lines from the drive itself (Track00, DriveNotReady, etc) and some from the controller (Verify Error, Overrun, etc). These are the normal lines read using the _KStatus command to guide the execution of a disk operation. The test lines are read using the _KTest command by diagnostic microcode or software to directly test the control and status lines leading to the disk.

Some of the Status signals should only be sampled on word boundaries. The CRC error flag, for instance, is only valid after the last bit of the CRC checksum has been seen. Sampling on word boundries also gives the microcode an entire word time, as opposed to one bit time, to freeze the final status flags of a data transfer. This sampling is done by the Word Status Register.

Write Data Register

Data is sent from the processor to the controller in 16 bit words. The words are buffered in the Write Data register before being loaded into a shift register. The buffer is automatically cleared before a transfer begins. It is loaded by the microcode in response to each service request during a transfer. By calculating the minimum and maximum latencies between request and service, one may be assured that the buffer is always loaded after the previous word has been used but before the current word is needed.

Read Data Register

Like the Write Data register, this is a single word of 16 bits. It is loaded from the controller's shift register each time a word boundry passes. Just before it is loaded, a service request is sent, asking the disk microcode to remove the word. As with the Write Data buffer, one may assure oneself that this will always happen after the buffer is loaded but before it is loaded again.

A wrap-around feature has been included in this controller allowing diagnostic microcode to verify that data may be written and read correctly. The method for using the feature depends on the disk being controlled. The SA4000 provides one clock used throughout the controller. The data sent out is intercepted just before the final drivers and inserted into the input data stream. It is then shifted back into the shift register. By having the microcode start a write operation, then perform reads instead of writes, one may verify that the data being written is correctly re-received. Note that the re-received data will be a rotated version of the data sent.

The SA1000 drive supplies no clock. The clock used to write the data is derived from the stable processor clock. If this clock were used for the entire controller, the controller's data separator would not be tested. The data separator is tested by allowing it to re-produce the NRZ data using a clock derived from the re-received MFM data stream. Because of jitter between the derived clock and the reference clock, we may not reliably route the re-produced NRZ data back to the shift register. Hence one may not expect to see the data sent in the ReadData register. The address mark recognizer section of the data separator does record the polarity of bit 14 of the address mark however. It appears on the Header tag bit in the KStatus register. One may test the controller by sending address marks and sampling the Header tag status bit after each one. Each address mark must be sent in its own field, that is, the TransferEnable bit should be reset between each one. The Header tag status bit should match bit 14 of the address mark just written.

Service Request / Overrun Machine

As seen above, the controller must be able to generate service request to its microcode and determine whether the requests have been answered. This is the task of the Service Request/Overrun machine. The timing of Service Requests is based on the BitCount within a word, the time within a field, the operation being performed and the data rate of the disk. Only two disk types are supported and the data rates of both are fixed.

During data transfer operations, it is crucial that the disk microcode keep pace with the hardware. If the microcode is early or late, especially during write operations, disk data may be destroyed. The Overrun section of this machine will set the Overrun signal whenever a buffer is needed by the controller before it has been serviced by the microcode. Thereafter, no data may be written (the disk's WriteEnable line is turned off) and the Service Request signal is set until the microcode finishes the operation and turns it off. The microcode should sample the status at the end of every operation, testing the Overrun signal. An unexpected consequence of turning off WriteEnable very

early in the writing of a field is that the drive will often get a WriteFault error. If WriteFault and Overrun occur together during debugging, it is best to investigate the Overrun first.

Service requests may be used not only to synchronize the transmission of data but also to sense status conditions. For example, it would be wasteful to burn 1/5 of the processor waiting 20 ms for an IndexFound signal. The same holds true for a SeekComplete. These and other signals may be used to generate service requests directly. The microcode may then yield its click to the emulator while waiting. The signals are chosen using the Operation field of the Control register.

6.1.4 Detailed Register Description

KCtl Register

Head Select					Drive	Fault	Reduce	Step	Direct	Firm-	Trans-	Write	Wakeup Control		Write
16	8	4	2	1	Select	Clear	IW		In	ware Enable	fer Enable	CRC	0	1	Enable

KStatus Register

Head Select′					Seek	Track	Firm-	Index	Sector	SA1000	Drive	Write	Over-	CRC	Verify
16	8	4	2	1	Com- plete	00	ware Enable	Found	Found/ Header Tag	/ SA4000′	Not Ready	Fault	run	Error	Error

KTest Register

Disk Read Clk	Disk Read Data	Disk Output Clk	Disk Write Data	Seek Com- plete′	Direct- tion In′	BHoriz	Reduce IW′	TTL- Video	Sector′	Drive Select′	BVert′	TTL- Video′	Step′	Read Gate′	Write Gate′

Control Register

The register is loaded by the processor when a "KCtl _ xx" type instruction is executed in microcode. This may also be done as part of a Mesa "Output" instruction. The command word is divided into two parts intended for the drive and the controller. The meaning of the bits in the Drive Control field are explained fully in the appropriate Shugart manuals. They are listed below with a brief description. A list of Operation bit meanings is given below. Use of all bits in the control word will be given in the section on microcode usage. Control lines required by the drive but not listed below are the responsibility of the controller, not the microcode.

Drive Control

HeadSelect1 - HeadSelect16: These 5 bits are used to select one of the read/write heads. They are not latched by the drive; all commands must contain them. For example, when one writes a field by sending a write command and a write CRC in succession, the proper head select bits must be present in both commands. To ensure the drive's setup times are met, a command word containing the proper HeadSelect lines should be sent at least 20 uS before one containing the HeadSelect lines and the operation to be performed.

DriveSelect: The DriveSelect bit has been included even though only one drive may be connected at a time. This is because releasing DriveSelect has useful side effects. The SA1000 type drives lack a FaultClear input, Write Faults are cleared by de-activating the DriveSelect signal. The SA4000 drive has a feature enabling it to cut the power to its stepper motors when not selected. This can result in a substantial power savings. The power may be cut by software when the drive has been idle for some nominal interval. When re-selected, one must wait 20 ms before using the drive. This time interval may be sensed using the SeekComplete signal which is automatically cleared when the drive is de-selected.

FaultClear: The FaultClear bit is only active when an SA4000 drive is connected to the controller. Write Faults on the SA1000 are cleared by turning off DriveSelect as explained above. An SA4000 WriteFault is cleared by activating both DriveSelect and FaultClear then de-activating FaultClear. If the WriteFault remains, the drive is probably broken.

ReduceIW: This bit is only significant when writing on the SA1000 type drives. These drives require the write current to be reduced on cylinders 128 through 255. This bit should be set by the microcode when writing on these cylinders. During read and verify operations on SA1000 disks and during all operations on SA4000 disks, this bit is ignored.

Step, DirectionIn: The position of the read/write heads on both the SA1000 and SA4000 disks is controller by a stepper motor. The heads will move one cylinder for each complete pulse of the Step line. A pulse is sent by sending two control words. In the first, the Step line is set, the the second, it is reset. The direction of movement is governed by the DirectionIn line. When DirectionIn is set during a series of step commands, the heads will move towards the higher numbered cylinders in the middle of the disk. To satisfy the disk's setup times, a command word containing the proper DirectionIn bit should be sent at least one cycle before the first Step pulse It is also recommended that microcode should wait for SeekComplete before beginning any stepping operation.

Both the SA1000 and SA4000 drives have two stepping modes, *normal* and *buffered*. In normal mode, a 1 microsecond pulse is sent every millisecond. The heads move every time a pulse is sent. This mode is used during a recalibration so the Track00 signal may be sensed. In buffered step mode, a series of Step pulses with a mimimum period of 2 uS and minimum width of 1 uS is accumulated in the drive. Once 350 uS have elapsed without pulses, the drive moves the heads. If more step pulses arrive once the heads are in motion, their final position is undefined. Thus, buffered step mode should be used by microcode, not software, so pulse timing may be rigidly controlled.

Operation Control

FirmwareEnable: The FirmwareEnable bit is set whenever the disk microcode is running. In addition to acting as a status bit for higher level software, it is used to generate a service request for overhead operations.

TransferEnable: TransferEnable is set whenever a data transfer is taking place. A data transfer encompasses exactly one field of a sector. Writing or reading the data of a sector will generally require three data transfers (verify header, verify label and write or read data). The transfer operation includes the recognition or writing of the VFO synchronization pattern, sync word or address mark and the CRC checksum as well as transferring the data. When TransferEnable is reset, all the state machines used to transfer or recognize data are reset.

WriteCRC: The WriteCRC bit causes the CRC checksum to be written at the end of a field. The BTransferEnable and BWriteEnable lines must also be true for this to be accomplished. Proper use of this bit in writing a field will be explained in the section on microcode usage.

WakeupControl.(0,1): These bits together with TransferEnable are used to specify the condition generating the microcode service request. The conditions allowed are:

TransferEnable	WakeupControl.(0,1)	Condition
0	00	FirmwareEnable
0	01	SeekComplete
0	10	SectorFound (valid only on SA4000)
0	11	IndexFound
1	00	Word Ready from Read operation
1	01	Word Needed for Write or Verify operation
1	10	<no wakeup>
1	11	<no wakeup>

WriteEnable: The WriteEnable bit controls the write amplifier on the drive. In addition, it is used by the controller to decide when a write operation is taking place. The WriteGate to the drive is enabled only when WriteEnable and TransferEnable are true and Overrun is false.

Status Register

The status register is read using the "_ ~KStatus" clause in microcode. All status bits are inverted on the X bus because use of the comparable non-inverting drivers was forbidden when the board was designed. The bits will be described as though the inverstion were not present. It is expected that when the user either reads the bits into the CP or uses them as X bus branch conditions, the inversion will be taken into account.

There are two main purposes for status bits: diagnostic and operational. Some bits are included so diagnostic code may attempt to isolate a fault to either the drive or the controller. Operational bits are needed for normal operations. Diagnostic bits are generally those sent to the drive and also read back by the controller.

HeadSelect1'- HeadSelect16': These are diagnostic lines. They should give an inverted version of the head select lines in the control register. They are used to check that the proper head is actually being selected.

SeekComplete: This signal indicated the read/write heads are ready for use. It is set when the drive is ready , it is selected and the heads are not in motion. Head motion can be divided into two parts. First the stepper motor guides the heads to a new cylinder. Second, after they arrive, they vibrate for a few milliseconds. The first interval is called the seek time, the second is called the head settling time. The head settling time for both the SA1000 and SA4000 drives is about 20 mS. This can be much larger than the seek time for short seeks. The SA1000 drives supply their own head settling delay so their SeekComplete really means the heads have stopped. The SeekComplete signal from the SA4000 drives means only that the stepper motor has arrived, the 20 mS must be added externally. This is done in the controller hardware (by counting 29 sector pulses). Thus as far as the user is concerned, SeekComplete always means the heads have moved and settled. This counting of 29 sector pulses when an SA4000 type disk is attached is the controller hardware's only assumption about the number of sectors on a track. If the number of sectors on the SA4000 or SA4100 type of disk is increased, some sort if external delay will be needed.

Track00: This status line becomes active whenever the disk's read/write heads are over cylinder 0. It is probably only valid when SeekComplete is asserted. It is used by microcode and software to recalibrate the heads. Note there are a few cylinders beyond cylinder, just as there a few beyond the maximum cylinder. A recalibration algorithm should take this into account. In particular, simply stepping out from the current position is not guaranteed to lead to cylinder 0.

FirmwareEnable: This bit is used to indicate the microcode is active. It directly reflects the FirmwareEnable control bit. It is mostly by convention that this bit is set while the microcode is active; it would be possible to turn it off when the service requests are derived from another source. The convention is useful when synchronizing software with disk microcode.

IndexFound: The index pulse from the drive occurs once per revolution and lasts between 1 and 10 uS. It is used to mark a specific position on the disk, usually the beginning of sector 0 on all tracks. The IndexFound bit is a latched version of the drive's index pulse. The latch is cleared using the "ClrKFlags" clause in microcode. the IndexFound flag may also be used to generate service requests.

SectorFound/ HeaderTag': The meaning of this bit depends on the drive connected. When an SA4000 or SA4100 type drive is being controlled, a latched version of the drive's sector pulse is available here. The latch may be cleared using the "ClrKFlags" clause in microcode. The SectorFound flag is commonly used to generate a service request so the microcode may detect the start of a sector.

The SA1000 drives have no sector pulse. In order to find the beginning of a sector, the microcode commands the controller to verify each field as it arrives. The address mark used for header fields differs from that used for label and data fields. The header address mark has a 0 in bit 14, the address mark used for label and data fields has a 1 there. After reading a field, the value of bit 14 is displayed on this status bit when an SA1000 type drive is connected. Using it, microcode may verify that the field seen was indeed a header field in addition to having the correct data and CRC. The polarity was chosen so this bit could be used as an error indicator when looking for the correct header (1 => not a header). Use of this bit is explained further in the section on microcode usage.

SA1000/SA4000': This bit is set when an SA1000 type drive is attached to the controller. It is reset when an SA4000 or SA4100 type drive is attached. The two classes of drives require completely different cables. This bit is connected to a line that is grounded in the SA4000 and SA4100 cables and is pulled up in the SA1000 cable. Note the controller gives no hint about the number of heads per track or other drive variables. Determination of other disk parameters is initially done using experimentation. It is expected that configuration information will be recorded on the disk for normal use.

DriveNotReady: The drive's Ready line is inverted and sent here. The Ready line indicates the drive has power, is warmed up, is selected and is generally ready for use. The line is inverted here so it may be used as an error flag (not ready => error). Software and/or microcode should wait for this line to become active after power on before initiating any operations. In addition, it should be checked after each operation to ensure the disk hasn't broken.

WriteFault: Each type of drive can detect some internal error conditions. On the SA4000 and SA4100 drives these include WriteGate without write current in the selected head or vice versa, multiple heads selected, WriteGate active when Ready inactive and WriteGate and ReadGate active simultaneously. The SA1000 set is less comprehensive including only write current without WriteGate and multiple heads selected. When a WriteFault occurs (not necessarily only during write operations), it is latched in the drive. This status bit is a buffered version of the drive's latch. In general, service personnel prefer that software not automatically clear this line when an error is detected. This gives them some chance to see which condition caused the problem. This line should be cleared at the beginning of an operation. On the SA4000 and SA4100 type drives, it is cleared by asserting both DriveSelect and FaultClear in a command word, then sending a command with only DriveSelect. The SA1000's WriteFault is cleared by de-selecting the drive (writing a command word with DriveSelect=0) for at least 500 ns. If, because of some hardware condition, an Overrun occurs, the controller will immediately clear WriteEnable. This sometimes causes a WriteFault. The WriteFault will then persist through subsequent operations until cleared though the Overrun may vanish with the next operation. When having a WriteFault problem, it is best to see if it is caused by an Overrun.

Overrun: It is important to minimize damage to the disk if the processor runs wild and spuriously enables a write operation. If the controller's service requests for data are not answered, the Overrun bit will be set and WriteEnable turned off. If this happens early in the field being written, the drive will sometimes detect a WriteFault as explained above. Presence of this bit means either the controller or the drive is broken or that the jumpers on the drive are not correct. Disk microcode should check this status bit after every operation.

CRCError: The controller contains a 16 bit cyclic redundancy code (CRC) generator and checker. The WriteCRC control bit is used to append the generator's contents to each field written. After each field read or verified, this bit should be checked by microcode to ensure the field had the correct CRC. Like all the error bits, this one is set only when there has been an error. The CRCError bit is valid only just after the checksum word has been processed by the checker. There is a one word window for the microcode to stop the transfer, freezing the status. This is discussed in the microcode usage section. The CRCError bit is reset using the microcode's "ClrKFlags" clause before each operation.

VerifyError: The verify operation compares bits on the disk with a template in memory. It is used mainly to find headers and check labels. The verify operation is implemented by writing the template to the controller while it is reading the disk data. If one or more of the bits differs, the VerifyError bit is set. It is reset using the "ClrKFlags" clause in microcode.

Test Register

This register is used by diagnostic code to read signals on the cables leading from the HSIO board. In this way, the diagnostic code may decide whether a particular fault lies in the HSIO card or in the attached peripheral. The register is read using the "_ ~KTest" clause in microcode.

DiskReadClk: This signal is used only when controlling SA4000 and SA4100 drives. It allows the processor direct access to the disk's 140 ns clock. Since this clock is not synchronized with the processor clock, any given sample of it may return either a 1 or a 0. Diagnostic code should read it repeatedly to see if it changes state. The online diagnostics require detection only of stuck-at faults.

DiskReadData: This is the data directly from the disk. The SA4000 and SA4100 disks return NRZ (Non Return to Zero) data; the SA1000 returns 50 ns MFM (Modified Frequency Modulation) pulses. Again, the diagnostic microcode only hopes to catch this line changing state with repeated samples.

DiskOutputClk: The SA4000 drives use this clock to sample the controller's write data. The SA1000 drives use it as a time base for seek operations. It is another signal diagnostic code can sample.

DiskWriteData: This can actually be controlled by the diagnostic code. By writing words of either all 0's or all 1's this line can be set to 0 or 1.

SeekComplete': This is a version of SeekComplete directly from the cable. The controller delays an multiplexes this line before sending it to the KStatus port (see above).

DirectionIn': This is one of the signals sent to the drive that is re-received from the cable. It is used to test the control register and the drivers.

BHoriz: Display signals are also available in this register. This is the horizontal sync signal sent to the monitor. It is active for ~7 uS every 28.8 uS. As usual, it may be sampled by diagnostic code.

ReduceIW': The version of the ReduceIW signal (see Control Register above) on the interface cable to the SA1000 disk is available here. It may be directly controlled by the diagnostic code.

TTLVideo: This is the positive true version of the video signal sent to the monitor. Since this has a minimum pulse width of 19.59 ns, it probably shouldn't be sampled arbitrarily. One may set the border pattern to all zeros or all ones then have the display controller send all border pattern. In this way, the video signal will usually take on the known value. About 1/4 of the time (7/28.8) it will always be set to zero for horizontal retrace.

Sector': The SA4000 and SA4100 drives send a pulse at the beginning of each sector. The pulses are 1.1 uS in duration and occur roughly every 710 uS. By diligent sampling, diagnostic code may see this line change state.

DriveSelect': Like ReduceIW and DirectionIn, this line is available directly from the interface cable to test the control register and drivers.

BVert': This is the display's vertical sync signal. It is active LO. It may be set or reset directly in the DCtl register.

TTLVideo': This is the negative true version of the display's video signal. It was included in addition to TTLVideo so that both halves of the differential driver might be tested.

Step': This is another cable signal available to test the control register and drivers.

ReadGate': Only the SA4000 and SA4100 drives use ReadGate'. It is set by the controller during all read and verify transfers. Diagnostic code may start a read or verify operation then sample this signal.

WriteGate': This is the version of write enable sent to the drive. If data is not supplied by microcode after turning on WriteEnable, this signal should remain active LO for one word time, then go inactive. If the controller is serviced by either writing or reading data or writing a control word each time a service request is sent, this signal should remain active.

ReadData Register

Data read from the disk resided in one 16 bit buffer. It is read by microcode using the "_ KIData" clause. When a tranfer is in progress, one word must be read each time the controller requests service. Since the controller will request service in consecutive disk clicks, the disk microcode may use only 1 click to transfer the data. In addition, when SA4000 or SA4100 drives are connected, the data in the ReadData register is only valid in cycle 2. The timing is so close that it could only be valid in one of the cycles. Cycle 2 was chosen so the data could be written to memory.

WriteData Register

Data to be written or verified is stored in this register using the "KOData _" clause. The register holds a single 16 bit word and must be filled each time the controller sends a service request. As with all data tranfers, the microcode has only 1 click to read memory, increment the memory address, decrement the word count and decide if the end of the transfer has been reached. When the SA4000 or SA4100 is connected, the "KOData _" statement should only be executed in cycle 3. Generally data is written to the disk from memory and memory data is available in cycle 3. Note that one may substitute a read from KIData or a write to KCtl for the write to KOData in cycle 3. The read from KIData might be used during a wrap-around test and the write to KCtl is always used to send the WriteCRC command at the end of a field.

5.1.5 Microcode Usage

The most useful document for one starting to write microcode for the disk is existing disk code. The Pilot disk microcode is stored on [Idun]<WDlion>DiskDlionA.mc and [Idun]<WDlion>DiskDlionB.mc. This code is amply commented. It is broken into two files only because it is too large for Bravo to handle. The disk microcode also makes use of two definitions files stored on [Idun]<WDlion>, DiskDlion.df and Dandelion.df.

The beginning microcoder should read the *Dandelion Microcode Reference* to become acquainted with a great many interesting and obscure Dandelion facts. This discussion will assume a reasonable facility with Dandelion microcode.

The DiskDlion microcode was written to provide adequate performance while taking as few microinstructions as possible. It was decided that the SA4000 and SA4100 type disks would have 28 sectors per track (same as the Dolphin) and the SA1000 disks would have 16. Each sector has the three standard fields, Header, Label and Data. The Header field has 2 words and the Data field has 256. The Label field was originally 8 words long but finally grew to 12 words. The microcode had to be written so operations could be carried out on runs of consecutive sectors crossing track boundries. It was hoped that the microcode could fit in 128 control store words but 256 words was acceptable. The current code fits in 236 words.

The requirement for processing consecutive sectors puts severe timing constraints on the code. It limits the amount of inter-field, inter-sector and inter-track overhead allowed. The original code took a compact command representation, parsed it and generated the necessary control words. This code not only did not meet the timing requirements, it was also much too large. The second version of the code required the user to specify series of disk operations as small program of simple instructions in the IOCB. This took advantage of the fact that the same task might be needed many times in a run of pages, but code to implement that task would only occur once. An instruction to the disk microcode might be Increment and Skip If Zero or TranferField. This approach also allowed the user great flexibility at the head level; diagnostics could use the standard disk microcode and the disk format could be changed without changing the microcode. The resulting code took only 128 words but did not satisfy the performance requirements. The final version is based on the second one, with a "Transfer Run of Pages" command and a "Load Parameters" added. The parameters specify the operation to be performed on each field, the length and location of each field in memory and the error mask to be used.

This document will assume that the reader wishes to know how to use the controller hardware, not how to load parameters or determine a disk format. The controller hardware is designed to assist with the transfer of a single field within a sector. It has no knowledge of the number of cylinders, heads or sectors on the disk (except as noted in the explanation of the SeekComplete status bit). The DiskDlion microcode has a subroutine called TransferField that accepts as input the field’s operation, length and location in memory. It is used for all read, write and verify operations. The rest of this chapter will be concerned with the TransferField subroutine.

Although the same routine is used to perform all operations on all fields with both the SA1000 and SA4000 type disks, the operations will be explained separately. The reader may use the TransferField routine as an example of how they may be combined. General principles which apply to all operations will be explained first.

The controller hardware contains no information about the length of the field it is processing. When writing, it writes the data given until it receives a disabling control word instead of a data word. The same is true of reading and verifying. The length of each field is determined by microcode.

Timing, especially for the SA4000 and SA4100 disks, is critical. Those drives contain data separators which should only be enabled when the heads are over synchronization gaps containing

all zeros. The microcode calculates the position of the read/write heads by dead reckoning. It can sense the index and sector pulses from the drive and can know the number of microinstructions executed since the pulse. As a result of this, the number of microinstructions executed between calls to TranferField cannot be a function of the operation being executed. In fact, the number clicks executed between the end of a field and the beginning of the next field must be independent of operation. Of course, it is reasonable for the number of instructions to be a function of the field. For example, the number of clicks executed between the end of the Label field and the beginning of the Data field should not depend on the operations performed on either the Label or Data fields though it may differ from the number of clicks between the Header and Label fields.

Writing on the SA4000 and SA4100

Each field on an SA4000 or SA4100 has 4 parts. These are:

Name	length	value
Synchronization gap	7 words	0000
Synchronization word	1 word	FFFF'X
Data	2 words	Header Field, or
	12 words	Label Field, or
	256 words	Data Field
CRC checksum word	1 word	calculated CRC checksum

The data separator in the drive needs at least 8 uS (~4 word times) to acquire the data stream. The microcode can only know the position of the read heads to an accuracy of plus or minus one click. Delaying one click after the nominal beginning of the synchronization pattern gives a real delay of from zero to two clicks. To ensure at least a 1 click delay, the code must wait for two clicks. This means the real delay could be three clicks so the synchronization gap is 3+4 or 7 words (where the time between clicks ~= 1 word time).

Code for writing on the SA4000 or SA4100 disk should proceed as specifed below. Writing the Header field is used as the example, differences between the Header and other field will be explaned later.

1. Prepare the parameters used for writing the Header field.

2. Send a command to the controller containing the number of the head to be used, DriveSelect, FirmwareEnable and WakeupControl=3. This causes the next click to take place just after the next sector mark has been found. If the field is not the Header field, this step must be skipped. The command word would be 0426'x + 800'x*HeadNumber. Note that if the Header for Sector 0 is desired, one must have the microcode find the index mark and count 27 sectors marks before starting this step. Having done this, the next sector mark must belong to sector 0. One could simply find the index mark and start writing if one were willing to make the operation of writing the first sector different than that of writing the rest of them.

3. After finding the sector mark, N_h clicks may be used for further field set up. The minimum time between when the Find Sector control word is sent and the write is started should be 10 uS (5 clicks) to give the drive time to select the heads properly.

4. The control word is sent starting the write. This control word contains the number of the head to be used, DriveSelect, FirmwareEnable, TransferEnable, WakeupControl=1 and WriteEnable. It is 0433'x + 800'x*HeadNumber.

5. The controller will write the first two words of synchronization pattern automatically. The microcode should provide 5 more words of 0 to KOData; all in cycle 3.

6. The microcode supplies one word of FFFF'x to the controller in cycle 3. This is the synchronization word used by the controller hardware to find the word boundries in the serial bit stream when the field is read.

7. Microcode should execute a loop which transfers one data word per click to the KOData port. All transfers should take place in cycle 3. See the DiskDlionB.mc file for an example of such a loop.

8. A control word should be sent causing the CRC checksum to be appended to the field. The control word is identical to the one used to start the write operation with the addition of the WriteCRC bit. It is 043B'x + 800'x*HeadNumber.

9. The same control word should be sent again. The controller is pipelined to the extent that one word is being sent to the disk while the next word is received from the processor. Thus the controller cannot be stopped now as this would cause the CRC to be chopped off. Some word must be sent to the controller to prevent an Overrun condition. Sending the same control word is as easy as anything else.

10. A command should be sent disabling the Controller. It should contain the number of the head to be used in the next field, DriveSelect and FirmwareEnable. It is 0420'x+800'x*HeadNumber.

11. The DriveNotReady, WriteFault and Overrun status bits should be checked. If there was an error, the operation should probably be aborted. The disk task's double bit memory error flag in the MStatus register should also be checked. The errors recorded while the disk task is reading memory do not cause a trap but they are recorded.

The process of writing fields other that the Header is quite similar. Step 2 may be eliminated since the sector has been found and the head number established. The number of clicks used for setup should be minimized, there is no minimum value. One should take care that the number of clicks executed between fields is independent of the operation performed on the fields.

Writing on the SA1000

This is intentionally quite similar to writing on the SA4000. The differences are that the SA1000 has no sector marks, it uses an address mark instead of a synchronization word and one is required to wait for 2 clicks to elapse after starting the CRC write intead of 1.

Because of the fact that there are no sector marks on the SA1000, the position of a Header directly determines the position of a sector. For this reason, individual sectors cannot be formatted; one must format an entire track in one run. The microcode finds the index mark and writes sectors as fast as possible. Once a track is formatted, it is, of course, possible to write the Label and Data fields of its sectors individually. Shown below is the sequence used to write the Header of Sector 0 on a track. When writing other Headers in the formatting run, the step used to find the index mark is eliminated.

Because address marks are used to define the beginning of fields, all previous address marks on a track must be erased before formatting. This is done in the Pilot system by having the head tell the microcode to write a very long sector (the length of a track). Any legal MFM pattern is adequate.

The format for a field on the SA1000 is:

Name	length	value
Synchronization gap	7 words	0000
Address Mark	1 word	A141'x - Header Field
		A143'x - Label or Data Field
Data	2 words	Header Field, or
	12 words	Label Field, or
	256 words	Data Field
CRC checksum word	1 word	calculated CRC checksum

1. Prepare the parameters used for writing the Header field.

2. Send a command to the controller containing the number of the head to be used, DriveSelect, FirmwareEnable and WakeupControl=2. This causes the next click to take place just after the index mark has been found. If the field is not the Header field of Sector 0, this step must be skipped. The command word is 0424'x + 800'x*HeadNumber.

3. After finding the sector mark, N_h clicks may be used for further field set up. The minimum time between when the Find Sector control word is sent and the write is started should be 10 uS (5 clicks) to give the drive time to select the heads properly.

4. The control word is written starting the write. This control word contains the number of the head to be used, DriveSelect, FirmwareEnable, TransferEnable, WakeupControl=1 and WriteEnable. It is 0433'x + 800'x*HeadNumber.

5. The controller will write the first two words of synchronization pattern automatically. The microcode should provide 5 more words of 0 to KOData; all in cycle 3.

6. The microcode writes the data word A141'x to the controller in cycle 3. This triggers the writing of the Header's address mark. The real address mark is an illegal MFM string. It can be distinguished from ordinary data and is used by the controller hardware to find the start of a field.

7. A loop should be executed which transfers one data word per click to the KOData port. All transfers should take place in cycle 3. See the DiskDlionB.mc file for an example of such a loop.

8. A control word should be sent causing the CRC checksum to be appended to the field. The control word is identical to the one used to start the write operation with the addition of the WriteCRC bit. It is 043B'x + 800'x*HeadNumber.

9. The same control word should be sent two more times. The controller is pipelined to the extent that one word is being sent to the disk while the next word is received from the processor. Thus the controller cannot be stopped now as this would cause the CRC to be chopped off. It additionally appears that if a short tail is not written after the CRC, it cannot be read correctly. This is why two extra cycles are taken. A word must be sent to the controller in each cycle to prevent an Overrun condition. Sending the same control word is as easy as anything else.

10. A command should be sent disabling the Controller. It should contain the number of the head to be used in the next field, DriveSelect and FirmwareEnable. It is 0420'x+800'x*HeadNumber.

11. The DriveNotReady, WriteFault and Overrun status bits should be checked. If there was an error, the operation should probably be aborted. The disk task's double bit memory error flag in the MStatus register should also be checked. The errors recorded while the disk task is reading memory do not cause a trap but they are recorded.

Writing other fields in the same sector differs only in that Step 2 can be eliminated and the address mark written in Step 6 is A143'x. This allows the microcode to distinguish between Headers and other fields when the fields are read. It is still important that the number of clicks executed between fields is independent of the operations performed on those fields.

Reading Data from the SA4000 and SA4100

The main differences between reading and writing are that one must find the synchronization gap instead of creating it and read data instead of writing it. The operations for reading a Header will be shown. The differences involved in reading other fields will be explained later.

1. Prepare the parameters used for reading the Header field.

2. Send a command to the controller containing the number of the head to be used, DriveSelect, FirmwareEnable and WakeupControl=3. This causes the next click to take place just after the next sector mark has been found. If the field is not the Header field, this step must be skipped. The command word would be 0426'x + 800'x*HeadNumber.

3. After finding the sector mark, N_h+2 clicks must be used for further field set up. Note that this is same N_h used when writing a field. The extra two clicks are used to guarantee that the read heads are inside the synchronization gap when they are enabled. The minimum time between when the Find Sector control word is sent and the read is started should be 10 uS (5 clicks) to give the drive time to select the heads properly.

4. The control word is sent starting the read. This control word contains the number of the head to be used, DriveSelect, FirmwareEnable, TransferEnable and WakeupControl=0. It is 0430'x + 800'x*HeadNumber.

5. The controller will find the synchronization word automatically. The first service request announces that the synchronization word is in the KIData buffer. This should be read. It may then either be saved or discarded. It is provided for diagnostic purposes.

6. The microcode should execute a loop which transfers one data word per click from the KIData port. All transfers should take place in cycle 2. See the DiskDlionB.mc file for an example of such a loop. Note that if the buffer address calculated in cycle 1 crosses a page boundry, the memory write operation will be aborted. Data pages in the Pilot world are always page aligned so the last click executed when transferring data must not increment the memory address. See the *Dandelion Microcode Reference* for further details.

7. An extra word or command must be read or written. This gives the controller time to process the CRC checksum at the end of the field. If the extra transfer is left out, the controller will detect an Overrun. If a command is sent, it should be the original read command.

8. A command should be sent disabling the Controller. It should contain the number of the head to be used in the next field, DriveSelect and FirmwareEnable. It is 0420'x+800'x*HeadNumber.

9. The DriveNotReady, WriteFault, Overrun and CRCError status bits should be checked. If there was an error, the operation should probably be aborted.

As usual, Step 2 is eliminated and the time is Step 3 is decreased when reading other fields. Note that the delay in the read version of Step 3 is always 2 clicks longer than in the write version of the corresponding field. For example, if there are 4 clicks between the times a Header operation is stopped and the write of a Label is started, there should be 6 clicks between the times a Header operation is stopped and a Label read is started. The extra two clicks are provided by TransferField in this code, so the time between calls to TransferField must be independent of the operation.

Reading from an SA1000

Headers are very seldom read. They are written only when the disk is being formatted. Normally they are verified. To read Header n of a track, one usually finds the index mark and reads the next n+1 Headers; all into the same buffer. This complication is not germane to this discussion. The process of reading Sector zero's Header will be shown, the normal modifications required to read other Headers and other fields will be pointed out.

1. Prepare the parameters used for reading the Header field.

2. Send a command to the controller containing the number of the head to be used, DriveSelect, FirmwareEnable and WakeupControl=2. This causes the next click to take place just after the index mark has been found. If the field is not Sector zero's Header field, this step must be skipped. The command word would be 0426'x + 800'x*HeadNumber.

3. After finding the index mark, N_h+2 clicks must be used for further field set up. Note that this is same N_h used when writing a field. The data separator used for the SA1000 is on the controller board and has no requirement about being turned on over the synchronization gap. The reading process should, however, begin promptly so Sector zero's Header will be found first.

4. The control word is sent starting the read. This control word contains the number of the head to be used, DriveSelect, FirmwareEnable, TransferEnable and WakeupControl=0. It is 0430'x + 800'x*HeadNumber.

5. The controller will find the address mark automatically. The first service request announces that the address mark is in the KIData buffer. This should be read. It may then either be saved or discarded. It is provided for diagnostic purposes.

6. The microcode should execute a loop which transfers one data word per click from the KIData port. All transfers should take place in cycle 2. See the DiskDlionB.mc file for an example of such a loop. Note that if the buffer address calculated in cycle 1 crosses a page boundry, the memory write operation will be aborted. Data pages in the Pilot world are always page aligned so the last click executed when transferring data must not increment the memory address. See the *Dandelion Microcode Reference* for further details.

7. An extra word or command must be read or written. This gives the controller time to process the CRC checksum at the end of the field. If the extra transfer is left out, the controller will detect an Overrun. Note if a command is sent, it should be the original read command.

8. A command should be sent disabling the Controller. It should contain the number of the head to be used in the next field, DriveSelect and FirmwareEnable. It is 0420'x+800'x*HeadNumber.

9. The HeaderTag, DriveNotReady, WriteFault, Overrun and CRCError status bits should be checked. If there was an error, the operation should probably be aborted. Note the HeaderTag status bit here is set if the field read was not a Header. It is available on the status bit that would have been used for SectorFound if an SA4000 had been connected.

As usual, Step 2 is deleted when not looking for Sector zero's Header field. Label and Data fields are generally read by verifying all fields encountered until a match for the desired sector's Header field is found, then reading the next fields in order. There is no requirement that the SA1000 data separator be turned on over a field of zeros but it should be enabled at least 4 word times before the address mark of the field to be read or verified.

Verifying Data on the SA4000 and SA4100

A verify operation combines the read and write operations. Data is read both from the disk and from memory and compared on the controller board. As far as the microcode is concerned, a verify starts like a read with the data separator enabled to find the field. Once the field is found, a verify is like a write in that data is sent to the controller.

The procedure for verifying a Header will be shown. As explained above, this is by far the most common operation performed on Headers. DiskDlion microcode uses the verify operation to locate the Header for the proper sector. Microcode could easily be written that woke up on every SectorFound pulse and maintained a current sector number. This was not done for simplicity.

1. Prepare the parameters used for verifying the Header field.

2. Send a command to the controller containing the number of the head to be used, DriveSelect, FirmwareEnable and WakeupControl=3. This causes the next click to take place just after the next sector mark has been found. If the field is not a Header field, this step must be skipped. The command word would be 0426'x + 800'x*HeadNumber.

3. After finding the sector mark, N_h+2 clicks must be used for further field set up. Note that this is same N_h used when writing a field. The extra two clicks are used to guarantee that the read heads are inside the synchronization gap when they are enabled. The minimum time between when the Find Sector control word is sent and the read is started should be 10 uS (5 clicks) to give the drive time to select the heads properly.

4. The control word is sent to start the verify. This control word contains the number of the head to be used, DriveSelect, FirmwareEnable, TransferEnable and WakeupControl=1. It is 0432'x + 800'x*HeadNumber. In the same click as the control word is written, but after it is written, the first memory template word must be sent to the controller. It must be sent in the same click because the next service request will not be generated until the controller has started comparing the first memory and disk words. It must be sent after the control word because the WriteData buffer is held cleared until then. All words sent before the verify operation is enabled are lost.

5. The controller will find the synchronization word automatically. The first service request announces that the second template word is needed for comparison. This is the beginning of the verify loop.

6. The microcode should execute a loop which transfers one template word per click to the KOData port. All transfers should take place in cycle 3. See the DiskDlionB.mc file for an example of such a loop.

7. Two extra words or commands must be written. This gives the controller time to process the CRC checksum at the end of the field. If the extra transfers are left out, the controller will detect an Overrun. If commands are sent, they should equal the original verify command.

8. A command should be sent disabling the Controller. It should contain the number of the head to be used in the next field, DriveSelect and FirmwareEnable. It is 0420'x+800'x*HeadNumber.

9. The DriveNotReady, WriteFault, Overrun, CRCError and VerifyError status bits should be checked. If there was an error, the operation should probably be aborted. Note that if the verify operation is being used to find the proper Header, errors in the DriveNotReady, WriteFault and Overrun are fatal whereas CRC and Verify errors only indicate the wrong Header was found. One should try every Header on the track before giving up.

The usual remarks about eliminating Step 2 and shortening the delay in step 3 apply when verifying Label or Data fields. A Verify or CRC error found when verifying a Label or Data field is always fatal. Pilot normally issues operations of the form: verify Header, verify Label, read or write Data.

Verifying Data on an SA1000

Verifying Headers is also the principle method used to find sectors on the SA1000 disks. Since the SA1000 has no sector marks however, one cannot guarantee where the reading process will begin unless the index mark is sensed. For this reason, address marks are used. These are MFM patterns that meet the data separator timing requirements but cannot occur in normal data. When enabled, the controller waits until an address mark is found before starting the verify operation. It is also quite likely that the first address mark found will not belong to a Header field. For this reason, Header address marks have a 0 in bit 14 while Label and Data address marks have a 1 there. This bit is shown on the Header Tag status bit. It may be used as an error indicator when reading or verifying Header fields.

For the sake of consistency, the process of verifying Sector zero's Header will be shown, though one seldom begins a verify operation by finding the index mark on the SA1000.

1. Prepare the parameters used for verifying the Header field.

2. Send a command to the controller containing the number of the head to be used, DriveSelect, FirmwareEnable and WakeupControl=2. This causes the next click to take place just after the index mark has been found. If the field to be verified is not Sector zero's Header field, this step must be skipped. It is normally skipped anyway. The command word would be 0426'x + 800'x*HeadNumber.

3. After finding the index mark, N_h+2 clicks may be used for further field set up. Note that this is same N_h used when writing a field. The data separator used for the SA1000 is on the controller board and has no requirement about being turned on over the synchronization gap. The reading process should however begin promptly so Sector zero's Header will be found first if this is desired.

4. The control word is written starting the verify. This control word contains the number of the head to be used, DriveSelect, FirmwareEnable, TransferEnable and WakeupControl=1. It is 0432'x + 800'x*HeadNumber. In the same click as the control word is written, but after it is written, the first memory template word must be sent to the controller. It must be sent in the same click because the next service request will not be generated until the controller has started comparing the first memory and disk words. It must be sent after the control word because the WriteData buffer is held cleared until then. All words sent before the verify operation is enabled are lost.

5. The controller will find the address mark automatically. The first service request announces that the second template word is needed for comparison. This is the beginning of the verify loop.

6. The microcode should execute a loop which transfers one template word per click to the KOData port. All transfers should take place in cycle 3. See the DiskDlionB.mc file for an example of such a loop.

7. Two extra words or commands must be written. This gives the controller time to process the CRC checksum at the end of the field. If the extra transfers are left out, the controller will detect an Overrun. Note if a command is sent, it should be the original verify command.

8. A command should be sent disabling the Controller. It should contain the number of the head to be used in the next field, DriveSelect and FirmwareEnable. It is 0420'x+800'x*HeadNumber.

9. The HeaderTag, DriveNotReady, WriteFault, Overrun, CRCError and VerifyError status bits should be checked. If there was an error, the operation should probably be aborted. Note that if the verify operation is being used to find the proper Header, errors in the DriveNotReady, WriteFault and Overrun are fatal whereas HeaderTag, CRC and Verify errors only indicate the wrong field was found. One should try every field on the track before giving up.

When Label or Data fields are verified, Step 2 is left out and the delay in Step 3 can be shortened. The HeaderTag bit is also ignored at the end of a Label or Data field operation.

Conclusion

This concludes the section on usage of the controller. The grand scheme for using the disk proceeds as follows:

1. After power on, wait for DriveNotReady to drop.

2. Clear the WriteFault line as shown in the Control Register section.

3. Recalibrate the read/write heads by stepping 20 cylinders in, then 222 (202+20 for SA4000 or SA4100) or 276 (256+20 for SA1000) out, looking for Track00 after each step is complete.

4. Seek to the desired cylinder by having the microcode issue the proper number of pulses on the Step line with the desired direction set on DirectionIn.

5. Perform the desired data transfer as outlined above.

6. If there are errors, retry. If the errors involve the WriteFault line, clear it and retry. If WriteFault errors persist, make sure Overrun isn't responsible. If the errors indicate the proper sector can't be found, try recalibrating.

7. Repeat Steps 4 through 7 as necessary.

5.2 Trident Disk Controller

(To be added)

6.0 Ethernet Controller

(To be added)

7.0 LSEP Controller

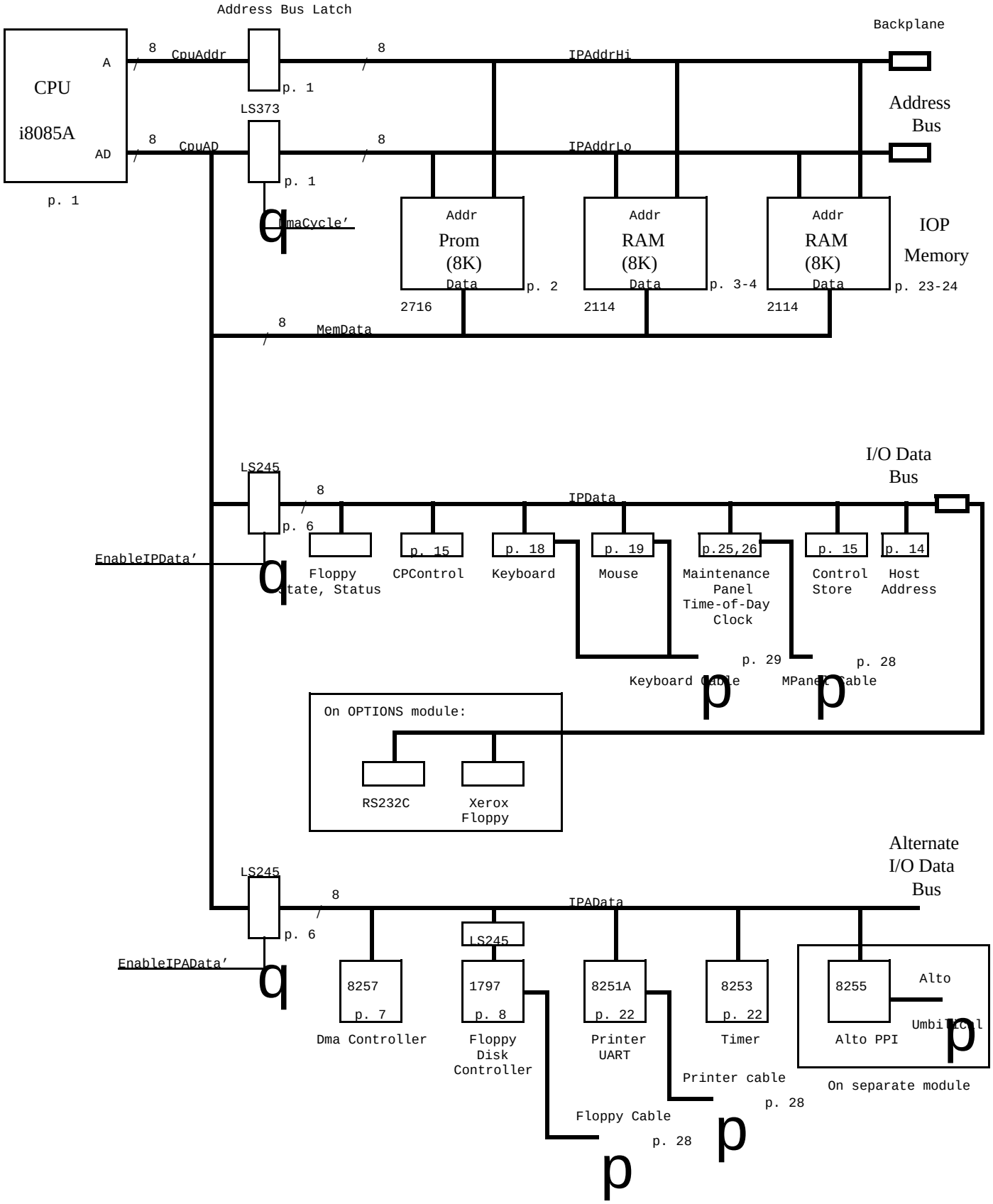
(To be added)

8.0 Magnetic Tape Controller

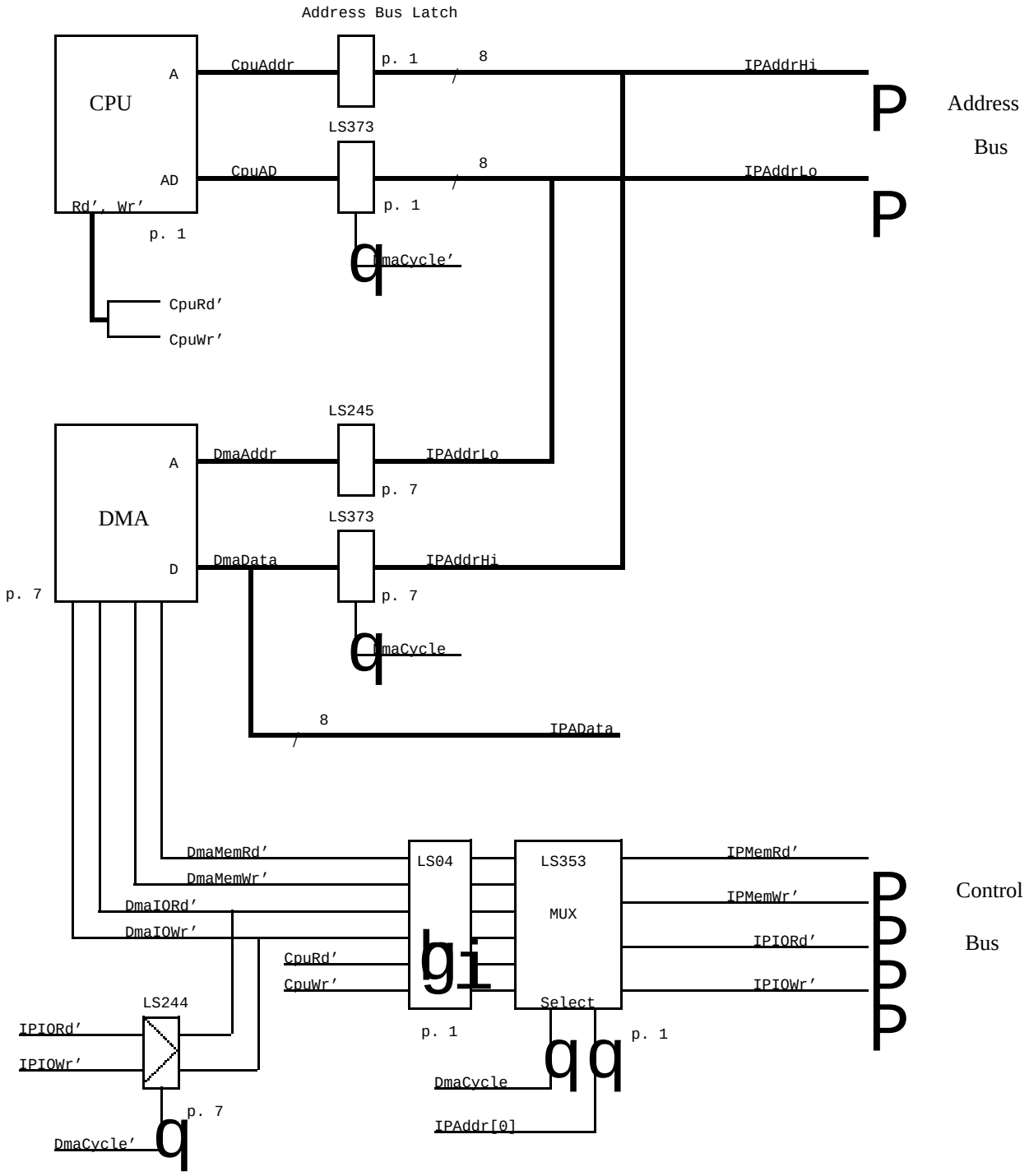
(To be added)

9.0 Input/Output Processor (IOP)

(To be added)



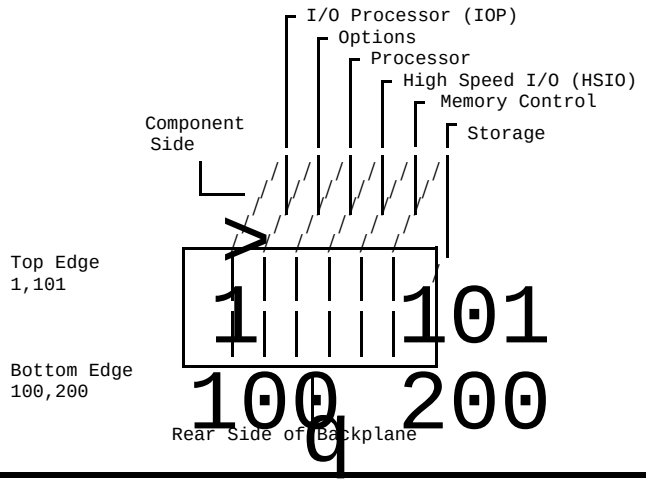
BLOCK DIAGRAM: I/O PROCESSOR DATA PATHS



BLOCK DIAGRAM: I/O PROCESSOR CONTROL ORGANIZATION

Dandelion Backplane

Physical arrangement



Files

[Iris]<Workstation>Backplane>
Backplane-C.press
Backplane-C.dm

Backplane Signals

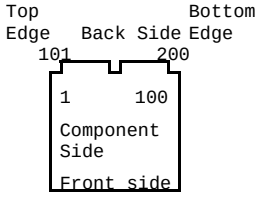
Card	IOP	Options	Central Processor	High Speed I/O	Memory Control	Storage
Total Signal lines used	131	166	140	141	155	66

170 max. per card

I/O Connectors on front of boards						
	Floppy Keyboard Printer MaintenanceP Alto umb.	LSEP/Ethernet RS232/RS366		SA4XXX SA100X Display		

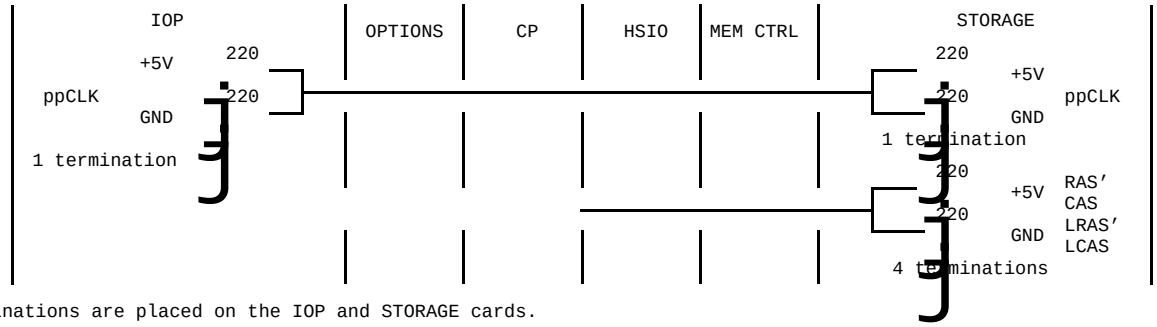
Power distribution

Backplane Power & Ground	30 lines total
Voltage	Backplane Pins
+12 V	1,101
+ 5 V	50,51,150,151
Gnd	10,20,30,40,60,70,80,90,110,120,130,140,160,170,180,190
- 5V	100,200
- 12 V	98,198
No Conn.	97,99,197,199



2- 100 pin connectors
Pins on .1" centers
.6" between pins 50 & 51
AMP # 530826-3

Termination of clock signals



Terminations are placed on the IOP and STORAGE cards.

XEROX SDD	Project Dandelion	Backplane Description General Characteristics	File WSBackplane.sil	Designer Ogus	Rev C	Date 9/26/80

	IOP			OPTIONS			CP			HSIO		
02			02	Cycle.1’	Click.0	02	Cycle1’	Click.0	02	Cycle1’	Click.0	02
03			03	Cycle.2’	Click.1	03	Cycle2’	Click.1	03	Cycle2’	Click.1	03
04			04	Cycle.3’	Click.2	04	Cycle3’	Click.2	04	Cycle3’	Click.2	04
05	Spare22	Spare23	05	Spare22	Spare23	05			05	RAS’	LRAS’	05
06	Spare2		06	Spare2		06			06	CAS	LCAS	06
07	Spare20	Spare21	07	Spare20	Spare21	07	Spare20	Spare21	07	WPulse	DR/C	07
08	Spare18	Spare19	08	Spare18	Spare19	08	Spare18	Spare19	08			08
09	ppCLK		09	ppCLK		09	ppCLK		09	ppCLK		09
11	Spare16	Spare17	11	Spare16	Spare17	11	AllowWrite		11	AllowWrite		11
12	IOPClk		12	IOPClk		12			12			12
13	Spare14	Spare15	13	Spare14	Spare15	13	MAR_	mem	13	MAR_	mem	13
14	Spare12	Spare13	14	Spare12	Spare13	14		_MStatus’	14		_MStatus’	14
15	IOPDataOut	BRClk	15	IOPDataOut	BRClk	15	MapRef	MCtl_’	15	MapRef	MCtl_’	15
16	SelTroyMode	Spare9	16	SelTroyMode	Spare9	16	Refresh’		16	Refresh’		16
17	Wait	IOPReset’	17	Wait	IOPReset’	17	Wait	IOPReset’	17	Wait	IOPReset’	17
18	TrIndex	TrHdLd	18	TrIndex	TrHdLd	18	Spare26	Spare27	18	Spare26	Spare27	18
19	TrReady	TrStep	19	TrReady	TrStep	19	Spare24	Spare25	19	Spare24	Spare25	19
21	IOPODData_’	IOPCtl_’	21	IOPODData_’	IOPCtl_’	21	IOPODData_’	IOPCtl_’	21			21
22	TrTK00	TrDirIn	22	TrTK00	TrDirIn	22	KODData_’	KCtl_’	22	KOData_’	KCtl_’	22
23			23	EODData_’	EICtl_’	23	EODData_’	EICtl_’	23	EOData_’	EICtl_’	23
24	TrWrProt	TrWrGate	24	TrWrProt	TrWrGate	24	DCtlFifo_’	DCtl_’	24	DCtlFifo_’	DCtl_’	24
25	TrRdData	TrWrData	25	TrRdData	TrWrData	25	DBorder_’		25	DBorder_’		25
26	EWrite’	EOCtl_’	26	EWrite’	EOCtl_’	26	EWrite’	EOCtl_’	26	EWrite’	EOCtl_’	26
27	KCmd_’	IOOutSp4_’	27	KCmd_’	IOOutSp4_’	27	KCmd_’	IOOutSp4_’	27	KCmd_’	IOOutSp4_’	27
28			28	POData_’	PCtl_’	28	POData_’	PCtl_’	28	POData_’	PCtl_’	28
29	Spare6	Spare7	29	Spare6	Spare7	29	Spare6	Spare7	29	Spare6	Spare7	29
31			31	EIData’	EStatus’	31	EIData’	EStatus’	31	EIData’	EStatus’	31
32	Spare4	Spare5	32	Spare4	Spare5	32	_KIData’	_KStatus’	32	_KIData’	_KStatus’	32
33			33		KWrite’	33	_KTest’	KWrite’	33	_KTest’	KWrite’	33
34	_IOPIData’	_IOPStatus’	34	_IOPIData’	_IOPStatus’	34	_IOPIData’	_IOPStatus’	34			34
35	_IOInSp2’		35	_IOInSp2’	PrtReq’	35	_IOInSp2’	PrtReq’	35	_IOInSp2’	PrtReq’	35
36	IOPALE	Spare3	36	IOPALE	Spare3	36	IOPALE	Spare3	36	IOPALE	Spare3	36
37	CSParErr		37	CSParErr	EndLine’	37	CSParErr	EndLine’	37		EndLine’	37
38	IODisp.0	IODisp.1	38	IODisp.0	IODisp.1	38	IODisp.0	IODisp.1	38	IODisp.0	IODisp.1	38
39	YIODisp.0	YIODisp.1	39	YIODisp.0	YIODisp.1	39	YIODisp.0	YIODisp.1	39	YIODisp.0	YIODisp.1	39
41	X.0	X.1	41	X.0	X.1	41	X.0	X.1	41	X.0	X.1	41
42	X.2	X.3	42	X.2	X.3	42	X.2	X.3	42	X.2	X.3	42
43	X.4	X.5	43	X.4	X.5	43	X.4	X.5	43	X.4	X.5	43
44	X.6	X.7	44	X.6	X.7	44	X.6	X.7	44	X.6	X.7	44
45	X.8	X.9	45	X.8	X.9	45	X.8	X.9	45	X.8	X.9	45
46	X.10	X.11	46	X.10	X.11	46	X.10	X.11	46	X.10	X.11	46
47	X.12	X.13	47	X.12	X.13	47	X.12	X.13	47	X.12	X.13	47
48	X.14	X.15	48	X.14	X.15	48	X.14	X.15	48	X.14	X.15	48
49			49	Y.0	Y.1	49	Y.0	Y.1	49	Y.0	Y.1	49
52			52	Y.2	Y.3	52	Y.2	Y.3	52	Y.2	Y.3	52
53			53	Y.4	Y.5	53	Y.4	Y.5	53	Y.4	Y.5	53
54			54	Y.6	Y.7	54	Y.6	Y.7	54	Y.6	Y.7	54
55			55	Y.8	Y.9	55	Y.8	Y.9	55	Y.8	Y.9	55
56			56	Y.10	Y.11	56	Y.10	Y.11	56	Y.10	Y.11	56
57			57	Y.12	Y.13	57	Y.12	Y.13	57	Y.12	Y.13	57
58			58	Y.14	Y.15	58	Y.14	Y.15	58	Y.14	Y.15	58
59	DmaReqC’	DmaAckC’	59	DmaReqC’	DmaAckC’	59	YH.0	YH.1	59	YH.0	YH.1	59
61	DmaReqA’	DmaAckA’	61	DmaReqA’	DmaAckA’	61	YH.2	YH.3	61	YH.2	YH.3	61
62	DmaReqB’	DmaAckB’	62	DmaReqB’	DmaAckB’	62	YH.4	YH.5	62	YH.4	YH.5	62
63	DmaCycle	ExtWaitReq’	63	DmaCycle	ExtWaitReq’	63	YH.6	YH.7	63	YH.6	YH.7	63
64	IOPIntReq0	IOPIntReq1	64	IOPIntReq0	IOPIntReq1	64	Pt.0	Pt.1	64	Pt.0	Pt.1	64
65	IOPIntReq2	IOPIntReq3	65	IOPIntReq2	IOPIntReq3	65	Pt.2		65	Pt.2		65
66	IOPSel.0’	IOPSel.1’	66	IOPSel.0’	IOPSel.1’	66	Disp-Proc’	MemErr	66	Disp-Proc’	MemErr	66
67	IOPSel.2’	IOPSel.3’	67	IOPSel.2’	IOPSel.3’	67			67	DAddr.0	DAddr.1	67
68	IOPSel.4’	IOPSel.5’	68	IOPSel.4’	IOPSel.5’	68			68	DAddr.2	DAddr.3	68
69	IOPAddr.00	IOPAddr.01	69	IOPAddr.00	IOPAddr.01	69			69	DAddr.4	DAddr.5	69
71	IOPAddr.02	IOPAddr.03	71	IOPAddr.02	IOPAddr.03	71			71	DAddr.6	DAddr.7	71
72	IOPAddr.04	IOPAddr.05	72	IOPAddr.04	IOPAddr.05	72			72	DAddr.8	DAddr.9	72
73	IOPAddr.06	IOPAddr.07	73	IOPAddr.06	IOPAddr.07	73			73	DAddr.10	DAddr.11	73
74	IOPAddr.08	IOPAddr.09	74	IOPAddr.08	IOPAddr.09	74			74	DAddr.12	DAddr.13	74
75	IOPAddr.10	IOPAddr.11	75	IOPAddr.10	IOPAddr.11	75			75	DAddr.14	DAddr.15	75
76	IOPAddr.12	IOPAddr.13	76	IOPAddr.12	IOPAddr.13	76	IOPAddr.12	IOPAddr.13	76	DData.0	DData.1	76
77	IOPAddr.14	IOPAddr.15	77	IOPAddr.14	IOPAddr.15	77	IOPAddr.14	IOPAddr.15	77	DData.2	DData.3	77
78	Spare0	Spare1	78	Spare0	Spare1	78	Spare0	Spare1	78	DData.4	DData.5	78
79	IOPMemRd’	IOPI/ORd’	79	IOPMemRd’	IOPI/ORd’	79			79	DData.6	DData.7	79
81	CSWE.a’	CSWE.b’	81	CSWE.a’	CSWE.b’	81	CSWE.a’	CSWE.b’	81	DData.8	DData.9	81
82	CSWE.c’	CSWE.d’	82	CSWE.c’	CSWE.d’	82	CSWE.c’	CSWE.d’	82	DData.10	DData.11	82
83	CSWE.e’	CSWE.f’	83	CSWE.e’	CSWE.f’	83	CSWE.e’	CSWE.f’	83	DData.12	DData.13	83
84	IOPReq’	ClrIOPReq’	84	IOPReq’	ClrIOPReq’	84	IOPReq’	ClrIOPReq’	84	DData.14	DData.15	84
85			85	DPReq’	ClrDPReq’	85	DPReq’	ClrDPReq’	85	DPReq’	ClrDPReq’	85
86			86	EReq’	ClrRefReq’	86	EReq’	ClrRefReq’	86	EReq’	ClrRefReq’	86
87	IOPMemWr’	IOPI/OWr’	87	IOPMemWr’	IOPI/OWr’	87	KReq’	ClrKFlags’	87	KReq’	ClrKFlags’	87
88	RefReq’	ReadCSEn’	88	RefReq’	ReadCSEn’	88	RefReq’	ReadCSEn’	88	RefReq’		88
89	EORound	IOPWait	89	EORound	IOPWait	89	EORound	IOPWait	89	EORound		89
91	WrTPCHigh’	WrTPCLow	91	WrTPCHigh’	WrTPCLow	91	WrTPCHigh’	WrTPCLow	91			91
92	IOPData.0	IOPData.1	92	IOPData.0	IOPData.1	92	IOPData.0	IOPData.1	92			92
93	IOPData.2	IOPData.3	93	IOPData.2	IOPData.3	93	IOPData.2	IOPData.3	93			93
94	IOPData.4	IOPData.5	94	IOPData.4	IOPData.5	94	IOPData.4	IOPData.5	94			94
95	IOPData.6	IOPData.7	95	IOPData.6	IOPData.7	95	IOPData.6	IOPData.7	95			95
96	SwTAddr	SwTAddr’	96	SwTAddr	SwTAddr’	96	SwTAddr	SwTAddr’	96			96
	1-100	101-200		1-100	101-200		1-100	101-200		1-100	101-200	

Above diagram is rear view (wiring side) of backplane.
ALL NUMBERS ARE IN DECIMAL.

Dandelion Backplane Signals - 1

Stamen1-4.sil in: [Iris]<Workstation>Backplane>Backplane-C.dm

Rev C	9/26/80
Ogus	

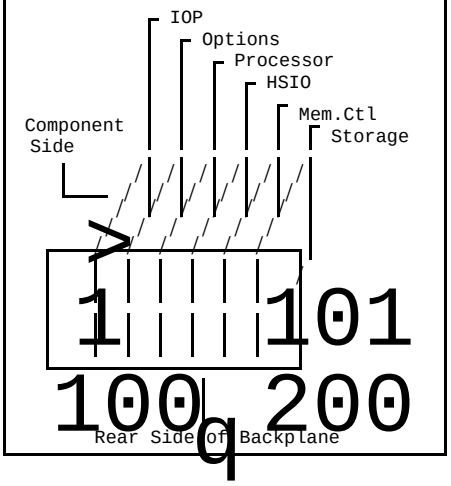
	MEM CTRL			STORAGE		
02	Cycle1′		02			02
03	Cycle2′		03			03
04	Cycle3′		04			04
05	RAS′	LRAS′	05	RAS′	LRAS′	05
06	CAS	LCAS	06	CAS	LCAS	06
07	WPulse	DR/C	07			07
08			08			08
09	ppCLK		09	ppCLK		09
11	AllowWrite		11			11
12	Bank0′		12	Bank0′		12
13	MAR_	mem	13			13
14		_MStatus′	14			14
15	MapRef	MCtl_′	15			15
16	Refresh′	CRefresh′	16	Refresh′	CRefresh′	16
17	Wait		17			17
18	SDO.00	SDO.01	18	SDO.00	SDO.01	18
19	SDO.02	SDO.03	19	SDO.02	SDO.03	19
21	SDO.04	SDO.05	21	SDO.04	SDO.05	21
22	SDO.06	SDO.07	22	SDO.06	SDO.07	22
23	SDO.08	SDO.09	23	SDO.08	SDO.09	23
24	SD0.10	SDO.11	24	SD0.10	SDO.11	24
25	SD0.12	SDO.13	25	SD0.12	SDO.13	25
26	SDO.14	SDO.15	26	SDO.14	SDO.15	26
27	SDO.16	SDO.17	27	SDO.16	SDO.17	27
28	SDO.18	SDO.19	28	SDO.18	SDO.19	28
29	SDO.20	SDO.21	29	SDO.20	SDO.21	29
31			31			31
32	SAddr.00	SAddr.01	32	SAddr.00	SAddr.01	32
33	SAddr.02	SAddr.03	33	SAddr.02	SAddr.03	33
34	SAddr.04	SAddr.05	34	SAddr.04	SAddr.05	34
35	SAddr.06	SAddr.07	35	SAddr.06	SAddr.07	35
36	Y1Latch	Y0Latch	36	Y1Latch	Y0Latch	36
37	Bank1′	Bank2′	37	Bank1′	Bank2′	37
38	MRef′	Write′	38	MRef′	Write′	38
39			39			39
41	X.0	X.1	41			41
42	X.2	X.3	42			42
43	X.4	X.5	43			43
44	X.6	X.7	44			44
45	X.8	X.9	45			45
46	X.10	X.11	46			46
47	X.12	X.13	47			47
48	X.14	X.15	48			48
49	Y.0	Y.1	49			49
52	Y.2	Y.3	52			52
53	Y.4	Y.5	53			53
54	Y.6	Y.7	54			54
55	Y.8	Y.9	55			55
56	Y.10	Y.11	56			56
57	Y.12	Y.13	57			57
58	Y.14	Y.15	58			58
59	YH.0	YH.1	59			59
61	YH.2	YH.3	61			61
62	YH.4	YH.5	62			62
63	YH.6	YH.7	63			63
64	Pt.0	Pt.1	64			64
65	Pt.2		65			65
66	Disp-Proc′	MemErr	66			66
67	DAddr.0	DAddr.1	67			67
68	DAddr.2	DAddr.3	68			68
69	DAddr.4	DAddr.5	69			69
71	DAddr.6	DAddr.7	71			71
72	DAddr.8	DAddr.9	72			72
73	DAddr.10	DAddr.11	73			73
74	DAddr.12	DAddr.13	74			74
75	DAddr.14	DAddr.15	75			75
76	DData.0	DData.1	76			76
77	DData.2	DData.3	77			77
78	DData.4	DData.5	78			78
79	DData.6	DData.7	79			79
81	DData.8	DData.9	81			81
82	DData.10	DData.11	82			82
83	DData.12	DData.13	83			83
84	DData.14	DData.15	84			84
85	SDI.20	SDI.21	85	SDI.20	SDI.21	85
86	SDI.18	SDI.19	86	SDI.18	SDI.19	86
87	SDI.16	SDI.17	87	SDI.16	SDI.17	87
88	SDI.14	SDI.15	88	SDI.14	SDI.15	88
89	SDI.12	SDI.13	89	SDI.12	SDI.13	89
91	SDI.10	SDI.11	91	SDI.10	SDI.11	91
92	SDI.08	SDI.09	92	SDI.08	SDI.09	92
93	SDI.06	SDI.07	93	SDI.06	SDI.07	93
94	SDI.04	SDI.05	94	SDI.04	SDI.05	94
95	SDI.02	SDI.03	95	SDI.02	SDI.03	95
96	SDI.00	SDI.01	96	SDI.00	SDI.01	96
	1-100	101-200		1-100	101-200	

Above diagram is rear view (wiring side) of backplane.
All numbers are in DECIMAL.

Dandelion Backplane - 2

Rev C	9/26/80
Ogus	

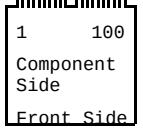
Physical arrangement of cards:



Power & Ground

Voltage	Pins
+12 V	1,101
+5 V	50,51,150,151
GND	10,20,30,40,60 70,80,90,110,120, 130,140,160,170, 180,190
-5 V	100,200
-12 V	98,198
No Conn	97,99,197,199

Card Edge Connector



2- 100 pin connectors
Pins on .1" centers
.6" between pins 50 & 51

2 - AMP # 530826-3

Stamen5-6.sil in:
[Iris]<Workstation>Backplane>Backplane-C.dm