AD-A066 240 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 17/5
PROTOTYPE AUTOMATIC TARGET SCREENER.(U)
JAN 79 D E SOLAND, M O SCHROEDER, R C FITCH DAAK70-77-C-0248
UNCLASSIFIED 79SRC4 NL

| OF |
AD
A066240

END
DATE
FILMED
5-79
DDC

PROTOTYPE AUTOMATIC TARGET SCREENER

by

D.E. Soland
M.O. Schroeder
R.C. Fitch
D.V. Serreyn
T.G. Kopet

8 January 1979

Quarterly Report for Period
1 October 1978 – 31 December 1978

Night Vision and Electro-optics Laboratory
Fort Belvoir, Virginia 22060

Honeywell
SYSTEMS & RESEARCH CENTER
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOV'T ACCESSION NUMBER	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE (AND SUBTITLE) 6 PROTOTYPE AUTOMATIC TARGET SCREENER		5. TYPE OF REPORT/PERIOD COVERED Quarterly Progress Report 1 October - 31 December 1978
		6. PERFORMING ORG. REPORT NUMBER 14 79SRC4
7. AUTHOR(S) 10 D. E. Soland, M. O. Schroeder, R. C. Fitch, D. V. Serreyn, T. G. Kopet		8. CONTRACT OR GRANT NUMBER(S) 15 DAAK70-77-C-0248
9. PERFORMING ORGANIZATIONS NAME/ADDRESS Honeywell Systems and Research Center, 2600 Ridgway Parkway, Minneapolis, Minnesota 55413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 16 1 E263710DK70 14 010CJ
11. CONTROLLING OFFICE NAME/ADDRESS Night Vision and Electro-Optics Laboratory, Fort Belvoir, Virginia 22060		12. REPORT DATE January 8, 1979 17 14
		13. NUMBER OF PAGES 56
14. MONITORING AGENCY NAME/ADDRESS (IF DIFFERENT FROM CONT. OFF.) 11 8 Jan 79		15. SECURITY CLASSIFICATION (OF THIS REPORT) Unclassified 12 63p.
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (OF THIS REPORT)
Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

Infrared	Target recognition	Image enhancement
FLIR	Pattern recognition	
Target Cueing	Image processing	
Target screening	Real time	

20. ABSTRACT (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)
This report is the fifth quarterly progress report for contract DAAK70-77-C-0248, Prototype Automatic Target Screener. The objective of the effort is to design an automatic target screener to be used with thermal imaging systems employing common module components.

HD-168 REV 11/74

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 55 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

402 349 JOB

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (concluded)

LIST OF TABLES

SECTION I

INTRODUCTION AND SUMMARY

This is the fifth quarterly technical progress report for contract number DAAK70-77-C-0248, Prototype Automatic Target Screener (PATS). The first two quarterly reports documented the Phase I design study. The third quarterly report included a description of the final target classifier design for the target data base currently available and the results of the hardware and CPU1 software system design tasks. This report continues the description of subsystem design details and the status of hardware fabrication and software coding and presents results of checkout of the edge circuit subassembly, the first subsystem to be completed and checked out. This report covers the period from 1 October to 31 December, 1978.

The program objective is to produce a design for an automatic target screener. The screener will reduce the task loading on the thermal imager operator by detecting and recognizing a limited set of high-priority targets at ranges comparable to or greater than those for an unassisted observer. A second objective is to provide enhancement of the video presentation to the operator. The image enhancement includes (1) automatic gain/brightness control to relieve the operator of the necessity to continually adjust the display gain and brightness controls and (2) DC restoration to eliminate artifacts resulting from ac coupling of the infrared (IR) detectors.

Image enhancement will also include local area gain and brightness control to enhance local variations of contrast and compress the overall scene dynamic range to match that of the display. This circuitry has been completed, and examples of its performance on videotaped thermal image data were included, with the circuit description, in the first quarterly report.

The DC restoration image enhancement circuit eliminates the streaking associated with loss of line-to-line correlation on the displayed image because of the ac coupling of the detector channels.

This report consists of five sections. Section II describes further results of detailed circuit designs. Section III includes a description and results of the edge circuit checkout. Section IV summarizes the status of the software design and coding tasks, including a description of the functions to be implemented in higher order language code in CPU2. These functions include diagnostic routines for system integration and checkout as well as functional routines as part of the PATS operational system. Section V summarizes plans for the next reporting period.

SECTION II

HARDWARE DESIGN

This section describes those design tasks that were either completed, modified, or added during this reporting period. For review purposes, the PATS hardware tasks were broken down into the following subparts:

- Image Enhancement
- Edge Signal
- Bright Signal
- Interval Generation
- CPU1
- Memory 2 (intensity information)
- CPU2
- Symbol Generation
- Sync and Timing

In previous reports, the design tasks for image enhancement including DC restore, edge signal, and CPU1 were reported.

During this reporting period, the following has taken place:

1. A modification to the sync and timing section has been made.
2. A task for the interface between CPU1 and CPU2 has been added.
3. A task for building a writable control store in the Intel MDS has been added.

This section includes a status table of the various hardware modules used in PATS, a disucssion of the designs modified or added, and a discussion of the results obtained on the edge checkout.

STATUS OF MODULES

Table 1 presents the status to date of the functional subassemblies defined for PATS. The percentage completed is a rough estimate of where we are with each task. One hundred percent means that the task is essentially complete but changes may be made during checkout. Included in the status are preliminary schematics to be used for build and checkout. Some functions are broken down to reflect the actual number of boards used in the system.

SYSTEM SYNCHRONIZATION AND TIMING

Since the last reporting period, the system synchronization and timing unit has been totally designed. There are two boards in the unit. The first board provides sync separation and video switching. The second board generates timing signals common to the other PATS system functions.

Sync Separation and Video Switching

The sync separation and video switching section is shown in a block diagram in Figure 1. The 525- or 875-line format input video goes into a composite sync separator which generates a delayed composite sync signal (CSYNCD). The CSYNCD signal goes into a sync signal generator producing a black video clamp signal (BCLMP), a field indicator (FLDIN), horizontal sync (HSYNCD) and vertical reset (VRST). Timing for these signals for even

TABLE 1. STATUS OF PATS HARDWARE (PERCENTAGE COMPLETED)

Subpart	Boards	Design	Schematics	Build	Check-Out
Image Enhancement					
Adaptive Contrast Enhancement	1	100	100	100	0
DC Restore	1	100	100	100	0
Edge	1	100	100	100	90
Bright	1	100	100	100	50
Interval	1	65	0	0	0
CPU1 (Digital Processing Subsystem)					
Processor Inc Multiplier	1	100	80	0	0
Microprogram Memory	1	100	80	0	0
FIFO/DMA I/F	1	80	0	0	0
Memory 1	2	95	50	0	0
CPU1/CPU2 I/F (inc in CPU2)	1	0	0	0	0
Memory 2					
A/D, Summation	1	100	100	100	0
Memory Control and Refresh	1	100	100	100	0
Memory 512 x 512 x 2	4	100	100	25	0
CPU2					
CPU with 16K Memory (KD 11-HC)	2	NA	NA	NA	90
PROM Board (MRV11-AA)	1	NA	NA	0	0
Serial Port (DLV11) inc Printer/ Keyboard	1	NA	NA	NA	100
Refresh/Bootstrap (REV11-C)	1	NA	NA	NA	100
Floppy Controller (RXV11-BA) inc Floppies	1	NA	NA	NA	100
Symbol Generator (included in CPU2)	1	0	0	0	0
Sync and Timing					
Sync Separator and Video Switches	1	100	100	100	15
Sync Generation	1	100	50	100	15
Writable Control Store	1	100	0	60	0

NA - Not applicable to PATS Design Tasks

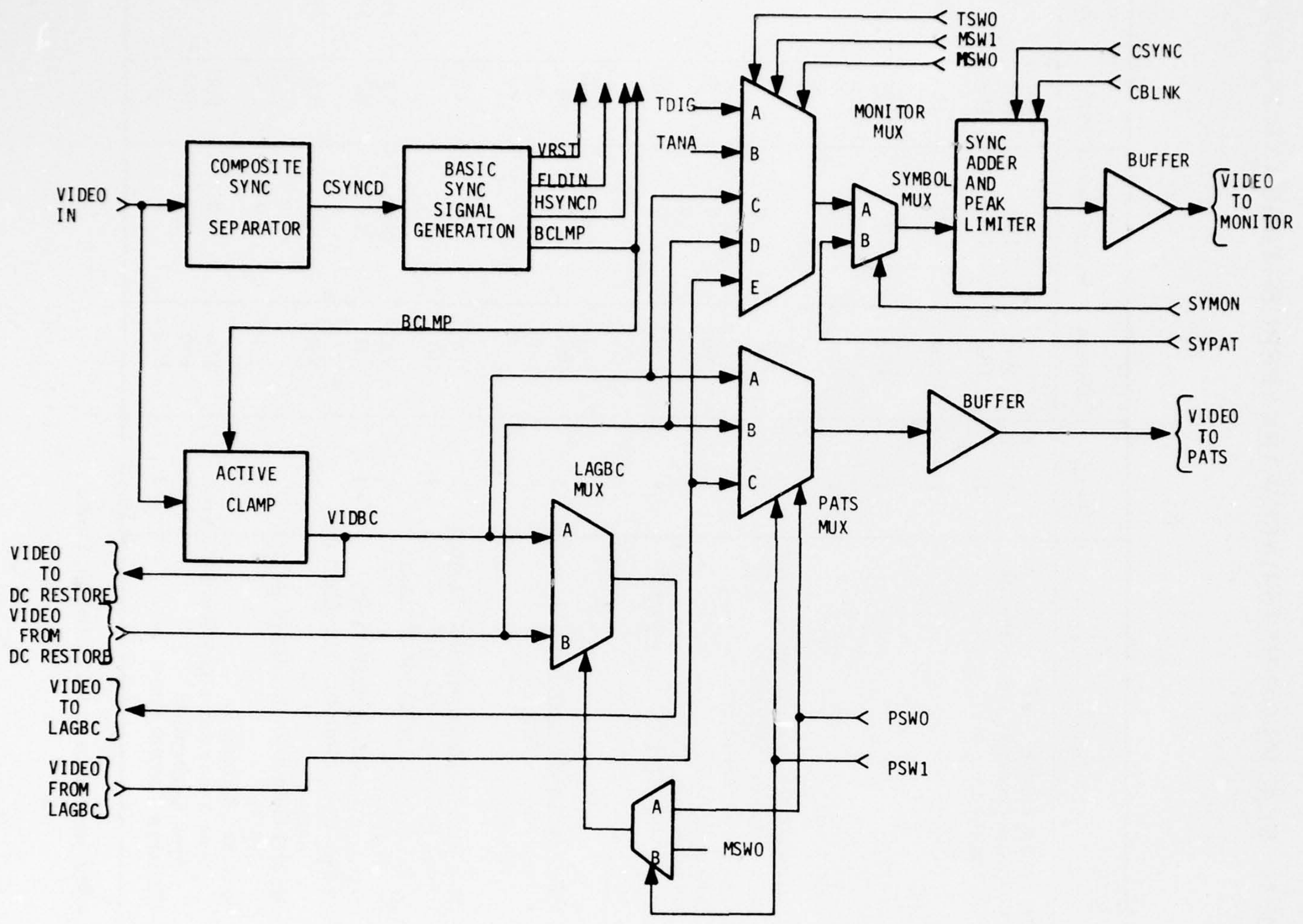

Figure 1. Sync Separation and Video Switching Block Diagram

and odd fields is shown in Figure 2. The black clamp signal (BCLMP) is generated by the trailing edge of CSYNCD ($t_{bc} \approx 1$ μsec). The horizontal sync pulse is generated from CSYNCD such that it matches CSYNCD outside of the sync serrating and equalizing regions and has a fixed period of t_H (the horizontal sync period). A vertical reset signal (VRST) is generated about 10 μsec after the first serrating pulse in each field and lasts until the next equalizing pulse. The field indicator FLDIN transition occurs at the low-to-high transition of VRST.

The input video shown in Figure 1 is clamped by the BCLMP signal causing the black level in the video to be at a zero level. The resulting video (VIDBC) goes to the input of the synthetic DC restoration unit and to three multiplexers: the LAGBC mux, the PATS mux and the MONITOR mux. The mux selection inputs are set by switches designated as PSWØ, PSW1, MSWØ, MSW1 and TSWØ. The LAGBC mux directs VIDBC or the video from DC restore to the LAGBC unit. The PATS mux sends VIDBC or video from DC restore or video from LAGBC to the PATS processing units (interval, A/D, and summation, etc. units). The MONITOR mux selects the same three inputs as the PATS mux plus a digital (TDIG) or analog (TANA) test signal from the PATS processing units. The values of the switches and their effects on the mux units are summarized in Tables 2-4. An "X" in the table means the switch can be "0" or "1".

The output of the PATS mux is buffered to the other PATS unit. The output of the MONITOR mux goes to the SYMBOL mux. The output of the SYMBOL mux is either the MONITOR mux output or a symbol pattern (SYPAT). The mux is controlled by the symbol-on code (SYMON). The signals SYPAT and SYMON will come from the symbol generator unit in PATS. The output of

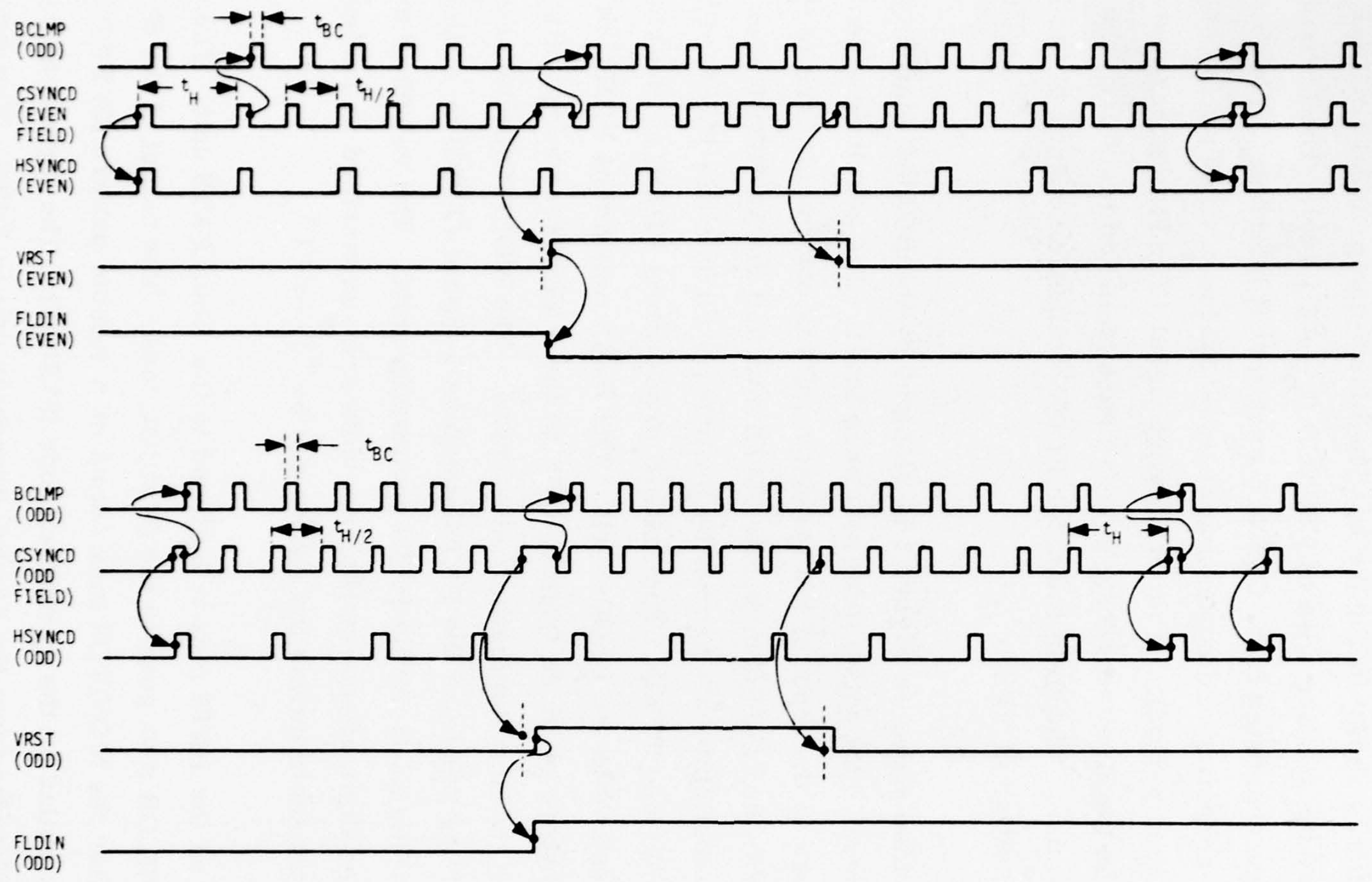

Figure 2. Basic Sync Signals Generated

TABLE 2. PATS MUX OUTPUT SWITCH SELECTION

PSW1	PSW0	PATS Mux Output
0	0	VIDBC
0	1	Video from DC Restore
1	X	Video from LAGBC

TABLE 3. MONITOR MUX OUTPUT SWITCH SELECTION

MSW1	MSWØ	MSWØ	MONITOR Mux Output
0	0	0	VIDBC
0	1	0	Video from DC Restore
1	X	0	Video from LAGBC
0	0	1	TDIG
0	1	1	TANA
1	X	1	Not Connected

TABLE 4. LAGBC MUX OUTPUT SWITCH SELECTION

PSW1	PSWØ	MSWØ	LAGBC Mux Output
0	X	0	VIDBC
0	X	1	Video from DC Restore
1	0	X	VIDBC
1	1	X	Video from DC Restore

the SYMBOL mux goes into the sync adder and peak limiter where blanking (CBLNK) and composite sync (CSYNC) are added to give a standard 1 V peak-to-peak composite video signal. The video is then buffered to drive a standard video monitor.

System Timing Generator

The system timing generator produces sync signals and clocks that are synchronized to the incoming video sync pulses and are used by the other PATS units. A block diagram is shown in Figure 3. Two clocks are generated by phase-lock multiplying the horizontal sync signal HSYNCD from the sync separator and video switching unit. The clocks produced are 50 percent duty cycle. The 512-clock has 512 clock pulses per active video in the horizontal scan line and the 455-clock has 455 pulses per total horizontal scan line. The 512-clock is used in digital sampling functions and the 455-clock is used for the analog CCD devices. The phase-lock multiplying loop for the 455-clock uses HSYNCD and the clock frequency divided by 455 as its inputs. The PLL functions shown in Figure 3 are a digital phase comparator and a low-pass filter. Two VCOs are used in the loop to increase the frequency range to allow both 525 and 875 line operation. The VCO is selected by LRATE in the 455 mux (LRATE = 0 for 525 and LRATE = 1 for 875). The mux output is divided by two to obtain the 455-clock with a 50 percent duty cycle.

The 512-clock phase-lock multiplying loop is identical to that of the 455-clock except that its clock-dividing section is more complex. The 512-clock drives the horizontal timing generator. The HSYNC output of the generator is the phase detector input for the loop. The horizontal and vertical timing generators with the vertical reset circuitry and combining logic make up a

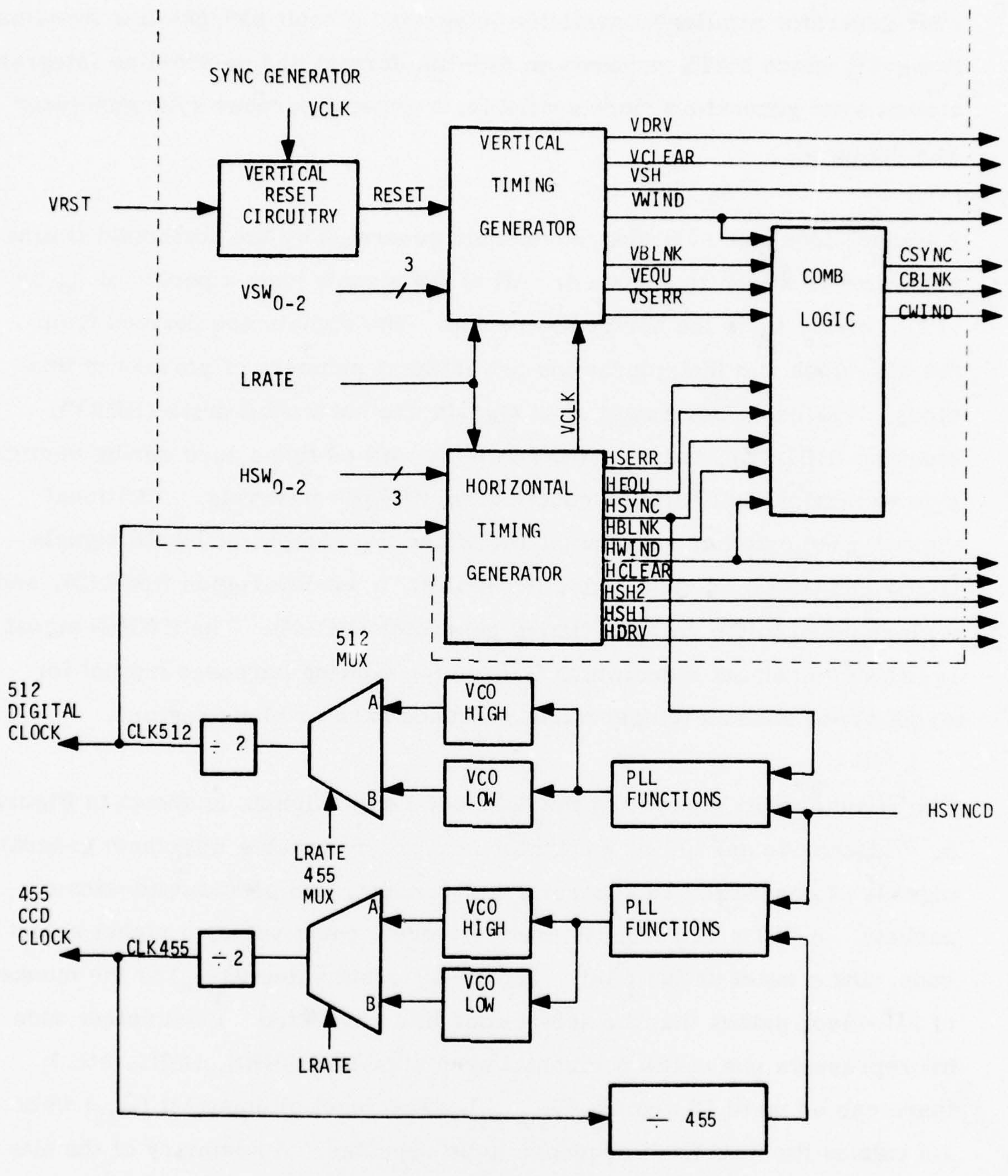

Figure 3. System Timing Generator Block Diagram

sync generator similar to available integrated circuit 525/625 line systems. However, since PATS requires an 875-line format and no 875-line integrated circuit sync generators were available, a general purpose sync generator was designed.

A typical sequence of timing waveforms generated by the horizontal timing generator is shown in Figure 4. All of the signals have a period of t_H or $.5t_H$, where t_H is the horizontal period. The signals are derived from the 512-clock and their durations are integral numbers of periods of this clock. The basic horizontal sync signals are horizontal drive (HDRV), blanking (HBLNK), and sync (HSYNC), as well as those used during vertical sync serration (HSERR) and equalization (HEQU) intervals. Additional signals used by other sections of PATS are two sample-and-hold signals (HSH1 and HSH2), a clear pulse (HCLEAR), a window region (HWIND), and a clock pulse to the vertical timing generator (VCLK). The HWIND signal is used to mask out video which is used for viewing purposes but not for target areas such as temperature reference bars or black regions.

The structure for generating the horizontal sync signals is shown in Figure 5. This horizontal timing generator is a programmable waveform generator capable of generating 16 separate, independent, and periodic waveform packets. A 512 x 24 PROM is used to store a count value, a signal select code, and a reset or jump bit. The 10-bit count value (Q_{0-9}) is the number of 512-clock pulses that the select code bits stay fixed. Each select code bit represents one of the horizontal sync signals (HSERR, HSH1, etc.); there can be up to 13 signals (Q_{10-22}). The reset or jump bit (Q_{23}) when set causes the waveform sequence to be repeated. A summary of the bits and meanings is shown in Table 5. The nine-bit address for the PROM

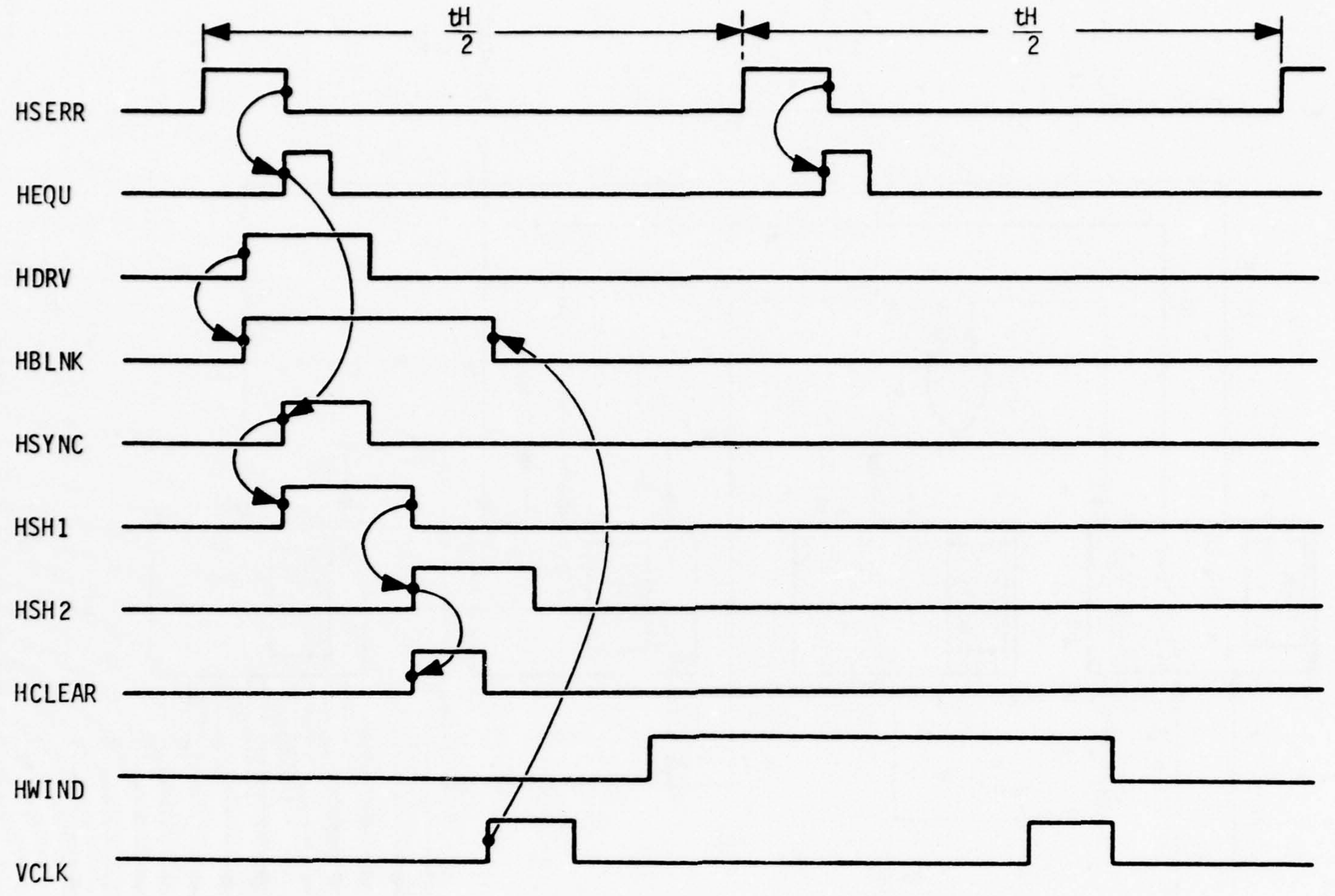

Figure 4. Typical Horizontal Timing Waveforms

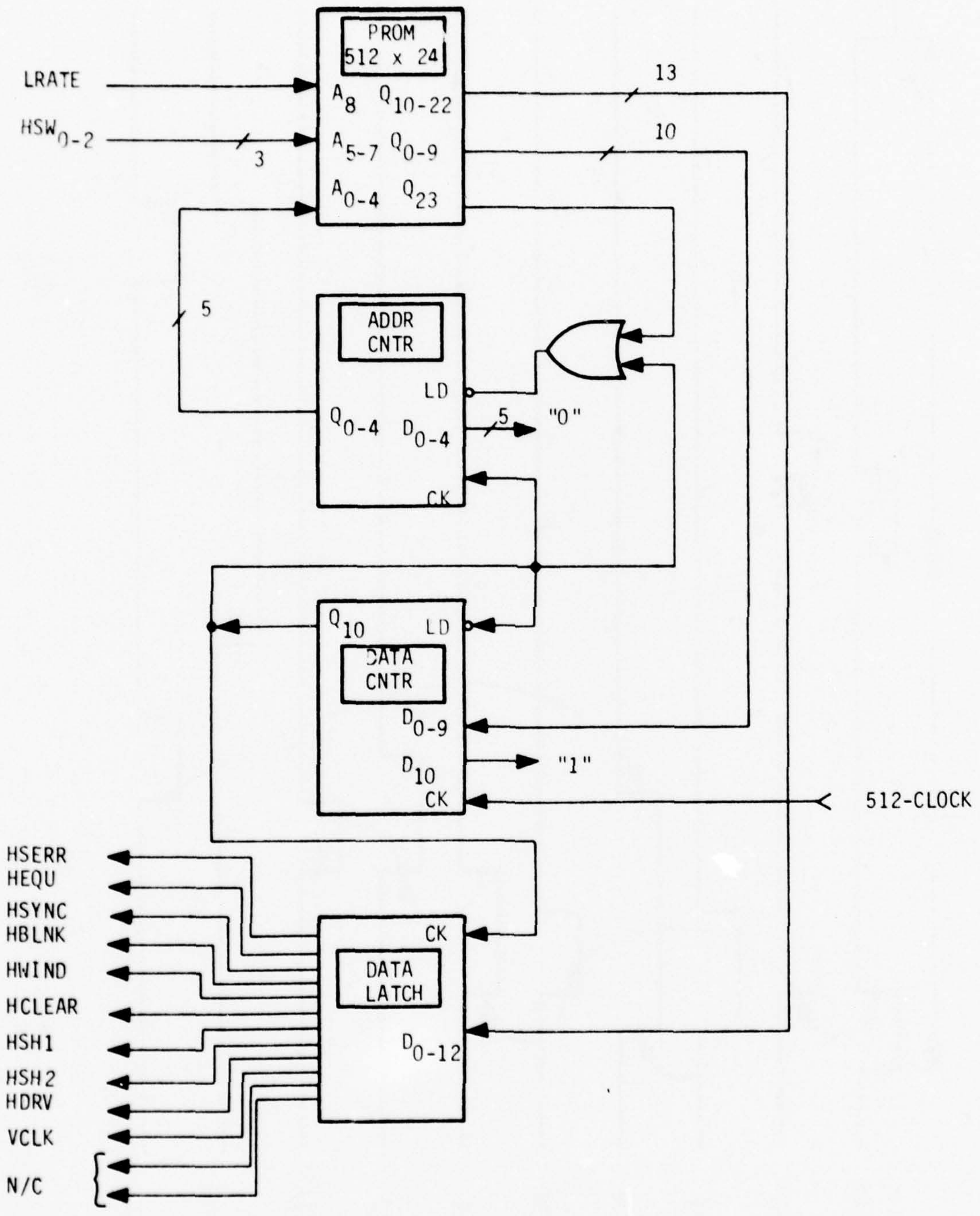

Figure 5. Horizontal Timing Generator

TABLE 5. HORIZONTAL PROM BIT FORMAT

Prom Bit(s)	Definition
Q_{0-9}	10-Bit Counter Value (Negative 2's Complement of Actual Count Value Minus One)
Q_{10}	HSERR ("1" enables signal)
Q_{11}	HEQU
Q_{12}	HDRV
Q_{13}	HBLNK
Q_{14}	HSYNC
Q_{15}	HSH1
Q_{16}	HSH2
Q_{17}	HCLEAR
Q_{18}	HWIND
Q_{19}	HCLK
Q_{20}	Not Used
Q_{21}	Not Used
Q_{22}	Not Used
Q_{23}	JUMP (if "0")

is generated by a five-bit address counter (A_{0-4}), the LRATE signal (A_8), and a three-bit switch HSW_{0-2} (A_{5-7}). The most significant address bits (A_{5-8}) are fixed in normal operation. The address counter is clocked and reset by the jump bit Q_{23} and by Q_{10} of the 11-bit data counter. When Q_{10} of the data counter goes from 0 to 1, the data counter is loaded with the count value (Q_{0-9}) from the PROM; the data latch is loaded with the signal select codes Q_{10-22} and the address counter is incremented if Q_{23} is 1 or loaded with zeros if Q_{23} is 0. Because of the count technique, the count value loaded into the PROM must be the negative 2's complement of the decremented count value. This means that the minimum count value is two.

The VCLK output generated by the horizontal timing generator serves as the clock to the vertical timing generator. This timing generator is identical in structure to that in Figure 5 except that an external reset control is available to allow external synchronization. Typical timing waveforms for the serration, equalization, blanking, and drive are shown as VSERR, VEQU, VBLNK, and VDRV. Three additional signals are added for other PATS functions, these being a clear (VCLEAR), a sample-and-hold (VSH), and a window (VWIND) signal for masking out lines at the top and bottom of the video field. The vertical PROM format for generating these signals is shown in Table 6.

The vertical timing generator must be locked to the video by a reset signal from the sync separator and video switching section; the horizontal timing generator is phase-locked and needs no reset signal. The vertical reset circuitry synchronizes VRST to VCLK and then presets the address counter in the timing generator so that the vertical timing starts with the second serrating pulse in the serration field.

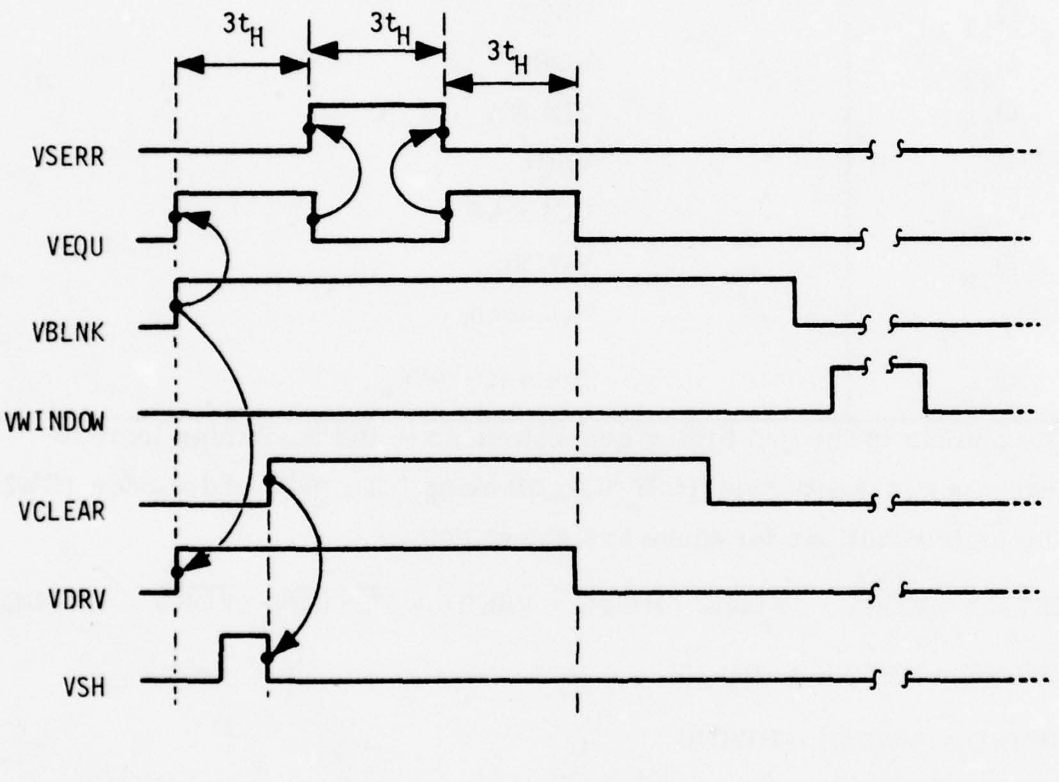

Figure 6. Typical Vertical Timing Waveforms

TABLE 6. VERTICAL PROM BIT FORMAT

Prom Bit(s)	Definition
Q_{0-9}	10-Bit Count Value (Negative 2's Complement of Actual Count Value Minus One)
Q_{10}	VSERR ("1" enables signal)
Q_{11}	VEQU
Q_{12}	VDRV
Q_{13}	VBLNK
Q_{14}	VSH1
Q_{15}	VCLEAR
Q_{16}	VWIND
Q_{17-22}	Not used
Q_{23}	Jump (if "0")

The outputs of the two timing generators go to the combining logic to generate composite sync (CSYNC), blanking (CBLNK), and window (CWIND). The logic equations for these are shown below:

$$\text{CSYNC} = \text{VDRV} \cdot (\text{VEQU} \cdot \text{HEQU} + \text{VSERR} \cdot \overline{\text{HSERR}} + (\overline{\text{VDRV}} \cdot \text{HYSNC})$$

$$\text{CBLNK} = \text{VBLNK} + \text{HBLNK}$$

$$\text{CWIND} = \text{VWIND} \cdot \text{HWIND}$$

The outputs on the system timing generator are all buffered to drive 20 TTL-S loads.

CPU1/CPU2 INTERFACE

Initially, this function was to be primarily a one-way communication from CPU1 to CPU2. The only data to be transferred were target position and classification data. However, during initial checkout we require known data to be stored in all the memories. Because of this a high speed data transfer is necessary in both directions.

A first definition of this interface is shown in Figure 7. CPU1 and CPU2 communicate via DMA transfers between Memory 1 and Memory 2 in CPU1 and the CPU2 (LSI 11/2) memory. Two separate DMA controllers are used, controlled by a handshake. Each controller has registers which store the address of the first word to be transferred and the transfer length.

In order for CPU1 to talk to CPU2, the following must occur:

1. CPU1 loads its DMA registers and interrupts CPU2 to request DMA.
2. CPU2 services the CPU1 interrupt by loading its DMA register and then sending a go-ahead signal to both DMA controllers.
3. The DMA executes and is terminated by the CPU1 DMA controller, sending a signal to the CPU2 DMA which in turn interrupts CPU2.

In order for CPU2 to talk to CPU1, the following must occur:

1. CPU2 loads the register in the CPU1 DMA controller either via single cycle DMA or a parallel I/O mode.
2. CPU2 loads the register in its own DMA controller and then issues the go-ahead to both DMAs.
3. The DMA will terminate automatically.

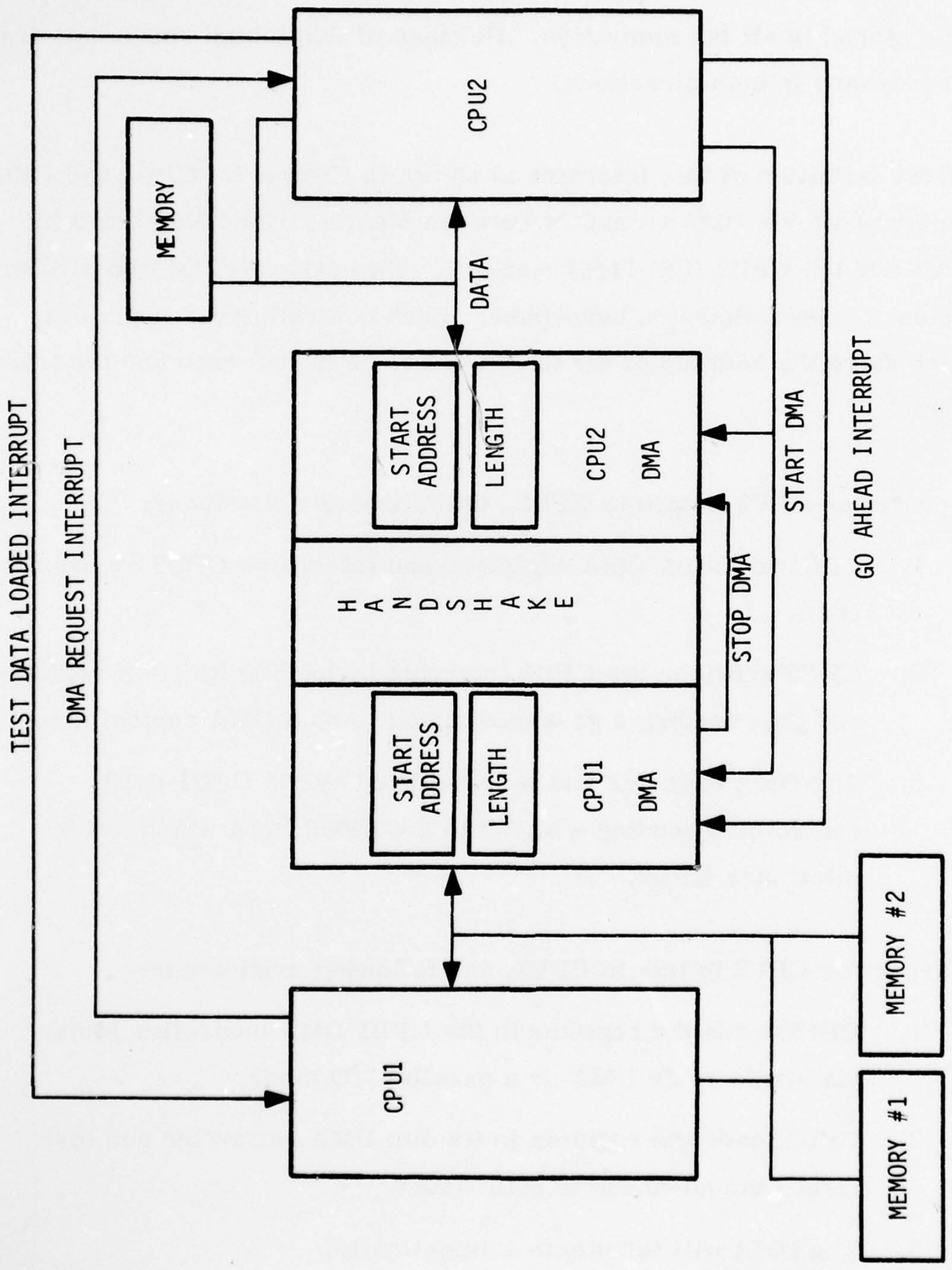

Figure 7. CPU1/CPU2 Interface

Note that while a DMA is in progress, both CPUs are prevented from accessing memory. CPU1 can continue to run as long as it doesn't access memory.

This is the preliminary definition of what must happen in order to have data transfer occur in both directions. A more complete definition will be included in the next report.

WRITABLE CONTROL STORE

In order to check out the microprogram coding, three options are available. The options are:

- EPROM
- PROM
- Writable Control Store (RAM)

The EPROM option would require an additional design since there are few EPROM/PROMS that are directly compatible. Also, the EPROM option only allows non-real time operation of CPU1.

The PROM option will be included in the final operation configuration. However, during initial checkout, several microprogram coding changes can be expected. This could become expensive if a new PROMS were programmed for each change.

The third option gives us real time checkout as well as the capability to dump the microprogram from floppy disk directly onto the RAM in the writable control store.

The requirement for this option is basically an interface between the Intel MDS bus and the memory. The interface to the microprogram memory is a cable which connects into the microprogram memory data and address lines as shown in Figure 8.

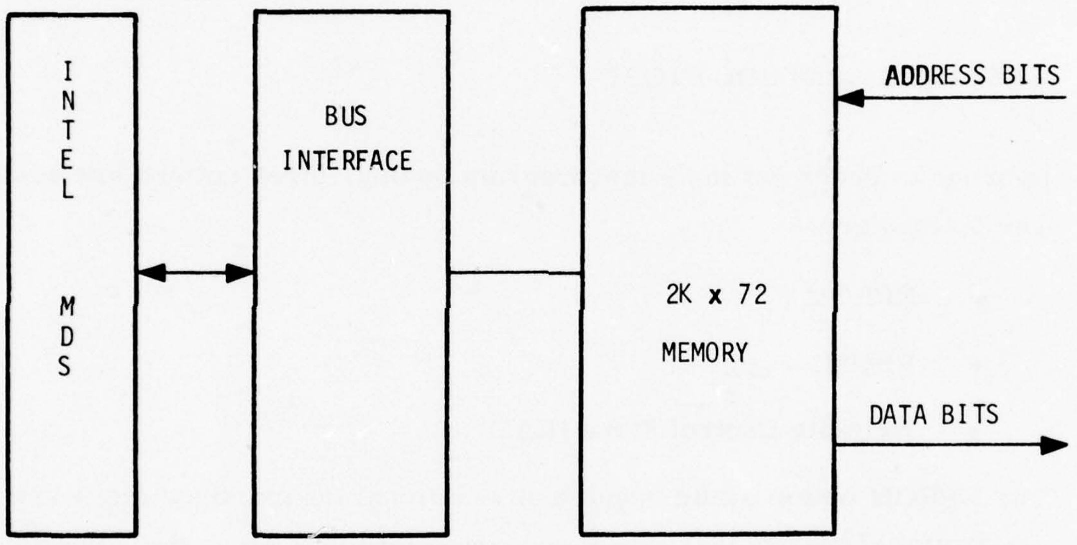

Figure 8. MDS Writable Control Store

SECTION III

EDGE CIRCUIT CHECKOUT

The edge circuit board has been built, wired, and functionally checked. Two potential problems necessitating minor hardware redesign have been noticed. The first is that an edge signal occurs at the beginning of each scan, due to blanking/grey level transition; this must be gated out. The second problem is that the video input to the edge circuitry must stay within prescribed voltage limits or the CCDs will saturate.

Figures 9 through 16 are video pictures taken at various points in the circuit. Figure 9 is the video input from a camera. Figure 10 is the difference signal or the input to the absolute value circuit. White indicates a high rising edge while black indicates a following edge. Figure 11 is the absolute value output.

Figure 12 is the output of the integrator on a line-by-line basis. The data is negative and hence will grow darker as one goes from left to right on the image. Figure 13 is the output of the sample/hold. This remains constant across a scan line. Figure 14 is the logical edge output with high multiplier value for K_E. Figure 15 has a lower multiplier value. Figure 16 shows the edge superimposed upon the video. Note the horizontal delay which will be compensated for in the interval circuit.

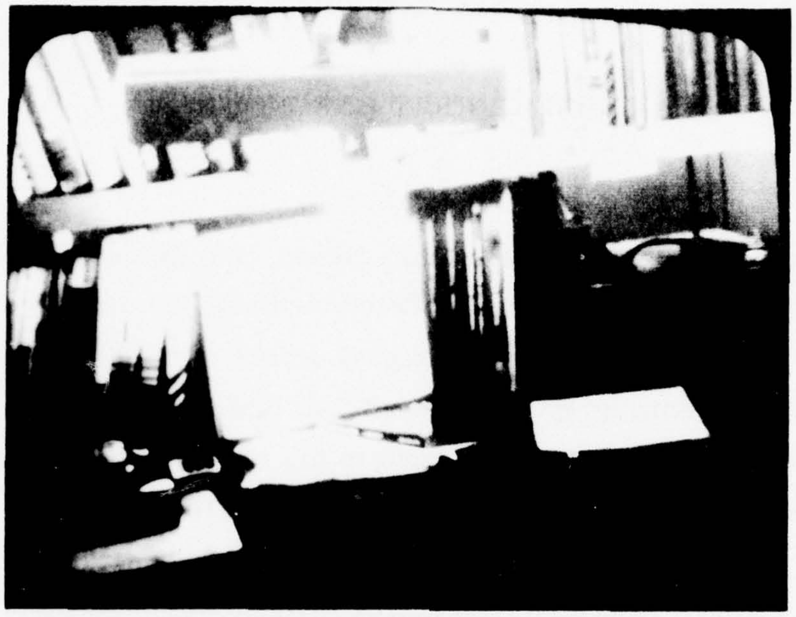

Figure 9. Video Input

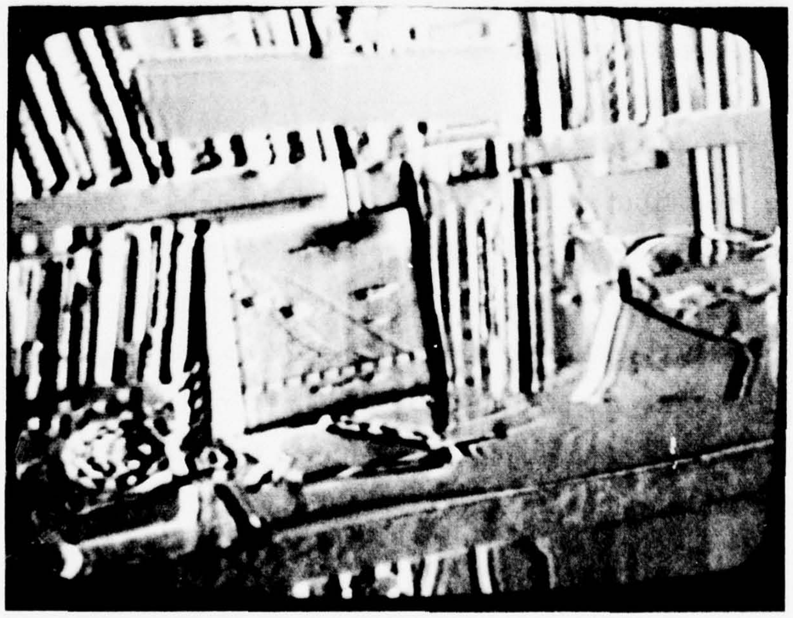

Figure 10. High Pass Filtered Video (Edge)

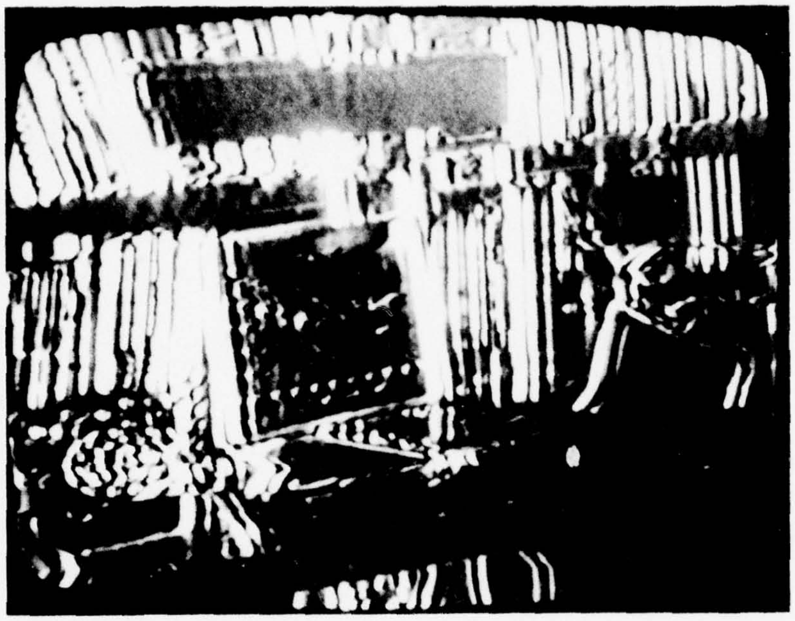

Figure 11. Absolute Value of High Pass Filtered Video

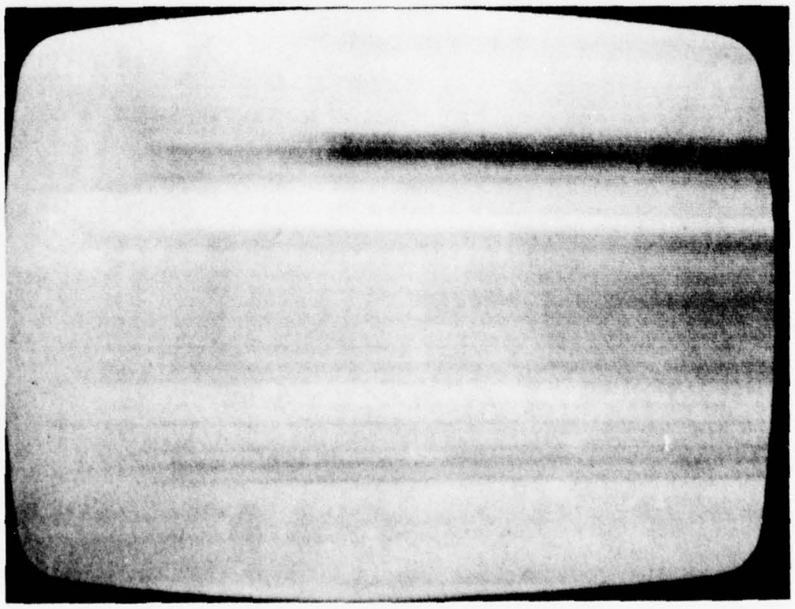

Figure 12. Video of Figure 11 Integrated Across Each Line

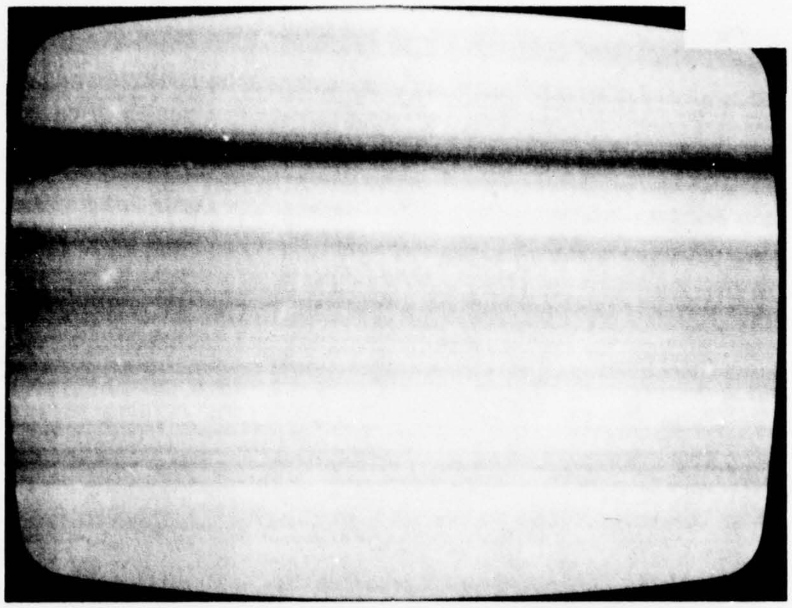

Figure 13. Video of Figure 11 Integrated, Sampled at End of Line, and Held

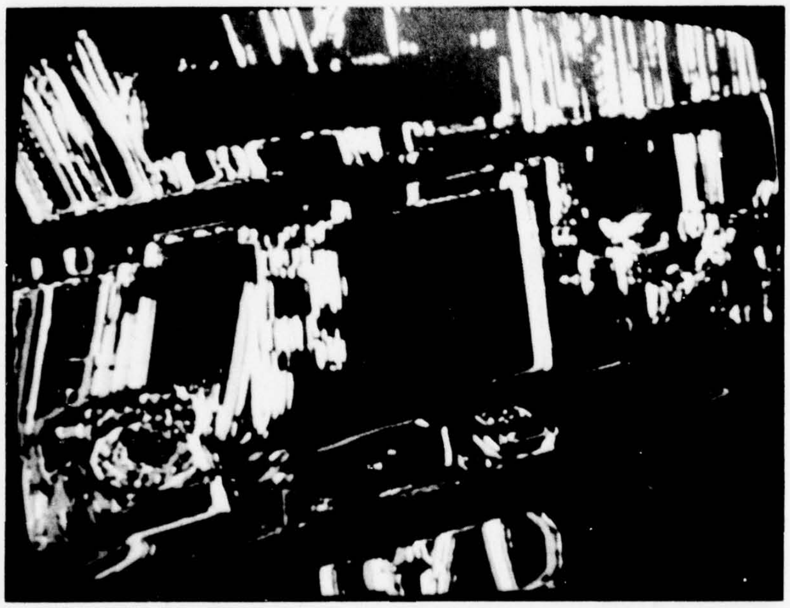

Figure 14. Edge Board Output (High Threshold Settings)

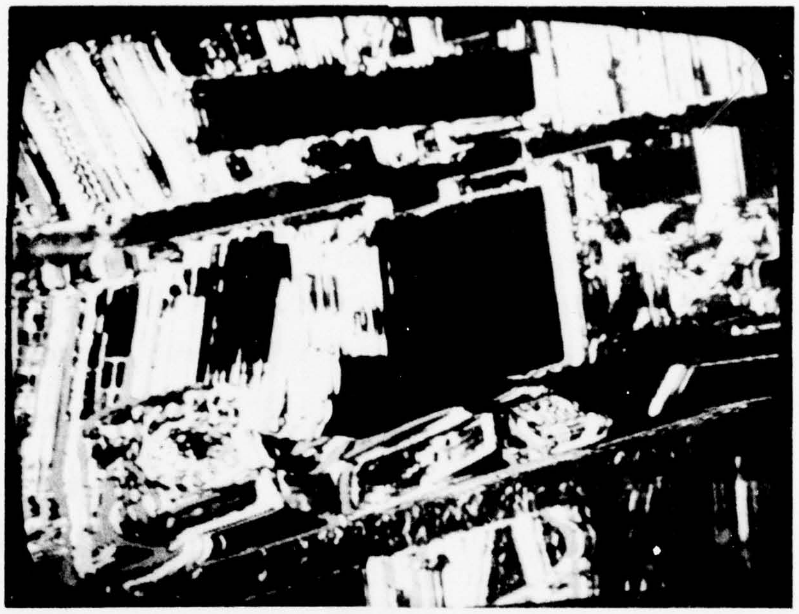

Figure 15. Edge Output (Lowered Threshold Settings)

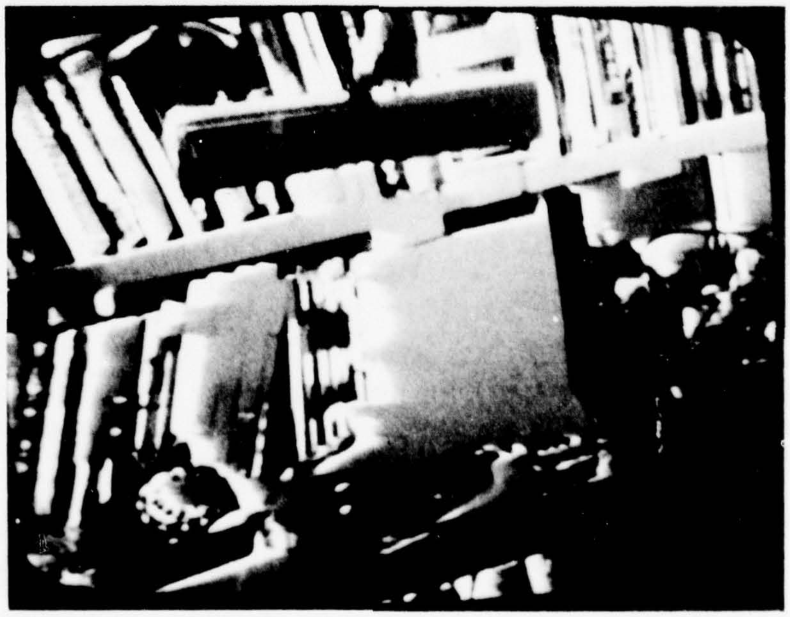

Figure 16. Edge Output Superimposed on Input

SECTION IV

SOFTWARE

In this section, the status of various software functions is discussed. Several diagnostic modules have been added. These are primarily related to the use of CPU2 in checkout of the hardware. A discussion of the algorithm used for bin matching is also included.

CPU1 FIRMWARE

This section will describe the bin matching implementation software, discuss changes made in the CPU1 microinstruction format since the last report, and give worst case performance estimation for the CPU1 modules coded thus far.

The algorithm being used to implement the bin matching function was briefly sketched in a previous quarterly report[1]. The algorithm utilizes all the bin matching criteria embodied in the PATS software simulation,[1,2] but does so more efficiently.

[1] D. E. Soland, et. al., "PATS Quarterly Progress Report," Contract Number DAAK70-72-C-0248, Honeywell Systems and Research Center, Minneapolis, Minnesota, June 15, 1978, pp. 81-87.

[2] D. E. Soland, et. al., "PATS Quarterly Progress Report," Contract Number DAAK70-72-C-0248, Honeywell Systems and Research Center, Minneapolis, Minnesota, January 15, 1978, pp. 78-82.

Bin Matching

Algorithm Description--The purpose of the bin matching algorithm is to match or combine intervals from successive scan lines into two dimensional objects. These objects or "bins" are then passed to feature computation firmware and ultimately classified as either target or clutter.

Intervals are placed into Memory 1 in a packed format by the interval generation and direct memory access hardware. The interval data is dumped at the end of each line in the format shown in Figure 17. Data for a given line is terminated with a zero word, which is overwritten when data is generated for a subsequent line (the significance of this zero word will be discussed below). Data is only dumped to memory for lines which contain intervals. The PATS front end also generates end of line and end of frame interrupts to CPU1. The service routine for the end of line interrupt increments a counter, and the routine for the end of frame interrupt sets a flag indicating that end of frame has occurred.

The bin matching firmware uses a routine called INTERVAL which fetches and unpacks interval data from memory and passes it back to the main body of the algorithm. Whenever INTERVAL encounters a zero word where the next line of data should be starting, it knows that, at least temporarily, it has reached the end of the data in Memory 1. This indication, together with the state of the end of line counter and end of frame flag, allows the routine to set three status flags: EOD (end of data) is set true when INTERVAL has run out of data in Memory 1 and the end of frame flag has been set; DATA is set true whenever INTERVAL successfully finds interval data in Memory 1; and NEWLINE is set true

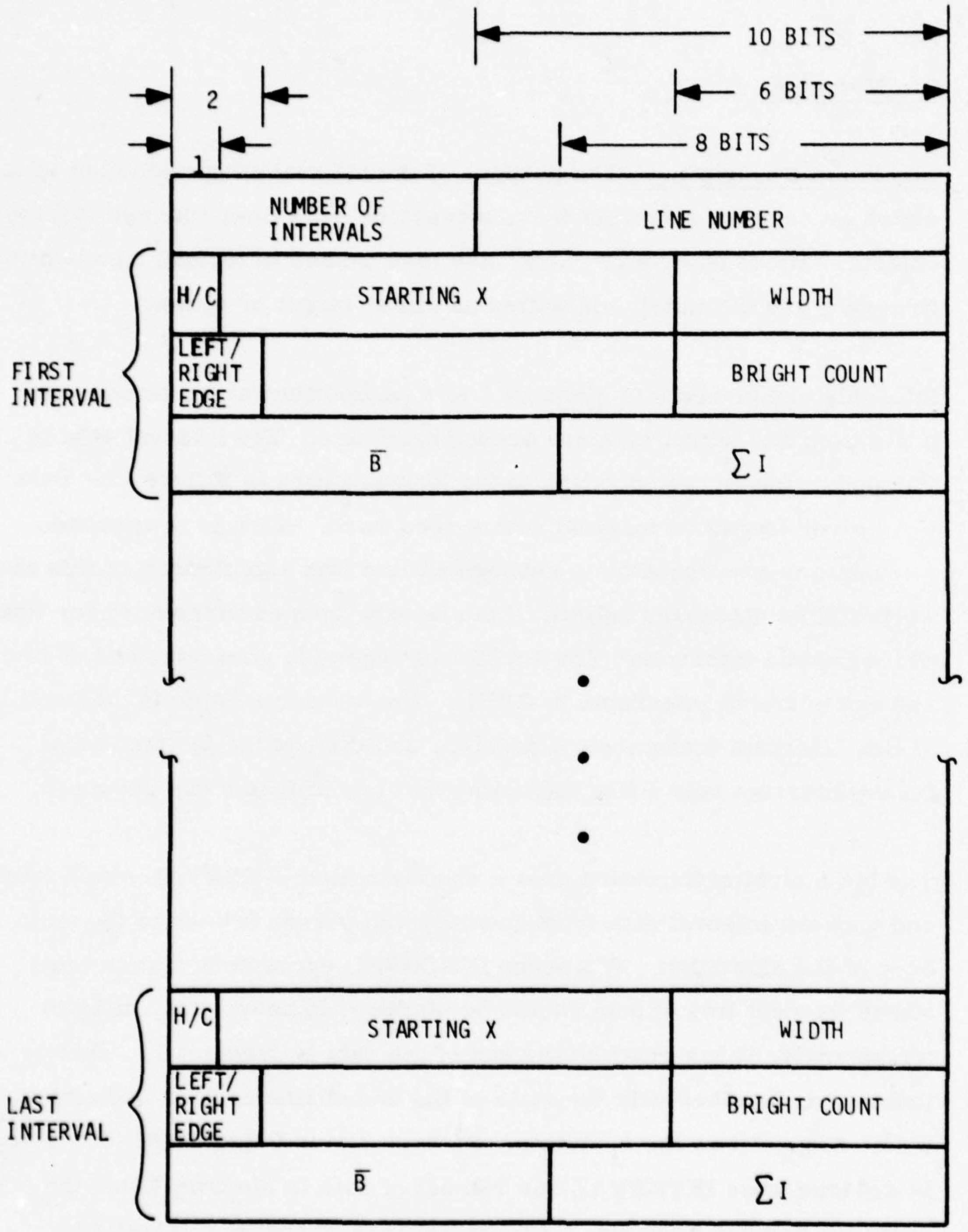

Figure 17. Interval Data Format

whenever INTERVAL encounters the start of a new line, whether or not it has data. The main body of the bin matching algorithm examines these flags prior to processing any of the interval data returned by INTERVAL.

Intervals are either hot or cold and, similarly, each bin produced by bin matching is either hot or cold; i.e., we do not mix intervals of different "colors" in the same bin. The bin matching algorithm is designed to process hot and cold bins and intervals independently of one another. That is, the algorithm produces in one pass through the interval data the same results it would produce in two separate passes, where each pass was only processing intervals of one color and ignoring those of the other. Re-entrant coding of the bin matching algorithm makes this possible. In essence, the algorithm possesses two sets of state variables, one for hot interval processing and the other for cold interval processing. The algorithm accepts interval data from INTERVAL for one interval at a time. When the algorithm encounters an interval opposite in color to the interval previously processed, it saves the algorithm state just prior to the acquisition of the new interval and restores the state corresponding to the color of the new interval. This context switch essentially involves swapping address pointers. The data dependent on each interval color is organized into a contiguous block of storage in Memory 1. There is one block for hots and one for colds and a pointer to each. A context switch is carried out by substituting the working pointer of the algorithm with the pointer to the block for the appropriate color.

Once an interval has been acquired and the appropriate context set, the algorithm attempts to match the interval to a bin. A workspace of approximately 7K words in Memory 1 is divided into continguous blocks of

168 words each. Each bin or object is developed in one of these blocks. Table 7 describes the data block format. A bin's data block is updated each time a new interval is matched to that bin. Each block can accomodate data for up to 32 intervals. These data blocks are stored by bin color (i.e., hot or cold), and active bins of the same color are ordered in increasing value according to the midpoint of the last interval assigned to each bin. This order relation is indicated by linking the bins together into two singly linked lists. There are two list headnodes (one for hots and one for colds) and, as Figure 17 shows, each bin data block has a word for linking it to another bin. Keeping bins of the same color ordered by midpoint systematizes the bin matching process. When matching an interval against all bins of the same color currently in existence, certain bins may be immediately ruled out. For example, if, in going through a bin list, one bin is found to be horizontally to the right of the interval, it is then known that all bins following this bin in the list are also to the right of the interval and cannot match it. Another fact used to make the bin matching process more efficient is that intervals from a given line occur in increasing order according to the interval midpoints. In this way, for example, if an interval is encountered which is to the right of all the bins currently in existence, it is then known that all other intervals from the same line as the latter interval will also not match any of the bins.

Bin matching proceeds in the following manner. An incoming interval of a given color is matched against elements of the bin list corresponding to that same color by starting at some bin and working towards the end of the list by following address links. When data from a new line is encountered, this starting point is initialized with the current headnode

TABLE 7. FORMAT OF BIN DATA BLOCK

Word		
0	Address link to next bin	
1	Midpoint of last interval in bin	
2	Starting address of last interval in bin	
3	Starting line number	
4	Total interval count	
5	Active interval count	
6	H/C (≥ 0: hot; <0: cold)	
7	Intensity sum	
8	X	
9	X + width - 1	
10	Width	Interval 1
11	Feature word 2	
12	Feature word 3	
13	X	
14	X + width - 1	
15	Width	Interval 2
16	Feature word 2	
17	Feature word 3	
.	.	
.	.	
.	.	
164	X	
165	X + width - 1	
166	Width	Interval 32
167	Feature word 2	
168	Feature word 3	

on the list. To start off, the algorithm checks if the bin list being examined is empty or if the end of the list has been encountered (these conditions are equivalent since both involve accessing zero or null pointers). If either of these conditions is true, a new bin can be started. Otherwise, the interval is compared with the bin at the current starting point. If the left endpoint of the interval is greater than the right endpoint of the last interval currently in the bin, then it is known that the interval cannot match this bin but it might match one farther down the list. In this case, the algorithm links to the next bin in the list and repeats the whole process, starting with the check for end of list. If the left endpoint of the interval is less than or equal to the right endpoint of the bin, then we have a case of possible overlap and another comparison is made. This time, if the right endpoint of the interval is less than the left endpoint of the bin, it is then known that the interval lies entirely to the left of the bin currently being examined; since the interval also did not match the bin just prior to the current one, the interval must be between the two bins, so a new bin may be started. If, however, the right interval endpoint is greater than or equal to the left bin endpoint, then interval and bin overlap and bin matching criteria can be checked. The first criterion is midpoint correspondence. If the midpoint of the interval falls within the bin or the midpoint of the bin falls within the interval, a match is achieved. If the first criterion is not satisfied, then the second criterion, intensity correspondence, is checked. If:

$$\bar{I}_b \geq 10 \cdot |\bar{I}_b - \bar{I}_i|$$

where $\bar{I}_b$ = average intensity of last interval in bin

$\bar{I}_i$ = average intensity of interval, then a match is achieved. If this criterion is not satisfied, then the algorithm links to the next bin and restarts the whole process with the end of list check.

If a bin match is achieved, then the algorithm applies the same bin matching process to subsequent intervals, trying to match them to the same bin at which the first bin match occurred. As long as intervals match this bin, they are concatenated together to form a single larger interval. When an interval is finally encountered which does not match this bin, then the (possibly compound) interval which did match it is used to update the bin. The algorithm then links to the next bin in the list and restarts the bin matching process with the new interval.

When a bin is completed (i.e., the missed scans criterion has been satisfied), the origin of the bin is passed to the clutter classifier. If the bin is classified as clutter, it is removed from the list of active bins it was a part of and is instead linked into a list of bins which can be reused. The bin is processed no further. If the bin is classified as target, it is also removed from its active bin list and is linked into the list of bins which will be passed to the recognition classifier. In this way, all bins in a frame are classified as clutter or target before any are recognized as being a particular type of target. Doing classification of bins in parallel with bin matching relieves the bin memory requirements since some bins are being thrown away at the same time others are being started. Only partial classification of bins is carried out at this time since the recognition classifier requires access to Memory 2. Memory 2 is totally unavailable to CPU1 while the frame is being digitized. So, to make maximum use of the processing time available to CPU1 during this digitization period, it is best to use CPU1 to do things which do not involve accesses to Memory 2, i.e., nonrecognition functions.

When the algorithm determines that a new bin must be started, it first checks the list of reusable bins mentioned in the previous paragraph. This list is, of course, initially empty. If this list is empty, the algorithm then references a pointer to the first location in memory which has not yet been used for bin storage.

Flow Chart

Figure 18 is a flow chart of the various operations discussed above. The diagram is not extremely detailed but suffices to indicate the flow of control between the major operations. A few words about the notation are in order. INTVL, IV, and IVNXT are all vectors of interval features. INTVL is fetched by the routine INTERVAL which was described previously. I1 and I2 are, respectively, left and right endpoints of the interval currently being examined; J1 and J2 are, respectively, left and right endpoints of the current bin. The logical flags EOD, NEWLINE, and DATA are all set by the routine INTERVAL.

Bin Matching Summary

In short, the bin matching implementation attempts to maximize efficiency by minimizing the expected number of comparisons which must be made against a given interval. Bins are organized into two separate lists, and an interval is never compared against the elements of more than one list. In addition, within each list bins are ordered according to the way they

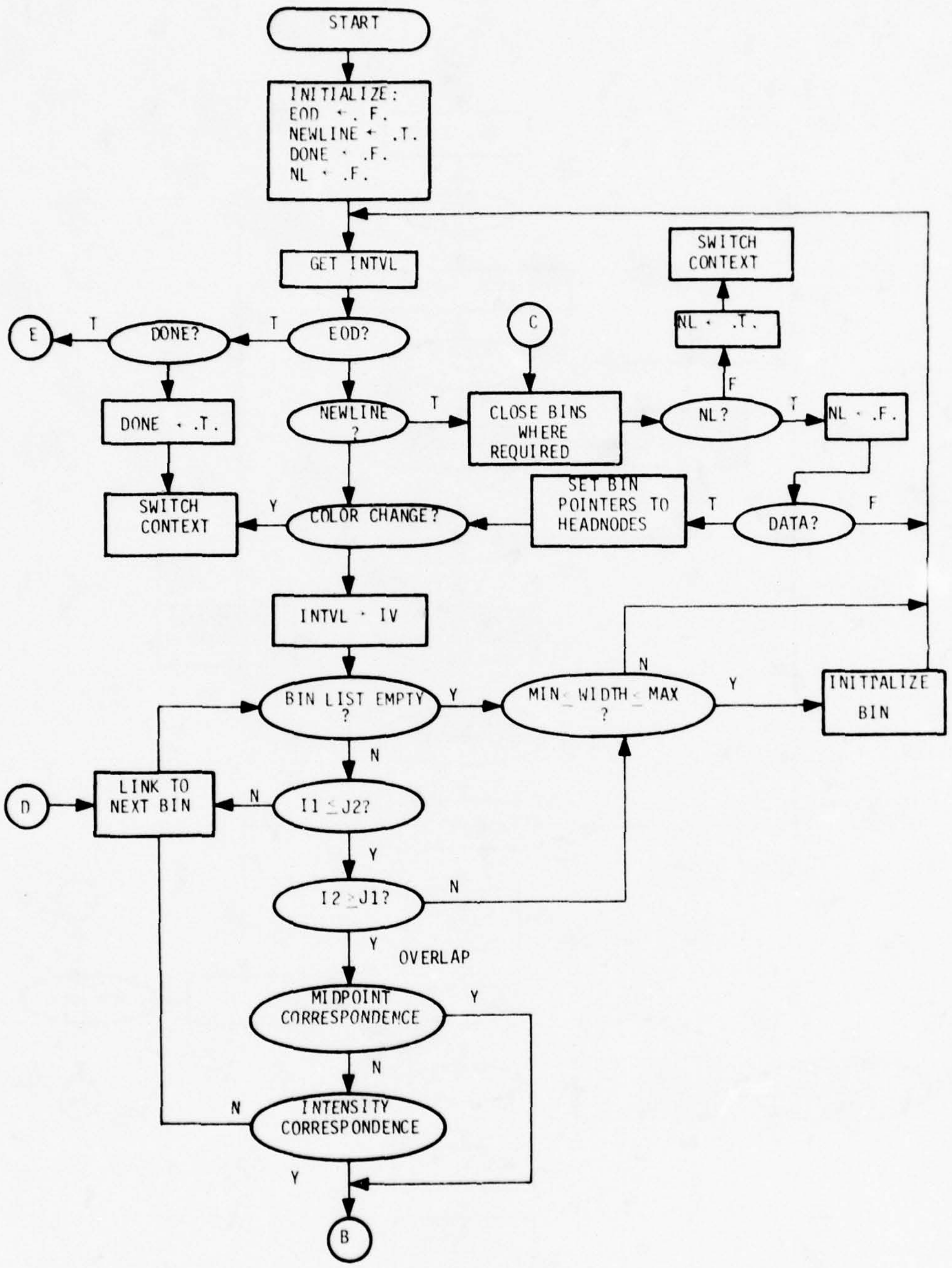

Figure 18. Flow Chart of Bin Matching Algorithm

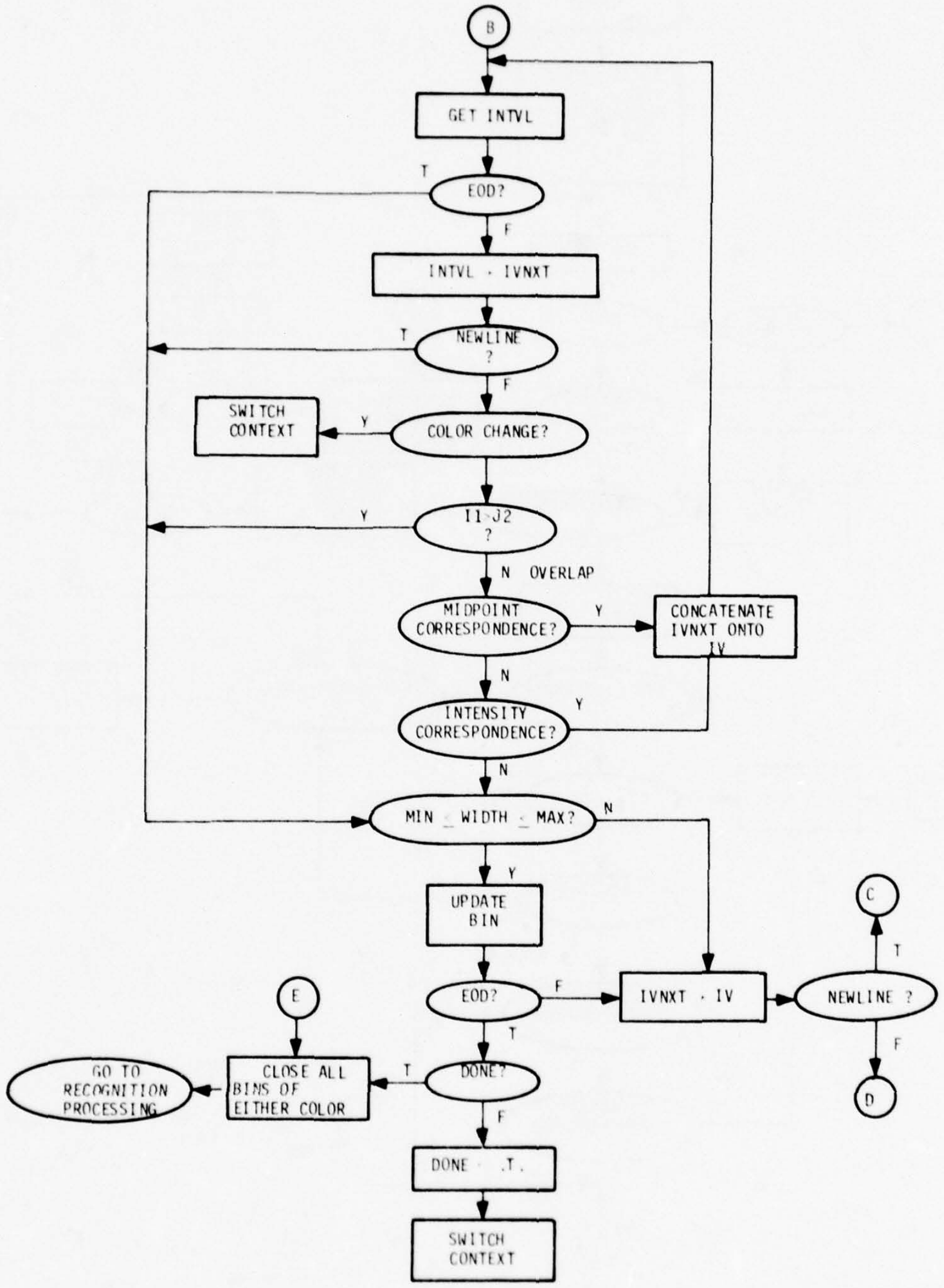

Figure 18. Flow Chart of Bin Matching Algorithm (concluded)

line up horizontally across the frame. The algorithm makes use of this by establishing a simpler necessary condition, namely bin/interval overlap, prior to doing a full-blown test of the bin matching criteria. This results in the latter, rather expensive test being done only when necessary, instead of on every bin.

MICROINSTRUCTION FORMAT

Since the last quarterly report, the CPU1 microinstruction has grown to 69 bits. The format of the revised microinstruction is shown in Figure 19. Changes to the format published in the previous quarterly report are described below.

A bit was added to control a two-to-one 16-bit multiplexer whose output is connected to the DA input on the Am2903 array. This bit selects either the literal field from the microinstruction or the Y output from the Am2901 array as the DA input. This change allows data transfer from the 2901s to the 2903s; previously, we could only transfer data from the 2903s to the 2901s. This bit occupies bit 61 in the new format, and its default value is 0 (i.e., select literals field).

Another bit was added to control status register swapping. An additional status register was added to the 2903s for the purpose of saving the contents of the primary status register when, for example, an interrupt is being serviced and one wants to preserve the status in the main routine. This bit occupies bit 57 and its default value is 0 (i.e., no swap).

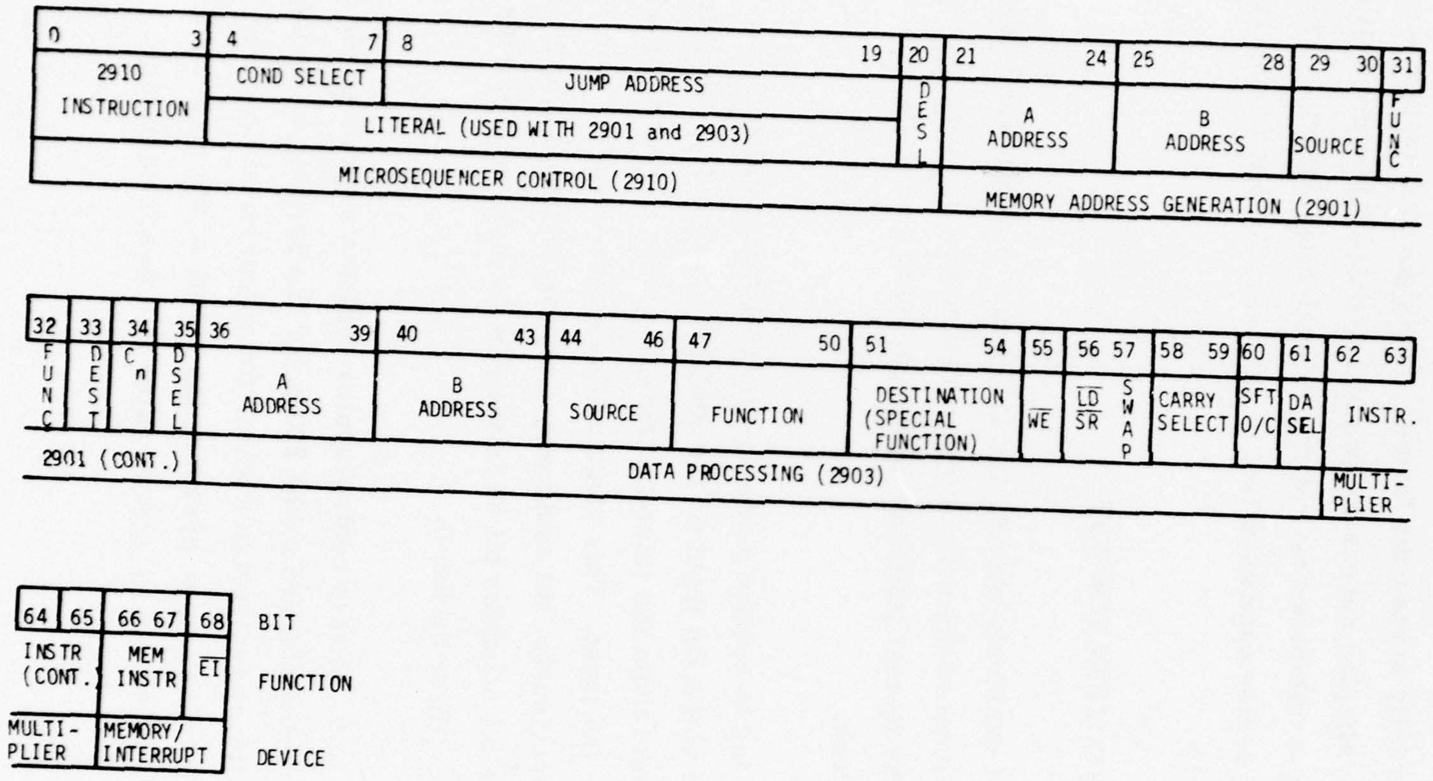

Figure 19. Modified CPU1 Microinstruction Format (Programming Model)

A third bit (bit 69) was added to control interrupt enabling/disabling. The default for this bit is 0 (i.e., enable all interrupts). CPU1 will accept eight discrete interrupts, each having a different priority. There is no interrupt mask (i.e., no selective interrupt disabling).

Other bits in the microinstruction word were rearranged to make room for these additional bits, as is shown in Figure 19. An additional change was made to the memory control bits. These two bits (bits 66 and 67 in the new format) were encoded to provide an additional mode of control when reading Memory 1.

EXECUTION TIME ESTIMATES

Execution time estimates for the various CPU1 firmware modules appeared in a previous report.[3] These estimates were made prior to the generation of any detailed microcode. Revised worst case estimates have now been made for the clutter classifier, median filter, and moment feature computations based on generated microcode. The estimates assume a total of fifty 32 x 32 objects passed to the clutter classifier, 10 of which are ultimately passed to the median filter and moment feature computations. The execution times are as follows:

[3] D. E. Soland, et. al., "PATS Quarterly Progress Report," Contract Number DAAK70-72-C0248, Honeywell Systems and Research Center, Minneapolis, Minnesota, June 15, 1978, p. 91.

Clutter Classifier	- 25.7KμI x 0.2μs/μI =	5.14 ms
Median Filter	- 36.5KμI x 0.2μs/μI =	7.30 ms
Moment Features	- 61.4KμI x 0.2μs/μI =	12.28 ms
		24.72 ms

These new estimates compare favorably with the old ones. New estimates have not yet been made for bin matching and the K -nearest neightbor recognition classifier but we believe the final estimates will be favorable, since the firmware currently generated only uses roughly 25 percent of the 100 ms available per processed frame.

CPU2 SOFTWARE

The initial design is complete for CPU2 software. A convenient way to divide the software for discussion purposes is to split it into two categories: diagnostics and non-diagnostics. The diagnostic software includes facilities for memory tests, for CPU interface tests, for display tests, and for exercising the system with test data. Non-diagnostic software includes interframe analysis, cueing, a supervisor that controls processing, a routine that controls the system during training, and assorted support subroutines. Some routines are used by both diagnostic and non-diagnostic procedures.

Table 8 lists the routines used primarily in diagnostics. Table 9 lists the other routines. Figure 20 shows how they all relate to one another. Appendix A describes the function of each routine.

Notice there are two memory tests, two routines for testing memory. MEMTEST is a far more elaborate test than CHEKMEM. MEMTEST will

TABLE 8. CPU2 SOFTWARE USED PRIMARILY FOR DIAGNOSTICS

CHEKMEM
DISPTST
MEMTEST
PROTOCL
SIMULAT

TABLE 9. OTHER CPU2 SOFTWARE

General	Interframe analysis	Cueing
CPU1INPUT	COSTIT	APC
DMACODE	ELECT	CUEIT
DMAINVL	GETUM	ERASE
DMARES1	HUH	NONTARG
DMARES2	INTER	NOTHING
DMAVEC1	PAIRUM	TANK
DAMVEC2	PICK	TRUCK
INITIAL		UIT
ISR1		
ISR2		
OPINPUT		
SUPER		
TRAIN		

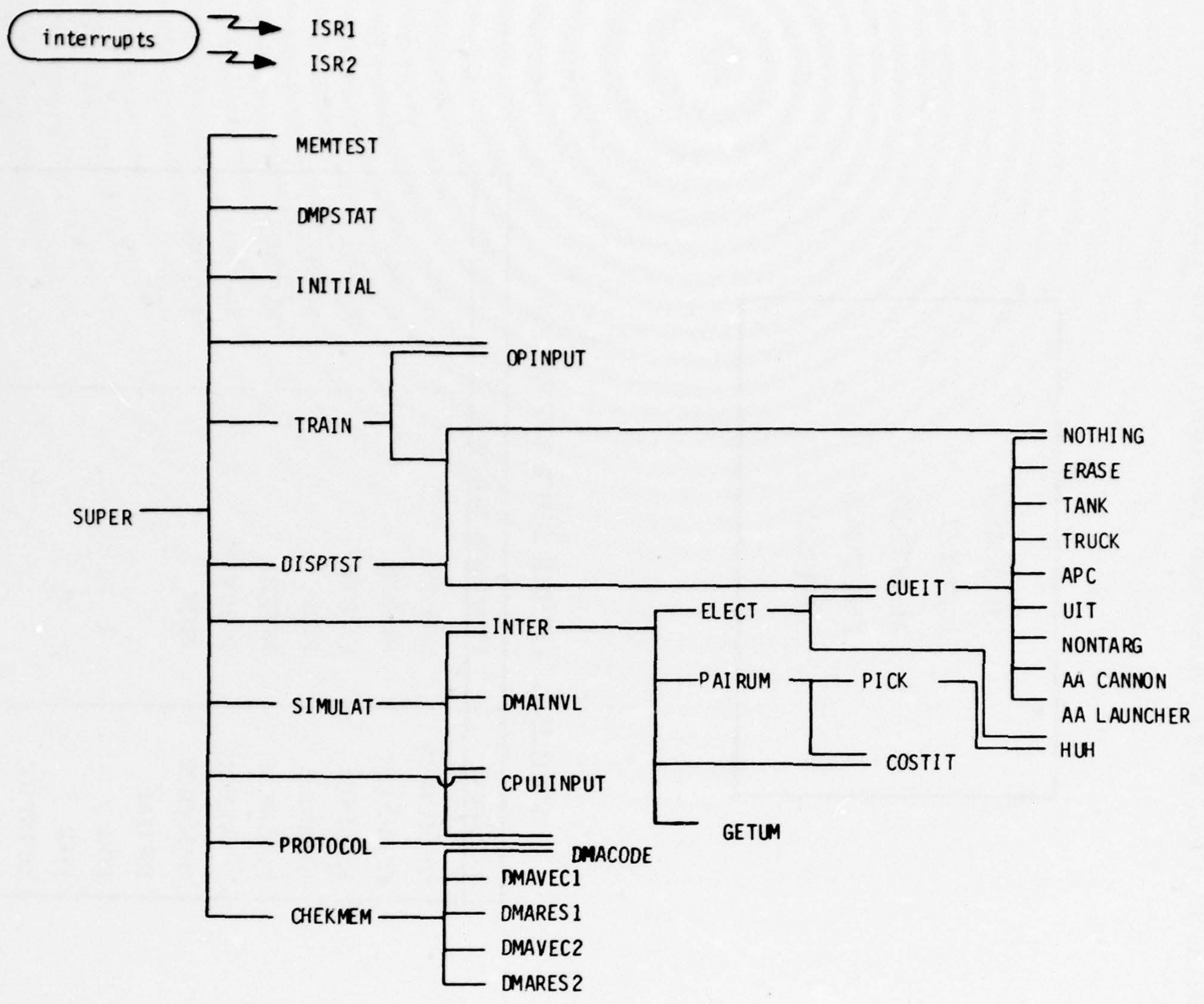

Figure 20. CPU2 Hierarchy of Software Modules

be used to help find problems while Memories 1 and 2 are being built and debugged. After the memories seem error free and are being used frequently MEMTEST will not be used very often. CHEKMEM is intended for use as a quick check during the software system build. If the software suddenly produces strange results the programmer will be able to use CHEKMEM as a debugging tool to assure himself there are no gross memory problems.

The simulation of real time execution using SIMULAT will allow us to read frames of imagery and interval data from floppy disk. These data will be written in Memories 1 and 2 for processing by CPU1. When the object features are passed to CPU2 the objects can be processed through inter-frame analysis and cued. Intermediate results will be read from floppy disk and compared with those passed to CPU2. Differences will then cause error reports to aid in debugging.

The interframe analysis routines increase the reliability of object classification across several frames. Currently the number of frames used for decision making is one frame while in the training mode. The software will be coded so the number of frames can be set as a parameter. The default will be three. When the number of frames is one, interframe analysis is effectively turned off.

The basic operation in CPU2 is one of waiting for something to happen. In non-real time mode the supervisory routine SUPER asks the operator what he wants to do. Given an input SUPER begins the processing as requested. In training, simulation, or real time mode SUPER is in a loop checking flags. The flags are set by the interrupt service routines ISR1 and ISR2 when CPU1 sends data to CPU2. The flags set depend upon the kind of data passed. Error messages are typed out immediately. Intermediate results are typed and/or compared with expected values by SIMULAT. Object features are passed to INTER for interframe analysis. Then objects to be cued are used by CUEIT. In training mode TRAIN interacts with the operator to classify an object. Then the object features and classification are stored on floppy disk.

SECTION V

PLANS FOR THE NEXT REPORTING PERIOD

During the current reporting period, we planned to complete the fabrication of all circuit boards and begin checkout at the board level. However, because of some design problems with CPU1 and delays in part deliveries, the circuit fabrication task will extend through the first half of the next reporting period. Checkout of the boards completed to date has begun and will continue through the next reporting period. Also, software coding and checkout for both CPU1 and CPU2 will continue through the quarter.

APPENDIX A

CPU2 SOFTWARE MODULE DESCRIPTIONS

PRECEDING PAGE BLANK

APPENDIX A

CPU2 SOFTWARE MODULE DESCRIPTIONS

APC:	cues armored personnel carriers by superimposing the letter A over the object. The size of the letter is proportional to the size of the object.
CHEKMEM:	performs a simple test of Memories 1 and 2. The test uses CPU2 to write the address of each word in the word, then reads and verifies the value. Then CPU1 is instructed to read the words and verify the values. Finally CPU1 writes zeroes to all words and CPU2 reads and verifies them. So each CPU writes and reads what the other one has written.
COSTIT:	used in interframe analysis, computes the cost of claiming that an object in frame N is the same as an object in frame N-1. The costs are computed for all possible pairings of objects in the two frames. The cost is the sum of the absolute differences of the object features. So the most similar objects have the lowest cost.
CPU1INPUT:	decides what kind of data has been passed to CPU2 from CPU1 and then decides what to do with the data. The first word passed to CPU2 contains a code for the kind of data sent: 0 = object features, 1 = debugging information like intermediate results, 2 = an error code. Error codes cause an error message to be printed immediately.

	Intermediate results are printed and/or compared with expected values for validity. Object features are kept for use by interframe analysis.
CUEIT:	controls the writing and erasing of symbols in the graphics memory. It does not do the writing; it merely sets up the arguments and calls the proper symbol generation routine.
DISPTST:	performs a test of the graphics system. All vertical and horizontal lines are displayed one at a time for the operator to verify. Then all symbols generation subroutines are used to produce their symbols for verification.
DMACODE:	passes a control word to CPU1 from CPU2. The control word tells CPU1 what procedure is to be followed. The word to be sent is passed in to DMACODE. Then DMACODE concerns itself with all the interface protocol needed to send the message and verify that it has been received.
DMAINVL:	sends interval data to Memory 1 during simulation with test data. The interval data are read from floppy disk by SIMULAT and passed to DMAINVL. This routine then concentrates on the interface protocol necessary to send the data to Memory 1 and verify that it has been received.
DMARES1:	used during testing of Memory 1, reads Memory 1 beginning at a specified location and stores the values in vector RESULTS for verification later. DMARES1 concentrates on interface protocol needed to read the data from Memory 1.

DMARES2:	the same as DMARES1 except that Memory 2 is the memory tested.
DMAVEC1:	used during testing of Memory 1, writes a vector of data from CPU2's memory into Memory 1 via the DMA interface. The routine receives the data from the calling routine so DMAVEC1 concentrates on passing the data by following the interface protocol and verifying that the transfer has occurred.
DMAVEC2:	same as DMAVEC1 except that Memory 2 is the memory under test.
DMPSTAT:	used at the end of a training session, statistics collected for all objects detected in all frames during a training session are saved in arrays. At the end of the training session the statistics can be printed by entering execution code 6 when the system asks for a code.
ELECT:	used in interframe analysis to decide what classification should be assigned to an object, calls the cueing routine to cue the object. ELECT counts the frames in which the object occurs and the classification assigned to the object in each frame. The classification assigned is the one which occurs most often in the frames queried.
ERASE:	used in the cueing procedure to find cued targets whose cues can be erased. The lifetime of a cue is defined in terms of the number of frames which will be displayed while the cue is retained. When the number of frames exceeds the defined lifetime ERASE removes the cue.

GETUM:	used in interframe analysis, reads the object features passed to CPU2 from CPU1. The features were stored in a temporary buffer when CPU1 interrupted CPU2. GETUM copies the features to a permanent buffer for evaluation by interframe analysis.
HUH:	used in interframe analysis routines to compute frame identifiers for use as subscripts in other subroutines. The identifiers are computed relative to the current frame identifier.
INITIAL:	assigns initial values to variables and vectors.
INTER:	controls interframe analysis.
ISR1:	the interrupt service routine entered when the interrupt means a DMA has been completed.
ISR2:	the interrupt service routine entered when CPU1 wants to transfer data to CPU2.
MEMTEST:	tests Memories 1 and 2 using a moving inversions test. This procedure inverts the data of each address sequentially, creating an access time by the jump from one address to another which contains different information. Read/write/read operations are performed with both forward and backward address sequences.*

*The procedure has been reported by J. Henk De Jonge and Andre' J. Smulders in "Moving Inversions Test Pattern is Thorough, Yet Speedy," Computer Design, Vol. 15, No. 5, May 1976, pp. 169-173.

NONTARG:	used in the training mode. An object detected by CPU1 is initially displayed and cued with the letter U meaning unidentified target. After the operator decides what the object is and puts in the class code the object is cued with the appropriate symbol. If the operator decides that the object is not a target, subroutine NONTARG is called to cue the object with the letter O. The operator then knows the proper class code has been assigned to the object.
NOTHING:	used by the graphics system to draw a big letter X across the whole screen when, in the training mode, CPU1 has found no targets. The operator then knows processing on that frame is complete and no targets have been found.
OPINPUT:	used by the supervisory routine to interact with the operator. Queries or directives are printed and input is accepted by OPINPUT. The input is verified as at least being valid, if not correct, before the input is passed to other routines for use.
PAIRUM:	used in the interframe analysis procedure. Subroutine PICK links objects in frame N with objects in frame N-1. If there are objects in frame N which cannot be matched with objects in frame N-1, subroutine PAIRUM tries to match those objects with unmatched ones in frame N-2. So PAIRUM links objects in frame N with objects in frame N-2.
PICK:	used in interframe analysis to decide which objects in frame N should be matched with which objects in frame N-1. The pointers that link objects across frames are established by PICK.

PROTOCOL:	used in the debugging mode to test the CPU interface by exercising the protocol. Standard messages are passed back and forth between CPU1 and CPU2 many times.
SIMULAT:	controls system simulation during checkout with test data input from floppy disk.
SUPER:	the main program which controls the whole system. SUPER contains the idle loop where control resides until there is a command from the operator to do something or until CPU1 sends an interrupt. Flags are checked in this loop to decide what should be done in response to inputs from these sources.
TANK:	cues tanks by superimposing the letter T over the object. The size of the letter is proportional to the size of the object.
TRAIN:	controls the system during training. The main functions are to cue each object passed into CPU2 from CPU1 so the operator can identify them and input the classification code. Then the object features are output to floppy disk along with the classification code.
TRUCK:	cues trucks by superimposing the letter W over the object. The size of the letter is proportional to the size of the object.
UIT:	cues unidentified targets by superimposing the letter U over the object. The size of the letter is proportional to the size of the object.

AA CANNON:	cues track-mounted radar controlled anti-aircraft cannon by superimposing the letter C over the object.
AA LAUNCHER:	cues track-mounted anti-aircraft missile launcher by superimposing the letter M over the object.