

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8558	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE DESIGN OF A REAL-TIME SIGNAL SORTER		5. TYPE OF REPORT & PERIOD COVERED Final report on one phase of an NRL problem
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Timothy R. Husson and Robert H. Evans		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62734N; WF34-372-091; 57-0534-0-2
11. CONTROLLING OFFICE NAME AND ADDRESS Pacific Missile Test Center Point Mugu, CA 93042		12. REPORT DATE February 1, 1982
		13. NUMBER OF PAGES 33
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Air Systems Command Arlington, VA 20360		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		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) Signal processing Signal sorting Microprogramming Associative memory		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This is a final report on a project to design, build, and test a model of a high-speed signal sorting system. The digital inputs to this system would be radar receiver data at rates up to one million input pulses per second. The signal sorter must identify and sort the different signals in the data system. This report describes the software simulation which was created and used to design the signal-sorter architecture and algorithms. Also described is the hardware prototype model which was built and successfully tested.		

DD FORM 1473
1 JAN 73EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

CONTENTS

INTRODUCTION	1
SOFTWARE SIMULATION	2
SIGNAL-SORTER ARCHITECTURE	2
Interrogation in the CAM	3
Microprocessor Array	3
List-Forming Processor	5
CAM Load Processor	6
EMITTER PROCESSING ALGORITHM	6
SIMULATION RESULTS	11
COMPARISON OF ARCHITECTURES	14
SIGNAL-SORTER PROTOTYPE HARDWARE	18
Microprocessors	19
Content-Addressable Memory (CAM)	22
List ..	23
CAM Controller	24
LABORATORY TESTING	24
Test Configuration	24
Test Results	27
SUMMARY ..	28
REFERENCES	29
APPENDIX — Glossary of Terms	30

THE DESIGN OF A REAL-TIME SIGNAL SORTER

INTRODUCTION

The purpose of signal sorting is to process data from a radar receiver at high rates and identify various emitters as the sources of pulses. This problem is very acute in a threat-engagement scenario characterized by a large amount of electromagnetic activity and the need to respond to threats in real time. Current front-end processing schemes used in real-time systems cannot process receiver outputs with more than a million pulses per second, which could occur in such a scenario, on a pulse-by-pulse basis. This research focused on the design of a system that could handle these high pulse densities.

The processing strategy that has been developed as a result of this work is to filter out quickly the pulses of those emitters which have already been identified, thus allowing more processing time for the unidentified emitters. This filtering is done in a content-addressable memory (CAM),* as shown in Fig. 1. The data on the emitters that have been identified are stored in the CAM. These data are compared to the data from the input pulses. The matching in the CAM is done on the emitter parameters detected by the receiver, namely, direction of arrival (DOA) and carrier frequency (FREQ). Pulse width is also generally determined by the receiver. However, uncertainties in actually detecting and accurately measuring the edges of the pulse render this parameter less useful for sorting. Therefore, pulse width is not used in this matching scheme. Since the size of the CAM is limited and the number of active emitters is not known beforehand, the CAM is loaded with the parameters of the emitter's pulses which are next expected to arrive at the receiver. It is possible to load the CAM with the next expected pulses if the last pulse arrival time and the pulse repetition interval (PRI) are known. The emitter pulses which are matched in the CAM need no further processing except to compute the expected next arrival time, which is a simple addition. This assumes a stable PRI or a PRI which varies over a small window.

Several other factors complicate this filtering task. Missing pulses, overlap of pulses, and measurement uncertainties in the receiver increase the amount of processing needed to identify the PRI of each particular emitter. Emitters which intentionally vary their PRI add further problems in computing PRIs and next arrival times. If the signal sorter is part of an airborne system, the DOA parameter of the emitters will change with time. Some emitters can also vary the FREQ with time. Since DOA and FREQ are the primary sorting parameters, an extra amount of processing is needed if either of these parameters changes, because the pulses will not match in the CAM.

The size and power constraints of airborne systems rule out the use of the large, high-speed supercomputers which would be theoretically necessary to achieve the required processing speeds. But by use of the CAM filtering scheme with a small array of microprocessors, a signal sorter can be constructed using large-scale-integration (LSI) circuits which can handle the high data rates and meet the size and power constraints. The end result of this work was the construction of a laboratory prototype of a real-time signal-sorting system.

*A glossary of terms is included as an appendix.
Manuscript submitted October 15, 1981.

SOFTWARE SIMULATION

To test various architectures and algorithms prior to constructing a hardware model of the signal sorter, a software simulation was first written. The simulation was written in FORTRAN and modeled the system to the register level of detail.

In addition to the simulation of a signal sorter system, a software model of the environment was constructed to provide an input data stream for testing the sorter operation. The environment model can generate an interleaved set of pulses, such as would be seen in a real environment. Various types of emitters can be generated. This includes regular emitters with stable parameters and exotic emitters such as those that vary their PRI or frequency. The model allows various signal densities and different mixes of emitter types to be run. All of the emitter parameters are changeable on a run basis, but the same environment can be generated to run against different sorter designs for comparison.

A model of an antenna and receiver system was also made as part of the overall simulation. The antenna system parameters were chosen to model a feasible system. The ability to measure the angle of arrival on a pulse-to-pulse basis was modeled in the antenna and receiver system. This parameter was shown to be important in the signal sorting process.

Various signal-sorter options and architectures were evaluated using these software models, and some results of these studies have been reported [1-4]. The architecture that appeared to work best for dense environments is shown in Fig. 1.

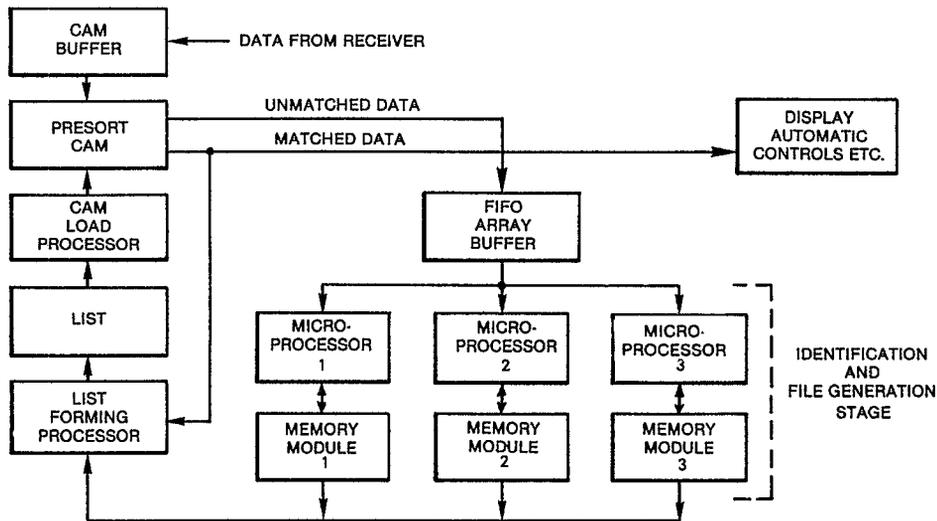


Fig. 1 — Signal-sorting system block diagram

SIGNAL-SORTER ARCHITECTURE

As shown in Fig. 1, the signal-sorting system is partitioned, both logically and physically, into several different subtasks: matching the input data from the receiver in the content-addressable memory (CAM), emitter identification and file management in the microprocessor array, forming of the List of next expected pulse arrivals, and loading the CAM from the List. These subtasks all run simultaneously and asynchronously. This is accomplished by the use of first-in-first-out (FIFO) buffers between these different subsystems. The FIFOs smooth out the variations in input data rates to the different subsystems and allow them to perform their tasks asynchronously as required.

Interrogation in the CAM

The input to the CAM buffer consists of the DOA and FREQ parameters from the receiver and the time of arrival (TOA) of the pulse from a real-time clock in the system. The DOA and FREQ of the new pulse are tested against all the emitters in the CAM simultaneously. A CAM is a memory that is accessed, not by addresses, but by contents. In the scheme implemented in this system, an exact match of all bits is required for a pulse to match. For a matched pulse, an expected next time of arrival (NTOA) is computed from the TOA and the pulse repetition interval (PRI), which is stored in a memory parallel to the CAM, as shown in Fig. 2. The matched data and the NTOA are then passed to the List-forming processor so that they can be loaded into the CAM at the proper time for the arrival of the next pulse from that emitter.

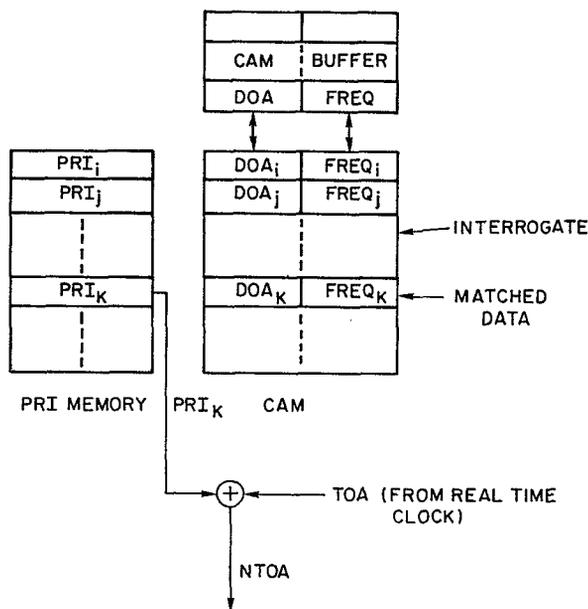


Fig. 2 — The CAM hardware

If a pulse is not matched in the CAM, it is passed to the microprocessor array through the FIFO ARRAY buffer for further processing to determine the reason why it was not matched. There are several reasons why a pulse might not match in the CAM. New pulses which had not yet been characterized would not match. Also, missing pulses and pulses with varying parameters would cause non-match conditions. Since the match criterion is an exact match of all bits, quantization errors will prevent some matches of previously characterized emitters.

Microprocessor Array

The FIFO ARRAY buffer receives the data from the pulses which are not matched in the CAM. The buffer contains in parallel the DOA, FREQ, and TOA of the pulse. The output of the FIFO ARRAY buffer is connected to the microprocessor array. The microprocessor array consists of three identical, parallel microprocessors. Each processor has a 4096-word random-access memory (RAM), which contains the emitter files.

The emitter files are partitioned among the three separate memories according to DOA. There are 64 DOA cells, each of which covers approximately 5.6° in azimuth. Each processor memory contains one-third of the DOA cells, distributed in such a fashion that adjacent DOA cells are in different

processors, as shown in Fig. 3. This interleaved arrangement facilitates the file-searching procedure. Since the DOA of a moving emitter should only change by one between consecutive pulses, and any uncertainty in DOA measurement would be one cell, it is only necessary to search for an unmatched pulse in the current DOA cell and the two adjacent to it. Therefore, the three-processor configuration allows the three DOA cells to be searched simultaneously. Figure 3 shows a pointer stored in the first location of each DOA cell. This pointer is the address of the next available free location in the cell. By looking at this value, the processor can tell directly how many emitters are in that DOA cell.

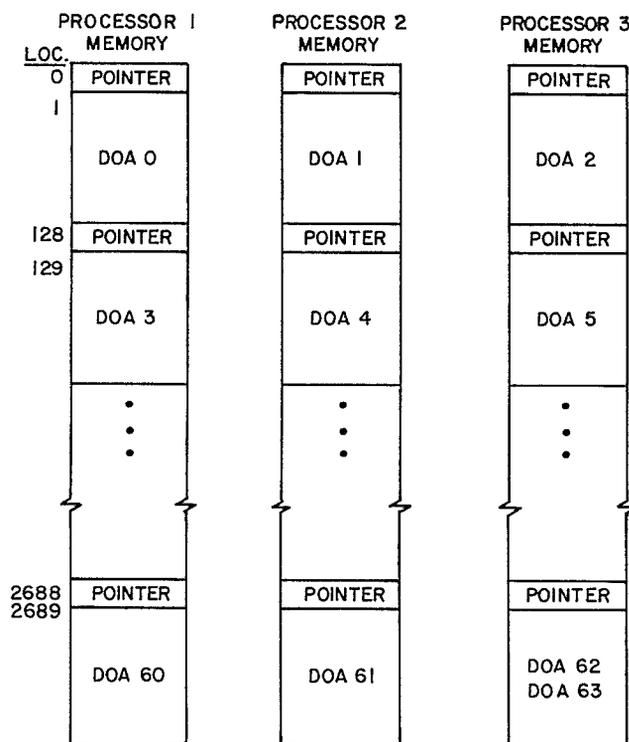


Fig. 3 — Microprocessors' memory map (emitter file)

Two other parameters which help speed up the file-management task are derived from the DOA field through hardware-mapping tables. The FUNCTION tells the microprocessor whether it contains the DOA file of the current pulse or the adjacent DOA cell file for that pulse, either above or below. The BLOCK ADDRESS gives the beginning address within the RAM memory of the DOA file to be searched. Using this address, the processor can access directly the proper DOA cell.

In the microprocessors, the emitter pulse is compared with those in the same DOA cell and also the two adjacent DOA cells. A between-limits match is performed on the FREQUENCY parameter of the pulse. If a match is found, the parameters are updated in the file and the information is passed to the List-forming stage. An emitter which is not found in the file is added to the file.

Five parameters are stored in the file for each emitter. The FREQUENCY and TIME OF ARRIVAL inputs to the processor, along with the computed PULSE REPETITION INTERVAL and TYPE are kept in the file. The number of pulses seen (NPR) is also kept for those emitters which have not yet been identified and classified by TYPE and PRI.

Greater flexibility could be achieved in the microprocessor array if all the memories were accessible by all the processors. This could allow more overlap in processing among the three processors than is currently possible. However, this advantage would be negated by the memory-contention problems and the added memory-contention hardware. Another trade-off area is the fixed maximum size of the DOA bins. No provision was made in this work for a possible overflow condition (more than 25 emitter entries in one 5.6° DOA cell). A more flexible, dynamic memory allocation scheme would also increase the required processing time and decrease overall throughput.

List-Forming Processor

The List is an ordered arrangement of the emitter pulses which have been identified and are being held until each is loaded into the CAM just prior to its next pulse arrival. The List consists of a number of FIFO bins which are ordered as a sequence of time slots. Each bin is loaded with those emitters whose expected next arrival times fall within the same time slot. The contents of a bin of the List are then loaded into the CAM one emitter at a time during the appropriate time slot, as determined by the real-time clock. No ordering is done on the data within a given bin other than the first-in-first-out characteristics of the buffers. Figure 4 illustrates the List and the CAM loading data paths.

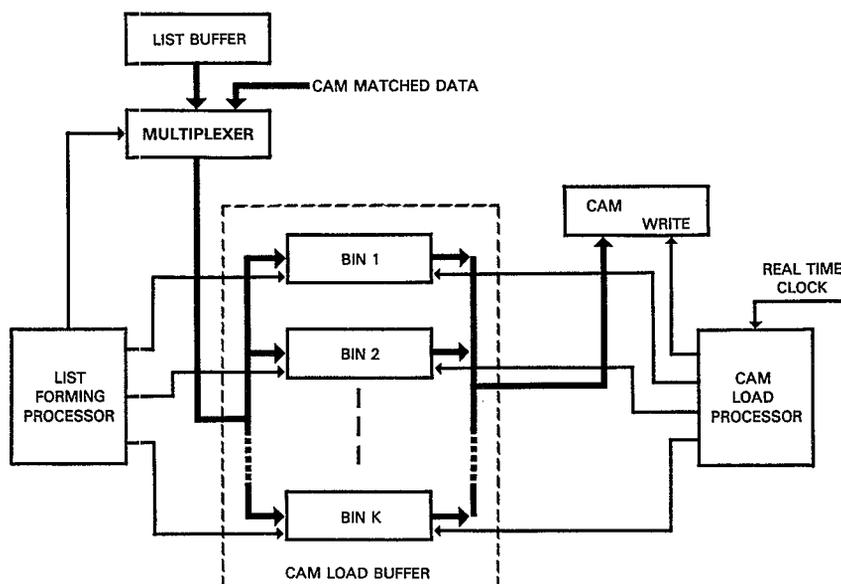


Fig. 4 — List-forming/load-CAM stages

An emitter is loaded into the proper bin of the List corresponding to its NTOA. The association of a particular bin with a given NTOA is determined by a group of consecutive bits in the NTOA referred to as "time-slot" or "time-window" bits. To have a uniform distribution of the emitters in the bins, and to be able to load the emitters into the CAM to provide the highest hit ratio, the appropriate values for the time slot should be chosen. This includes the number of bits in the time window, which determines the number of bins, and the location of the bits within the NTOA word, which determines the width of the time window.

Many different configurations of the List were tried using the simulation. Several different configurations of the number of bins and the time window were shown to be effective for a large range of different environments. It was found, as expected, that the optimum list configurations were related to the expected range of PRIs in the environment. Some of these configurations were not practically feasible because they required an excessive amount of hardware. Several of the configurations did fit

within both the size and performance constraints. Thus, the hardware was built to allow several different List configurations to be used. The List can be configured as either 8 or 16 bins with time slots of either 128 or 256 μ s.

The List-forming processor receives the data from either the microprocessor array through the List Buffer or the CAM match circuitry and loads it into the proper bin in the List. This List is ordered in time only by the bins which contain the emitters and not within the bins themselves.

CAM-Load Processor

The CAM-load processor loads the CAM from the List during those time periods when the CAM is not busy processing the data from the receiver. A real-time clock is used to select the bin from which data are loaded into the CAM. The time slot, as described above, is generated from the real-time clock and is used to determine the module number (bin) from which data are loaded into the CAM.

This procedure of loading the CAM in time slots attempts to make efficient use of the limited CAM space by the loading of only those emitters into the CAM which the system expects to see during the next period of time. Emitters are loaded into the CAM during the same time slot as their next expected arrival time. The loading of an emitter into the CAM precedes its expected arrival time by an amount of time less than or equal to the size of the time slot. However, delays in loading the CAM from the List due to multiple entries in the same time slot could cause the emitter data to be loaded into the CAM after the expected arrival time.

When the CAM is loaded from the List, a window is applied on the real-time clock, to determine the bin for CAM loading, identical to the window used on the NTOA word to determine the bin for loading the List. Once the bin is determined, its data are unloaded sequentially from top to bottom (i.e., FIFO). The emitters in the List are ordered only by next arrival time into the proper bin. A fully time ordered list would overload the List-forming processor and the CAM-load processor.

EMITTER PROCESSING ALGORITHM

An emitter pulse is channeled to the microprocessor array when it does not match with the contents of the CAM. This could occur because of a new emitter, whose parameters are not yet in the file; a previously missing pulse, which caused the emitter not to be in the CAM; pulse overlap, which distorted the parameters detected by the receiver; receiver measurement quantization error; DOA change; or varying parameters, such as PRI or frequency. The task of the microprocessor array is to determine whether the unmatched pulse is from a new emitter or is due to one of the other causes listed. If it is from a new emitter, the processor will attempt to compute the pulse repetition interval (PRI) using the successive pulses seen. The parameters of the new emitter will be stored in the proper DOA cell of the emitter file. The processors will also assign a type to each emitter as a regular (stable PRI) emitter, a pulse-group emitter, a CW emitter, or a jittered PRI emitter.

Figure 5 shows the flowcharts for the processing algorithm. The processing of a new pulse from the buffer is begun simultaneously in all three microprocessors. Therefore, a processor must wait until both of the others are finished with the previous pulse. A processor shows this ready state by setting the SYNC flag. When the three SYNC flags are set and a pulse is in the FIFO ARRAY buffer, the OK flag is set true. This signals the processors to read the next set of emitter data from the buffer. The five data words read from the FIFO ARRAY buffer are DOA, FREQ, TOA, FUNCTION, BLOCK ADDRESS.

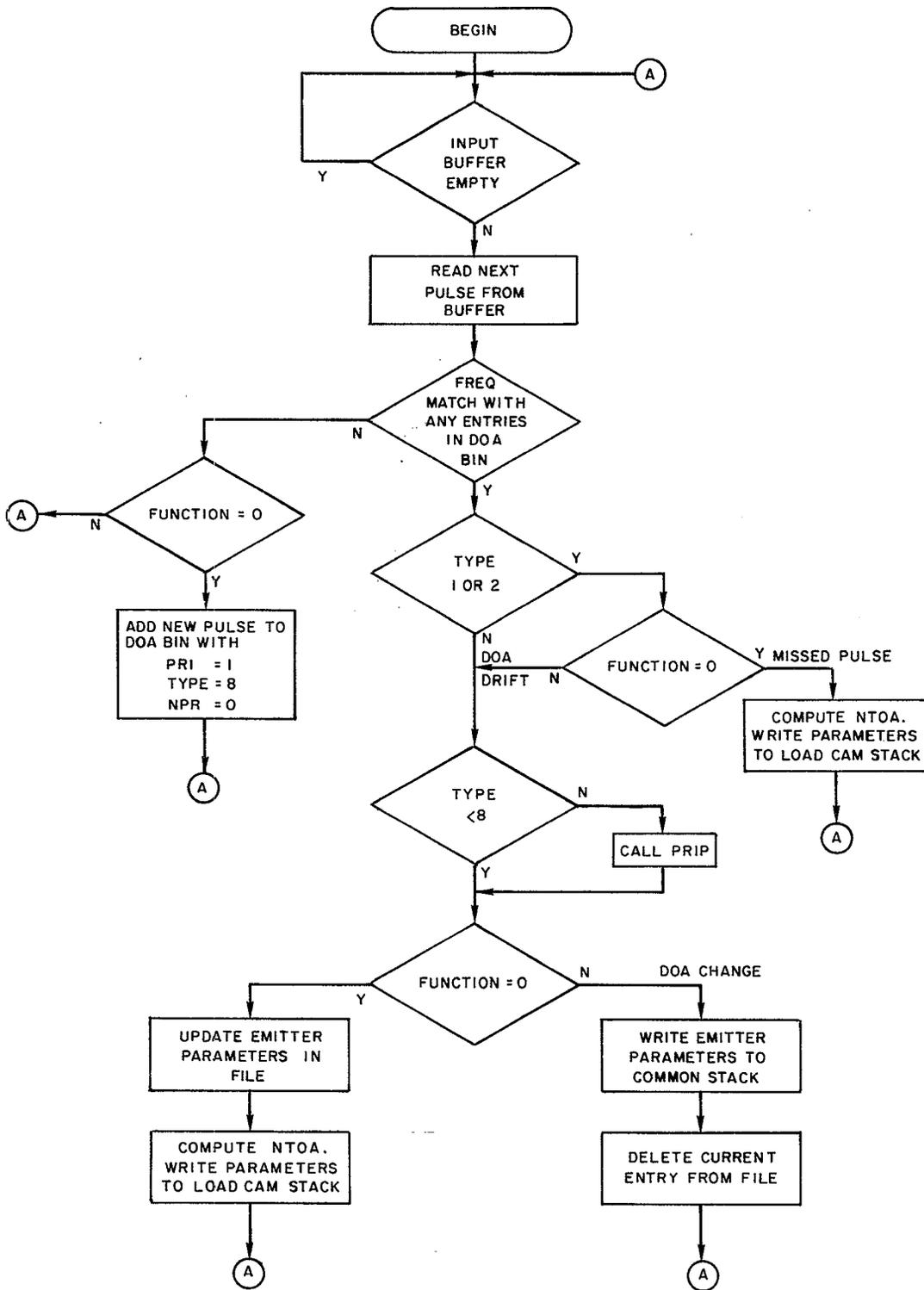


Fig. 5(a) — Pulse-processing algorithm

HUSSON AND EVANS

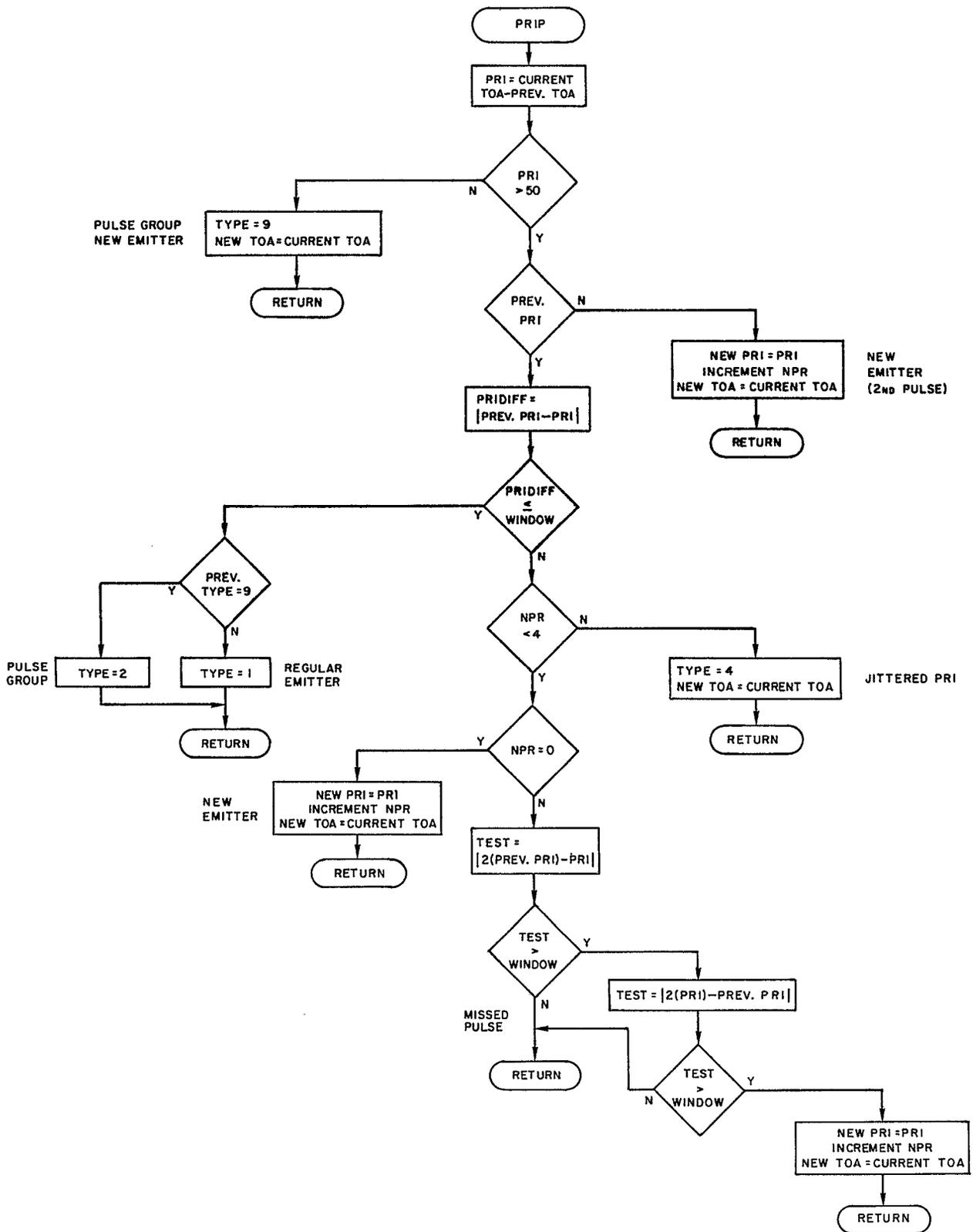


Fig. 5(b) — PRI calculation routine

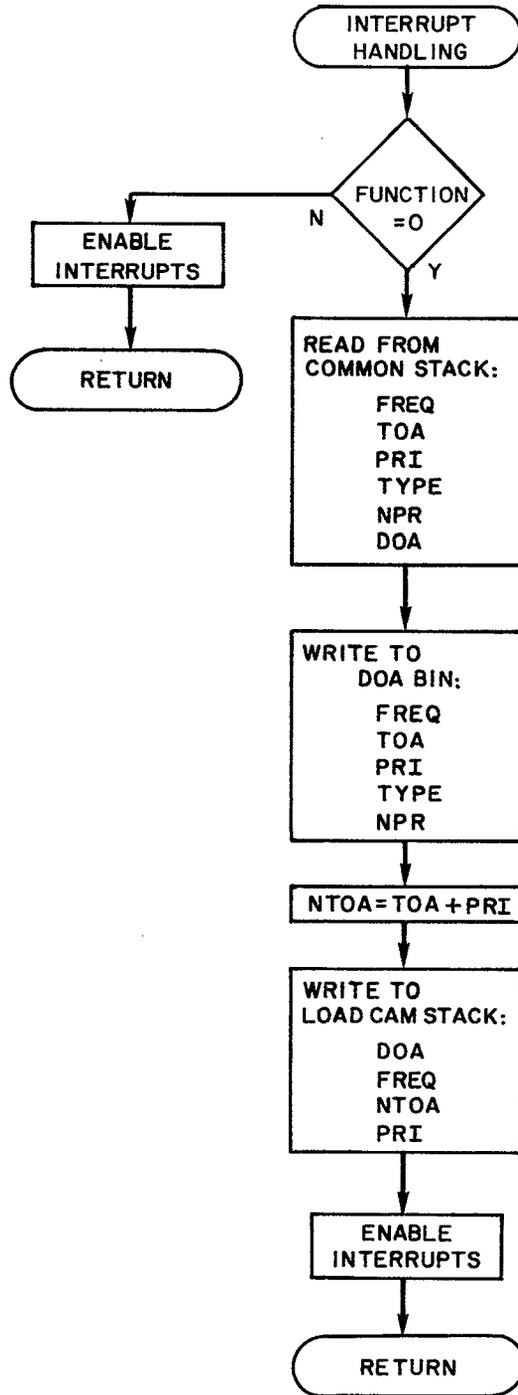


Fig. 5(c) — Interrupt handling routine

The three processors are synchronized only during the time when input data are read from the FIFO ARRAY buffer which is common to them. The processors are not necessarily synchronized during the execution of the entire algorithm.

The BLOCK ADDRESS points to the RAM address of the DOA cell to be searched. This memory location contains a pointer to the next available address in that memory block. If the processor finds no entries in the file in the DOA block being examined, a test of the FUNCTION parameter is performed. If FUNCTION = 0, which signifies that the processor contains the correct DOA cell for the pulse being processed, the new emitter is stored in the file. Note that the other two processors, whose FUNCTION parameters will be 1 and 2 for this emitter, will be simultaneously searching the adjacent DOA cells for a DOA change. Five parameters are stored in the file for each emitter. The FREQ and TOA were read from the FIFO ARRAY buffer. The PRI is initially set to -1 on the first pulse seen from an emitter, because at least two pulses are needed to compute it. The type is initially set to 8, which means a new emitter whose PRI and TYPE have not yet been determined. The NPR parameter, which is the number of pulses seen, is initialized to zero. A flag is also set in the processor to show that a new emitter has been seen, and the memory pointer for that DOA cell is also updated.

If the DOA block in memory is not empty, each of the emitters in the block is compared to the new emitter for the FREQUENCY parameter. This match is a between-limits test, with a programmable window to allow for small frequency variations. If no FREQUENCY match is found, the same procedure for adding a new emitter to the file is performed.

A match of the FREQ in the proper DOA cell means that the emitter had previously been put in the file. The next test is to check whether the PRI and TYPE had already been identified. If the TYPE is 1 or 2, the emitter had previously been identified. If the FUNCTION is 0, meaning no DOA change, a missed pulse is assumed. The expected next time of arrival (NTOA) is computed and the emitter parameters are sent to the load CAM stack (LCS). If the FUNCTION of the matched emitter is not 0, the DOA has moved to one of the adjacent cells. The emitter parameters must then be moved to the new correct DOA block in the proper processor. The emitter data are put into the common stack and an interrupt is generated in the other two microprocessors.

The interrupt handling procedure is shown in Fig. 5(c). If the FUNCTION is not 0, the processor does not contain the file of the DOA cell for this emitter. The processor then simply reenables the interrupt and waits for the other processors to complete. If FUNCTION = 0, the parameters are read from the common stack and stored in the proper DOA cell. The NTOA is then computed and the emitter parameters are passed to the LCS. The interrupts are then enabled, and that processor is ready to begin processing the next pulse.

If the PRI of the matched pulse has not yet been computed (TYPE = 8 or 9), the PRIP subroutine is executed. The flowchart for this subroutine is shown in Fig. 5(b). This subroutine attempts to compute the PRI and set the emitter TYPE for newly seen emitters. For regular emitters, three pulses (two PRIs) are used to verify PRI stability. A stable PRI is defined to be within a programmable limit. A 3- μ s window was used in the simulation, and a 16- μ s window was found to be needed during the actual hardware testing. This window is dependent on the accuracy of the TOA computation and the uncertainties and delays in the data generator. The implications of the choice of window sizes for both PRI and frequency matching are several. Small windows allow measurement uncertainties without building up the emitter file and causing extra processing. A larger window may allow some emitters with frequency or PRI agility to be identified, however it may not allow discrimination between two emitters with close parameters. Previous studies have shown the relationship between resolution and the discrimination problem. After the PRIP routine is executed, the emitter parameters are updated in the emitter file. If a new PRI has been computed, the parameters are also passed to the load CAM stack so the emitter can be put into the CAM to filter future pulses.

SIMULATION RESULTS

The use of a software simulation of the signal sorter allowed much better monitoring of system performance and operation than could be done on the actual hardware in real time. This included running the same scenario against different configurations for comparison. Any point in the system is easily monitored in the software program, making performance evaluation and testing easier.

A more difficult problem is the criteria to use to make performance comparisons. Through trial and error it was found that key performance measures for the signal sorter include the number of "hits" or matches in the content-addressable memory, or more precisely the hit/miss ratio; the sizes or maximum number of words of the various FIFO buffers separating the subsystems; and the processing delays of pulses through the system. The hit/miss ratio gives an overall indication of how well the data filtering is being done. The sizes of the buffers tell whether particular subsystems are keeping up with the data rates; whether the processing load is well distributed among the subsystems; and how the processing load varies with time, such as when additional emitters are first seen. The processing delay is the amount of time from when an emitter pulse is seen at the receiver until it is processed through the system. This is important for a real-time response in a threat scenario.

It was seen that, as expected, these various performance measures were coupled. Changing a parameter in the system configuration could have different effects on the different performance measures. Therefore, comparisons between various architectures and system configurations must take into account several of the performance measures. No specific attempt at combining the different performance measures into a single overall performance number or rating has been made.

The simulation results discussed in this report all use the same basic architecture shown in Fig. 1. Other architectures were studied and some of the results have been reported [1-4]. Different parameters of the system were varied and compared using the simulation program. Some of the more interesting results will be shown.

Theoretically, all parameters in the simulation program can be changed from run to run. However, some parameters are generally held fixed once the basic architecture has been designed. These include such things as the number of microprocessors in the emitter identification array, the processing times for different functions in the system, and the maximum buffer sizes allowed. Some of these were fixed due to physical and technological constraints and some were found by experimentation. The other parameters being tested, which were therefore variable from run to run, were set interactively by user inputs. Figure 6 shows an example of the user interaction with the program. The user starts the simulation by entering SEG MESS4 at the computer terminal. The program then prompts the user for the different input parameters and waits for the user to enter the desired values.

Figure 6 also shows the outputs to the user terminal for a run of the simulation. The user input parameters are shown as part of the output. The NUMBER OF EMITTERS is the number of emitters desired in the scenario for the run. The parameters for these emitters are then set in the data generator program, using the random number generator within a specified range of typical parameter values. The MAX. SEPARATION BET. ON TIMES is the period of time during which all the emitters are turned on. For example, in the case shown, all the emitters will be on by 0.02 s after the start of the run. Having all the emitters turn on in a short period of time (0.02 s) simulates a worst-case processing load for the system. The beginning of the run is set as time 0.0 s. The ASSOC. PROC. TIME IN MICROSEC is the amount of time required for processing an input pulse in the content-addressable memory subsection. This includes the amount of time to read the pulse from the buffer, interrogate the CAM, and place the matched or unmatched data in the proper buffer. The MPP PROC. TIME IN MICROSEC PER INSTRUCTION is the machine cycle time of the microprocessors being simulated. The CAM MANAGER PROC. TIME IN MICROSEC refers to the amount of time required by the system to load the parameters of one emitter from the List to the CAM. All of these processing times

HUSSON AND EVANS

```

OK, SEG MESS4
NUMBER OF EMITTERS = 50
MAX SEPARATION BET. ON TIMES = .02
ASSOC. PROC. TIME IN MICROSEC = 1.0
MPP PROC. TIME IN MICROSEC PER INSTRUCTION = 0.3
CAM MANAGER PROC. TIME IN MICROSEC = 1.0
# CAM REGS = 24
INITIAL # OF MOD. IN THE LCAM STACK = 16
INITIAL # OF BITS SHIFTED = 512
ADVANCE LOAD TIME IN MICROSEC = 0
NUMBER OF EMITTERS IDENTIFIED BEFORE CONFIG = 200
RUN TIME IN SEC. = 0.1

NMCNT=      9  MCNT=   1498  MAX= 580  CAME=  2  MPPB=  5  UPDEUF=  0
MTIME=0.007158  NCWL=   1520  MLCAMBUF=  3  IDOAD=  2
ICFD=  0  NEA=   1660  NIA= 161  INLT1=  9  INLT2=  2  INLT3=  57
  0  0  1  0  0  3  0  0  0  0  1  0  1  1  1  6
  0  2  3  0  0  1  1  0  1  1  0  0  2  0  0  1
  0  0  2  0  2  0  2  1  0  3  1  2  2  0  1  2
  4  0  1  0  0  0  1  1  0  1  0  0  0  0  0  0
    TOTAL EMITERS IN MPPR= 50
    13 14 14 11 13 10 13 12 12 13 13 12 14 12 16 16

**** STOP

```

Fig. 6 — Typical simulation run

were determined by the architecture used and the speeds of commercially available components. By changes in these processing-time parameters different parts of the system can be sped up or slowed down for experimental purposes and worst-case analyses.

The # CAM REGS parameter allows the user to set the number of words in the CAM in the simulation. This is important because a content-addressable memory consumes a much larger amount of space per bit than a conventional random-access memory and it should be kept as small as possible without degrading performance. The next two user input parameters set the configuration of the List. The INITIAL # OF MOD. IN THE LCAM STACK sets the number of bins in the List, and the INITIAL # OF BITS SHIFTED allows the user to enter the time slice per bin in microseconds. The RUN TIME IN SEC. is the length of the simulated engagement. The length of a run should be at least several times the amount of time needed for all the emitters to turn on, to show both the transient and steady-state operation of the system.

Also shown in Fig. 6 are the resulting outputs of the simulation run. NEA is the total number of pulses entering the system from the receiver during the run. NIA is the number of pulses which did not match in the CAM and NMCNT is the number of these misses which occurred after the emitters had been identified and put in the emitter file (steady state). MCNT is the number of pulses which matched in the CAM. The maximum sizes of the CAM input buffer and the MPPR FIFO array buffer during the run are shown as CAMB and MPPB. MAX is the maximum delay of a pulse through the system in microseconds.

The distribution of emitters in the emitter file among the 64 DOA cells is shown next, along with the total number of emitters in the file at the end of the run. The last set of output numbers shows the maximum instantaneous sizes of each of the List bins during the run.

These outputs provide cumulative statistics on the performance of the signal sorter during the run. For example, all the displayed results show maximum values or total counts. To judge performance more fully, it is necessary to see how some of these performance measures varied during the run rather than to see just the total or maximum values. For this reason the ability to provide plots of any variable in the program vs time was added to the simulation.

From the simulation-run example shown in Fig. 6, several variables were plotted vs time, and the plots are shown in Fig. 7. Figure 7(a) shows the count of the number of pulses which miss in the CAM. It is also the total number of pulses going into the microprocessor array. From this graph, it is seen that there are two distinct portions of the run. All of the emitters are turned on during the initial 0.02 s of the run, and they generate a high rate of misses in the CAM until they have all been identified and put into the emitter file. Once the emitters have been identified, the rate of misses decreases drastically, showing the effectiveness of the steady-state filtering in the CAM during this run. The parameter MPPR [Fig. 7(b)] shows the maximum number of pulses in the buffer between the CAM circuitry and the microprocessor array. All of the missed pulses in the CAM pass through this buffer. This value peaks during the time when the emitters are still being identified and the rate of input to this buffer is high, as is shown by Fig. 7(a). Figure 7(c) shows the processing delay for pulses through the system. This parameter is also greatest during the initial phase of the run, when all the new emitters are being identified and the processing load is greatest. The processing delay is defined as the amount of time from the time of arrival of the pulse to the time the microprocessor array has completed processing that pulse. This includes the processing times in the CAM and the microprocessor array and the delays in the FIFO buffers.

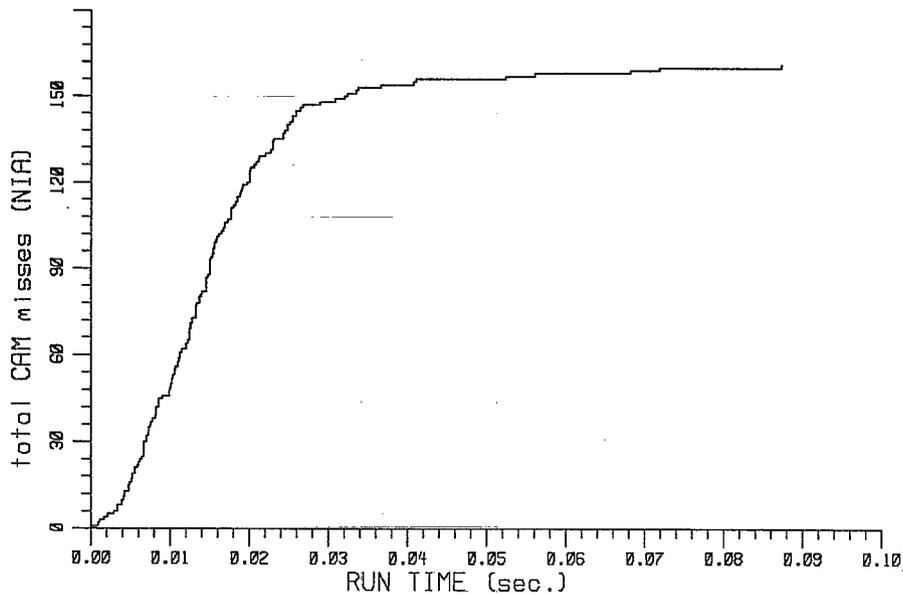


Fig. 7(a) — CAM misses vs time (50 emitters)

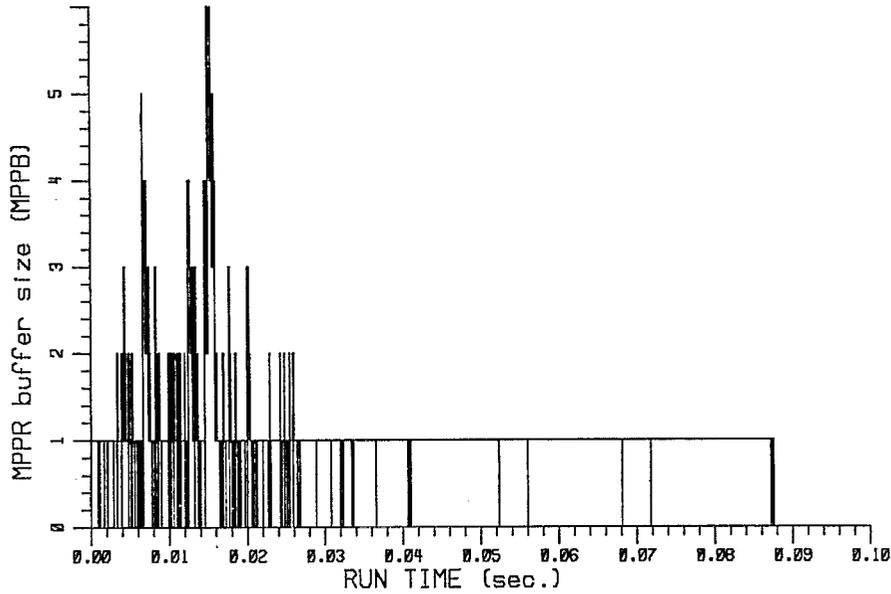


Fig. 7(b) — Array processing buffer size vs time (50 emitters)

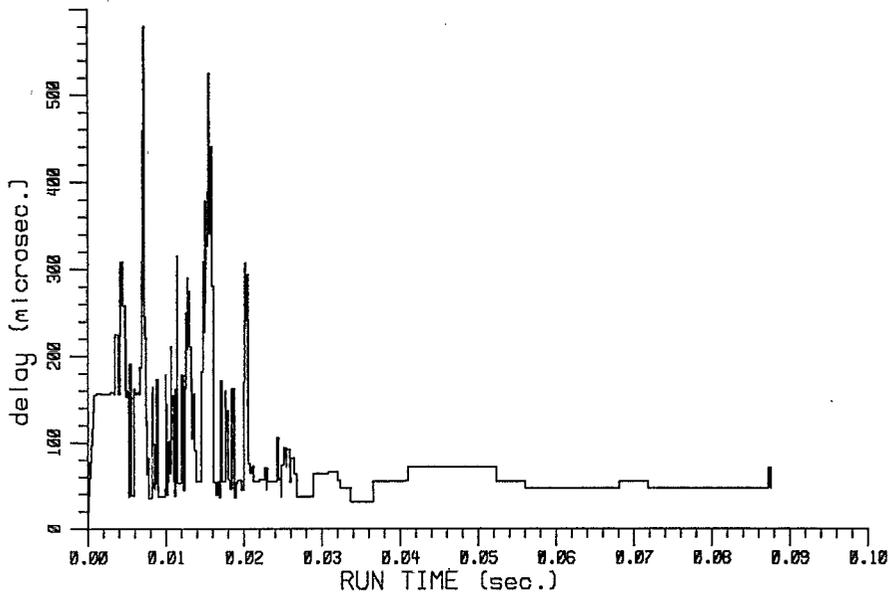


Fig. 7(c) — Pulse-processing delay vs time (50 emitters)

COMPARISON OF ARCHITECTURES

These simulation tools were developed to allow comparisons of different system configurations in a much easier way than by changing and rechanging hardware. This section will show how some different system parameters were determined and tuned using the simulation. Other system architectures have been examined [1,2] as well as different architectures for the microprocessor array [3,4]. The example used shows the time-slice size of the List bins being varied to determine an optimal List configuration. The computer outputs of runs using three different bin sizes, with all other system parameters held constant, are shown in Figs. 8(a) through 8(c). By examining the value of NMCNT (nonmatches in the content-addressable memory front-end), we can see that a bin size of 512 μ s works

OK, SEG MESS4
 NUMBER OF EMITTERS = 80
 MAX SEPARATION BET. ON TIMES = .02
 ASSOC. PROC. TIME IN MICROSEC = 1.0
 MPP PROC. TIME IN MICROSEC PER INSTRUCTION = 0.3
 CAM MANAGER PROC. TIME IN MICROSEC = 1.0
 # CAM REGS = 24
 INITIAL # OF MOD. IN THE LCAM STACK = 16
 INITIAL # OF BITS SHIFTED = 256
 ADVANCE LOAD TIME IN MICROSEC = 0
 NUMBER OF EMITTERS IDENTIFIED BEFORE CONFIG = 100
 RUN TIME IN SEC. = 0.1

NMCNT= 565 MCNT= 1737 MAX= 921 CAMB= 2MPPB= 12 UPDBUF= 0
 MTIME=0.018287 NCWL= 2894 MLCAMBUF= 8 IDOAC= 4
 ICFD= 0 NEA= 2546 NIA= 808 INLT1= 565 INLT2= 4 INLT3= 641
 0 0 3 0 3 0 1 1 4 2 1 3 2 7 4 0
 2 3 2 0 2 1 0 1 1 1 0 1 1 1 0 0
 2 1 0 3 0 3 2 1 0 0 0 2 1 3 0 2
 1 0 1 3 2 2 2 1 1 2 1 0 0 0 0 0
 TOTAL EMITTERS IN MPPR= 80
 17 15 15 14 13 14 14 15 19 17 14 13 16 15 17 17

**** STOP

Fig. 8(a) — Simulation run with 256- μ s List bin size

HUSSON AND EVANS

OK, SEG MESS4

NUMBER OF EMITTERS = 80

MAX SEPARATION BET. ON TIMES = .02

ASSOC. PROC. TIME IN MICROSEC = 1.0

MPP PROC. TIME IN MICROSEC PER INSTRUCTION = 0.3

CAM MANAGER PROC. TIME IN MICROSEC = 1.0

CAM FEES = 24

INITIAL # OF MOD. IN THE LCAM STACK = 16

INITIAL # OF BITS SHIFTED = 512

ADVANCE LOAD TIME IN MICROSEC = 0

NUMBER OF EMITTERS IDENTIFIED BEFORE CONFIG = 100

RUN TIME IN SEC. = 0.1

```

VMCNT=    25  MCNT=   2277  MAX=  847  CAMB=  2MPPB=  11  UPDBUF=  0
MTIME=0.017958  NCWL=   2329  MLCAMBUF=  4  IDOAB=  4
ICFD=  0  NEA=   2546  NIA=  268  INLT1=  25  INLT2=  4  INLT3= 101
  0  0  3  0  3  0  1  1  4  2  1  3  2  7  4  0
  2  3  2  0  2  1  0  1  1  1  0  1  1  1  0  0
  ?  1  0  3  0  3  2  1  0  0  0  2  1  3  0  2
  1  0  1  3  2  2  2  1  1  2  1  0  0  0  0  0
    TOTAL EMITTERS IN MPPR= 80
  17 20 17 19 20 23 16 22 22 17 19 21 21 19 24 18
    
```

**** STOP

Fig. 8(b) — Simulation run with 512- μ s List bin size

OK, SEG MESS4

NUMBER OF EMITTERS = 80

MAX SEPARATION BET. ON TIMES = .02

ASSOC. PROC. TIME IN MICROSEC = 1.0

MPP PROC. TIME IN MICROSEC PER INSTRUCTION = 0.3

CAM MANAGER PROC. TIME IN MICROSEC = 1.0

CAM REGS = 24

INITIAL # OF MOD. IN THE LCAM STACK = 16

INITIAL # OF BITS SHIFTED = 1024

ADVANCE LOAD TIME IN MICROSEC = 0

NUMBER OF EMITTERS IDENTIFIED BEFORE CONFIG = 100

RUN TIME IN SEC. = 0.1

```

NMCNT=   374  MCNT=   1928  MAX=  847  CAMB=  2MPPB=  11  UPDBUF=  0
MTIME=0.017958  NCWL=   2662  MLCAMBUF=  8  IDOAD=  4
ICFD=  0  NEA=   2546  NIA=  617  INLT1=  374  INLT2=  4  INLT3=  450
  C  0  3  0  3  0  1  1  4  2  1  3  2  7  4  0
  2  3  2  0  2  1  0  1  1  1  0  1  1  1  0  0
  2  1  0  3  0  3  2  1  0  0  0  2  1  3  0  2
  1  0  1  3  2  2  2  1  1  2  1  0  0  0  0  0
    TOTAL EMITERS IN MPPB= 80
    35 38 33 37 29 34 40 37 35 39 31 38 38 32 36 41
    
```

**** STOP

Fig. 8(c) — Simulation run with 1024- μ s List bin size

best when there are 16 bins in the List, for the environment that was used in the run. The data-generator routine for these runs allowed emitters to be generated with pulse repetition interval (PRI) values ranging between $500 \mu\text{s}$ and 8 ms . The number of emitters in the scenario was 80. All emitters were regular-type emitters, meaning that they had stable frequency and PRF parameters. The total pulse density was on the order of 25,000 pulses per second.

When the List bin size is $512 \mu\text{s}$, the total time taken to cycle through all 16 bins is 8.192 ms , which is greater than the maximum time between pulses of any emitter in the scenario. This configuration therefore works best. The main reason why a bin size of $256 \mu\text{s}$ does not perform as well is that the system will cycle through all the bins of the List in 4.096 ms . For those emitters whose PRI is greater than this value, the problem of where to put the next pulse in the List occurs. These emitters may not get into the CAM at the proper time, and thus more misses will occur in the CAM, which will generate a greater processing load on the system. A bin size of $1024 \mu\text{s}$ causes a different effect. Since many emitters will have PRIs less than the time-slice size of a bin, more than one copy of the same emitter may be in one bin simultaneously. This could cause the number of entries in a bin to exceed the CAM size, which could cause emitters in the CAM to be overwritten by others before the emitter pulse arrives. This again would cause an increase in the number of misses in the CAM. Figure 9 graphically shows the count of CAM misses vs time during the three runs. It can be seen that the performance of the system during the initial emitter turn-on and identification phase was identical for all three runs.

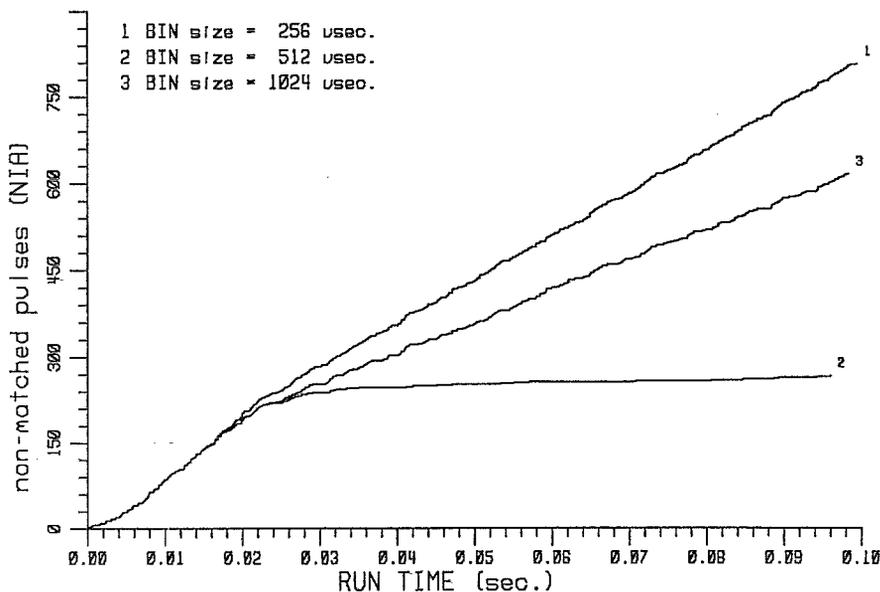


Fig. 9 — CAM misses vs List bin size

SIGNAL-SORTER PROTOTYPE HARDWARE

The signal-sorter system shown in Fig. 1 and described in the Signal-Sorter Architecture section has been constructed as a laboratory prototype. The system consists of 18 wire-wrapped 23 cm by 23 cm (9 in. by 9 in.) circuit boards (Fig. 10). Schottky Bipolar semiconductor logic is the predominant type employed. This section provides details of the various subsections of the system.

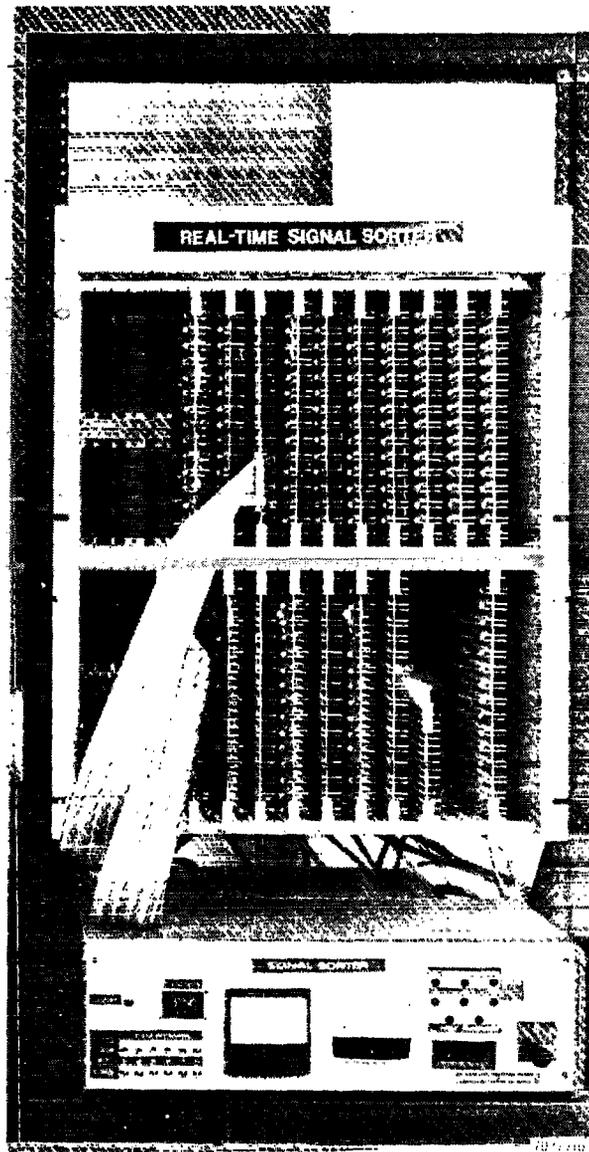


Fig. 10 — Signal-sorting system

Microprocessors

The three identical processors in the microprocessor array were custom designed for the signal-sorting task using Advanced Micro Devices AM2901 bit-slice microprocessors as the central processing elements. Four 2901s were used to construct a 16-bit microprocessor with a basic machine cycle time of 300 ns. The instruction execution times range from 0.6 to 1.2 μ s including the instruction fetch from memory.

There were several reasons why the 2901 bit-slice microprocessor was chosen for this task. It was determined that a word width of 16 bits would be needed, and at the time of the initial design there were no 16-bit microprocessors available. This word width requirement, along with the fact that the required processing speeds were much faster than that of any of the available 8-bit microprocessors,

meant that a custom-designed processor was needed. The other advantage of the bit-slice approach was that it allowed a custom-designed, microprogrammed instruction set to be implemented. This further enhanced the processing-speed capabilities. Among the several available bit-slice microprocessors, the AM2901 was chosen because it was 4 bits wide, it used a single power supply, and it had many compatible support circuits, since its internal electronics were the popular TTL type.

The architecture and instruction set of the processors were designed specifically for this signal-sorting task, but with enough instruction-set flexibility to allow different identification algorithms to be programmed. Many custom instructions were microprogrammed to perform several tasks simultaneously during one instruction cycle. Each processor occupies two circuit boards. On what is called the main board are the arithmetic logic unit (ALU), the control unit, the program memory (1 K × 16), and the interrupt logic, as shown in Fig. 11. The auxiliary processor board (Fig. 12) contains the data memory (4 K × 16) and the I/O buffers to the Common Stack FIFO and List buffer FIFO. The two boards are linked by two unidirectional 16-bit data busses and various other clock and control signals. The D bus is a tristate input bus which is also connected to the output of the FIFO ARRAY BUFFER. Figure 13 shows the fabricated auxiliary processor board. The three processors are linked together by the interrupt signals and the Common Stack. The Common Stack is a 64-word FIFO that is used to pass data among the three processors.

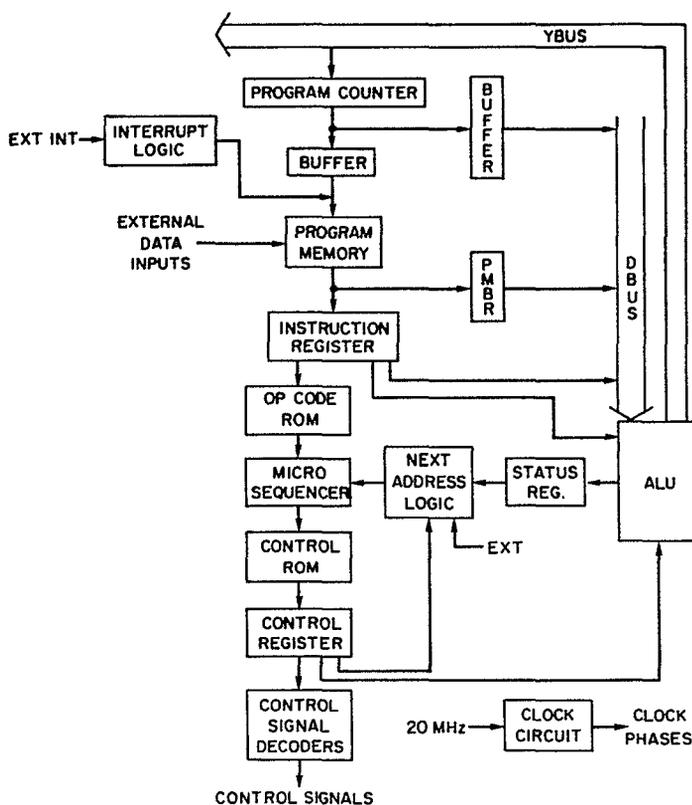


Fig. 11 — Microprocessor block diagram

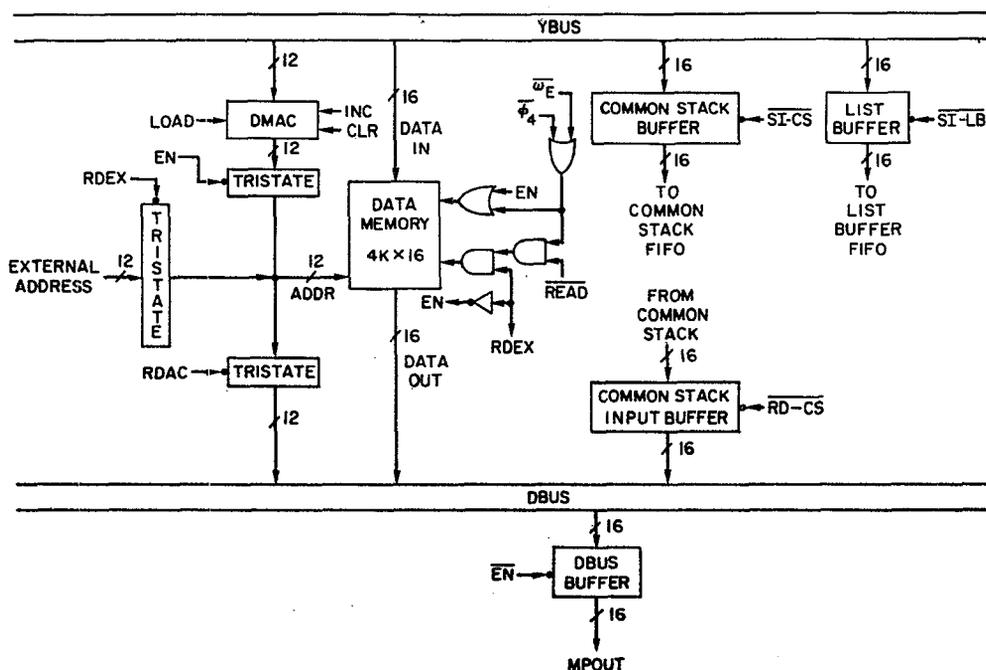


Fig. 12 — Microprocessor auxiliary board block diagram

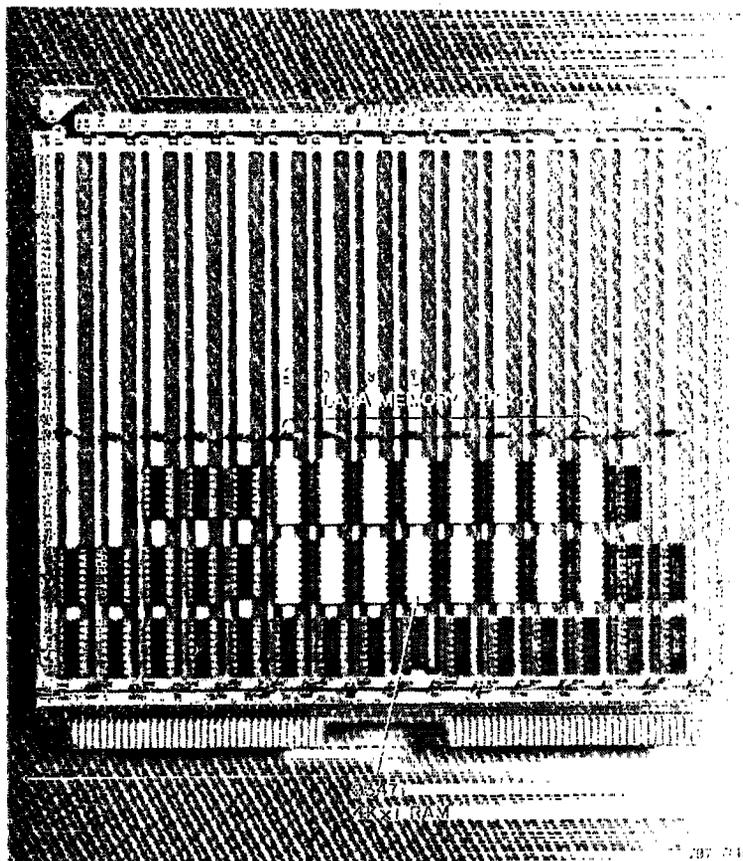


Fig. 13 — Microprocessor auxiliary board

The instruction set of the processors is microcoded and is contained in programmable read-only memories (PROMs). This means that the instruction set itself can be easily added to or modified. If a production system was actually built using a fixed processing algorithm, the algorithm could be coded directly in microcode, which would reduce the overall processing time by more than 50% since all the overhead of instruction fetches would be eliminated. This would allow higher data rates to be handled by the system.

Content-Addressable Memory (CAM)

The CAM hardware consists of the array of content-addressable memory circuits, the CAM buffers, the match logic and the PRI memory (Fig. 14). This circuitry is all controlled by the CAM controller, which will be described in detail later.

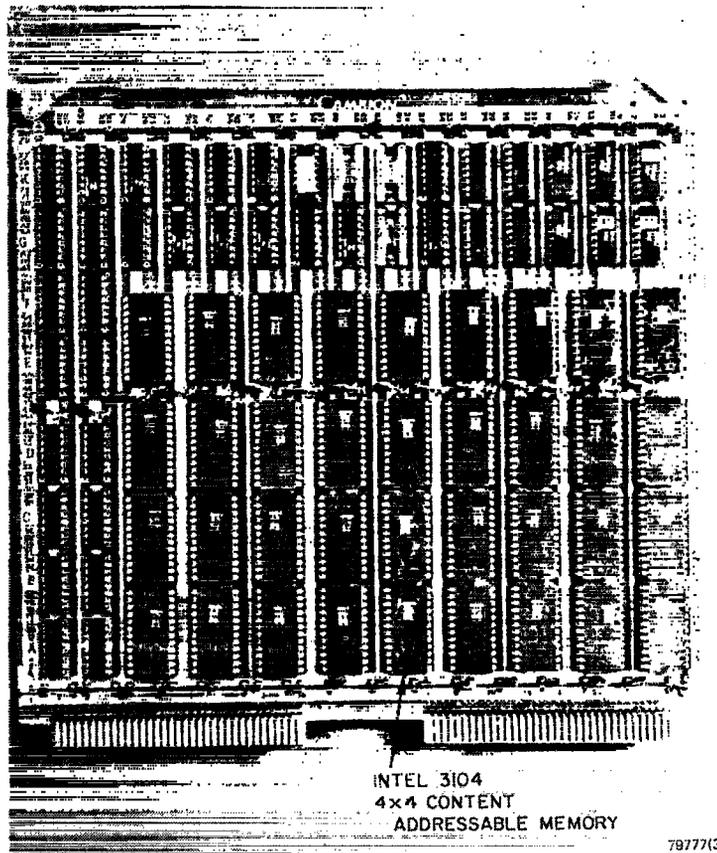


Fig. 14 — CAM board

The CAM array and the CAM input buffer are shown in Fig. 15. The CAM array consists of 36 content addressable memory modules of 4 bits by 4 words each. They are configured to form a CAM array of 24 words of 24 bits each. The CAM word is separated into two fields, 12 bits for the FREQUENCY and 12 bits for the DOA, although only 6 bits are currently being used for this field. The CAM can perform a MATCH/NO MATCH search of its entire memory in 80 ns. All bits in a word must match for the MATCH line to be set true.

The CAM input multiplexer allows either of two inputs into the data input lines of the CAM. One input source is the data loaded into the CAM from the List. The other source is the CAM input

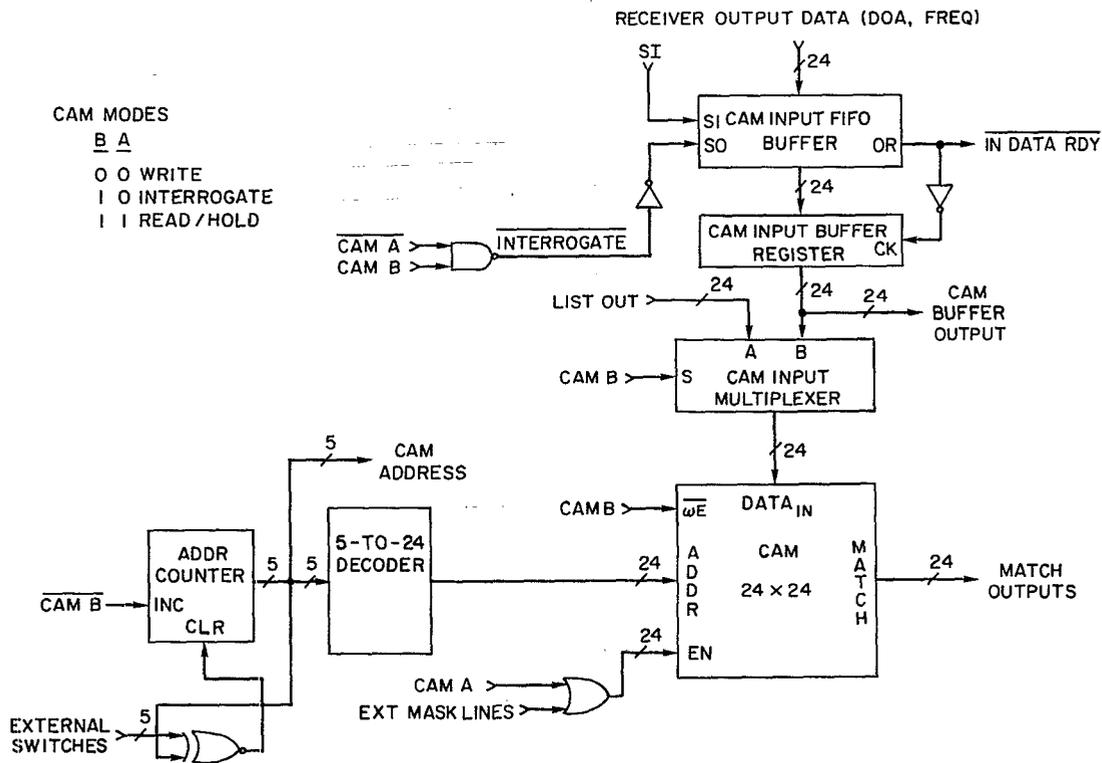


Fig. 15 - Content-addressable memory

buffer which contains the digitized receiver outputs. The CAM buffer is a 64-word by 24-bit first-in-first-out (FIFO) buffer, which gets its inputs from the receiver outputs, or from the data stream generator in the case of the laboratory tests.

The PRI memory is a 24-word by 12-bit memory configured in parallel to the 24-word CAM array. It holds the PRI values of the emitters currently in the CAM. These PRI values are used to generate the expected next time of arrival (NTOA) of those pulses which match in the CAM. The MATCH logic decodes the MATCH outputs from the CAM to the address of the CAM location that was matched. This address is used to get the proper value from the PRI memory to calculate the NTOA.

List

The List hardware consists of the 16 List bins, the List input multiplexer, and the List buffer, as shown in Fig. 4. Included with each bin of the List is logic which allows the List to be configured as either 8 or 16 bins and also logic which keeps track of the number of emitters in each bin of the List and the maximum number in the bin since system reset.

Each bin is a FIFO of 64 words by 36 bits. The bin size is 64 words because the high-speed FIFO devices used to fabricate it are 64 words deep by 4 bits wide. The bits are divided among three fields, 12 bits each for DOA, FREQUENCY, and PRI. The determination of which bin in the List to load an emitter into is based on four bits from the expected next time of arrival word.

The List buffer is the buffer between the microprocessor array and the List. This has also been referred to as the Load CAM Stack. The size of this buffer is 64 words by 16 bits. The four words for

each emitter (DOA, FREQ, PRI, NTOA) are loaded into the buffer by the processors in a serial-by-word fashion. Another FIFO was added in parallel to the List buffer to output the same data to an external display device (the minicomputer, in this case).

The List input mutliplexer selects between the two sources of inputs to the List. These two sources are the List buffer, with the identified emitters from the microprocessor array, and the matched data from the CAM array.

CAM Controller

The CAM controller is a microprogrammed controller which performs three functions in the signal-sorter system. These tasks are to control the interrogation of the CAM with the input data, to load the CAM from the List, and to load the List. The controller logic is diagrammed in Fig. 16. The microcontroller used is the Advanced Micro Devices AM2911, and the microprogram is stored in programmable read-only memories (PROMs) for ease of programming and modification. The basic micro-cycle time of the controller is 300 ns. At this clock rate, it requires 900 ns (3 microcycles) to process an input pulse from the CAM input buffer to either the List (CAM match) or the FIFO ARRAY buffer (CAM mismatch). Loading the CAM from the List requires 600 ns. The time required to load the List from the LIST BUFFER (microprocessor array) is 1.8 μ s. This procedure involves demultiplexing the data from the FIFO into a single parallel word to be loaded into the List. During this time, the CAM controller cannot process any incoming pulses. The theoretical limit of the current CAM controller scheme is a maximum input data rate of about 500,000 pulses per second. Higher data throughput rates could be achieved by increasing the amount of logic in the CAM controller to overlap the multiple tasks.

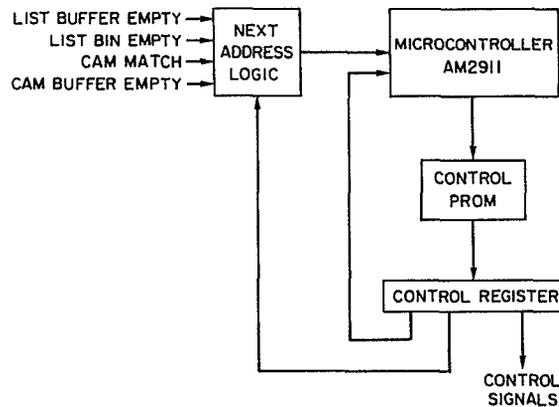


Fig. 16 — CAM controller

A flowchart of the CAM controller process is shown in Fig. 17. Since the controller has multiple tasks, they are prioritized in the order (1) process input pulse data, (2) load the List, (3) load the CAM. These priorities are implemented in the microcode by the order in which the controller checks the status flags.

LABORATORY TESTING

Test Configuration

The signal-sorter system has been interfaced to two general-purpose minicomputers for testing in the laboratory. The minicomputers are a Prime P300 (Fig. 18) and a Prime P400. Both are 16-bit

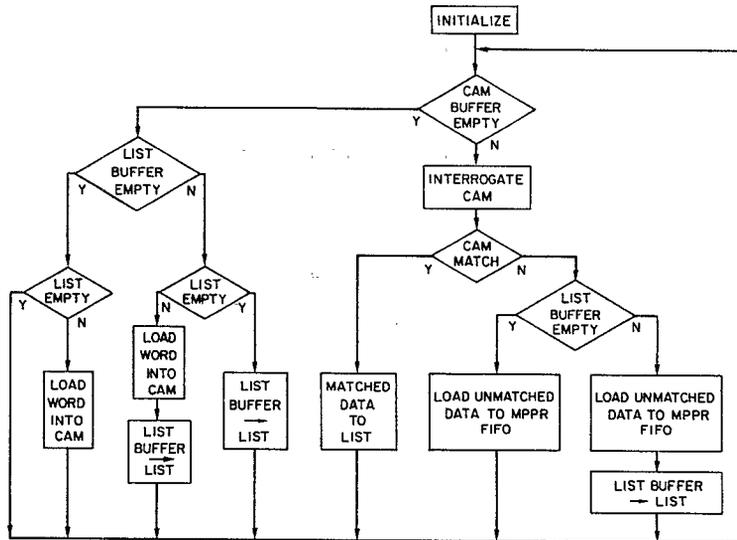


Fig. 17 - CAM controller algorithm

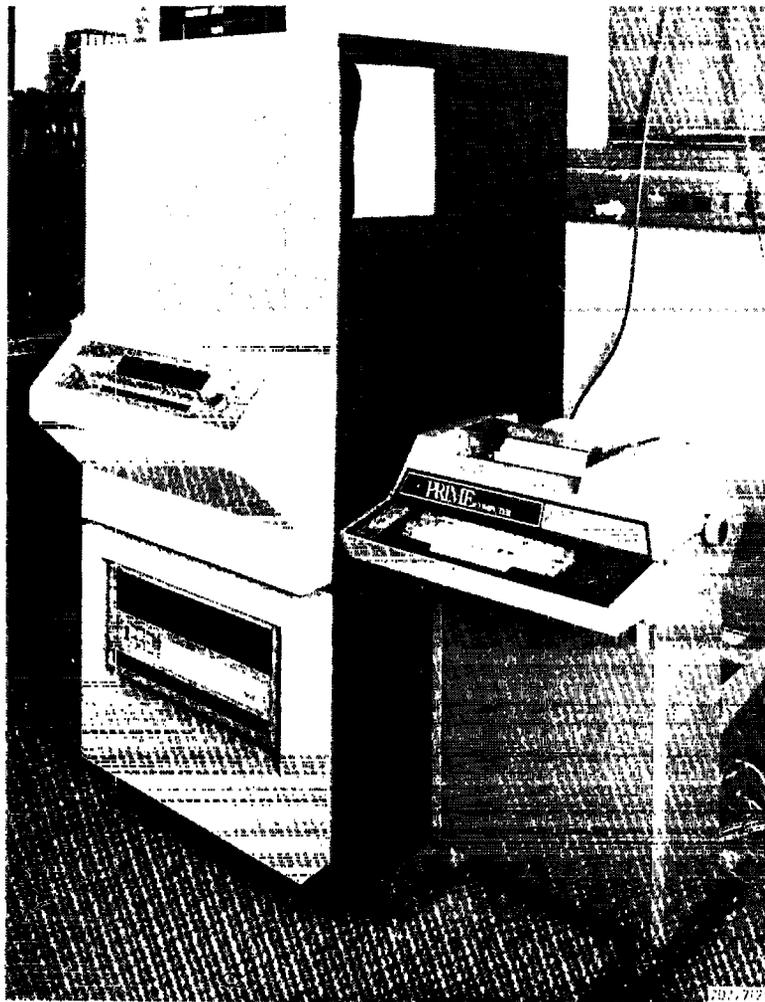


Fig. 18 - Prime P300 minicomputer

general-purpose computers with parallel input and output interfaces. The laboratory configuration is shown in Fig. 19. The P300 computer is used to load the software into the microprocessors, initiate and control the signal-sorter operation, monitor the hardware test points, and provide the input data stream to the sorter. The P400 computer receives information from the signal sorter to provide a real-time display of the emitters that have been identified.

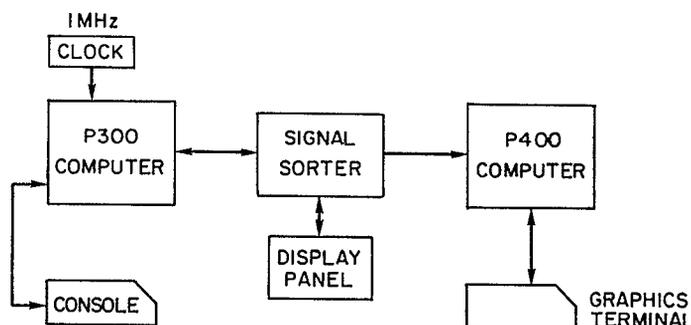


Fig. 19 — Laboratory test configuration

The interface to the P300 consists of four 16-bit unidirectional busses plus the appropriate handshaking signals. A single control program in the P300 directs the operation of the signal sorter and controls the data-stream generator, based on operator commands which are input through the computer's console. The control functions allowed are load the microprocessor software, system reset, system run, clear CAM, and generate the test data stream. The monitoring functions allow the operator to examine the microprocessor memories (emitter file) and the sizes of the List buffers. Two types of data streams can be generated for testing. A set of pulses for a small number (less than 20) of emitters can be generated. The operator specifies the DOA and FREQ parameters for each emitter and also the number of pulses to be input to the signal sorter. This allows known bit patterns to be moved through the sorter to test the overall hardware operation.

The other data stream generator subroutine uses the same data generator routine used in the software simulation. The operator input to the routine is the number of emitters desired. The program then generates a set of emitters with random parameters. From the emitter parameters a list of pulses seen by the simulated receiver, ordered in time, is created in the memory of the P300. This list is then output to the signal sorter in real time, based on the TOAs computed. An external 1-MHz clock provides the timing for the computer to output the data stream. This data stream allows for a more thorough test of the hardware and the signal sorting processing scheme. Both of the data-stream generators can provide repeatable data streams for comparing different system configurations and for verifying hardware operational status.

The hardware monitor function of reading the microprocessor data memories allows the operator to look at the emitter files. This can show how many emitters have been identified and their stored parameters (DOA, FREQ, TOA, TYPE, PRI).

The display program on the P400 computer provides a real-time display of those emitters identified by the microprocessor array. The display is simply a dump of the data passing on the interface between the microprocessor array and the List buffer of the signal sorter. The data passing this point are those emitters which have been identified by the processors and those emitters which were previously identified but for some reason did not match a pulse in the CAM. The display program currently has no provision for purging nonactive emitters from the graphics display, but it is useful for providing real-time feedback of system operation in terms of seeing what emitters have been identified.

Several other external signals are also being monitored. These include the state of the microprocessors (RUN, HALT, FAULT), the microprocessor SYNC signals, and the number of NO MATCHs in the content-addressable memory. The microprocessors' status and the SYNC are displayed in LEDs on the control panel. The number of NO MATCHs is kept in an external hardware counter.

Test Results

Basic tests of each hardware module were made during the integration process. After the system was completely built and tested, further tests were conducted to determine if the system performed like the software simulation. In doing these tests, several limitations of the laboratory test facility were discovered. The worst limitation was on the data-stream generator. Both the number of simultaneous emitters and the number of pulses per second that could be generated by the Prime P300 computer were smaller than desired for a full test. When more than ten simultaneous emitters were generated, the computer-output delays caused the PRIs of the emitters to appear unstable to the signal sorter. Since the loading of the CAM in the signal sorter is based on stable PRIs, more pulses did not match in the CAM than would be expected theoretically.

Table 1 shows five runs made while the number of emitters was varied. All emitters generated for these runs were TYPE 1 emitters, that is, with stable PRI and frequency parameters. When ten emitters were input to the signal sorter, the delays caused by the P300 output caused some of the emitters to be identified as TYPE 4 emitters, with an unstable PRI parameter. The percentage of non-matching pulses in the CAM also increased. Theoretically, if an emitter has a stable PRI, only three pulses are required for identification. For ten emitters, the minimum number of missed pulses in the CAM would be 30. When the number of emitters was increased to 20, both the percentage of non-matches and the percentage of emitters identified as TYPE 4 increased.

Table 1 — Laboratory Test Results (CAM Size = 24, Run Time = 0.06 s)

Run	Number of Emitters	Total Number of Pulses	Number of CAM Nonmatches	TYPE 1 Emitters Identified	TYPE 4 Emitters Identified
1	5	516	19	5	0
2	5	508	23	5	0
3	10	1696	53	9	1
4	10	1696	63	8	2
5	20	2564	250	13*	8

*One duplicate entry in emitter file

The overall logic of the signal-sorter system was verified by these tests, even though direct comparison could not be made to the simulation results and denser environments could not be run. When runs were made with five emitters, the results showed that the CAM filtering and the emitter identification worked as designed. Another test of the CAM filtering procedure was made by varying the number of words in the CAM. The CAM size is easily varied by the control-panel switches. The results showed that when the number of CAM words was decreased below the number of emitters in the environment, more input pulses were missed in the CAM. This is as expected and agrees with previous results obtained using the simulation. Some of the results of this test are shown in Table 2.

If a better data-generating scheme could be obtained, further tests could be performed on the hardware. Based on extrapolation of the limited results, performance of the signal should match that of the software simulation.

Table 2 — Test Results: Effect of Varying CAM Size
(Number of Emitters = 5, Run Time = 0.06 s)

Run	CAM Size	Total Number of Pulses	Number of CAM Nonmatches
1	24	516	19
2	12	556	21
3	4	483	54

SUMMARY

The software simulation of the signal-sorter system showed that identifying many emitters in a dense environment in real time is possible with current technology. Using the simulation, an architecture was developed which would both handle the high data rate requirements and be feasible to construct with current state-of-the-art components. The resulting architecture was tested against various environments with the simulation. The simulation results and a paper design of the system showed that the theoretical processing limit of the signal-sorter system was on the order of 500,000 pulses per second. The major constraining factor is the speed of the content-addressable memories and the CAM controller circuitry. The CAM controller circuitry could be speeded up by the use of a hard-wired logic design instead of the firmware-based microcontroller.

A hardware prototype of the signal-sorting system for testing in the laboratory was then built. The hardware in this system was described in a previous section. After all the modules were tested and found to be functional, the integrated system was interfaced to a minicomputer in the laboratory for system testing. Included in the hardware that was built were many test points and additional circuitry for monitoring the system operation.

Tests were run using this laboratory configuration, and the closed-loop signal sorter was able to identify and filter effectively multiple emitters in a data stream. Limitations in the general-purpose minicomputer interface limit the density of an environment that can be generated to less than 50,000 pulses per second. This data rate is much smaller than the projected limit of the system which was built. As of now, the hardware system has not been tested to its limits. However, the performance of the hardware for the less-dense environments closely matches the performance of the simulation.

It is hoped that the system will be tested further with a data generator that can provide a higher data rate. It would also be desirable to interface the signal sorter to a receiver front end in the laboratory for accurate real-time tests of these architectural concepts. This signal sorter would act as a preprocessor to an overall EW system, which would use the emitter information determined by the signal sorter to provide identification, classification, and real-time response.

One problem which was not fully solved in this work was the identification and classification of exotic emitters. Examples of this class are frequency-agile and random-PRI emitters. The predicted pulse time of arrival scheme described in this report, using the matching of direction of arrival and frequency in a CAM, would not be as effective in identifying and filtering these types of emitters. Runs made with the current simulation show this to be true. Schemes such as adding another CAM to hold these classes of emitters or adding special exotic-emitter processors to the system have been suggested. The testing of these possible solutions will require more work to be done using the simulation programs.

REFERENCES

1. R. Evans, "Architectural Considerations in Signal Sorter Design," Ph.D. Dissertation, The George Washington University, 1975.
2. A. Abdalla and R. Evans, "Real-Time Signal Sorting in a Dense Environment," *Proceedings of the 1975 Conference on Information Science and Systems*, Johns Hopkins University, Baltimore, Md., Apr. 1975, p. 422.
3. A. Abdalla and R. Evans, "Simulations of a Multiprocessing System for Real-Time Signal Sorting," *Proceedings of the Ninth Hawaii International Conference on System Sciences*, University of Hawaii, Honolulu, Jan. 1976, p. 184.
4. A. Abdalla, L. Cornell, and R. Evans, "Microprocessors Pipeline for Real-Time Signal Sorting," *Proceedings of the Tenth Hawaii International Conference on System Sciences*, University of Hawaii, Honolulu, Jan. 1977, p. 86.

Appendix

GLOSSARY OF TERMS

Angle of Arrival — The direction of an incoming pulse with respect to the boresight of the receiving platform.

Bit-Slice Microprocessor — A section of a microprocessor that may be combined in parallel with other such sections to form complete CPUs with various word lengths.

Content-Addressable Memory (CAM) — A random-access memory which is accessed by the data contents of the memory rather than by an address.

CAM Controller — Control circuitry which provides timing and control signals for the CAM Load Processor and the List Forming processor (see Fig. 1).

CAM Load Processor — Processor which provides the task of loading the CAM from the List (see Fig. 1).

Common Stack — The buffer used for transferring data from the microprocessor array to the List-Forming processor.

Direction of Arrival (DOA) — See angle of arrival.

Emitter — The source of a radar pulse.

FIFO — First-in-first-out buffer used to interface subsystems having different data-rate capabilities.

FIFO Array Buffer — The buffer between the CAM circuitry and the microprocessor array (see Fig. 1).

List — The memory circuitry which holds the pulse parameter data which will be loaded into the CAM (see Fig. 1).

List Bins — Partitions of the List which are used to order the emitter data in the List in a next time of arrival sequence (see Fig. 2).

List-Forming Processor — Processing portion which loads the proper List bin with the data passed from the content-addressable memory or the Common Stack.

Microprocessor — The central processing unit (CPU) of a small computer, implemented on one or a few integrated circuit packages.

Microprocessor Array — The group of parallel microprocessors which perform the main signal-identification task (see Fig. 1).

Microprogramming — The implementation of a control function of a processing system as a sequence of control signals stored in a control memory.

Pulse Repetition Frequency — Reciprocal of the pulse repetition interval.

Pulse Repetition Interval — The time period between successive pulses from an emitter.

Signal Sorter — A system which can sort emitters based on their parameters from a multiemitter data stream out of a radar receiver.

Time of Arrival (TOA) — Emitter pulse parameter relating to the time the pulse arrived at the receiving system.

