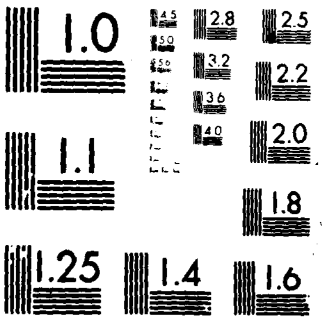

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

A DISTRIBUTED PROCESSOR ARCHITECTURE FOR BMD SIGNAL AND DATA PROCESSING APPLICATIONS

R. L. SOUTH
R. J. PURDY
Group 27

TECHNICAL NOTE 1979-84

21 DECEMBER 1979

LEXINGTON MASSACHUSETTS

ABSTRACT

The signal processor in large Ballistic Missile Defense (BMD) systems includes a subsystem, known as the Post Processor, which selectively reduces the received data to rates which are appropriate for the subsequent general purpose Data Processing System. This report presents an analysis of the processing requirements for the Post Processor and describes a multiprocessor, multibus architecture which appears well matched to the processing and throughput requirements.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION AND SUMMARY

Large Ballistic Missile Defense systems employ a Signal Processing chain which reduces the unprocessed radar data and generates target metrics. These metrics are supplied to a Data Processing System for analysis and decision making. Within the Signal Processor, there are three major stages of processing: Matched Filtering, Parameter Estimation, and Data Reduction. This last function includes such operations as environment mapping, removal of ambiguous responses and coarse removal of spurious signals, a process known as 'Bulk Filtering'. As part of an effort to develop the Advanced Digital Signal Processor (ADSP), Lincoln Laboratory has developed candidate designs for a processing system, known as the Post Processor, which performs the data reduction function. The architecture is a multiprocessor, multibus structure with significant processing capability. This architecture proved to be sufficiently general that, in addition to performing signal processing functions, it also has the potential of performing some data processing functions such as Track Initiate and Target Characterization.

This report describes the basic system environment and discusses and characterizes the algorithms that must be executed. The structure and frequency of the algorithm execution are examined and the nature of the data processing task is assessed. This leads to three major observations: the interfunction time is longer than the interpulse time, some algorithms require multiple transmissions, and some algorithms can be partitioned. Therefore, if functions and portions of functions are distributed to different processors, significant processing performance can be achieved. The processors in this organization are coupled to the radar and each other via a number of high speed data paths with specialized access conventions. This allows the data to flow and to be exchanged in the system in ways which match the needs of each application. These observations lead to the suggestion that a multiprocessor, multibus structure is an appropriate architecture for a Post Processor. The description of the overall organization is followed by a description of an individual processor and by a discussion of the required control and data management. The report continues with a discussion of performance. Additional issues which must be addressed are discussed in the conclusions.

Chapter 2

SYSTEM CONTEXT

Ballistic Missile Defense radar systems have the general structure shown in Figure 1. These systems consist of a Data Processing System (DPS) which provides overall system control and decision making, a control unit which decodes commands and provides the appropriate timing, the radar which generates, transmits, and receives RF energy and the signal processor which provides matched filtering for the received waveforms, generates the necessary target metrics and, in general, selectively reduces the data volume and rate to values which are appropriate for the DPS.

In this report, it is assumed that the signal processor has the basic components of the Advanced Digital Signal Processor* (ADSP), including a Post Processor, as shown in Figure 2.
This overall structure provides a meaningful context within which the performance of the Post Processor architecture can be described and analyzed.

The processing chain in this system consists of Analog to Digital Converters (A/D's), an Input Buffer (IB), a Digital Convolver System (DCS), a Detection Processor (DP), and Output Buffers (OB), followed by the Post Processor with its Report Buffers.

The A/Ds convert the data on three radar channels (SUM, AZ, EL) and deposit the data in an Input Buffer. The IB is the first buffer in the signal processor. It therefore provides the smoothing of the entire radar data stream so that the data can be evenly processed through the remainder of the signal processor. The DCS provides the matched filtering operation as required in radar systems. After the matched filtering operation, the data are sent to the Detection Processor (DP). The Detection Processor provides basic thresholding operations and target parameter estimation (range, velocity, amplitude). All data plus threshold results and the parameter estimates are available to the Post Processor. In addition, the Detection Processor provides the following outputs directly to the Post Processor at the clock rate of the DP.

1. Normalized complex filter and spectrum data received from the convolver.

2. Magnitudes of convolver output data sets.

----

* Purdy, Robert J., et al. "Digital Signal Processor Designs for Radar Applications", Technical Note 1974-58, Lincoln Laboratory, M.I.T., (31 December 1974), DDC No. AD-B001419-L (Vol. 1) and AD-B001420-L (Vol. 2).

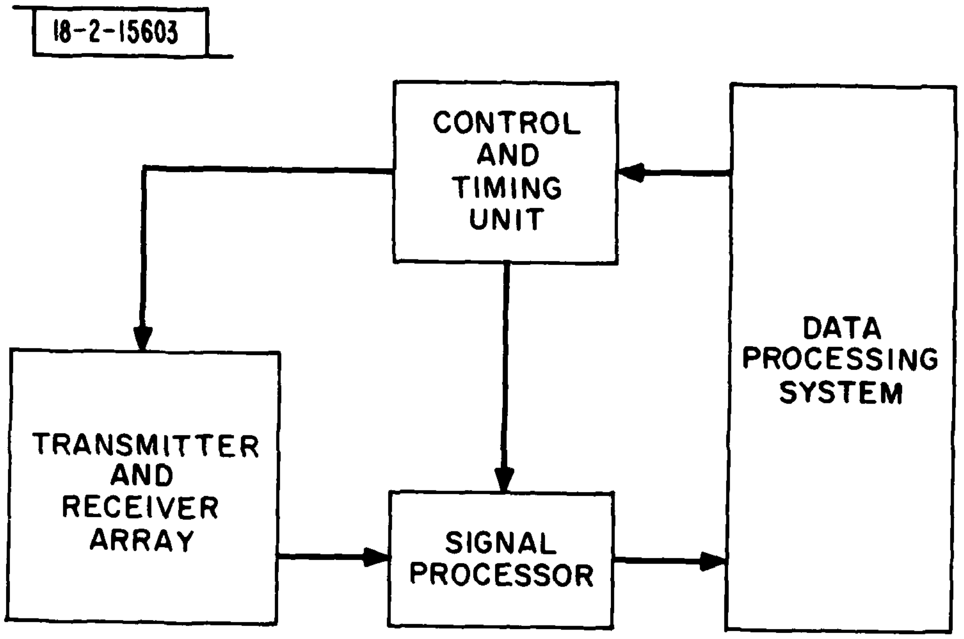

Fig. 1. Radar system block diagram.

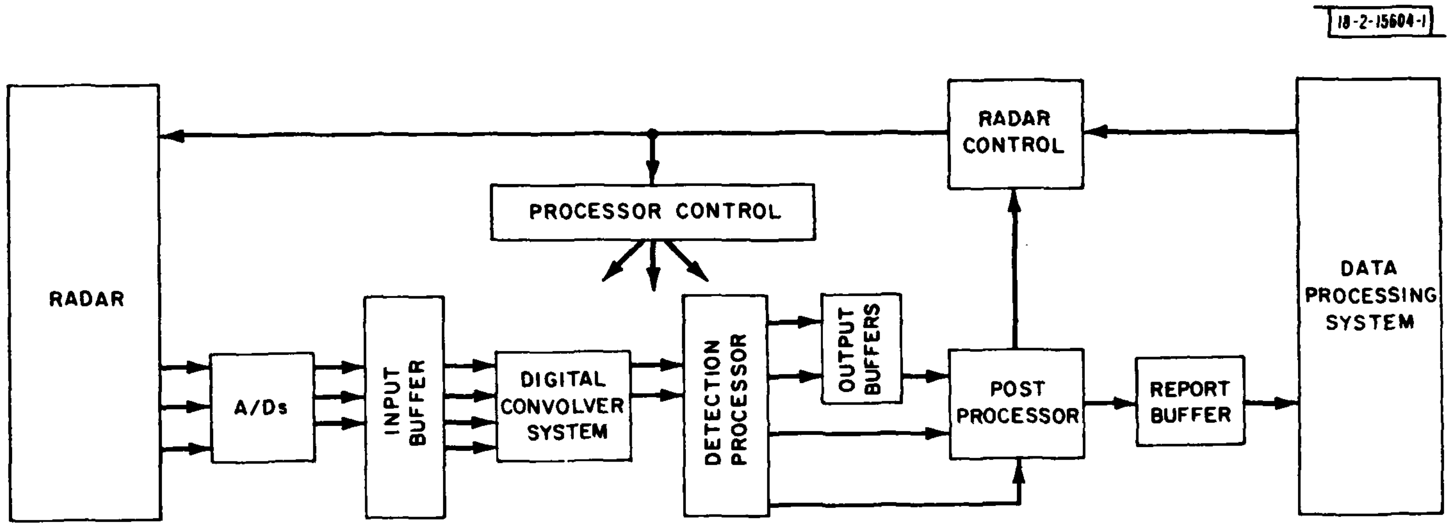

Fig. 2. Signal processor system diagram.

3. Two one-bit threshold comparisons for convolver output data sets.

The Post Processor accepts the output from the DP and executes a wide range of algorithms which develop target and environment metrics. One algorithm, Bulk Filtering, requires an immediate transmission of a verify waveform when a search waveform detects a cluttered environment. Since there is a large time delay through the Data Processing System this requirement necessitates a preemption of a position in the command queue and necessitates the path between the Post Processor and the Radar Control, as indicated in Figure 2.

Although the expression 'Post Processor' is appropriate from the point of view of the ADSP, it functions as a pre-processor from the point of view of the DPS. In this intermediate position in the processing chain, the fundamental processing objective is to reduce selectively the quantity of data. The Post Processor thus provides the interface between the high speed signal processing and the high complexity data processing.

Chapter 3

ALGORITHMS

The Post Processor is required to perform a substantial amount of processing, particularly in an ADSP configuration where much of the data are retained after the matched filtering operation. Some of the algorithms which traditionally have been suggested are listed and described below.

3.1 SPECTRUM ESTIMATION

This algorithm is used to provide a simple estimate of the frequency spectrum. The magnitude of the frequency spectrum (8K-16K elements) is divided into intervals. Within each interval, using only points which exceed a threshold, the sum, mean and variance are calculated.

3.2 CLUTTER MAP

This is a much more stressing algorithm which calculates the same statistics as the Spectrum Estimate but does so over the entire range-Doppler space. As before, the space is divided into range-Doppler intervals and the statistics are calculated for each. However, the computations are done on as much as 64K points (eight, 8K Doppler channels). This algorithm presents a computation problem (because of the volume) and a precision problem (because of the summing). The clutter map algorithm is sufficiently stressing that it may warrant a special hardwired implementation.

3.3 ADAPTIVE FILTER

This operation attempts optimization of the transmitter waveform for clutter suppression in certain time periods in which it is crucial to suppress clutter. It is also these periods in which the Post Processor experiences its heaviest processing load.

3.4 BULK FILTER

The matched filter response for a uniform pulse burst waveform contains a series of ambiguous responses spaced symmetrically about the true response of an object. These ambiguities must be removed to avoid overloading the system with false reports. While the true response has the largest amplitude, it is also possible that noise and particularly interference from other object responses may fall nearby in range. This causes the perceived location in range to shift and the amplitudes to change.

All of the Bulk Filter algorithms developed so far are predicated primarily on the fact that the range ambiguity interval is proportional to the pulse spacing of the burst waveform. Therefore, if two measurements with different pulse spacing (search and verify) are made, the two sets of data can be compared in range, and, in principle, only the true responses in each data set line up in range, as indicated by Figure 3.

While several Bulk Filtering algorithms have been developed, one algorithm, known as the Combined Algorithm, is assumed here because it is computationally stressing. Inputs to the algorithm are lists of target metrics from both search and verify pulses plus the threshold data sets from both measurements. The computation is done in three distinct stages called Group, Edit, and Coincidence. The coincidence stage is, in principle, as described above. The other two stages provide preliminary processing and sorting of the data. All of the Bulk Filtering stages are shown in Figure 4 and described below.

3.4.1 Bulk Filter: Group

Since the range ambiguities of any given object are spaced regularly in range, the first task is to sort responses of the search and verify signals into equivalence classes (groups) based on the range ambiguity interval, as shown in Figure 4a.
This is accomplished by expressing the range location of the responses as kR + i where:

R	is the range ambiguity interval
k	is an integer 0,1,2,3,...
i	is an integer index identifying the range cell relative to the kth ambiguity interval into which a given response falls.

Thus, i classifies an object into a given equivalence class (group) and k identifies the ambiguity interval in which the object lies.

In the absence of noise or clutter, an object and all of its ambiguities would be grouped into the same equivalence class by this process. Within each equivalence class, all of the members are further grouped by range-rate to separate objects which are at the same range (or rather of the same equivalence class) but travelling at different velocities. (Figures 4a, b, and c do not include the range rate dimension. Only single range rate is assumed for these illustrations.)

Within each subgroup (range and range-rate sorted) of each equivalence class, the range index k of the member elements is examined. Where the range index has a breakpoint (in numerical order, k increments by more than one between two elements), the subgroup is divided into smaller subgroups. This

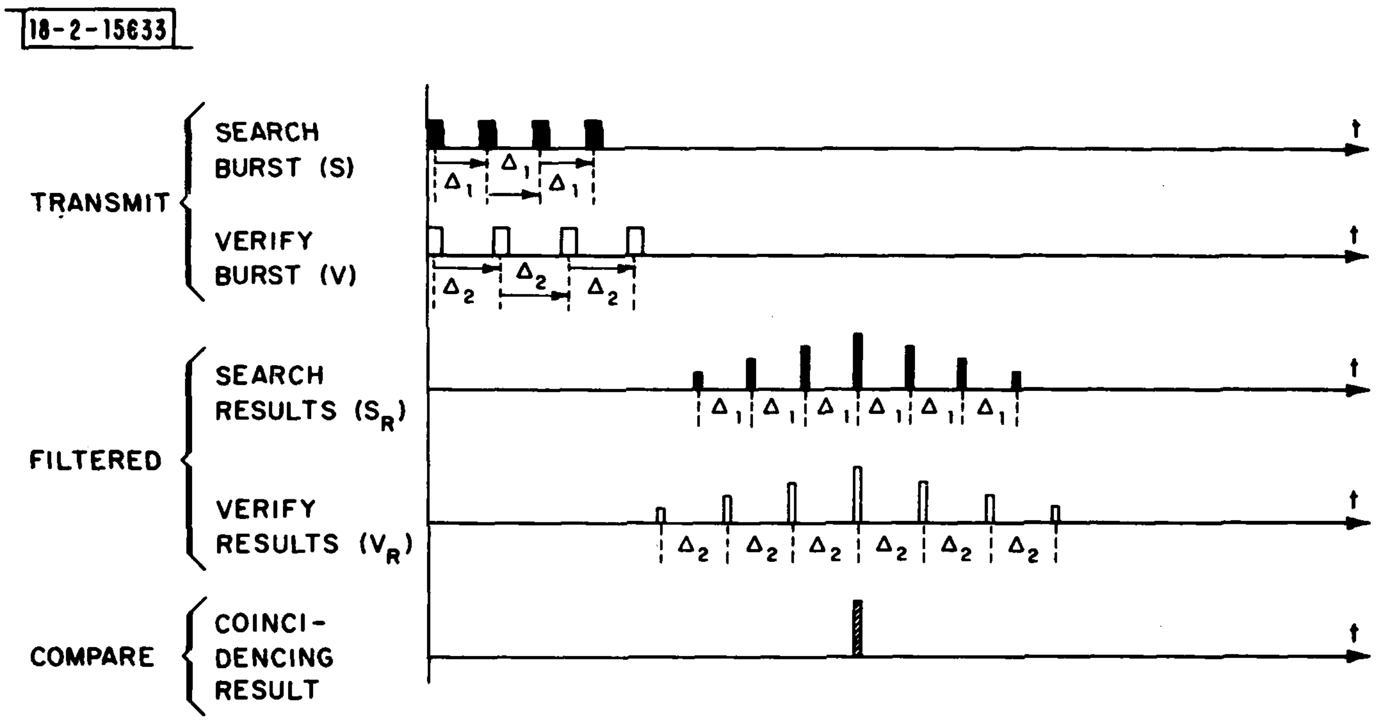

Fig. 3. Bulk filtering algorithm.

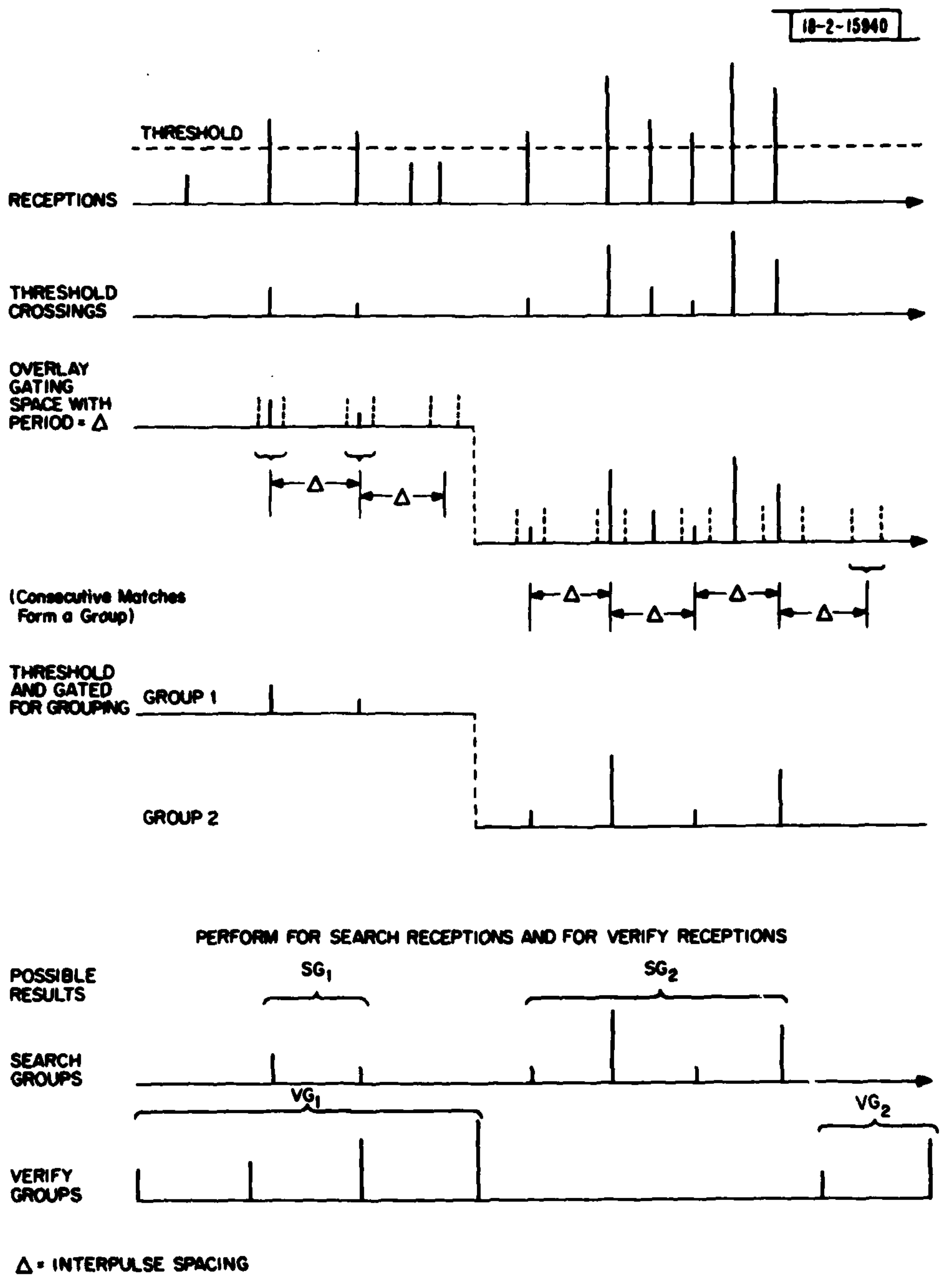
18-2-15940
THRESHOLD
RECEPTIONS
THRESHOLD CROSSINGS
OVERLAY GATING SPACE WITH PERIOD = Δ
Δ
Δ
(Consecutive Matches Form a Group)
Δ
Δ
Δ
Δ
THRESHOLD AND GATED FOR GROUPING
GROUP 1
GROUP 2
PERFORM FOR SEARCH RECEPTIONS AND FOR VERIFY RECEPTIONS
POSSIBLE RESULTS
SG1
SG2
SEARCH GROUPS
VG1
VG2
VERIFY GROUPS
Δ = INTERPULSE SPACING

Fig. 4a. Bulk filter - group.

tends to separate objects of equivalent range and range-rate but different absolute ranges. All subgroups which have fewer than some arbitrary number (a variable parameter) of elements are eliminated. The grouping function is also performed on the verify returns. Sample results are shown at the bottom of Figure 4a.

3.4.2 Bulk Filter: Edit

The subgroups resulting from the Grouping process are regarded as possible objects. Before these subgroups are passed to the coincidence stage that selects one or a few objects as the real objects, they are edited in order to further refine the data base as shown in Figure 4b. In each subgroup, the element with largest amplitude is tentatively selected as the real object (and ideally would be). The remaining objects are discarded and the subgroup is re-populated with ideally offset ambiguities spaced symmetrically about the element selected as the real object. These edited (idealized) groups are then passed to the Coincidence stage.

3.4.3 Bulk Filter: Coincidence

The final stage performs coincidencing between the search and verify pulses and is the basic ambiguity removal step discussed earlier. This step is illustrated in Figure 4c. The ideal subgroups from the verify receptions are matched against the groups of threshold data of the search receptions. Also, the ideal subgroups from search are matched against the verify threshold data. In the coincidence process, a match occurs when any of the threshold crossings in the interval coincide with a member of the threshold data groups. Such a match is a designation and is declared a possible object. The results of both coincidences are merged, as shown at the bottom of Figure 4c. The measured range and range-rate data of the objects are preserved.

3.5 TRACK INITIATE

The Bulk Filter algorithm outputs designations (range, range-rate pairs) to the Track Initiate process. The object of Track Initiate is to collect several observations (typically 2 to 7) of a potential object which are self-consistent as a series of observations of a ballistic trajectory. A very simple constant velocity model has been examined which requires that each new observation be located reasonably close to the range as projected from the last observation. Objects which do not correlate for several consecutive measurements are dropped from the list. Objects which appear to be sufficiently consistent are passed to the Data Processing System where the Tracking algorithm further refines the trajectory.

A good way to implement this algorithm would be to maintain the Track Initiate list in range order (by beam). This requires some local reordering of the list, each time the 'known' objects are projected in range, due to the

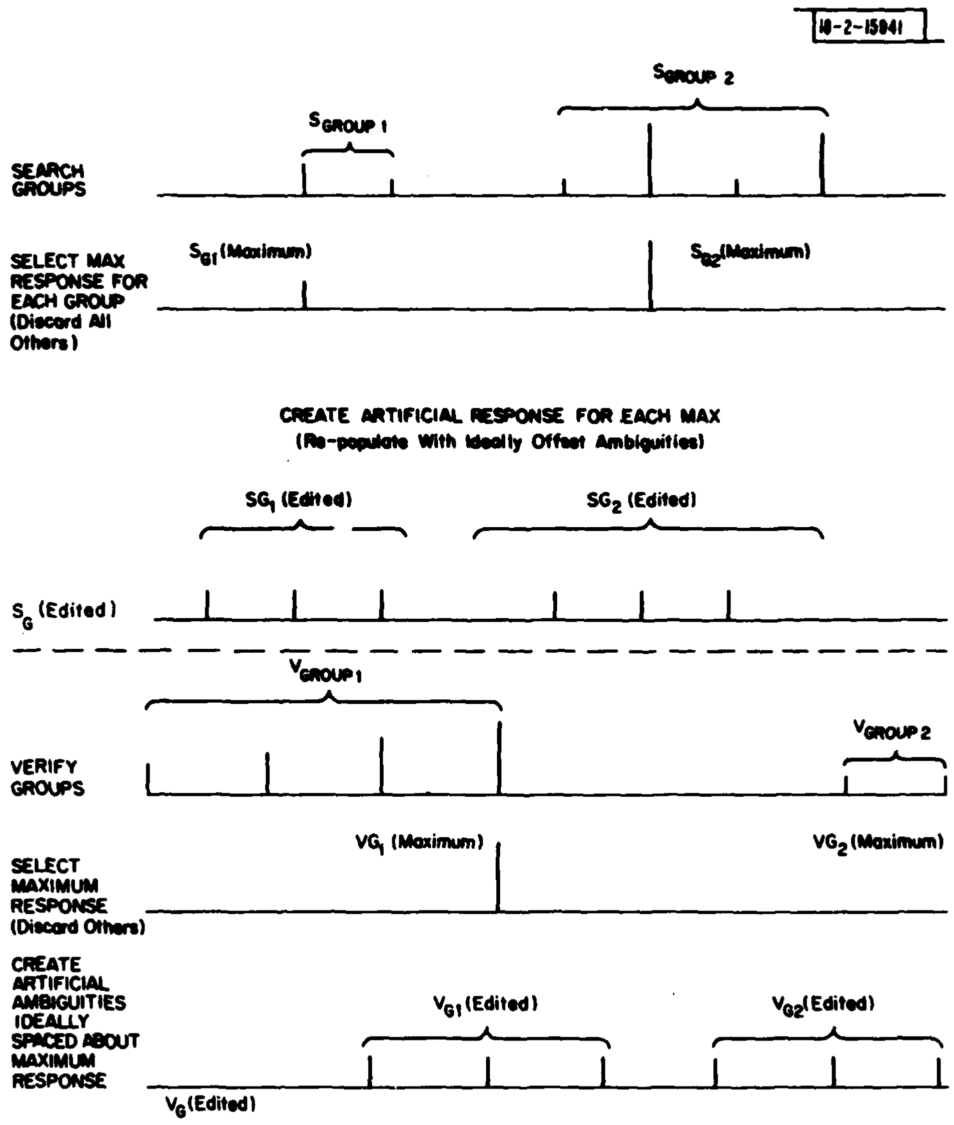

Fig. 4b. Bulk filter - edit.

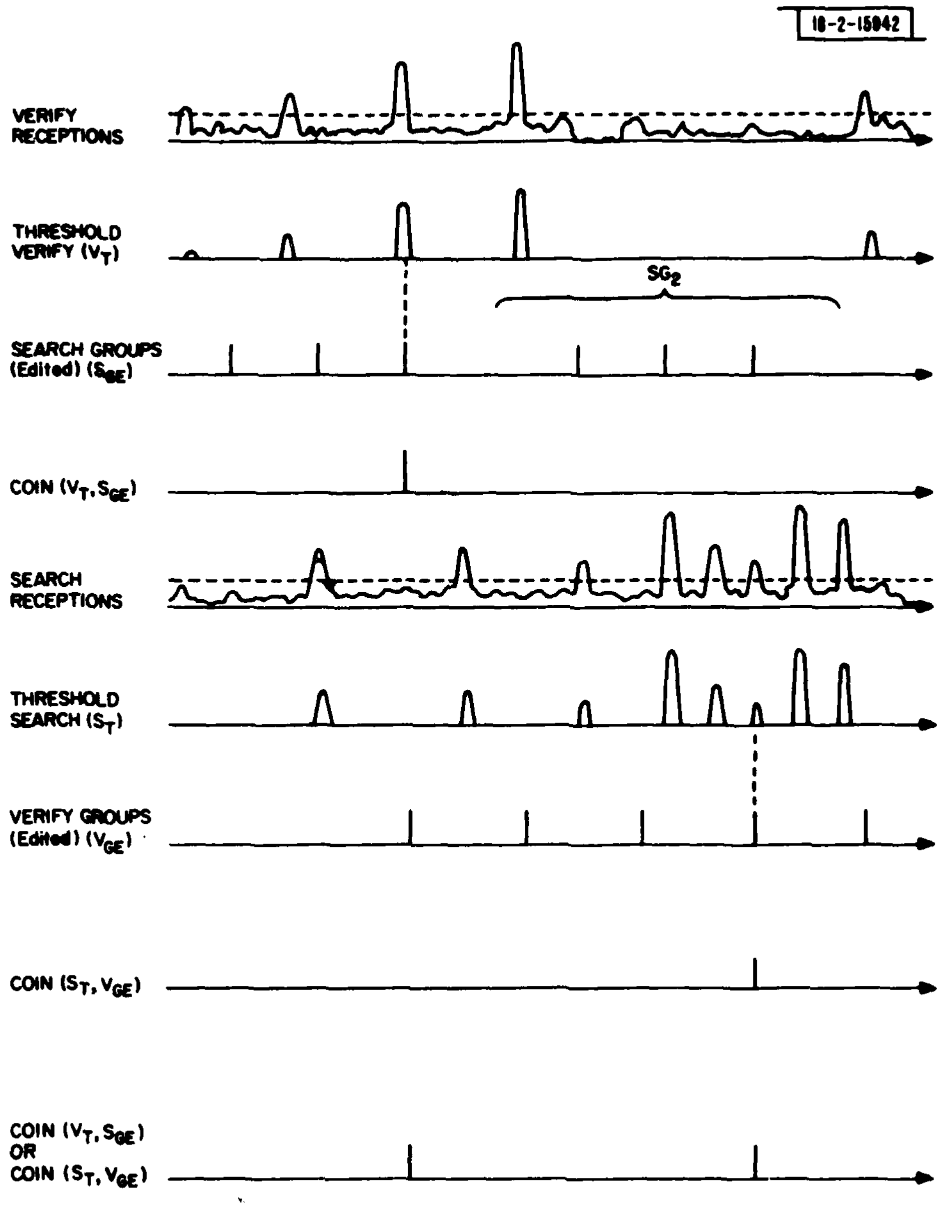

Fig. 4c. Bulk filter - coincidence.

different velocities. The list of designations from the Bulk Filter is range ordered, so the next step is to perform a merge operation. This approach allows a binary search to match designations from the Bulk Filter with objects in the Track Initiate list. The computation is thereby reduced to a manageable level. Additionally, an object may switch beams between any two observations. This would require examination of monopulse data or direction from the DPS.

3.6 TRACK

The Tracking algorithm develops target metrics (spatial location, velocity, acceleration) and discriminants - characteristics which tend to distinguish RV's from clutter and decoys. Monopulse measurements are periodically made on all beams in which there is an item in track. This provides new spatial (range, azimuth, elevation) measurements from which new velocities and acceleration can be calculated. The technique used, Kalman filtering, consists of updating a precision model for a ballistic object with each new set of measurements. This requires integrating equations of motion for an object and performing a series of matrix additions, multiplications and one inversion to arrive at a new state vector and covariance matrix. The covariance matrix measures how well the data fit the model. After the model is updated, additional variables are calculated. These include altitude, velocity and ballistic coefficient. A simpler model, assuming constant velocity, may be used with a higher frequency of target position update. The simple model eliminates the matrix inversion and generally simplifies all other matrix operations (due to the presence of many zero entries). The net computational difference between the two models is approximately an order of magnitude. It is anticipated that the Tracking function should be performed in the DPS.

3.7 SEARCH

Search consists primarily of scanning the list of targets in a given beam for new targets. Potentially threatening objects appearing in a beam give rise to a sequence of measurements designed to accurately track ballistic paths and measure characteristics of these objects. The Post Processor initiates this sequence of measurements on subsequent transmissions by scanning the mark list, reformatting the metric parameters of designated targets, and sending this information to the DPS.

Chapter 4

ARCHITECTURAL IMPLICATIONS

The algorithms discussed above are useful guidelines for developing a sense of the nature of the processing problem. To this end, this section attempts to identify the fundamental nature of the data to be processed and the fundamental structure of the algorithms to be implemented.

4.1 DATA BASE

Some algorithms require multiple transmissions. These processing algorithms operate on a data base which spans some time interval and some number of successive measurements. There are roughly five categories of algorithms.

1. Single pulse operations: Functions for which a single data set is sufficient to perform its function. Examples are Clutter Processing, Adaptive Filtering and Searching.

2. Pulse pair, pulse triplet operations: Algorithms, like the Bulk Filter algorithm, which combine data sets of two or three successive pulses in a beam to produce a single output data set for the beam.

3. Multiple pulse operations: Algorithms which collect data for several successive pulses in a beam before producing a result. The number of pulses used may be data dependent. Examples are Track Initiate and Target Characterization.

4. Pulse sequence operations: Algorithms, such as radar scheduling, which accomplish their task by operating over a long sequence of measurements. This is performed in the Data Processing System.

5. Continuous operations: Processing which runs for an indefinite length of time or for a number of measurements. Examples are: Tracking, Threat Analysis, Engagement Conduct. These also would be performed in the Data Processing System.

There are two major areas where the data base time interval impacts the processing requirements. First, algorithms which have long data base time intervals generally must restrict the amount of saved data. Algorithms which operate on every element of a raw data set quickly utilize very large amounts of storage if they require that the entire data set be saved for each of many successive pulses.

Second, because several algorithms require multiple transmissions, in principle the time taken for all transmissions for a given algorithm is available for the processing of that algorithm. This fact is exploited in the development of requirements for the Post Processor.

More specifically, BMD radars may transmit 2500 pulses per second (PPS) which may be distributed as shown in Figure 5. Thus, there are approximately 625 search pulses, 625 verify pulses, and 625 Bulk Filter calculations per second. Therefore, while the interpulse period is 400 microseconds, the interfunction period is 1600 microseconds.
While traditionally it has been assumed that the processing must be completed in the interpulse period (400 microseconds), the interfunction period (1600 microseconds), is actually available. This observation leads to an important conclusion: Many radar signal processing functions are not repeated with every transmission.

If it is assumed that the scheduler in the DPS distributes the functions or measurements evenly over time, it is possible to use the interfunction time rather than interpulse time to do the processing. Therefore, at any given time, many functions could be processed in parallel on different data sets. This is illustrated in Figure 6 which shows a time line consisting of Search (S), Track (T), Characterization (C) and Verify (V) transmissions and receptions.* The transmissions are shown as upward pulses and the receptions as downward pulses. Thus as shown in Figure 6, at any given time, the three portions of a complete Bulk Filter, a Characterization, and a track initiate can be performed simultaneously.

4.2 STRUCTURE OF THE DATA PROCESSING FUNCTIONS

In this section the structure of the different algorithms will be identified. The basic types are listed below.

4.2.1 Simple Algorithms

Simple algorithms are defined as ordinary, straightforward, single calculation structures. This can mean in-line computation, functional organization, iterations, etc., but it generally implies that a data set is processed as one functional unit of operation.

---

* The assumption of an even and alternating distribution in time, as shown in Figure 6, implies that there exists a trade off between optimizing the post processing function and optimizing the scheduling algorithm in the DPS. This is an important system level trade off which would have to be addressed when implementing an actual system.

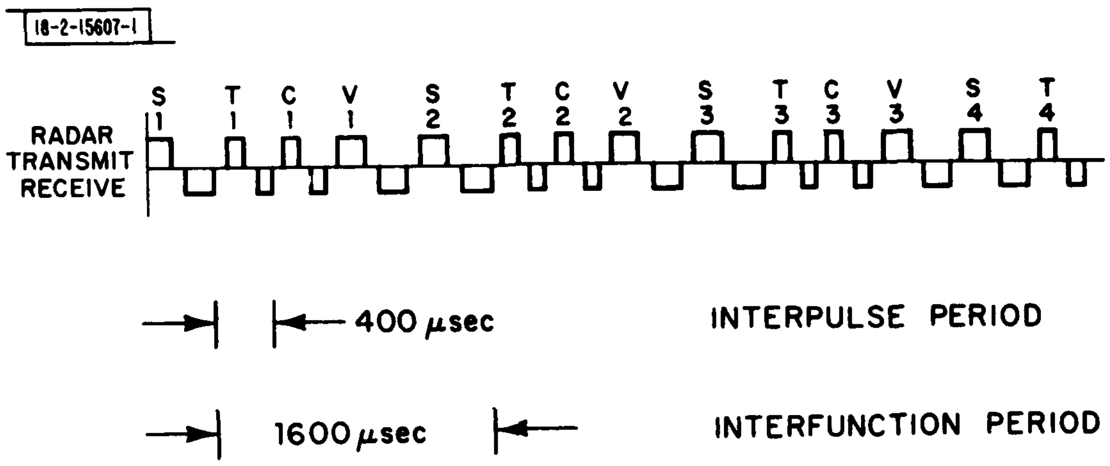

Fig. 5. Processing requirements.

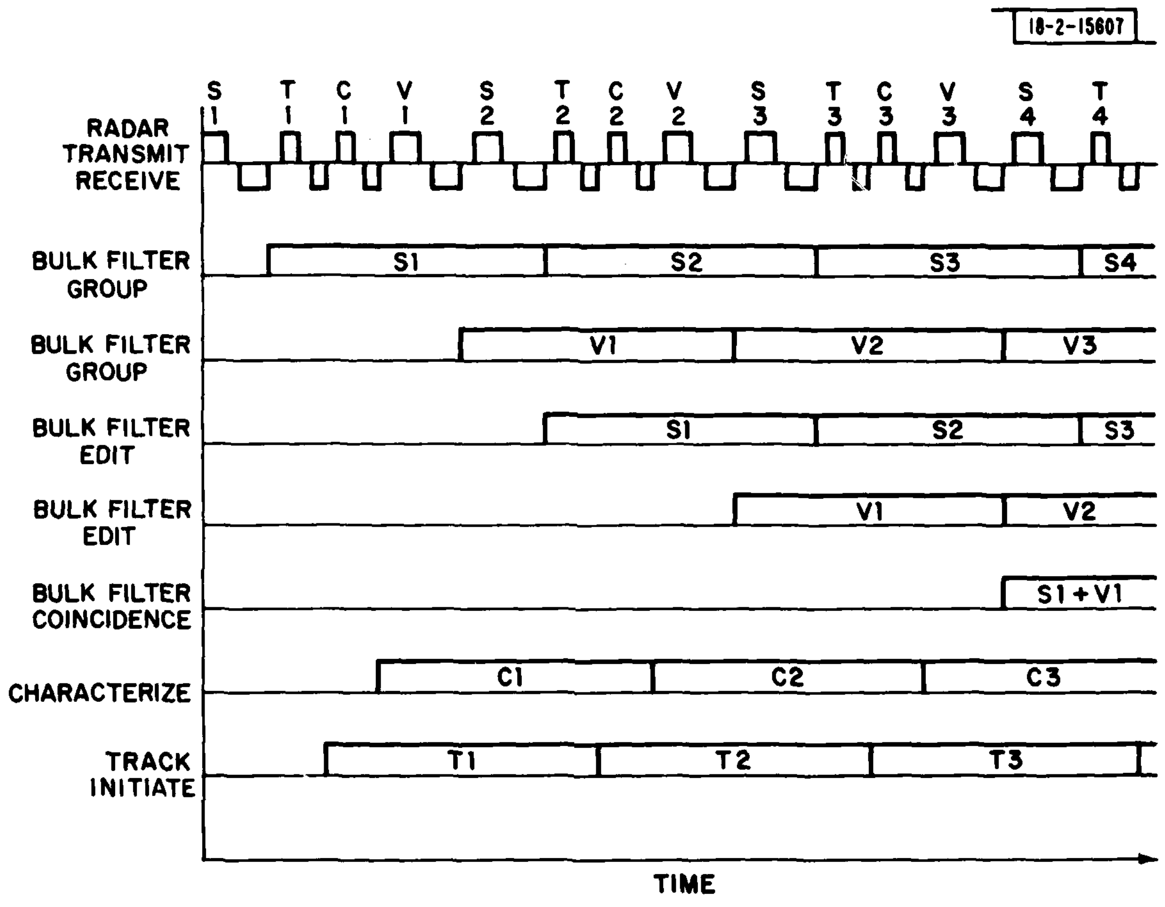

Fig. 6. Processing time line.

4.2.2 Staged Algorithms

Sometimes performance sequences of operations in one pass is very unwieldly. In such cases, the total computation is separated into two or more stages. This type of operation is referred to as a staged or pipelined algorithm. The Bulk Filter is an example of an algorithm that can be pipelined. Thus, the Bulk Filter algorithm has three stages -- Group, Edit, Coincidence -- each of which can be performed in parallel on different data sets.

4.2.3 Parallel Algorithms

Some processes can be parsed into two or more concurrent computations with the results combined later. This is referred to as a parallel algorithm. Matrix operations can frequently be expressed as a combination of the given operation on a set of lower order submatrices.

Since several algorithms can be staged and, equivalently, some have computations which can be performed in parallel, it is possible to improve the throughput for a given algorithm by having several different subsystems in the Post Processor process different portions of an algorithm concurrently. This fact is exploited in the Post Processor design.

4.3 DATA DISTRIBUTION STRUCTURE

The data sets from the radar occur in large (4K - 16K) discrete blocks and each such data set tends to be processed by a different algorithm than its immediate neighbors (in time). This directly suggests that data sets need to be distributed rapidly (because of size) and to multiple locations (because of different processing requirements). Due to the large volume and high data rates, a comprehensive system of high speed busses is required. In the design discussed in this report, this bus system is referred to as the Data Distribution Bus (DDB) and it handles all volume transfers of data within the system. The types and modes of data and control distribution which can be envisioned are listed below:

a. large blocks of data
b. fast, low volume, synchronous data or control
c. moderate volume, asynchronous data or control

4.4 ARCHITECTURE RATIONALE

The basic nature of the data and algorithms provides the rationale for the architecture presented in this report. Specifically, there are three main characteristics, which are summarized as follows:

1. There often is a comparatively long interfunction period.

2. Several algorithms require multiple transmissions.

3. Algorithms often can be partitioned into staged or parallel tasks.

These characteristics provide the basic motivation for suggesting a multi-processor design. Since there exists the possibility of multiple functions and subfunctions operating simultaneously, a multi-processor design appears well suited for the processing problem.

The different sizes and types of data that need to be transferred within the processor, and the consequent different protocol, also suggest that a multiple bus structure is appropriate. The resulting multiprocessor, multibus system is described in the next Section.

Chapter 5

MULTIPROCESSOR STRUCTURE

The Post Processor architecture suggested here is shown in Figures 7 and 8. It consists of a set of processors embedded in a network of data and communication busses. This Section describes the overall interconnection structure, and the next Section describes the structure of an individual processor.

5.1 DATA PATHS

5.1.1 Data Distribution Bus

The Data Distribution Bus (DDB) is a collection of high speed busses which transfer large blocks of data. Each bus is at least one DCS word* wide. Access to the busses is controlled by a single central bus controller, known as the Dispatcher (Figure 8.), which has the ability to connect and disconnect other system components from any individual bus. While only one data source is connected to a bus at any one time, there may be several data sinks connected. This allows distribution of a data set to different processors at the same time. Bus protocol is simple in that it requires confirmation, either explicit or implicit, that all parties to a data set transfer are connected and ready. This is followed by placing data words and a clock (data valid) signal on the bus. There is no confirmation for individual words. In fact, there may be several data words propagating at once down a bus. The controller looks for confirmation from all connected data sinks to determine that the proper number of words are received. Retransmission of the data set to one or more destinations may be required. Transmission of error detection information is a necessary feature to ensure adequate system integrity.

5.1.2 Register File Bus

The Register File Bus provides access to the register sets of all of the processors in the system. Its objective is to provide a very fast interchange on a word basis among cooperating processors for either data or control purposes. Design objectives would be to achieve access times of 15-25 ns. These times

----

* One important factor in processing is the total time required to transfer a data set from one part of a system to another. A single 8K data set requires 234 microseconds to transfer at a 35 MHz rate. The time required to transfer sequentially the DCS output of many 8K Doppler channel data sets over a single channel can be minimized by providing wide busses.

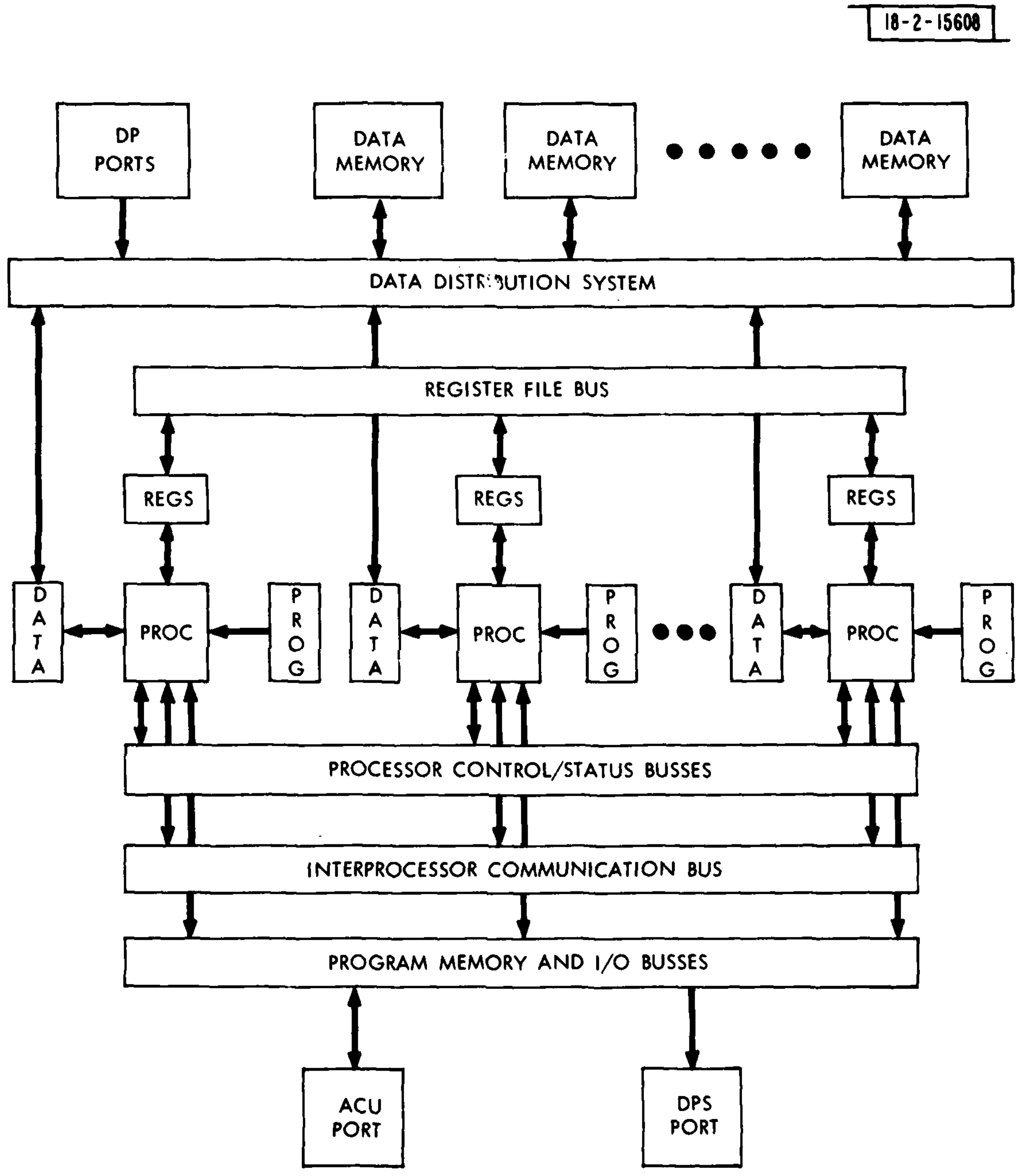

Fig. 7. Multiple bus, multiple processor architecture.

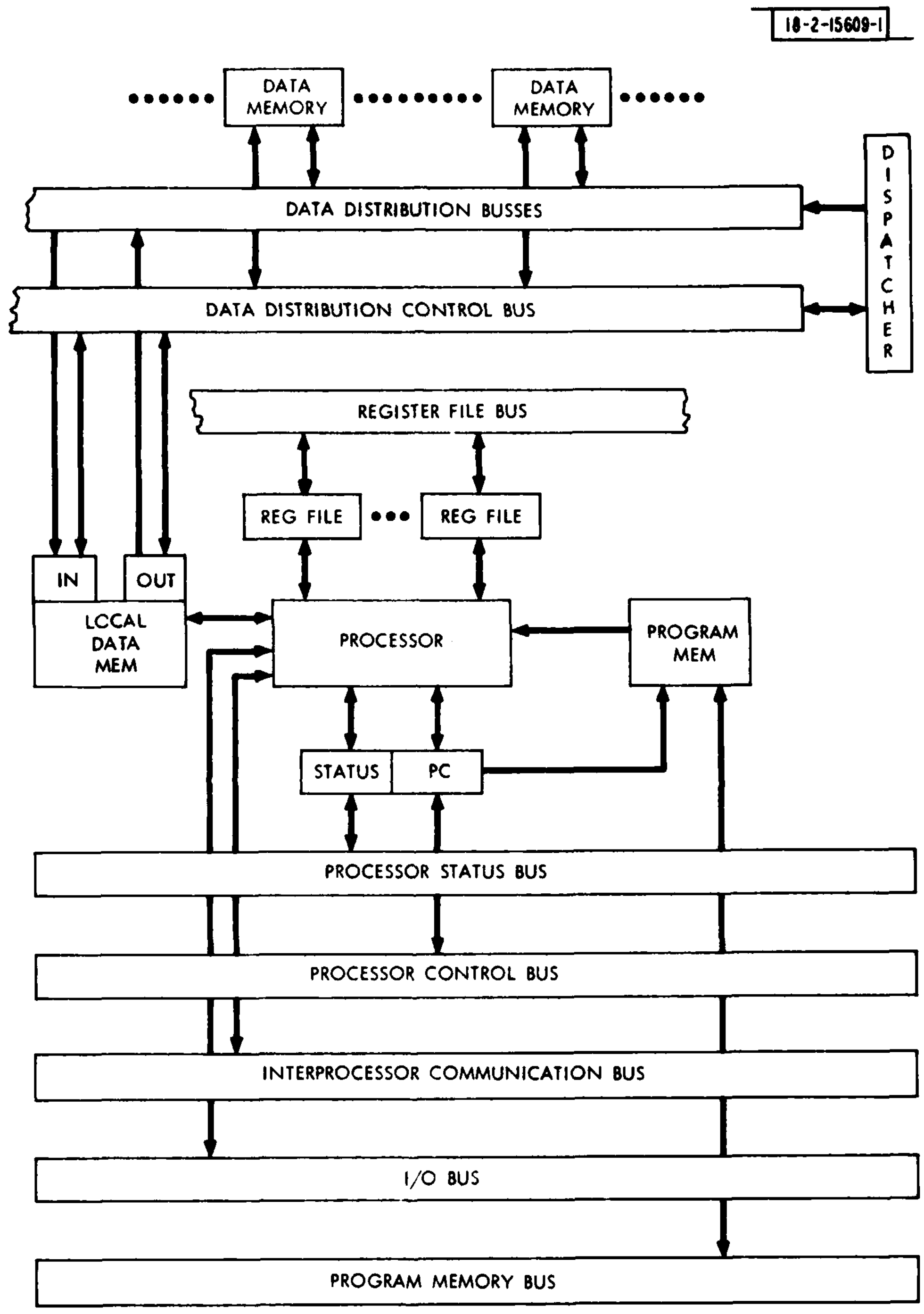

Fig. 8. Slice of the multiple bus, multiple processor architecture.

are, of course, highly sensitive to bus lengths and contention resolution. Addressing consists simply of a processor address plus a register address. Protocol for this bus is not defined at present but an asynchronous approach has some advantages. In an asynchronous design, the initiator would transmit address, data, and a function code to the destination processor/register. Any response would be transmitted back in the same fashion; i.e., as a separate, unrelated operation. The interface of the register files and the bus must not involve the processor itself except insofar as is necessary to allow the processor to make requests. Register file logic must mediate contention between processor access and bus access.

5.1.3 Interprocessor Communication Bus

The Interprocessor Communication Bus is intended to make available to all processors a means for asynchronous transmission of moderate amounts of data. This is used whenever it is not critical for the exchanged data to be synchronized, explicitly acknowledged, or treated on a priority basis by the receiver. The interfacing to the bus is provided via a bi-directional queue manager. Data to be transmitted to other processors is entered into the outbound queue. The queue manager gains access to the bus, signals the destination(s) and transmits the data. The originating processor must be able to determine the status of any message; e.g., Still Waiting to be Sent, Transfer in Progress, Error in Transmission, Unable to Send. Any message not in the queue has been sent successfully by default (point-to-point protocol). The receiving queue manager simply enters the message in its inbound queue where the local processor can access it. Normally, the queues are treated as FIFO's but in some applications it may be desirable to implement the ability to transfer messages out of order.

5.1.4 I/O Bus

The I/O bus is the path to the outside world. It accepts control messages from the control units and transfers processed data to the DPS. It also provides a connection to attached peripheral devices.

5.1.5 Program Memory Bus

The program memory bus loads the program memories of the individual processors. In fact, this function can be handled by the I/O bus and is simply separated and noted for functional clarity.

5.1.6 Processor Control/Status Busses

The control portion of this bus provides the means for control over each individual processor by any one of them. It primarily allows the program location counter to be forced to any desired address and initializes some processor states. This capability permits a control processor to determine the allocation of resources and it allows any processor to acquire subservient processors to implement parallel computations without incurring a protocol overhead.

The status portion allows determination of the state of any processor by another. This includes status bits that are fixed as well as some that are user-defined. The latter allows parallel processors to communicate status which is defined only temporarily by mutual convention.

The bus itself must be fast if it is to be used for close coupling of parallel processors. A synchronous protocol may avoid some control problems. This bus might also be used for diagnostic and monitoring functions.

5.1.7 Data Distribution Control Bus

The Data Distribution Control Bus (DDCB) interconnects the DDB controller and all users of the DDB. All requests for DDB resources and all setup commands are routed through this path.

5.1.8 Data Distribution Controller

Data distribution is envisioned to be a system of busses which are separately managed by a central controller, either a special box or one of the processors. The central controller is referred to as the Dispatcher. Requests for service are sent to the Dispatcher via the DDCB along with sufficient information to enable the Dispatcher to assign a bus or busses and to set up the transfer. The Dispatcher causes all receivers of the data block and the sender to be connected to the selected resources and verifies that the system is ready for transfer of data. A start signal is sent to the requester indicating that the transfer can begin. Error signals are sent to the Dispatcher which, in turn, informs the requester and maintains a history of such events.

The Dispatcher should be constructed as a fast microcoded processor since it must behave in a reasonably intelligent fashion and must be easily modified as experience dictates.

5.2 MEMORIES

5.2.1 Global Data Memories

There might be a need for large memories for buffering large data sets. These would be connected to the DDB and would be both receivers and senders of data.

These memories would be interfaced to the DDB through a manager which accepts requests, queues them, and honors them on a FIFO basis. Requests might include some simple structured access which reorders the data blocks either on input or output. The memory controllers also are candidates for a microcoded processor.

5.2.2 Local Data Memories

5.2.2.1 Multiport Organization

Each processor has its own data memory in order to provide maximum possible memory access speed. The memory unit requires three ports to support overlapped input, output, and processor requests. The memory must be separated into three or more independent modules where each port is connected to a different module. The object of this is to insure that staging of the next data set to be processed and transfer of the results of the computations on the previous data do not interfere with the processing of the current data set.

5.2.2.2 Input/Output Ports

The input and output ports are connected to the DDB and have the capability of reordering data during a data set transfer. Primary control of these ports resides with the processor that is exercising overall system control. The concept removes the burden of managing interprocessor data flow from the processors; i.e., the processors do not have to know their own position within a staged algorithm or the system as a whole. Rather, an individual processor is responsible only for applying its function to data sets which arrive in its input bin, and for placing the results into its output bin. The remaining data flow management responsibilities rest more properly with that processor which is serving as the system control processor.

5.2.2.3 Processor Port

The processor interface to the local data memory is primarily a random access port. Structured data access must be supported but there may be some advantages to having this function reside within the processor. Structured addressing subsystems in the processor generate a stream of memory addresses for read and write operations. Since the processor data access requirements span such a large variety of techniques, it seems appropriate to keep the local memory port simple and to respond to random addressing at maximum possible speed.

5.2.3 Register Files

The registers for this system occupy a low memory address space. This eliminates the need for separate memory and register operand instructions. Although some time penalty is incurred when they are referenced as memory, explicit references to registers should incur no penalty. If enough bits are available in the instruction, the memory delay problem is eliminated since the operand address field is explicitly declared to be of a particular nature.

Register files are internal to the processors but in this system they have a system level interaction because of their connection with the register file bus. The impact of this dual role needs to be examined.

Chapter 6

INDIVIDUAL PROCESSOR DESIGN

The most important feature of the suggested Post Processor design is the availability of data and control communication paths among the individual processors. However, for adequate performance, the individual processors must be organized to promote maximum computation speed. This Section examines the design of the individual processors.

6.1 GENERAL ORGANIZATIONAL CONCEPTS

Data flow and program operation within each processor are important since it is desirable to maximize the amount of useful computation per instruction/data-access. This principle tends to eliminate instructions, which in turn saves time.

Figure 9 shows the principal data paths and logic for an individual processor. Operands are collected from one of many possible sources. Some Lesser Operations (LOP's) are performed on each and the results are placed on the two operand busses A1 and A2. Major function boxes are also connected to A1 and A2. These function boxes use one or both operands as input and place the results on an output bus. The result is further modified (by a LOP) and sent to one of many possible destinations.

A source of particular interest is the structured access boxes. They generate a complex address stream for the local data memory and hold the results in a short FIFO until they are called. Other sources include the register, memory and various other busses. Some intermediate results are held in internal registers and are available as operands for the next instruction.

6.2 LESSER OPERATIONS (LOPS)

LOPS are function modules which perform minor operations on operands before they are delivered to a Major Operation (MOP) module. This saves instructions which do nothing more than a minor operation on an operand to prepare it for a MOP module. Since data in memory may not be in precisely the form required at the moment, programs frequently contain instructions which alter operands. Figure 10 illustrates the general organization of a LOP and shows how various operations are generated. As indicated, an operand can be regarded as a full word or as two independent half words.

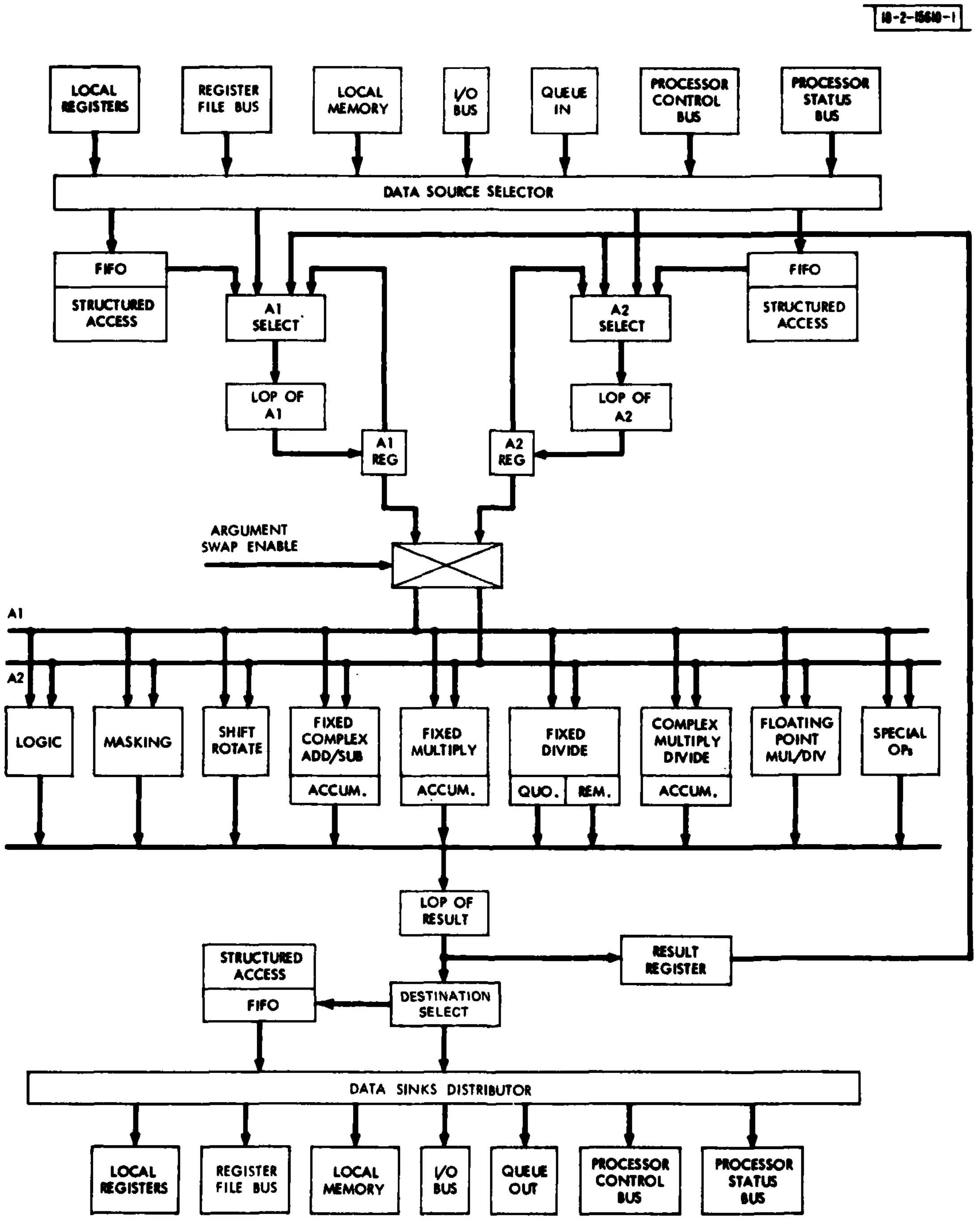

Fig. 9. Individual Processor.

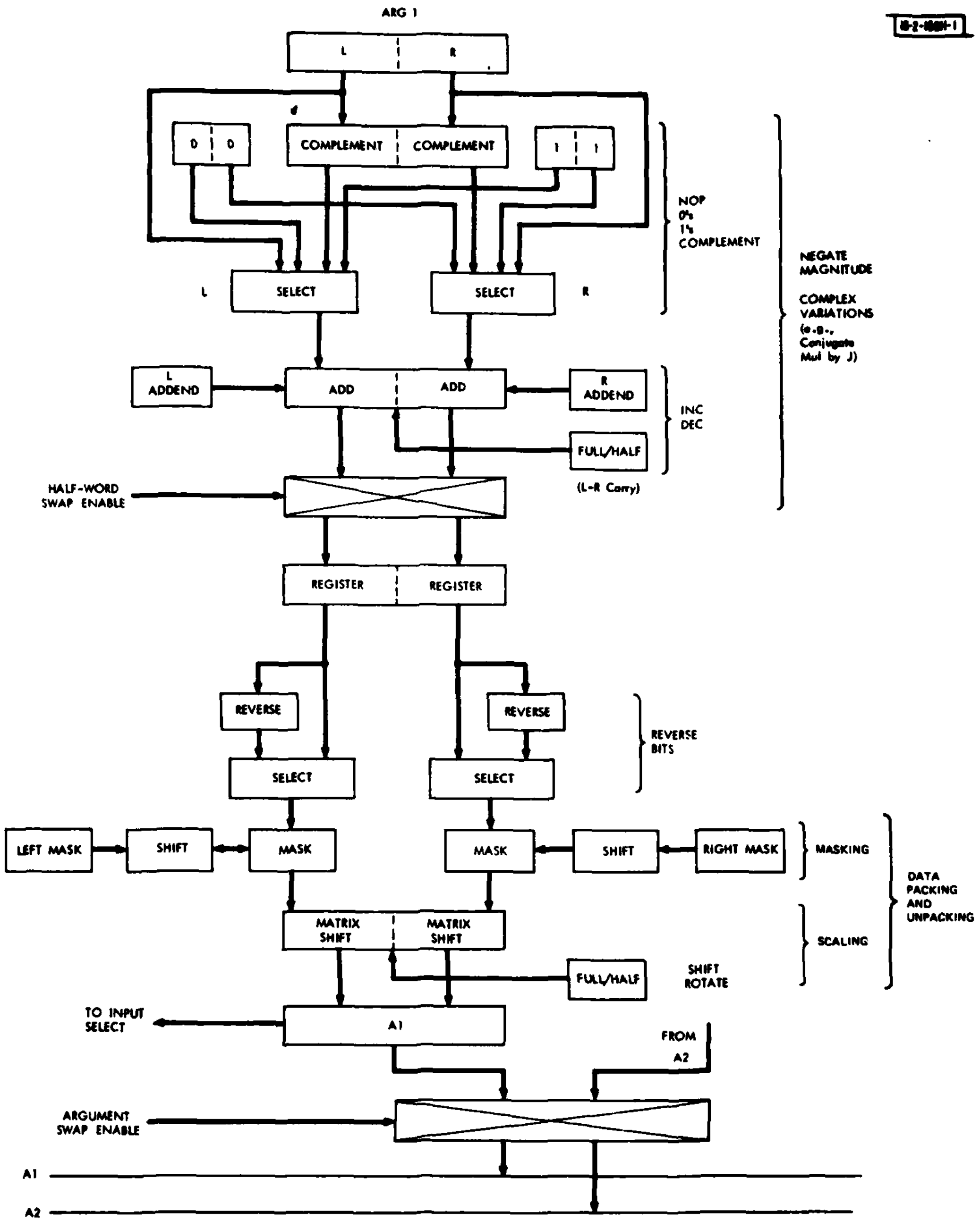

Fig. 10. Single argument processing block diagram.

A LOP can implement a host of useful modifications to an operand. A tentative list is given below:

a) NOP -- deliver the operand unmodified to the argument bus

b) set to zeros

c) set to ones

d) complement (1's complement)

e) negate

f) take the absolute value of

g) complex number modifications; e.g., conjugate, multiply by j, etc.

h) increment/decrement -- can use a subfield from the instruction to implement additions larger than 1

i) reverse bits

j) masking

k) shift, rotate, binary scaling

l) swap argument (not a part of LOP proper) -- for those MOPs which are not symmetric, such as divide.

6.3 MAJOR OPERATIONS (MOPS)

Major operations (MOPS) are those operations normally found in computers, i.e., integer arithmetic, floating point arithmetic, logical operations, etc. They are predominantly two argument functions and generally require more extensive amounts of logic to accomplish than the operations found in LOPS. The structure allows MOPS to be optional; i.e., one can configure any processor with as many or as few of these modules as desired. Thus, the processor can be tailored for particular applications. Each module buffers two operands. This allows some overlapping of the operation with the generation of the next operand pair.

Each is configured to implement a family of operations. For example, a complex arithmetic module performs real arithmetic and possibly double precision real arithmetic.

Listed below are some candidates for major operation modules:

a) NOP -- transfer an operand to the result bus

b) LOGIC -- bit operations such as AND, OR, EXCLUSIVE OR, EQUIVALENCE, etc.

c) MASKING -- manipulation of subfields within a word or multiple word unit

d) SHIFTING -- shift and rotate functions

e) FIXED POINT ARITHMETIC -- both complex and real operations. This could be separated into three different modules such as add/subtract, multiply, and divide. It is likely that this module would have an accumulator.

f) FLOATING POINT ARITHMETIC -- both complex and real operations. These also can be separated into three smaller modules and has an accumulator.

g) SPECIAL OPERATIONS -- special, application dependent functions, such as square root, trigonometric functions, etc.

Chapter 7

DATA MANAGEMENT, ACCESS, AND SYSTEM CONTROL

The problems of data handling and system control dominate the design of a distributed processor system. Data handling exterior to processors is referred to as data management and data handling interior to processors is referred to as data access. Overall system control is also discussed in this Section.

7.1 DATA MANAGEMENT

A single raw data set from the radar must be tagged so that, as it flows through the full signal processing system, the data set and processed results can be properly identified. Control information from the control unit, indicating which algorithm processes which data sets, are interpreted so that data can be transmitted to one or more destinations. More than one independent algorithm might require the same data set.

Another task which must be performed by the data management involves mapping or rearrangement of a data set into a form optimal for data access problems at the individual processor. This may require passing a data set through an intermediate processor or, in some cases, it might be handled locally by some very simple structured access logic. Data management also must contend with the transfer of processor output to other destinations.

The objective of an exterior structure of data management and control (which may be directed by one of the processors) is to offload from the individual processors those tasks which are normally very time-consuming. The implementation of this data management scheme requires several busses for passing control information throughout the system along with several independent data busses for providing access to all processors. Some busses must be more than one word wide to reduce the total time required to transfer the large data sets. This problem becomes less severe after the data have been reduced (e.g., after Bulk Filtering).

As an example of Data Management, Figure 11 shows the flow for the Bulk Filter algorithm.
The Figure shows a three-stage Bulk Filter algorithm with data and control information exchanged between processors. As pictured, the first (search) pulse of the search-verify pair is transferred to the local data memory of the grouping stage processor while the verify pulse data is being transferred (over a separate bus) to a global memory. The verify pulse data are transferred to the grouping processor memory while the search pulse is being processed. When the grouping processor completes the search pulse processing, it initiates a block transfer to the local memory of the edit processor and then

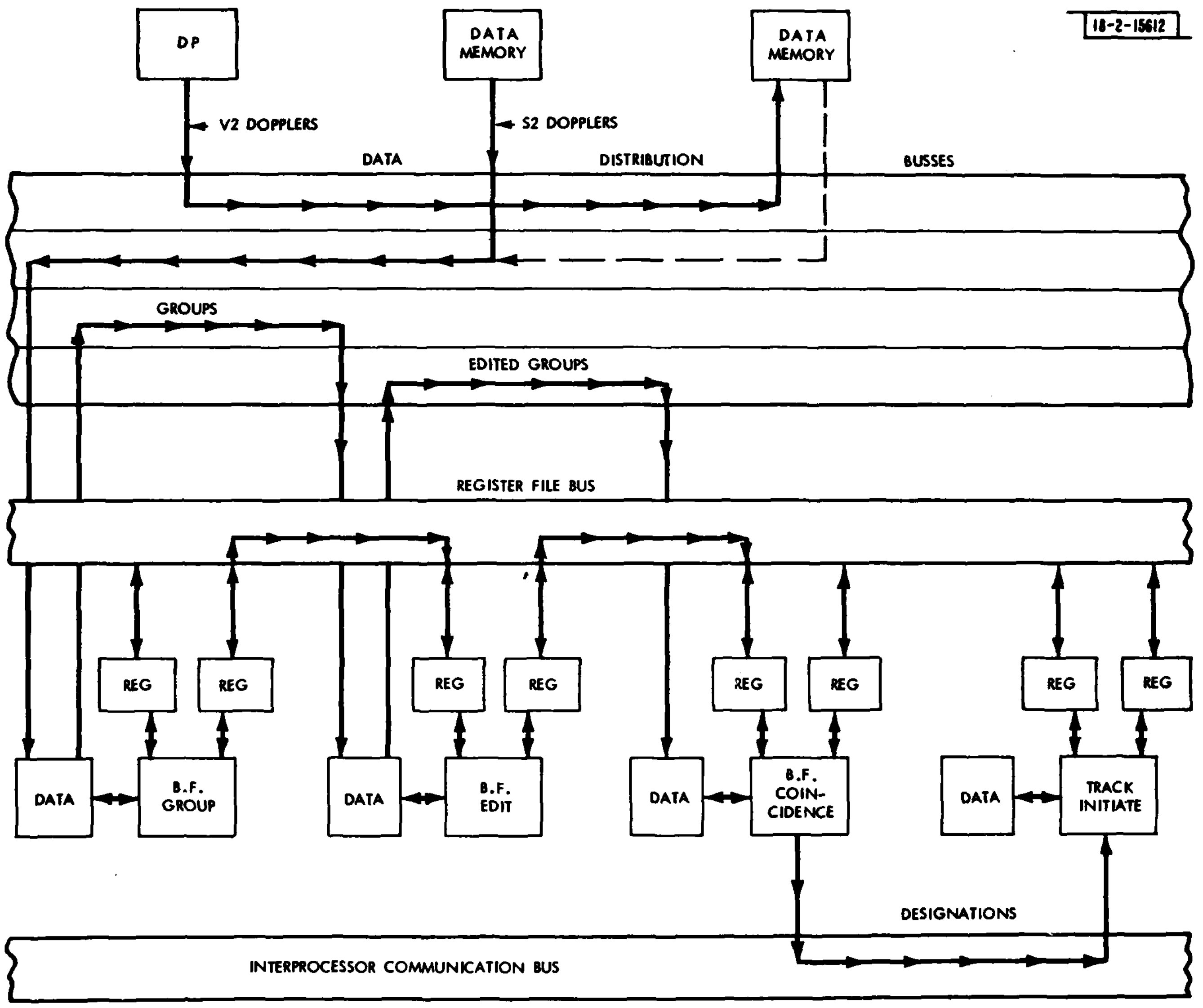

Fig. 11. Processing example.

directs its attention to processing the verify pulse data. The transfer of control data via the register file bus is also shown. The editing processor initiates a block transfer of the edited groups to the local memory of the coincidence stage. Meanwhile, the verify pulse groups are transferred into the edit processor local memory. Not shown is the direct transfer of the threshold data sets of both search and verify pulses to the coincidence processor. These data sets arrive long before the transfer of the edited data groups. The coincidence processor then matches the search groups with the verify threshold results. The resulting list of designations are sent via the asynchronous communication bus to the track initiate processor for correlation with previous designations.

7.2 DATA ACCESS

Data access is the term applied to the local data handling problems of an individual processor. These problems include supplying operands, dealing with intermediate results, outputting results, exchanging interprocessor information, etc.

7.2.1 Structured Data Access

There are numerous calculations which operate on a stream of data which is extracted from a data set according to some regular process. The address sequence, therefore, is algorithm dependent, not data dependent. Simple logic can be utilized to implement several such common structured addressing sequences and hence can be very effectively utilized as an operand lookahead. Some suggested forms of access which have applicability to the Post Processor Function problem are listed below.

1. sequential addresses
   a. 1D subarray (from a multidimensional array):

   address(k) = kI + A

   where
   - A = array base address
   - I = address increment
     (as an example, I equals 1 if accessing a column in a column-stored array)
   - k = sequence index

   This is implemented by initializing a register with A and simply adding the increment I successively.

   b. 2D subarray:

address(k) = kI1 + jI2 + A

where k ranges over some interval for each value of j. Note that this case reduces to a series of one dimensional sequences where A is replaced by A + jI2.

2. indirect structured -- The word accessed in either of the above modes is the address of the data word.

The structured access controller would fill a FIFO (or empty one if used for destination storage) while the program would retrieve each successive operand by selecting the controller as an operand source. Note that two or more such controllers provide some very powerful processing modes in which the main processor logic does not have to expend its resources performing complex address calculations.

7.3 SERIAL BIT STRING PROCESSING

The Detection Processor provides several threshold comparisons as packed words where each bit represents a test of a corresponding word from the data set. The algorithms which use this data do so in two principal ways.

a) Random -- select a single bit from a large sequence. If done in hardware, this requires the parsing of addresses into word (32 bits) and bit address pieces, then either the shifting of the selected word or the selection of the proper bit.

   The selected bit's neighbors are often examined, and this requires a more complex capability. The ability to extract a random, arbitrary length substring from a large set is a desirable feature.

b) Sequential --scan the whole set, step one or more bits at a time, perform a simple test on each substring. This can be implemented by providing the structured access controller with bit string parsing capabilities.

7.4 PROGRAM CONTROL

Program control is, in essence, either conditional or unconditional execution of program segments. Conditional execution is either data dependent or event dependent. Flags and other status data provide a means for saving data and event conditions for later testing. In most computers, conditional skips and jumps provide the required alteration in instruction sequence. Conditional execution of other types of instructions may be done also.

Two special forms of control are subroutine calls and iterations. Overlap of instruction fetches, operand collection, execution, and result storing, are the common techniques that gain processor throughput. Unfortunately, conditional program branches perturb this significantly as they represent a fork, frequently with no means for determining a-priori which path is taken. Thus, the processor pipe empties while waiting for the branch condition to be settled.

Some processors look ahead on both paths of a fork -- a technique which doubles the resource use of the lookahead. The approach taken is to look for common situations where the knowledge of the programmer can be utilized to aid the processor in handling conditional branches.

Many iterations are controlled by a precomputed number (the iteration limit) or a constant. This number is used solely to count the times through a program segment (frequently incrementing an index register for each pass and falling through when the count goes to zero). The overhead represented by the iteration control instruction including the break in the processor pipe is simply a function of the length of the program loop. The percentage of time spent on a single decrement and branch instruction controlling a fifty instruction loop may be of little concern. However, for a five to ten instruction loop, the percentage becomes of interest. There are many common calculations for which this is the case. Examples are matrix operations, FFT butterflies, integration, etc. For large data sets, these kinds of calculations require very large computation times, and frequently, ninety percent (or more) is spent in a short loop.

To help this situation, a loop control instruction is proposed which enables appropriate logic to automatically fetch instructions from a declared interval of memory, maintains an index register and checks the termination condition. Index modification can be done prior to the next pass through the loop and buffered until the new value is needed. Alternatively, the loop could use a structured access controller to provide a data stream -- the latter is simpler than an indexed access and more effective. The index register can still be utilized for situations where the iteration count itself is data.

Dealing effectively with simple conditional executions such as instruction skips is not particularly easy, especially since they frequently occur as a sequence implementing a short decision tree. Single instruction skips can be implemented very effectively, albeit at the expense of program memory, by embedding conditional execution as part of every instruction. This could be as simple as a three bit subfield which defines all tests of the arithmetic comparison type, or a larger subfield which is used in explicitly performing a test prior to execution of the instruction. These possibilities must be examined more closely for the best tradeoff in instruction size vs benefits. In any case, all forms of branching, including subroutine calls and returns, must have conditional execution subfields. In addition, the ability both to branch on existing condition codes and to perform explicitly a branch test must be implemented.

At a detailed level of the system, some programs may be implemented in microcode. The same technique can also be applied at the macroinstruction level to provide some complex operations. This technique is different from simply writing functions since it requires a more highly structured presentation of arguments to the function. However, it provides concise expression for some multiple instruction operations. Examples include vector additions, dot product, etc. with multiple element vectors.

7.5 SYSTEM CONTROL

There are two dominant concepts in the system architecture: data distribution and system control. Data distribution is described above. System control deals with the problems that range from the control of individual algorithms to the integration of multiple algorithms into a cohesive and functional unit.

At the algorithm level, it is noted that many functions can be either staged (split into multiple consecutive processing stages) or processed in parallel (split into multiple concurrent processing elements). Staging is supported through a variety of data communication paths, each optimized for a different kind of exchange. Paralleling is supported by very fast data and control paths which allow any processor to offload a portion of its computation to another on a very low overhead basis. Any algorithm starting on a given processor can cause other processors to support it in implementing either a staged or parallel process.

At the system level, the System Resource Manager (one of the processors) uses the same hardware facilities to control tasks (algorithms) in the system. It is responsible for initiating the correct set of algorithms as the need arises, routing the proper data sets to each task and transferring the results of a task to the right destination. Note that the individual algorithm, need not know about these details. The System Resource Manager must know only that a given algorithm requires certain input data sets, some number of processors and, perhaps, output data formatting and destination control. It will not need to know the details of the local structuring/control of a processing subgroup. An interesting aspect of this architecture is that the System Resource Manager can, as load demands and available resources permit, initiate different implementations of the same algorithms, i.e., it could select a single processor version when that suffices or a multiple processor version when required. Furthermore, the System Resource Manager can initiate more than one task of the same kind for very heavy demand. Program memories can be preloaded with several different algorithms or smaller memories can be loaded in real time from a bulk memory for more economical configurations.

7.6 DATA TYPES

7.6.1 Numbers

The Digital Convolver System outputs complex numbers in a specialized 27 bit hybrid floating point format. There is a single 5 bit exponent and two 11 bit mantissas. For purposes of post processing, complex numbers might be mapped into a symmetric form where both real and imaginary parts are identical in structure.

Possible candidates are:

a) 7 bit exponent, 9 bit mantissa which yields a 32 bit form.

b) 8 bit exponent, 12 bit mantissa which yields a 40 bit form.

c) 8 bit exponent, 16 bit mantissa yielding a 48 bit form. This would provide the maximum useful precision for radar processing requirements.

The following are proposed as a set of data types to be supported by hardware logic:

a) Fixed point real numbers

   1. Integers, 16 and 32 bit

   2. Fractions, 16 and 32 bit

b) Floating point real numbers

   1. 24 bit form: 8 bit exponent, 16 bit mantissa

   2. 32 bit form: 8 bit exponent, 24 bit mantissa

c) Complex numbers (assuming symmetric real and imaginary components)

   1. 32 bit fractional form: 16 bit fractions

   2. 48 bit floating form: 8 bit exponents, 16 bit mantissas

7.7 DATA STRUCTURES

Besides individual numbers, which are used in numerical calculations, there are other forms of data which are derived by imposing appropriate structure on sets of numbers. Examples are matrices, list structures, and strings which impose a structure on sets of numbers, as opposed to individual numbers in the set. Special hardware designed for specific complicated data structures is probably not justified since the structured access subsystem will provide significant and probably sufficient capability. The Post Processor must accomodate these data structures. Therefore it must be determined whether it would be cost effective to provide hardware support for these types of structures. Hardware support implies the ability of the instruction set to handle these forms as another, more complex, type of number.

The structured access subsystems proposed in the Post Processor design allow for handling of structured data but this is not the same as being able to issue, for example, a matrix multiply instruction. Complex data types are probably not justifiable for the Post Processor application whereas the structured access boxes are justifiable.

7.8 DATA MAPPING PROCESSORS

The Post Processor must be able to accomodate a large volume of data transfer. The original input form of an input data set is not always in the required form or the most computationally efficient form for a given algorithm. This suggests implementation of simple processors for the purpose of mapping data sets from a given form into a form required by the algorithm. This results in lower overall computation time when the process is properly staged. Mapping includes data conversion, restructuring and reordering. These processors must also format the output data prior to transfer to the DPS.

Chapter 8

PERFORMANCE

The distributed processing architecture suggested for the Post Processing function exploits long interfunction periods and the sequential nature of many algorithms to yield a system with the potential for high data processing and throughput rates. It is important to examine the Post Processor performance within the radar environment to determine whether this potential can be realized. It is not possible to develop a preliminary assessment using a real system design since a detailed hardware design has not been performed. However if some nominal values for hardware timing are assumed, it is possible to make a paper estimate of system performance. Specifically, the following processing parameters are achievable with ECL 10K and 100K technology and are used in the subsequent calculations.

a) 50 ns instruction time (net, assuming that fetch, process, and return functions are overlapped)

b) 50 ns memory cycle (random access) for local data memory

c) 50 ns 16 bit multiply

d) 150 ns 16 bit divide

Some of the candidate algorithms discussed earlier were partitioned into adds, multiplies, memory access, overhead, etc. The processing times listed above allow the calculation of computation times for these algorithms. In the Tables that follow, a data set size and timing budget is indicated for each algorithm. These parameters are then used to determine the indicated processing times.

TABLE 1
TRACK INITIATE

Data Set Size: 100 designations from the Bulk Filter
150 objects in the beam track list

Process: Each designation must be compared with the projected range of objects in the track list. A pass is made through the track list to update the projected range to correspond to the time base for the Bulk Filter designations. In this process the object list is kept in range order. Due to different velocities some local reordering of the list will occur. The designation list (which is already ordered by range and range rate) can be efficiently merged with the object list.

Timing Budget:

Object list update:

250 ns	Project new range (rate*T + range)
200 ns	Perform local range ordering
200 ns	Overhead

650 ns * 150 = 97.5 microseconds

Merge:

2000 ns (avg)	Search object list for range, range-rate slot per designation
500 ns	Add to object or create new entry
300 ns	Output object if good enough

2800 ns * 100 =280.0 microseconds

Total =377.5 microseconds (use 380)

TABLE 2

CLUTTER PROCESSING

Data Set Size: 9*7384 = 66456
(includes spectral estimate)

Process: Basic timing is completely dominated by the calculation of SUM of X and SUM of X^2 on each bin. This is estimated by assuming structured access for source data plus automatic iteration control. Loop consists of two instructions:

RMADS <SRC>, R2	Cumulative multiply source times itself, send results to R2
ADD <ARG1>, R1	Add previous arg to R1

At end of loop: R1 has SUM of X, R2 has SUM of X^2. This loop is performed for each bin. Since the bin size is very large (e.g., 66456/16 = 4153 elements per bin), the overhead associated with loop setup and saving results is negligible.

Timing Budget:
<100 ns per iteration> * 66456
= *6645.6 microseconds/pulse*

TABLE 3
SEARCH

Data set size: 512 groups of 4 words each.

Process: Reorder data fields to format desired by DPS. Assume structured data access to get groups in order plus automatic iteration control:

MOVE 4 words to register	200 ns
several byte operations (budget for formatting)	600 ns
store group	200 ns
	1000 ns loop

Timing Budget:

<1000 ns> * 512 =	512 microseconds
60 inst. overhead	3 microseconds
	515 microseconds

TABLE 4
COMBINED BULK FILTER: GROUP

Data Set Size: 512 marks (range, range rate pairs)

Process: The approach is to classify each mark and insert in proper group. A linear list of list heads (1 per range group) is used with subgrouping by range rate. Each list head points to a binary tree list structure.

Timing Budget:

50 ns	Move range (X)to Rl
150 ns	Integer divide Rl by range ambiguity interval
50 ns	Store (Rl, Rl+1) pair in RCLASS, RCLASS+1 Rl = range interval Rl+1 = group ID
100 ns	Move proper pointer to R2
50 ns	Jump if R2 not zero to BULK1, check adjacent range groups for empty and go to BULK1 if true Otherwise, create new list
400*50%= 200 ns	SUBTOTAL

TABLE 4 (CONTINUED)
COMBINED BULK FILTER: GROUP

Timing Budget (continued):

```
 150  BULK1:     Compare proper pointer
                 to range + 1
                 check range rate
          150 ns
  50             Jump equal to BULK2
 150             Move proper pointer to R2
 _______
 200*50%= 100 ns SUBTOTAL
           50 ns Jump not end of list to BULK1
                 Create new subgroup
                 (equivalent to append)
 200  BULK2:     Move proper pointer to R$^2$
 200             Append new element to
                 subgroup R2
  50             Loop
 ____________
          450 ns SUBTOTAL
 ____________
   512 * 1350 ns  = 588.800 microsec
 arbitrary overhead 100.000 microsec
                    _______
                    688.800 microsec./pulse
                      or approximately
                     1400 microsec./pulse pair
```

TABLE 5
COMBINED BULK FILTER: EDIT

Data Set Size: Estimate 150 groups
(from Grouping)

Timing Budget:

Scan group overhead		1500 ns/group
Split at discontinuity	(200*50%)	100 ns/group
Flush small groups	(200*50%)	100 ns/group
		1700 ns
		(for 150 groups)

Estimate that 100 groups after flushing operation, avg. 5 elements

Find maximum sigma	600 ns/group
Idealize groups with 9 ambiguities	800 ns/group
	1400 ns/group

```
Pulse time est:  150*1700 ns + 100*1400 ns
                        = 395 microseconds/pulse
     Overhead:             80 microseconds
     Pulse Budget:        475 microseconds


     pulse pair budget:   950 microseconds
```

TABLE 6
COMBINED BULK FILTER: COINCIDENCE

```
Data Set Size:  900 ambiguities
              + 7384 threshold bits
Process:  For each ambiguity, check for any bits set in corresponding
           range interval of threshold data.


          150  divide ambiguity range by word size
          200  get double word and shift by remainder of division
               test masked interval for non-zero
           50  go to loopend on zero
           20 (200*10%) output range, range rate pair if not zero
           50  loop
          ____
          470 ns  per element


           <470 ns>*900 =    423 microseconds per group
           overhead           27 microseconds
                            ____
                             450 microseconds per pulse
                             or
                             900 microseconds/pulse pair
```

The resulting computation times and the assumed data set sizes are summarized below.

TABLE 7
PERFORMANCE ANALYSIS RESULTS

ALGORITHM	DATA SET SIZE IN WORDS	PROCESS TIME IN MICROSEC	PC/S **
Clutter Proc	9*7384=66456	6645	150
Search	512	515	1942
B.F.-Group	2*512	1400	714
B.F.-Edit	2*512	950	1053
B.F.-Coin.	9*100=900 Ambiguities/Pulse	900	1111
Track Initiate	100 Designations From B.F. 150 Object list	380	2632

** Potential Calculations/Second

These results allow several conclusions. First, the Clutter Processing Time is long and, consequently, if many clutter maps are to be performed per second, it would be preferable to implement this function in high speed dedicated hardware. Second, the Bulk Filtering times are excessive for a single interpulse period. However, by executing the three stages (of three calculations) in parallel, 'the maximum single stage time of 1.4 ms is within the interfunction time calculated earlier of 1.6 ms. Lastly, all of the other processing times are consistent with realistic interfunction times.

Chapter 9

CONCLUSIONS

The proposed Post Processor Architecture is a programmable distributed processor system with a potentially high throughput capability. This architecture appears to be well matched to the Post Processing function since it exploits the long interfunction period, multiple transmission data base, and sequential nature of the Post Processing algorithms. The distributed nature of the system allows it to process large amounts of data.

The nature of the individual processors must be examined further. There are two general directions the design of the processors could follow. First, they could all be made identical. This would be inefficient since many processors would have excess capability. However, this approach would facilitate fabrication, would provide for redundancy, and would allow graceful degradation in the event of a processor failure. The second approach would tailor each processor to a particular task. This is efficient but the proliferation of designs would lead to complexity and rigidity.

The problems of bus and processor contention must be critically examined. In any distributed processing system, the problems encountered in distributing the data and allocating the processor resources are fundamental and must be addressed. Once protocols and procedures are developed, it would be useful to verify their effectiveness by simulation or prototype hardware development.

Another problem area is the partitioning of algorithms between processors. This requires a detailed knowledge of the algorithms and the processor architecture, and this implies that a certain amount of programming must be done in a lower level language.

Such issues as detection and fault finding hardware and software, and actual hardware implementation need to be fully addressed. Nonetheless, the distributed processor architecture described appears to have an organization and potential capability that is well matched to the Post Processing task as it is presently envisioned.

Chapter 10

ACKNOWLEDGEMENT

The authors are particularly indebted to Miss Susan J. Andrews and Mr. James E. Clark for their excellent and timely support during the preparation of this report.

Appendix A

GLOSSARY

ACCUM	Accumulator
ADSP	Advanced Digital Signal Processor
ARG	Argument
A/D	Analog to Digital
BC1	Binary Channel 1
BC2	Binary Channel 2
BC3	Binary Channel 3
BF	Bulk Filter
BMD	Ballistic Missile Defense
DC	Digital Convolver
DDB	Data Distribution Bus
DDCB	Data Distribution Control Bus
DEC	Decrement
AZ	Azimuth
EL	Elevation
DIV	Divide
DP	Detection Processor
DPS	Data Processing System
ECL	Emitter Coupled Logic
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
I/O	Input-Output
IB	Input Buffer
INC	Increment
K	times 1024 (i.e., 4K = 4096)
Km	Kilometer(s)
LL	Lincoln Laboratory
LOP	Lesser (Minor) Operation
MEM	Memory
MHz	Megahertz (10^6 cycles per second)
MOP	Major Operation
ms	Millisecond(s) ($1/10^3$ seconds)
MUL	Multiply
NOP	No operation
ns	Nanosecond(s) ($1/10^9$ seconds)
OB	Output Buffer
OBM	Output Buffer Memory
OP	Operand
PC	Program Counter
PP	Post Processor
PP1	Post Processor Port 1
PP2	Post Processor Port 2

PPS	Pulses Per Second
PROC	Processor
PROG	Program
QUO	Quotient
RDC	Radar Data Conditioner
REG(S)	Register(s)
REM	Remainder
RF	Radio Frequency
ROM	Read Only Memory
RV	Re-entry Vehicle
S	Search
STR	System Technology Radar
V	Verify
XFER	Transfer

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

19

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 18 ESD-TR-79-301	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* 6 A Distributed Processor Architecture for BMD Signal and Data Processing Applications.		5. TYPE OF REPORT & PERIOD COVERED 9 Technical Note
		6. PERFORMING ORG. REPORT NUMBER Technical Note 1979-84
7. AUTHOR(s) 10 Robert L. South and Robert J. Purdy		8. CONTRACT OR GRANT NUMBER(s) 15 F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element Nos. 63304A and 63308A
11. CONTROLLING OFFICE NAME AND ADDRESS Ballistic Missile Defense Program Office Department of the Army 5001 Eisenhower Avenue Alexandria, VA 22333		12. REPORT DATE 11 21 December 1979
		13. NUMBER OF PAGES 60
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB Bedford, MA 01731 12 59		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

14 TN-1979-84

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

distributed processing	Advanced Digital Signal Processor (ADSP)
multiprocessors	bulk filtering
Ballistic Missile Defense (BMD)	post processor

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The signal processor in large Ballistic Missile Defense (BMD) systems includes a subsystem, known as the Post Processor, which selectively reduces the received data to rates which are appropriate for the subsequent general purpose Data Processing System. This report presents an analysis of the processing requirements for the Post Processor and describes a multiprocessor, multibus architecture which appears well matched to the processing and throughput requirements.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

207650 [illegible]