(1)

DTIC FILE COPY

wsmrftr.doc

Report DAAD07-87-C-0102 FTR

HIGH SPEED CASCADABLE SIGNAL PROCESSING CIRCUITS

Dr. Michael Andrews
Octavio Morales
Bradley Hamilton
Space Tech Corporation
125 Crestridge Drive
Fort Collins, Colorado 80525

April 29, 1988

Final Technical Report
September 16, 1987 to March 15, 1988

Prepared for
U.S. Army White Sands Missile Range
Commanding Officer, STEWS-ID-T
White Sands Missile Range, New Mexico 88002

[illegible]

Octavio J. Morales
Assistant Principal Investigator

88

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS -------------
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Space Tech Corporation; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army White Sands Missile Range
6c. ADDRESS (City, State, and ZIP Code) 125 Crestridge Drive, Fort Collins, Colorado 80525	7b. ADDRESS (City, State, and ZIP Code) Commanding Officer, STEWS-ID-T, U.S.Army White Sands Missile Range, New Mexico 88002-5143
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAD07-87-C-0102
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
665502	1P65502M40		

11. TITLE (Include Security Classification)
High Speed Cascadable Signal Processing Circuits

12. PERSONAL AUTHOR(S)
Andrews, Dr. Michael; Morales, Octavio; Hamilton, Bradley

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical	FROM 16Sep87 TO 15Mar88	1988 April 29	84

16. SUPPLEMENTARY NOTATION The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Cascadable-Processors, Vector-Processors, Real-Time Processors, DSP, High-Speed ALU'S, Two-Dimensional Algorithms, Parallelism, Microprogrammable.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
Signal-processing algorithms are multiply/accumulate intensive. An Expandable Vector Accelerator (EVA) architecture is presented which incorporates novel computing methods using advanced IC devices. This architecture is capable of executing speed-intensive radar-signal processing and spread-spectrum processing tasks including FFT, LMS, LS, convolution, correlation, and spectrum analysis.

Preliminary board-level designs have been developed to integrate specialized high-speed, cascadable-signal-processing circuits to process a myriad of real-time algorithms. Typical applications for the EVA include range and range-rate data-processing, digital focusing, real-time Kalman filtering, real-time Target Motion Resolution (TMR) processing of MPS-36 and FPS-16 radars data, and processing image/pattern information for real-time, multi-munitions-scenes optical trackers .

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Foo Lam	22b. TELEPHONE (Include Area Code) (505)678-3010	22c. OFFICE SYMBOL STEWS-ID-T

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

HIGH SPEED CASCADABLE SIGNAL PROCESSING CIRCUITS

Space Tech Corporation
125 Crestridge Drive
Fort Collins, Colorado 80525

Project Summary

The original intent of this project was to identify fast and flexible arithmetic engines suitable for real-time radar, signal- and image-processing applications. To that extent, preliminary design specifications for an architecture known as the Expandable Vector Accelerator (EVA) have been developed. This is an architecture capable of executing numerous real-time algorithms via two separate but tightly coupled components, a high speed DSP machine and a general purpose data processor.

The Vector Processing Hardware (VPH) is a speed-optimized architecture capable of processing vectors of complex data. The architecture is arranged so that a high degree of parallelism and concurrency can be achieved. This feature, along with a duplication of on-chip resources at the board level, are the primary reasons for the high throughputs achievable with the VPH. The overall concept has been theoretically confirmed by employing the FFT algorithm as a test-bed. The results of the study revealed that a quad VSP-325 configuration can yield a 1024 complex FFT result in 460 us. This is a significant accomplishment for a board-level product.

The Cascadable Processing Hardware (CPH) is also speed optimized. However, the architecture is configured for those applications where the concern for high precision and wide dynamic-range is at a premium. By means of sophisticated control circuitry, the CPH can be reconfigured dynamically to expand or reduce the width of the data words, thus optimizing the architecture for larger or smaller wordlengths and increasing overall throughput.

An EVA-like architecture places a great deal of demands on the system interface configuration. Overall speed and versatility are paramount issues. Based on the findings presented herein, the VME system configuration appears to satisfy all of the demands required by the high-performance, cascadable-processing engine.

Aside from the FFT study mentioned above, other theoretical-analyses for mapping algorithms onto the EVA architecture were performed. A partitioning scheme for the Kalman filter routine was also presented. For realistic state-vector dimensions, the CPH and the VPH can execute the Kalman filter algorithm in real-time by using a scheme that allows some values to be constant over several samples. Several techniques for edge-detection and enhancement, clutter removal, and target tracking have been studied as well. This study was made to determine optimal strategies for removing clutter from missile data (in real-time) so that tracking can be enhanced. The basic principle of our approach is based on the commonly known fact that if background can be edge-detected, then a cut-and-fill operation can eliminate this "noise." The best edge-detection algorithms are thus important so that digital focusing can be more readily accomplished.

RESTRICTIVE MARKINGS DECLARATION

Certain research findings contained within this technical report were developed through independent research at Space Tech Corporation or were acquired under non-disclosure from third parties. As such, the restrictive markings provision of contract DAAD07-87-C-0102 pertaining to Government Purpose License Rights, as defined in the DOD FAR Supplement Section 27.471 (May 1987), apply to the following:

Page 21.	Section 5.1.1 including Figure 2
Page 30.	Figure 11
Page 32.	Figures 12a and 12b
Page 42.	Figure 21
Page 45.	Figure 22
Page 50.	Figure 23

These sections and figures are identified within the report with the marking, "Company Proprietary Information".

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 Introduction

The overall objective of contract DAAD07-87-C-0102 was to determine fast yet simple programmable signal-processors for cascadable implementation. A major focus of concentration was to develop algorithm specifications for a dedicated multi-processor architecture for data- and signal-processing applications. The result of these efforts is the hardware, firmware, and interface specifications for an architecture known as the Expandable Vector Accelerator (EVA). This arithmetic engine is ideally suited for applications such as range instrumentation, radar signal processing, image processing, Kalman and digital filtering, and real-time target-motion resolution. To develop these specifications, Space Tech Corporation divided the overall project scope into five technical objectives. They were:

1. Study and organize the EVA architecture into efficiently coupled modules for radar and signal processing. In this step, data transfer techniques were investigated to increase I/O transfers at the chip and board levels. Optimal trade-offs were determined among engineering parameters of power, board size, and speed of operation so as to render EVA machinery fast and efficient for laboratory and range instrumentation applications.

2. Determine the optimal trade-offs between fixed- and floating-point number systems. Also, analyze the rounding/truncation issues and/or the overflow/underflow issues with respect to fixed-point and floating-point operations in the EVA. The objective was to identify efficient word lengths for signal processors in EVA-like architectures.

3. Study optimal ALU configurations that speed up signal processing in EVA architectures. The objective here was to determine the ideal configuration (16x16, 32x32, or larger multipliers) which supports the processing bandwidths required.

4. Research the usage of fast controller circuits that may utilize centralized or distributed PLAs. The objective of this step was to improve arithmetic processing speeds while reducing or at least maintaining low control wire count from the control unit to the control points in the architecture.

5. Research microprograms for fixed- and floating-point signal processing algorithms executable on EVA architectures. The objective was to developed sets of signal processing micro-routines that could be ported across architectural changes.

The following sections describe the efforts undertaken at Space Tech Corporation to satisfy the objectives set forth above. In addition, justification for the options selected are presented.

1.1 Introduction to the EVA Architecture

The EVA is an architecture concept whereby high-speed yet versatile and efficient computations are a must. In order to reach an acceptable compromise between these conflicting needs, the process of selecting the building blocks for each component of the EVA architecture considered several issues. Minimum/maximum cascadable increments (8, 16, or 32 bits CPH only), execution speed, versatility, availability, amount of "glue logic" needed, overall chip count, and maximum utilization of available resources are just a representative sample of the issues considered.

Figure 1 shows the block diagram of the 32-bit EVA architecture containing the **Vector Processing Hardware** (VPH) and the **Cascadable Processing Hardware** (CPH) modules. It has been determined that all of the modules will connect to the VMEbus. The VMEbus data transfers between modules can handle up to 32 bits in one transfer, however the CPH allows up to 64-bit on-board data manipulations when two CPH modules are incorporated. One CPH module will support up to 32-bit wordlengths. This cascadability allows users to maximize the use of available resources.

The VPH is ideally suited for high-speed signal-processing applications where efficient, complex-data number-crunching is of the utmost importance. The heart of the VPH (the ZR34325 also referred to as the VSP-325) is capable of executing high-level, vector oriented instructions which embed the DSP algorithms directly into the device, allowing efficient algorithm execution. Moreover, a VSP-325 based architecture facilitates algorithm partitioning in the sense that multiple VSP-325s can be paralleled in order to share in the data processing requirements. Hence, while the VSP-325s perform parallel processing with interleaved I/O on the data from one RAM section, the host or the CPH can be up-loading or down-loading data into the other memory bank of the VPH. Once the current activities are completed, the roles of the VPH memory banks are reversed. This function-swapping is the primary reason for the efficiency and high throughputs attainable with the VPH.

2.0 EVA System Bus Configuration

In order to fully capitalize on the processing power of an EVA architecture, the system bus configuration must be equally capable of interfacing with the host, and within modules of the architecture. A study was made to identify the most optimal bus arrangement which allows maximum exploitation of the capabilities of the EVA architecture. The study did not consider 16-bit bus configurations such as the STD bus, MULTIBUS I, UNIBUS, and Qbus. The reason is that these systems do not satisfy current DSP and/or military real time demands, nor are they capable of supporting the dynamic range required in such applications.

Of the bus configurations considered, the VME and the MULTIBUS II systems are the most talked about and, by far, the most profitable in the market. Other relatively new, but fast rising entrants into the market include the NuBus, Fastbus, and Futurebus. The characteristics of these buses are highlighted in subsequent sections.

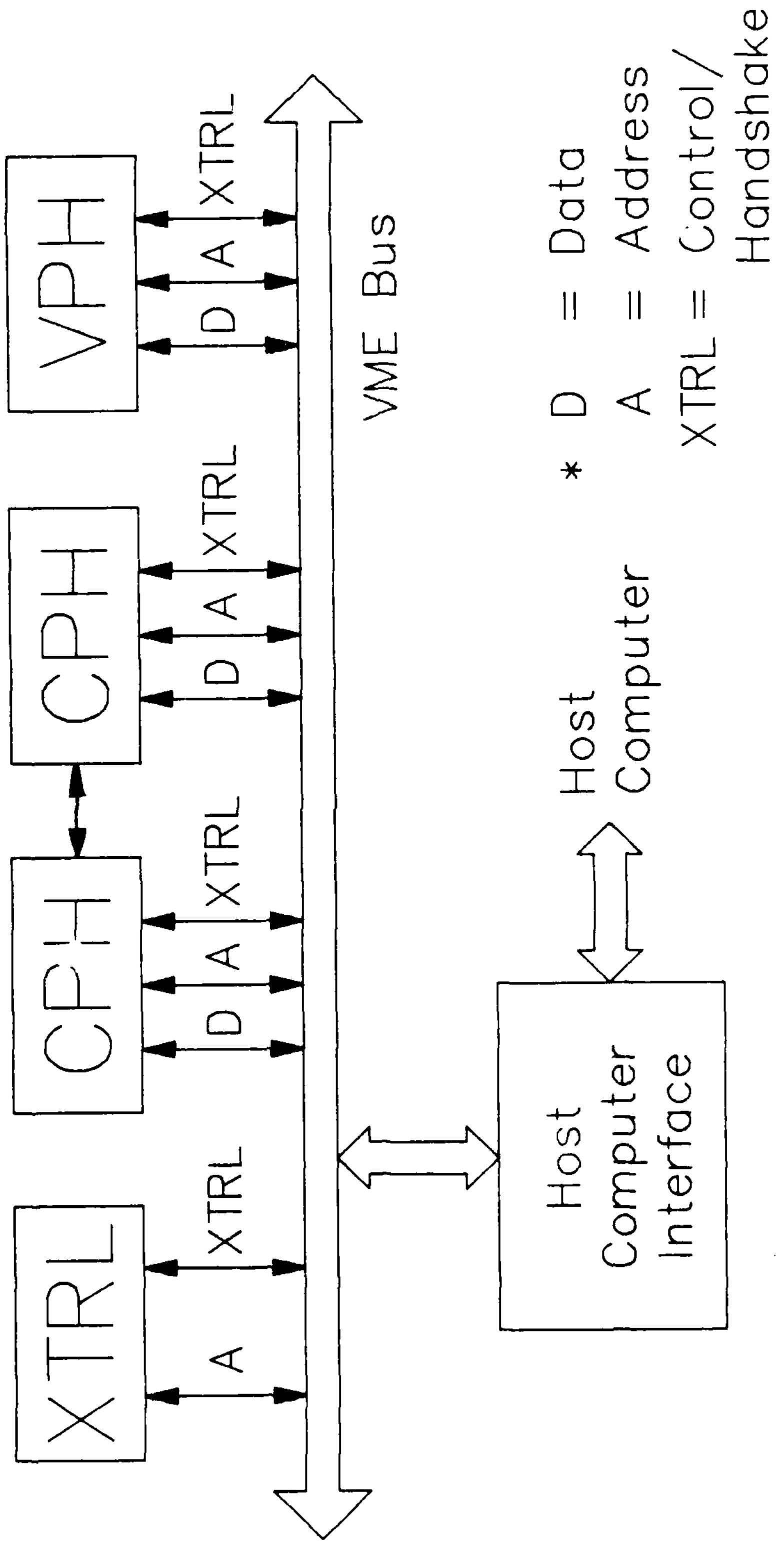


Figure 1. 32-Bit EVA Architecture.

2.1 Bus Comparisons

The second generation buses emerging nowadays are considered more of an architecture than simple buses. Unlike their predecessors, these systems provide more than the ordinary DMA transfers. They are capable of transmitting variable length data (8, 16, and 32-bits). Their addressing range is also expandable up to 32-bits. Most importantly, these systems are better able to handle multi-processing tasks by providing sophisticated bus arbitration techniques. Given the wide spectrum of enhancements across the board, the criteria used to categorize and compare the different systems is based on the following:

- transfer protocol (synchronous vs. asynchronous)
- signal lines (multiplexed vs. non-multiplexed)
- interrupt handling (dedicated vs. virtual)
- arbitration (centralized vs. distributed)

2.1.1 Transfer Protocol

Even though advantages/disadvantages exist for each type of transfer protocol, the synchronous vs. asynchronous issue must be examined in detail to determine the best solution for the EVA architecture. A synchronous interface offers a great deal of design simplifications over its asynchronous counterpart. Synchronized clocking allows simpler interface designs by referring every single event with respect to a master system clock. Another advantage of synchronous buses is a tighter control of the bus interface by fixing the maximum clock frequency. By the same token, however, the synchronous design specifies the maximum transfer rate of the system. This does not necessarily mean that the asynchronous design is faster. The point is that the asynchronous bus transfer can adjust to the transfer rate of the device accessing the bus, thus making it much more versatile. Furthermore, another big advantage of the asynchronous interface is that it provides the ability to mix various speed devices within the system, without restraining individual throughputs. Of course, a drawback is that since the asynchronous design is event driven, more knowledge, thought and experience must be placed in designing the "cause-effect" type logic required.

The EVA architecture will require a great deal of flexibility in its interface. Moreover, the speed of the devices to be employed must be complemented by the speed of the interface logic. Otherwise, the interface bottleneck will greatly degrade system performance, thus reducing overall bandwidth. In view of the requirements for high overall throughput, it seems logical to recommend an asynchronous bus architecture for use in conjunction with the EVA.

2.1.2 Signal Lines

Conceptually, the multiplexed vs. non-multiplexed issue is easy to understand. The key concept is efficiency. In terms of trade-offs, the efficiency concept can be divided into area vs. speed. Multiplexing allows an entire bus to fit in a single connector, and reduces power requirements by cutting address and data lines in half. The obvious drawback is that multiplexing typically requires twice the execution time for the same length transfer.

Keeping in harmony with the rationale employed in selecting an asynchronous protocol, the speed up provided by a non-multiplexed environment is equally desirable. The objective is to minimize the interface bottlenecks, and thus improve overall throughput.

2.1.3 Interrupt Handling

Task scheduling, a basic requirement in real-time multi-processor systems, is effectively handled through interrupt requests/grants. Of paramount importance is the time required to arbitrate and service the interrupt. The two approaches listed below (directed vs. virtual) differ in the time and the hardware required to service an interrupt request.

In a directed interrupt system, dedicated modules are employed for interrupt generation and handling. This set up relieves other masters in the system for higher priority tasks. Typically, three modules are employed. They are generally called the interrupter module, the interrupt handler module, and the arbiter. The name associated with each module is indicative of the function performed.

A virtual interrupt system, on the other hand, does not require the hardware complexity exhibited in the dedicated interrupt structure. Interrupt generation and handling is performed under the control of each individual master in the system. A processor enables an interrupt by passing a message to a memory-mapped location of another processor. This involves arbitrating the bus, gaining access to it, and finally passing the necessary information.

Clearly, a processor will not achieve maximum throughput if it must participate in the process of monitoring and responding to interrupt signals. Even though the virtual interrupt system requires less hardware than its counterpart, the time involved in assigning a task to another processor may not be feasible for real-time applications. As a result, it is concluded that an interface system with a dedicated interrupt structure will best complement the performance demands of an EVA-like architecture.

2.1.4 Bus Arbitration

Bus arbitration determines which one of all interrupting processors will gain access to the bus. Two general methods are considered: centralized and distributed.

In the centralized bus arbitration technique, a system arbiter module is responsible for assigning ownership to the bus. Any master wishing to gain control of the bus has its own requester module responsible for making the proper bus request to the system arbiter, through its own bus request line.

Bus ownership is then granted via priority, round robin, or other arbitration schemes.

In a distributed arbitration system only one bus request line exists. Each processor wishing to become bus master attempts to drive its own arbitration ID number on the bus. If the number on the bus matches the processor's arbitration ID number, that processor becomes the present bus owner.

The issue of centralized vs. distributed arbitration constitutes the closest debate of all items considered. A drawback of the centralized arbitration technique is that it is limited to as many arbitration levels as there are bus request lines in the system arbiter. Distributed arbitration, on the other hand, allows as many discrete levels as there are potential bus masters in the system. The question of exactly how many arbitration levels will be needed in the EVA is not known at this time. Nonetheless, it is highly doubtful that this number will exceed the 20 masters serviced by the VME.

2.2 The Bus Architectures

A brief description of each bus considered is presented in this section. Of the buses studied (VME, MULTIBUS II, Nubus, Futurebus, and Fastbus), the Fastbus holds the distinction of having the least amount of technical literature available. The Nubus is not far behind, even though a copy of the bus specifications have been requested several times.

2.2.1 The VME Architecture

The VME is a versatile bus architecture which provides asynchronous, non-multiplexed transfers between master and slave devices. It allows mixtures of 8, 16, or 32 bits of data as well as 16, 24, or 32 bits for addresses. The size of data and address paths are determined on a cycle by cycle basis. Due to its asynchronous transfer protocol, the VMEbus is very timing fault-tolerant. In addition, various processors and peripherals can operate at various speeds without having to wait for proper timing to get on/off the bus.

In terms of raw bandwidth, the VMEbus is capable of transferring 32-bit data at up to 40 Mbytes/sec. It utilizes dedicated interrupt lines, as well as centralized arbitration for up to 20 masters in the system. The standard board size for this bus architecture is (160 x 234 mm), even though a variety of non-standard board sizes are also supported. Table 1 summarizes the VME characteristics, and provides the means for easy comparison with the other buses considered.

2.2.2 The MULTIBUS II

The MULTIBUS II architecture is an enhanced version of Intel's MULTIBUS I. Like the VME system, the MULTIBUS II is specifically designed for high performance systems requiring 32-bit data transfer rates of up to 40 Mbytes/sec. It is also capable of transferring 8, 16, or 32 bits of data through an address space 16, 24, or 32 bits wide. Unlike the VME system, however, addressing on the MULTIBUS II is multiplexed and its transfer protocol is synchronized. Other differences include a virtual interrupt structure, distributed arbitration, and board sizes. The standard board size for the MULTIBUS II is (220 x 234 mm). It is not known if non-standard boards exist.

A very attractive feature of the MULTIBUS II is its built-in error detection mechanism. Subsystem buses within the MULTIBUS II utilize parity to detect errors in data transmission. Cyclic redundancy checks are also employed to detect errors in serial communication links. The MULTIBUS II is also summarized in Table 1.

2.2.3 The Nubus

The Nubus architecture, from Texas Instruments, is very similar to that of the MULTIBUS II. That is, it is a synchronous, multiplexed bus with a virtual interrupt structure and distributed arbitration. It is optimized for data transfers over a 32-bit address space. The data bus, however, is configurable as an 8, 16, or 32-bit bus.

The bandwidth of the Nubus is also comparable to that of the MULTIBUS II (37.5 Mbytes/sec for 32 bit transfers). However, there is a difference in standard board size, where the Nubus supports a standard size of (101.6x 327.02 mm). A larger board, with dimensions of (279.4 x 355.6 mm), is also available.

2.2.4 The Futurebus

IEEE's Futurebus sports the second fastest bandwidth of all buses considered (120 Mbytes/sec). It provides asynchronous data transfers over multiplexed signal lines. Its interrupt protocol and arbitration procedure are similar to those of the MULTIBUS II and Nubus (virtual and distributed, respectively). It is capable of transferring 8, 16, or 32 bits of data over a 32-bit address bus. A board size standard of (160 x 220 mm) exists, although boards of (160 x 280 mm) and (160 x 400 mm) are also acceptable.

2.2.5 The Fastbus

Very little information is available on this bus architecture, other than the fact than it possesses the highest bandwidth of the buses considered (160 Mbytes/sec) and that its transfer protocol is asynchronous. Its high bandwidth, however, makes it deserving of further investigation. This study shall be conducted during phase II of this project prior to the development of an EVA prototype.

2.3 EVA System Bus Recommendation

An EVA-like architecture places a great deal of demands on the system interface configuration. Overall speed and a high degree of versatility are paramount issues. Based on the findings presented herein, the VME system configuration appears to satisfy all of the demands placed by a high performance cascadable processing engine such as the EVA.

TABLE 1. B U S C O M P A R I S O N S

	VME	MULTIBUS II	NUBUS	FUTUREBUS	FASTBUS
TRANSFERS:					
Synchronous		X	X		
Asynchronous	X			X	X
SIGNALS:					N/A
Multiplexed		X	X	X	
Non-multiplexed	X				
INTERRUPTS:					N/A
Dedicated	X				
Virtual		X	X	X	
ARBITRATION:					N/A
Centralized	X				
Distributed		X	X	X	
ADDRESS BUS WIDTH	16, 24. or 32	16, 24, or 32	N/A	32	N/A
DATA BUS WIDTH	8, 16, or 32	8, 16, or 32	N/A	8, 16, 32	N/A
TRANSFER RATE	40 Mbytes/sec	40 Mbytes/sec	40 Mbytes/sec	120 Mbytes/sec	160 Mbytes/sec
ERROR CHECKING	No	Parity	N/A	N/A	N/A
STANDARD BOARD SIZE	(160x234 mm) (6.3x9.21 in)	(220x234 mm) (8.66x9.21 in)	(101.6x237.02 mm) (4x12.87 in)	(160x220 mm) (6.3x8.66 in)	N/A

N/A: Not Available

For instance, its asynchronous transfer protocol allows a system to achieve technology independence. That is, the bus configuration can adapt its transfer rate to that of the device accessing the bus, thereby eliminating transfer bottlenecks in a system composed of a mixture of slow and fast devices. Moreover, the asynchronous protocol insures longevity of the system. This is accomplished by providing the flexibility of incorporating faster devices into the system design, without having to redesign or upgrade the interface block. The latter, by the way, fits the philosophy of the EVA architecture. That is, to provide the ability to upgrade the system performance as superior technology is developed.

Other characteristics of the VME system, which further enhance system performance include non-multiplexed signal lines and a dedicated interrupt handling scheme. Both maximize overall throughput, at the expense of more hardware and, perhaps, higher power requirements.

An attractive feature of the MULTIBUS II is its capability to detect errors during data transmission. Another advantage over the VME system is the "unlimited" ability to support several masters in the system. For the EVA, however, the VME's support of up to 20 masters is sufficient.

From a marketing and support stand-point, the VME system is also the appropriate choice. The VME architecture is by far the most proliferated, supported, and economical bus interface system in the market. A multitude of "off-the-shelf" VME enhancement products are readily available. Moreover, the VME standard has gained wide acceptance at White Sands Missile Range and other military installations throughout the country. As noted in [1], the VMEbus has become the most popular bus architecture in both Europe and the U.S.A. Several major manufacturers, including Tektronix, AT&T, and IBM have endorsed the VME standard. The estimated current markets for VMEbus products is $65 million. That figure is expected to grow to approximately $750 million within four years [2]. Thus conformance with this pre-specified norm would most certainly guarantee performance, support, and compatibility across a wide range of applications. A summary of our study to incorporate the VME system within the EVA architecture is now discussed.

3.0 The VME System Interface

The VME system has become the most designed-in high performance bus in the market. The main reason for its proliferated use is that the VMEbus is an open architecture, unlike INTEL's MULTIBUS or DEC's BI bus. The VME system is a versatile architecture which provides asynchronous, non-multiplexed transfers between master and slave devices. It allows mixtures of 8, 16, or 32 bits of data, as well as 16, 24, or 32 bits for addresses. The size of data and address paths are determined on a cycle by cycle basis. By nature of its transfer protocol, the VMEbus is very timing fault-tolerant. Various processors and peripherals can operate at different speeds without having to wait for proper timing to get on/off the bus.

The VME system architecture is configured around three bus structures. They are the VMEbus, the VMXbus, and the VMSbus. The system interface configuration for the EVA is discussed in this section. VME interface protocols, addressing modes, transfer types, and bus arbitration schemes are presented [3].

The VMEbus provides the basic parallel data transfer medium between system components. In a single processor system, the VMEbus is the primary data transfer path. In this situation, little need exists for the VMXbus or the VMSbus. These secondary buses become highly beneficial in a multiple processor/controller environment. The VMEbus can be functionally divided into four sub-buses: data transfer, arbitration, interrupt, and utilities.

3.1.1 Data Transfer

A VMEbus can be configured with either a 16- or 32-bit data bus, allowing data transfer up to two or four bytes simultaneously. For a dual-byte transfer, the even byte resides on D8-D15, and the odd byte is placed on DO-D7. For a quad-byte transfer, byte 0 occupies D24-D31 and byte 3 occupies DO-D7.

The memory is arranged in groups of four-bytes. Address lines A02-A31 dictate which 4-byte group is being recognized, and the control lines DSO, DS1, AO1 and LWORD combine to allow access to any combination of these four bytes.

To enable any single byte of data, the control signals must fulfill the following conditions:

1) LWORD must be high if any single or double combination is desired.
2) For an even byte (byte 0,byte 2), DSO=1, DS1=0.
3) For an odd byte (byte 1,byte 3), DSO=0, DS1=1.
4) For the lower two bytes, AO1=0.
5) For the upper two bytes, AO1=1.

To access any two bytes:

1) Both data strobes are driven low.

2) The initial value of AO1 determines the starting double-byte block. This value is valid for only one cycle, then the two groups are alternatively selected.

To access the lower word, all four control lines must be driven low.

3.1.2 Addressing Modes

A Master broadcasts a 16-, 24-, or 32-bit address. The different address modes are labeled as short, standard or extended. When a Slave responds to an address, it must drive either DTACK or BERR low. At the end of the data transfer, the Slave must release control of these lines.

When a Master reads data from a Slave, the data must remain valid until the Master returns the first data-strobe to high. Once a master has driven its data-strobe(s) low, it must not drive them high until receiving a data-transfer acknowledge or a bus-error response.

3.1.2.1 Block Transfer

The VMEbus allows block transfers up to 256-bytes in order to simplify and to speed up address decoding. The Master facilitates block-transfers by keeping the address strobe low. It repeatedly drives the data strobe(s) low in response to data-transfer acknowledgments from the Slave, and transfers data to/from sequential memory locations. The Slave is responsible for generating the next address for each data-strobe transition.

3.1.2.2 Unaligned Transfer

A 32-bit Master can communicate with a 32-bit Slave in an unaligned, double- or triple-byte transfer if the Slave is configured correctly. A triple-byte, unaligned transfer can access bytes 0-2 or bytes 1-3, and a double-byte, unaligned transfer must access bytes 1-2 of the address specified by A1-A31.

3.1.2.3 Location Monitor

A Location Monitor monitors data-transfers over the DTB, detecting accesses to the locations it has been assigned to watch. When an access to one of these assigned locations occurs, the Location Monitor generates an on-board signal.

3.1.2.4 Address Only Capability

The address-only cycle does not allow data-transfer. During the execution of this cycle, it does not drive the address strobe low. This cycle can enhance board performance by broadcasting an address before the Slave, that will receive the address, has been determined. Thus, it allows the Slaves to decode the address concurrently with the CPU board.

3.1.3 Typical Data Transfer Cycles

Once exclusive control of the DTB is granted to a Master, a data transfer can be initiated. The Master first drives the address lines with the desired address and Address-Modifier (AM) code, then it provides the appropriate control signals to specify which bytes out of the 4-byte group are needed. The Master must wait for a specified set-up time, then it will drive the **AS** low which signals the Slave when the address is stable and valid.

Each Slave determines whether it should respond by examining the address lines, the address modifier lines, and the IACK line. If the address or AM lines do not correspond with a given Slave or if **IACK** is low, then that particular Slave will not respond to the Master's request. While this is happening, the Master drives **WRITE** to the appropriate level and verifies that **DTACK** and **BERR** are high, ensuring the Slave (from the previous cycle) is no longer driving the data bus. If this is the case, the Master then drives DS0 and DS1 to the correct levels.

The responding Slave determines which four-byte group and which bytes of that group are to be accessed from the address and control lines. It then starts the transfer. After it has retrieved the data from its own internal storage and placed it on the data bus lines, the Slave signals the Master by driving **DTACK** low until the Master relinquishes the low signal on either DSO or DS1.

When **DTACK** goes low at the Master, the Master captures the data, releases the address lines and drives either DSO or DS1 and AS high. The Slave responds by releasing DTACK, which is pulled high by the backplane terminators.

3.1.3.1 Address Pipelining

A Master can perform address pipelining by broadcasting an address for the next cycle while the data-transfer from the previous cycle is still in progress. However, a Master can change this address upon reception of the **DTACK** or **BERR** signal from the previous Slave. To prevent deciphering an incorrect address, the Slave should initiate data transfers to and from the bus on the falling edge of each data-strobe.

3.2 Data Transfer Bus Arbitration

The data-bus arbitration allows several Masters to share a common bus. The VMEbus arbitration subsystem prevents two Masters from using the bus simultaneously, at the same time it schedules requests from multiple Masters, optimizing bus use. For additional insurance against two Masters controlling the same bus, a Master must wait until it detects **AS** high before turning on its DTB drivers.

The VMEbus supports three arbitration schemes: prioritized, round-robin, and single level. Prioritized arbitration assigns the bus according to a fixed priority scheme where each of the four bus-request lines has a priority level assigned, from the highest (BR3) to the lowest (BRO). Round-robin arbitration assigns the bus using a rotating-priority basis. When the bus is granted to the requester, the highest priority for the next arbitration is assigned to the next lower bus request line. Single level arbitration only accepts requests on **BR3**, and relies on BR3's bus-grant daisy-chain to arbitrate the requests. Once the Arbiter grants the DTB to a requester and detects that it has driven **BBSY** low, it will then drive the bus grant line high.

The arbitration bus consists of six bused VMEbus lines and four daisy-chained lines. The signals entering the daisy-chain on each board are called Bus-Grant In lines (BGxIN), while the signals leaving each board are called Bus-Grant Out lines (BGxOUT). The lines which leave slot n as BGxOUT enter slot n+1 as BGxIN, thus the daisy-chain arbitration line is formed.

In the VMEbus arbitration system, a Requester module drives a bus-request line (BRO through BR3), a bus-grant out line (**BGOOUT** through BG3OUT) and the bus-busy line (**BBSY**). When a Requester drives a bus request line, the low signal level is daisy-chained through each board until it reaches the arbiter board in slot one. If a board does not exist in one of the slots, then the signal must be connected to the next board. When a board receives a signal it is propagated to the next board, unless the Requester on that board is also requesting the DTB on the same level. The Arbiter receives the bus request signals and decides which module will receive control of the DTB at the next available time.

3.2.1 Bus Busy Line

Once control of the DTB has been granted via the bus-grant daisy-chain, the Requester drives **BBSY** low. The Requester then has control of the DTB until it releases **BBSY**, allowing the Arbiter to grant DTB to some other Requester.

3.2.2 Bus Clear Line

If a priority arbitration scheme is used, the Arbiter drives BCLR low to inform the Master it is currently in control of the DTB. When a higher priority request is pending, the current Master does not have to relinquish the bus within any prescribed time limit. Therefore, it can continue transferring data until it reaches an appropriate stopping point. Its on-board Requester will then release BBSY and allow the next Master to take control of the bus.

3.2.3 Requester

Two types of requesters exist on the VME system, the Release-When-Done (RWD) requester and the Release-On-Request (ROR) requester. A RWD requester releases the DTB when the module is finished with the bus. A ROR requester gives up control of the **BBSY** line only when it has finished using the bus and when another module requests the DTB. Therefore, an ROR requester monitors all of the bus request lines which increases its hardware, but it also decreases the amount of traffic on the DTB.

3.3 Priority Interrupt Bus

The VMEbus includes a Priority Interrupt Bus which provides the signal lines needed to generate and service interrupts. Interrupters use the Priority Interrupt Bus to send interrupt requests to an Interrupt Handler (IH). The VMEbus supports interrupt subsystems that fall into one of the following two groups:

a) Single handler systems. These have only one IH that receives and services all bus interrupts. The service routine is executed by the supervisory processor.

b) Distributed systems. These have two or more IH's, with each handler servicing only a subset of the bus interrupts. This type of architecture is well suited for distributed-computing applications, where multiple, co-equal processors execute the application software.

3.3.1 Priority Interrupt Bus Lines

The Priority-Interrupt-Bus consists of seven interrupt-request signal lines, one interrupt-acknowledge line, and one interrupt-acknowledge daisy-chain.

3.3.2.1 Interrupt Request Lines

Seven interrupt request lines exist, including IRQ1-IRQ7. Interrupts are requested by driving an interrupt-request-line low. In a single IH system, these interrupt request lines are prioritized, with IRQ7 having the highest priority.

3.3.2.2 Interrupt Acknowledge Line

The Interrupt Acknowledge line (IACK) runs the full length of the backplane and is connected to the IACKIN pin of slot one. When IACKIN is driven low, the IACK daisy-chain driver, located in slot one, transmits a falling edge down the interrupt acknowledge daisy-chain. The Interrupt-acknowledge daisy-chain operates in the same manner as the daisy-chain of the arbitration bus. These connections prevent two or more interrupters from responding to an interrupt-acknowledge bus-cycle. If an interrupter receives a falling edge on its IACKIN line and it needs to respond to an IACK cycle, then it disrupts the chain and does not pass the falling edge to the next interrupter in the sequence.

3.3.3.1 Interrupt Handler

The IH uses the DTB to read a STATUS/ID from the Interrupter. In this respect, the IH acts like a Master and the Interrupter acts like a Slave. However, there are important differences, including:

a) The IH always drives IACK low, where a Master either drives it high or does not drive it at all.

b) A Master drives the AM lines, but an IH is not forced to drive these lines. In addition the IH only uses the lowest three address lines (A01-A03) to indicate which of the seven interrupt request lines are being acknowledged.

c) An IH never allows the data bus to communicate with the interrupter, as opposed to a Master which usually drives the data bus in a normal DTB cycle.

3.3.3.2 Interrupter

Once an Interrupter receives a falling edge on the interrupt-acknowledge daisy-chain input, it performs one of two tasks. If the acknowledging-level on the three address lines is equivalent to the interrupt request line it is using, the Interrupter supplies a STATUS/ID to the data bus. If the interrupt request level is different, the Interrupter will pass the IACK signal to the next Interrupter in line.

As with an Interrupt Handler and a Master, there are several differences between an Interrupter and a Slave:

a) An Interrupter responds only when IACKIN is low, where a Slave is not allowed to respond under this condition.

b) A Slave monitors the AM lines, but an Interrupter does not. Furthermore, an Interrupter monitors only the lower three address lines and does not monitor the **WRITE** line.

c) A Slave must match precisely its data-length with that of the Master, whereas an Interrupter is permitted to respond with data of a different size than that requested. Under this circumstance, all undriven data lines will be nominally high because of the terminators on the backplane.

The two methods of release used by an Interrupter are Release-On-Register-Access (RORA) and Release-On-AcKnowledge (ROAK). RORA Interrupters release their interrupt-request upon access to a control- or status-register, while ROAK Interrupters release their interrupt-request-line when the IH reads its STATUS/ID.

3.3.4 Typical Interrupt Operation

A typical interrupt sequence can be divided into three phases:

Phase 1: The interrupt request phase.
Phase 2: The interrupt acknowledge phase.
Phase 3: The interrupt servicing phase.

Phase 1 starts when an Interrupter drives an interrupt request line low and ends when it gains control of the DTB. During phase 2 the Interrupt Handler uses the DTB to read the Interrupter's STATUS/ID. During phase 3 an interrupt-servicing routine is executed.

3.4 Utility Bus

The utility bus is composed of all the signals needed for timing, initialization and diagnostic capability for the VMEbus. This bus includes signals that provide periodic timing and coordinate the power-up and power-down sequences for the VME system. It provides a 16 MHz clock source, a system reset line, an AC fail line, a system fail line, a 5 volts source, a plus and minus 12 volts source, and an optional 5 volts standby source.

3.4.1 The System Clock Driver

The system clock is an independent, non-gated, fixed frequency, 16 MHz, 50-percent-duty-cycle signal. The SYSCLK driver is located on the system controller board which is located in slot one. It provides a known time base that is useful for counting off time delays.

3.4.2 The Serial Clock Driver

THe Serial Clock driver provides a fixed frequency, special waveform signal used by VMSbus modules that reside on VMEbus boards. Its waveform is specified in the VMSbus specifications.

3.4.3 The Power Monitor

This module detects power failures and signals the VMEbus system in time to effect orderly shutdown. When power is then reapplied to the system the Power Monitor ensures that all other VMEbus modules are initialized.

3.4.4 System Reset

The System Reset signal (SYSRESET) may be driven low by any VMEbus board to initialize the system. The only stringent requirement is the line must be held low for a minimum period of 200 ms.

3.4.5 AC Fail Signal

The **ACFAIL** signal is an open-collector driven signal which indicates that the AC input to the power supply is no longer available or that the required AC input voltage levels are not being met.

3.4.6 System Fail Signal

The **SYSFAIL** signal is also an open-collector driven signal that indicates a failure has occurred in the system. This signal may be generated by any board on the VMEbus.

3.5 The VMXbus

The high speed VMXbus provides a secondary private access sub-system to the VMEbus. Since it is a private bus, there is less arbitration delay, allowing a processor to access VMEbus memory with few, if any wait states [4]. The VMXbus protocol is designed to work very efficiently to access program memory. Like the VMEbus, it is also asynchronous, therefore it will not become obsolete as processor clock speeds increase.

3.6 The VMSbus

As multiple processors are added to a system, an important need develops to quickly communicate events between processors. The VMSbus provides a communications medium external and concurrent to all system buses [5]. This serial bus is useful for coordination of processor activity, resource and system management, diagnostics, and other real time communication.

4.0 Number Format Comparison

Several number formats are amenable for integration into the EVA architecture. Possibilities include fixed-point, block-floating-point, floating-point, and cascadable-floating-point. Each format offers several advantages and disadvantages. The following sections summarize the format parameters by discussing speed, precision, dynamic range and overhead characteristics of each format. For consistency, the same wordlength is assumed for all comparisons.

4.1 Fixed-Point Format

For fixed-point numbers, the format can be signed or unsigned. In either case, fixed-point processors perform arithmetic operations faster than the other types [6]. Since the binary point is in a fixed position, the precision of the number may not be maximized. Leading zeros will reduce the quantity of significant digits, since the bits will show a position rather than an actual value. The precision is therefore dependent on the magnitude of data being manipulated.

Dynamic range is the biggest drawback of the fixed-point format. Table 2 compares the dynamic range for different word sizes. Note that for a signed integer, 256 bits are required to equal the available dynamic range of a 32-bit floating-point number.

Table 2. Dynamic Range Comparison.

Word Length	Dynamic Range $[20\log_{10}(2^{(\#bits)})]$
16 Bits	96 dB
32 Bits	193 dB
64 Bits	385 dB
128 Bits	771 dB
256 Bits	1541 dB

In terms of software overhead, fixed-point algorithms require extra support for overflow/underflow determination, and for other shortcomings such as limited dynamic range and variable precision. In terms of hardware overhead, they require the simplest "glue-logic" and processors of any format considered.

4.2 Block-Floating-Point Format

The block-floating-point format is most effective when all the numbers are of similar magnitude. As numbers grow (or decrease) during the execution of algorithms such as the FFT, the binary point can be shifted (actually physically shift the bits) one position for each FFT column to keep overflows from occurring. An "exponent" register is then used to keep track of how many and in which direction shifts occur, or in other words, where the binary point is.

Block-floating-point arithmetic is slower than straight fixed-point, but it is still faster than floating-point since fixed-point processors are used. The extra time is spent updating the "exponent" register and shifting all data values when an update occurs.

The precision for block-floating-point is the same as for fixed-point, but it is less dependent on magnitude. The number of significant digits is equal to or greater than that in fixed-point arithmetic, since the data is allowed to shift up or down when a predetermined number of leading/trailing zeros are encountered.

With the extra register to keep track of the binary point position, the dynamic range can be made much larger than in fixed-point (depending on the register size). The dynamic range of each block-floating-point number is lower than in floating-point because all data have the same exponent.

Additional hardware is required to monitor number growth, to update the "exponent" register, and to normalize the result. Some of this overhead can be transferred to software, but it will be assumed for now that hardware will perform these operations. This is a reasonable assumption given the processing power available in the EVA.

4.3 Floating-Point Format

Fixed-point dynamic range and precision limitations cause errors in DSP algorithms much like noise sources in analog systems [7]. As discussed in previous sections, the microprogrammer must keep track of scaling and overflow. In floating-point, concern over these issues is left up to the processor [8]. Speed is lower than in the fixed-point format, but fewer programming concerns and less hardware overhead help make up for the small speed loss.

Floating-point has less available precision than fixed-point and block-floating-point. However, precision is not dependent on the data's magnitude; instead, maximum precision is always available because the mantissa is normalized. Moreover, floating-point offers much better dynamic range than fixed-point. Depending on the size of the "exponent" register in block-floating-point, floating-point could offer less, the same or more dynamic range. Overflow and underflow can still occur in floating-point, but a 32-bit format has a dynamic range of over 1500 dB.

4.4 Cascadable-Floating-Point Format

The cascadable-floating-point format proposed in this section uses discrete fixed-point processors to implement floating-point arithmetic operations. The format is expandable, allowing additional bits to be added to the mantissa and/or the exponent the same way it is done with a fixed-point, bit-slice architecture. The speed is slower than all the formats considered, but the operations can be pipelined to boost throughput.

Both precision and dynamic range can be the same as in floating-point. However, precision and dynamic range are easily increased beyond that of floating-point by increasing the mantissa and the exponent size, respectively.

Cascadable-floating-point requires the largest hardware and software overhead of all formats. Fixed point processors must perform floating-point operations. Hence, if the architecture is pipelined to increase throughput, programs become more complex.

A precise engineering solution to the number format dilemma can be only obtained through analysis of the kernel processors considered, and mapping of the algorithms onto the processors. As a result, the architectural study described in the following section leads to a number format recommendation.

5.0 EVA Architectural Considerations and Components

Through extensive comparative analyses and continuous interfacing with WSMR technical contacts, the major I.C. building blocks for the EVA architecture have been selected. It has been concluded that the B2110/B2120 chip set from Bipolar Integrated Technologies and the ZR34325 (also referred to as the VSP-325) from ZORAN Corporation are ideally suited for the demanding number-crunching inherent to the EVA. Following sections present the findings leading to the above mentioned conclusions, and provide a set of benchmarks to illustrate the capabilities of the devices selected.

The EVA is an architectural concept whereby high speed yet versatile and efficient computation are a must. In order to reach an acceptable compromise between these conflicting needs, the process of selecting the building blocks for each component of the EVA has been carefully completed. For the cascadable modules (the CPH), for instance, several issues were considered. Minimum/maximum cascadable increments (8, 16, or 32 bits), execution speed, versatility, availability, amount of "glue logic" needed, overall chip count, and maximum utilization of available resources are just a representative sample of the issues considered.

The decision making process for selecting the VPH building blocks is less complex. Only two candidates are capable of satisfying the high throughputs demanded in an EVA-like architecture. Namely, AT&T's DSP32C and ZORAN Corporation's ZR34325 (also referred to as the VSP-325) provide 32-bit floating-point capabilities. Of the two, ZORAN's device is far superior. Both provide execution cycle times of 80 ns. However, the DSP32C is not flexible enough to efficiently execute complex multiplies. Consequently, the difference in algorithm execution times for certain DSP applications is noticeable, with the VSP325 holding the edge.

5.1 The Vector Processing Hardware (VPH)

The VPH is ideally suited for high speed signal processing applications. In this environment, efficient number-crunching of complex data is of paramount importance. To meet this requirement, the VSP325 is capable of executing high-level, vector oriented instructions which embed the DSP algorithms directly into the device. Moreover, the nature of the architecture and instruction set allow multiple VSP-325s to be paralleled on the same bus for improved performance. This device is detailed in the sections to follow.

5.1.1 The VSP-325 Vector Signal Processor

The VSP-325, shown in Figure 2 [9], is a speed-optimized processor ideally suited for real-time signal processing applications. Its architectural configuration is ideal for algorithms such as one- and two-dimensional FFTs, convolution, correlation, filtering (FIR and IIR), matrix operations, polynomial expansions, modulation/demodulation, minimum/maximum detection, thresholding and comparison, and other related functions. This device offers high throughputs due to the high-level, vector oriented nature of its instruction set. Each high level instruction can operate on a vector of up to 64 complex words. Each data word conforms to the IEEE 754-1985 32-bit floating-point format, although conversions to/from 2's complement are also supported.

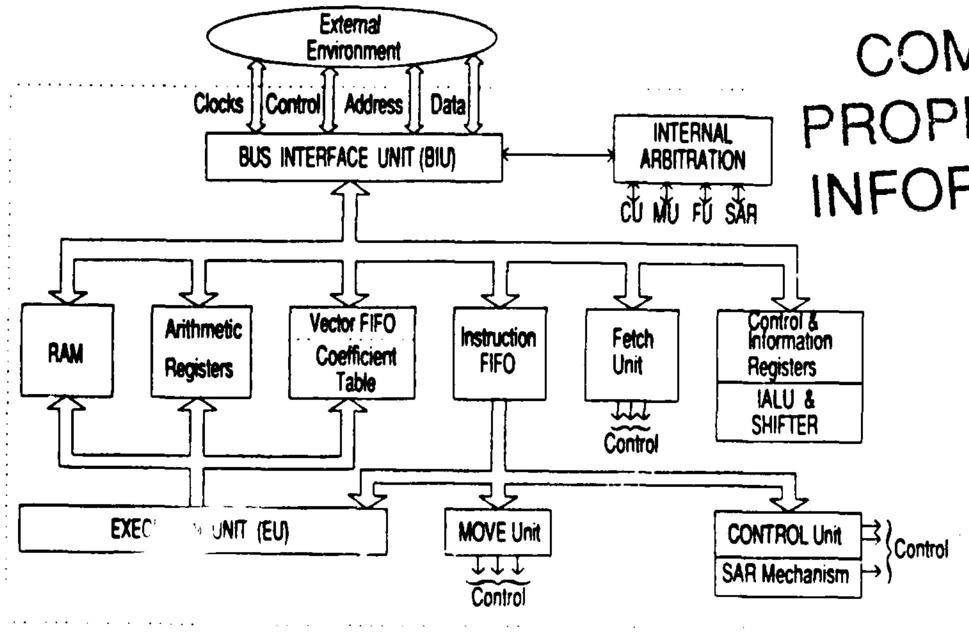


Figure 2. VSP-325 Architecture.

The VSP-325 is composed of seven essential blocks which contribute to its high performance capabilities. They are the Bus Interface Unit (BIU), the Execution Unit (EU), the Move Unit (MU), the Fetch Unit (FU), the Vector Unit (VU), the Control Unit (CU), and a block of memory and registers. These components are discussed below.

5.1.1.1 The Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) enables DMA transfers between the VSP-325 and the external environment. It provides non-multiplexed address (24 bits) and data (64 bits) paths for improved I/O interfacing. Its transfer mechanism resembles that of conventional microprocessors whereby Bus Request (BRQ) and

Bus Acknowledge (BACK) signals are exchanged between the devices wishing to access the bus.

5.1.1.2 The Execution Unit (EU)

The Execution Unit (EU) is responsible for all floating-point arithmetic operations within the VSP-325. The EU is optimized for complex arithmetic. It includes a multiplier, an adder and subtracter, a set of real and imaginary accumulators, and a set of registers. Data is loaded into the EU from two sources: internal RAM and the Vector Unit/coefficient table. External RAM data is buffered by the Vector Unit prior to entering the EU.

5.1.1.3 The Move Unit (MU)

The Move Unit (MU) is capable of address generation in direct, indirect, or indexed modes. This block of hardware is responsible for data transfers between the BIU and internal memory. It also has the capability to bit-reverse addresses "on-the-fly." All these features make the MU the most important component of the VSP-325, since it allows partitioning of the Vector Signal Processor into two independent I/O and processing blocks. The end result is a highly efficient DSP engine.

5.1.1.4 The Fetch Unit (FU)

The Fetch Unit (FU) provides the control required to fetch and execute VSP-325 instructions. Once instructions are fetched, they are stored in an instruction FIFO, out of which they are subsequently executed. Recall that the VSP-325 operates on a "high-level" set of instructions, thus the fetch mechanism is optimized to ensure minimum instruction fetching/decoding time.

5.1.1.5 The Vector Unit (VU)

As noted earlier, the Vector Unit (VU) works in conjunction with the MU to effectively transfer data from external memory to the EU. It operates as a four word FIFO to allow movement of external data independently of the EU. Hence, it allows the EU to concentrate on data processing with little concern about I/O.

5.1.1.6 The Control Unit (CU)

The Control Unit (CU) is responsible for coordinating operation among the VSP-325 internal registers. A total of 17 registers are available, three for arithmetic operations and 14 for information and control. Moreover, a coefficient look-up table is also provided to enhance algorithms such as FFTs, DFTs, modulation, and demodulation. The VSP-325 internal registers are listed in Table 3. Their names are indicative of the functions performed.

Table 3. VSP-325 Internal Registers.

Arithmetic Registers	Control and Information Registers
Real Accumulator	Program Counter
Imag. Accumulator	Stack Pointer
Min/Max Values	A-Base Register
	B-Base register
	SAR Counter
	Loop Counter
	Extra Register
	Min/Max Index Register
	Mask Chain
	Mode Register
	Status Register
	Interrupt Pointer
	Interrupt Mask
	Interrupt Status

5.1.2 VSP-325 Instructions

The instruction set of the VSP-325 can be divided into four functional categories: data movement, registers, arithmetic, and control. Overall, 52 instructions are provided some of which operate on data vectors and/or single constants. The high-level nature of the instruction set allows entire algorithms to be executed by invoking a single command. For instance, "FFT," "FIR," or "IIR" are functional instructions which execute the algorithms by the same name. A number of parameters are associated with each instruction. These parameters specify the number of data points, starting/ending addresses, and other algorithm-specific values.

5.1.3 Case Study - 1024-Point Complex FFT Execution

To illustrate the capabilities of the VSP-325, a 1024-point complex FFT is used as a test case. Recall that internal memory in the VSP-325 can be configured as a RAM section of 64 complex words, or as two independent sections, each 32 complex words deep. By so doing, data I/O and processing may be overlapped. This way, while one RAM section is transferring data into or out of the VSP-325, the other RAM section is used for data processing. Thus data I/O is "free." This highly "compute-bound" characteristic allows multiple VSP-325 to be combined on the same bus, thus increasing throughput beyond that of a single device.

Given the structure of internal memory, the kernel for large complex data FFTs (N > 32) is based on the single instruction FFT for 32 complex values. For instance, a 64-point complex FFT may be executed by separating the input data into even and odd components. The even values are loaded into the VSP-325 and the first pass of a 32-point FFT is executed. Meanwhile, the other RAM section receives the 32 odd values of the input data. Upon completion of the first pass of the FFT (for both even and odd values), the results are stored back in external memory in the same order in which they were loaded in. Next, a full 32-point complex FFT is executed on the top half of the intermediate results. The bottom half of the intermediate results are multiplied by the appropriate twiddle factors, followed by a full 32-point complex FFT. The last step involves the bit-reversal for the full 64 complex values.

A similar procedure is followed if the desired transform length is 1024 complex points. Here, the input data is originally divided into 32 blocks of 32 complex values each (every 32nd point taken). Next, the FFT sequence is repeated as outlined above. The point of the illustration, however, is to provide execution time estimates (for 1024-point complex FFTs) for architectures utilizing single and multiple VSP-325s. In each case considered below, the initialization step prior to each wave of FFT processing is ignored since the time overhead is negligible (on the order of 5 micro-seconds).

5.1.3.1 Single VSP-325 Configuration

The block diagram for the single VSP-325 configuration is shown in Figure 3. In this case, the first and second wave of FFT execution times are identical. Only the original loading and the final storing of 64 complex values into/out-of the VSP-325 count as I/O overhead; the reason is that the remaining of the data I/O is "hidden" during FFT execution times. This is illustrated in the timing analysis shown in Figure 4. From Figure 4, the FFT execution for a 1024-point complex signal requires 21064 clock cycles. At a rate of 80 ns per clock cycle, the FFT execution time is approximately 1.7 milli-seconds.

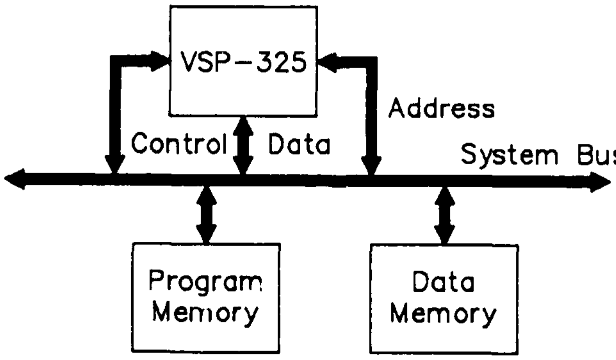
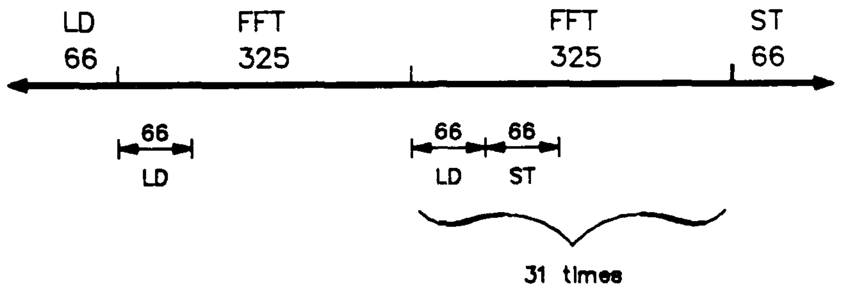


Figure 3. Single VSP Configuration.



Number of clock cycles per wave : $66 \times 2 + 325(32) = 10532$

x2 waves : $\underline{\times 2}$
21064

Execution time = 21064 cycles x (80ns/cycle) $\simeq$ $\underline{1.7\text{ms}}$

Figure 4. Timing Analysis - Single VSP-325 System.

5.1.3.2 Dual VSP-325 Configuration

In the dual VSP-325 configuration, two VSP-325s share a common address/data bus. By so doing, they both participate in the number-crunching process. The architectural block diagram is shown in Figure 5, while the timing analysis is illustrated in Figure 6. Notice that in this configuration, only one of the processors can utilize the bus at any given time. Hence, I/O from processor #2 begins upon completion of I/O from its counterpart. As shown in the timing analysis, each processor executes a load and a store within the FFT execution time frame. Both of these I/O operations require a total of 264 clock cycles, as compared to the 325 cycles required for the 32-point complex FFT. As a result, this system configuration still operates in a "compute-bound" fashion, whereby data I/O is "hidden" between FFT execution times.

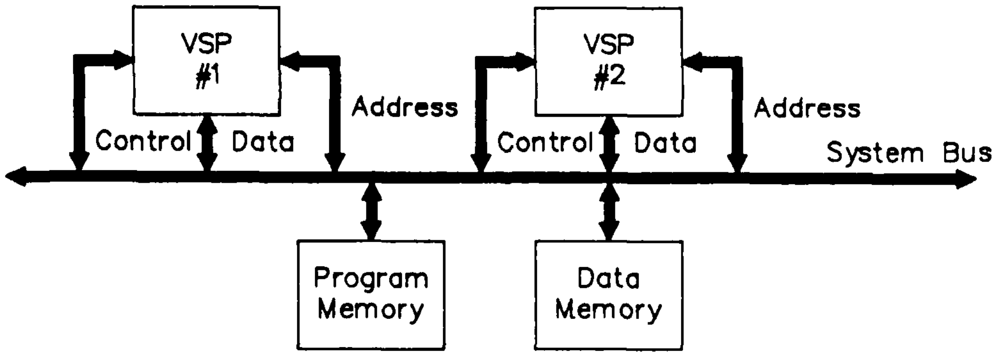
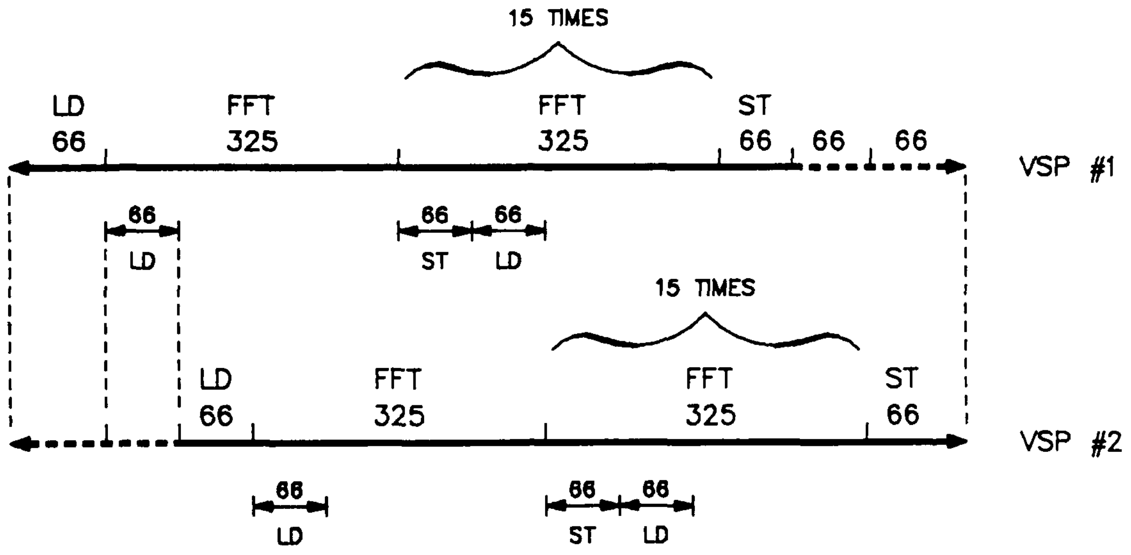


Figure 5. Dual VSP-325 Configuration.



$$\text{Number of clock cycles per wave} : 66\text{x}4 + 325\text{x}16 = 5464$$

$$\text{x2 waves} : \quad \frac{\text{x2}}{10928}$$

$$\text{Execution time} = 10928 \text{ cycles x } (80\text{ns/cycle}) \simeq \underline{880\text{us}}$$

Figure 6. Timing Analysis - Dual VSP-325 System.

Since both processors participate in the data processing phase, the FFT execution time is reduced by roughly a factor of two compared to the single VSP-325 configuration. The time required to execute each wave of the 1024-point complex FFT is reduced to 5464 clock cycles. The entire process, therefore, only requires 10928 cycles. At the 80 ns rate, **the 1024-point complex FFT is effectively executed in approximately** 880 **micro-seconds.**

5.1.3.3 **Quad VSP-325 Configuration**

In the Quad VSP-325 configuration, the four VSP-325s can be paralleled on the same bus, or on two different bus structures. The single bus configuration is shown in Figure 7. As the timing analysis reveals (Figure 8), the internal memory on all VSP-325s is configured as a single block of 64 complex words. The reason is that since four VSP-325s share the same system bus, the overlapped I/O and arithmetic capability of each processor can not be exploited. Each wave of FFT processing requires 4810 clock cycles, thus the 9620 cycles required by the 1024-point complex FFT are executed in **approximately** 775 **micro-seconds.** This is clearly not a very efficient alternative, since a two-fold increase in processing power only yielded a 13% speed-up (from 880 micro-seconds down to 775 micro-seconds). The main reason for the inefficiency is the emergence of bus contention problems during I/O transfers. This is evident from the timing analysis shown in Figure 8.

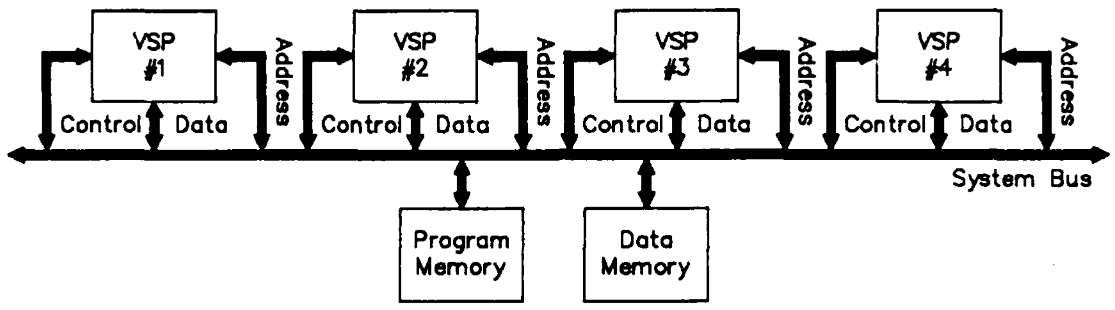
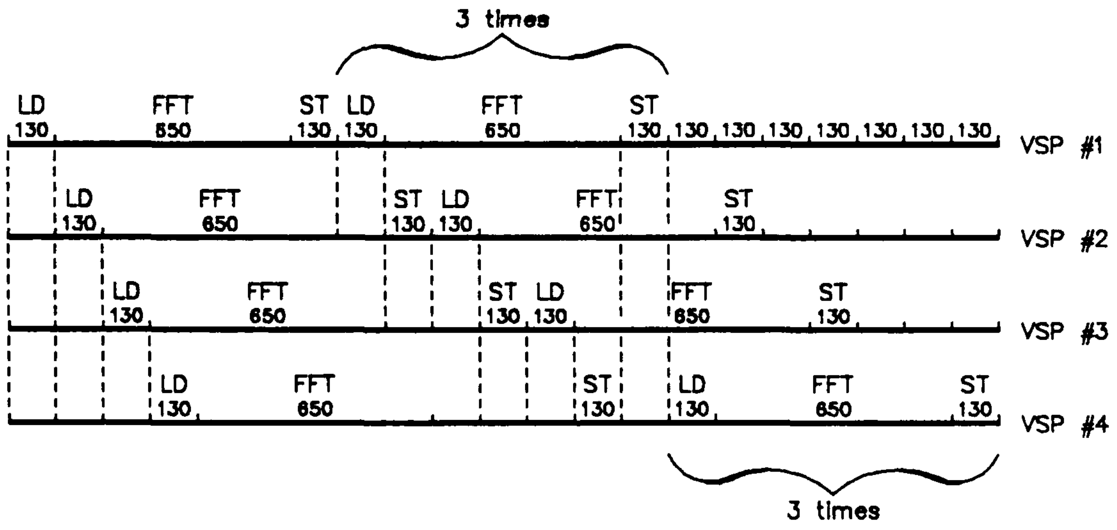


Figure 7. Quad VSP Configuration - Single Bus.



Number of clock cycles per wave : 130x2 + 650 + 130x13 + 650x3 = 4550

x 2 waves : x 2 = 9100

Execution time = 9100 cycles x (80ns/cycle) ≃ 730μs

Figure 8. Timing Analysis - Quad VSP Configuration (Single Bus).

With two parallel buses, the bus contention problems are eliminated. This set-up, shown in Figure 9, is similar to the dual VSP configuration in the sense that two VSP-325s are assigned per bus. As a result, the processors can exploit their overlapping I/O and processing capabilities. In this case, however, four processors equally share in the number-crunching process, thus enhancing overall throughput. As seen in the timing analysis (Figure 10), I/O is once again "free." The total number of clock cycles required to perform the 1024-point complex FFT is reduced to 5728 cycles. Thus the effective FFT

execution time is approximately 460 micro-seconds. This is certainly one of the fastest, if not the fastest, real-time floating-point processors available. The fact that the processor can fit on a standard VME module only magnifies the significance of the technological breakthrough.

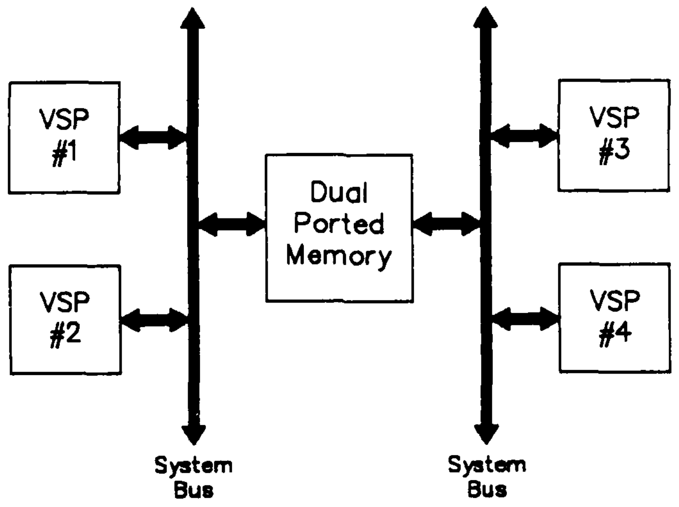
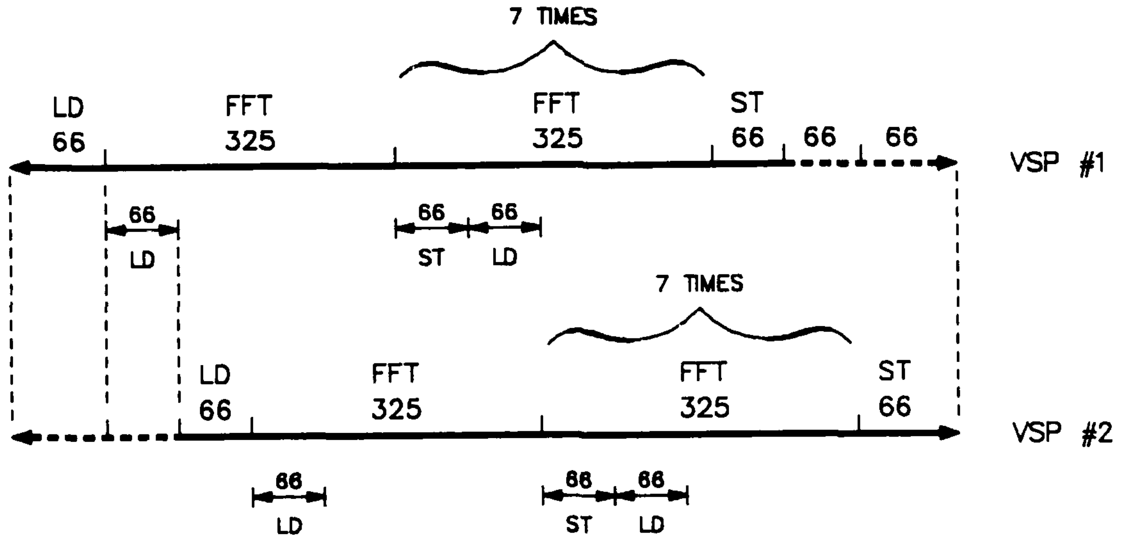


Figure 9. Quad VSP Configuration - Parallel Buses.



VSP #3 – Same as VSP #1

VSP #4 – Same as VSP #2

Number of clock cycles per wave : 66x4 + 325x8 = 2864

x2 waves : x2 = 5728

Execution time = 5728 cycles x (80ns/cycle) ≃ 460μs

Figure 10. Timing Analysis - Quad VSP Configuration (Two Buses).

5.1.4 Architectural Implementation - The VPH

The architecture of the VPH is internally optimized for matrix and signal processing tasks. This is primarily true due to the vector oriented nature of the processors employed. Recall that the VSP-325s are capable of processing blocks of up to 64 complex words with a single instruction. Moreover, a VSP-325 based architecture facilitates algorithm partitioning in the sense that multiple VSP-325s can be paralleled in order to share in the data processing requirements. This is facilitated by the fact that internal memory can be configured as two independent blocks of 32 complex words each. As a result, data I/O and processing can be interleaved by using alternate memory sections. This concept is extended outwards from the VSP-325s by duplicating the VPH on-board memory. Hence, while the VSP-325s perform parallel processing with interleaved I/O on the data from one RAM section, the host or the CPH can be up-loading or down-loading data into the other memory bank of the VPH. Once the current activities are completed, the roles of the VPH memory banks are reversed. This function-swapping is the primary reason for the efficiency and high throughputs attainable with the VPH.

A block diagram of the VPH is shown in Figure 11. The MC68010 microprocessor is used for data pre-processing and data-flow control to/from the host, the CPH, or the VSP-325s. In order for the MC68010 to communicate with the VSP-325s, it must first gain access to the appropriate system bus. This can be implemented by means of the bus-request/bus-acknowledge capabilities of the microprocessor. To communicate with the host or the CPH, the MC68010 must do so via the I/O latch and the control latch shown in Figure 11. The MC68010 is also used to monitor the status of all VSP-325s. Upon completion of a processing task by any/all of the VSP-325s, the MC68010 must redirect inputs/outputs to the appropriate memory segments. Likewise, the MC68010 must also pre-set the appropriate registers in the VSP-325s for further data processing.

All four VSP-325s are connected in parallel, with the exception of the chip-select, the bus-request, the bus-acknowledge, and the interrupt pins. The reason is that all four processors participate in the data processing operations. The bus-request and bus-acknowledge pins of each VSP-325 are connected to a Bus Arbitration Module (BAM). This bus arbiter can be implemented with PALs or with the Motorola MC68452. In this application, the BAM monitors the bus request line for each VSP-325, and if a request has been issued, the bus is granted to the appropriate processor by means of its bus-acknowledge line. Once the BAM has granted ownership of the bus to a particular VSP-325, it will prevent another VSP-325 from gaining access to it by simply not issuing a bus acknowledge. A priority arbitration scheme can also be implemented with the BAM in the event that VSP-325s issue simultaneous bus requests.

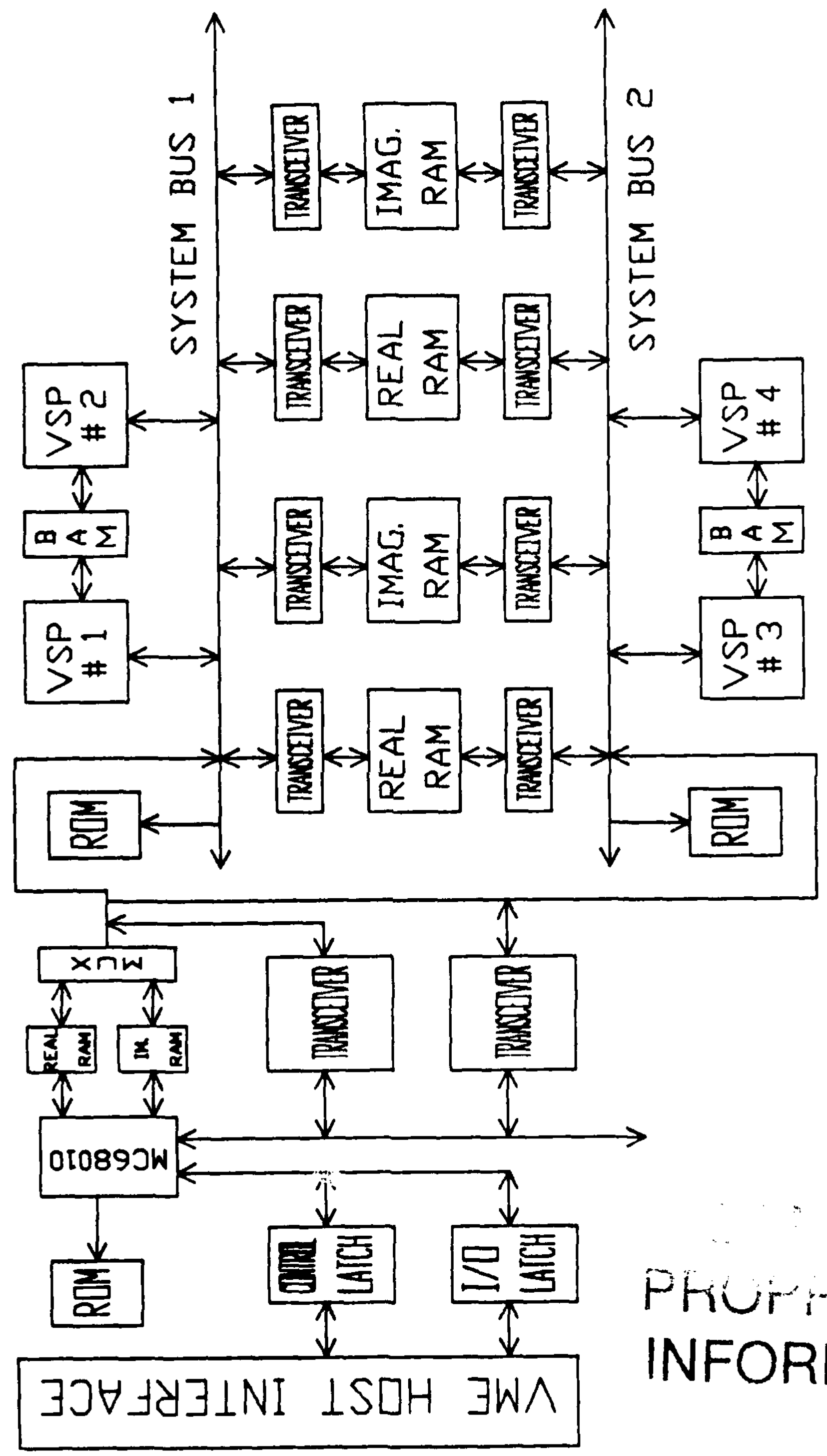


Figure 11. Vector Processing Hardware.

With this architectural layout, the VPH is capable of executing DSP algorithms at very high rates. A set of preliminary execution time estimates are given in Table 4.

Table 4. Preliminary Algorithm Execution Estimates.

Algorithm	Execution Time
1024 real element dot product	175.5 us
1024 point complex FFT	
- one VSP-325	1.7 ms
- two VSP-325s	880.0 us
- four VSP-325s (single bus)	730.0 us
- four VSP-325s (two parallel buses)	460.0 us
Two Dimensional Complex FFT (256x256)	137.8 ms
Two Dimensional 3x3 Convolution (per output)	1.1 us
Matrix Multiply (50x50)	11.0 ms
32-tap FIR filter (128 real points)	382.4 us

5.1.4.1 VPH Microcode Layout

In addition to the internal optimization of the VPH structure, the architecture is also optimized externally in the sense that a host, such as a general purpose computer or the CPH, may easily access any of the internal resources via a simple function request. Upon validation of the request, the VPH (by means of the MC68010) informs the host whether or not the function request is valid. If the VPH can service the request, it first sets up all of the internal resources needed, prior to informing the host that it is ready to receive data. On completion of the data transfer, the VPH can immediately process the data and may simultaneously handle another I/O request as long as at least one of the RAM sections is inactive. Once the data processing is finished, the VPH informs the host that it may retrieve the results.

To facilitate this external/internal optimization of the VPH architecture, two independent microprogram memories are required. One sector, for external control, contains the microinstructions for the MC68010 microprocessor. The other, for internal control, contains the microinstructions for the VSP-325s. As a result, the two processing sections in the VPH (MC68010 and VSP-325s) can operate independently and concurrently on data from either RAM section, or directly from the host. The microcode format for external control is given in Figure 12a, while that for internal control is provided in Figure 12b.

From Figure 12a, it is readily seen that up to 32 functions can be executed with each of the processing sections of the VPH. Either processing section is capable of operating on blocks of data up to 65536 words. Recall that the MC68010 is provided for data pre-processing, data post-processing, or both. It can also perform data processing and/or logic functions while the VSP-325s perform signal processing tasks. Up to now, the MC68010 functions have not been clearly defined since most of the processing and interface requirements are not known yet. Nevertheless, the majority of the signal processing tasks have been already identified and are listed in Table 5.

Table 5. VSP-325 Functions.

Complex FFT	Inverse Complex FFT
Real FFT	Inverse Real FFT
Two Dimensional FFT	Convolution
Correlation	Two Dimensional Convolution
Two Dimensional Correlation	FIR Filter
IIR Filter	Matrix Multiplication
Matrix Addition	Vector Multiplication
Vector Addition	Division
Reciprocal	Square Root
Arc Tangent	DCT
Logarithm	Min/Max Detection
Thresholding	Comparison
Modulation	Demodulation

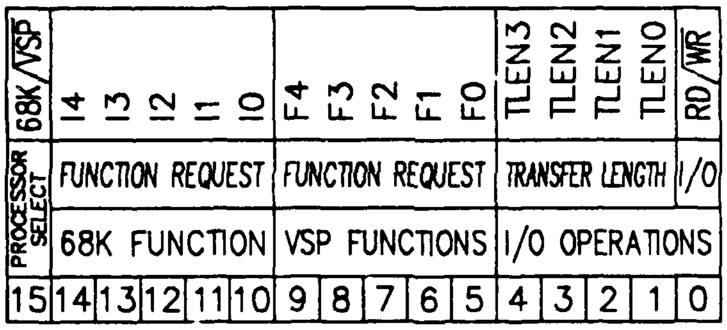


Figure 12a. External Control Microcode Format.

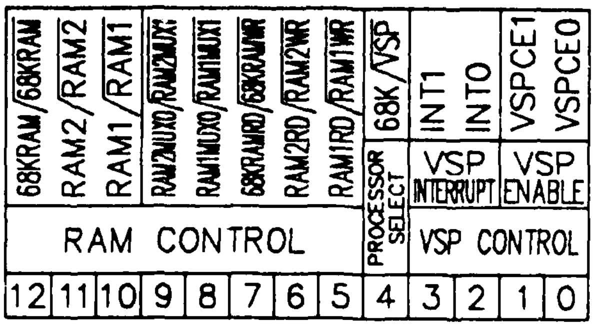


Figure 12b. Internal Control Microcode Format.

The microcode format for internal control is shown in Figure 12b. The layout shown in Figure 12b depicts the arrangement of the VPH internal resources in order to execute the function requested. For instance, bits zero through three determine which of the VSP-325s is enabled or interrupting at any given time. Bit four determines which of the processing sections of the VPH is enabled, while bits five through twelve control the individual RAM sections.

5.2 The Cascadable Processing Hardware (CPH)

The main thrust of development for the CPH of the EVA architecture is to organize it into efficiently coupled modules for signal and data processing algorithms. Each module should have the exact same architecture whether it is the least-significant or most-significant board. Several options have been identified and their characteristics studied. They include: a cascadable ALU with external multiply and divide support, a cascadable ALU with facilities on chip to implement multiply and divide, and a high-speed ALU-Multiply chip-set with multiply and divide functions available within the set.

Word length requirements include handling 8-, 16-, 32-, 48- and 64-bit fixed-point data. In addition, 32- and 64-bit floating-point operations must also be supported, whether by a separate floating-point module or by the same fixed-point module performing the floating-point operations. The hardware must also provide cascadability, where the number of modules required is determined by the word length.

The following sections discuss the various options for module size, board architecture, chip set selection and floating-point module alternatives. An architecture comparison is then made. The recommended module size is 32 bits because the kernel processor selected is a 32-bit processor.

5.2.1 Optimum Module Size

The lower and upper word length bounds are 8 bits and 64 bits, respectively. To determine the optimum number of bits per board, tradeoffs between speed desired, hardware used, glue logic required and board dimensions needed have been investigated.

An 8-bit module is inappropriate because a 64-bit word length, requires eight modules. As a solution, this architecture is far too bulky, heavy and slow. Moreover, excessive glue logic is required to route inter-board connections. A 64-bit module is also deemed inappropriate because an 8-bit word length causes too many resources to be idle, thereby yielding undesirable waste. This only leaves 16-bit and 32-bit modules to consider. Whether a 16- or a 32-bit module is used, it will support a 16-bit fixed-point and 32-bit floating-point incremental word length.

Looking at 16-bit and 32-bit modules, both offer different advantages. For the 16-bit module, wasted resources are minimized as the word length is varied. A smaller board is all that is needed to handle the shorter word length on each module. However, since the hardware requires four boards to provide the maximum word length, the overall size is greater. Since the hardware is land based, the board size is not as critical as other parameters (i.e. speed).

For the 32-bit module, less interboard connections are required. Including more logic on each board also minimizes propagation delay because interconnections are shorter. Less decoding glue logic is needed since all signals either stay on the first board or propagate to a second board. This is in contrast to the 16-bit module where the signal must choose a correct path to three other boards. A smaller quantity of boards are required to achieve the maximum capabilities of the system. Less space is required in the backplane because the boards will only take up two slots instead of four as is the case with 16-bit modules. As will be covered in a later section, the 32-bit module can also use faster parts to perform the same operations.

After examining the parameters related to module size (propagation delay, parts speed and interboard connections), 32 bits stands out as the best choice for incremental module size. This conclusion is supported by the architectural analysis described in the following sections.

5.2.2 Board Architectures

Three types of architectures have been identified which display the desired characteristics. The three architectures studied are: a cascadable ALU with external multiply and external divide support, a cascadable ALU with facilities on chip to implement multiply and divide, and a high speed ALU-Multiply chip-set with multiply and divide functions implemented within the set. These architectures are now discussed in further detail.

5.2.2.1 Multiplier/ALU Set (16 Bits) With Newton-Raphson Divide

The architecture involving a cascadable ALU with external multiply and divide support uses the IDT49C402 16-bit cascadable ALU and the IDT7210 16x16-bit multiplier. A table-look-up (with the appropriate glue logic) is also used to implement divisions. The divisions use an algorithm based on the Newton-Raphson method for computing roots of an equation. The architecture most logically supports a 16-bit module since both the ALU and the multiplier are 16 bits.

Cascadable additions and subtractions are implemented easily for all fixed-point wordlengths since the ALU is designed for that operation. However, the multiplies and divides are not supported by the cascadable ALU. A 16 x 16-bit multiply is performed by a 16 x 16-bit multiplier. For longer wordlengths, the module must combine special multiplies, shifts and adds. For the worst case, a 64 x 64-bit multiply is described using the 16 x 16-bit multipliers.

To multiply two 64-bit operands using 16 x 16-bit multipliers, the operation is divided into 16-bit sections. For example, if a_3 represents the most significant 16 bits of the 64 bit word "A", and a_0 represents the least significant 16 bits of the 64 bit word "A", the multiplication is partitioned into several submultiplies and adds as shown in Figure 13.

```
                         |<-  64 BITS  ->|
                          a3   a2   a1   a0     <- WORD A
                        x b3   b2   b1   b0     <- WORD B
                        ---------------------
                          a3b0 a2b0 a1b0 a0b0
                     a3b1 a2b1 a1b1 a0b1
                a3b2 a2b2 a1b2 a0b2
        +  a3b3 a2b3 a1b3 a0b3
        ---|<-------128 bit result---------->|
```

Figure 13. 64 x 64-bit Multiply.

Care must be taken when aligning the partial products according to their significance because each submultiply result (for instance a_0b_0) is 32 bits long. Therefore, the most significant 16 bits of a_0b_0 will be added to the least significant 16 bits of a_1b_0, and so on. The results, properly lined up, are shown in Figure 14. The term $a_0b_0a_0b_0$ represents the 32-bit result of multiplying a_0 by b_0, yielding the most significant 16 bits (leftmost a_0b_0) and the least significant 16 bits (rightmost a_0b_0).

```
                          a3   a2   a1   a0     <- word A
                        x b3   b2   b1   b0     <- word B
                        ---------------------
                                    a0b0a0b0
                                a1b0a1b0
                            a2b0a2b0
                        a3b0a3b0
                                a0b1a0b1
                            a1b1a1b1
                        a2b1a2b1
                    a3b1a3b1
                            a0b2a0b2
                        a1b2a1b2
                    a2b2a2b2
                a3b2a3b2
                        a0b3a0b3
                    a1b3a1b3
                a2b3a2b3
       +    a3b3a3b3
       -----|<-------128 bit result---------->|
```

Figure 14. The Proper Alignment of Partial Products.

For the least-significant 16 bits of the result, the least-significant half of a_0b_0 is dropped down. For the second least-significant 16 bits of the result, the most-significant part of a_0b_0 is added to the least-significant part of a_1b_0. The carry flag is then added to the next stage (the third vertical column of adds). This partial result is then added to the least-significant part of a_0b_1, again adding the carry flag to the next (third) stage. Continue to add the properly aligned submultiply results, as shown lined up vertically in Figure 14, to obtain the 128 bit result. If a 64 bit result is desired, the most significant 64 bits are used.

Each 64 x 64-bit multiply, therefore, requires 16 multiplies, 24 bit-slice adds, and 24 carry adds. A considerable amount of glue logic is required to move the data for numerical calculation. The number of carry adds can be reduced to 6 by incrementing a counter on each stage every time a carry occurs and adding the final carry count to the next most significant stage.

The divide operation takes the equation

$$C = A/B \tag{1}$$

and calculates the reciprocal of B to evaluate

$$C = A * (1/B). \tag{2}$$

The reciprocal calculation is based on the Newton-Raphson algorithm. The equation for each iteration [10] is

$$X_{i+1} = X_i * (2 - (B * X_i)). \tag{3}$$

A 64k x 16 table-lookup PROM will provide the first guess (seed) to start the process. The seed value must fall between

$$0 < \text{seed} < 2/B, \quad B > 0 \tag{4a}$$

and

$$0 > \text{seed} > 2/B, \quad B < 0 \tag{4b}$$

for the algorithm to converge. Iterations will continue until an acceptable error is achieved. The error is reduced quadratically for each iteration; therefore, the number of bits of accuracy roughly doubles each iteration. For example, the reciprocal of -.3 is -3.333333333 to ten decimal places. If the seed was -2.0, the error after each iteration is shown in Table 6.

Table 6. Error After Each Iteration (to Ten Decimal Places).

ITERATION	Xi	ERROR TO TEN DECIMAL PLACES
0	-2.0	1.333333333
1	-2.8	0.533333333
2	-3.248	0.085333333
3	-3.3311488	0.002184533
4	-3.333331902	0.000001431

For floating-point operations, a completely separate module must be developed. As it turns out, the architecture has almost the same block structure as the 32-bit BIT-chip module (to be discusses in section 5.2.2.3).

5.2.2.2 Architecture Using Cascaded 16-Bit ALU with Facilities for Multiply and Divide

The architecture shown in Figure 15 uses the AM29203 cascadable ALU with facilities on-chip to implement fixed-point multiply and divide. The module requires several processor interconnections that need extra glue logic.

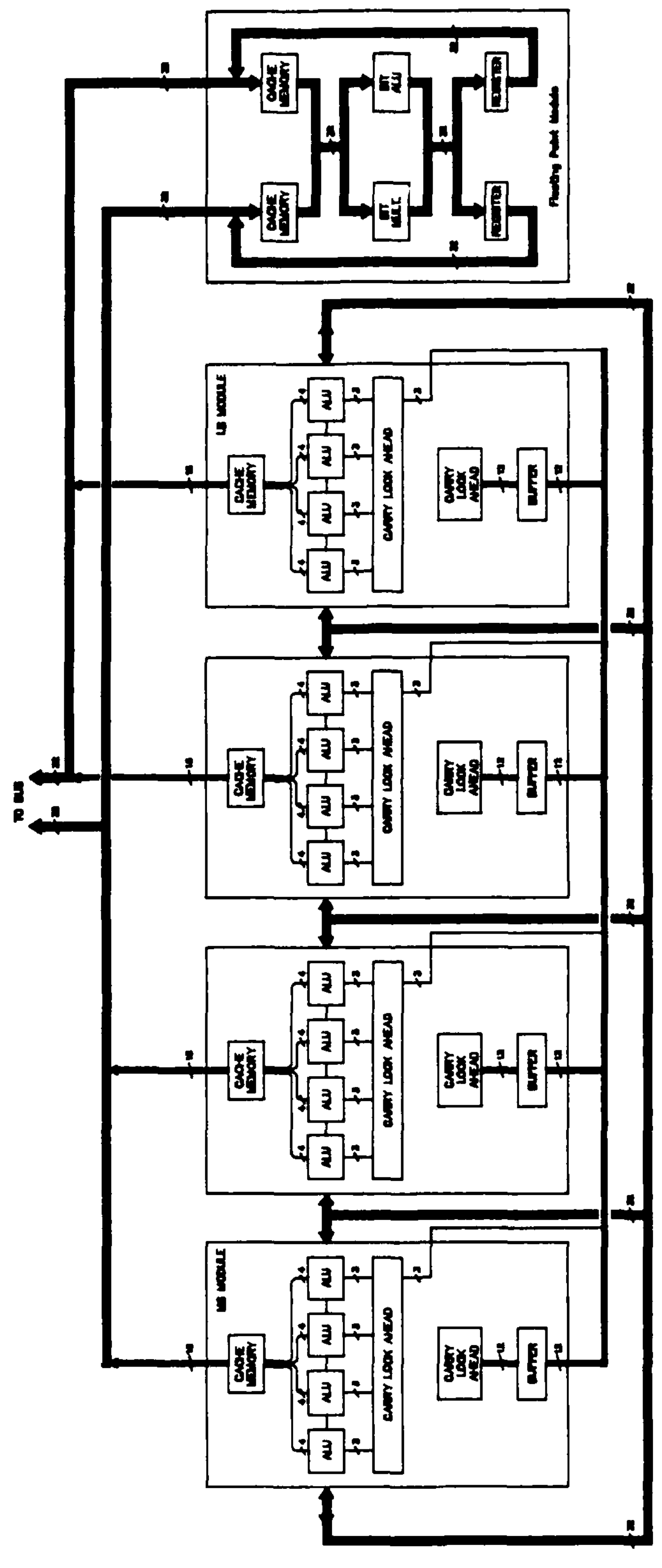


Figure 15. Architecture Using a Cascaded 16-Bit ALU with Facilities for Multiply and Divide.

It also requires numerous control lines to configure the selected wordlength on the architecture. This architecture also requires a separate floating-point module that will be discussed later. The fixed-point architecture is best suited to the 16-bit module size, since the ALU is 4-bits wide and can be easily cascaded to 16 bits with one carry-look-ahead generator.

Adds and subtracts are easily cascadable for all fixed-point wordlengths. Speed is increased by using a carry-look-ahead generator which allow carries to be anticipated and transferred to more-significant bit-slice sections without waiting for the carries to ripple through each stage. With four ALUs on each module, one carry-look-ahead generator is required for each module, as shown in Figure 16a [10]. If the modules are cascaded, one more carry-look-ahead level is required, as shown in Figure 16b [10]. Since all modules are identical, three of the four modules (for a 64-bit wordlength) must disable their second-level carry-look-ahead generator. Additional hardware is required to correctly route the generate, the propagate, and the carry lines to the active generator.

Unsigned and two's-complement multiplication are both supported by the AM29203. For an N x N-bit multiply, N cycles are required to calculate the result. Figure 17 [10] shows the algorithm for a 16 x 16 bit two's-complement multiply. Figure 18a and 18b [10] identifies processor interconnections required for the same multiply. The same structure is used for longer wordlengths, but it is expanded to include more bit-slices and to take more clock cycles.

The AM29203 also supports two's complement divide. As in the case for multiply, several cycles are required to calculate the result. Figure 19 [10] shows the algorithm for a two's-complement divide. Figure 20a and 20b [10] identifies processor interconnections required for the divide. The same structure is used for longer wordlengths, though it is transferred over more bit-slices and requires more clock cycles.

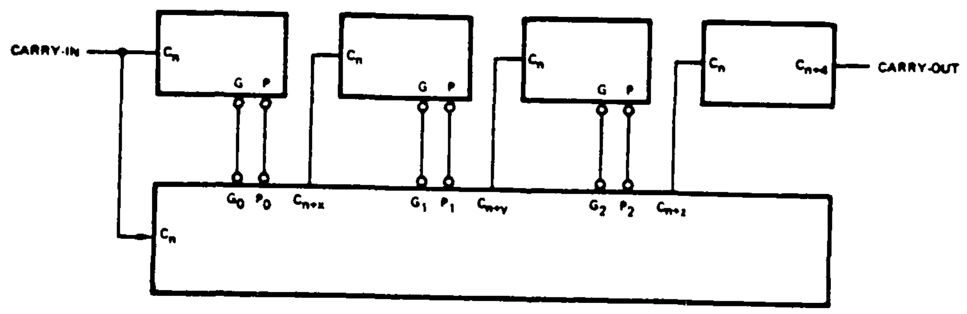


Figure 16a. One Level of Carry-Look-Ahead On A Module.

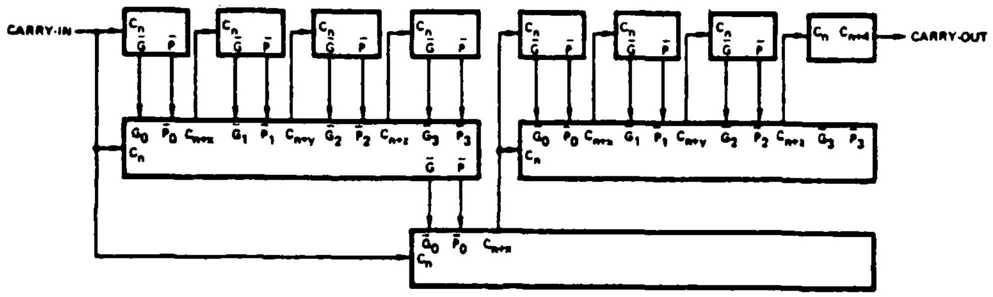

CARRY-IN
CARRY-OUT
C_n $\bar{G}$ $\bar{P}$
C_n C_{n+4}
$\bar{G}_0$ $\bar{P}_0$ C_{n+x} $\bar{G}_1$ $\bar{P}_1$ C_{n+y} $\bar{G}_2$ $\bar{P}_2$ C_{n+z} $\bar{G}_3$ $\bar{P}_3$
C_n
$\bar{G}$ $\bar{P}$

Figure 16b. Two Levels of Carry-Look-Ahead.

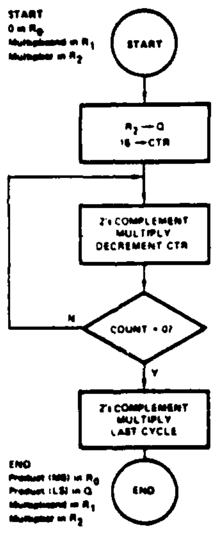


Figure 17. Flow Chart for Two's-Complement 16 x 16 Multiply.

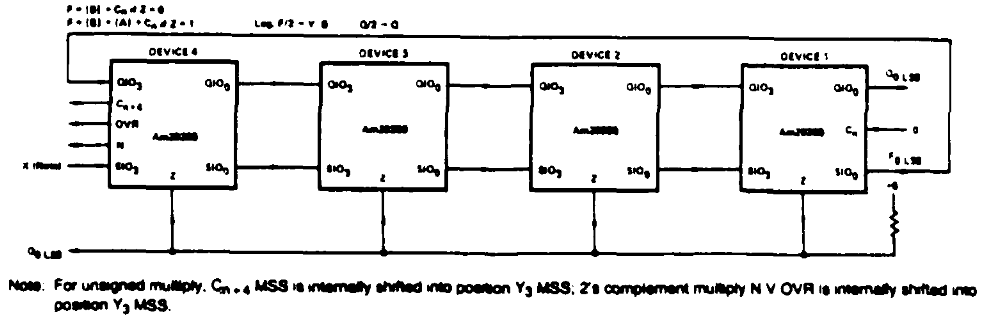

DEVICE 4
DEVICE 3
DEVICE 2
DEVICE 1
QIO_3
QIO_0
C_{n+4}
OVR
N
SIO_3
SIO_0
Z
Am29203
C_n
Q_0 LSB
F_0 LSB
Note: For unsigned multiply, C_{n+4} MSS is internally shifted into position Y_3 MSS; 2's complement multiply N V OVR is internally shifted into position Y_3 MSS.

Figure 18a. Processor Interconnections for First 15 Cycles of the 16 x 16-Bit Two's Complement Multiply and for All 16 Cycles of the 16 x 16-Bit Unsigned Multiply.

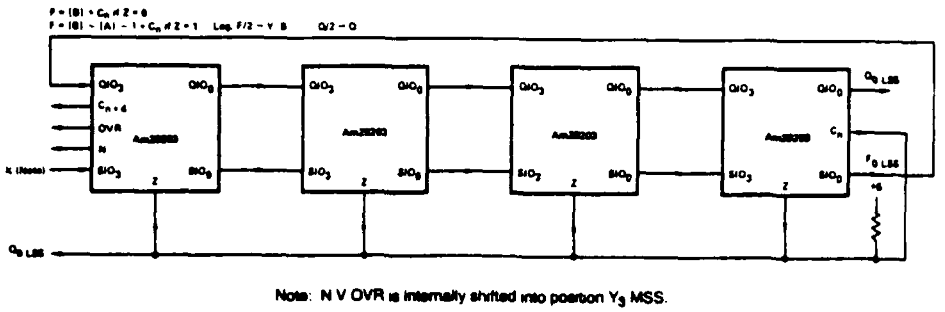

F = (B) + Cn if Z = 0
F = (B) − (A) − 1 + Cn if Z = 1   Log. F/2 → Y B   Q/2 → Q
QIO3
QIO0
Cn+4
OVR
N
Am29203
SIO3
SIO0
Z
X (Note)
Q0 LSS
Cn
F0 LSS
+5
Note: N V OVR is internally shifted into position Y3 MSS.

Figure 18b. Processor Interconnections for the 16 x 16-Bit Two's Complement Multiply, Last Cycle.

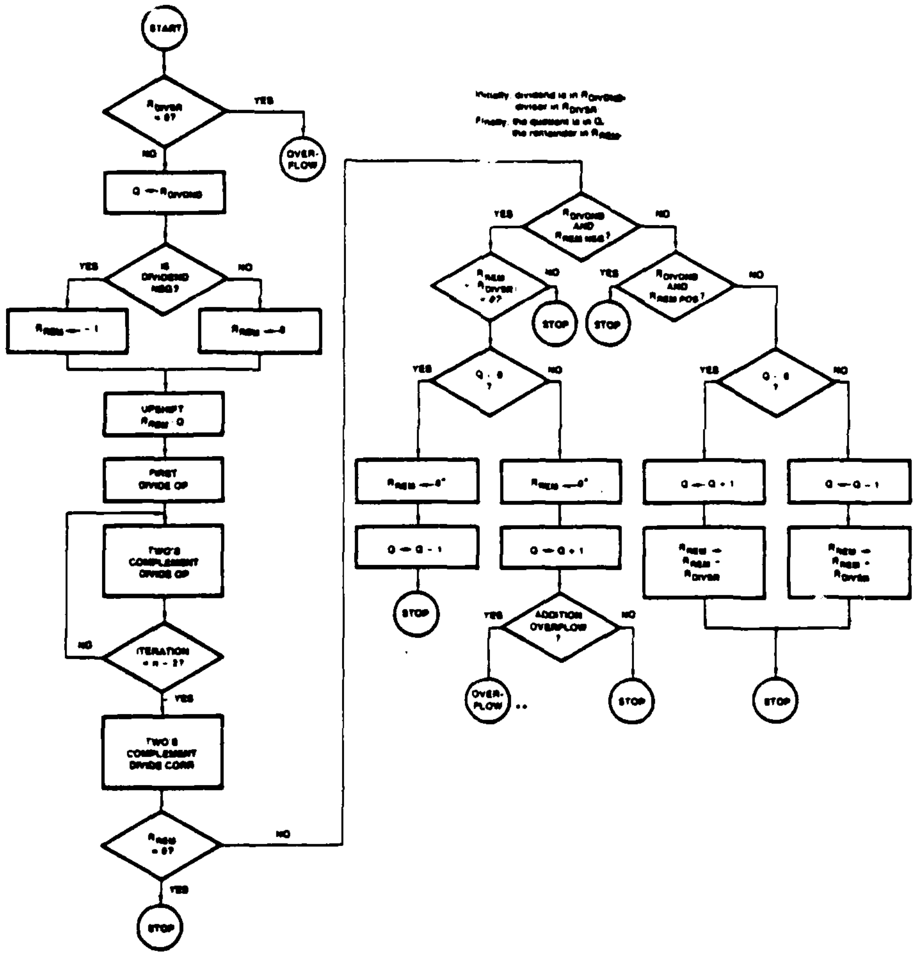


Figure 19. Flow Chart for Two's-Complement Divide.

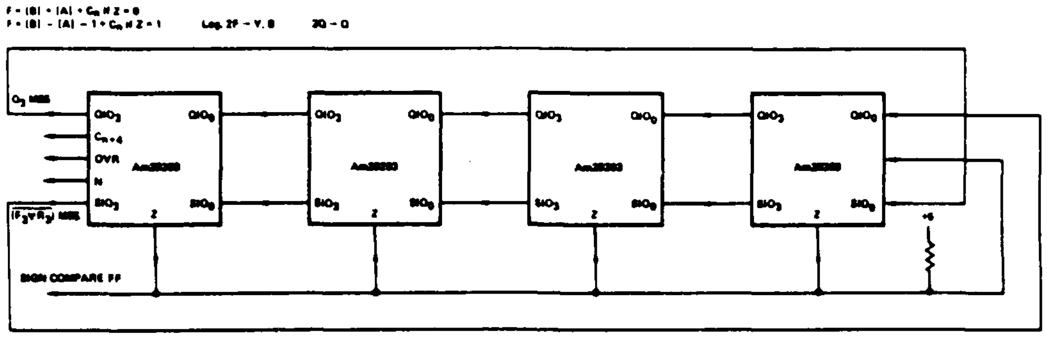


Figure 20a. Processor Interconnections for the Two's-Complement Divide.

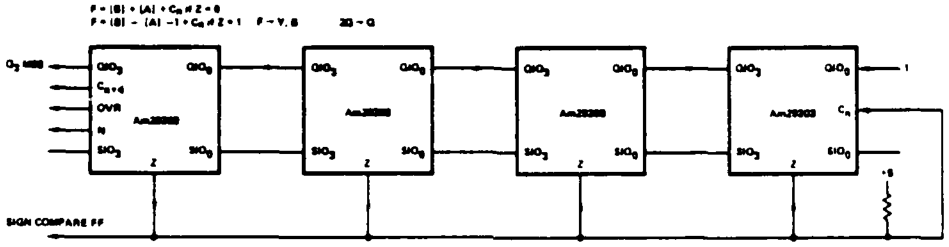


Figure 20b. Processor Interconnections for the Two's-Complement Divide Correction.

The AM29203 supports an 80 ns clock cycle, but for a memory-access/arithmetic-operation/memory-write sequence, several cycles are required. Table 7 lists the number of cycles and the time required for the sequence of operations from memory-read to memory-write. The times listed take into account only a single operation without pipelining.

Table 7. Fixed-Point Operation Times.

WORD	NUMBER OF CYCLES				TIME (ns)			
LENGTH	+	-	*	/	+	-	*	/
16	4	4	20	30	240	240	1200	1800
32	4	4	36	46	240	240	2160	2760
48	4	4	52	62	240	240	3120	3720
64	4	4	68	78	240	240	4080	4680

5.2.2.3 Architecture Using High-Speed ALU-Multiply Chip Set with Multiply and Divide

The architecture involving a high speed ALU-Multiply chip-set with integrated multiply and divide functions, shown in Figure 21, uses the B2110/B2120 Multiplier/ALU from Bipolar Integrated Technologies.

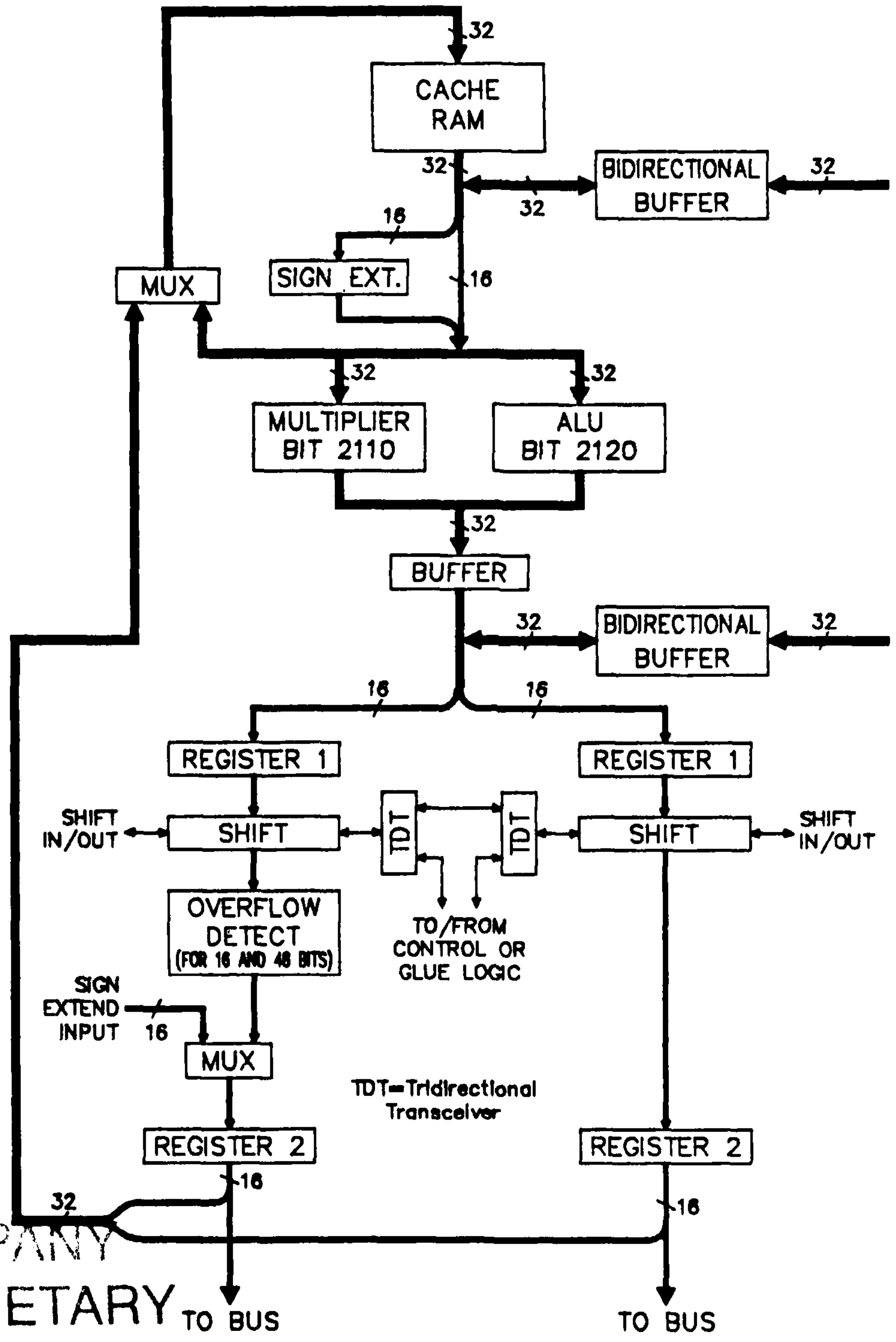


Figure 21. Architecture Using a High-Speed ALU-Multiply Chip Set with Multiply and Divide.

This architecture requires only two 32-bit buses and a couple of shift input and output lines between modules. Best of all, this architecture does not require a separate floating-point module. The chip set performs fixed-point add, subtract, multiply, boolean functions, shift and rotate. It also performs floating-point add, subtract, multiply, divide, square root, absolute value, negative, min/max and compare. The set also provides conversions between fixed- and floating-point [11]. The architecture most logically supports a 32-bit module since the ALU/Multiplier pair have 32-bit data lines.

The chip-set does not have its own internal register file, but dual-port RAM can be employed to act as a register file and cache memory. Integrated Device Technologies manufactures a fast (35 ns access time) dual port memory (IDT7132/7142) that would be ideal for this application. Bipolar Integrated Technology manufactures a five-port (2 read, 2 write, 1 read/write) register file that has a clocked read cycle of 12 ns (typical). It is organized as 64 18-bit registers [12]. The BIT register file is inefficient because there is no need for five ports, but it can still be implemented.

Even though the BIT chip-set is not designed to be cascadable, it is so versatile and fast that it handles 32- and 64-bit fixed-point and 32- and 64-bit floating-point add, subtract, multiply and divide without any extra support hardware or glue logic. However, for 8-, 16-, and 48-bit fixed-point wordlengths, operations are performed on either 32- or 64-bit data and extra hardware interprets the data as the correct length. One big advantage to the architecture introduced in Figure 21 is that add, subtract, multiply, and divide can be performed on the chip itself without the need to reconfigured the architecture to perform the operation off the chip.

For all 16-bit fixed-point instructions, the data is sign-extended as it is loaded into the processors. Therefore, the processor can correctly handle the 16-bit data as 32 bits. The processors then execute the instruction and pass the result on to the shift stage. The least significant 16 bits are on the right. All shifts and rotates are done at this point, instead of on the chip, since the most significant bit can be accessed through the tri-directional transceiver. As the data proceeds through the architecture, overflow detection hardware checks the most-significant 16 bits for data overflow.

All 32-bit fixed-point instructions are handled with one board, much the same way as in the 16-bit fixed-point case. However, minor differences exist. The sign extend and overflow-detect functions (not including the actual chip's overflow detect) are disabled. Also, the tri-directional transceivers allow direct communication between the shift sections.

For 48- and 64-bit wordlengths, two boards are required, connected as shown in Figure 22. The right module handles the least-significant 32 bits. For 48- and 64-bit fixed-point subtract, one operand is inverted and added to the second operand,then a 1 is added to the result. During addition, the least-significant 32 bits are added independently from the most-significant 16 or 32 bits. Once these adds have occurred, the carry from the least significant board is added to the most significant board.

In the case where 48- and 64-bit fixed-point multiply and divide are implemented, the architecture is arranged as shown in Figure 22. When performing these operations, the least-significant module loads its 32 bits (through the top bi-directional buffer) into the most-significant module's multiplier chip. The most significant module then sends its 16 or 32 bits into the multiplier which performs a double-precision multiply or divide. The least-significant 32 bits are broadcast back to the least-significant module (through the bottom bidirectional buffer) and stored into register 1. The most-significant bits stay in the most-significant module, and are transmitted to the shift stage at the same time the least significant bits are sent to the shift stage.

Register 2 on the board is used when the bus width is 32 bits and a longer wordlength is used. Both modules supply their 32 bits to register 2 at the same time. One module then stores its data into register 2 while the other module accesses the bus. The following clock cycle, the second operand is then written to the bus.

In the case where 32-bit floating-point adds, subtracts, multiplies and divides are implemented, the architecture is arranged as previously shown in Figure 21. When performing these operations, only one board is required and the hardware beyond register 1 is unnecessary, since all functions to be performed are done by the BIT chip-set. However, this hardware is still used in other configurations.

When 64-bit floating-point adds, subtracts, multiplies and divides are implemented, the architecture is arranged as was shown in Figure 22. When performing these operations, the least-significant module loads its 32 bits into the most-significant-module's multiplier/ALU. The most-significant module then sends its 32 bits into the multiplier/ALU, which performs a double-precision operation. The least-significant 32 bits are broadcast back to the least-significant module and stored into register 1. The most-significant bits stay in the most significant module and the rest follows the same pattern as in the 48- and 64-bit fixed-point configuration already discussed.

Comparing the floating-point hardware shown in Figure 21 and Figure 22 to the floating-point hardware required for the previously discussed architectures, the only difference is the hardware beyond register 1. All other interconnections are the same. It makes little sense to have the same architecture discussed in this section implement only floating-point operations and a completely new architecture perform the fixed-point functions, when a single architecture can do it all with a small increase in hardware.

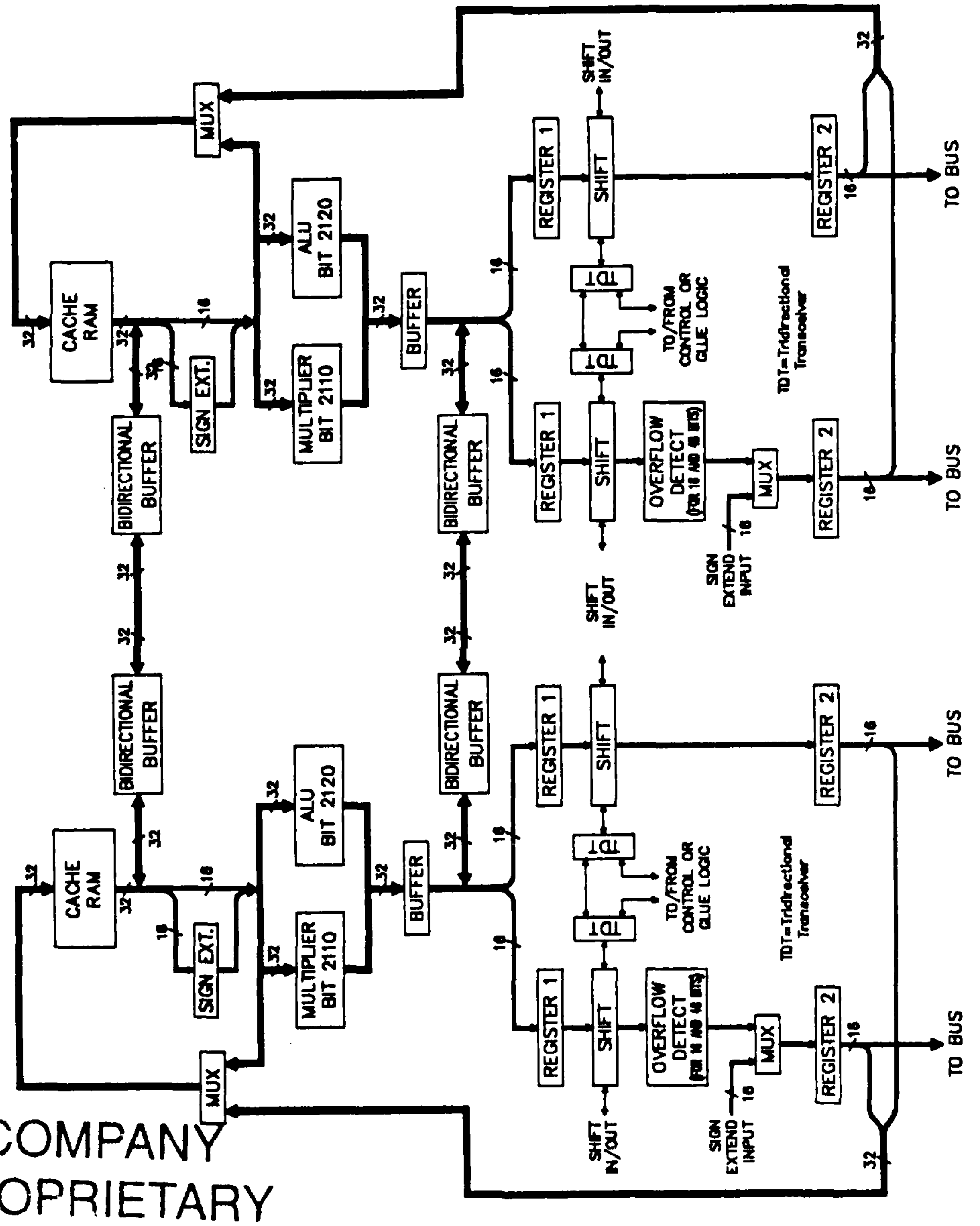


Figure 22. Two-Board Architecture for the 32-Bit Module.

The B2110/2120 chip set supports a 50 ns clock cycle; but for a memory-access/arithmetic-operation/memory-write sequence, several cycles are required. Table 8 lists the number of cycles and the time required for the fixed-point sequence of operations from memory-read to memory-write. Table 9 lists the same parameters only for floating-point operations. The times listed take into account only a single operation without pipelining.

Table 8. Fixed-Point Operation Times.

WORD	NUMBER OF CYCLES				TIME (ns)			
LENGTH	+	-	*	/	+	-	*	/
16	5	5	5	9	250	250	250	450
32	5	5	5	9	250	250	250	450
48	9	11	9	15	450	550	450	750
64	9	11	9	15	450	550	450	750

Table 9. Floating-Point Operation Times.

WORD	NUMBER OF CYCLES				TIME (ns)			
LENGTH	+	-	*	/	+	-	*	/
32	5	5	5	7	250	250	250	350
64	8	8	9	11	400	400	450	550

5.2.3 Architecture Comparisons

The architectures have been compared at the hardware level in the previous sections. The architectures still offering potential are now compared on the basis of execution speed. Table 10 compares board speed of the 16-bit AMD bit-slice module with the 32-bit BIT fixed- and floating-point module. Table 10 also lists the number of operations performed in each algorithm.

Table 10. Speed Comparisons.

NUMBER	DATA TYPE	ALGORITHM	#ADDS	#MULTS	#DIV	FIXED-POINT 16 BIT MODULE 16 BITS	FIXED-POINT 16 BIT MODULE 64 BITS
1	CMPX	1024 RADIX 2 FFT	30720	20480	0	31.95 ms	90.93 ms
2	REAL	4X4 MATRIX MULT	48	64	0	88.32 us	272.6 us
3	REAL	10X10 MATRIX MULT	900	1000	0	1.416 ms	4.296 ms
4	REAL	64X64 MATRIX MULT	257919	262144	0	376.5 ms	1.131 sec
5	REAL	4 X 4 MATRIX SCALE	0	16	1	21.0 us	69.96 us
6	REAL	64X64 MATRIX SCALE	0	4096	1	4.192 ms	16.72 ms
7	CMPX	COVARIANCE MATRIX GEN.	65536	131072	0	173.0 ms	550.5 ms
8	CMPX	32X32 LU DECOMPOSITION	29301	30229	32	43.36 ms	130.5 ms
9	CMPX	32X32 GEN. MAT. INVERT	156277	161173	496	231.8 ms	697.4 ms
10	CMPX	16 TAP FIR	64	64	0	92.16 us	276.5 us
11	REAL	16 TAP FIR	16	16	0	23.04 us	69.12 us
12	REAL	64 X 64 MATRIX ADD	4096	0	0	983.0 us	983.0 us
13	CMPX	32 X 32 MATRIX ADD	4096	0	0	983.0 us	983.0 us
14	REAL	1024 POINT VECTOR ADD	1024	0	0	245.8 us	245.8 us
15	CMPX	1024 POINT VECTOR ADD	2048	0	0	491.6 us	491.6 us
16	REAL	1024 VEC. MULT (SCALAR)	1023	1024	0	1.470 ms	4.400 ms
17	REAL	1024 VEC. MULT (VECTOR)	0	1024	0	1.230 ms	4.180 ms
18	CMPX	1024 VEC. MULT (SCALAR)	4095	4096	0	5.898 ms	17.69 ms
19	CMPX	1024 VEC. MULT (VECTOR)	2048	4096	0	5.410 ms	17.20 ms
20	CMPX	1024 CONVOLUTION	92160	62464	0	97.08 ms	277.0 ms

Table 10. (Continued).

NUMBER	FLOATING-POINT 16 BIT MODULE 32 BITS	FLOATING-POINT 16 BIT MODULE 64 BITS	FIXED-POINT 32 BIT MODULE 16 BITS	FIXED-POINT 32 BIT MODULE 64 BITS	FLOATING-POINT 32 BIT MODULE 32 BITS	FLOATING-POINT 32 BIT MODULE 64 BITS
1	10.24 ms	18.94 ms	12.80 ms	24.58 ms	12.80 ms	21.50 ms
2	22.40 us	42.40 us	28.00 us	52.80 us	28.00 us	48.00 us
3	380.0 us	715.0 us	475.0 us	900.0 us	475.0 us	810.0 us
4	104.0 ms	195.1 ms	130.0 ms	246.9 ms	130.0 ms	221.1 ms
5	3.500 us	6.900 us	4.450 us	7.950 us	4.350 us	7.750 us
6	819.5 us	1.639 ms	1.020 ms	1.840 ms	1.020 ms	1.840 ms
7	39.32 ms	75.37 ms	49.15 ms	91.75 ms	49.15 ms	85.20 ms
8	11.92 ms	22.36 ms	14.90 ms	28.28 ms	14.89 ms	25.34 ms
9	63.64 ms	119.4 ms	79.59 ms	151.0 ms	79.54 ms	135.3 ms
10	25.60 us	48.00 us	32.00 us	60.80 us	32.00 us	54.40 us
11	6.400 us	12.00 us	8.000 us	15.20 us	8.000 us	13.60 us
12	819.2 us	1.430 ms	1.020 ms	2.050 ms	1.020 ms	1.640 ms
13	819.2 us	1.430 ms	1.020 ms	2.050 ms	1.020 ms	1.640 ms
14	204.8 us	358.4 us	256.0 us	512.0 us	256.0 us	409.6 us
15	204.8 us	358.4 us	256.0 us	512.0 us	256.0 us	409.6 us
16	409.4 us	767.7 us	511.8 us	972.3 us	511.8 us	870.0 us
17	204.8 us	409.6 us	256.0 us	460.8 us	256.0 us	460.8 us
18	1.638 ms	3.070 ms	2.050 ms	3.890 ms	2.050 ms	3.480 ms
19	1.230 ms	2.360 ms	1.540 ms	2.870 ms	1.540 ms	2.660 ms
20	30.92 ms	57.24 ms	38.66 ms	74.19 ms	38.66 ms	64.97 ms

5.2.4 CPH Microcode Layout

The CPH preliminary architecture design, shown in Figure 20, requires 115 bits of microcode to control. Figure 23 maps out the bit pattern showing bit position, field name, subfield name and bit description. The 115 bit word is broken into the seven fields shown in Table 11.

Table 11. Fields of the CPH Microcode Wor`.

	FIELD NAME	MNEMONIC	# SUBFIELDS	# BITS
1.	Sequencer	SEQ	5	21
2.	Address Generation	AG	10	31
3.	Buffers	BUF	3	5
4.	Multiplier/ALU	MA	4	16
5.	Shift	SHIFT	6	18
6.	Overflow	OV	1	3
7.	Cache Bus Control	BUS	5	21
	Totals		34	115

The sequencer field is responsible for controlling program-flow using the IDT49C410 (preliminary selection) microprogram sequencer. The IDT49C410 is capable of addressing up to 65,536 words of microprogram memory, however the initial design is only accessing 2048 words. This depth is expandable by adding extra bits to the branch-addr/loop-counter subfield and increasing the number of address lines. The 16 available instructions allow versatile sequencing including branch, loop, jump, etc. throughout all of the available microprogram memory. A 33-deep stack is also available which allows nested looping. For conditional branching, 16 possible conditions/flags can be selected from microcode for increased versatility.

The address generation field provides efficient address sequencing for signal- and matrix-processing in addition to efficiently generating addresses unrelated to previously generated addresses. The 17 most significant bits in the AG field are dedicated to FFT addressing. DIT/DIF, radix 2/4, and pre-scrambled-data algorithms are supported up to length 64k. Special programming considerations are required to allow larger transforms. The remaining 14 microcontrol bits instruct a two-dimensional counter on how to increment to th^ next location based on the previous address generated. The field also monitors the cache control lines.

The buffer field is responsible for configuring the architecture correctly when two CPH boards are incorporated. The buffers eliminate the possibility of enabling two sources of data onto the same bus. When only one board is utilized, the buffers are always enabled.

The multiplier/ALU field controls processor instructions and processor enabling. The most significant subfield enables the sign extend hardware, which is used when operating on 16- or 48-bit wordlengths. The instruction subfield selects the proper arithmetic, shift, rotate, compare or boolean operation. The remaining subfields properly enable data in-to and out-of the multiplier and ALU chip-set.

The shift field sets up the control for the 74AS897 (preliminary selection) 16-bit barrel shifter. Several subfields are available to enable and determine what values are shifted in. The remaining subfields select the shift operation to be performed.

The overflow field selects the appropriate correction to be performed if an overflow occurs. The overflow-detect hardware is only used when manipulating 16- and 48-bit wordlengths. The BIT chips have their own internal overflow-detect for 32- and 64-bit wordlengths. As a result, there is no need to enable the additional correction hardware for these wordlengths.

The cache bus sub-field controls the data written into the cache memory. Registers are included to hold data until the bus is ready. Moreover, when two boards are cascaded together, the least-significant 32 bits can be transferred on the system bus in one cycle while the most-significant 32 bits are transferred on the system bus in the following cycle. The sign extend input extends the sign while manipulating 16- and 48-bit wordlengths. This allows error-free wordlength changes during execution time. Immediate data can be written into the cache directly from microcode using the sign extend input. This increases flexibility and should also aid in the test and debug stages. Additional subfields are available to properly route the data to the cache memory or onto the system bus.

The microcode EPROMS shall be placed as close as possible to the chips they control. This distributed control technique minimizes propagation delays in addition to minimizing bus interconnections. The 11-bit-wide address is transmitted to each of the control chips as opposed to transmitting all 115 bits from a central area all over the boards as would be the case in a centralized control.

5.3 EVA Architectural Recommendations

Extensive comparative analyses and continuous interfacing with WSMR technical contacts have helped identify the major I.C. building blocks for the EVA architecture. It has been concluded that the B2110/B2120 chip set from Bipolar Integrated Technologies and the ZR34325 from ZORAN Corporation offer high speed and flexibility for the EVA number-crunching requirements.

In the case study of a 1024-point complex FFT, a single VSP-325 configuration executes the algorithm in approximately 1.7 milli-seconds. In the dual VSP-325 configuration, the 1024-point complex FFT is executed in approximately 880 micro-seconds. For the quad VSP-325 configuration with two parallel buses, the execution time is approximately 460 micro-seconds. This is one of the fastest, if not the fastest, real-time floating-point signal processors available. Equally impressive benchmarks are possible if the B2110/B2120 chip set is employed for the CPH. After considering the trade-offs between the speed desired, amount of hardware required, and board size needed, a 32-bit incremental module size employing the BIT chip set is recommended. The primary reason is that fixed- and floating-point operations are supported by the same processors, thus maximizing efficiency, throughput, and versatility.

Bit	Field	Subfield group	Unit
114	D10	BRANCH ADDRESS/LOOP COUNTER	SEQUENCER (SEQ)
113	D9		
112	D8		
111	D7		
110	D6		
109	D5		
108	D4		
107	D3		
106	D2		
105	D1		
104	D0		
103	I3	INSTRUCTION	
102	I2		
101	I1		
100	I0		
99	$\overline{\text{CCEN}}$	$\overline{\text{CCEN}}$	
98	$\overline{\text{RLD}}$	$\overline{\text{RLD}}$	
97	D	CONDITION CODE SELECT	
96	C		
95	B		
94	A		
93	I1	FFT INSTRUCTION	ADDRESS GENERATION (AG)
92	I0		
91	$\overline{\text{OE}}$		
90	AS3	FFT ADDR. SELECT	
89	AS2		
88	AS1		
87	AS0		
86	TL3	FFT TRANSFORM LENGTH	
85	TL2		
84	TL1		
83	TL0		
82	TSEL		
81	RADIX	FFT RADIX	
80	$\overline{\text{PSD}}$	FFT TRANSFORM	
79	DIT/DIF		
78	$\overline{\text{SEL}}$		
77	DIMENSION		
76	UP/DW1	2-D COUNTER	ADDRESS GENERATION (AG)
75	LOAD1		
74	HOLD1		
73	UP/DW2		
72	LOAD2		
71	HOLD2		
70	REG1EN	2-D REG EN	
69	REG2EN		
68	MUXSEL B	BIT MASK	
67	MUXSEL A		
66	LOADEN		
65	MUXSEL A	CACHE MUX	
64	READ/WRITE	CACHE CONTROL	
63	OUTEN		
62	DIR	TOP BI-DIR	BUFFERS (BUF)
61	EN		
60	DIR	BOTTOM BI-DIR	
59	EN		
58	EN	BUFFER FEEDBACK	
57	SELECT	SIGN EXTEND	MULTIPLIER/ALU (MA)
56	XSEL	X,Y SOURCE	
55	XEN		
54	MSWEN		
53	YEN		
52	I7	INSTRUCTION	
51	I6		
50	I5		
49	I4		
48	I3		
47	I2		
46	I1		
45	I0		
44	RESET		
43	MSWSEL	T OUT	
42	ZEN		
41	EN	REGISTER	SHIFT
40	BIT 31	SHIFT IN/OUT	
39	BIT 16		
38	BIT 15	SHIFT IN/OUT	SHIFT (SHIFT)
37	BIT 0		
36	M2	SHIFT INSTRUCTION	
35	M1		
34	M0		
33	IP	SELECT	
32	OP		
31	$\overline{\text{NORM}}$	CONTROL	
30	HEX/BIN		
29	S		
28	$\overline{\text{TP}}$		
27	$\overline{\text{OEY}}$		
26	DIR	TDT	
25	I1		
24	I0		
23	I1	INSTRUCTION	OVERFLOW (OV)
22	I0		
21	EN		
20	LI 15	LOAD IMMEDIATE	CACHE BUS CONTROL (BUS)
19	LI 14		
18	LI 13		
17	LI 12		
16	LI 11		
15	LI 10		
14	LI 9		
13	LI 8		
12	LI 7		
11	LI 6		
10	LI 5		
9	LI 4		
8	LI 3		
7	LI 2		
6	LI 1		
5	LI 0		
4	I1	INSTRUCTION	
3	I0		
2	EN	EN	
1	EN	S.E. MUX	
0	EN	REG 2	

Figure 23. Microcode Layout for the CPH.

6.0 Applications of EVA

6.1 Kalman Filter Realization

The Kalman filter is very useful for radar tracking systems using phased array techniques. Performing the algorithm in real-time is very desirable. The Cascadable Processing Element and the Vector Processing Element were examined to see if either could execute the Kalman filter efficiently in real-time. A scheme is described that will allow real time calculation by assuming some values to be constant for several samples observed.

6.1.1 Kalman Filter

Kalman filters have been implemented in many phased array radar tracking systems. The actual target parameters estimated are given in the state equation

$$\underline{x}(K+1) = A\ \underline{x}(K) + B\ \underline{u}(K). \quad (5)$$

The output equation

$$\underline{y}(K) = C\ \underline{x}(K) + \underline{v}(K) \quad (6)$$

yields the observed parameters from the radar system that approximate the actual parameters.

The Kalman filter is very computation intensive [13], requiring five equations to perform the algorithm

$$\underline{X}'_K = A\ \underline{X}''_{K-1} \quad (7)$$

$$P_b(K) = A\ P_a(K-1)\ A^T + M \quad (8)$$

$$K(K) = P_b(K)\ C^T\ [C\ P_b(K)\ C^T + Q_V]^{-1} \quad (9)$$

$$\underline{X}''(K) = \underline{X}'_K + K(K)\ [\underline{Y}(K) - C\ \underline{X}'_K] \quad (10)$$

$$P_a(K) = [I - K(K)\ C]\ P_b(K), \quad (11)$$

where

$$M = B\ Qu\ B^T \quad (12)$$

Equations (7)-(11) form the Kalman filter where (7) is an initial estimate, (8) is the error correlation before an update, (9) is the Kalman gain, (10) is the final estimate, and (11) is the error correlation after update. The components of each equation are explained in Table 12.

Table 12. Kalman Filter Equation Components.

ELEMENT NAME	ELEMENT DESCRIPTION	DIMENSION
u(K)	= RANDOM INPUT	nx1
v(K)	= ADDITIVE NOISE	mx1
x(K)	= STATE VECTOR	nx1
y(K)	= OBSERVED VECTOR	mx1
n	= NUMBER OF ACTUAL STATES	n
m	= NUMBER OF OBSERVED STATES	m
A	= CONSTANT FROM STATE EQUATION	nxn
B	= CONSTANT FROM STATE EQUATION	nxn
C	= CONSTANT FROM OUTPUT EQUATION	nxm
M	= TEMPORARY CONSTANT	nxn
Qu	= EXPECTED VALUE OF u (AUTOCORRELATION)	nxn
Qv	= EXPECTED VALUE OF v (AUTOCORRELATION)	mxm
x(K+1)	= NEW VALUE OF STATE VECTOR	nx1

Note A, B, C, Qu, Qv, and M are predetermined constants and no calculation is required inside the algorithm.

Real-time Kalman filter processing is desirable to provide the results instantaneously. Technical contacts at WSMR have suggested a 50 kHz sampling rate for this study. Since the filter is matrix- and vector-operation intensive, numerous calculations are required. Techniques to decouple the Kalman filter were Investigated.

The equations can not be calculated in parallel because (9), (10), and (11) depend on the results calculated in (8), (7), and (8) respectively. If one module was to calculate the five equations, it would make sense to calculate (7) first, then sequentially through (11). The number of operations for each equation is shown in Table 13.

Table 13. Number of Operations In The Kalman Filter Equations.

EQUATION	OPERATION	# MULTIPLIES	# ADDS	# RECIPROCALS
(7)	[nxn] [nx1] mult	n^2	$(n-1)n$	0
(8)	[nxn] [nxn] mult	n^3	$(n-1)n^2$	0
	[nxn] [nxn] mult	n^3	$(n-1)n^2$	0
	[nxn] [nxn] add	0	n^2	0
(9)	[nxn] [nxm] mult	n^2m	$(n-1)nm$	0
	[mxn] [nxm] mult	m^2n	$(n-1)m^2$	0
	[mxm] [mxm] add	0	m^2	0
	[mxm] inverse*	$(5/6)m^3 + 3m^2 -(29/6)m + 5$	$(5/6)m^3 + 2m^2 -(11/6)m + 5$	m
	[nxm] [mxm] mult	m^2n	$(m-1)nm$	0
(10)	[mxn] [nx1] mult	mn	$(n-1)m$	0
	[mx1] [mx1] sub	0	m	0
	[nxm] [mx1] mult	nm	$(m-1)n$	0
	[nx1] [nx1] add	0	n	0
(11)	[nxm] [mxn] mult	n^2m	$(m-1)n^2$	0
	[nxn] [nxn] sub	0	n^2	0
	[nxn] [nxn] mult	n^3	$(n-1)n^2$	0

* [14]

Adding all of the operations together, there are

$$(5/6)m^3+m^2(2n+3)+m(2n^2+2n-(29/6))+3n^3+n^2+5 \qquad (13)$$

total multiplies and

$$(5/6)m^3+m^2(2n+2)+m(2n^2-(11/6))+3n^3-n^2-n+5 \qquad (14)$$

total additions and

$$m \qquad (15)$$

total reciprocals. No additional calculations are required for matrix transpose since special addressing can perform the transpose operation. In the worst case, m is set equal to n making (13) equal to

$$(47/6)m^3+6m^2-(29/6)m+5 \qquad (16)$$

and (14) equal to

$$(47/6)m^3+m^2-(17/6)m+5. \qquad (17)$$

The Vector Processing Element (VPH) takes

$$12m^3 + 56m^2 - 8m + 76 \qquad (18)$$

operations to perform the Kalman filter algorithm which uses an 80 ns clock cycle. The VPH has special cases for matrices larger than 8x8 because the chip has a limited amount of memory for the matrix elements causing more overhead operations, so (18) is only true for m=n less than 9.

Table 14 compares the required calculation time for different dimensions of state vectors (m and n) for the proposed architectures. Table 15 lists what sampling rate could be achieved for each size of state vector and also how many boards would be required for a 50 kHz sampling rate.

Table 14. Calculation Time Required for Several Dimensions of State Vectors for The Full Algorithm.

SIZE (n=m)	CASCADABLE 16-BIT FIXED-POINT FORMAT	CASCADABLE 32-BIT FLOATING-POINT FORMAT	VECTOR PROCESSING ELEMENT
1	21.2 us	6.7 us	10.88 us
2	117.8 us	37.9 us	30.40 us
3	364.7 us	119.6 us	70.40 us
4	829.4 us	275.3 us	136.6 us
5	1.6 ms	528.5 us	234.9 us
6	2.7 ms	902.7 us	370.9 us
7	4.2 ms	1.4 ms	550.4 us
8	6.2 ms	2.1 ms	779.2 us
9	8.8 ms	3.0 ms	945.8 us
10	12.0 ms	4.1 ms	1260. us

Table 15. Achievable Sampling Rates With One Cascadable Module and Number of Modules Required For A 50 kHz Sampling Rate for The Full Algorithm.

SIZE (N=M)	CPE FIXED-PT SAMPLING RATE (Hz)	CPE FLOAT-PT SAMPLING RATE (Hz)	VPH SAMPLING RATE (Hz)	BOARDS REQUIRED FOR 50 kHz SAMPLING: FIX-PT FORMAT	BOARDS REQUIRED FOR 50 kHz SAMPLING: FLOAT-PT FORMAT	BOARDS REQUIRED FOR 50 kHz SAMPLING: VECTOR PROC. ELEMENT
1	47k	149k	91.91K	2	1	1
2	8.5k	26.3k	32.9K	6	2	2
3	2.74k	8.4k	14.2K	19	6	4
4	1.2k	3.6k	7.3K	42	14	7
5	633	1.9k	4.3K	84	27	12
6	373	1.1k	2.7K	134	46	19
7	238	704	1.8K	211	72	28
8	161	476	1.3K	311	105	39
9	114	335	1.1K	440	150	48
10	83	245	.79K	599	205	63

Obviously, for a typical state vector size of around 6, it is impractical to implement the entire algorithm on the VPH or the CPE in real time. Too many modules are required. Another method to implement the Kalman filter is to assume (8), (9) and (11) are constant over several samples. This eliminates a large number of calculations for each sample. Only $3m^2$ multiplies and $(3m^2 - m)$ additions are required for each sample. The price paid for this increase in speed is a decrease in accuracy. Table 16 compares the required calculation time for different dimensions of state vectors (m and n). Table 17 lists what sampling rate could be achieved for each size of state vector and also the number of boards required for a 50 kHz sampling rate.

Table 16. Calculation Time Required for Several Dimensions of State Vectors for The Approximated Algorithm.

SIZE (n=m)	CASCADABLE 16-BIT FIXED-POINT FORMAT	CASCADABLE 32-BIT FLOATING-POINT FORMAT	VECTOR PROCESSING ELEMENT
1	4.1 us	1.3 us	3.2 us
2	16.8 us	5.5 us	9.0 us
3	38.2 us	12.8 us	20.3 us
4	68.2 us	23.0 us	38.7 us
5	106.8 us	36.3 us	65.6 us
6	154.1 us	52.5 us	102.4 us
7	210.0 us	71.8 us	150.6 us
8	274.6 us	94.0 us	211.5 us
9	347.8 us	119.3 us	255.0 us
10	429.6 us	147.5 us	337.9 us

Table 17. Achievable Sampling Rates With One Cascadable Module and Number of Modules Required For A 50 kHz Sampling Rate for The Approximated Algorithm.

SIZE (N=M)	CPE FIXED-PT SAMPLING RATE (Hz)	CPE FLOAT-PT SAMPLING RATE (Hz)	VPH SAMPLING RATE (Hz)	BOARDS REQUIRED FOR 50 kHz SAMPLING: FIX-PT FORMAT	FLOAT-PT FORMAT	VECTOR PROC. ELEMENT
1	245k	800k	312.5k	1	1	1
2	59.5k	181.8k	111.6k	1	1	1
3	26.2k	78.4k	49.2k	2	1	2
4	14.7k	43.5k	25.8k	4	2	2
5	9.4k	27.6k	15.2k	6	2	4
6	6.5k	19.1k	9.8k	8	3	6
7	4.8k	13.9k	6.6k	11	4	8
8	3.6k	10.6k	4.7k	14	5	11
9	2.9k	8.4k	3.9k	18	6	13
10	2.3k	6.8k	3.0k	22	8	17

The estimates presented in Table 17 are much more realistic than the estimates listed in Table 15. Real-time calculations are easily obtainable for the CPE module (floating-point format) and the VPH module using a 50 kHz sampling rate. For fixed-point formats, the number of modules required approaches the limit of the number of modules that should be allowed. In one system, efficiency drops noticeably when incorporating approximately eight modules. In this case, the VPH is slower than the CPE because the VPH requires overhead cycles that are noticeable for small calculations.

The system, if it were configured to perform the Kalman filter, would consist of the required number of modules (determined by the sampling rate and the size of the state vector). Instead of pipelining the five equations (for one sample) between the boards, each board would be responsible for its own separate sample. The first sample would be sent to module 1 and calculated. As sample 1 is being calculated, sample 2 would be sent to module 2 and calculated, and so on. Only $x''(k)$ and $P_a(k)$ are required to be passed from the k^{th} to the $k^{th}+1 (\mathrm{mod}\ n)$ modules. An additional board is required to multiplex and to demultiplex the samples and the new estimates, respectively. After a predetermined number of samples have been processed, (8), (9) and (11) are recalculated for the next set of samples. This causes a discontinuity in real-time sampling since time is taken away from calculating (7) and (10) to calculate (8), (9) and (11).

A partitioning scheme for the Kalman filter routine has been presented. For realistic state-vector dimensions, the Cascadable Processing Element and the Vector Processing Element can execute the Kalman filter algorithm in real-time by using a scheme that allows some values to be constant over several samples. This eliminates a large number of calculations for each sample. The price paid for this increase in speed is a decrease in accuracy. For a typical state vector dimension of 6, the Kalman filter would require three CPE modules to support a 50 kHz sampling rate. If the VPH were to implement the algorithm, six modules would be required. If both the CPE and the VPH performed parallel Kalman filter processing, even higher sampling rates could be achieved.

6.2.1 Digital Focusing

The VPH architecture has natural ALU structures to support high speed digital focusing. In the multi-mode automatic tracking system at WSMR the VPH can execute short FFTs in microseconds. Images can be transformed to the frequency domain, edges and critical masspoints can be more easily determined, and focusing accomplished in one frame time. The procedure is to compute the 2D FFT on a zoomed locality, execute a Laplacian derivative (second derivative) to sharpen up edges for contouring, and track the frequency lines of interest. Those which increase or decrease intensity after median filtering denote focus status.

The digital focusing requirements for blurred images at WSMR will assist lens focusing and can be met with this architecture using the vector processing module as follows. As a VME double height board, it can plug directly into existing WSMR backplanes to perform a zoom on a 256x256 frame to a 16x16 frame where the target can be 2D-FFT'd as shown in Figure 24. The 2D-FFT will give a spectral window of the spatial content. Movement of the spectral peaks will identify blur or focus from frame to frame. Prior to the FFT, an edge enhancement step may make the spectral analysis clearer. Recall that close up focus is highly desired anyway. This is very important for multiple munition scenes. The P2 connector on our board can use the VS and HS signals from WSMR boards to sync pixel clock, window region, frame region, and an eventual target classifier signal (which may be simply a binary on/off trigger). Our RS232 output can then be fed directly to WSMR tracker gear. The application for this technology is shown in Figure 24.

The actual functions which the board will perform looks like the sequence shown in Figure 25. Using a 6-bit TRW flash converter, the digitized signal is fed to a frame grabber. Then either edge enhancement is performed (if needed) or the 2D-FFT is executed on the zoom segments. From this point, a digital amplitude detection on the spectral contents is done. Finally, the relative position estimate is made and signals to the WSMR stepper motors will drive the lensing mechanisms to perform fast real time focusing. A rotator can align the image to a fixed axis (so a simple 1D-FFT is used) or we can perform a 2D-FFT on complex data. Our board can perform a 16x16 2D-FFT in 1.8 milliseconds and easily keep up with a 60 per sec Frame rate of 16.6 ms.

6.2.2 Multi-Spectral Data Processing

The CPH and VPH modules can be coupled to integrate six sensor inputs. These modules have sufficient registers and memory to capture a "data pass", colocate, and align the frames. The procedure is to take one sensor as primary, register others along this boundary, compute the RMS error of each datum position, and update the pointers to data base. These computational steps take less than 40 usecs for short frames.

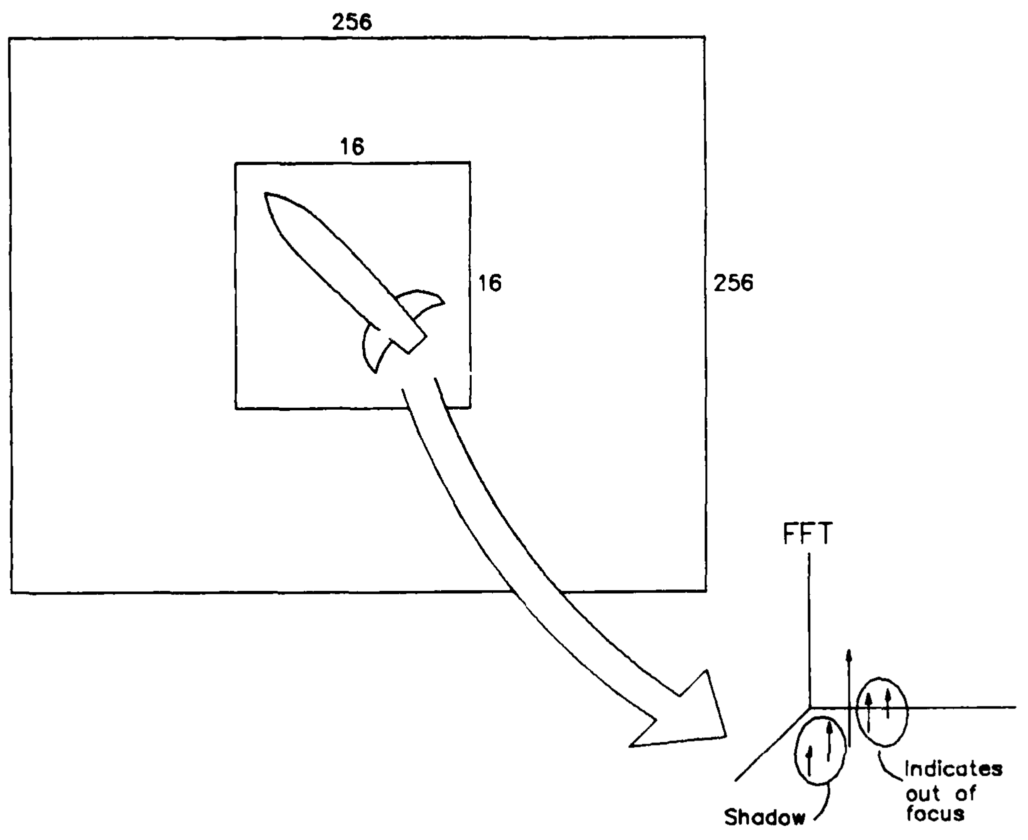


Figure 24. Frame Zoom and Spectral Focus Analysis.

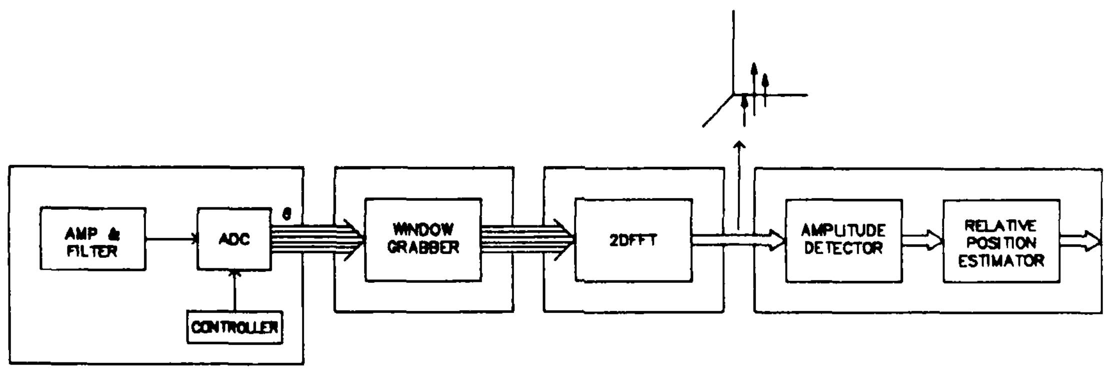


Figure 25. Digital Focuser Steps.

6.2.3 Digital Filtering With Simulation Models

Conventional polynomial process models are fast. However, simulation models are more realistic. The VPH is a microprogrammable architecture which can incorporate novel algorithms without resorting to slow high level code. Direct computation of Kalman, LS, LMS, and splines make this architecture better suited to real-time digital filtering.

6.2.4 Real-time Processor for Large Aperture Array

The VPH can be microprogrammed to compute the angular resolution cells in detected targets by monitoring 9 adjacent cells in the focal plane. The VPH is organized to compute "array-like" data efficiently because sufficient registers and ALU power exists in its internal vector processors that can process up to 128 element vectors in one microinstruction! A nine element vector by vector multiply can be executed in an estimated 80 nanoseconds.

6.3 Best Fit Edge Detection For Missile Imaging and Digital Focusing

The texture, contrast, and "noisiness" of missile tracking imagery data belongs to the class of visual images that requires unconventional and non-classical processing techniques. For instance, the digital Laplacian operator cannot be directly applied to these images without further modification. This is vividly portrayed in the application of classical edge detection techniques to visual images that are very "busy", which tends to amplify the granularity of an image rather than generate useful edge detection. A comparative study of classical edge detection techniques is described with actual atmospheric data obtained from geostationary satellite data.

This study was made to determine optimal strategies for removing clutter from missile data in real-time so tracking can be enhanced. Most of the clutter data analyzed is cloud backgrounds. In principle, if clouds can be edge-detected, then a cut-and-fill operation can eliminate the "artifact". The best edge-detected algorithms are thus important so that digital focusing can be more easily accomplished.

6.3.1 Technical Background

Use of finite difference approximations to linear isotropic derivative operators such as the gradient and Laplacian as edge detectors are well documented [15]. Given the point (i, j), the magnitude of the digital gradient of f is

$$[\Delta_x f(i, j)^2 + \Delta_y f(i, j)^2]^{\frac{1}{2}} \tag{19}$$

where

$$\Delta_x f(i, j) = f(i, j) - f(i - 1, j) \tag{20}$$

and

$$\Delta_y f(i, j) = f(i, j) - f(i, j - 1). \qquad (21)$$

A popular approximation to this expression is the Roberts' gradient approximation [16]:

$$[(g(i, j) - g(i + 1, j + 1))^2 + (g(i + 1, j) - g(i, j + 1))^2]^{\frac{1}{2}} \qquad (22a)$$

where

$$g(i, j) = \sqrt{f(i, j).} \qquad (22b)$$

6.3.2 Classical Edge Detection Techniques

In this study, the following variation to the Roberts' gradient was used (see [15])

$$\max (|f(i, j) - f(i + 1, j + 1)|, |f(i + 1, j) - f(i, j + 1)|). \qquad (23)$$

As noted in [15], the differences in this expression are symmetrical about the point (i + 1/2, j + 1/2) and hence, the Roberts' gradient should be considered an approximation to the continuous gradient at that point rather than (i, j).

The discrete Laplacian of f at the point (i, j) is given by

$$\nabla^2 f(i, j) = f(i + 1, j) + f(i - 1,j) + f(i, j + 1) + f(i,j - 1) -4f(i, j) \qquad (24)$$

Recalling that the second derivative of f at the point x_0 can be approximated by

$$f''(x_0) = \frac{f (x + h) + f(x - h) - 2f(x)}{h^2} \qquad (25)$$

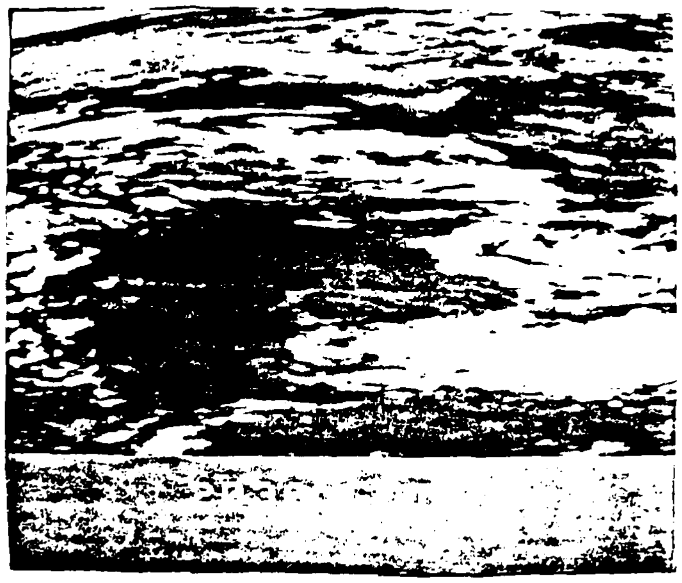
where h is the positive grid parameter, we can interpret the digital Laplacian combination of the finite difference approximations to be "horizontal" and "vertical" derivatives of f at (i, j) with stepsize h=1.

We have applied the Roberts' gradient (23) and the Laplacian (24) to some digital infrared GOES-1 data (Figure 26a). Although the Laplacian and the Roberts' gradient perform nearly equally well on the smaller cloud masses, both fail to yield a contiguous boundary on the larger clouds. As an edge detector, the Laplacian suffers from the defect noted by Rosenfeld and Kak [15] of being more sensitive to the detection of lines, line ends and points over edges. Hence, the somewhat "noisy" result, especially in the larger cloud masses. This "noisiness" has also been noted by Eberlein and Weszka [17]. It should be noted, however, that their definition of the Laplacian magnitude differs from ours and is given by

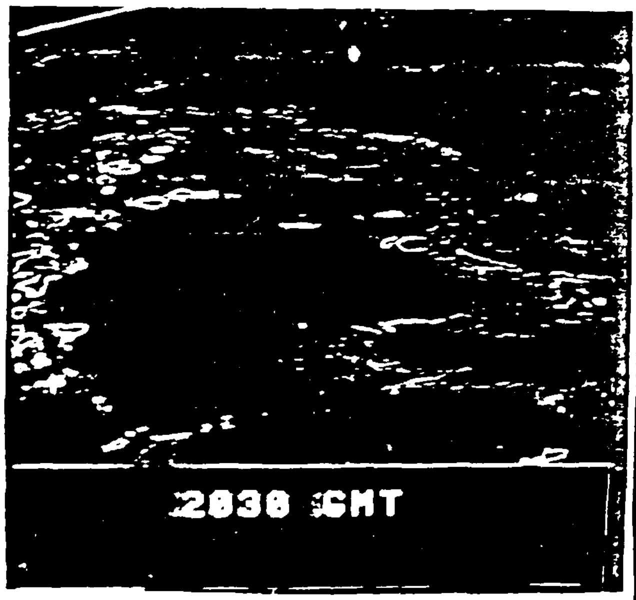
$$\nabla^2 f(i, j) = \Big|\frac{1}{9} [f(i - 1, j - 1) + f(i - 1, j) + f(i - 1, j + 1) + f(i, j - 1) + f(i, j) + f(i, j + 1) + f(i + 1, j - 1) + f(i + 1, j) + f(i + 1, j + 1)] - f(i, j)\Big| . \qquad (26)$$

Using the gradient approximation

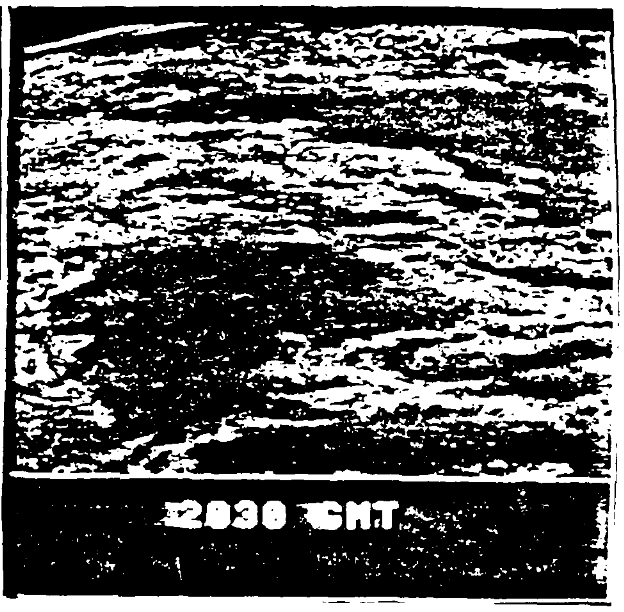
$$\max\left[\,\left|f(i-1,j-1)+f(i-1,j)+f(i-1,j+1)-f(i+1,j-1)-f(i+1,j)-f(i+1,j+1)\right|,\ \left|f(i-1,j-1)+f(i,j-1)+f(i+1,j-1)-f(i-1,j+1)-f(i,j+1)-f(i+1,j+1)\right|\,\right], \qquad (27)$$



(a)



(b)



(c)

Figure 26. Results of Applying the Roberts' Gradient (b) and the Absolute Value of the Laplacian (c) to Some Infrared GOES-1 data (a). Image Visibility was Enhanced by Linearly Spreading Values From 0 to 110 to the Full 0 to 255 Range.

there is a slight, but noticeable improvement in the edge detector output of ERTS-1 data when (26) is subtracted from (27) at each point.

To extend this idea into the nonlinear domain, we formed the product the Roberts' gradient and the Laplacian given by (23) and (24) respectively with the result shown in Figure 27. The results are less than excellent.

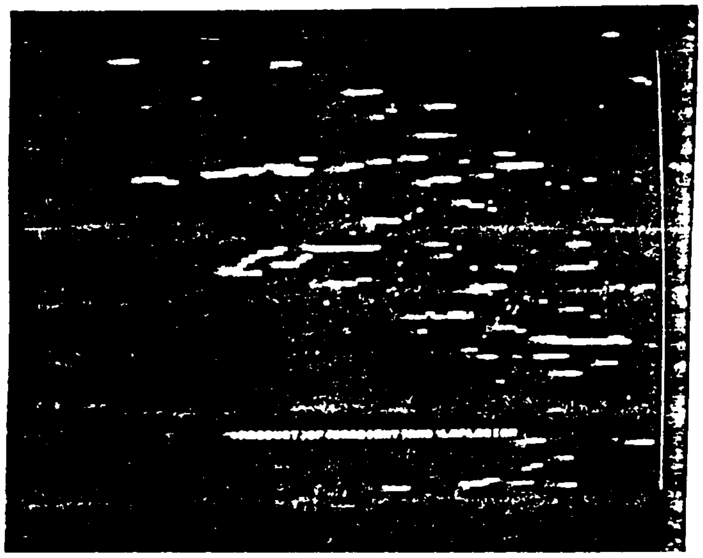


Figure 27. Result of Forming the Product of the Roberts' Gradient and the Laplacian.
Values from 0 to 5 Were Linearly Stretched to 0 to 255.

As noted earlier, the discrete Laplacian is merely the linear combination of the finite difference approximations to the "horizontal" and "vertical" derivatives of f at (i, j). A totally symmetric operator can be created by including also the two "diagonal" derivative approximations to obtain

$$\nabla^2 f(i, j) + [f(i - 1, j - 1) + f(i + 1, j + 1) - 2f(i, j)]/\sqrt{2}$$
$$+ [f(i - 1, j + 1) + f(i + 1, j - 1) - 2f(i, j)]/\sqrt{2} \qquad (28)$$

where the $1/\sqrt{2}$ factor is included to compensate for the fact that the diagonal neighbors are $\sqrt{2}$ times as far from (i, j) as are the horizontal and vertical ones. The results of applying this operator is presented in Figure 28 and, for comparison, an enlarged area is shown in Figure 29 with the Roberts' gradient (23) and symmetric Laplacian (28) (applied to the inverse of the original output). This inversion procedure forces the cloud edge to be displayed within the cloud. Some improvement is evident in the symmetric Laplacian over the Laplacian given in (24). Note that the diagonal right to left diagonal translation of the Roberts' gradient result is not present in the symmetric Laplacian picture.

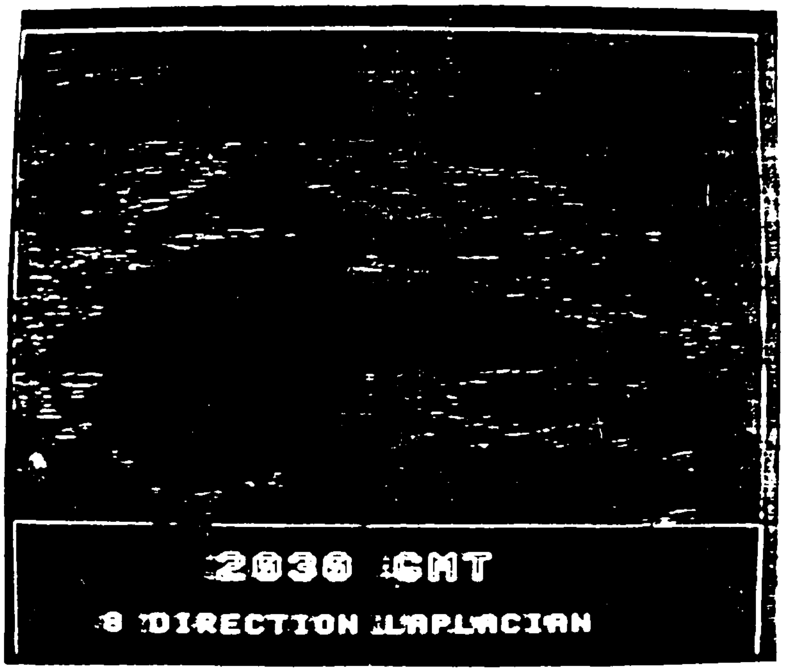


Figure 28. Result of Applying the Symmetric Laplacian.

One of the defects of the above methods is the existence of segmented cloud borders in the operator output. A weighted two-dimensional high pass filter,

$$f(i, j) = f(i, j) - R/4 \tag{29}$$

where

$$R = \frac{\sum_{\ell=1,r} \sum_{m=1,r} W_{\ell,m}\, f(i - (\frac{r+1}{2}) + \ell,\ j - (\frac{r+1}{2}) + m)}{\sum_{\ell=1,r} \sum_{m=1,r} W_{\ell,m}} \tag{30}$$

with

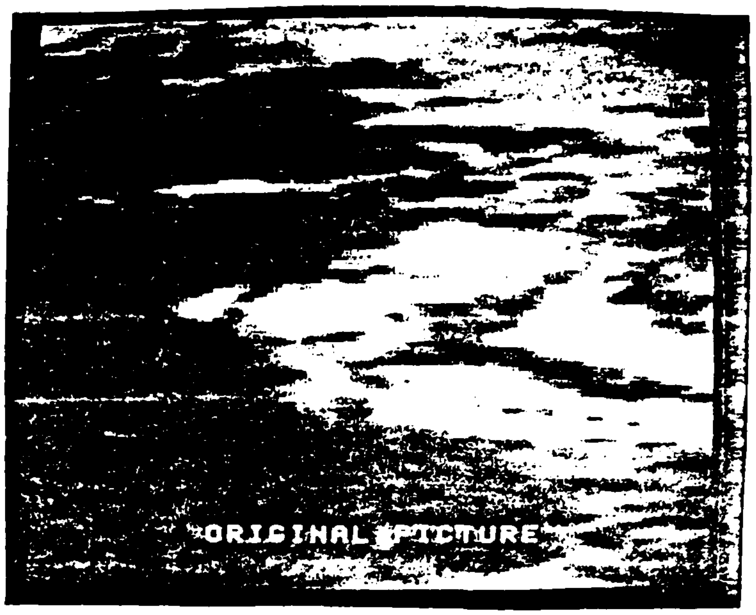
$$W_{i,j} = w_i \times w_j,\ i, j = 1, \ldots, r \tag{31a}$$

where

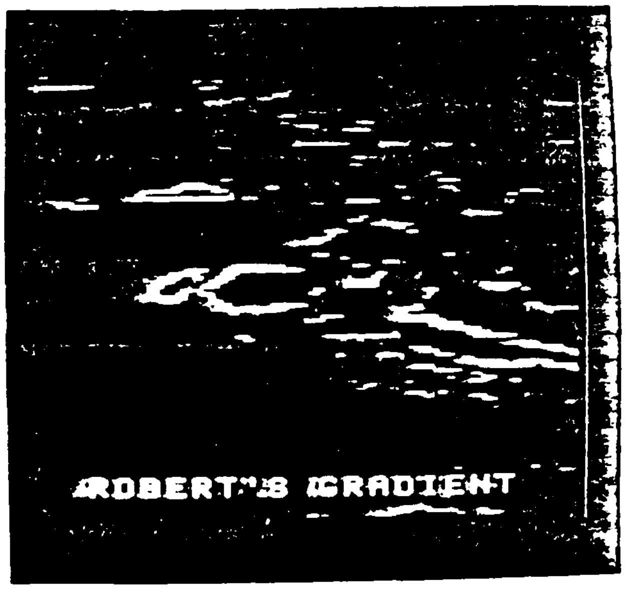
$$w_i = \begin{cases} i & ,i = 1, 7 \\ r + 1 - i & ,i = 8, r \end{cases} \tag{31b}$$

for r = 13, was applied to the satellite image and the Roberts' gradient was then applied with the result shown in Figure 30. As can be seen from this picture, the filtering technique shows promise of closing the gaps prevalent in the former methods.

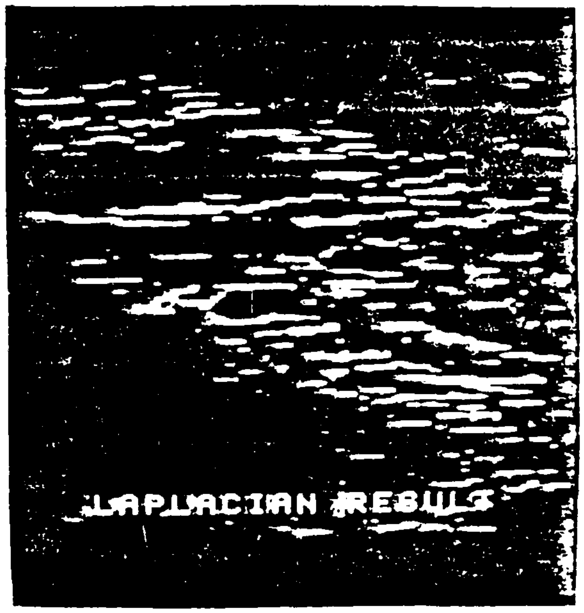
Based on the results of this study, a useful combination of filters and operators would likely be that of a high pass filter and an extended version of the gradient in which more neighbors of (i, j) are included in the operator equation.



(a)



(b)



(c)

Figure 29. Result of Applying the Roberts' Gradient (b) and the Symmetric Laplacian (Applied to the Inverse) (c) to an Enlargement of an Area of the GOES-1 Data of Figure 26.

6.3.3 Sobel Edge Detection

The Sobel algorithm for a 3x3 window computes

$$S = [[[a+2b+c) - [2 + 2f + e]]^2 + [[a+2h+g] - [c+2-d+e]]^2]^{1/2} \qquad (32)$$

on the 3x3 window

$$\begin{matrix} a & b & c \\ h & z & d \\ g & f & e \end{matrix} \qquad (33)$$

The architecture fine-tuned for this algorithm must perform row-wise and column-wise addressing. The ZORAN VSP 161/325 is ideally suited. The instruction sequence for row or column vectors becomes

	INSTR	CLOCK CYCLES
1.	LD	8
2.	ADDR	13
3.	MULTR	15
4.	ADDR	13
		49x.1 = 4.9 us

This is repeated four times along with two squaring (2xMGSQ) plus one square root for an estimated total computation time per pixel of (4.9x4+2x.8+10) = 31.2 us. On a 16x16 ZOOM, real time Sobeling is now feasible with frame rates of 60 per sec.

6.3.4 Optimal Tracker Architecture System Description

The general features of the system (see Figure 31) include: (a) computer control of both internal data routing and user interaction, (b) computer command interface with manual diagnostic test panel, (c) data input multiplexing for data stream merge and/or separation, (d) eight solid state memory planes, eight bits deep for 512x512 pixel storage reconfigurable in any combination from 64 one-bit planes to 8 eight-bit planes and randomly accessible at varying access cycles in read or write mode, (e) memory output multiplexing to route the data stream to any of five look-up tables (LUT's), (f) five video rate and format output ports via the LUT's with high speed digital-analog converters (DAC's), (g) a video tape recording output port with automatic red-green-blue (RGB) to NTSC [18] conversion for recording format, (h) a microprogrammable timing and control unit for variable recording, transmission, playback, video display and direct-memory-access (DMA) by computer, and, (i) a high speed output port for self-diagnostic tests.

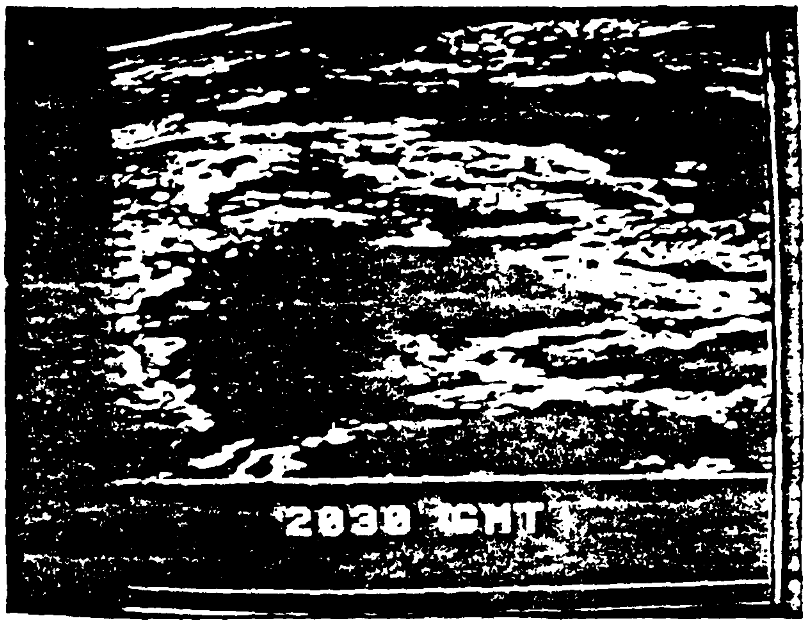


Figure 30. Result of Applying the Roberts' Gradient to an Image Which Has Been Subjected First to a 2-D High Pass Filter.

Due to inadequate reproduction, the above Figures do not accurately display the results obtained in this study. Photographic prints of Figures 26 through 30 are available upon request.

6.3.4.1 Command Processor

The primary control link is the Command Processor (see block 2 of Figure 31) which can perform both in automatic or manual mode. In automatic mode, the computer issues necessary commands via programs generated for various weather map manipulations. At present, 17 primary commands or instructions are available and are listed in the Appendix. Note that each command has several sub-commands or select codes. For example, the Set Output Multiplexers (SOM) command can issue the seven select codes for the output MUXes which route appropriate bits to various LUT's. Remaining primary commands behave similarly and, when combined with their respective subcommands, can generate up to 58 instructions.

These instructions are then decoded by the Command Processor (much like the instruction register of a conventional computer) to actuate several signal and control lines. In the current configuration, commands essentially control input multiplexers, output multiplexers, memory addressing and cursor routing which are primarily hardwired functions. More importantly, all instructions for the command processor unit can also be generated by a manual test panel which conveniently serves as a diagnostic module. Static and dynamic diagnostics can be exercised via this manual panel off-line from the computer if desired.

6.3.4.2 Memory System

The memory is conceptually configured as a memory array much like STARAN is configured as an array processor. Bits or words can be vectorized in both instances; however, unlike STARAN which has a multitude of processing elements for data stream computation. Our approach uses memory planes reconfigurable on demand in either bits or words in any vector pattern. This reconfiguration occurs dynamically (between vertical or horizontal retrace of video) by Command Processor instructions which latch various input and output multiplexers as shown in blocks 3 and 5 of Figure 31. The same principle of "single-instruction-stream multiple-data-stream" for STARAN processors applies to the memory blocks. The 16 bit data words from the computer interface can be streamed out to several planes of the memory simultaneously via the input multiplexers, one of which is shown in Figure 32. Upon exiting memory planes, the data stream (of any width up to 8 bits) can be repacked in any fashion by the output multiplexers.

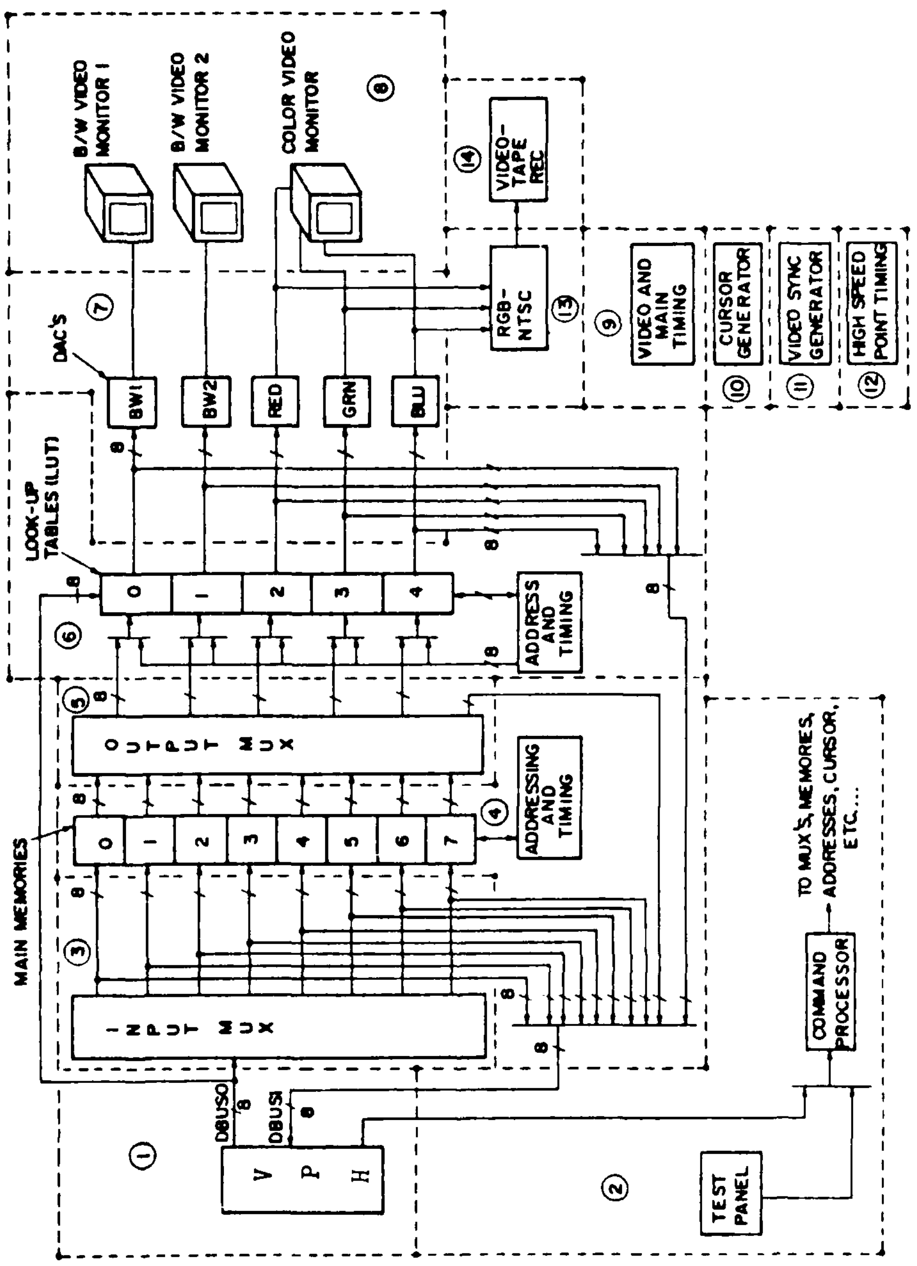


Figure 31. Video Tracker System.

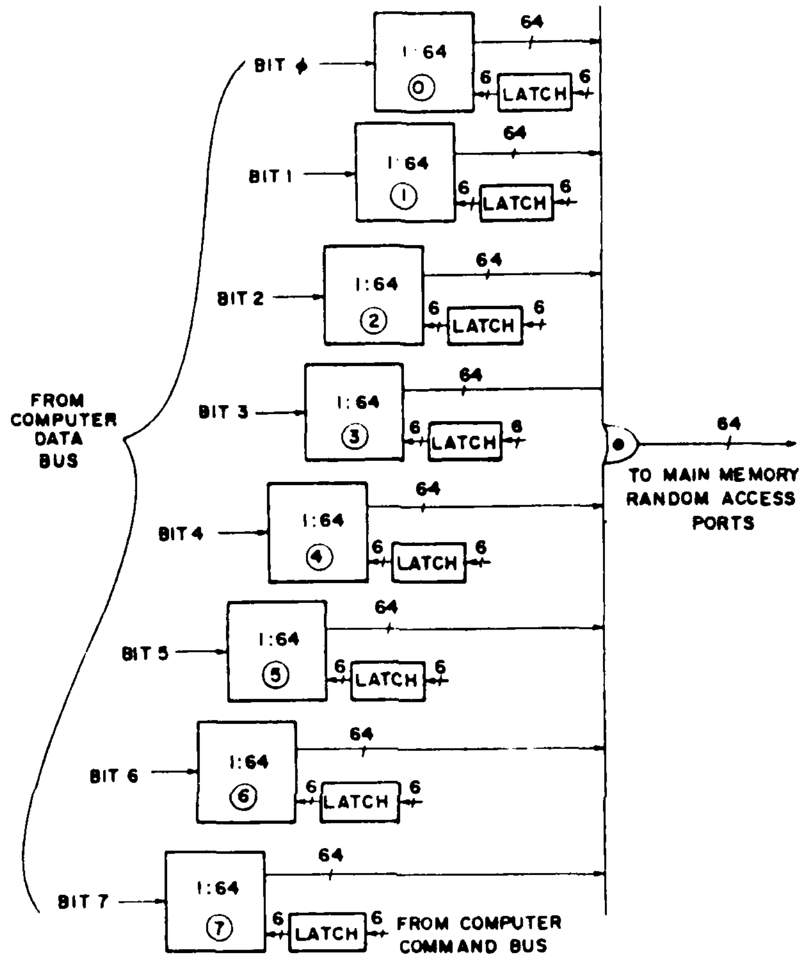


Figure 32. Input Multiplexer.

Two useful operations, which the input/output multiplexers and memory planes rapidly perform, are the mirror and shift operations. Mirror essentially transposes right and left images, while shift operates on individual pixels to scale contrast up or down. The objective of this memory loading scheme is to reduce execution time required for manipulating bits and words in the computer itself and provide for instantaneous picture replication on several output channels, while allowing for future incorporation of arithmetic capability in the separate output channels.

The primary mass storage medium consists of NMOS random access memory [19]. Supporting the memory are the address/multiplexer unit and a timing/control module. Each block (see Figure 33) contains two data input/output ports, a half-duplex random access read/write port (for computer interface) and a sequential or video read port. In sequential access, 16 consecutive one bit words are retrieved, loaded into a 16 bit high-speed shift register and clocked out as a single bit/plane at video data transfer rates, while parallel access is commonly obtained via the half-duplex port. Maximum transfer rates for the video and RAM ports exceed requirements for conventional 30 frames/sec refresh, and DMA, respectively.

One major engineering task involved the design of a flexible addressing scheme, described next, to provide for refresh of the dynamic RAM's regardless of the state of the system without inhibiting video presentation (introducing flicker, lines, streaks). As shown in Figure 33, three multiplexed addresses, MADDR, VADDR, and RADDR, make up the memory address. The random access address (MADDR) is 18 bits wide, externally loaded and automatically incremented after each access. The sequential mode or video address (VADDR) is 14 bits wide (bits 0-3 are zero) and is incremented by (VAINC) and cleared by (VACLR) through external video timing. The upper 9 bits of the address can be viewed as a Y-address and the lower 9 bits as an X-address. On displaying the memory on the video monitor, Y is the vertical direction and X the horizontal. To read through the memory sequentially using MADDR simply requires incrementing the address with Y the most significant and X the least significant bits. In video or sequential access from memory to the monitor, the even Y-addresses are read first and then the odd (interlace). For this reason the Y-address in VADDR is rotated so bit 8 is in bit 0. In incrementing VADDR, bit 8 of the X-address carried into bit 1 of the Y-address and bit 8 of the Y-address carries into bit 0 of the Y-address. Bit 0 of the Y-address is actually a field indicator for interlace, so the field indicator FRMA (from the video sync generator) is used as bit 0. The third address register is the refresh address RADDR. This address has only 6 active bits (0-3 and 10-27 unused) since a plane can be completely refreshed by incrementing address bits 4-9 and by enabling the refresh option.

In contrast to the main memory blocks which, then considered with the input/output multiplexers, dominate bit/pixel management, the five look-up tables (LUT's) provide the real-time transformations on the data stream. Among these functions for the high speed 256x8 RAM's [20] are scaling and histogram generation useful to radiometric conversions, pseudo-color enhancement of multi-band imagery, space variant radiometric correction for removal of vignetting and/or camera shading, and dimensionally reduction or bandwidth compression (when coupled with the arithmetic processor). The LUT's primarily perform in two states, either as mass storage for digital computer access/retrieval or for high speed imagery functions, with control provided via VSRMD (see Figure 34). In the computer state, LUT's are loaded with the desired output functions (e.g., exponential or anti-log similarity, contrast shading, etc.); while in the video state the LUT's serve strictly as table look-up memory.

Bit/pixel shuffling is primarily performed via the input and output multiplexers. Data words from the computer are stored in the memory blocks via eight Input Multiplexer units (see Figure 32) which route one-bit of eight to any of 64 memory planes. Selection is made by a six-bit latch loaded from the Command Processor unit. Data retrieval from the eight memory-blocks is routed via eight Output Multiplexer units (see Figure 35) to the LUT's. Similar to the Input Multiplexers, the Command Processor unit generates a five-bit code to select various paths via the output multiplexers.

Each of the five-line output buses on the multiplexer is wire-ORed bit-by-bit with seven other multiplexer outputs.

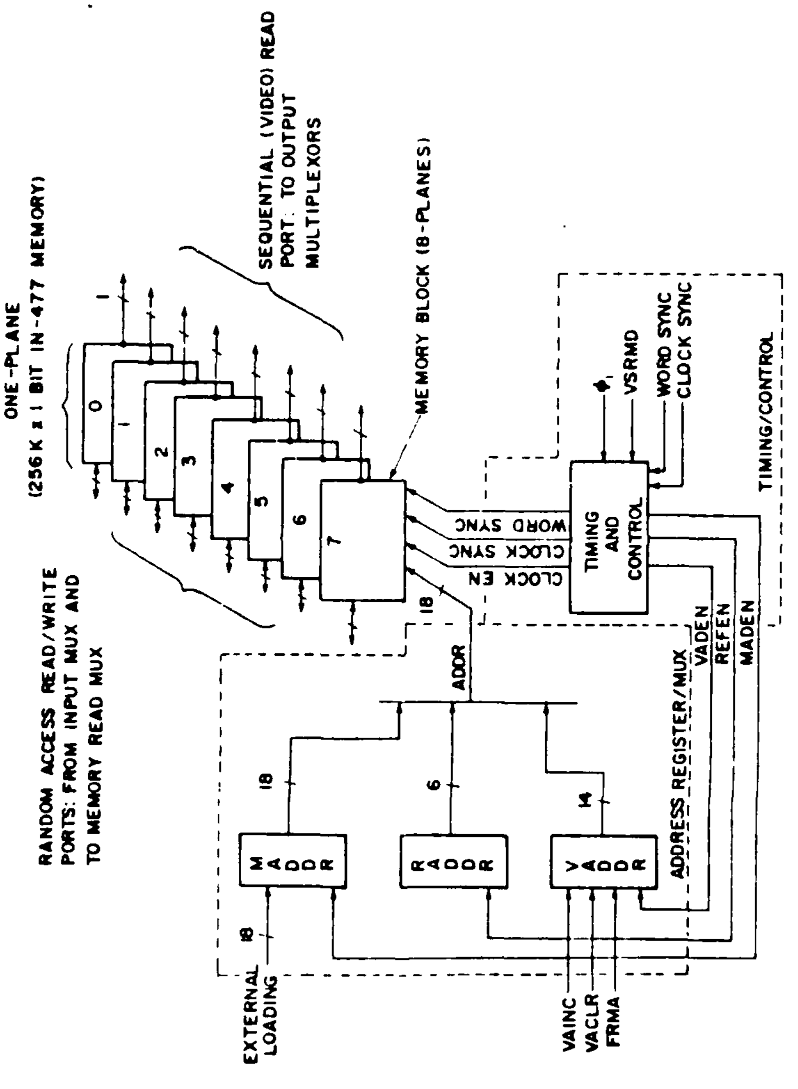


Figure 33. Main Memory System (one plane shown).

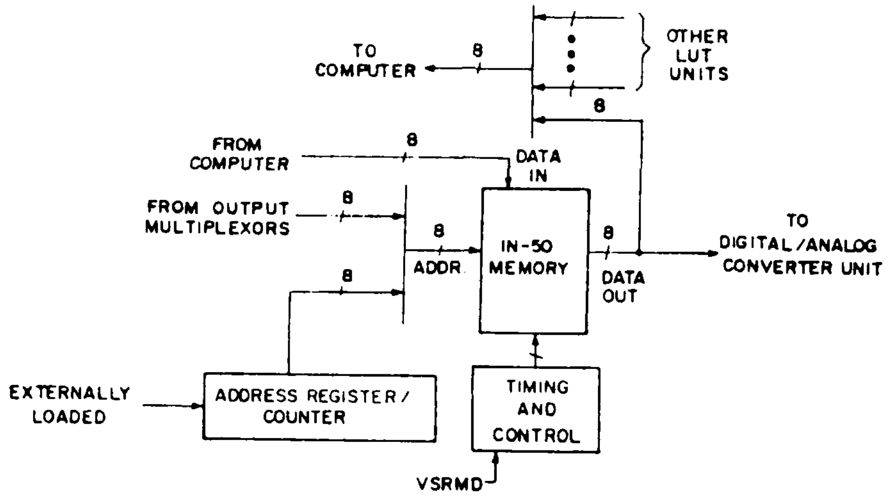


Figure 34. LUT Unit.

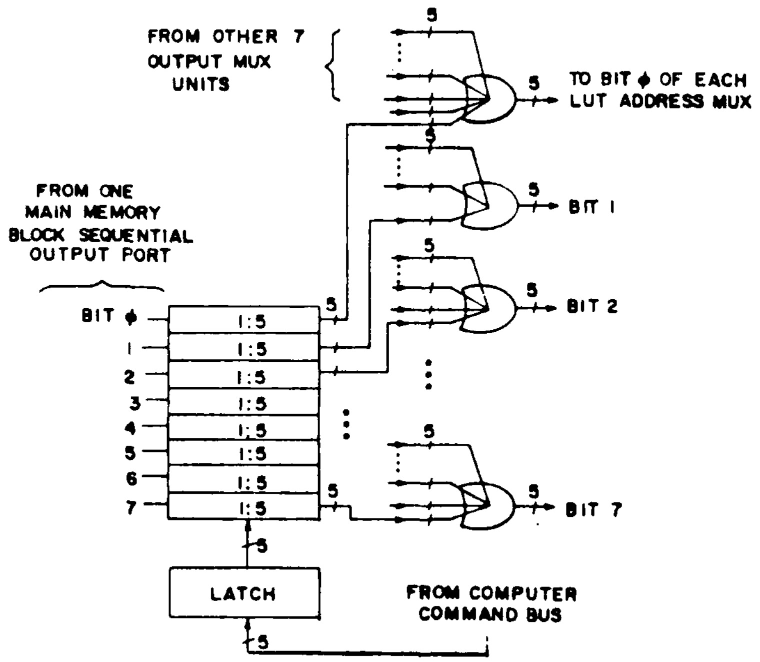

FROM OTHER 7 OUTPUT MUX UNITS
5
5
TO BIT ϕ OF EACH LUT ADDRESS MUX
5
5
BIT 1
FROM ONE MAIN MEMORY BLOCK SEQUENTIAL OUTPUT PORT
5
5
BIT 2
BIT ϕ
1
2
3
4
5
6
7
1:5
5
5
5
BIT 7
5
LATCH
FROM COMPUTER COMMAND BUS
5

Figure 35. Output Multiplexer.

6.3.4.3 Video System

The video system can consist of two raster scan video monitors (black/white and color), video synchronize generator, timing unit and interface circuitry. It is principally designed for high pixel resolution, video bandwidth for 30 frames/sec refresh, and system versatility. Typically, the first two objectives can be achieved by employing high quality monitors, on-line convergence algorithms (to correct for beam spreading in 3-gun color displays), and conventional clocking circuitry. However, system versatility, for example, the ability to change from NTSC (American [18]) to PAL (European [21]) television sync format with minimal hardware modification requires unconventional procedures. In our system, a novel microprogrammable control unit is imbedded in the video sync generator to provide both the power and flexibility of the video system. Furthermore, this microprogrammable feature permits more stable timing circuitry when driven with a phase-lock loop either externally (60 Hz) or internally (crystal oscillator).

The primary role of the video sync generator (see Figure 36) is to provide necessary synchronization for horizontal and vertical retrace; however, most systems also rely on the sync generator to clock data into the video monitors. For stable images (jitter-, flicker-free), synchronization to either the horizontal retrace frequency, f_H, or the vertical retrace frequency, f_V is sufficient. In our case, the data clock was synchronized at a constant phase difference with an integer multiple of f_H, as shown in Figure 37. To do so, we specify some f_{sck} as a multiple of twice f_H and combine f_V, f_H and f_{sck} with a phase-lock loop on either of two base frequency sources. f_{ref} (60 Hz power line frequency) and f_{co} (a crystal oscillator). f_V, f_H and f_{sck} then provide the primary drive signals to the pulse forming circuits of drive (YVD, YHD), blank (YVB, YHB), equalization (YEQV, YEQH), and serration (YSRV, YSRH) where (YXXV) and (YXXH) correspond to vertical and horizontal synchronization, respectively. The relationship of the vertical drive signals with f_V and f_H is shown in Figure 38. For NTSC format, equalize and serrate require three horizontal sync (YHSYNC) periods and, for our monitors, the blanking intervals for vertical and horizontal blank are 1 msec (t_{VB}) and 10 us (t_{HVS}), respectively.

Since, for color displays, the composite sync (TCSYNC) predominantly governs framing operations, drive, serrate and equalize must be coupled with the blanking intervals. However, although fields A and B (assuming interlaced scanning) modify the composite sync to account for half lines at picture top and bottom, the vertical sync signals can be obtained by dividing twice the horizontal frequency by an odd number and, with a 512 complete visible line format. f_H can be derived from the vertical blanking interval, t_{VB}, such that

$$f_H = \frac{513}{2} * \frac{60}{(1-60t_{VB})} \tag{34}$$

where 513 corresponds to complete visible lines plus two half lines and synchronization to the power line frequency of 60 Hz is assumed. If, then, f_{sck} is chosen carefully, horizontal timing for NTSC as well as for 512x512 displays can be achieved which naturally reduces circuitry in the video timing unit. A functional diagram of the video sync generator, shown in Figure 36, depicts the coupling of the phase-lock module (block 3), the programmable memory modules for NTSC or 512x512 (blocks 4-9), and the interface circuits (blocks 1, 10) to external clock control and monitor drive, respectively. Besides the concern for programmability, the video sync generator must inhibit instabilities; hence, the low pass filter in the phase-lock loop must be carefully chosen, Therefore, since a phase-lock open loop transfer function, H(s), with just an integrator is intolerable, an additional pole and zero were incorporated in the transfer function.

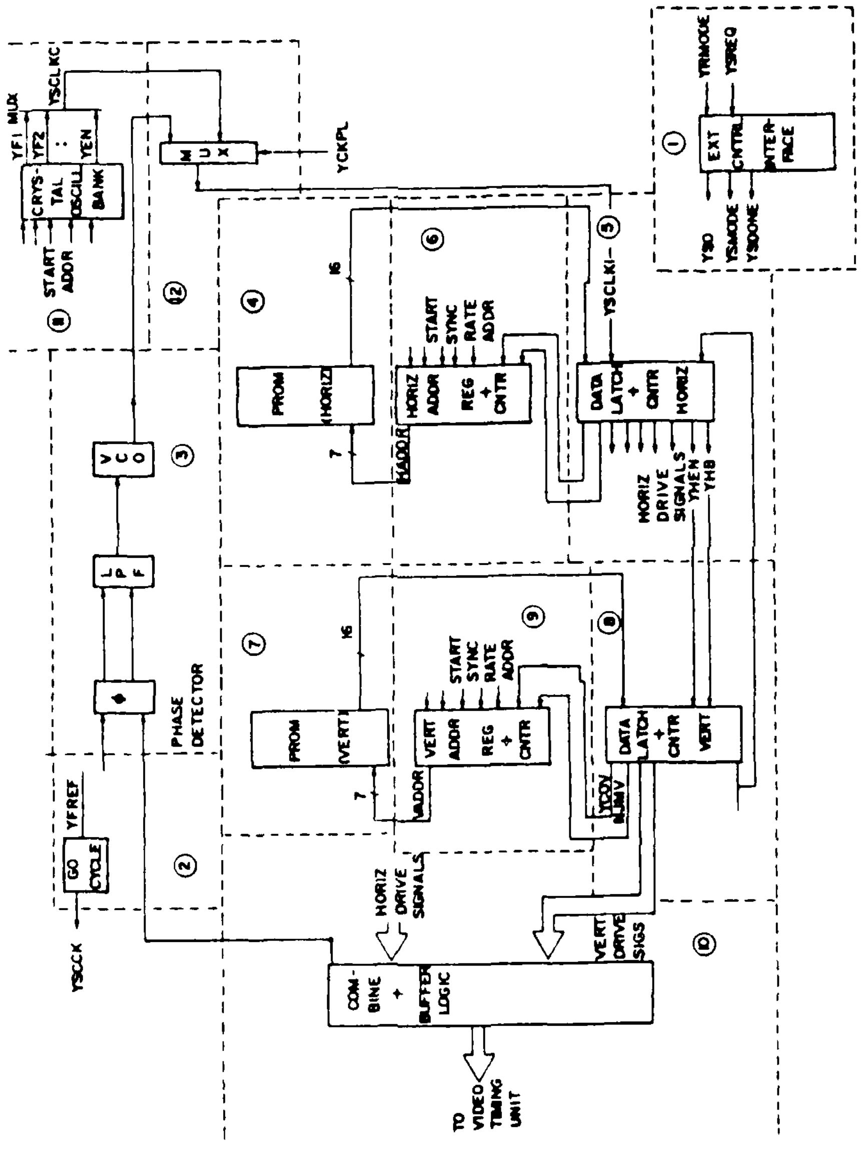


Figure 36. Video Sync Generator.

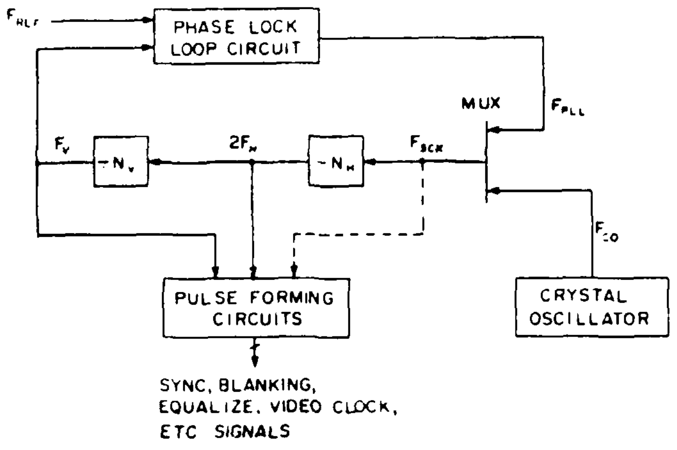


Figure 37. Phase-Lock Synchronizer Loop.

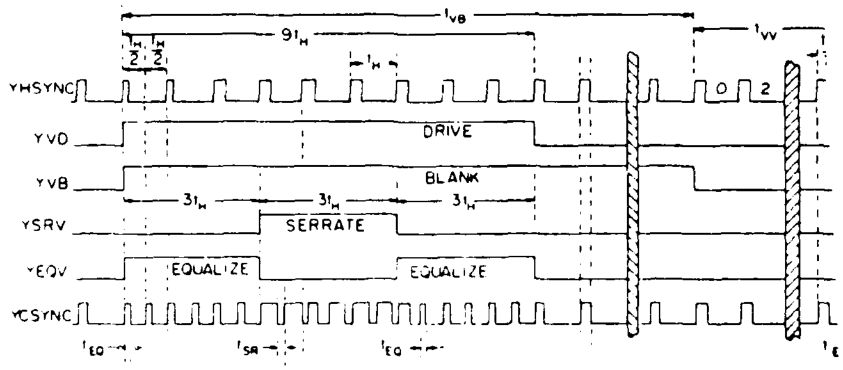


Figure 38. Typical Vertical Sync/Drive Relationship.

The heart of the system versatility resides in the microprogrammable memory units of the video sync generator. In our architecture, two PROM's each for horizontal and vertical synchronization are provided in blocks 4 and 7 of Figure 36. Each PROM coupled with hardwired logic sequences (in blocks 6 and 7) constitute the microprogrammable control unit with microinstruction formats, i.e., the control fields designations, for horizontal and vertical sync control shown in Tables 18 and 19. Note that each microinstruction must specify, not only the pulse duration (bits 0-9, or counter values), but also the repetition rate of each signal (bit 15 or the jump bit which, in most cases, causes the current microinstruction or sequence of microinstructions to be repeated). Generally, the jump bit is enabled on the last microinstruction to repeat the pulsing sequence. The operation initiates with a start map address for any sequence outputted from block 6 as the sync rate selection. Then, blocks 5 and 8 latch microinstructions from the PROM's which contain control signals for serrate, drive, equalization, and blank. For horizontal synchronization, bit 10 controls color burst which is not needed for the vertical signals and remains unused. The current microprogram terminates with a test on the jump bit. Our microprograms are designed by Algorithmic State Machine techniques [22]. Here, the state qualifier pair with assumed address is employed with bit 15 of each microinstruction utilized as "branch enable". Such techniques have proved useful, not only during design but also later during diagnosis and field maintenance. The inherent versatility of a microprogrammable control unit thus supports not only field maintenance but simple modifications for many video formats (NTSC, PAL, etc.).

Table 18 Horizontal PROM Word Format

Bit(s)	Control field
15	0 = Enable jump 1 = NOP
14†	1 = Enable serration gate 0 = NOP
13†	1 = Enable horizontal drive 0 = NOP
12	1 = Enable equalization gate 0 = NOP
11	1 = Enable horizontal blank 0 = NOP
10	1 Enable color burst gate 0 = NOP
9–0	Counter value (2's complement 0–1024)

†Note: If Bit 14 is a 0 and Bit 13 is a 1, Horizontal Sync is enabled.

Table 19 Vertical PROM Word Format

15	0 Jump enable 1 - NOP
14	1 = Equalization gate enable 0 = NOP
13	1 = Serration gate enabled 0 = NOP
12	1 - Enable vertical drive 0 = NOP
11	1 = Enable vertical blank 0 = NOP
10	Not used
9–0	Counter value (2's complement 0–1024)

6.3.4.4 Cursor Generator

The cursor generator unit primarily controls timing, color, and position of a real-time cursor with the capability to respond to computer or operator commands (joystick) for adjustable size and color format. Both flexibility and high speed prompted use of hardwired configuration coupled closely to the final digital output. Therefore, besides interpreting inherent cursive parameters requested of the command processor unit (notably among these are: (a) cursor on/off, (b) region blank, (c) region invert, and (d) cross), the cursor control section itself also generates various commands to the digital-analog converters (e.g., route LUT data to DAC, route cursor to DAC, generate block level, etc.). In the configuration, cursor patterns assume either box or cross shapes in any pseudo-color or transparent mode. To further enhance minimal circuitry, a cursor parameter set shown in Figures 39 and 40 was adopted. The physical cursor dimensions in Figure 39 depict the position (X, Y relative to upper left frame corner), size (DX, DY), and interior/exterior partition zone (DW). These physical specifications in conjunction with the zone denominators of Figure 40 (e.g., zone B identifies cursor corners, zone D identifies the interior, etc.) enhance the command processor instruction set while reducing cursor circuitry. Design of the cursor generator unit subsequently led to similar class 4 ASM link-state machines for X and Y control.

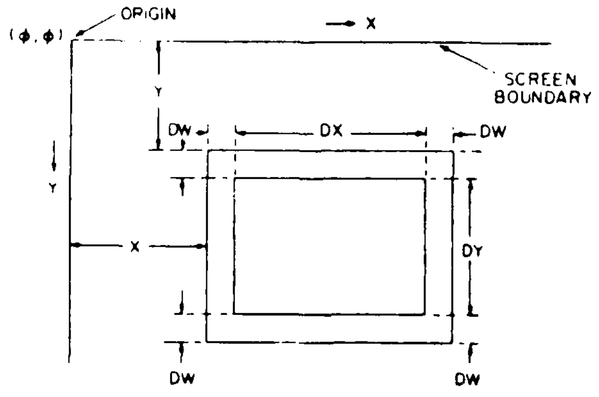


Figure 39. Physical Cursor Dimensions.

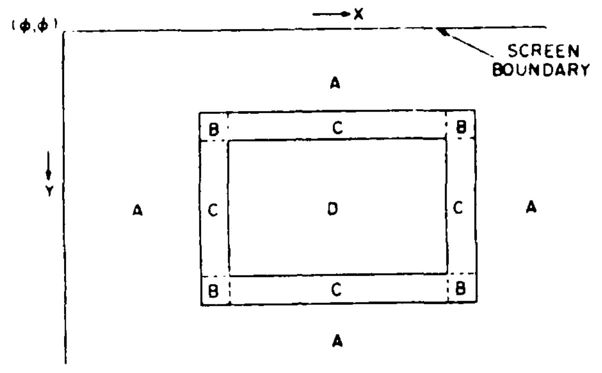


Figure 40. Cursor Zone Denominators.

6.3.4.5 Algorithm Complexity

The algorithmic complexity of a video tracker is directly related to the arithmetic structure (i.e., the number of ALU's, physical location, and interconnections). If, as is typically found, the computational power is resident to a computer essentially off-line from the processing system, then the algorithmic complexity is proportional to the computer itself. However, image processing systems now appear with the computational structure imbedded directly in the data stream in parallel fashion made possible by the relative economy of microprocessors. Therefore, the algorithmic task is no longer relegated solely to the computer but rather to the architecture of the entire image processor system. In small configurations, for example [23], multiple microprocessors are configured as independent ALU's. At each output channel from the main memory banks are located small numerical processors capable of executing +, -, *, and /. This architecture supports pixel-pixel computations and is adequate for many algorithms. In large systems, such as STARAN, multiple ALU outputs can be programmed to several associative memories which permit simultaneous pixel (parallel) and bit (serial) manipulation. In our system, a combination of the techniques extracted from [23 and STARAN are included to provide both pixel and bit manipulation while attempting to maintain a modestly sized configuration. At present, we rely solely upon ZORAN devices. In a final configuration, the system may have GAPP devices in the output section with processor and data stream selection controlled by the off-line computer. The future architecture is chosen to effect both local and global spatial form [15].

7.0 Conclusions and Recommendations

The original intent of this project was to identify fast and flexible arithmetic engines suitable for real-time radar, signal- and image-processing applications. To that extent, preliminary design specifications for an architecture known as the Expandable Vector Accelerator (EVA) have been developed. This is an architecture capable of executing numerous real-time algorithms via two separate but tightly coupled components, a high speed DSP machine and a general purpose data processor. These blocks are discussed below.

The Vector Processing Hardware (VPH) is a speed-optimized architecture capable of processing vectors of complex data. As such, the building blocks of the VPH were selected so that algorithm and data partitioning could be implemented with relative ease. Although this is not always feasible, the architecture is arranged so that a high degree of parallelism and concurrency can be achieved. This feature, along with a duplication of on-chip resources at the board level, are the primary reasons for the high throughputs achievable with the VPH. The overall concept has been theoretically confirmed by employing the FFT algorithm as a test-bed. The results of the study revealed that a quad VSP-325 configuration can yield a 1024 complex FFT result in 460 us. This is a significant accomplishment for a board-level product.

The Cascadable Processing Hardware (CPH) is also speed optimized. However, the architecture is configured for those applications where the concern for high precision and wide dynamic-range is at a premium. By means of sophisticated control circuitry, the CPH can be reconfigured dynamically to expand or reduce the width of the data words. Hence, if an application involves multiplication/addition-intensive tasks, the CPH can expand its wordlength to accommodate those cases where overflow/underflow occur, thus eliminating round-off errors. The converse also applies. For instance, if an application only requires processing on 8-bit data, the CPH can be configured to operate as an 8- or a 16-bit machine, thus optimizing the architecture for smaller wordlengths and increasing overall throughput.

In order to fully capitalize on the processing power available on an EVA-like architecture, the system-bus configuration must be equally capable of interfacing with the host, and within modules of the architecture. A brief literature survey has been conducted to identify the most optimal bus arrangement. The study did not consider 16-bit bus configurations such as the STD bus, MULTIBUS I, UNIBUS, and Qbus. These systems do not satisfy current DSP and/or military real-time demands, nor are they capable of supporting the dynamic-range demanded in such applications.

Of the bus-configurations considered, the VME and the MULTIBUS II systems are better capable of handling the requirements of an EVA machine. An EVA-like architecture places a great deal of demands on the system interface configuration. Overall speed and versatility are paramount issues. Based on the findings presented herein, the VME system configuration appears to satisfy all of the demands required by the high-performance, cascadable-processing engine.

Aside from the FFT study mentioned above, other theoretical-analyses for mapping algorithms onto the EVA architecture were performed. A partitioning scheme for the Kalman filter routine was also presented. For realistic state-vector dimensions, the CPH and the VPH can execute the Kalman filter algorithm in real-time by using a scheme that allows some values to be constant over several samples. This eliminates a large number of calculations for each sample. The price paid for this increase in speed is a decrease in accuracy. For a typical state vector dimension of 6, the Kalman filter would require three CPH modules to support a 50 kHz sampling-rate. If the VPH were to implement the algorithm, six modules would be required. If both the CPH and the VPH performed parallel Kalman filter processing, even higher sampling rates could be achieved.

Several techniques for edge-detection and enhancement, clutter removal, and target tracking have been studied as well. This study was made to determine optimal strategies for removing clutter from missile data (in real-time) so that tracking can be enhanced. The basic principle of our approach is based on the commonly known fact that if background can be edge-detected, then a cut-and-fill operation can eliminate this "noise." The best edge-detection algorithms are thus important so that digital focusing can be more readily accomplished.

Finally, the preliminary microinstruction-word layouts for all of the components of the EVA architecture were determined. Moreover, it has been determined that a distributed control approach offers numerous advantages, particularly when the architecture requires complex control of highly sophisticated components. Propagation delays are minimized, and narrower buses can be utilized. The minimized microinstruction-word for the CPH is totally parallel, requiring no on-board decoding. The same is true for the VPH microcode format. Both of these preliminary microword layouts offer a great deal of versatility and task concurrency for each module of the EVA architecture.

REFERENCES

[1] Fischer, W., and P. Roper, "Versatile Bus Suits Real-time Processor Applications," Computer Design, June 15, 1984.

[2] Schreck, G., "A Designer's guide to the VMEbus," Electronic Components News, June 1985, pp. 13-17.

[3] VMEbus Specification Manual, Revision C.1, Micrology pbt.,Inc.,eds., Printex Publishing, Inc., October, 1985.

[4] Shlomo, P.T., and J.A. Black Jr., "VMEsystem: The Most Popular 32-bit Architecture," VMEsystem Architecture and Technology, Motorola Microsystems.

[5] Balph, T., and D. Artusi, "VME: A System Architecture for Industrial Control," Motorola System Design News, Vol. 1, No. 1, 1985, pp. 10-15.

[6] Cavanagh, J.F. Digital Computer Arithmetic Design and Implementation, McGraw-Hill, New York, 1984.

[7] Andrews, M., and R. Fitch, "Finite Word Length Arithmetic Computational Error Effects on the LMS Adaptive Weights,", IEEE International Conference on Acoustics, Speech, and Signal Processing, April 1977, Hartford, CT.

[8] Ferro, F., and K. Ulery, "25-MFLOPS DSP looks for new worlds to conquer," Electronic Products, September 15, 1987.

[9] Vector Signal Processor (ZR34325) Product Description, ZORAN Corporation, 1987.

[10] Advanced Micro Devices, Bipolar Microprocessor Logic Interface Data Book, Sunnyvale, California 94088, 1985.

[11] Bipolar Integrated Technologies, Floating-Point Chip B2110/B2120 Preliminary Data Sheet, Beaverton, Oregon 97006, 1987.

[12] Bipolar Integrated Technologies, Product Summary of High Performance VLSI Solutions for Digital Signal Processing and High-End Data Processing, Beaverton, Oregon 97006, 1987.

[13] Azimi, M., Colorado State University Class Notes, Fort Collins, Colorado 80524, 1987.

[14] Maron, M.J., Numerical Analysis: A practical approach, Macmillan Publishing Co., New York, 1982.

[15] Rosenfeld, A., and A.C. Kak, Digital Picture Processing, Academic Press, 1976.

[16] Tippett, J.T. et. al., Optical and Electro-Optical Information Processing, M.I.T. Press, pp. 159-197, 1965.

[17] Eberlein, R.B., and J.S. Weszka, "Mixtures of Derivative Operators as Edge Detectors," Computer Graphics and Image Processing, Vol. 4, pp. 180-183, 1975.

[18] Federal Communication Commission, Television Synchronizing, Public Notice 53-1663, 1953.

[19] Intel Corporation, IN-477 Technical Manual, Memory System Division, Sunnyvale, CA, 1976.

[20] Intel Corporation, IN-50 Technical Manual, Memory System Division, Sunnyvale, CA, 1976.

[21] Townsend, B., <u>PAL Colour Television</u>, Cambridge University Press, New York, 1970.

[22] Clare, C.R., <u>Designing Logic Systems Using State Machines</u>, McGraw-Hill, New York, 1973.

[23] Adams, J., and R. Wallis, New Concepts in Display Tecnnology, Computer, Vol. 10, No. 8, 1977.

Appendix. Command Processor Instruction Set.

Command	Select code	Data	Description
0	X	X	Clear all main memory enable bits
1	0–7	Bits 0–5	*Set output multiplexors* SC = MUX number Data: Bit 0 routes 8-bits of data to LUT 0 Bit 1 to LUT 1 ⋮ Bit 5 to high speed port

2	0–7	0–63	*Set input multiplexors*
			SC = MUX number
			Data = Main memory Plane to route Computer bit to

3	0–7	Bits 0–7	*Set main memory enable latches* SC = Main memory block Data: Bit 0 a 1 enables Plane 0 Bit 1 a 1 enables Plane 1 ⋮ Bit 7 a 1 enables Plane 7
4	0–4	0–255	*Set look-up-table read* SC = LUT # Data = Address to be read

5	0–4	0–255	*Set look-up-table write*
			SC = LUT #
			Data = Address to write

6	X	0–7	*Enable main memory block to be read* Data = Block #
7	X	0–511	*Load main memory + high speed port X-address* Data = 9-bit X-address
$^{10}8$	X	0–511	*Load main memory + high speed port Y-address* Data = 9-bit Y-address
$^{11}8$	0–4	0–511	*Load cursor position/size parameters* SC = 0 load X 1 load Y 2 load DX 3 load DY 4 load DW Data = Value 0–511
$^{12}8$	5	Bits 0–3	*Set cursor modes* Data: Bit 0 a 1 is cursor on Bit 1 a 1 is invert on Bit 2 a 1 is blank on Bit 3 a 1 is cross on

Appendix. (Con't).

Command	Select code	Data	Description
10_8	6	Bits 0–4	*Set cursor color* Data. Bit 0 a 1. Red on Bit 1 a 1. Green on Bit 2 a 1. Blue on Bit 3 a 1. B/W 1 on (white) Bit 4 a 1. B/W 2 on
11_8	7	X	*Enable high speed read mode*
12_8	X	Bits 0–1	*Start/stop video mode + sync rate set* Data: Bit 0 a 1. set video mode a 0. clear video mode Bit 1 a 1. set 525 sync a 0. set 545 sync
13_8	X	X	Set *main memory* write mode
14_8	X	X	Set main memory read mode
15_8	X	X	Enable return of command channel flag
16_8	X	X	Disable return of CFLAG
17_8	X	X	Clear all read/write modes (except video mode)