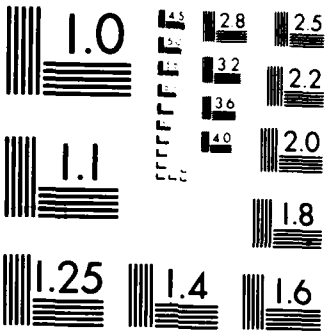
AD-A191 581 NOSC (NAVAL OCEAN SYSTEMS CENTER) ADVANCED SYSTOLIC 1/1
ARRAY PROCESSOR (ASPP)(U) NAVAL OCEAN SYSTEMS CENTER
SAN DIEGO CA J P LOUGHLIN DEC 87

UNCLASSIFIED F/G 12/6 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

UNCLASSIFIED
SE

DTIC FILE COPY

T DOCUMENTATION PAGE

1a. **AD-A191 581**

1b. RESTRICTIVE MARKINGS

2.

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	6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION
Naval Ocean Systems Center	NOSC	Naval Ocean Systems Center

6c. ADDRESS (City, State and ZIP Code)	7b. ADDRESS (City, State and ZIP Code)
San Diego, California 92152-5000	San Diego, California 92152-5000

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
Office of Naval Technology	ONT	

8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	AGENCY ACCESSION NO.
Arlington, VA 22217	62721N	EEB2	RS21243	DN088 690

11. TITLE (include Security Classification)

NOSC Advanced Systolic Array Processor (ASAP)

12. PERSONAL AUTHOR(S)

J.P. Loughlin

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Professional paper/speech	FROM Aug 1987 TO Aug 1987	December 1987	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	microsequencer algorithms systolic architecture

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The design of a high-speed (250 million 32-bit floating point operations per second) two dimensional systolic array composed of 16 bit/slice microsequencer structured processors will be presented. System design features such as broadcast data flow, tag bit movement, and integrated diagnostic test registers will be described. The software development tools needed to map complex matrix-based signal processing algorithms onto the systolic processor system will be described.

Presented at SPIE International Technical Symposium, 17-21 Aug 1987, San Diego, CA.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (include Area Code)	22c. OFFICE SYMBOL
J.P. Loughlin	619-553-2531	Code 741

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

To appear in the Proceedings of the SPIE International Technical Symposium, Vol. 827-13, Real Time Signal Processing X, San Diego, CA 18-19 August 1987

NOSC Advanced Systolic Array Processor (ASAP)

Joseph P. Loughlin

Signal Processing Branch, Naval Ocean Systems Center
271 Catalina Blvd., San Diego, California 92152

ABSTRACT

The design of a high-speed (250 million 32-bit floating point operations per second) two dimensional systolic array composed of 16 bit/slice microsequencer structured processors will be presented. System design features such as broadcast data flow, tag bit movement, and integrated diagnostic test registers will be described. The software development tools needed to map complex matrix-based signal processing algorithms onto the systolic processor system will be described.

1. INTRODUCTION

The Navy is focusing research at the Naval Ocean Systems Center (NOSC) in the area of the processing of signals collected from large arrays of sensors. Enhanced performance through the use of powerful matrix based signal processing algorithms operating on large volumes of data promise efficient extraction of signal information under conditions of noise and corruptive media effects. Several important matrix based algorithms used to process signal data require large numbers of computations. A computational requirement which increases at a rate corresponding to the square, cube or to the fourth power of the sensor array element composition number are required for such matrix processing. To maintain a realistic processing throughput in Navy system applications, a geometrically increasing demand is being placed on the computational throughput of the signal processing hardware. To address these demands, NOSC is investigating the application of systolic architectures hosting matrix-based signal processing algorithms.

2. ARCHITECTURAL CONSIDERATIONS

A careful study of the classical computer architectures previously used for signal processing shows either serious bottlenecks in data movement or inefficiencies in computational power due to the general nature of the design. Figure 1, shows the basic elements of a Von Neumann computer architecture. The throughput available in this type of machine is directly related to the instruction execution rate. As the performance of the semiconductor components used to implement such a machine levels off, an upper limit in processing throughput is reached.

As shown in Figure 2, the growth in performance level of the integrated circuit has been sufficient during the past two decades to promise enhanced system performance based solely on the new devices as they become available. The technology which promised such growth in the past years is becoming mature and presently the growth is less pronounced. At the same time, the system requirements for orders of magnitude increases in computational throughput force the pursuit of different design avenues which provide much more room for performance enhancements.

An enhancement on the Von Neumann architecture that attempts to remove the dependence of processing throughput on device performance is the parallel processor, Figure 3. This architecture uses the fact that most algorithms may be disassembled into component parts which may be performed by multiple processors sharing a common memory. The concept of incrementally adding additional processors to the signal processing task to multiply the apparent throughput of the system is theoretically sound. However, incremental enhancement in throughput with each additional processor rapidly reaches a point of diminishing returns. Figure 4 shows this point to occur with about 10 processors in the system. This throughput ceiling results from two mechanisms fundamental to a parallel signal processor of this type. The first limitation is present due to the overhead incurred in scheduling parts of the signal processing task to available processors in the system. This task assignment activity must be performed by a supervisory processor or be attached to each individual processor as an overhead to each processor in the system. Beyond 10 processors, this arbitration and scheduling becomes very time consuming. Another, and possibly more serious problem jeopardizing the promised throughput of this architecture is the system bus bandwidth limit. As more processors are attached to the system bus, the transfer bandwidth of the bus becomes the system bottleneck threatening to idle processors.

Proposing the use of parallel processors to multiply the throughput of a signal processing system is the basis for the systolic array architecture introduced by H. T. Kung in

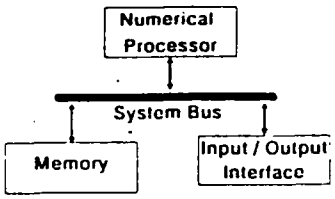


Figure 1. Von Neumann Architecture

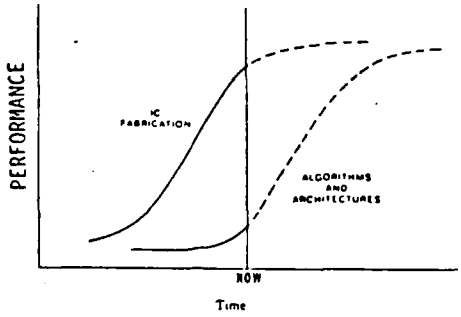


Figure 2. Processing Growth Potential

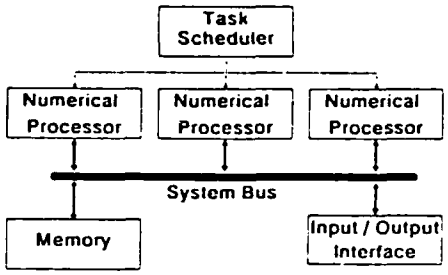


Figure 3. Parallel Processor Architecture

1978.[1] However, this architecture avoids the eventual degradation of incremental throughput enhancement as each additional processor is introduced into the system. Being conceived expressly to perform only signal processing (primarily matrix-based) problems the systolic array architecture addresses the previous parallel processor performance drawbacks described above in an elegant way. The extremely regular structure of the operations needed to perform many signal processing tasks identified a parallel processor structure. However, all of these processors perform identical parts of an algorithm while responding to only locally available data. Furthermore, with each new and identical processor added to the architecture, additional buses are also included. The issue of scheduling multiple processors is now equated to arranging appropriate coincidence of input data to each processing element. The processor to bus ratio is maintained regardless of the size of the array, thus avoiding the data bus bandwidth limitation described in earlier parallel processor architectures.

3. ALGORITHMS

All signal processing generally falls into two categories which are identified by their type of execution. The first category, typified by the Fast Fourier Transform, contains those algorithms which consist of a collection of arithmetic operations and well defined data movements which are performed repeatedly. The execution of an algorithm which is in this category is not affected by the actual numerical values being processed or any intermediate values incurred. This type of algorithm when mapped onto a systolic array results in a single instruction/multiple data (SIMD) architecture or regional groupings of SIMD routines with well defined timing relationships. The use of architectures with extremely long pipelines such as bit serial processors host such algorithms quite nicely. The second category, typified by Single Value Decomposition matrix operation, exhibits variability in the actual course of execution of the algorithm depending on the values of the data circulating within the processor array.

The mapping of this type of algorithm is best accomplished by loosely coupled autonomous processing elements executing in a multiple instruction/multiple data, (MIMD), systolic architecture. The overall signal processing task often requires a combination of one or more algorithm components from both categories just described. In addition to the programming flexibility needed at each systolic element, information required to guide algorithm execution may not be available at this low level. The global communication of data to a centralized algorithm control may be necessary to reconfigure groups of elements in the array to perform the next portion of the signal processing task.

4. TESTBED ARCHITECTURAL SPECIFICATIONS

The systolic array testbed system, Figure 5, incorporates architectural features which allow it to host a large variety of signal processing algorithms. Our experience with our previous testbed * and mapping approaches identified for similar architectures indicated a need for a system architecture which contained a superset of the basic systolic characteristics. The regular and orderly movement of data through an array of processing elements executing a rigid cycle of primitive operations appears to be too limiting and awkward for the execution of single value decomposition or Gram-Schmidt orthogonalizers. Previous

* NOSC 8 X 8 processor array hosting Gentleman-Kung least squares matrix solution.

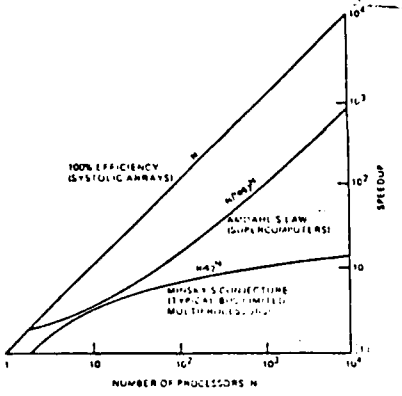


Figure 4. Multiprocessor Efficiencies

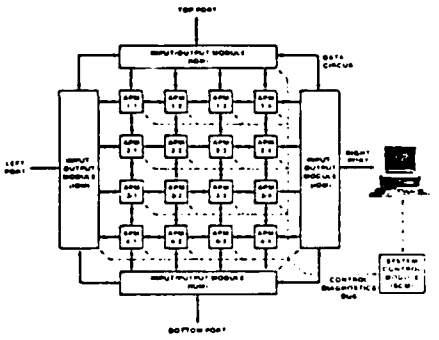


Figure 5. Systolic Testbed Architecture

mapping of the SVD[2] and parallel models constructed to study the execution of a GS orthogonizer * indicate that additional array capabilities not normally associated with a classical systolic array were required to guarantee maximal use of the processing power contained within. The complete list of architectural features of this hybrid system array are as follows:

4.1. Systolic features

a. Two dimensional array of 16 identical processors which support nearest neighbor communication of data.
b. Four identical boundary processors which handle data communication with external data ports.
c. A system controller responsible for control of algorithm execution.

4.2. Enhanced features

d. Data Communication -- Broadcast of data across an entire row, column, diagonal, or array global.
e. Autonomous program execution at each processor with tight or loose coupling to neighboring processors
f. Tag Communication -- Instruction or status information tagged to data movement throughout the array.

In addition to the above array architectural features, enhancements to the structure of the individual processing elements have been included to extend the parallel nature of the processing within the array hardware. These features are expressly needed to avoid computational bottlenecks within the processing element and ease programming.

4.3. Additional element features

g. Four independent input/output ports.
h. Simultaneous multiplication and add or subtract on independent pairs of operands.
i. Capability of concurrent I/O and numeric computation.
j. Autonomous program execution with extensive branch capability resident.
k. Operand Address generation.

The mapping of an algorithm, especially one such as an SVD or Gram-Schmidt orthogonalizer requires the target software to support sophisticated and interactive development tools. To this end, elemental features included expressly to allow the algorithmist or programmer to verify proper operation of the algorithm are identified below:

4.4. Additional diagnostic features

l. Built-in diagnostic set-scan capabilities for component fault isolation and power up confidence testing.
m. Down-loadable instruction microcode.
n. Program trace capability.
o. Parity checking for instruction word integrity during run-time.

*VAX simulation of the Dobson adaptive beamformer/nullsteerer algorithm.

p. Host access to all memory contents supported.
q. Mode utilities to handle computational errors, illegal program execution and breakpoints.

The testbed system has been constructed to allow the host computer to be used not only as an experiment driver, but to also support the following developmental tasks:

a. Provide a vehicle for the generation of microcode which is downloaded to the target hardware processing elements.
b. Provide a vehicle for the generation of input matrix data through the use of appropriate simulation programs.
c. Host output data formatting and display utility programs necessary to access performance of algorithms hosted on the target hardware.
d. Handle the development utilities of initiating confidence testing, downloading executable microcode, interfacing simulated input/output data during algorithm execution on the target testbed hardware.
e. Act as an intelligent algorithm controller during algorithm execution.
f. Support the algorithm diagnostic utilities available with the architecture of the target hardware.

5. HARDWARE IMPLEMENTATION DETAILS

The systolic array testbed system, refer to Figure 5, is composed of 16 Arithmetic Processing Modules (APM), 4 Input/Output Modules (IOM), a System Control Module (SCM) and the system host control computer (an IBM-AT for this initial configuration). The 16 APM's are systolically connected via orthogonal 40-bit bidirectional parallel data buses to adjacent APM's or to an IOM on each of the boundaries of the 4 X 4 square array. Data communication to hardware external to the array occur via the 4 external ports (top, bottom, right, and left). An additional 40-bit bus, called the data circus, is included in the processor architecture allowing the data movement around the periphery of the systolic array structure. Communication between the elements of the systolic array processing elements and the system control module is handled by a global bus called the Array Control Bus (ACB). The host computer communicates control, diagnostics and data to the APM's and IOM's via the SCM.

6. ARITHMETIC PROCESSOR MODULE FUNCTIONAL FEATURES

Each APM is composed of 330 off-the-shelf integrated circuits and has been constructed on a 16" X 18" wire wrap circuit panel. Figure 6 identifies the major functional components of each APM and hints to the highly parallel architecture contained within each module. The computational power of the APM resides in the ALU which is composed of a pair of 8 MHz Weitek floating point processor chips, the 1033 multiplier and the 1032 ALU. The module architecture allows both of these chips to perform simultaneous computations on separate sets of operands while communication of neighboring processing modules may also occur. This degree of parallelism is achieved by the use of 4 Weitek register file chips (1066), which is a five ported 32 location 32-bit wide scratch pad memory. Because a total of 128 (4 X 32) locations for operand storage was deemed inadequate to support most signal processing algorithms, an additional 4K locations of single ported memory is included in this scratch pad function.

Combined with this data memory is a unique data address generator. The memory space can be segmented and used to support complex data manipulation. These memory segments (up to 256 locations) can also be used as circulating buffers when modulo address arithmetic is enabled. Additional memory and comparator circuitry is included to allow the designation of up to 256 memory pointers as 12-bit counters which can be used as control of program execution.

The I/O structure of the APM has been made highly parallel and reconfigurable to allow the greatest latitude for algorithm data movement. Each register file chip can be dedicated to data movement associated with each adjacent module. This allows the simultaneous movement of up to four different data packets during a single transfer interval. The data transfer occurs at the same rate as the internal computational rate, namely, 125 nanoseconds per transfer. The data flow network is capable of supporting a number of topological configurations. A characteristic of many signal processing algorithms is the need for some sort of global or broadcast data movement. Each APM can support broadcast in several different configurations. By moving data through the data flow network in a transparent mode, row, column and diagonal broadcasting is supported during a single clock cycle.

The on-board control of all the functional elements of the APM described so far originates from a micro-sequencer and instruction RAM. To accommodate the highly parallel

nature of the APM, the instruction word contained in the RAM is 176 bits wide. The micro-sequencer accepts pointing vectors to the starting address of desired program segments via the control bus. The program flow can be modified by testing the I/O handshaking, the contents of the data tag byte, auxiliary mode registers, or data related arithmetic operations.

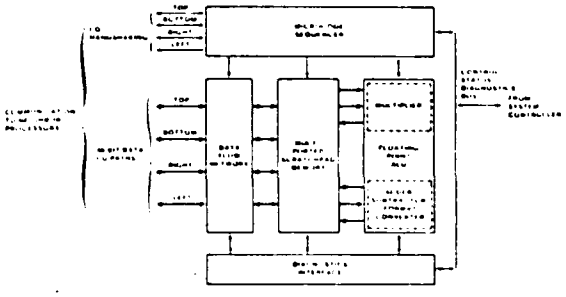


Figure 6. APM Functional Organization

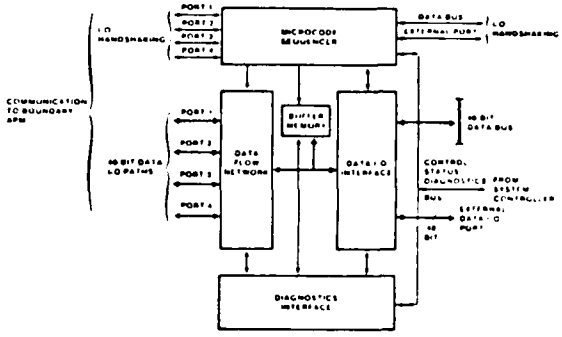


Figure 7. IOM Functional Organization

The APM uses a Berkley architecture where executable program memory is separate from data storage memory. The only communication occurring between program control and the contents of the data memory is under the condition defined as loading a program imbedded immediate constant into data memory. The contents of the data memory can only affect program flow indirectly as a result of tests for zero and positive resultants from the ALU. The floating point data type used by the ALU is not compatible with the address format of the micro-sequencer. This enhances the overall speed of the processor and limits the ultimate addressing flexibility of the processor.

The final functional block in the APM is the diagnostics interface which allows the system host computer to load or interrogate APM memory registers and internal buses. This interface is used for such functions as downloading instruction and data and has the additional feature of supporting initial debugging efforts, operation confidence testing, and fault isolation.

A trace buffer is included in the diagnostics interface for program development. This cyclic 4K buffer captures the most recent 12-bit instruction pointers generated by the micro-sequencer. An additional 4 test points deemed important for analysis are captured with each instruction pointer. Using special diagnostic routines, this information can be uploaded to the host computer for viewing.

7. INPUT/OUTPUT MODULE FUNCTIONAL FEATURES

Each IOM is composed of 190 integrated circuits and has been constructed on a 9" X 16" wire wrap circuit panel. Figure 7 identifies the major functional components of each of 4 IOM's in the system. To minimize the complexity of programming and the hardware debug cycle, the microcode sequencer and diagnostics interface are identical in function to those used in the APM. The IOM contains no data computational circuitry but is expressly designed to efficiently move data. The data flow network connects the data present at the boundary of the systolic array to the internal 4K buffer memory. The IOM handshaking and transfer rates are identical to the APM's. Each of the boundary IOM's has 2 nonsystolic ports included which serve important interface functions in the application of the array hardware. The external port comes complete with a separate set of handshaking signals which allow the intelligent communication of data with external hardware without interfering with the systolic movement of data within the array itself. The data bus (data circus) allows the IOM processors to act as a distributed interface system. The registration of data input and output to the array hardware with the external system hardware can be programmed into the IOM program.

8. SYSTEM CONTROL MODULE FUNCTIONAL FEATURES

The SCM is composed of 140 integrated circuits and is similar to construction to the IOM's. Figure 8 identifies its functional components and its relationship to the other system components. The main function of the SCM is to convert an extension of the host computer bus (16-bits) to a format used in the ACB (64-bits). The host computer can address each of the diagnostic and control registers contained in single processors or groups of APM's and/or IOM's. System status including global busy, ready, arithmetic

error detection, or system instruction parity memory fault can be monitored by the host computer via the SCM. The incremental algorithm commands (the selection of the desired microcode program modules) can be directed to modules of the array. The system clock originates on the SCM board and a separate copy of the clock is sent to each system module. This clock is programmable in speed, and can be incrementally controlled and used during hardware debugging and algorithm mapping. The SCM incorporates the circuitry needed to allow data movement between the host computer and any one of the 4 external ports. This feature is included to aid in the initial mapping of the algorithm in the absence of the balance of the external system hardware.

9. HARDWARE SYSTEM CONSTRUCTION

The system hardware, with the exception of the host computer, is housed in a 24" wide, 30" deep and 56" high equipment rack. A custom cardcage complete with fans was constructed which allows the mounting of the 16 APM cards in the front side of the backplane circuit card and the IOM's and SCM in the back. Due to the number of wide parallel buses and high clock speed of the array, all the systolic connectivity is contained in one 19" X 24" 10-layer circuit card. A special power distribution grid constructed from copper bar stock was attached to the backplane to accommodate the current load of the present system configuration (6500 IC's) and future enhancements up to 400 amps.

A secondary multibus cardcage has been included in the equipment cabinet to accommodate data acquisition system components and possible future use of a single board computer to replace the present IBM-AT host.

10. SOFTWARE DEVELOPMENT

The mapping of signal processing algorithms which incorporate complex manipulations of matrix data on an array of computational arithmetic elements requires a well structured suite of development tools. These tools allow the mathematician or programmer to verify the proper operation of all phases of an algorithm and compare systolically produced intermediate variables with corresponding values derived elsewhere.

To verify the successful integration of all the design features contained in the testbed hardware required the use of customized software running on a system-dedicated personal computer. The I/O resources of the host computer and the C programming language, were used to construct a fundamental set of software utilities called hardware primitives. These primitives allowed the design engineer to exercise the diagnostics/control bus of the testbed hardware and gain access to the functional parts of each of the array elements. Such rudimentary operations as forcing a single immediate microcode instruction to be placed into an element, permit the engineer to verify the proper wiring and correct logical design of small groups of integrated circuits. These lowest level primitives and others form the basic interface library used by the host personal computer to support all communication with the testbed hardware.

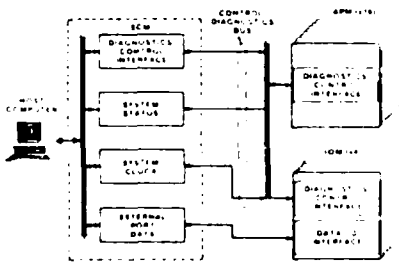


Figure 8. SCM Functional Organization

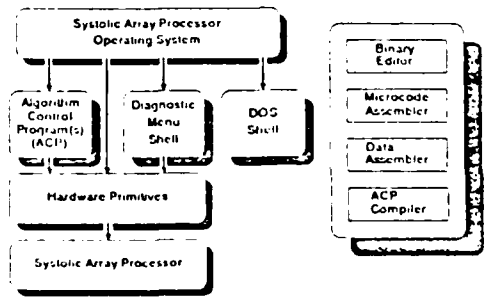


Figure 9. SAP Operating System

11. DEVELOPMENT TOOLS UMBRELLA PROGRAM

A systolic array processor (SAP) operating system, Figure 9, resides on the IBM-AT which provide a cohesive interface among the various components of the software development tools, the systolic array hardware and the file directories of the DOS operating system. Both diagnostic activities and algorithm execution is supported by this software environment. This custom operating system provides the user direct control of the hardware through the hardware primitives.

The large number of development tools necessary to support the mapping of complex algorithms onto a high performance parallel processor array of this type can be a source of confusion. Four user modes are built into the SAP operating system, direct command line access to all functions, a menu and cursor style function access, a DOS shell, and algorithm control program (ACP) shell.

In the menu shell, Figure 10, all of the hardware primitives are divided into groupings according to the processor board type. A further distinction according to primitive functional category provides an easy reference approach locating the appropriate primitive. Each menu entry contains online help documentation if necessary. This system of menu pages also includes a page dedicated to canned hardware confidence tests and in total allows organized reference and usage by inexperienced programmers. The menu interface uses either a mouse or cursor keys to select a primitive.

Numerous primitives require additional information in the form of command line arguments to execute. In those cases, the primitive calling name with references of expected argument syntax prompts the user for the required information.

The command line shell provides direct access to hardware primitives and other diagnostic utilities. It is reserved for the more experienced programmer and does not give any syntax prompts. This short cut also minimizes the cycle time while conducting interative hardware debugging tests.

Several utilities available in the SAP operating environment interface DOS files with the testbed hardware, such a memory write and buffer memory read operations. A user path to DOS is therefore included in the umbrella program. This provides a convenient access to directory listings.

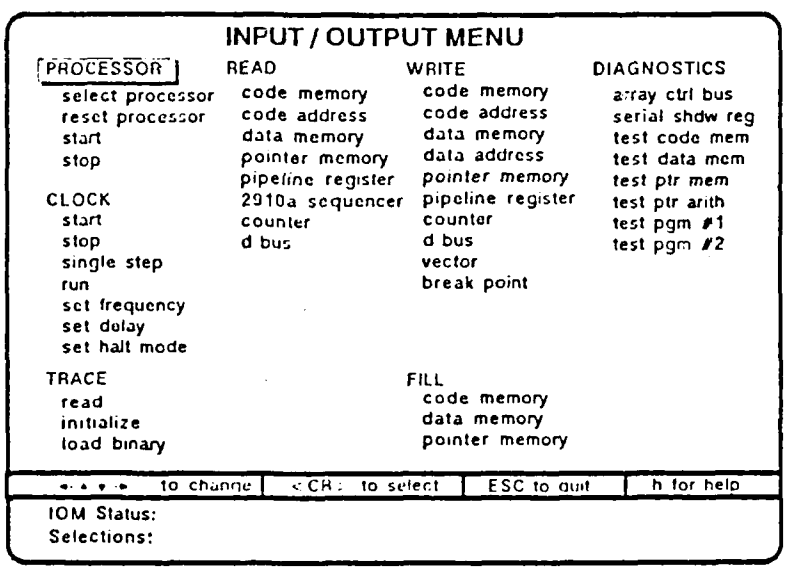


Figure 10. SAP Menu Screen

12. BUILDING TOOLS

There exist a number of routines which do not access the testbed hardware. Due to their size, these software utilities are stand alone and are accessed outside the control of the SAP operating environment.

12.1. Microcode assembler

The binary patterns needed in each microcode primitive are constructed by the hardware engineer on a microcode template worksheet. This information combined with comments is used to produce a microcode primitive library element. This library of microcode primitives is the fundamental interface with the programming execution on each individual element. While the primitives need only be constructed once, the error free assembly of groups of microcode primitives into larger microcode modules required a custom assembler to be constructed to aid the programmer. The actual microcode executable image to be downloaded to the processing element must adhere to some very specific format guidelines.

a. All bits not actively used in a microword to perform the intended function must assume a default value.
b. Operations involving pipelining must span multiple microwords with appropriate delays of affected microword bit fields.
c. Even parity must be established across each microword using a bit field transparent to the programmer.
d. A file header containing information about the microcode data contained within must be constructed to provide proper loading of the executable image.

The assembler has been constructed using the framework of the C programming language. The microcode primitives are configured as an extension of the C library. The microcode programmer is free to use the power of constructs contained within the C programming language with an enhanced library to produce large amounts of executable code with access to such features as automatic microcode address assignment, iterative assembly and parameter/variable passing.

The generation of executable microcode modules which can be downloaded to the target testbed hardware entails writing the source code of a C program using calls to the microcode primitives. The C source is compiled and linked in the normal manner to produce a program. When executed, this program produces three files, one of which is the executable binary memory image to be downloaded to the testbed. The remaining two files are ASCII files containing user comments extracted from the source program and pipeline comments contained in the microcode primitive library. These comments (identified with their location in the code memory image) appear in the files in the order which the microcode was generated. These files aid the programmer in two distinct ways. Initially, the microcode can be checked to verify that the microcode was assembled in the intended order and appropriate parameters and variables have been passed. Secondly, the contents of either one of these files can be used in conjunction with trace utility (described below) to provide the programmer with a disassembled display of the actual microcode execution on the target processor.

12.2. Binary editor

The microcode binary image contained in an executable file can be viewed directly by the use of a binary editor routine. Either the 64 or 176 bits with a corresponding instruction location in code memory is displayed on the CRT. By the use of a mouse or cursor keys, any bit or bits may be changed, microwords added or deleted. This tool is powerful and is used only during hardware debugging.

12.3. Data editor

The systolic array testbed was designed to support 32-bit data computations throughout. An additional 8-bit tag associated with each data word has been added. A small utility routine resides on the utilities menu which allows the construction of arbitrary length data file. In its initial version, this routine accepts decimal keyboard entry from the user. Later versions will accommodate the input from simulation program data output files.

Two software utilities have been constructed which provide access to selected variables associated with the execution of systolic algorithms. The trace utility is designed primarily to document the actual execution of the microcode program. The window utility, while presenting information about program flow, additionally allows the tracking of selected data memory locations of multiple processors. Both utilities are very important for testing the operation of signal processing algorithms hosted on the systolic array hardware.

12.4. Trace utility

A record of the actual execution of microcode primitives or modules on individual processing elements is an important tool for hardware confidence tests and for the verification of proper algorithm execution. Included in the APM's and IOM's is a trace buffer memory which captures the last 4096 instruction address pointers generated by the on-board microsequencer. In addition to the address pointers (12 bits) a flag field of four bits can be wired to any arbitrary test points of interest in the circuit. Through the use of hardware primitives, a trace utility program accesses this buffer and displays its contents on the host personal computer CRT. With this utility, the programmer with the aid of a listing of the comment file, can follow the execution of the program.

Another mode of the trace utility cause the comments in either the user comment file or pipeline file to be organized and displayed next to corresponding code memory address pointer resulting in a disassembled display of the executed program. This utility also incorporates displays in hexidecomal or decimal and supports address occurrence search.

The user comments and pipeline comment serve two different purposes. Displaying the user comments associates with each microcode instruction a comment that the user is free to assign during the microcode assembly phase. This comment allows the programmer to recall the purpose of the instruction or to identify exact positioning in the algorithm. The pipeline comments are hardwired into the microcode primitives upon their creation and serve as a debug tool at the lowest level of the hardware. The creator of a microcode primitive is allowed and encouraged to generate a comment identified with each field programmed in the microword. The intended usage of such flexibility is to allow verification of proper microcode construction at the output of the microcode assembler and the successful tracing of pipelined fields through complicated program execution on the testbed.

12.5. Windows

To properly monitor the execution of software residing on 20 microcoded floating point processing elements, a diagnostic utility called "windows" was designed. This utility allows the user to define specific locations in data memory on selected processor elements and receive updated reports about their contents under a variety of algorithm execution modes. The relative dependence of program flow and data manipulation can be monitored at several processor sites simultaneously. Other pertinent information about the status of array processing elements may also be monitored to confirm proper data communication throughout the systolic system.

13. ALGORITHM CONTROL PROGRAM

The testbed hardware can support the execution of a large range of signal processing algorithms. The complexity of the software which controls the execution of algorithm depends on several system architectural and algorithm functional requirements. The philosophy underlying the software development effort on this project is to afford maximum flexibility in the programming approach. An algorithm which is rigidly defined and is invarient in its execution could be entirely captured in a large microcode module which is simply called once by the host computer. The algorithms, such as the Dobson adaptive beamformer, footnote 2, require a combination of global communication, program branching, and host computer intervention. These characteristics necessitate a more intensive interaction between the microcode residing on each processor, its neighboring processor, and the host. To accommodate this interaction, an algorithm is considered here as a collection of callable microcode modules which reside in the processing elements and are an extension of the algorithm control language C library. The algorithmist or programmer approaches the mapping of an algorithm by composing a C source program which will execute on the host personal computer. In addition to the standard library, the programmer will have access to specialized library calls which perform the following functions:

a. Call for the execution of a selected module in one or more processing elements of the testbed hardware.
b. Open input matrix data files and make the contents available to the external data ports of the hardware.
c. Create output matrix data files and receive processed data from the array hardware.
d. Support algorithm breakpoints which suspend execution of the algorithm while the operator interrogates the state of the hardware.
e. Support repeated automatic interruption of the algorithm execution for the purpose of performing system "snapshots" of selected variables and accumulating their values in special diagnostic files.
f. Call utilities which display the status of the testbed hardware on the host's CRT. Algorithm control program source comments, variable values and other system information can be displayed at appropriate times in the algorithm execution.

Figure 11 depicts the interaction required between the IBM-AT host computer executing the algorithm control language program and the systolic array processor executing the requested microcode modules. This figure shows the controlling role of the personal computer host which commands the orderly execution of the "algorithm microcode pieces" in a sequential manner.

14. CONCLUSIONS

The four year development of the systolic array processor, diagnostic software and algorithm mapping tools is nearing completion. Preliminary work has commenced toward the mapping of the multiple signal classification (MUSIC) algorithm onto the systolic hardware with the ultimate goal to demonstrate a generic surveillance application. It is anticipated that the large efforts in constructing a omprehensive algorithm mapping environment will greatly aid systolic programming.

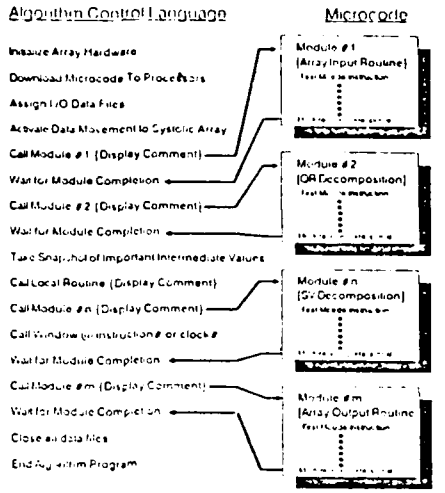


Figure 11. Systolic Software Execution

15. ACKNOWLEDGMENTS

The author wishes to thank Jerry Symanski at NOSC for his many suggestions and guidance in the design of the array architecture. Additional thanks is extended to Manfred Heigl, the principal hardware design engineer and an Australian participating in the exchange scientist program with the Navy laboratory command.

16. REFERENCES

1. Kung, H.T., "Why Systolic Architectures?", Computer Magazine, IEEE Computer Society, January 1982, pp 37-45

2. Symanski, J.J., "Implementation of matrix operations on the two-dimensional systolic array testbed", Proceedings of the SPIE International Technical Symposium, San Diego, CA, 21-26 August 1983

END

DATE

FILMED

6-88

DTIC