GEORGIA INSTITUTE OF TECHNOLOGY — OFFICE OF CONTRACT ADMINISTRATION

PROJECT ADMINISTRATION DATA SHEET

[X] ORIGINAL [ ] REVISION NO. ____

Project No. E-21-606 R6031-0A0 — GTRC/~~GIT~~ — DATE 10 / 22 / 85

Project Director: T. P. Barnwell — School/~~Lab~~ EE

Sponsor: Office of Naval Research, 800 North Quincy Street, Arlington, VA 22217-5000

Type Agreement: Contract No. N00014-85-C-0811

Award Period: From 9/1/85 To ~~8~~/31/87 (Performance) ~~10~~ 3/31/87 (Reports)

Sponsor Amount:	This Change	Total to Date
Estimated: $		$ 1,176,417
Funded: $	200,000	$ 200,000

Cost Sharing Amount: $ ____ Cost Sharing No: ____

Title: A Synchronous Multiprocessor Architecture for Digital Signal Processing

ADMINISTRATIVE DATA — OCA Contact R. Dennis Farmer x-4820

1) Sponsor Technical Contact:

Dr. David W. Mizell
Information Sciences Division
ONR Code 433
800 North Quincy Street
Arlington, VA 22217-5000
(818) 795-5971

2) Sponsor Admin/Contractual Matters:

Mr. Thomas A. Bryant
Office of Naval Research
Resident Representative
206 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0490 (404)881-4374

Defense Priority Rating: DO-C9 — Military Security Classification: ____

a — (or) Company/Industrial Proprietary: ____

RESTRICTIONS

See Attached Gov't. Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval – Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with Georgia Tech is less than $5,000 with prior ACO approval, if greater than $5,000 as determined by the ACO.

COMMENTS:

COPIES TO:

SPONSOR'S I. D. NO. 02.103.016.85.R01

Project Director	Procurement/GTRI Supply Services	GTRC
Research Administrative Network	Research Security Services	Library
Research Property Management	Reports Coordinator (OCA)	Project File
Accounting	Research Communications (2)	Other A. Jones

GEORGIA INSTITUTE OF TECHNOLOGY OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 10/5/88

Project No. E-21-606 / R6031-0A0 School/~~Lab~~ EE

Includes Subproject No.(s) G-36-649 / LeBlanc / ICS

Project Director(s) T. P. Barnwell GTRC/~~GAA~~

Sponsor Office of Naval Research

Title A Synchronous Multiprocessor Architecture for Digital Signal Processing

Effective Completion Date: 1/31/87 (Performance) 3/31/87 (Reports)

Grant/Contract Closeout Actions Remaining:

- [ ] None
- [X] Final Invoice or Copy of Last Invoice Serving as Final
- [X] Release and Assignment
- [ ] Final Report of Inventions and/or Subcontract: Already Submitted
  Patent and Subcontract Questionnaire sent to Project Director [ ]
- [X] Govt. Property Inventory & Related Certificate
- [ ] Classified Material Certificate
- [ ] Other

Continues Project No. Continued by Project No.

COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/~~GTRI Supply Services~~
~~Research Security Services~~
Reports Coordinator (OCA)
~~Program Administration Division~~
Contract Support Division (2)
~~Facilities Management (OEB~~
Library
GTRC
Project File
Other

A SYNCHRONOUS MULTIPROCESSOR ARCHITECTURE FOR DIGITAL SIGNAL PROCESSING

Final Report Submitted to the

Office of Navel Research

by

The Digital Signal Processing Laboratory

School of Electrical Engineering

Georgia Institute of Technology

Atlanta, Georgia 30332

September 19, 1988

Contents

1 Background

This is the final report for a research project which was selected for funding in 1985 by DARPA under the Strategic Research Initiative. The program was initiated in September of 1985, and was successfully pursed until it was terminated in August of 1986 for "lack of funds" at DARPA.

For the past ten years, the problem of implementing digital signal processing (DSP) algorithms on synchronous multiprocessors has been studied at Georgia Tech [1-15] under the sponsorship of the Joint Services Electronics Program (JSEP) and the Army Research Office (ARO). A recent result from that ongoing research program is the development of a set of formal techniques for the automatic realization of optimal synchronous multiprocessor implementations of a large class of digital signal processing algorithms on cyclo-static processors [14,16]. The resulting multiprocessor compiler operated from a graphical representation of the algorithm to be implemented, and generated synchronous multiprocessor implementations which are provably optimal in the sense that there exists no other implementation for the specified algorithm on the specified multiprocessor which is either faster or more efficient. In addition, the resulting implementations belong to the most efficient class of solutions in which the intrinsic synchrony of the system maintains the data precedence relations, and in which no cycles of any of the processors are used for system control.

In 1985, much of the recent work in the area of multiprocessor implementations of DSP algorithms focused on systolic processors. Unfortunately, systolic processors, as formally defined [17], often suffer from low procesŒsor efficiency and for many problems they cannot fully exploit the maximum inherent paraŒllelism of the algorithm [11]. *Skewed single instruction multiple data* (SSIMD) and *parallel skewed instruction multiple data* (PSSIMD), processors [8-10] overcome many of the weaknesses of systolic implementations. Like systolic implementations, they are implemented on synchronous arrays of identical processors with highly constrained communication structures and no broadcast modes. Unlike systolic processors, however, SSIMD and PSSIMD are always optimally efficient and, for recursive systems, they may be optimally fast as well. In addition, it is often

possible to automatically generate optimal SSIMD implementations [15]. However, optimal SSIMD and PSSIMD solutions do not exist for all problems of interest.

Cyclo-static processor solutions overcome the weaknesses of systolic processors and of SSIMD and PSSIMD solutions for the class of algorithms that can be described by cyclic shift-invariant flow graphs. Cyclo-static solutions are a broad family of processor (efficiency) optimal, synchronous, multiprocessor realizations that, by the application of appropriate design choices, can also be rate optimal, input-output delay optimal and communications optimal. For a given algorithm the cyclo-static family of solutions contains many classes which includes rate optimal SSIMD and PSSIMD solutions as special cases, when they exist.

1.1 Algorithm Descriptions

For the purposes of the past research, all algorithms have been specified as *shift-invariant flow graphs* (SIFG) [11]. A SIFG is defined as a directed graph in which all operations are specified at the nodes, and in which the branches are directed paths which specify the flow of data between nodes. The shift-invariant constraint requires that shifting the (set of) input sequences results only in a corresponding shift in the (set of) output sequences.

SIFG's are capable of representing a very large class of interesting DSP algorithms. Of course, they can easily represent all those systems which are representable by signal flow graphs. Hence, they can represent a large class of linear and non-linear as well as time-varying systems. In particular, they are capable of representing digital filters, DFT structures, FFT structures (indeed all fast discrete transform algorithms), correlation, convolution, homomorphic analysis, matched filtering, linear predictive analysis, LMS adaptive filter structures, direct form recursive least squares, and lattice form least squares, to name just a few. In addition, SIFG can also represent many algorithms which are not typically DSP algorithms, including a very large class of matrix operations as well as algorithms specified in terms of low level logic operations (e.g. a digital multiplier structure). In brief, SIFG's can represent all algorithms which do not include any data

dependent branch operations. Clearly, the vast majority of DSP algorithms, as well as other equivalent algorithms, belong to this class.

A *fully specified flow graph* (FSFG) is a SIFG in which the nodal operations are additionally constrained to be the atomic (kernel) operations of the underlying constituent processors which are to be used in the realization. Thus the atomic operations represent the smallest granularity at which possible parallelism may be exploited [11]. The name *generic flow* graph is used to distinguish between those SIFG's that are also FSFG's and those SIFG's whose nodes are not all atomic operations.

1.2 Fully Specified Flow Graph Bounds

The distinguishing characteristic of all this work is the underlying use of bounds on the optimal performance of an algorithm in the generation of the multiprocessor implementations. These bounds are characteristics of the cyclic graph defining the algorithm, not of the mechanism or type of realization. In all, three different bounds are used: the *iteration period bound* (previously called the *sample period bound* [11-13]), which is the minimum possible time between iterations of the algorithm; the *delay bound*, which is the minimum time between the availability of an input (set) and the availability of the corresponding output (set); and the *processor bound*, which is the minimum number of processors required to achieve the specified iteration period. Implementations which achieve the iteration period bound are said to be *rate-optimal.* Implementations which achieve the delay bound are called *delay-optimal.* Implementations which use the minimum possible number of processors for a specified iteration period are said to be *processor-optimal.* In addition, if an implementation is constrained to use only nearest-neighbor communications, it is said to be *communications-optimal.*

The concept of bounds on the multiprocessor realization of flow graphs was first introduced in the context of signal flow graphs by Fettweis [18] and later extended by Renfors and Neuvo [19] and Schwartz and Barnwell [12-14]. The basic assumptions are that the algorithm to be implemented is represented as a FSFG, and that the algorithm is to be implemented on a synchronous multiprocessor. It is also assumed that the computation

times for all the nodal operations are known. This condition is easily met in a homogeneous multiprocessor in which all the processing elements are the same. Nothing is assumed about the communications structure, I/O constraints, or the details of the realization. Hence, the resulting bounds reflect the absolute limits on computation rate and delay based on the atomic structure of the constituent processors.

Of course, many things besides the structure of the algorithm and the fundamental operational capability of the processors may limit DSP implementations. Clearly, things such as I/O bandwidths, external resource availability, the number of available processors, and the communications architecture may impact the achievable rate, delay, and processor efficiency of an algorithm. But in their own way, each of these aspects can be addressed and corrected. For a particular multiprocessor system and a particular FSFG, the above bounds are fundamental. Hence, if total implementations can be developed which achieve these bounds, then it is clear that no other implementations exist which can operate at a higher rate, with less delay, or with higher efficiency. It is this class of optimal implementations which are automatically generated by the cyclo-static compiler.

1.3 Systolic Processors

The area of systolic processors has attracted the attention of many researchers in parallel processing for scientific and signal processing tasks. The term systolic is assumed to mean a regular iterated array of computational elements with strict nearest neighbor communications. There is a global clock, and on every clock cycle each processor inputs data, operates on the data and outputs data at the end of the cycle. No information can flow further than one processor in one system clock cycle. Systolic processors are a special case of a static pipeline. Until very recently the area was dominated by systolic solutions to specific problems, presented without verification or derivation. However, a number of systematic and rigorous approaches to systolic implementations have now appeared, including one from our research program [12].

The development of the cyclo-static compiler really grew out of an attempt to understand why SSIMD and PSSIMD implementations could

be rate-optimal, processor-optimal, and communications-optimal while the systolic implementations for the same algorithms could not. There are really two separate reasons for the shortcomings of systolic arrays. The first is the fact that systolic processors are static pipelines. This means that any particular operation in an algorithm is assigned to a particular processor in the systolic array, and that operation is performed by that processor on every iteration. Hence, the operations are static and only the data moves through the multiprocessor. In contrast, SSIMD, PSSIMD, and cyclo-static processors are dynamic pipelines in which both the operations and the data move through the multiprocessor. The second reason is the global transfer clock. Indeed, this global transfer clock was the basic characteristic for which systolic arrays were named, giving the whole system it's "pumping" action [17]. The point is that there is no fundamental requirement that all the pipeline registers in the system be clocked simultaneously. Stated another way, there is no reason that each processor must perform I/O on every clock cycle. In contrast, the input-output operations in SSIMD, PSSIMD, and cyclo-static implementations move in parallel, non-overlapping wave front [16].

1.4 SSIMD Implementations

Historically, skewed single instruction multiple data (SSIMD) implementations were the first class of solutions which could overcome the systolic constraints and consistently achieve rate-optimal, processor-optimal implementations with nearest neighbor communications for a large class of interesting algorithms [8-10]. In SSIMD, exactly the same program is executed on each of the processors in the multiprocessor, and that program realizes exactly one iteration of the flow graph. In an SSIMD program, all of the arithŒmetic operations appear as exŒplicŒit inŒstructions, but the delay nodes are transŒformed into input-output pairs. In this way, the delay structure in the flow graph becomes the communications structure in the SSIMD realization [8-10].

For any given program and any given constituent processor, it is possible to compute a sampling period bound for the SSIMD realization [6]. This *SSIMD bound* for programs is equivalent to the iteration period bound for

fully specified flow graphs. Hence, if a program can be generated such that the SSIMD bound is equal to the iteration period bound, then the SSIMD realization is rate-optimal.

The SSIMD approach to flow graph realizations is very attractive for many reasons. First, for all SSIMD realizations in which the number of processors is less than the processor bound, the implementations are perfectly efficient and the use of N processors always increases the throughput by a factor of exactly N. Second, when the SSIMD iteration period bound is equal to the iteration period bound, as is the case for the majority of recursive digital filter structures, then there exists no multiprocessor solution using the same constituent processor which is faster or more efficient. Third, although the sample-period-bound concept is not involved, SSIMD realizations work equally well for non-recursive structures. Finally, and most importantly, the all-important communications architecture for the final implementation is completely specified by the delay node structure of the flow graph. In particular, by constraining all the delay nodes to be first order, all single-time-index (one-dimensional) SSIMD solutions can be realized with a nearest-neighbor unidirectional ring. A similar result applies to two-dimenŒsional flow graphs, [8-10]. However, if a more complex communications mechanism is available, then the flow graph can be defined to take advantage of it [6].

An SSIMD compiler has been demonstrated [15] which generates full multiprocessor implementations for a laboratory multiprocessor [5]. This compiler finds a rate-optimal SSIMD implementation, if it exists, and the best SSIMD implementation, if it it does not. SSIMD implementations are always processor-optimal and communications-optimal. Two important points should be made concerning this SSIMD compiler. First, its application is by no means limited to the laboratory multiprocessor around which it was developed, and it can quite easily be used in top-down design systems using microprocessors, signal processing chips, or VLSI realizations. Second, and more important, is the result that if a rate-optimal SSIMD solution exists, it is very easy to find. Stated another way, the information available from the computation of the flow graph bounds defines so precisely the character of a rate-optimal solution that it is very simple to

test whether an optimal SSIMD solution exists and to find it if it does. In contrast, finding the best sub-optimal solution is much more computationally intense. Hence we have the paradox that the most desirable optimal solutions are the easiest to find, but they may not always exist.

1.5 Cyclo-Static Implementations

A cyclo-static system is a synchronous multiprocessor system that is deterministically scheduled. The schedule (or program) is characterized by its periodicity, with period related to the iteration interval. The term cyclo-static connotes an idea similar to cyclo-stationarity of random processes.

Since the schedule of the processors is deterministic, the schedule automatically handles all precedence requirements, eliminating the need for synchronization or semaphore mechanisms. In cyclo-static systems the processors are only performing direct operations of the defining algorithm, or FSFG.

To understand the character of cyclo-static processors, it is illuminating to consider the program for the complete computation of one iteration as a pattern in the processor-time space (P X T). The processor space may be multidimensional, with the indexing of processors within a single dimension being a cyclic ring. For a one dimensional processor space, this diagram is a reservation table or Gantt chart. The *principle lattice vector* is a vector which connects a particular operation in one iteration to the same operation in the next iteration in (P X T). In this context, systolic processors and other static pipelines support implementations for which the principal lattice vector has only a time component. In contrast, cyclo-static schedules are distributed in both space and time, with principal lattice vectors that have both space and time components.

While any problem that iterates forever (or for a long enough time to render the effects of setup and flush negligible) and has a deterministic periodic program can be considered cyclo-static, this research has taken a narrower view. In addition to the above properties, there is an additional constraint that the solution be processor-optimal. It is the existence and ability to find processor-optimal solutions to problems with periodic programs that makes cyclo-static solutions unique.

Finding processor optimal solutions is significantly easier than finding suboptimal solutions. In fact, the more optimallity constraints imposed, (i.e. processor, rate, delay and communications optimal) the easier it is to find a solution, if it exists. Cyclo-static processor and rate optimal solutions exist for all recurrence systems defined by shift-invariant FSFG's. The existence of solutions meeting other combinations of optimallity criteria may not exist, and are problem dependent.

The cyclo-static compiler [16] fundamentally performs a highly constrained tree search in order to find rate-optimal, processor-optimal, communications-optimal, and sometimes delay-optimal cyclo-static solutions. The basic scheduling problem addressed by the cyclo-static compiler is well known to be NP-complete, and might hence normally be addressed using a heuristic rather than an optimal approach. One of the most surprising results of our research thus far is that the optimal cyclo-static scheduling problem is tractable for most recursive problems of interest. This is because the optimallity criteria impose powerful constraints on the size of the solution search space. The resulting problem is still NP-complete, but the order of the problem has been dramatically reduced. It is the loop structure of the graph, rather than the size of the graph, which fundamentally determines the computational complexity of the compilation procedure. Hence, it is often possible to compile very large graphs very quickly.

2 The OSCAR Research Program

Even though all of the research prior to 1985 was done in the context of a working laboratory multiprocessor computer system [5], the work has been primarily of a theoretical nature. Thus at the start of this program, it was fair to say that the fundamental approach had been validated and it appeared to have great promise for optimal implementation of massively computationally intense DSP algorithms on large, synchronous multiprocessor machines. The purpose of this research program was to extend and test these results on large, realistic algorithms in the context of large, powerful multiprocessor machines.

The proposed research was to have two major components. The first

component was the expansion and extension of the theoretical results. In addition to the work already in progress, this new work was to concentrate on the problem of extending the existing formalism to a broader class of problems, and to contexts in which more than one algorithm is being implemented simultaneously. The issues of the complexity of the compilation process for large problems would be addressed, as would the questions of data availability and buffering, and algorithm initiation and termination.

The second component was the development and utilization of a prototype for a synchronous multiprocessor computer specifically designed for cyclo-static implementations. The multiprocessor architecture which was designed as part of this program was not itself a radical departure from previously proposed architectures. It was rather an extension of traditional techniques which would allow for a radically new approach to programming and realizing digital signal processing algorithms. The final system was to be a 64 processor floating point machine which would operate with a peak arithmetic rate of 1.024 GIGAFOPS and a communications bandwidth of 64 Mbytes/second. Despite its impressive specifications, this is not a particularly expensive machine. Fundamentally, sophisticated compilation tools are used to generate provably optimal implementations for a relatively simple multiprocessor architecture to give an order-of-magnitude improvement in the cost/performance ratio.

The overall development of a working prototype multiprocessor was to be realized in a four phase effort. The first phase was the design and implementation of a single prototype processor. The second phase was the PC layout, construction, and testing of a two processor prototype. The third phase was the implementation of a 16 processor system, configured as a 4 X 4 square array. The multiprocessor compilers developed in the parallel research effort were to be integrated in the third phase. The fourth phase was the construction of a full 64 (or larger) processor system. The first three phases was to be performed in the digital signal processing laboratory at Georgia Tech, while the last phase was probably to be performed in conjunction with an outside contractor.

Before the termination of the program, all of phase one and most of phase two was complete. At the time of its termination, the contract was

on-schedule and on-budget. The multiprocessor which was designed was called the *Optimal Synchronous Cyclo-Static ARray* or *OSCAR.*

3 The OSCAR Architecture

It is desired that a realization should be processor optimal, rate optimal, and communications optimal. In order to meet these goals several constraints are implied by the cyclo-static compiler. Note that realizations that meet these criteria may exist without these constraints, for a given algorithm. However, the constraints make possible a practical compiler that can find solutions. Without these constraints, the problem of finding a schedule of operations (solution) becomes computationally intractable (NP-hard in the strong sense).

Thus the OSCAR was designed to meet the following constraints.
Constraints for Cyclo-Static

- All processors are homogeneous.
- The system is fully synchronous.
- The computational delay of all operations are data independent.
- The computational delay of all operations (measured in system clock units) are relatively non-prime.
- For a given operation, all operands can be fetched from adjacent processor or locally and result stored locally (dual: all operands fetched locally and result can be stored in adjacent processors or locally) with computational delay independent of local vs. adjacent processor location.
- Interprocessor communications between adjacent processor pairs is independent (w.r.t delay, conflict, etc.) of all other adjacent processor communications. If a processor has four adjacent processors, it can output to all four adjacent processors (dual: it can input from all four adjacent proces Œsors) simultaneously and without conflict.

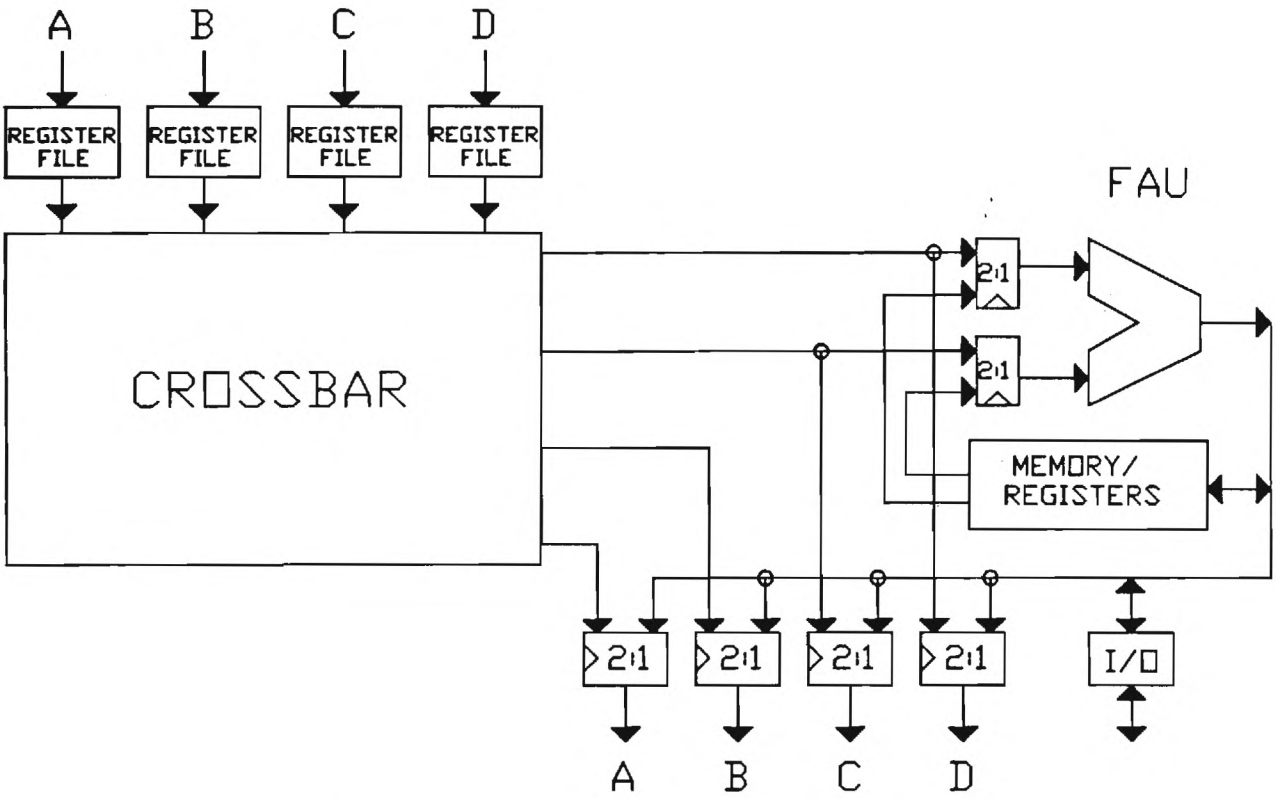

Figure 1: The system architecture for one of the OSCAR processors.

4 The Hardware Architecture

OSCAR is basically a regular square array (4 x 4) of 32 bit processors. Each processor consists of five fully parallel communications ports, a 32 bit floating point arithmetic unit (FAU), partitioned local memory, address generation unit, general purpose 32 bit integer processor, micro-controller with writable control store and a debugger/monitor microprocessor. The OSCAR system architecture is illustrated in Figure 1. In many ways, interprocessor communications is the true heart of the machine, so it will be discussed in the most detail.

Communications ports Each processor has five communication ports. Four of the ports are connected to the four nearest neighbors. The fifth port is cable pluggable and can be connected as a system input port, system

output port, or connected to another processor. This extra uncommitted communications link allows for the possibility of link fault tolerance as well as embedding other communications structure in the rectangular mesh (i.e. a binary tree, hexagonal mesh or a perfect shuffle [9]). Note that the input and output links are separate 32 bit parallel links and thus data can concurrently flow bi-directionally on all links.

In order to provide the concurrency and the high bandwidth of the ports, they are realized by a separate multiported register file with a parallel bypass latch for each port. The multiport allows for the a link input, a link output and a local processor read or write to the port to occur simultaneously. Within a processor, the five ports are interconnected by a full crossbar switch. This allows the ports to act as a fast general purpose, five in, five out, network communications switch. The communications switch is fully concurrent with internal processing in the processor node. The bypass latch is required to support the switch network function due to the lack of register files that are sufficiently fast. Data that is consumed (or generated) locally is placed in the register file for fast processor access.

Associated with each word of storage in the register file is a semaphore (full/empty) flag bit. Writing a word sets the flag and reading clears the flag. In the cyclo-static and systolic mode, attempting to read a word that has not been written yet generates a *I/O fault exception* that transfers control to the debugger/monitor. In data driven modes, attempting to write to a non-empty word puts the sender in the *wait-until-empty* state. Similarly, attempting to read an empty word puts the receiver in the *wait-until-full* state. In either wait state, a timer is activated. If the timer times out before exiting the wait state a *deadlock fault exception* is generated. Additionally, in the data driven mode, microcode can be used to control the register file so that it implements a FIFO buffer for data queuing. This allows for wavefront or data flow type machines.

Due to the high speed of the machine the different modes of operation have different system timings. In the cyclo-static mode, an I/O fault exception is handled after the fact on the next successive instruction. In data driven mode the system must be slowed down enough so that the semaphore can be tested and wait states inserted as appropriate.

In the cyclo-static or systolic mode, the fifth port contains an additional mechanism that allows wait stating the entire machine through a similar semaphore. This allows for simple synchronization to external input or output data streams. Thus, the OSCAR will support multiple concurrent tasks, in this case the semaphore of the fifth port will only wait state those processors that belong it's associated *task group.*

In the cyclo-static mode all operations have a major cycle time of 200 ns. The major cycle is subdivided into four minor cycles. Each 50 ns minor cycle is a communications cycle. The first three minor cycles are dedicated to three communications network switching slots, which are concurrent with a floating point operation execution. The fourth minor cycle is dedicated to allowing the result of the completed operation to be written to any subset of the five output ports (adjacent processors). Therefore each processor executes at 5 MFLOPS and there is a communications bandwidth per processor of (5 ports)*(4 bytes/port)/(50 ns), or 400 Mbytes/s. Since a processor can only consume two operands per operations, this would conventionally suggest that the peak communications bandwidth should only equal (2 words/operation)*(4 bytes/word)/(200 ns/operŒation) which equals 40 Mbytes/s.

Note that this suggests that there is a factor of 10 excess communications bandwidth. Recalling the list of constraints that make it possible to find cyclo-static solutions, it can be seen that there is a requireŒment of (5 ports)*(4 bytes/port)/(200 ns), or 100 Mbytes/s. Only one of the communications slots is strictly required. However, it may be that the particular algorithm to be realized requires more than four/five nearest neighbors. It is more flexible and economical to provide more communication bandwidth per port than to increase the number of ports. By allowing three extra communication cycles, it is possible to communicate with all processors with a city block metric distance of two (twelve neighbors vs. four), by requiring that they transfer on a minor cycle during the previous major cycle. Thus adjacent communication distance is a linear function of time. While all of the implications in terms of designing a compiler that can take maximum advantage of this communication structure have not been determined, preliminary results show it to be extremely flexible and powerful.

The other major need for the communication bandwidth is for algorithms that partition a large amount of data over the set of processors (i.e. image processing, matrix inversion, etc.). In these algorithms there is usually a need to shuffle back and forth moderate to large sets of data to handle cases such as overlapped boundaries. It is more effective to increase communications bandwidth so that these data transfers can be concurrent with other operations, then to dedicate machine cycles for only data movement.

Floating point arithmetic unit (FAU) The floating point arithmetic unit is designed around an Am29325 32 bit floating point processor chip. The FAU can perform floating point addition, subtraction, multiplication and comparison as well as convert from integer to floating point and floating point to integer. Iterative reciprocal and square root approximations are being studied for inclusion.

It should be noted that the Am29325 is non-pipelined and has a "gate delay" of ¡ 150 ns. This is significant when compared to alternate chips such as the Weitek which are five stage pipelines with a stage time of 200 ns. Pipelined arithmetic units are very difficult to keep full and introduce significant computational latencies.

The FAU is connected in what is effectively a partial crossbar to all of the port register files, the integer processor and the local memory. Operands and results can be sourced and sinked to most combinations of these resources.

Local processor memory The local processor memory is partitioned into a fast and slow 32 bit word memory. The fast memory is 64 Kwords of static RAM with a 50 ns cycle time. To achieve this speed, address generation is pipelined one minor cycle ahead. The slower memory is 256 Kwords (or greater) of dynamic RAM with a 150 ns cycle time.

Address generation unit (AGU) The address generation unit has not been fully designed at this time. However it generates an address for the local memory every 50 ns. This high speed limits the flexibility of the AGU. The design includes a 4K RAM table of 16 bit words (sufficient to address

the fast local memory). Addresses are generated by sequentially fetching the addresses from the *address table.* In addition there is a generator that supports address generation for traversing multidimensional arrays along a principal direction with a fixed stride, and for ring buffering multidimensional data.

Integer processor The integer processor is for the MIMD and data driven MIMD modes. In the cyclo-static and systolic modes it is not accessible. This is to support *coarse grain parallelism*, and is for the support of general purpose computing. The integer processor can use the FAU as a coprocessor, but at the penalty of lower floating point performance then in the cyclo-static and systolic modes. The integer processor will probably be a minimum configured M68020 32 bit microprocessor. The slow portion of the local memory is for program space for the integer processor. The choice of the M68020 was strongly influenced by its very high performance and the wide availability of existing software support.

Since there is no global memory, the integer processor communicate with adjacent processors through message passing or by a strict data driven mode. The semaphores flag of the ports are used for inter-process synchronization.

Micro-controller The micro-controller is implemented with a TI microsequencer and a *writable control store.* The controller is nanocoded with a highly horizontal nanocode.

Debugger/monitor . The debugger/monitor is a 80188 microprocessor system. Each processor's debugger is tied together on a common low speed address/data bus on the system backplane. It will serve three primary functions: 1) it will provide and load the *bootstrap* microcode into the writable control store in order to initiate processing, 2) it will set and detect a number of fault and control bits and communicate that information to the system level control software on the host and 3) it will load and execute diagnostic, debugging and monitoring software.

5 Discussion

OSCAR represents a different approach to building a multiprocessor system. For any synchronous multiprocessor, any algorithm and any criterion of optimallity, there exists one or more optimal implementations. However, the complexity of the task of determining the bounds on performance and of automatically finding the corresponding optimal implementations is prohibitive for most architectures. Hence, we are typically reduced to laboriously generating a few ad hoc implementations which may or may not be optimal.

OSCAR was a multiprocessor system with a few special features. These features do not make the OSCAR architecture a dramatic departures from traditional architectures, and the performance specifications of OSCAR are similar to those of many other multiprocessors. But because of these few features, optimal implementations for a very broad class of important algorithms can be automatically generated for OSCAR, whereas such implementations are very hard to find for other machines. This would have made OSCAR a uniquely usable multiprocessor.

6 Commentary

It would not be appropriate to complete this report without commenting on the program management for this project. This contact was obtained through a response to an open RFP from DARPA and ONR. The required proposal was technically very detailed, and explained explicitly and completely the goals of the research program. When we received the contract, we were told that our work was very important and we were told that the project was fully funded. We were pressured to move as quickly as possible.

This proposal came from a very healthy research program within the Digital Signal Processing Laboratory at Georgia Tech. At the start of the program, all of the key staff were already at Georgia Tech, but new staff had to be added handle their previous functions. Thus, this contract represent an ambitious undertaking for our research group. Nonetheless, our actual performance was outstanding. All of the start-up problems were solved,

and the project was always on-schedule and on-budget.

The project never had a technical monitor either at ONR or DARPA. A technical monitor was assigned before the contract started, but when he resigned from ONR, he was not replaced. We were never able to get any sort of technical review by anyone after the contract started. We found this very disturbing, since we felt we had many important things to say.

When the contract was canceled, it was done so for "lack of funds" at DARPA. This was done even though DARPA was simultaneously starting new contracts.

The total effect of all of this was very negative. DARPA offered, through an RFP, a chance to extend our research. It took great effort to respond to that RFP. Then, even though we accomplished every stated goal of our proposed research, our project was canceled without technical review. As a result of the shocking behavior of DARPA in this regard, several key people left Georgia Tech. Thus, by responding in good faith to a DARPA initiative, the result was great damage to a previously healthy program. In addition, of course, while wasting our time, DARPA also wasted the $350,000 spent on this contract.

Clearly, this kind of management cannot be in the national interest.

References

[1] T.P. Barnwell, C.J.M. Hodges, and S. Gaglio, "Efficient Implementations of One and Two Dimensional Digital Signal Processing Algorithms on a Multi Processor Architecture," *1979 International Conference on Acoustics, Speech, and Signal Processing*, Washington, DC, April 1977.

[2] T.P. Barnwell, S. Gaglio, and R.M. Price, "A Multi-Microprocessor Architecture for Digital Signal Processing," *Proceedings of the 1978 International Conference on Parallel Processing*, August 1978.

[3] C.J.M. Hodges, T.P. Barnwell, and D. McWhorter, "Implementation of an All Digital Speech Synthesizer Using a Multimicroprocessor Architecture," *1980 International Conference on Acoustics, Speech,*

and Signal Processing, Denver, Colorado, April 1980.

[4] T.P. Barnwell, III, "Optimal Implementation of Recursive Signal Flow Graphs on Synchronous Multiprocessor Architectures," *IEEE Workshop on Digital Signal Processing and VLSI*, Santa Barbara, CA, Sept. 1981.

[5] T.P. Barnwell and C.J.M. Hodges, "A Synchronous Multi-Microprocessor System for Implementing Digital Signal Processing Algorithms," *Professional Program Session Record 21 of Southcon/82*, pp. 21/4/121/4/6, March 1982 (invited).

[6] T.P. Barnwell, III, C.J.M. Hodges, and M. Randolf, "Optimum Implementation of Single Time Index Signal Flow Graphs on Synchronous Multiprocessor Machines," *Proceedings of ICASSP '82*, Paris, France, pp. 687-690, May 1982.

[7] T.P. Barnwell, III, and C.J.M. Hodges, "Optimal Implementation of Signal Flow Graphs on Synchronous Multiprocessors," *Professional Program Sessions Record 22 of Electro/82*, June 1982 (invited).

[8] T.P. Barnwell, III, "Synchronous Techniques for Signal Flow Graph Implementation," NSF Workshop on Digital Signal Processing, Washington, DC, June 1982.

[9] T.P. Barnwell and C.J.M. Hodges, "Optimal Implementation of Signal Flow Graphs on Synchronous Multiprocessors," *Proceedings of the 1982 International Conference on Parallel Processing*, Belaire, MI, August 1982.

[10] T.P. Barnwell, III, and C.J.M. Hodges, "Optimal Implementation of DSP Algorithms on Synchronous Multiprocessors," *Proc. of Workshop on Algorithmically Specialized Computer Organizations*, Oct. 1982.

[11] T. P. Barnwell III and D. A. Schwartz, "Optimal Implementation of Flow Graphs on Synchronous Multiprocessors," *Proc. 1983 Asilomar Conf. on Circuits and Systems*, Pacific Grove, CA, Nov. 1983.

[12] D. A. Schwartz and T. P. Barnwell III, "A Graph Theoretic Technique for the Generation of Systolic Implementations for Shift-Invariant Flow Graphs," *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, San Diego, CA, March 1984.

[13] D. A. Schwartz and T. P. Barnwell III, "Increasing the Parallelism of Filters Through Transformation to Block State Variable Form," *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, San Diego, CA, March 1984.

[14] D. A. Schwartz and T. P. Barnwell III, "Cyclo-static Multiprocessor Scheduling for the Optimal Realization of Shift-Invariant Flow Graphs," *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, Tampa, FL, March 1984.

[15] S. H. Lee, C. J. M. Hodges, and T. P. Barnwell III, "An SSIMD Compiler for the Implementation of Linear Shift-Invariant Flow Graphs," *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, Tampa, FL, March 1984.

[16] D. A. Schwartz, "Multiprocessor Techniques for the Realization of Shift Invariant Flow Graphs", Ph.D. Thesis, School of Electrical Engineering, Georgia Institute of Technology, June, 1985.

[17] C. E. Leiserson, *Area-Efficient VLSI Computation*, 1983, MIT Press, Cambridge, MA.

[18] Alfred Fettweis, "Realizability of Digital Filter Networks," *Arch. Elek. Ubertragung*, Vol. 30, Feb. 1976, pp. 90-96.

[19] Markku Renfors and Yrjo Neuvo, "The Maximum Sampling Rate of Digital Filters Under Hardware Speed Constraints," *IEEE Trans. on Circuits and Systems*, Vol. CAS-28, No. 3, March 1981, pp. 196-202.