SPEC Systems & Processes Engineering Corp.
Suite A • 1406 Smith Road • Austin, Texas 78721

156639
P-78

Final Report

DFT Algorithms for Bit-Serial GaAs Array Processor Architectures

Contract Number NAS5-30291

Prepared by:

Dr. Gary B. McMillian
Systems & Processes Engineering Corporation (SPEC)
1406A Smith Road
Austin, Texas 78721

Prepared for:

NASA Goddard Space Flight Center
Greenbelt Road
Greenbelt, Maryland 20771

(NASA-CR-183436) DFT ALGORITHMS FOR BIT-SERIAL GaAs ARRAY PROCESSOR ARCHITECTURES Final Report (Systems and Processes Engineering Corp.) 78 p CSCL 09B

N89-27361

Unclas
G3/61 0156639

August 19, 1988

SPEC
Systems & Processes Engineering Corporation

ABSTRACT

Systems & Processes Engineering Corporation (SPEC) has developed an innovative array processor architecture for computing Fourier transforms and other commonly used signal processing algorithms. This architecture is designed to extract the highest possible array performance from state-of-the-art GaAs technology. SPEC's architectural design includes a high performance RISC processor implemented in GaAs, along with a Floating Point Coprocessor and a unique Array Communications Coprocessor, also implemented in GaAs technology. Together, these data processors represent the latest in technology, both from an architectural and implementation viewpoint.

SPEC has examined numerous algorithms and parallel processing architectures to determine the optimum array processor architecture. SPEC has developed an array processor architecture with *integral* communications ability to provide maximum node connectivity. The Array Communications Coprocessor embeds communications operations directly in the core of the processor architecture.

A Floating Point Coprocessor architecture has been defined that utilizes Bit-Serial arithmetic units, operating at very high frequency, to perform floating point operations. These Bit-Serial devices reduce the device integration level and complexity to a level compatible with state-of-the-art GaAs device technology. Operating at clock frequencies in excess of 1 GHz, these Bit-Serial units compare favorably to parallel units implemented in silicon technology, while providing inherent radiation hardness and superior speed-power product of GaAs.

SPEC has selected Sun Microsystems' Scalable Processor ARChitecture (SPARC) as a basis for the high speed RISC processor. The SPARC is ideally suited for array processor applications, with a large register set, efficient instruction set, and simple implementation. The SPARC RISC processor has previously been implemented in a silicon gate array, with the design requiring less than 20,000 gates. This compares very favorable to other RISC processor implementations, which have required many times the device complexity.

SPEC has selected Vitesse Semiconductor's enhancement/depletion mode process for design implementation. Vitesse's GaAs foundry is now offering Standard Cell design capability up to 20,000 gates, which offers the best cost and performance alternative, and ensures success in a Phase II development activity.

In selecting SPARC basis for the processor, SPEC has ensured a high level of software support and design activity for successful commercialization of the product in Phase III. At the end of Phase II, SPEC will have demonstrated both a high performance DFT array processor architecture and GaAs RISC design.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

Systems & Processes Engineering Corporation (SPEC) has conducted an in-depth investigation of signal processing algorithms, with the goal of developing a high performance, GaAs based, systolic array architecture suitable for implementation of these algorithms. As a result of this study, SPEC has developed an array architecture suitable for implementing a broad class of signal processing algorithms, with possible application to other computing problems, including computational physics and artificial intelligence problems.

The study has been concentrated on algorithms for computing the Discrete Fourier Transform (DFT). The DFT has received considerable usage over the years in applications ranging from seismic well-logging, image analysis, and spectral analysis in the commercial and scientific sectors, to communications signals intercept and sonar detection in the military sector. In many of the applications, the computation of the DFT is very time critical. Real-time applications involving image compression, digital Fourier spectroscopy, and harmonic analysis require the transformation of high speed data signals by specialized hardware capable of providing high system throughput and minimum latency.

In addition to the Fourier transform, SPEC has investigated systolic array implementations of Kalman filter algorithms, Finite Impulse Response (FIR) algorithms, and image filtering algorithms.

The primary limitation, observed in all classes of algorithms, is node connectivity. Solving the connectivity problem is key to the successful implementation of a high performance array processor. Therefore, SPEC has developed an inter-processor communications architecture capable of providing flexible, high speed, and easily programmable inter-node communications.

SPEC's systolic array architecture features a GaAs implementation of a RISC processor, floating point coprocessor, and array communications coprocessor. A number of RISC architectures were evaluated for application to signal processing and compatibility with the state-of-the-art in GaAs integrated circuit technology.

SPEC has selected the Sun Microsystems Scalable Processor ARChitecture, SPARC, for implementation of the systolic array processor. This architecture features a large register set, a flexible coprocessor interface, a simple instruction set, and because of it's simplicity, can be implemented with near-term GaAs integrated circuit technology.

SPEC's architecture also features very high speed, low gate count, bit-serial arithmetic and communication units in the floating point and communication coprocessors, respectively. Utilizing the very high speed of GaAs, currently with clock rates in excess of 1 GHz, bit-serial units can be used to form the core of complex arithmetic and communication units. A bit-serial VLSI architecture is, in fact, ideal for implementation of the communication links between processors. GaAs based bit-serial floating point arithmetic units will be of comparable performance to highly parallel silicon floating point units, while maintaining the inherent radiation hardness of GaAs and the approximately 10:1 speed-power product advantage of GaAs over ECL integrated circuits.

The core GaAs RISC processor and coprocessors will maintain complete 32-bit external architectures. As a result, the processor and coprocessors will be suitable for implementation in single processor designs, as well as the proposed systolic array processor. This should significantly enhance the commercial viability of the product.

After evaluation of the state-of-the-art in GaAs standard cell design and the complexity of the processor architecture, SPEC is confident that the proposed design can be successfully demonstrated in a Phase II program and successfully commercialized in a Phase III program.

2.0 Systolic Array Topologies

A number of array topologies have been utilized in the past for interconnection of multiple processors. Array processors have been configured in 2-D, 3-D, Ring, Linear and Algorithm Specific topologies.

Data can be transferred between nodes through serial or parallel communication links, through dual-ported random access memory (RAM), or through shared (global) RAM. For arrays with connections between printed circuit boards, or for nodes with more than one or two communication links, implementation of dual-ported or shared RAM is impractical.

Utilizing GaAs integrated circuits, very high speed serial communication channels are realizable, with the principal speed limitation being the physical link. In fact, by devising a communications coprocessor to buffer the data and handle the physical link, the data transfer operation can be overlapped with the execution of floating point or fixed point calculations in the processor and other coprocessors.

The operation of array processors falls into two basic categories: Single Instruction Multiple Data (SIMD) stream operation and Multiple Instruction Multiple Data (MIMD) stream operation. In a SIMD processor array all processors execute a common instruction stream, while operating on separate data. In a MIMD processor array, each processor is capable of executing a separate instruction stream while operating on unique data.

An example of a mesh connected SIMD processor is the Geometric Arithmetic Parallel Processor (GAPP). The GAPP, is a array of 1-bit processors, each communicating with its nearest neighbor in a 2-D plane. NCR Corporation currently manufactures a 6 x 12 processor array implementation of the GAPP with a bit-serial ALU and 128 bits of static RAM per processor.

Examples of MIMD machines include the Intel HyperCube array processor and Inmos Transputer based array processors. The Intel HyperCube processor features an array of standard 80386 processors with 80387 floating point coprocessors connected in a 3-D array. The Inmos Transputer, which is a VLSI 32-bit processor with four 20 Mbit/sec serial communication channels available for inter-processor communication, can be configured in an arbitrary array topology. Inmos has developed the parallel processor programming language OCCAM for programming the Transputer.

2.1 Two Dimensional Plane Topology

Two dimensional topologies are suitable for computing matrix operations. For example, filtering of images using a local matrix transformation is ideally suited for a mesh connected 2-D systolic array.

A diagram of a 2-D array is shown in Figure 2-1. In this implementation each node processor communicates with its nearest neighbor through a high speed communications coprocessor. The communications coprocessor buffers the data and manages the physical interface between nodes.

Provisions can be made for failed processors to be mapped out of the array, with communications routed around the failed node.

2.2 Three Dimensional Plane Topology

A systolic array processor configured in a 3-D array is shown in Figure 2-2. In this configuration each node processor can communicate with its coplaner nearest neighbor and its nearest neighbors in adjacent planes.

Each communication coprocessor manages six full-duplex communication links. Data can flow in any of the six directions out of a node, and can flow in basically any direction in the array; between planes, within planes, and a combination of the two.

Each processor in the array can execute a portion of the algorithm, operating on all or a subset of the data, or each processor can execute the complete algorithm, operating on a fraction of the data.

2.3 Linear Topology

Pipelining of functional operations is usually accomplished with a linear array topology, such as that shown in Figure 2-3. With this array configuration each processor usually handles all of the data, performing one or more operations and passing the processed data to the next node.

Redundancy in communication links is also shown in Figure 2-3. This link redundancy serves multiple purposes: fault tolerance, increased throughput, data separation, and others. Multiple links between nodes can also be utilized in other array topologies.

An example of a node bypass is also shown in Figure 2-3. If a processor fails, then the communications bypass link can be activated to map out the failed node.

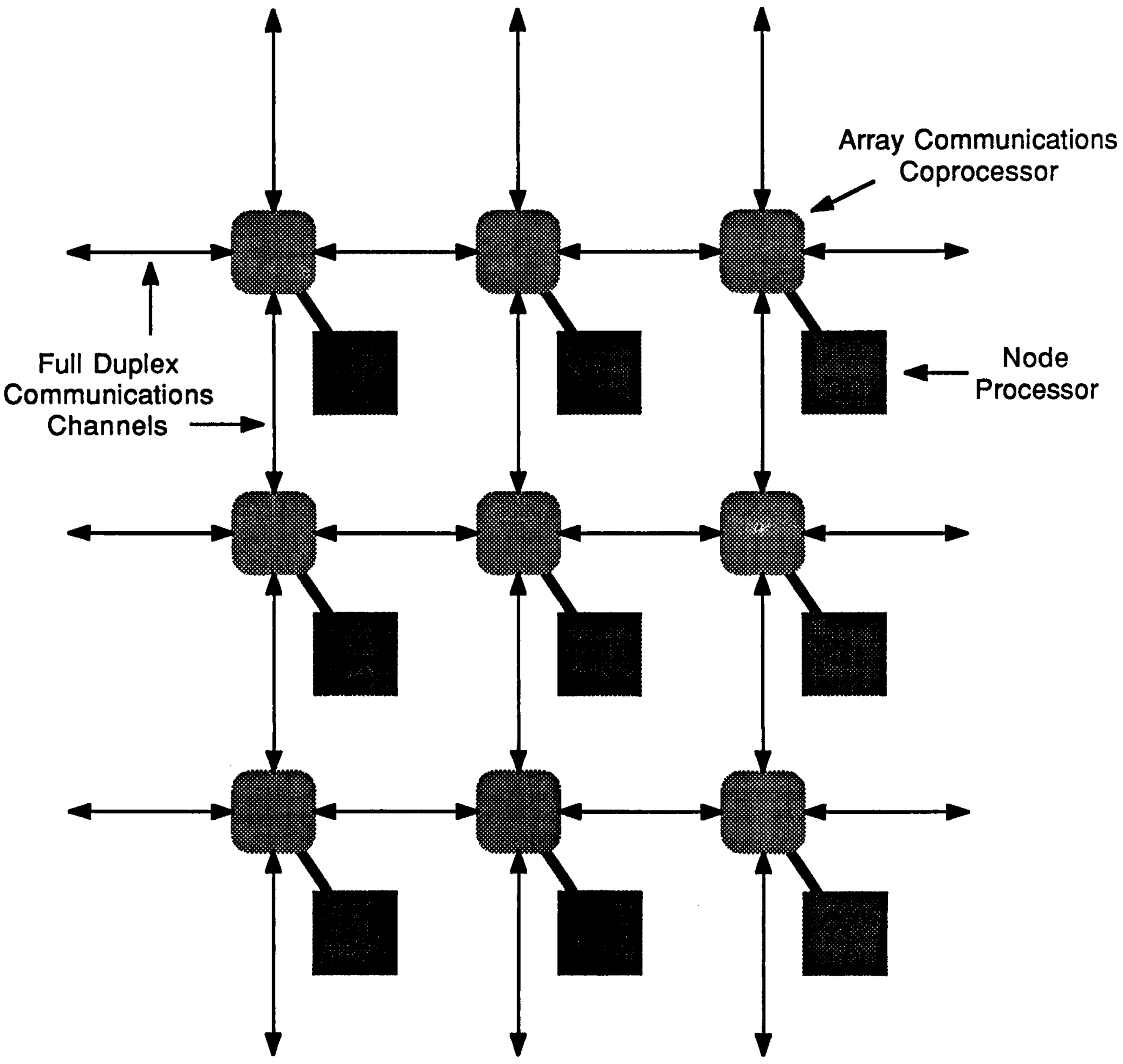

Array Communications Coprocessor
Full Duplex Communications Channels
Node Processor

Figure 2-1 Two Dimensional Array Topology.

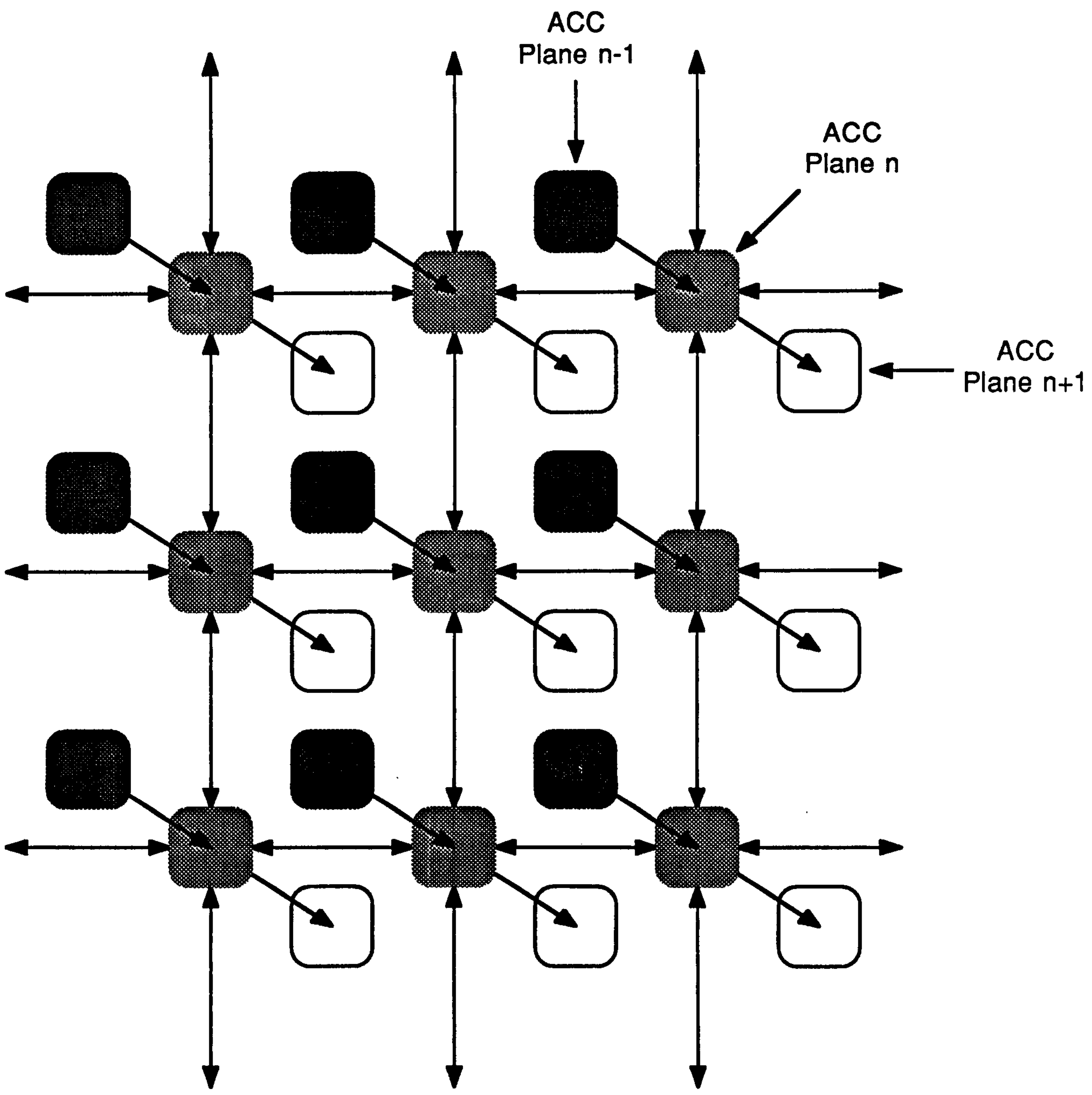


Figure 2-2 Three Dimensional Array Topology.

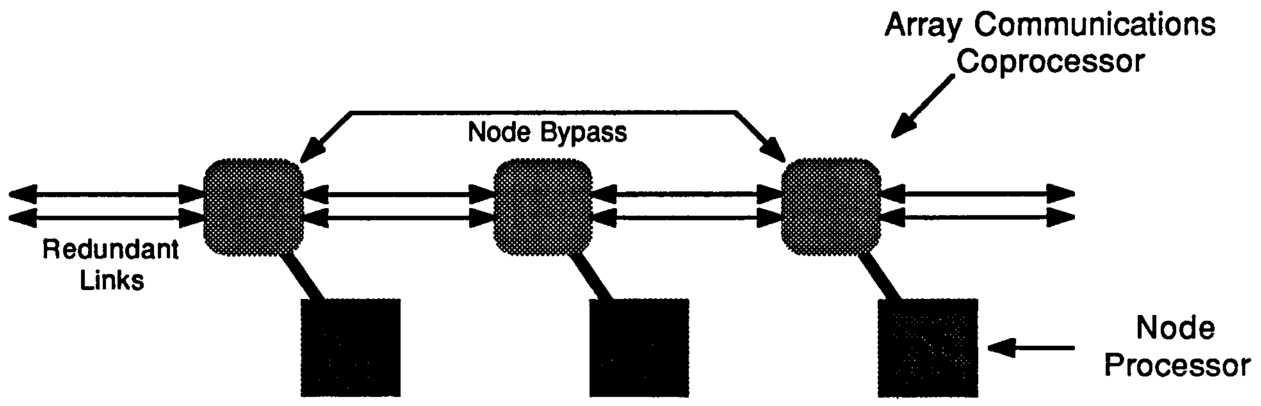


Figure 2-3 Fault-Tolerant Linear Array Topology.

Also, if the algorithm dictates, the node bypass can also be used to rapidly transfer data to a down stream processor. A node bypass link can easily be implemented in other array topologies as well.

2.4 Ring Topology

An example of a processor array configured as a ring is shown in Figure 2-4. This topology is basically a linear array, with the ends connected. This short-circuit can be utilized to transfer data between nodes over the shortest of two paths, instead of one path in a linear array.

This topology also offers a degree of fault tolerance, with a failed node breaking one but not both of the ring communication paths.

2.5 FFT Specific Topology

While 2-D, 3-D, and linear array topologies can be utilized to compute almost any conceivable algorithm, they generally are not optimized for a specific algorithm's communications requirements. Algorithm specific topologies can be developed to minimize inter-node communication requirements, i.e. each node only communicates with nodes that are required for data interchange and no data routing is required by a node processor.

An array topology optimized for a Radix-2, Radix-4, or Split-Radix Fast Fourier Transform (FFT) is shown in Figure 2-5. In this array data flows from left to right in the array, with each processor executing half of the operations required at each stage of the FFT. (The topology shown represents a length-32 transform which may be conveniently broken into 5 stages. A complete analysis is presented in Section 3.2.1.)

Each stage of the length-32 FFT involves the computation of 16 "Butterflies." Therefore, a highly parallel FFT optimized length-32 systolic array could have up to 16 processors per stage, with 5 stages for a total of 80 processors. Each processor would have to communicate with each of the 16 processors in the succeeding stage, i.e. have 16 serial communications links.

A more optimum design might employ 2 (4) processors at each stage, executing 8 (4) Butterflies and communicating with 2 (4) processors in adjacent stages, respectively.

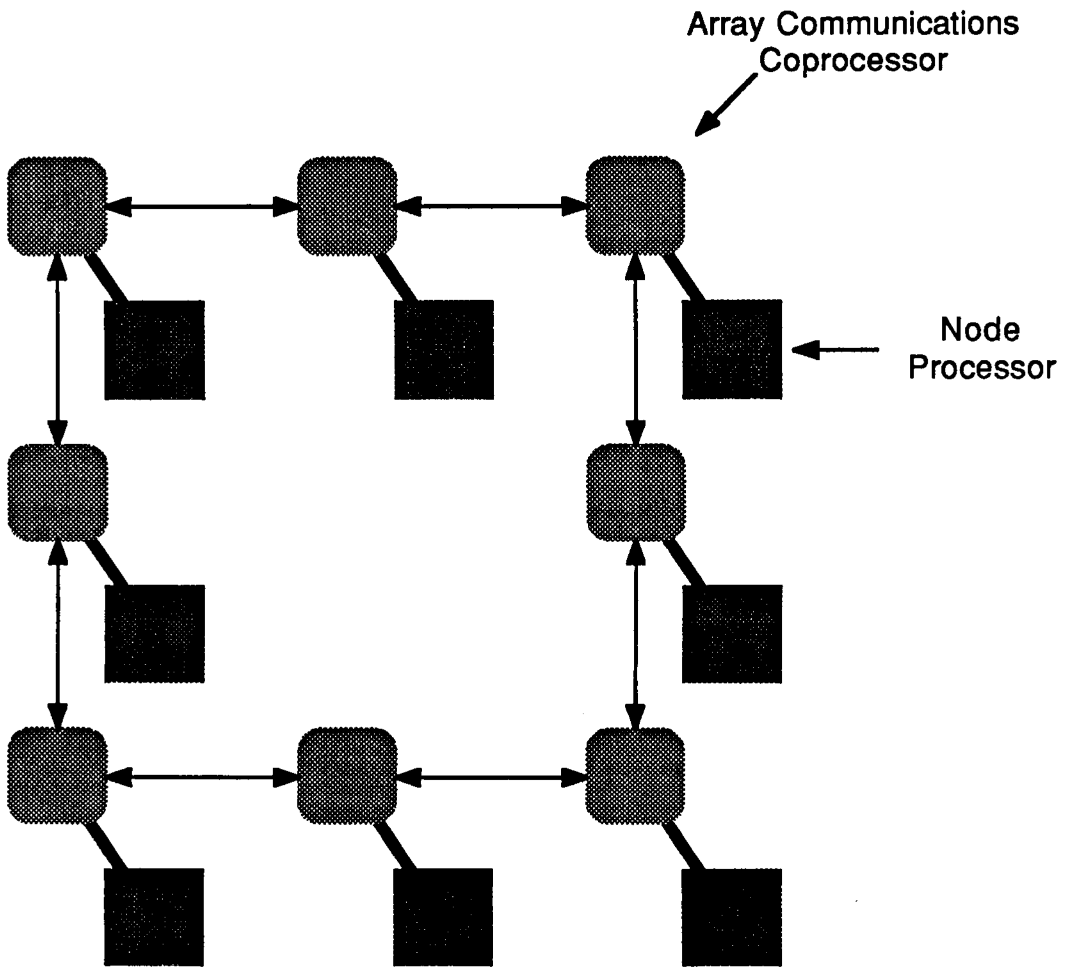


Figure 2-4 Ring Array Topology.

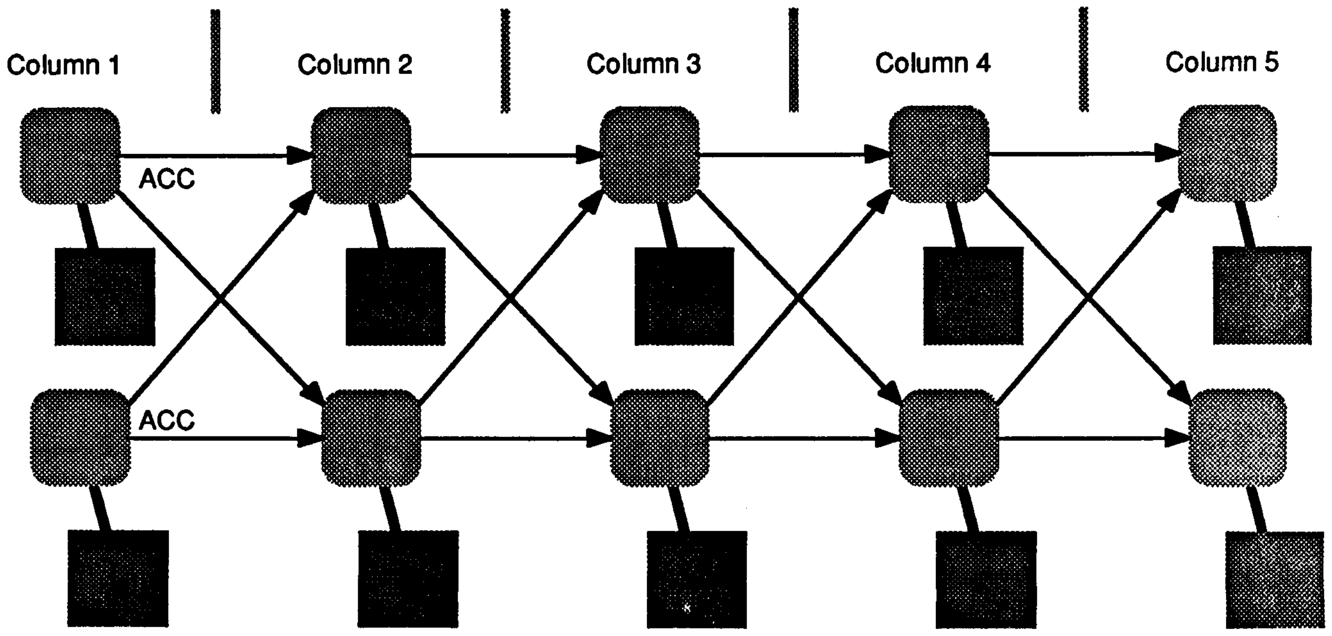


Figure 2-5 Fast Fourier Transform Array Topology.

The proposed design will have 8 full-duplex communication links, with expansion capability to 32 links. Thus, up to eight GaAs processors can be utilized at each stage of a highly parallel FFT. (The total number of stages will be dependent on the length of the transform.)

Other DFT specific array topologies will be discussed in detail in Section 3.2.

2.6 Dynamic Reconfiguration and Data Routing

Dynamic reconfiguration of an array topology can be easily accomplished by severing existing communication links between nodes and reconnecting the links in some other fashion with a digital switch.

As an alternative, data routing can be performed by an intelligent communication coprocessor. Data can be packetized and routed between processor nodes by the coprocessors. Each data packet includes a unique destination address along with the data. As the data is passed from coprocessor to coprocessor, each coprocessor examines the data packet address, determines the optimum path out of the coprocessor to the destination, and routes the packet along that path. This process continues until the data packet reaches the destination node.

This sophisticated routing methodology is very powerful, but has some draw backs. For example, communication links can become overloaded causing delays in transmission, and the coprocessor must be much more intelligent (and therefore more highly integrated) to process the packetized data.

3.0 Systolic Array Algorithms

3.1 Overview

A number of signal processing algorithms have been examined for suitability for adaptation to high speed systolic array processors, including the Fourier Transform, the Kalman filter, digital filters, and image processing filters and related algorithms.

Generally, there are many computational approaches to each algorithm, with each technique exhibiting selected advantages in terms of:

- Memory Requirements
- Relative Number of Multiplication/Division/Addition Operations
- Number of High Level Functions, such as Sine and Cosine
- Numerical Accuracy
- Adaptability to Parallel Processing Systems

3.2 Fourier Transform Algorithms

Since the development of the original Fast Fourier Transform by Cooley and Tukey[1], numerous techniques have been developed to compute the Fourier Transform, including the basic Radix-2 algorithms,[2-7] the Winograd (WFTA) algorithm,[8,9] the prime factor algorithm (PFA),[10,11] the Real-factor FFT,[12] and others.[13-15] The Radix-2 and Radix-4 algorithms are probably the most widely utilized transforms, and for most applications the most practical. The simple "Butterfly" structure of the algorithms allows the transforms to be done "in place", with intermediate results overwritten at the end of each successive stage to provide maximum memory efficiency, and the regular structure of the transforms allows a relatively simple software implementation. While the WFTA, PFA, and Real-factor FFT are more computationally efficient, requiring fewer total multiplications and additions, they are not as numerically well conditioned as the Radix-2 and Radix-4 algorithms.

The primary objective of most Fast Fourier Transform (FFT) algorithms is to reduce the total number of mathematical operations (or replace complex mathematical operations, e.g. multiplication, with less complex operations, e.g. addition) required to compute the transform. However, reducing the total number of mathematical operations may be less important than other considerations, such as: the flexibility in the values that the transform length, N, may assume; the numerical accuracy required for the particular implementation; and the possibility of doing the transform in place.

Some transform implementations are specifically designed to facilitate high speed parallel and pipelined hardware implementations, often at the expense of additional intermediate memory requirements or an increased number of mathematical operations.

The basic Fourier Transform of the data sequence x(n) is defined by

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi}{N}nk} \qquad 1$$

where x(n) represents an input data sequence, indexed by n, and X(k) represents the output data sequence, indexed by k. The indices n and k are commonly referred to as the time index and the frequency index, respectively. The time sequence interval, Δt, is assumed to be constant. The length of the data array to be transformed is denoted by the variable N. The transform equation is customarily written in a more abbreviated form

$$X(k) = \sum_{n=0}^{N-1} x(n)\, C_N^{nk} \qquad 2$$

where the coefficient C represents the complex exponential shown in Eqn. (1).

A decrease in the computational complexity can be realized by changing the indexing in Eqn. (2) to take advantage of symmetry in the evaluation of the complex coefficient.

3.2.1 Cooley-Tukey Radix-2 and Split-Radix FFT

Let the index n be represented by n = 2p + q, where q = 0, 1 and p = 0, 1, 2, ..., (N/2 - 1). Substituting into Eqn. (2), we find

$$X(k) = \sum_{q=0}^{1} \sum_{p=0}^{N/2-1} x(2p+q)\, C_N^{(2p+q)k} \qquad 3$$

or

$$X(k) = \sum_{q=0}^{1} C_N^{qk} \sum_{p=0}^{N/2-1} x(2p+q)\, C_{N/2}^{pk} \qquad 4$$

where the sum has been decomposed by a Radix-2 transform of the index n.

A similar procedure can be used to perform a Radix-4 decomposition of the index p. Let the index p be represented by p = 4s + t, where t = 0, 1, 2, 3 and s = 0, 1, 2, ..., (N/8 - 1). Substituting into Eqn. (4), we now find

$$X(k) = \sum_{q=0}^{1} C_N^{qk} \sum_{t=0}^{3} \sum_{s=0}^{N/8-1} x(2(4s+t)+q)\, C_{N/2}^{(4s+t)k} \qquad 5$$

or

$$X(k) = \sum_{q=0}^{1} C_N^{qk} \sum_{t=0}^{3} C_{N/2}^{tk} \sum_{s=0}^{N/8-1} x(8s+2t+q)\, C_{N/8}^{sk} \qquad 6$$

where the number of complex multiplications is given by N • (N/8 • 4 • 2) = N^2, the same as in Eqn. (2).

Now consider the case N = 32, where X(k) is given by

$$X(k) = \sum_{q=0}^{1} C_{32}^{qk} \sum_{t=0}^{3} C_{16}^{tk} \sum_{s=0}^{3} x(8s+2t+q)\, C_4^{sk} \qquad 7$$

and

$$C_4^{sk} = e^{-j\frac{2\pi}{4}sk} \qquad 8$$

The complex coefficient, C, can only assume four values, ± 1 or ± j.

Multiplication by any of these four values can be readily implemented in software or hardware without doing a full complex multiplication (4 real multiplications, 2 real additions), because multiplication by ± 1 is really only a change of sign at most and multiplications by ± j is only a change of sign and a swap. Both are elementary Butterflies.

Thus, the arithmetic complexity of the innermost loop of Eqn. (7) has been substantially reduced. The number of full complex multiplications has been reduced by a factor of four, a significant savings.

Similar Radix-2, Radix-4, Radix-8, or other higher radix decompositions can be used on the length-N transforms to achieve similar savings in arithmetic complexity.

Recently, a "Split-Radix" FFT has been reported[16-18] which is a combination of Radix-2 and Radix-4 decompositions. It has been shown that the number of multiplications and additions required for the Split-Radix FFT is less than the number required for either a Radix-2 or a Radix-4 transform. While the software implementation of the Split-Radix FFT is somewhat more complex than the Radix-2 implementation, it is amenable to the same arithmetic reduction programming techniques.

The Split-Radix FFT is based on the following decompositions of the even and odd terms of Eqn. (1):

$$X(2k) = \sum_{n=0}^{N/2-1} (x(n) + x(n + \tfrac{N}{2}))\, C_N^{2nk}$$

$$X(4k+1) = \sum_{n=0}^{N/4-1} [(x(n) - x(n + \tfrac{N}{2}) - j\,(x(n + \tfrac{N}{4}) - x(n + 3\tfrac{N}{4}))]\, C_N^{n}\, C_N^{4nk} \qquad 9$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} [(x(n) - x(n + \tfrac{N}{2}) + j\,(x(n + \tfrac{N}{4}) - x(n + 3\tfrac{N}{4}))]\, C_N^{3n}\, C_N^{4nk}$$

This decomposition relates a length-N DFT to one length-N/2 and two length-N/4 DFTs with twiddle factors[19]. Repetition of this decomposition process, for the N/2 and N/4 DFTs, generates the Split-Radix FFT in much the same way as the decimation-in-frequency Radix-2 Cooley-Tukey FFT is derived. The last stage of decomposition is in fact a Radix-2 decomposition.

Fast transforms, such as the Split-Radix FFT, are designed to minimize the total number of complex multiplications and additions. In a single processor implementation, this is important because all of the complex multiplications (4 real multiplications, 2 real additions) and complex additions (2 real additions) must be performed sequentially. Various coding techniques can also be utilized to reduce the computational complexity. For example, multiplications by j can be implemented in software (or hardware) as a swap of the real and complex parts with a change of sign of the resulting real part.

A diagram of a length-32 Split-Radix FFT implementation is shown in Figure 3-1. A generalization of the mathematical operation shown at each stage of the transform is given in Figure 3-2.

Examination of Figure 3-1 reveals that the Split-Radix FFT algorithm (like other Radix-N decompositions) is done in-place, with no additional intermediate storage memory required. For example, at the end of the first stage the contents of the memories containing the complex quantities x(0) and x(16) are replaced by the complex quantities (x(0) + x(16)) and (x(0) - x(16)), respectively. The only temporary storage required is within the computational element (see Figure 3-2) during the complex addition and multiplication process. The outputs from each computational element become new elements in the modified data array produced at the end of each stage. This in-place processing capability can be exploited to produce a highly parallel implementation architecture.

In a typical single processor Von Neumann implementation, the computer would be programmed to compute each of the Butterflies in stage 1, proceed to stage 2 and compute all the Butterflies, and continue this process until all of the Butterflies in the last stage, stage 5, are computed.

A transform of length $N = 2^m$ results in m stages, with N/2 Butterflies per stage. The resulting transformed data is in "bit-reversed" order and must be reordered after the last transform stage.

A hardware implementation of a Butterfly is shown schematically in Figure 3-3. Two complex inputs, A and B, are bussed into the processor, the sum computed and multiplied by the complex coefficient U, and the difference computed and multiplied by the complex coefficient D. The resulting complex products are subsequently bussed out of the processor. The complex coefficients U and D are static, and once loaded into the holding registers remain fixed.

Implementing each of the Butterflies shown in Figure 3-1 with a single processor results in a highly parallel, pipelined DFT processor. The outputs from the 16 processors in stage 1 are hardwired into the appropriate inputs of the stage 2 processors, with the outputs of the stage 2 processors hardwired into the inputs of the stage 3 processors. This data flow continues to the end of stage 5. Bit reversal of the the output data is accomplished by hardwiring the outputs of the stage 5 processors into the proper memory address.

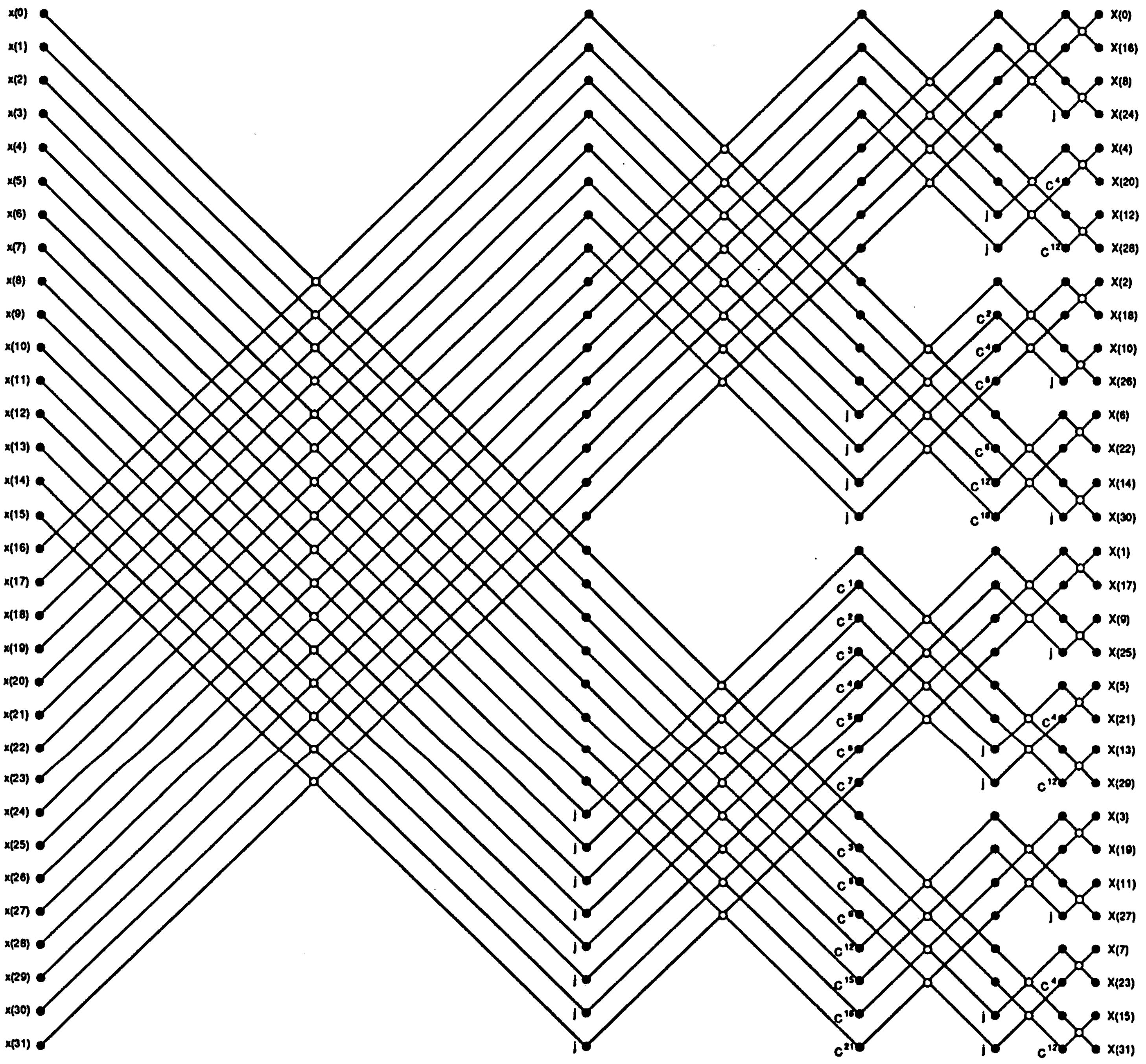


Figure 3-1 Length-32 Split-Radix DFT Algorithm Representation.

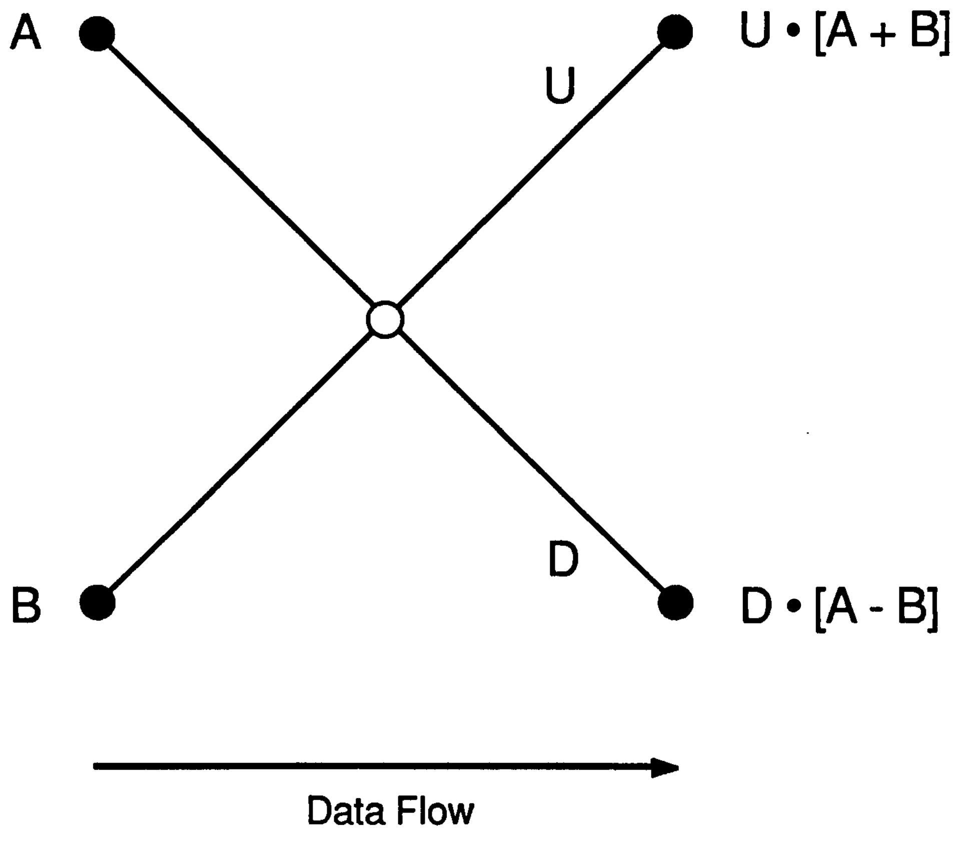


Figure 3-2 Generalization of Split-Radix DFT Node Arithmetic Operations.

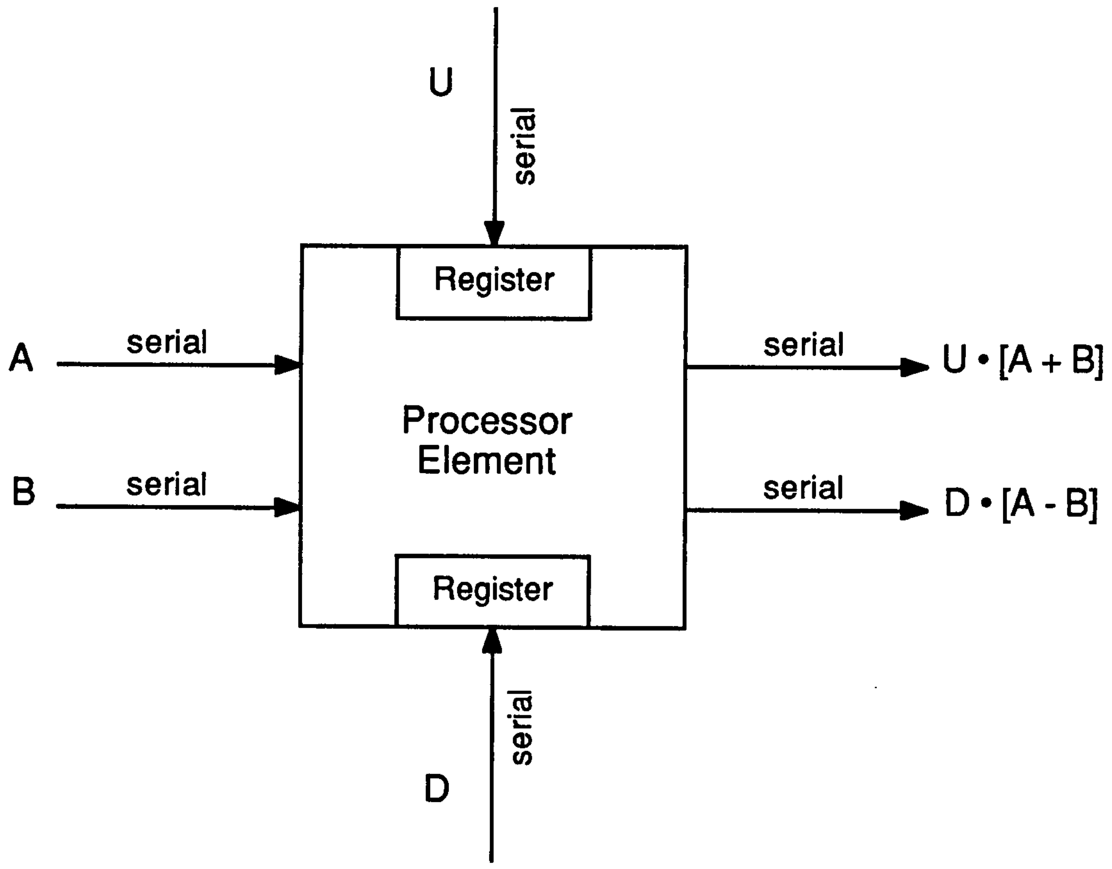


Figure 3-3 Single Processor Implementation of a DFT Butterfly.

Power	DFT Length	1 Butterfly/Node			16 Butterflies/Node		
		Nodes	Inputs	Outputs	Nodes	Inputs	Outputs
2	4	4	2	2	-	-	-
3	8	12	2	2	-	-	-
4	16	32	2	2	2	32	32
5	32	80	2	2	5	32	32
6	64	192	2	2	12	32	32
7	128	448	2	2	28	32	32
8	256	1024	2	2	64	32	32
9	512	2304	2	2	144	32	32
10	1024	5120	2	2	320	32	32
11	2048	11264	2	2	704	32	32
12	4096	24576	2	2	1536	32	32
13	8192	53248	2	2	3328	32	32
14	16384	114688	2	2	7168	32	32
15	32768	245760	2	2	15360	32	32
16	65536	524288	2	2	32768	32	32

Table 3-1 Single Stage DFT Processor Requirements.

Table 3-1 lists the number of processors required to compute DFTs of various lengths. For example, a length-1024 DFT, one Butterfly per node, array would require 5120 processors. Likewise a length-4096 DFT array would require 24576 processors, a prohibitive number of integrated circuits in both cases.

Therefore, either each processor must compute more Butterflies or each integrated circuit must include a larger number of processors to be practical. If each integrated circuit contains 16 processors, each capable of computing a Butterfly, or each processor sequentially computes 16 Butterflies, then a length-1024 DFT would require 320 integrated circuits, each with 32 inputs and 32 outputs. A length-4096 DFT would require 1536 integrated circuits, with the same number of inputs and outputs.

Another alternative to the single node, single processor approach is shown in Figure 3-4. A length-32 transform is implemented with a single set of 16 processors configured to compute all 5 stages of the transform. The inputs A_1 and B_1 are the inputs used by the processors in stage 1 of the transform with corresponding coefficients and outputs, A_2 and B_2 are used as inputs for the stage 2 calculation with corresponding coefficients and outputs. This recirculation continues through all 5 stages. The only increase in complexity in the processor

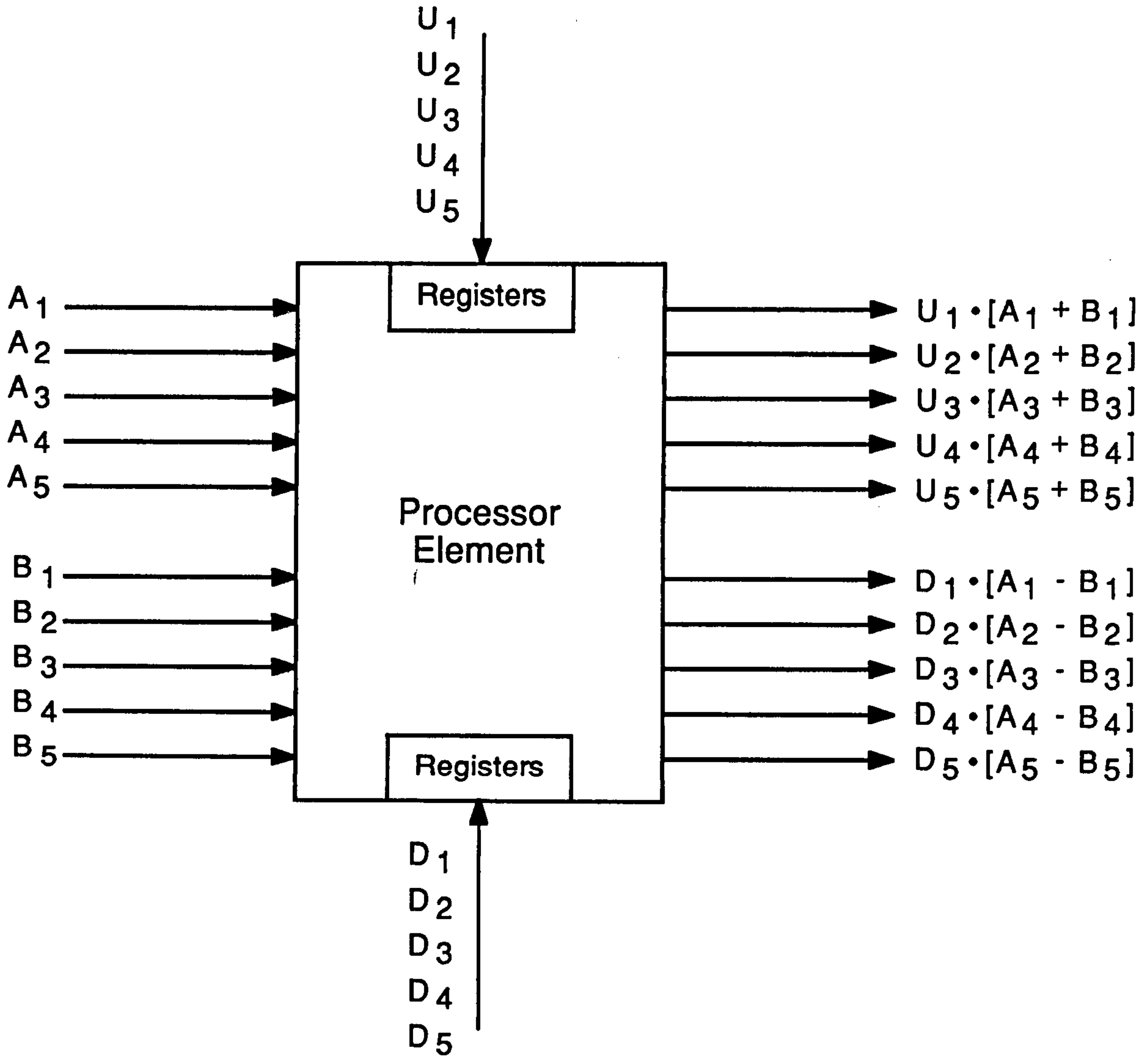


Figure 3-4 Multistage Processor Implementation of a DFT Butterfly.

Power	DFT Length	1 Butterfly/Node			2 Butterflies/Node		
		Nodes	Inputs	Outputs	Nodes	Inputs	Outputs
2	4	2	4	4	1	8	8
3	8	4	6	6	2	12	12
4	16	8	8	8	4	16	16
5	32	16	10	10	8	20	20
6	64	32	12	12	16	24	24
7	128	64	14	14	32	28	28
8	256	128	16	16	**64**	**32**	**32**
9	512	256	18	18	128	36	36
10	1024	512	20	20	256	40	40
11	2048	1024	22	22	512	44	44
12	4096	2048	24	24	1024	48	48
13	8192	4096	26	26	2048	52	52
14	16384	8192	28	28	4096	56	56
15	32768	16384	30	30	8192	60	60
16	65536	**32768**	**32**	**32**	16384	64	64

Table 3-2 Multiple Stage DFT Processor Requirements.

is a clock input to toggle between the multiple input and output connections and the corresponding complex coefficients, and additional holding registers to accommodate the complex coefficients. The internal computational elements of the processor are identical.

While the number of processors required is reduced by a factor of m for a length-2^m transform, the time required to compute a complete transform is now equal to the computation time per stage, T_s, times the number of stages, or $T_{total} = T_s \bullet m$.

Table 3-2 lists the number of processors required to compute various length-N transforms using this approach. For example, a length-1024 DFT requires 512 processors, with 20 inputs and 20 outputs. A length-4096 DFT requires 2048 processors, each with 24 inputs and 24 outputs. If however, each processor were to do two Butterflies instead of one, then the number of processors required for a length-1024 DFT is reduced to 256, with each processor requiring 32 inputs and outputs instead of 16.

These two highly parallel architectures represent a brute force approach to computing a Fourier transform. These approaches have the advantage of not requiring any intermediate storage memory and can conceivably compute, on average, an arbitrary length FFT in as little as one Butterfly computation period. However, an extremely large number of processors are required to implement this DFT specific systolic array architecture.

3.2.2 Block DFT

A decomposition of both the time and frequency indices, with a radix equal to the square root of the transform length, results in a transform comprised of two equal length transforms. This type of decomposition is restricted to transforms of length-4^m ($m = 1, 2, 3, ...$). The time index is decomposed by mapping the index into $n = a \bullet M + b$, where M is the kernel radix ($M = N^{1/2} = 2^m$) and a,b are the new time indices. The frequency index is mapped into $k = A + M \bullet B$ (the change in nomenclature interchanges the rows and columns of the transformed matrix).

For example, a length-4^2 transform, with a Radix-2^2 kernel, results in the time index decomposition $n = 4a + b$, with the indices a,b given by $a = 0,1,2,3$ and $b = 0,1,2,3$, with b incrementing faster. A similar decomposition for the frequency index k results in the mapping $k = A + 4B$, with the indices A,B taking on the values $A = 0,1,2,3$ and $B = 0,1,2,3$.

The square array x(a,b) maps into the sequential data array x(n) as shown:

$$x(a,b) = \begin{bmatrix} x(0,0) & x(0,1) & x(0,2) & x(0,3) \\ x(1,0) & x(1,1) & x(1,2) & x(1,3) \\ x(2,0) & x(2,1) & x(2,2) & x(2,3) \\ x(3,0) & x(3,1) & x(3,2) & x(3,3) \end{bmatrix} = \begin{bmatrix} x(0) & x(1) & x(2) & x(3) \\ x(4) & x(5) & x(6) & x(7) \\ x(8) & x(9) & x(10) & x(11) \\ x(12) & x(13) & x(14) & x(15) \end{bmatrix} \qquad 10$$

Multiplying the indices n and k, we find

$$nk = (4a + b)(A + 4B) = 4aA + bA + 4bB + 16aB \qquad 11$$

which results in the complex multiplier

$$C_{16}^{nk} = e^{-j\frac{2\pi}{16}nk} = e^{-j\frac{2\pi}{4}aA}\, e^{-j\frac{2\pi}{16}bA}\, e^{-j\frac{2\pi}{4}bB} = C_4^{aA}\, C_{16}^{bA}\, C_4^{bB} \qquad 12$$

Substituting Eqn. (12) into Eqn. (2) results in the governing transform equation

$$X(A,B) = \sum_{b=0}^{3} C_{16}^{Ab} C_4^{bB} \sum_{a=0}^{3} W_{16}^{ab} x(a,b) C_4^{aA} = \sum_{b=0}^{3} C_{16}^{Ab} X(A,b) C_4^{bB} \qquad 13$$

where the intermediate transform, X(A,b), is given by

$$X(A,b) = \sum_{a=0}^{3} W_{16}^{ab} x(a,b) C_4^{aA} \qquad 14$$

Note that Eqns. (13) and (14) are of identical structure, and the coefficients C_4 are identical. In order to condition the input data, we have introduced a sequence of "window" coefficients,[20,21] denoted by W. Introduction of the window coefficients into the equation results in this structural symmetry, which is exploited in the hardware architecture to implement a two stage transform with identical processor elements.

The transform can be implemented in two successive, identical stages. For a length-16 transform, each stage requires two 16-word memories, two 4-word memories, and a length-4 DFT processor capable of performing a complex multiply operation followed by a complex multiply accumulate operation.

The two stage transform defined by Eqn. (14) can be readily adapted to a multiple processor configuration. Processors can be configured to compute rows of the intermediate transform array X(A,b) in parallel. After a row of the intermediate transform array is computed, a second set of processors can utilize the data to compute either a row or column of the transformed array X(A,B) in parallel, depending upon the implementation. An array processor implementation of a length-16 block DFT is shown in Figure 3-5.

Parallelism in the first stage is achieved by dedicating a processor element to evaluation of X(A,b) for each value of the index b (requiring four processor elements for a length-4 transform). The summation over a, for each value of b, is shown in Table 3-3. Each dedicated processor operates on a single column of the x(a,b) data array, along with the corresponding window coefficients. The W•x products are computed only once and stored in memory. After the W•x products are computed, the processors cycle through the C_4 coefficients multiplying by the W•x products and accumulating the sum. The index into the C_4 array is determined by the value of A.

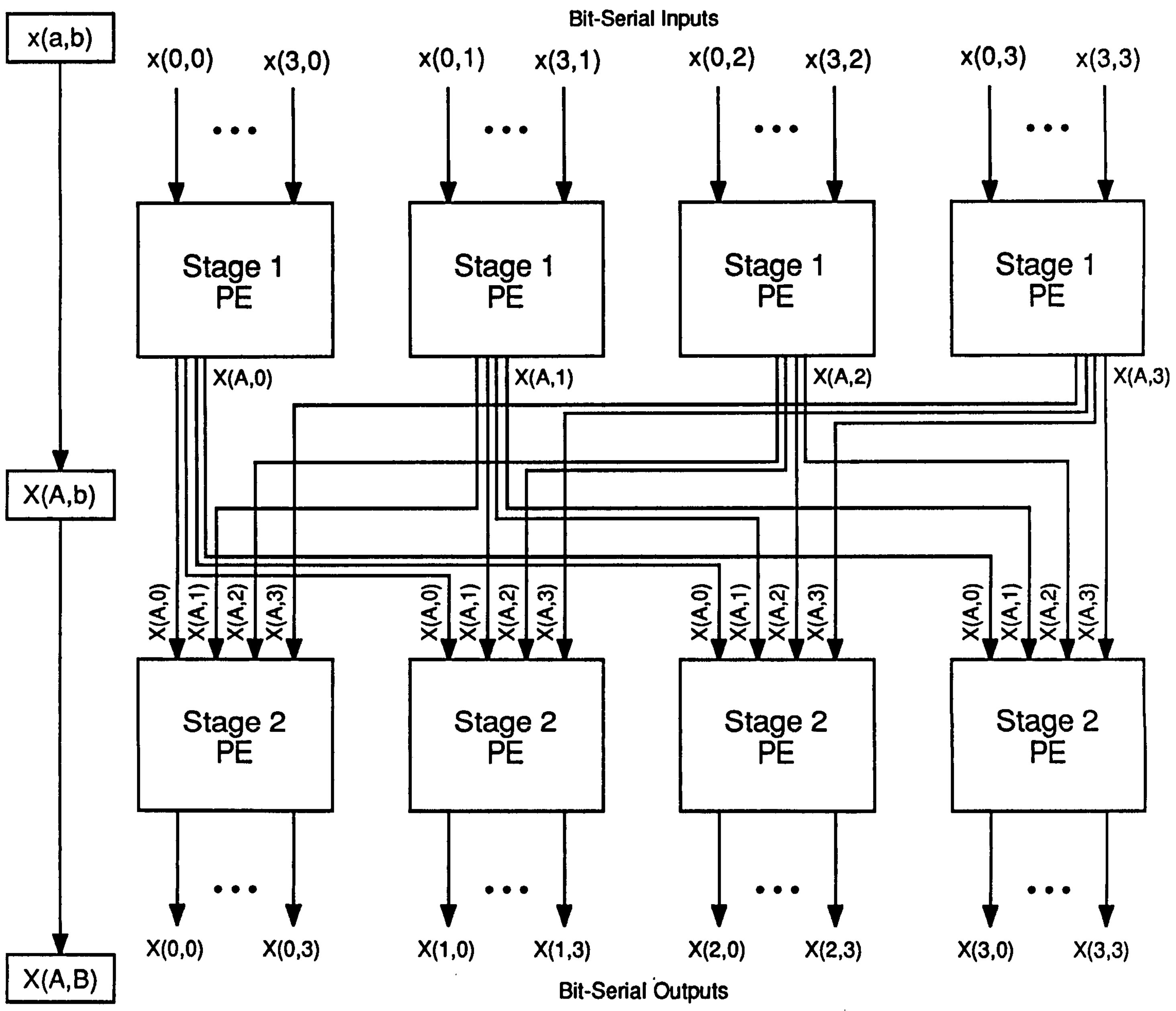


Figure 3-5 Length-16 Block DFT Array Processor.

	PE 1 (b = 0)	PE 2 (b = 1)	PE 3 (b = 2)	PE 4 (b = 3)	
	$W_{16}^{00}\,x(0,0)\,C_4^{0A}$	$W_{16}^{01}\,x(0,1)\,C_4^{0A}$	$W_{16}^{02}\,x(0,2)\,C_4^{0A}$	$W_{16}^{03}\,x(0,3)\,C_4^{0A}$	a = 0
	$W_{16}^{10}\,x(1,0)\,C_4^{1A}$	$W_{16}^{11}\,x(1,1)\,C_4^{1A}$	$W_{16}^{12}\,x(1,2)\,C_4^{1A}$	$W_{16}^{13}\,x(1,3)\,C_4^{1A}$	a = 1
	$W_{16}^{20}\,x(2,0)\,C_4^{2A}$	$W_{16}^{21}\,x(2,1)\,C_4^{2A}$	$W_{16}^{22}\,x(2,2)\,C_4^{2A}$	$W_{16}^{23}\,x(2,3)\,C_4^{2A}$	a = 2
	$W_{16}^{30}\,x(3,0)\,C_4^{3A}$	$W_{16}^{31}\,x(3,1)\,C_4^{3A}$	$W_{16}^{32}\,x(3,2)\,C_4^{3A}$	$W_{16}^{33}\,x(3,3)\,C_4^{3A}$	a = 3
SUM	= X(A,0)	= X(A,1)	= X(A,2)	= X(A,3)	

Table 3-3 Parallel Processor Evaluation of the Transform X(A,b).

A hardware implementation of the algorithm discussed above is shown in Figure 3-6 for the case b = 0. A column of the raw signal data is loaded into the input registers in a load operation. (This loading procedure, and the order in which the data becomes available, is application dependent.) After the data is loaded, the raw signal data is clocked out of the input registers and multiplied by a window coefficient. A set of window coefficients is downloaded into each processor prior to operation, with each processor containing one column of the window coefficient array. Thus, the memory required to store the window coefficients is evenly distributed over all of the processor elements.

The W•x products are utilized by the second stage of the processor to compute the sum shown in Table 3-3. The C_4 registers are common to all of the processors. A sum is computed for each value of A, using A as an index into the C_4 registers. For a length-4 transform each X(A,0) requires 4 multiply and accumulate cycles. The result is bussed out of the processor element serially to the appropriate stage two processors.

The memory required to hold the fixed constants is 8 complex words (2 sets of constants • 4 complex words/set) for a length-16 transform . For a length-1024 transform the total memory required per processor is 64 complex words (2 sets of constants • 32 complex words/set).

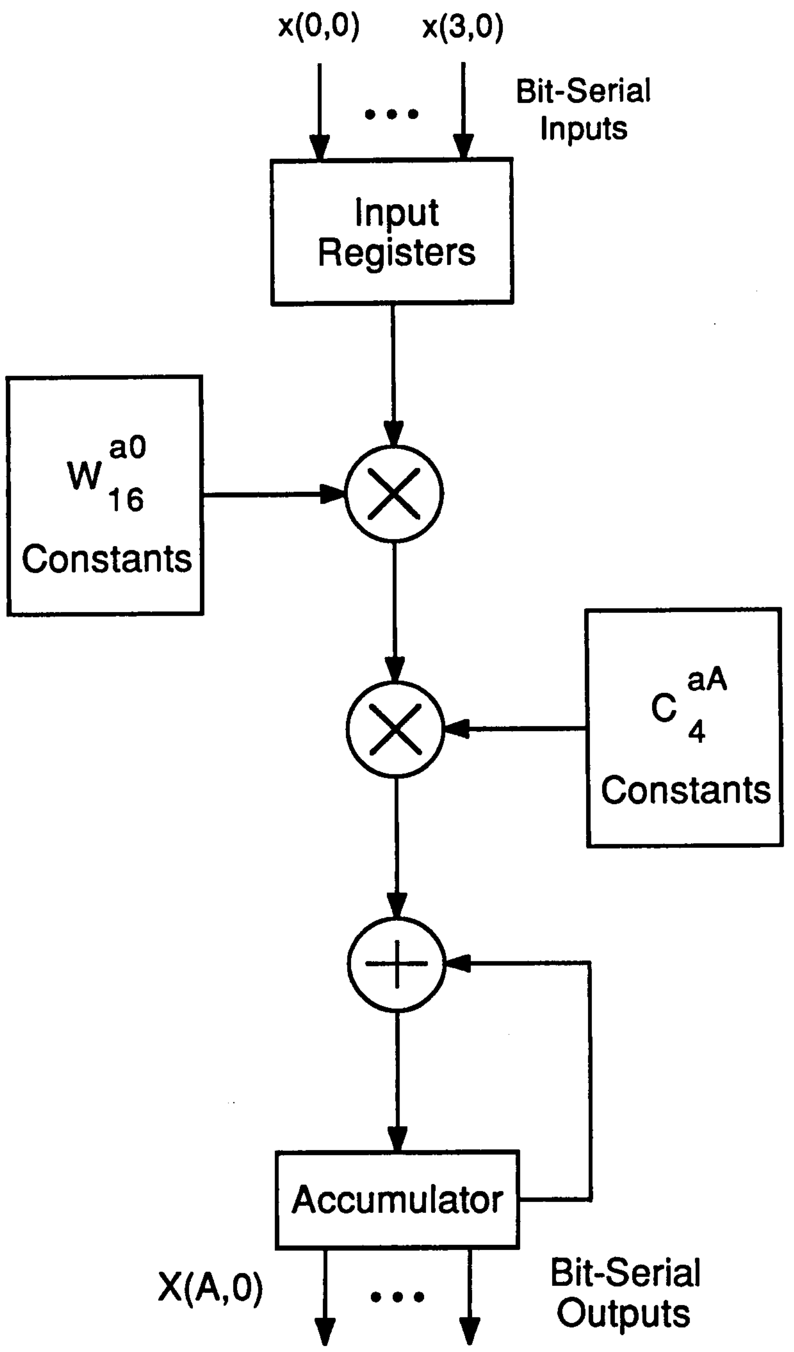


Figure 3-6 Stage 1 Computation of a Length-16 Block DFT.

	PE 1 (A = 0)	PE 2 (A = 1)	PE 3 (A = 2)	PE 4 (A = 3)	
	$C_{16}^{00} X(0,0) C_4^{0B}$	$C_{16}^{10} X(1,0) C_4^{0B}$	$C_{16}^{20} X(2,0) C_4^{0B}$	$C_{16}^{30} X(3,0) C_4^{0B}$	b = 0
	$C_{16}^{01} X(0,1) C_4^{1B}$	$C_{16}^{11} X(1,1) C_4^{1B}$	$C_{16}^{21} X(2,1) C_4^{1B}$	$C_{16}^{31} X(3,1) C_4^{1B}$	b = 1
	$C_{16}^{02} X(0,2) C_4^{2B}$	$C_{16}^{12} X(1,2) C_4^{2B}$	$C_{16}^{22} X(2,2) C_4^{2B}$	$C_{16}^{32} X(3,2) C_4^{2B}$	b = 2
	$C_{16}^{03} X(0,3) C_4^{3B}$	$C_{16}^{13} X(1,3) C_4^{3B}$	$C_{16}^{23} X(2,3) C_4^{3B}$	$C_{16}^{33} X(3,3) C_4^{3B}$	b = 3
SUM	= X(0,B)	= X(1,B)	= X(2,B)	= X(3,B)	

Table 3-4 Parallel Processor Evaluation of the Transform X(A,B).

An identical set of processor elements can be utilized to compute the stage-2, length-4 transform, using the intermediate transform, X(A,b), as input data. Now, instead of window coefficients, each processor element is downloaded with the appropriate C_{16} twiddle factors during the initialization procedure. The summation to be carried out (from Eqn. (13)) is shown in Table 3-4.

Each processor element is configured to compute a summation corresponding to a fixed value of the frequency index A. A block diagram describing the processor element configured for the A = 0 summation is shown in Figure 3-7. Serial data is received from the stage one processors in the input registers, and is subsequently multiplied by the appropriate C_{16} twiddle factors.

The C•X product is multiplied by the appropriate C_4 coefficient, indexed by B, and the product accumulated. The result is the X(0,B) term of the Fourier transform. The time and memory required to compute the second length-4 transform is identical to that of the first length-4 transform.

Table 3-5 shows the number of processors required to implement various length-N Block DFTs. As discussed, a length-16 transform requires two four processor stages, with each processor requiring four inputs and four outputs. A length-1024 transform requires two 32 processor stages, with 32 inputs and 32 outputs for each processor.

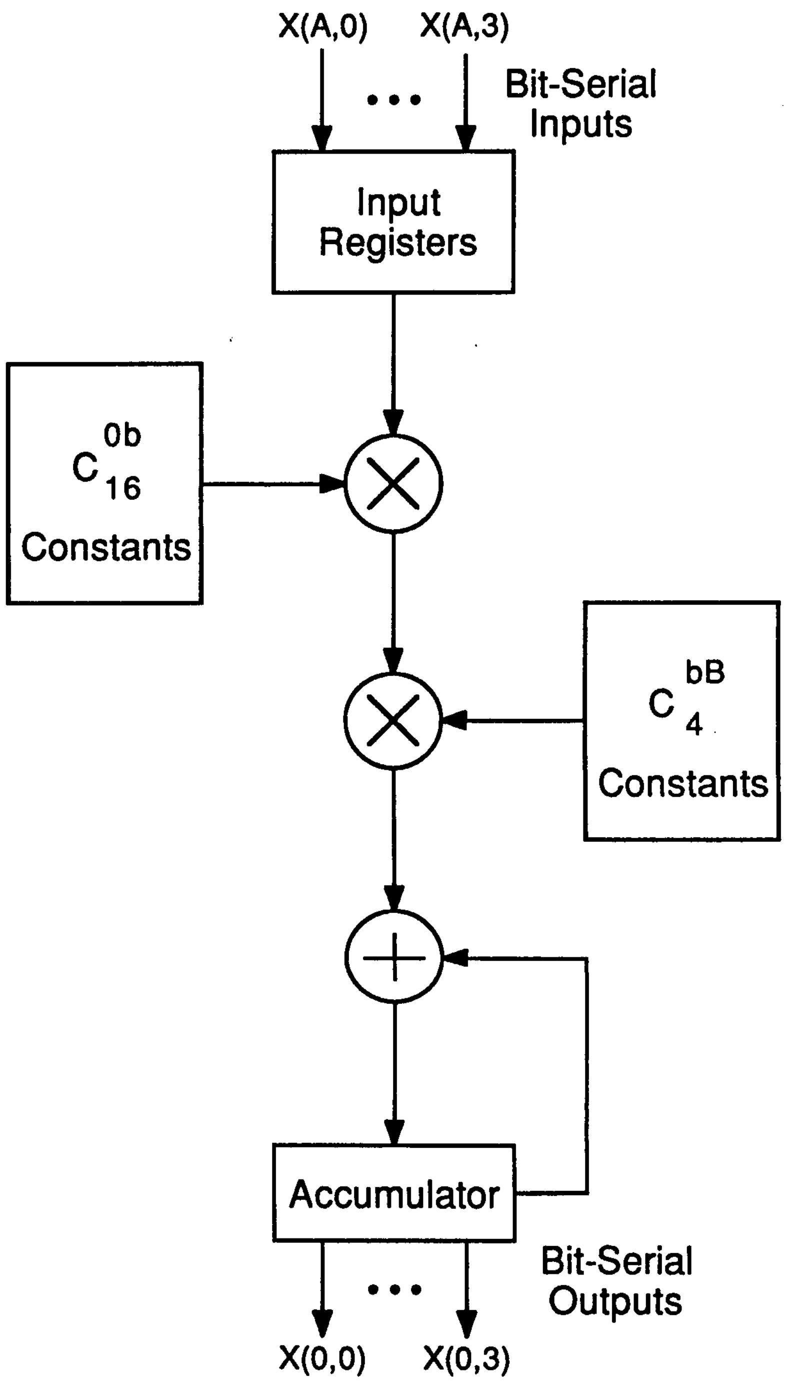


Figure 3-7 Stage 2 Computation of a Length-16 Block DFT.

Power	DFT Length	Processors	Inputs	Outputs
2	4	4	2	2
4	16	8	4	4
6	64	16	8	8
8	256	32	16	16
10	1024	64	32	32
12	4096	128	64	64
14	16384	256	128	128
16	65536	512	256	256

Table 3-5 Block DFT Processor Requirements.

The processor architecture described in Section 4 can accommodate a maximum of 32 input and 32 output ports. The initial implementation will be designed with 8 inputs and 8 outputs, thus the initial GaAs processor will be capable of computing transforms of up to length-64 using this algorithmic approach. A 32-port processor would be capable of handling DFT lengths of up to 1024.

3.3 Other Signal Processing Algorithms

In addition to the Fourier transforms discussed above, other related transforms such as the Fast Hartley transform,[22-31] and the cosine transform,[32,33] are well suited for implementation in a GaAs based systolic array. Two other commonly used signal processing processing algorithms are the Kalman Filter and the Finite Impulse Response Filter.

3.3.1 Kalman Filter

Recent investigations[34] have shown that parallel processing techniques can be utilized to implement the measurement update step of the Kalman filter. SPEC's processor design could be easily used to implement the parallel processor Kalman filter described in reference 34.

3.3.2 Digital Filters

Digital filters, which operate on a stream of data, could be inplemented using a linear array topology such as that shown in Figure 2-3. A simple digital filter used for smoothing simply averages nearest neighbors. This could be readily implemented with the proposed processor. More elaborate image filters, such as a 2-D nearest neighbor matrix filter, could be implemented by connecting each processor node to it's eight nearest coplaner neighbors (9 point matrix filter).

3.4 Review of Signal Processing Algorithms

The DFT algorithms presented are examples of the kinds of techniques that can be utilized to structure DFT algorithms in a manner such that implementation can be achieved in a highly parallel and distributed fashion. Each algorithm is conducive to implementation with very high speed GaAs Bit-Serial processors.

The massively parallel architecture requires little intermediate memory, only temporary registers within the processor elements, and can be configured to compute a length-1024 transform in time T_s , where T_s is the computation time per stage. The latency of the array processor is $T_s \bullet m$, where m is the total number of stages. An obvious drawback to this architecture is the large number of processor elements required to implement a large length-N transform.

The Block DFT implementation requires fewer processor elements, but requires more intermediate storage elements and more computations per processor. This technique is attractive however, because a much more reasonable number of processors is required to compute large length-N transforms. A drawback to this technique is the large number of input and output connections required to interconnect processor nodes.

4.0 GaAs Processor Architecture

4.1 Architecture Overview

SPEC has evaluated two basic approaches to designing a very high speed DFT systolic array processor, each requiring a comparable level of device integration.

The first alternative is to implement very elementary function, but highly integrated processors. Each processor is explicitly designed to compute a Radix-2 Butterfly and cannot be reprogrammed to compute other signal processing algorithms. In Section 4.2 we describe such an element. Previous investigators have designed integrated CMOS bit-serial processors to compute fixed point Butterflies (see Denyer and Renshaw).

The second alternative is to design a high performance reprogrammable processor capable of executing an instruction stream. This is the approach that SPEC has chosen to pursue in the Phase II program. While this approach is not as fast as the massively parallel, dedicated DFT Butterfly processor, it has a much broader range of applications and is a much more attractive commercial product.

Figure 4-1 shows the four basic processor families, Complex Instruction Set Computer (CISC), Reduced Instruction Set Computer (RISC), and Digitial Signal Processor (DSP). An example of a CISC processor is the Motorola 68030 microprocessor, which is a highly integrated device with many thousands of transistors. Likewise highly integrated DSPs, such as the Motorola 96000, and RISC processors, such as the Motorola 88000, are implemented in silicon and require in excess of 100,000 transistors. Clearly, processors of these complexities will not be practical in GaAs for several years.

Sun Microsystems has developed a RISC architecture which is suitable for implementation in GaAs. The current silicon based RISC processor used in Sun Series-4 workstations is implemented in a Fujitsu gate array. The design required approximately 16,000 of the available 20,000 gates in the gate array.

Sun Microsystems has licensed the SPARC architecture to Cypress Semiconductor which is currently producing a very high performance CMOS version of the processor. Sun Microsystems has also licensed the architecture to BIT which has an ECL version of the SPARC processor in development.

SPEC has held discussions with Sun Microsystems concerning licensing of the SPARC architecture for GaAs applications. In these discussions Sun has expressed a strong interest in licensing the technology to SPEC for either embedded applications or commodity chip production.

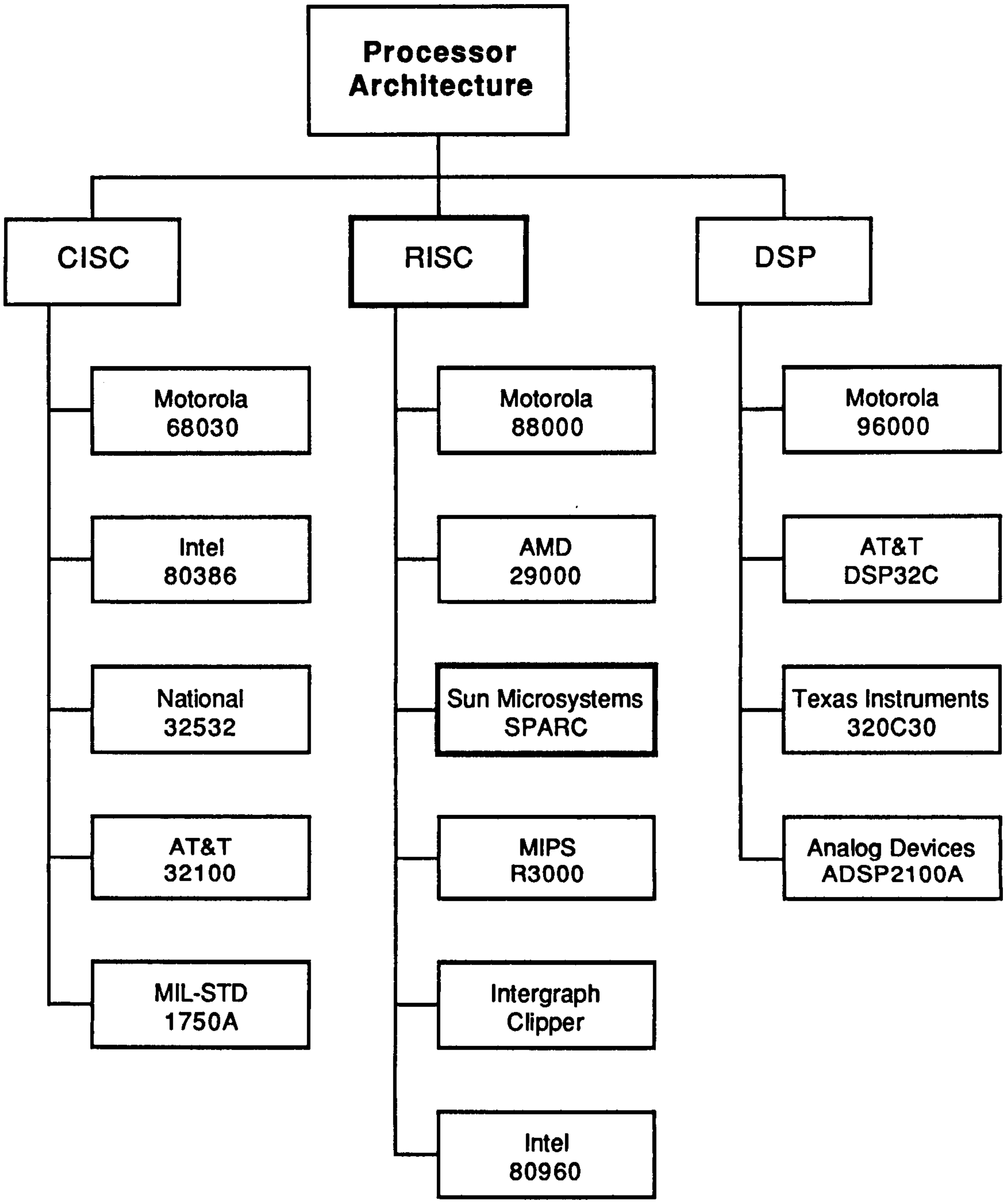


Figure 4-1 Commercial Processor Architectures.

SPEC is very confident that the SPARC RISC architecture can be realized in a GaAs standard cell design. Vitesse and GigaBit Logic currently offer 10,000 gate standard cells and are bringing 20,000 gate standard cells to market in the fourth quarter of 1988. It is virtually certain that in the time frame of the Phase II program standard cells of approximately 30,000 to 40,000 gates will be available.

It is apparent from recent literature[35-47] that numerous hardware design approaches have been investigated for high speed Fourier transforms. To date, no known DFT specific GaAs hardware has been developed and reported in the literature.

General purpose Bit-Serial techniques have been studied at length[48-54], and have been implemented in a number of applications.

4.2 Dedicated Bit-Serial DFT Processor Elements

The Butterfly processor shown in Figure 3-3 can be easily implemented as a Bit-Serial processor. Denyer and Renshaw describe techniques for implementing complete bit-serial processors in CMOS using a specialized silicon compiler. High level macros have been devised to implement lower level arithmetic functions such as add, subtract, multiply, and divide, and low level control functions such as bit delay. The arithmetic functions described are fixed point operations. Other investigators have reported bit-serial implementations of floating point arithmetic operations[55]. Similar implementations are now practical in GaAs.

A complete Radix-2 Butterfly processor is shown in Figure 4-2. A global clock is used to synchronously clock in the complex floating point variables A and B. A complex floating point adder and subtractor are used to compute the complex sum and difference. The difference (A - B) is clocked into a complex multiplier which computes the product of (A - B) • W, and the sum (A + B) is clocked into a delay which synchronizes the two outputs.

In a highly parallel implementation such as the one described in Section 3.2.1, the time required for a synchronous bit-serial system to complete a single transform stage is determined by the computational complexity of the most extensive calculation. Therefore, unless all computations within a single stage can be reduced in complexity, nothing is achieved by reducing the complexity of the remaining calculations. In an asynchronous system, in which each PE initiates processing only after receiving valid A and B inputs from the previous stage, time savings can be realized if the computational complexity is properly spread over computational nodes.

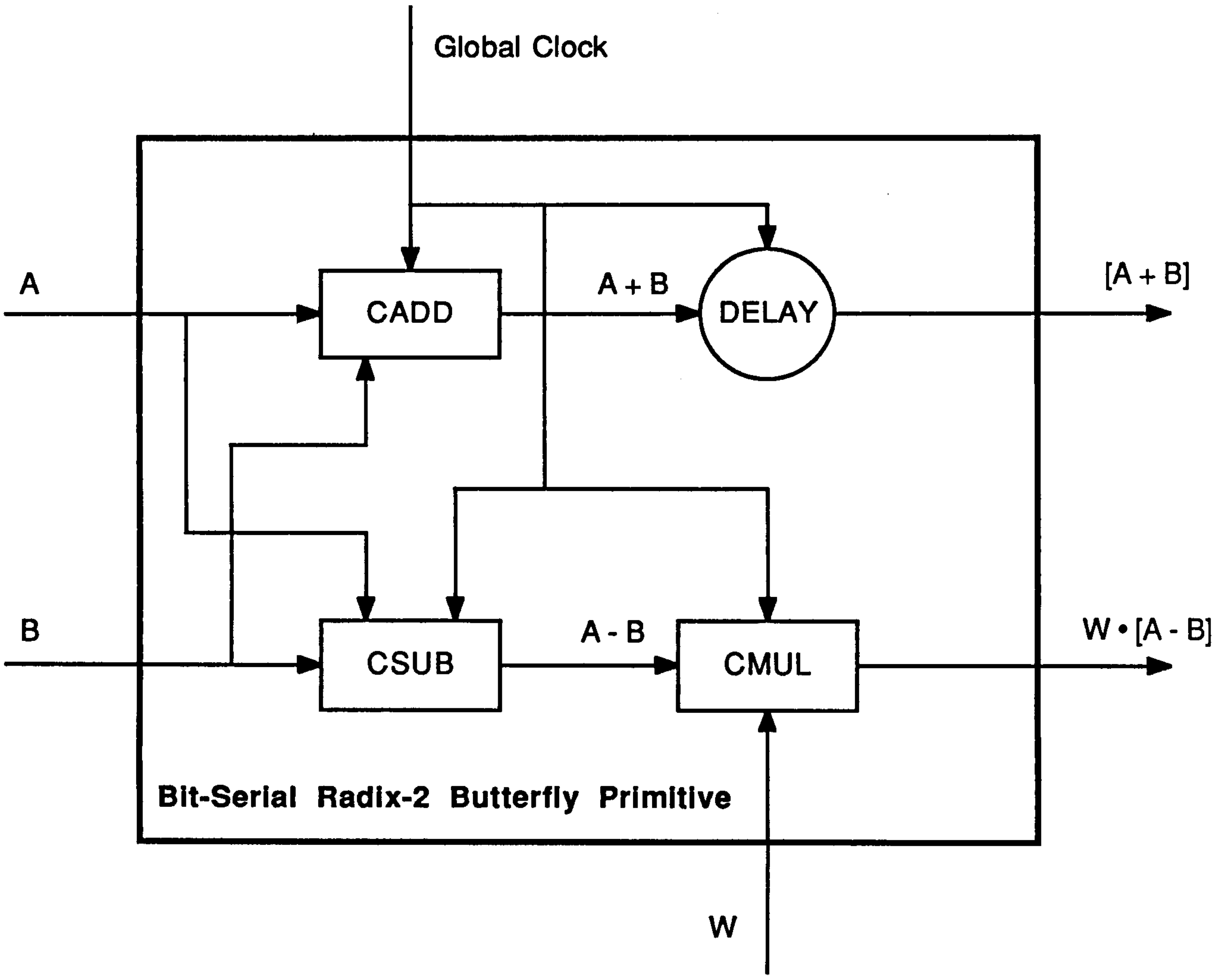


Figure 4-2 Dedicated Bit-Serial Floating Point Butterfly Processor.

Bit-Serial Arithmetic Operation	**Clock Cycles**
32-Bit Floating Point Addition	48
32-Bit Floating Point Subtraction	48
32-Bit Floating Point Multiplication	576
Complex Sequential Operation	
Complex Floating Point Addition	116
Complex Floating Point Subtraction	116
Complex Floating Point Multiplication	2536
Complex Parallel Operation	
Complex Floating Point Addition	58
Complex Floating Point Subtraction	58
Complex Floating Point Multiplication	634

Table 4-1 Floating Point Arithmetic Operation Timing.

The number of clock cycles required to compute a complex floating point Butterfly is equal to the time required to compute a complex addition/subtraction and a complex multiply,

$$C_{BF} = 58 + 634 = 692 \text{ clock cycles}$$

for completely parallel arithmetic operation and,

$$C_{BF} = 116 + 2536 = 2652 \text{ clock cycles}$$

for sequential arithmetic operation.

With a 1 GHz clock rate and completely parallel arithmetic operation, the time required to compute a Radix-2 Butterfly, T_{BF}, is equal to

$$T_{BF} = 692 \text{ cycles} / 1 \text{ GHz} = 692 \text{ nsec}$$

and for a 2 GHz clock

$$T_{BF} = 692 \text{ cycles} / 2 \text{ GHz} = 346 \text{ nsec}.$$

Therefore, the total time required to compute a completely parallel, 10 stage, length-1024 transform (latency) is 6.92 (3.46) μsec at 1 GHz (2 GHz).

These estimates assume that data transfer can be accomplished concurrently with the computation by buffering the serial inputs. The communication data rate, R_{comm}, required to transfer 64 bit complex data between stages is equal to

$$R_{comm} = 64 \text{ bits} / 692 \text{ nsecs} = 92 \text{ Mbits/sec}$$

for 1 GHz operation.

4.3 RISC Signal Processor

After careful review of DFT algorithms and other classes of algorithms, SPEC has reached the following conclusions concerning basic algorithm characteristics and array processor requirements:

- Numerically Intensive Operation
    - Requires Floating Point Coprocessor
- Communications Intensive Operation
    - Requires Communications Coprocessor
- Coefficient/Constant Intensive Operation
    - Requires Large Register Set or Fast Memory for Good Performance
- GaAs Technology - Less than ~ 25,000 Gate Standard Cell
    - RISC Architecture

The optimum solution set is:

- SPARC Processor
    - Floating Point Coprocessor Defined in Architecture
    - Compatible with Communications Coprocessor
    - Very Large Fixed Point and Floating Point Register Sets
    - Implemented in Silicon with a 20,000 Gate Array
    - Architecture Compatible with ECL and GaAs Implementation

4.3.1 SPARC Processor

The SPARC architecture defines an Integer Processor, a Floating Point Coprocessor (FPC), and an interface to a application dependent Coprocessor. The Processor, FPC, and Coprocessor can operate concurrently. Within the FPC separate floating point Addition, Subtraction, Multiplication, and Division Units can also operate concurrently.

A typical array processor node configuration is shown in Figure 4-3. The Processor extracts instructions from the instruction stream and routes applicable instructions to either the FPC or the user defined coprocessor.

The SPARC architecture has approximately 50 integer instructions, which fall into the following basic categories:

- Load and Store Instructions
- Arithmetic, Logical, and Shift Instructions
- Coprocessor Instructions
- Control & Transfer Instructions (Jumps, Calls, Traps, etc.)
- Read & Write Control Register Instructions

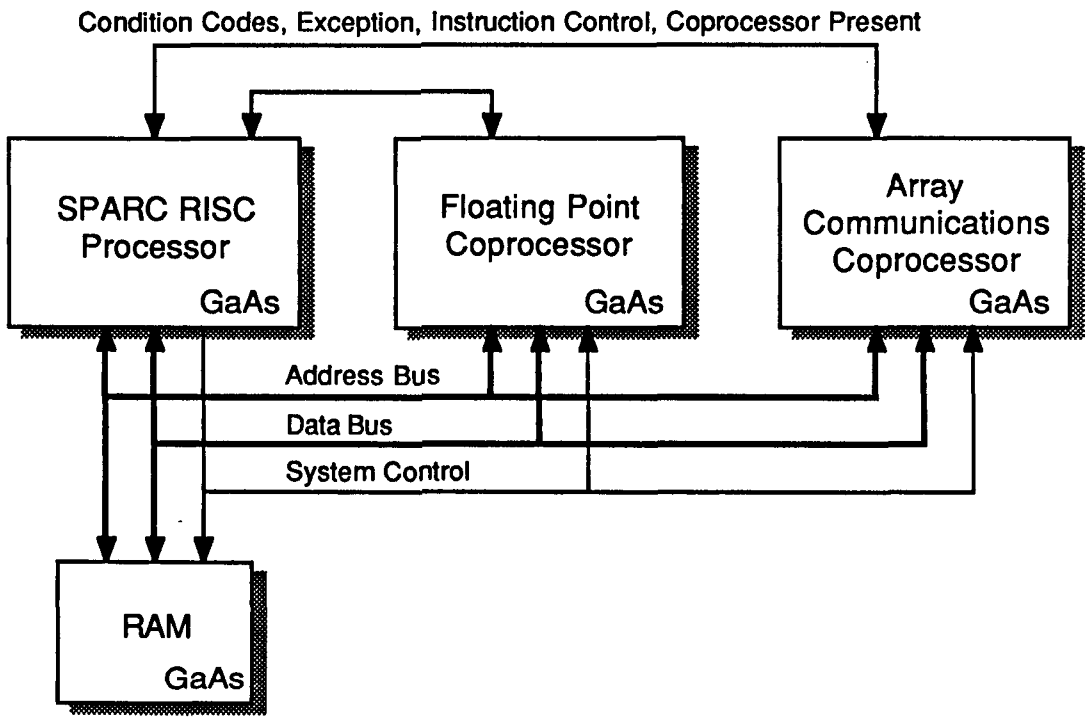

Condition Codes, Exception, Instruction Control, Coprocessor Present
SPARC RISC Processor
GaAs
Floating Point Coprocessor
GaAs
Array Communications Coprocessor
GaAs
Address Bus
Data Bus
System Control
RAM
GaAs

Figure 4-3 Systolic Array Computational Node.

A large number of registers are implemented in the SPARC architecture. The registers are organized as overlapping sets to facilitate CALL and RETURN procedures. The architecture specifies between 6 and 32 windows, with each window containing 32 registers organized as shown in Figure 4-4. Each task sees 8 GLOBAL registers, 8 LOCAL registers, 8 IN registers, and 8 OUT registers in a window.

The IN and OUT registers are shared across windows, i.e. window W3 IN registers are common to window W2 OUT registers, and W3 OUT registers are common to W4 IN registers. Procedures can quickly pass data through these shared registers.

This register architecture can be exploited by numerical algorithms passing data to subroutines, and as implemented by SPEC, the architecture allows subroutines to be nested six deep.

The SPARC architecture specifies two operating modes, USER mode and SUPERVISOR mode. Array configuration and control software can execute in the secure SUPERVISOR mode, while numerical routines will execute in USER mode.

A block diagram of the basic SPARC architecture to be implemented by SPEC is shown in Figure 4-5. The processor has separate 32-bit address and instruction/data busses. The operation of the processor is broken into two primary units, the Arithmetic and Logic Unit and the Shift Unit. Although the register file can have up to 32 overlapping windows, SPEC's baseline GaAs implementation will provide only the six required minimum.

Instructions can have up to two source registers and one destination register, thus two source operands can be operated on and returned to a different register by the Arithmetic and Logic Unit or the Shift Unit. Two Program Counters are maintained, one PC points to the current instruction address and one PC points to the next instruction address. All instructions are 32 bits and are aligned on 32-bit boundaries in memory, simplifying instruction decode and execution. Only Load and Store instructions are used to access memory. All other instructions operate only on internal registers.

A complete description of the SPARC architecture can be found in *The SPARC™ Architecture Manual* published by Sun Microsystems, Inc., Mountain View, California.

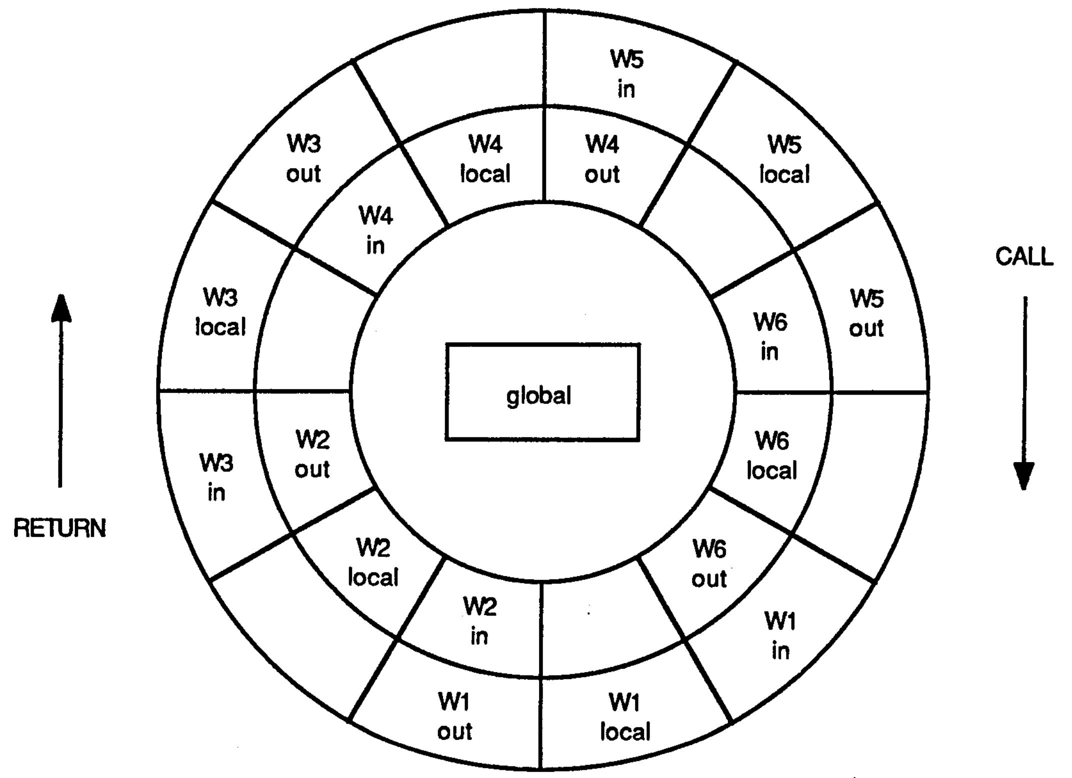


Figure 4-4 SPARC Register Window Architecture.

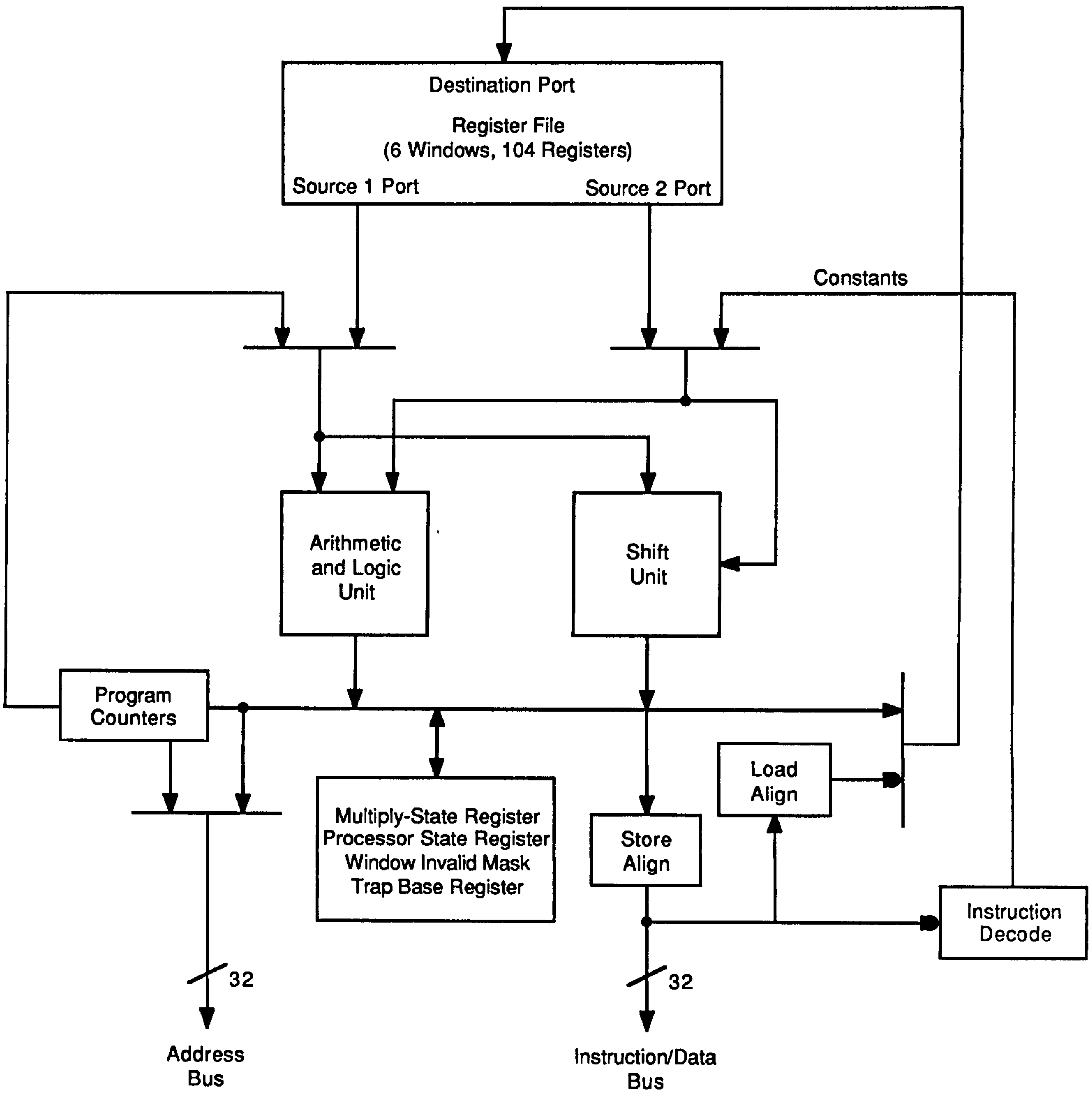


Figure 4-5 SPARC Processor Architecture.

4.3.2 Bit-Serial Floating Point Coprocessor

The SPARC architecture specifies the instruction set, control registers, and operation of a Floating Point Unit. The FPU may be a coprocessor or integral to the processor. SPEC's design allocates the FPU to a coprocessor, due to the expected gate count required to implement the unit.

SPEC's Floating Point Coprocessor (FPC) design consists of a register file, a Floating Point State Register, a four entry Floating Point Queue, an instruction decode and execution unit, a Multiplication Unit, a Division Unit, an Addition Unit, and a Subtraction Unit. The FPC will be designed in accordance with the ANSI/IEEE-754-1985 floating point specification. As specified by the SPARC architecture, each arithmetic unit can operate concurrently with other units. A block diagram of the FPC is shown in Figure 4-6.

High performance silicon Floating Point Coprocessors, such as the Motorola 68882 and the Intel 80387, are of equal or greater complexity than the accompanying CISC processor. SPEC has estimated the level of complexity for a full FPU implementation to be beyond the capability of current GaAs technology. Therefore, the arithmetic units will be implemented as bit-serial units to lower the design complexity.

This will degrade the performance of the FPC, but it will provide an avenue for implementing the FPC in near-term GaAs technology. Examination of the addition, subtraction, and multiplication times for bit-serial operation given in Table 4-1 reveals that these operations will, however, compare favorably in performance with fully parallel silicon units operating at much lower clock frequencies. As GaAs technology progresses, bit-serial operations can be expanded into fully parallel operations as technology permits.

A design for the Floating Point Multiplication Unit is shown in Figure 4-7. Data from register file source 1 and register file source 2 is loaded into exponent and fraction buffers at the initiation of the instruction. This load operation initiates both a bit-serial exponent add operation and a bit-serial fraction (mantissa) multiply operation. Normalization data is fed from the bit-serial multiplier to the bit-serial exponent adder and the new floating point exponent and fraction are placed in a result buffer. At the completion of the bit-serial operation, the instruction completes by placing the result in the proper destination register. Exception, Rounding, Overflow, and Trap information is communicated to the Floating Point Status Register during the completion stage of the multiplication instruction.

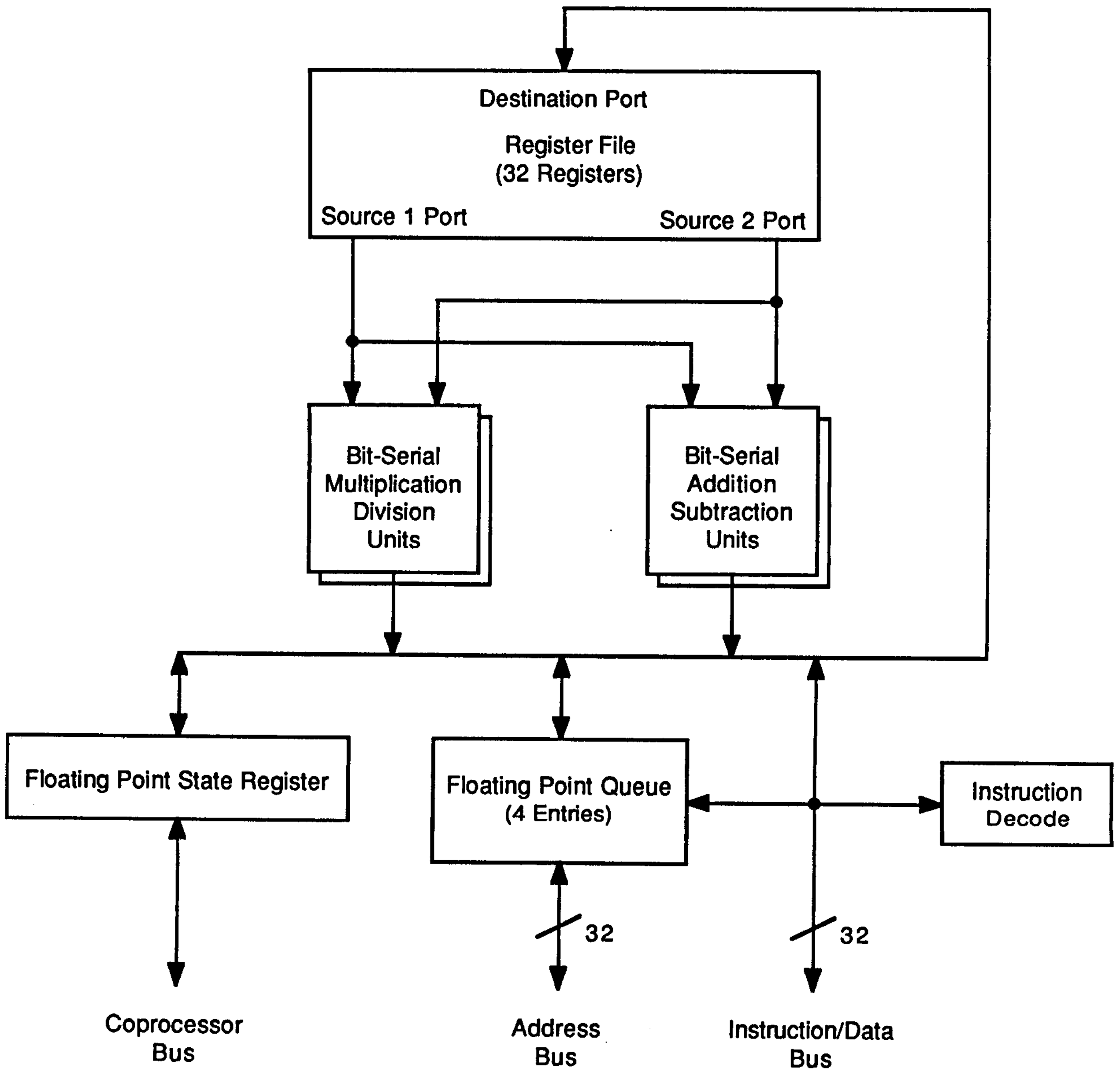

Destination Port
Register File
(32 Registers)
Source 1 Port
Source 2 Port
Bit-Serial Multiplication Division Units
Bit-Serial Addition Subtraction Units
Floating Point State Register
Floating Point Queue (4 Entries)
Instruction Decode
32
32
Coprocessor Bus
Address Bus
Instruction/Data Bus

Figure 4-6 SPARC Floating Point Coprocessor Architecture.

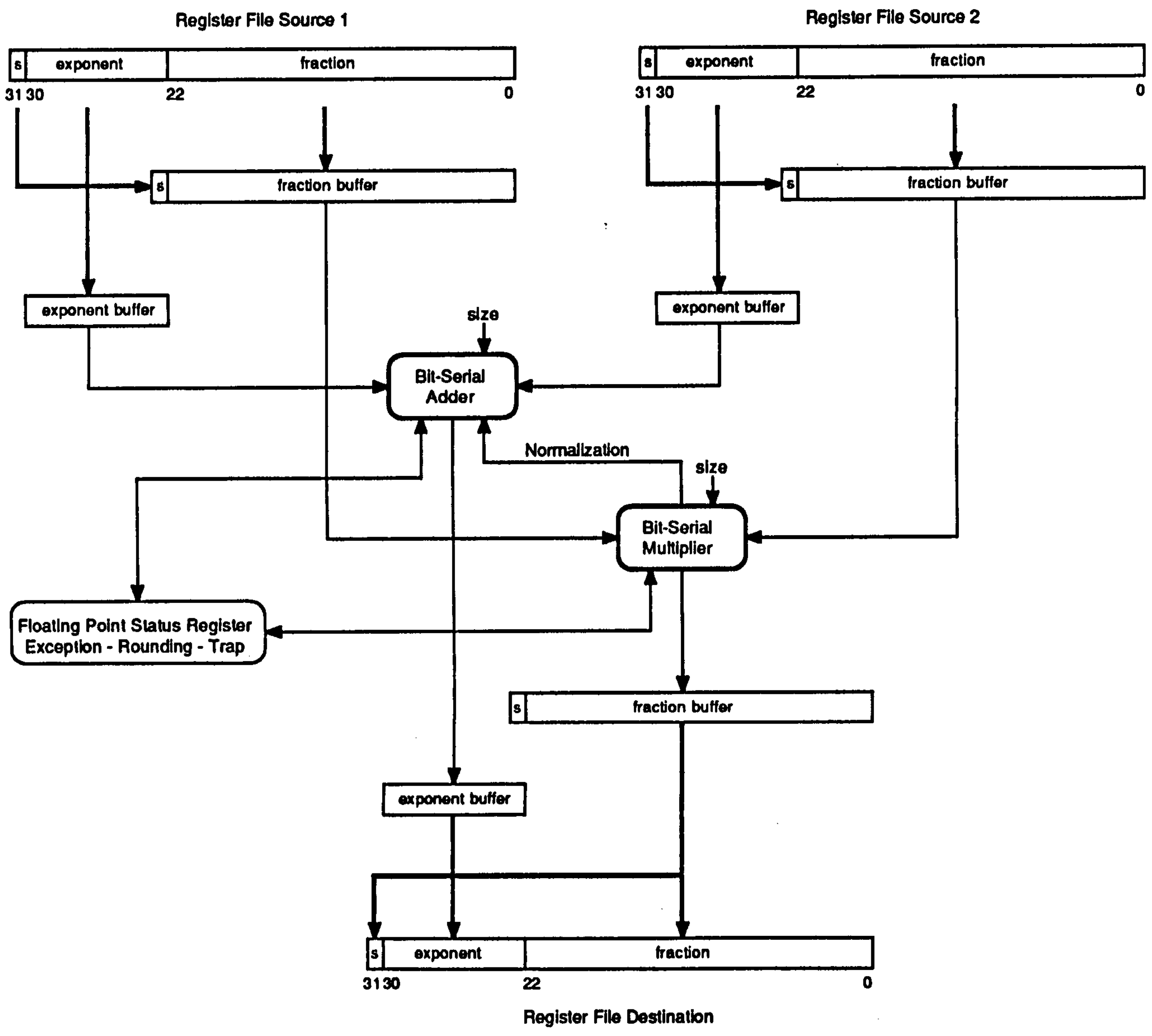


Figure 4-7 Floating Point Coprocessor Multiplication Unit.

A similar design has been established for the Floating Point Addition Unit, and is shown in Figure 4-8. Data from the Source 1 register and the Source 2 register is loaded into exponent and fraction buffers at the initiation of the floating point addition instruction. A bit-serial comparator compares the two exponents, determines which fraction is to be shifted and by how much, and passes the data to the proper bit-serial shifter (delay). After normalization, the fractions are added by the bit-serial adder, with the carry fed to the comparator. The resultant fraction and exponent are then fed into a buffer. At the completion of the addition instruction, the result is loaded into the proper destination register.

As shown in Figure 4-6, a four entry Floating Point Queue is provided to hold up to four instructions and instruction addresses. This design allows the FPC to operate concurrently with the processor and the Communications Coprocessor, discussed in the following section.

4.3.3 Bit-Serial Array Communications Coprocessor

To facilitate rapid inter-node data transfer, SPEC has developed a unique communications coprocessor architecture. This communications architecture offers several advantages over traditional inter-processor communication methodologies, such as dual-port RAM, global RAM, and low speed, memory mapped, serial communication devices.

An overview of the communications coprocessor architecture is shown in Figure 4-9. The coprocessor architecture includes a physical interface to the host processor, an instruction decode and execution unit, status registers, a communications queue for pending instructions and addresses, communication registers, and multiple communication units to service the physical link. Each communication unit includes input and output buffers, low level link control functions, and interface drivers/receivers.

A SPARC compatible implementation of the communications coprocessor is shown in Figure 4-10. The unit has full 32-bit address and instruction/data buses, an eight entry communications queue, a number of status registers, eight full-duplex bit-serial communication units, and 32 32-bit communication registers.

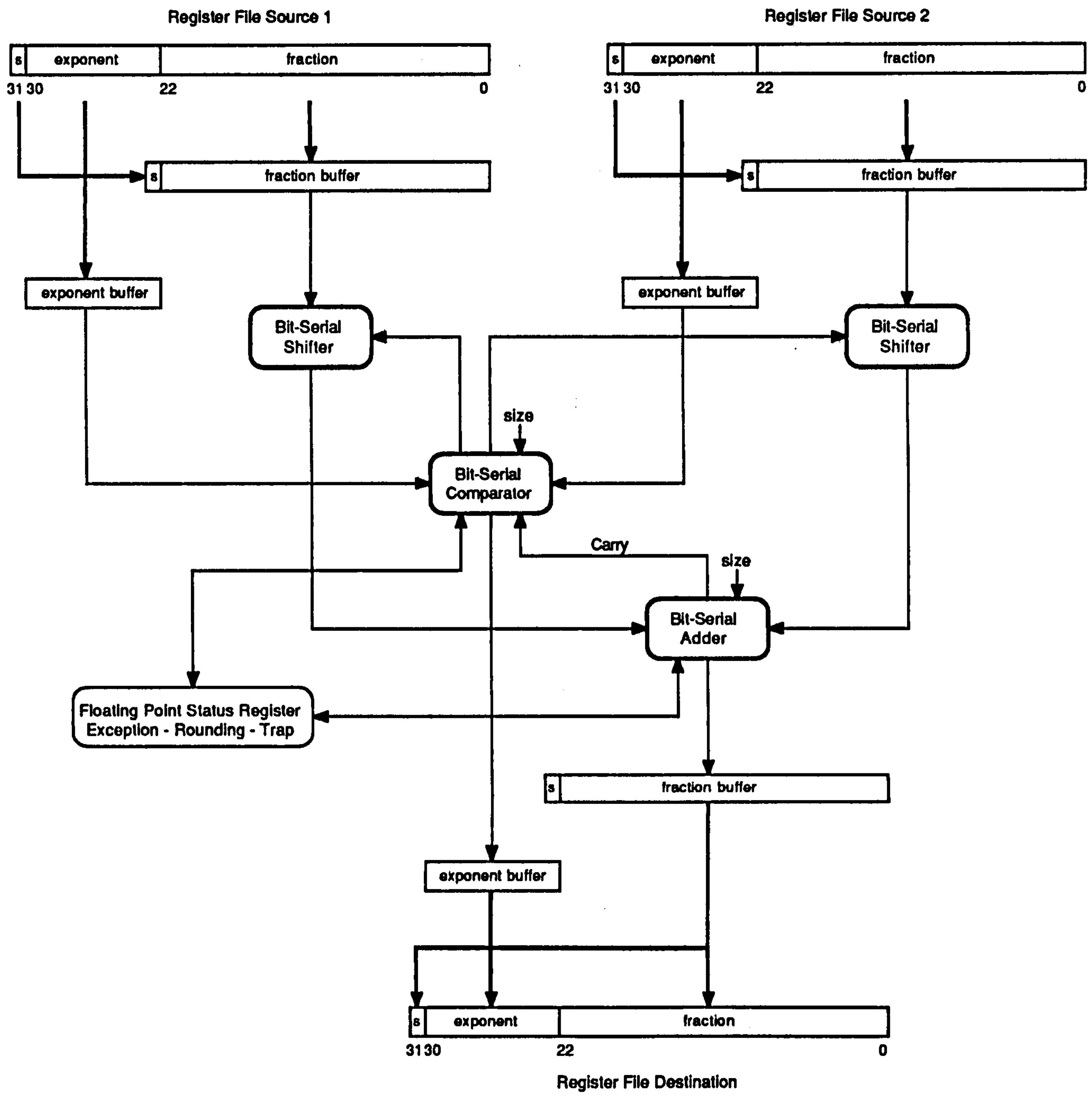


Figure 4-8 Floating Point Coprocessor Addition Unit.

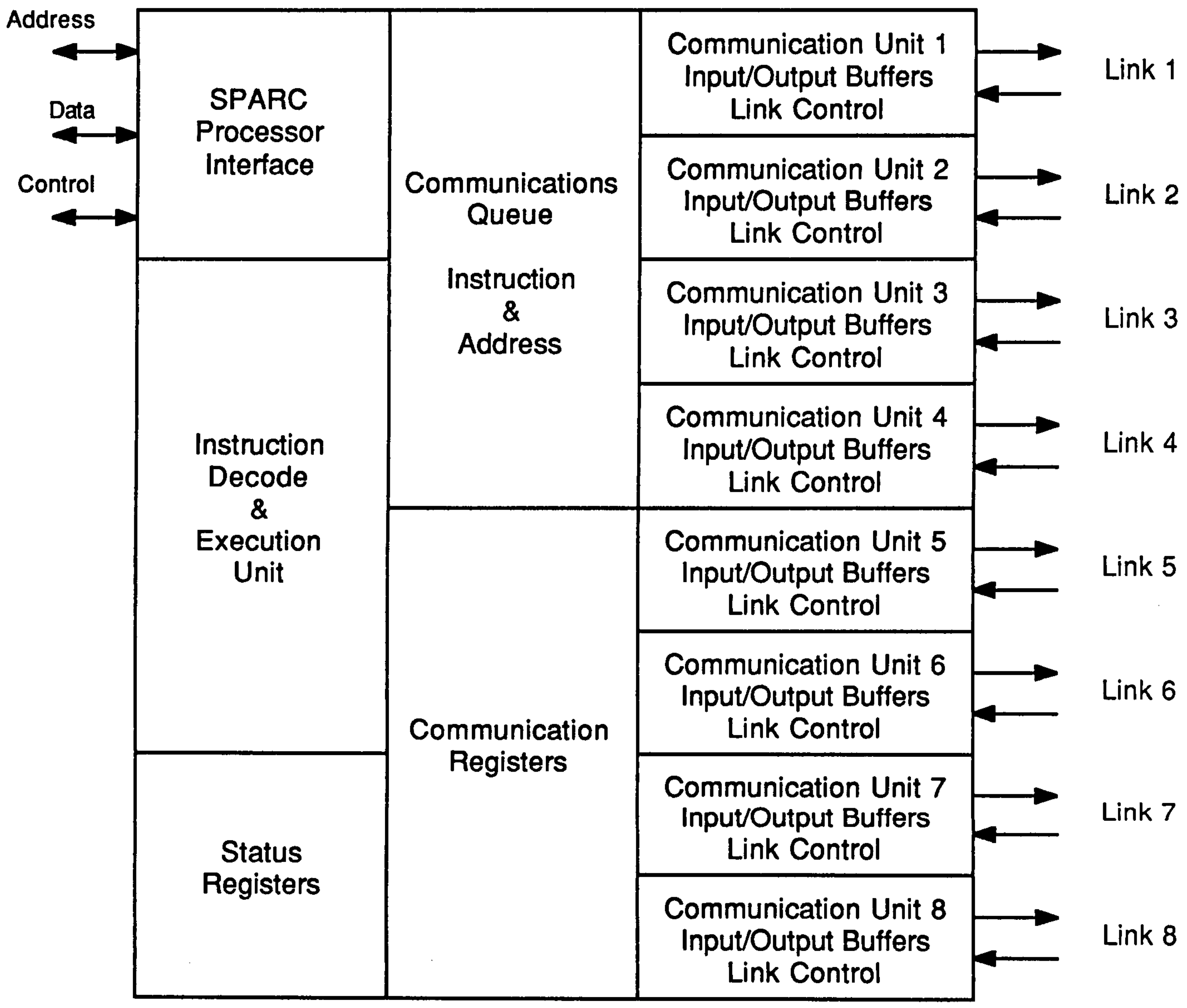

Address
Data
Control
SPARC Processor Interface
Instruction Decode & Execution Unit
Status Registers
Communications Queue
Instruction & Address
Communication Registers
Communication Unit 1 Input/Output Buffers Link Control
Communication Unit 2 Input/Output Buffers Link Control
Communication Unit 3 Input/Output Buffers Link Control
Communication Unit 4 Input/Output Buffers Link Control
Communication Unit 5 Input/Output Buffers Link Control
Communication Unit 6 Input/Output Buffers Link Control
Communication Unit 7 Input/Output Buffers Link Control
Communication Unit 8 Input/Output Buffers Link Control
Link 1
Link 2
Link 3
Link 4
Link 5
Link 6
Link 7
Link 8

Figure 4-9 Communications Coprocessor Overview.

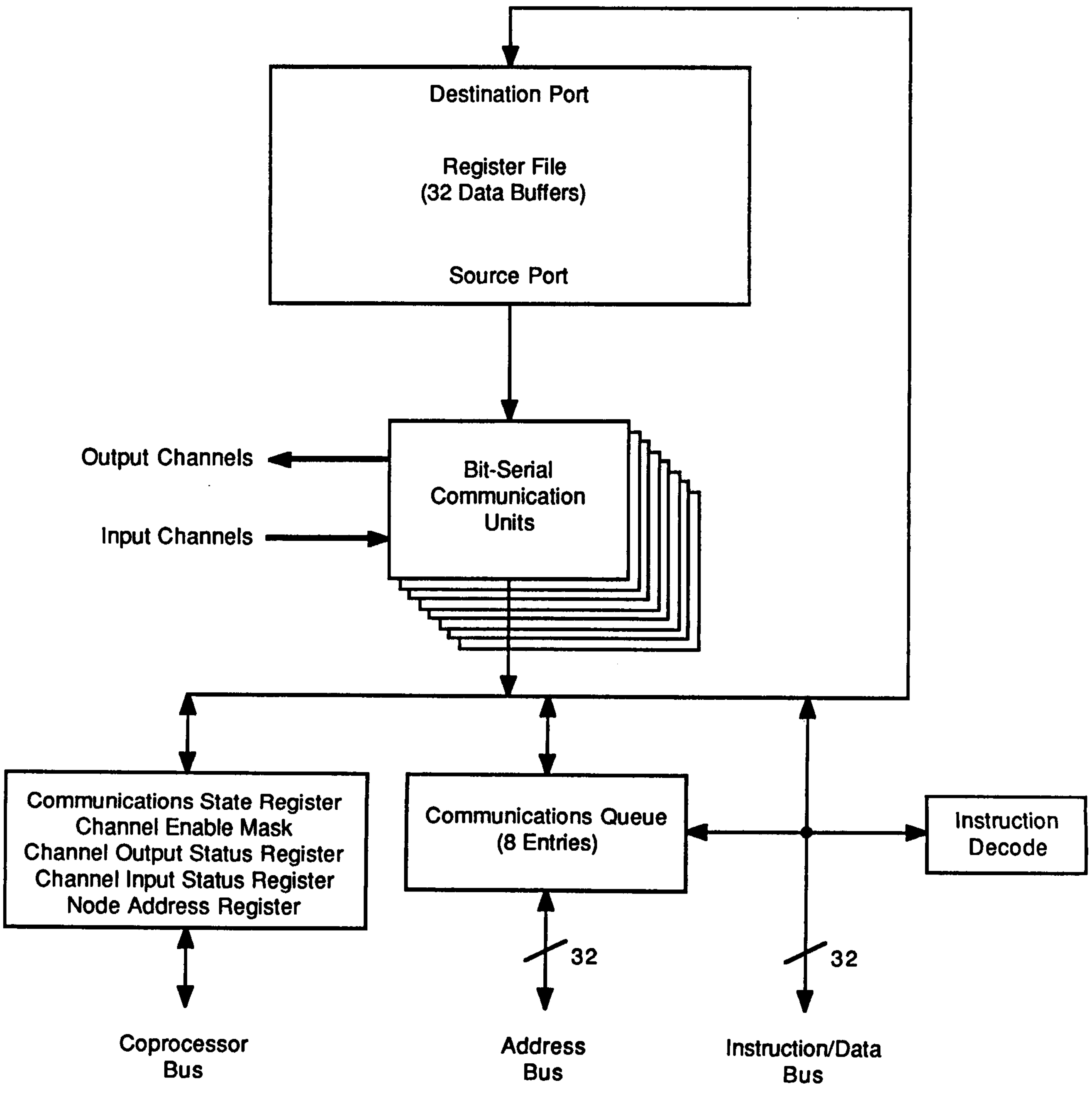


Figure 4-10 SPARC Communications Coprocessor Architecture.

The coprocessor includes a Communications State Register, for traps, exception control, condition codes, and queue control, a Channel Enable Register, to allow selective software enable of individual communication units, a Channel Output Status Register, to allow testing of output channel status, a Channel Input Status Register, to allow testing of input channel status, and a Node Address Register, which is set at power-up by external pin programming to enable the processor node to recognize its array position. All communications coprocessor registers are shown in Figure 4-11.

Operation of the bit-serial communication units is shown in Figure 4-12. Each communication unit operates independently and concurrently with the other units. Each unit is full-duplex, and can send and receive data simultaneously.

Data transfer between nodes is accomplished in 32-bit packets, with each communication unit transmiting up to 32 packets in a single SEND operation. Hardware handshaking between units is accomplished using a Device Ready (RDY) signal. Communication Unit registers are described in Figure 4-13. Instruction execution is depicted in Figure 4-14. A SEND instruction moves up to 32 data words from the Communication Registers to the specified Communication Unit Output Register, as the register becomes available. After loading the Output Register, control is passed to the Communication Unit for completion of the transmit operation. After the word is transmitted, control is returned to the main execution unit for completion of the SEND instruction.

Likewise, a RECV instruction performs the reverse operation, loading data from a Communications Unit Input Register into the Communication Registers.

The Communication Unit Input & Output Registers are not directly accessible by the host processor. All incoming and outgoing data passes through the Communication Registers, which are accessed using LOAD and STORE instructions.

Definition of the Communications Coprocessor LOAD instruction is detailed in Figure 4-15. The instruction definition follows the guidelines specified by the SPARC architecture. Three LOAD instructions are defined, Load Coprocessor Register (LDC), Load Double Coprocessor Register (LDDC), and Load Coprocessor State Register (LDCSR). The effective address is formed by the processor source registers or by a source register and a sign extended 13-bit offset.

The STORE instruction is defined in Figure 4-16. Four instructions are defined, Store Coprocessor Register (STC), Store Double Coprocessor Register (STDC), Store Coprocessor State Register (STCSR), and Store Double Coprocessor Queue (STDCQ).

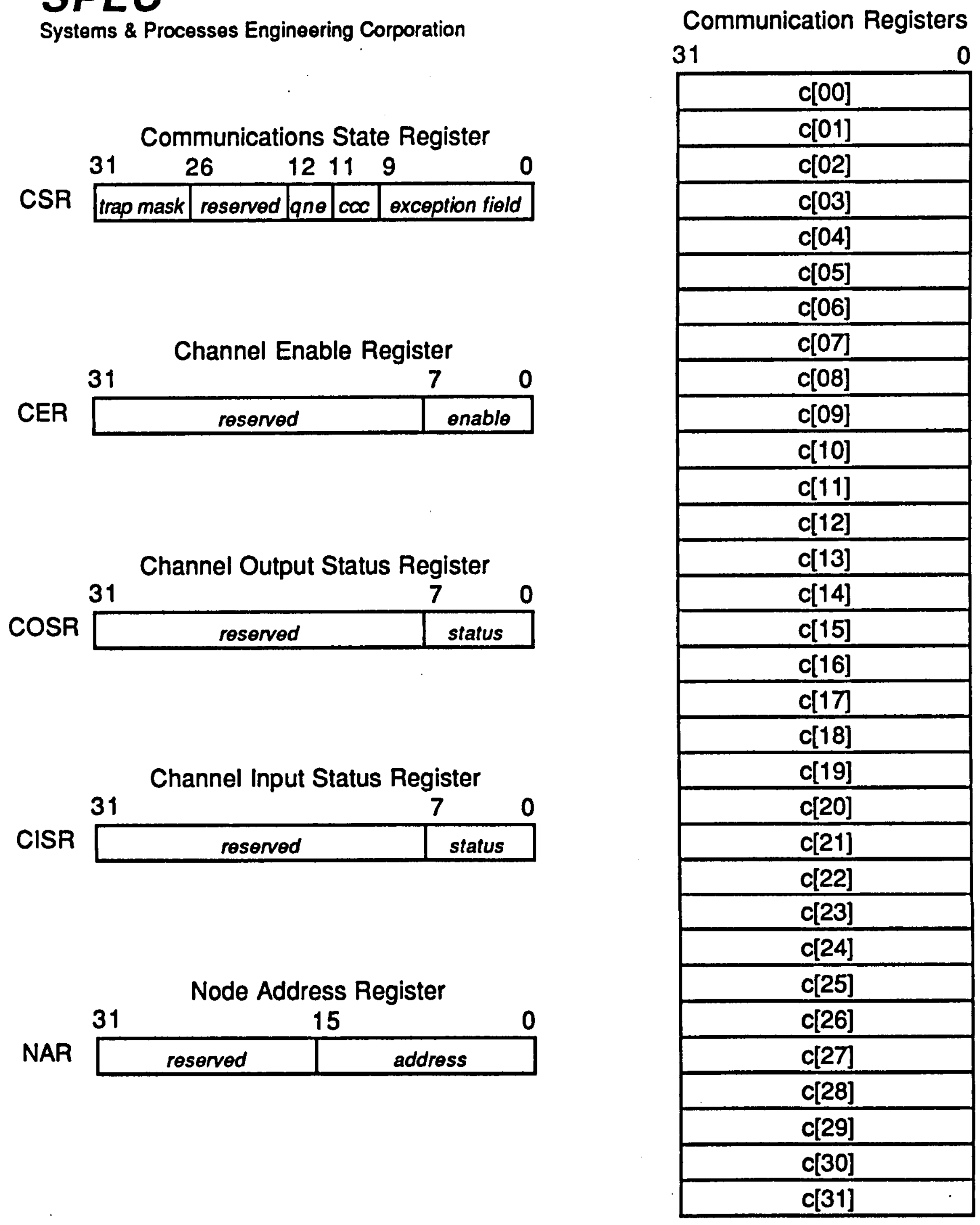


Figure 4-11 Communications Coprocessor Register Architecture.

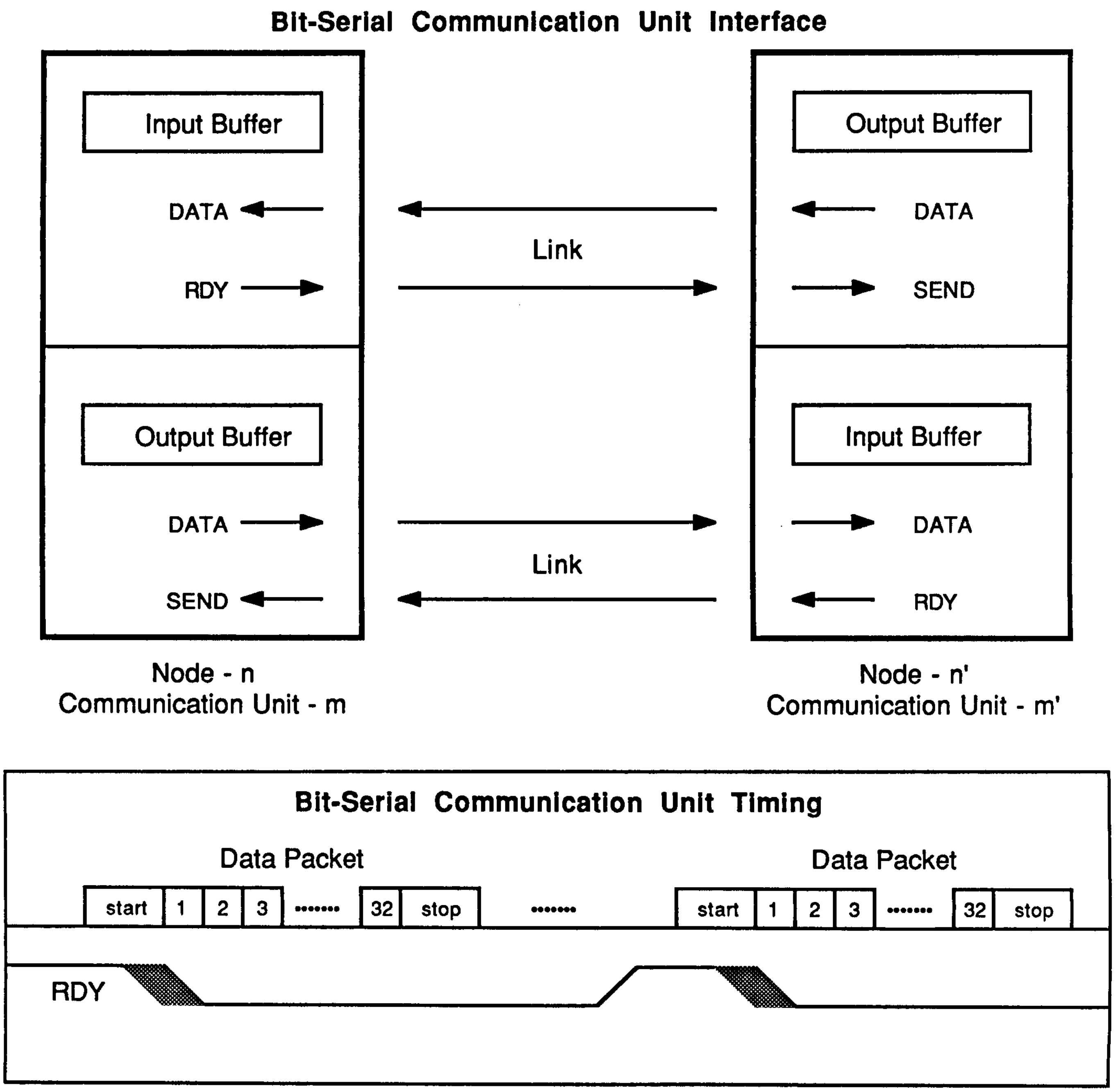


Figure 4-12 Inter-Node Communications.

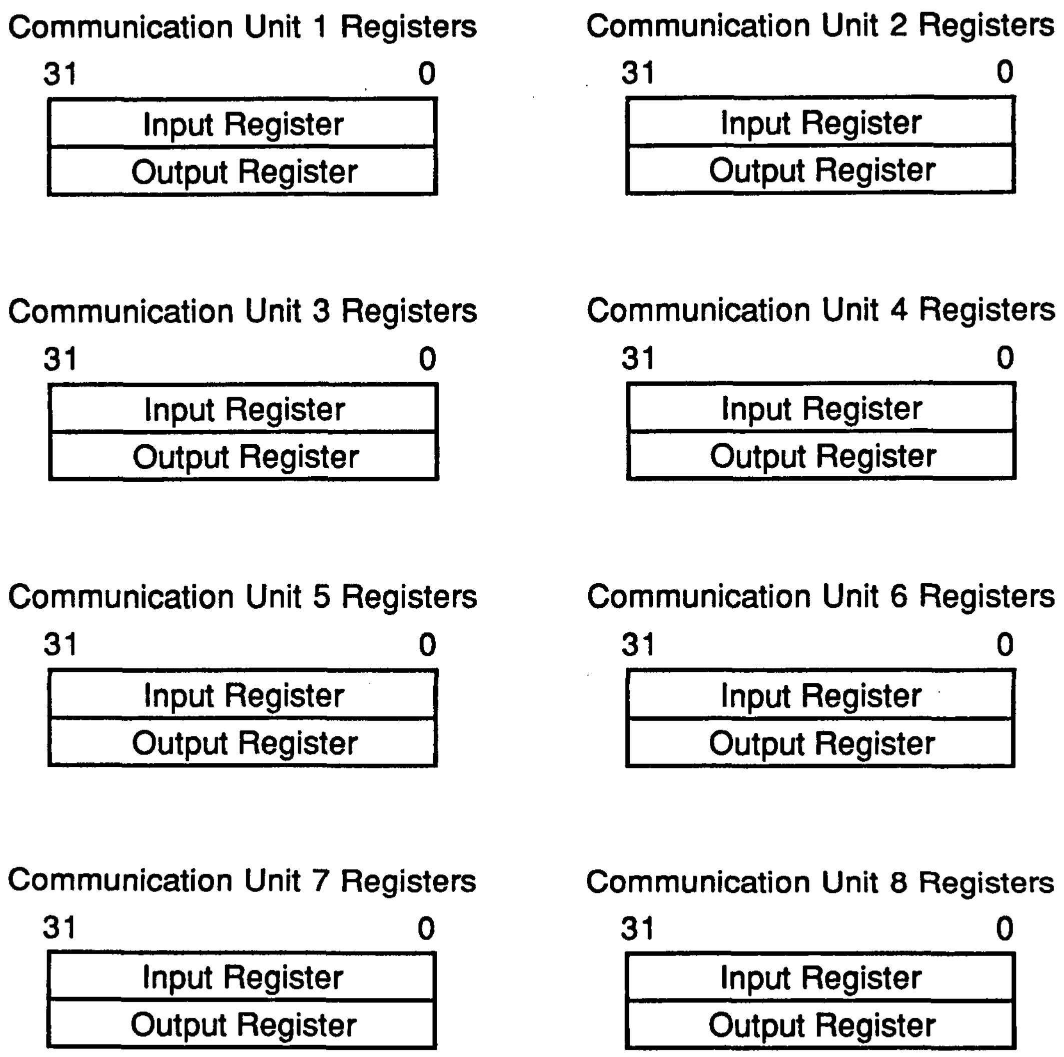


Figure 4-13 Communication Unit Register Architecture.

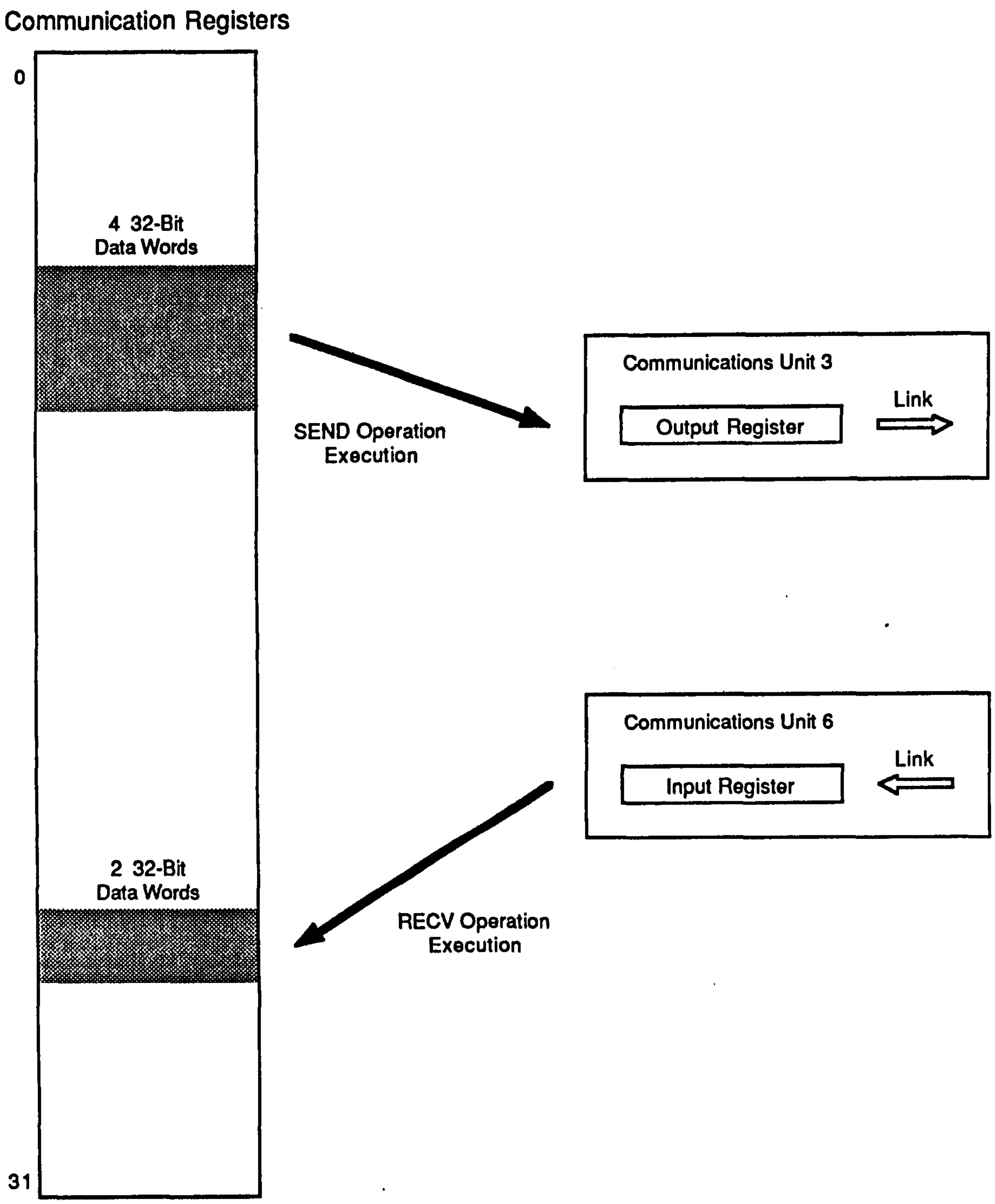
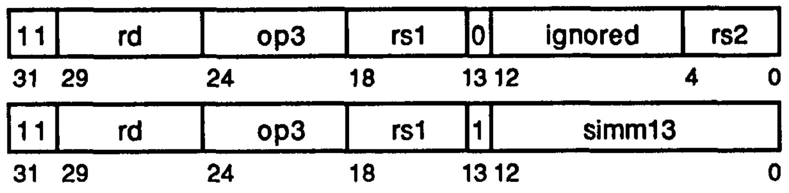


Figure 4-14 Communications Coprocessor SEND & RECV Operation.

Instruction Format



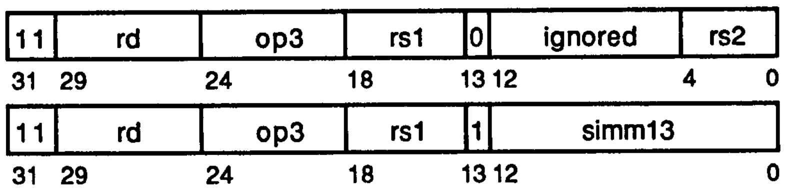
opcode	op3	operation
LDC	110000	Load Coprocessor Register
LDDC	110011	Load Double Coprocessor Register
LDCSR	110001	Load Coprocessor State Register

Assembly Language Syntax	
ld	[address], Cn
ldd	[address], Cn
ld	[address], CSRn

Effective Address:
r[rs1] + r[rs2] or
r[rs1] + sign_ext(simm13)

Figure 4-15 Communications Coprocessor Load Instructions.

Instruction Format



opcode	op3	operation
STC	110100	Store Coprocessor Register
STDC	110111	Store Double Coprocessor Register
STCSR	110101	Store Coprocessor State Register
STDCQ	110110	Store Double Coprocessor Queue

Assembly Language Syntax	
st	Cn, [address]
std	Cn, [address]
st	CSRn, [address]
std	CQ, [address]

Effective Address:
r[rs1] + r[rs2] or
r[rs1] + sign_ext(simm13)

Figure 4-16 Communications Coprocessor Store Instructions.

The SPARC architecture also defines Conditional BRANCH instructions for the coprocessor. The BRANCH instruction has a 22-bit displacement. Four condition codes are supported, supplying 16 specific branch conditions. The use of the condition codes is determined by the specific application. SPEC has identified two condition codes to be used with the TEST instruction to determine the status of the Input and Output Communication Unit buffers. Figure 4-17 describes the operation of the Communications Coprocessor conditional branch instruction.

The last class of coprocessor instructions defined by the SPARC architecture are the Coprocessor Operate instructions. These are general purpose instructions to be defined by the application. SPEC has identified four coprocessor operate instructions, Send Data Packet (CSEND), Receive Data Packet (CRECV), Test Input (Output) Status Register (CTSTcc), and Flush Communication Unit Buffer (CFLUSH). These instructions are specified in Figure 4-18.

The CSEND instruction *sends* up to 32 data words over a communications link and the CRECV instruction *receives* up to 32 data words over a link.

Instruction Format

00	a	cond	111	disp22
31	29 28		24	21 0

opcode	cond	bp_CP_cc[1:0] test operation
CBA	1000	Branch Always
CBN	0000	Branch Never
CB3	0111	Branch if 3 set
CB2	0110	Branch if 2 set
CB23	0101	Branch if 2 or 3 set
CB1	0100	Branch if 1 set
CB13	0011	Branch if 1 or 3 set
CB12	0010	Branch if 1 or 2 set
CB123	0001	Branch if 1 or 2 or 3 set
CB0	1001	Branch if 0 set
CB03	1010	Branch if 0 or 3 set
CB02	1011	Branch if 0 or 2 set
CB023	1100	Branch if 0 or 2 or 3 set
CB01	1101	Branch if 0 or 1 set
CB013	1110	Branch if 0 or 1 or 3 set
CB012	1111	Branch if 0 or 1 or 2 set

Assembly Language Syntax	
cba{,a}	label
cbn{,a}	label
cb3{,a}	label
cb2{,a}	label
cb23{,a}	label
cb1{,a}	label
cb13{,a}	label
cb12{,a}	label
cb123{,a}	label
cb0{,a}	label
cb03{,a}	label
cb02{,a}	label
cb023{,a}	label
cb01{,a}	label
cb013{,a}	label
cb012{,a}	label

Condition Codes:

0 - Input Buffer Status
1 - Output Buffer Status
2 - Reserved
3 - Reserved

{,a} optional annul bit:

If branch not taken and annul set, delay instruction not executed.

If branch taken, annul bit ignored and delay instruction executed.

Figure 4-17 Communications Coprocessor Branch Instructions.

Instruction Format

10	rd	op3	rs1	opc	rs2
31	29	24	18	13	4 0

opcode	op3	opc	operation
CSEND	110110	00000xxxx	Send Data Packet
CRECV	110110	00001xxxx	Receive Data Packet
CTSTcc	110111	00000yyyy	Test Input (Output) Status Register
CFLUSH	110111	00001zzzz	Flush Communication Unit Buffer

Assembly Language Syntax		Register Usage
send	Cn, CUm, xxxx	rd - Cn; rs1 - CUm; rs2 - ignored
recv	CUm, Cn, xxxx	rd - Cn; rs1 - CUm; rs2 - ignored
ctest	CSRn, yyyy	rd - ignored; rs1 - CSRn; rs2 - ignored
cflush	zzzz	rd - ignored; rs1 - ignored; rs2 - ignored

xxxx - Number of 32-Bit Words	Cn - Communications Register n
yyyy - Input (Output) Unit Number	CUm - Communications Unit m
zzzz - Communication Unit Number	CSRn - Communication Status Register n

Figure 4-18 Communications Coprocessor Operate Instructions.

5.0 System Simulation

5.1 Model Overview

SPEC has developed amodel for a computer program to simulate arbitrary systolic array processor topologies. The simulation is designed to model the data load operation, the computations performed at each node, the packet or data routing between nodes, and the operation to unload data from the array (see Figure 5-1).

The Systolic Array Simulator (SAS) program is designed to assess the real-time performance of an array processor. This assessment includes the processing rate of the node processor (specified in MIPS) and the communications rate between nodes (specified in Mbits/sec). The program is designed for asynchronous array operation.

5.2 Implementation

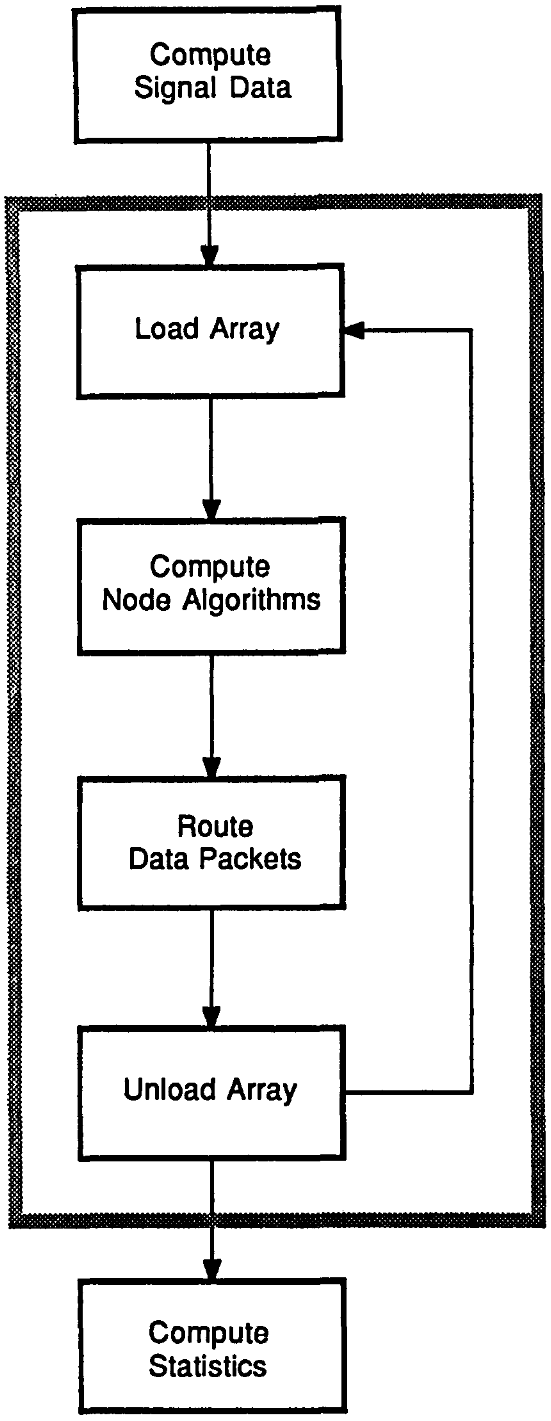
SAS is being implemented in the C programming language. At the writing of this report, development is currently in the specification stage. Structures are developed for the input data description, the node interconnection, the algorithm specification, and parameter entry.

An input data structure includes the raw signal data, and a time stamp for the arrival of each data point.

The node topology description includes the number of nodes and the interconnections made by each node. The program can read a data file which specifies the array topology.

When a data packet is created at a node it is entered into a packet array, which the main program loop analyzes during each pass to determine the state of the packet transfer. It is envisioned that eventually each node in the array will be capable of routing data packets, therefore provisions have been made for simulation of the routing algorithm at each node.

Currently, the program is capable of accepting data from the user specifying the dimensionality of the array (1-D, 2-D, or 3-D), the processing rate (MIPS), the communications rate (Mbits/sec), and the number of nodes in each dimension. The user can also select a routing methodology.



Systolic Array Simulator (SAS) Program

- Simulated Real-Time
  - Processor Rate (MIPS)
  - Communications Rate (Mbits/sec)
- Asynchronous Operation

Figure 5-1 Systolic Array Simulator (SAS) Functions.

The main program loop will execute the following sequences:

1. Load the data into the array at the communications data rate.
2. Simulate the signal processing algorithm and communications operations using "pseudo code" at each node.
3. Simulate the data flow from node to node.
4. Unload the data from the array as it becomes available at the outputs.
5. Continue execution of the main loop until all data has passed through the array.
6. Analyze the resultant data for numerical accuracy.
7. Analyze the array timing statistics gathered during the simulation.

SPEC is confident that this simulator will be completed early on during the Phase II program to support the analysis of optimum communications methods to be employed in the systolic array. Since the user can specify the algorithm to be executed at each node, the simulator should be very helpful in designing parallel processing algorithms, and assessing their capabilities and function.

6.0 Proposed Phase II Implementation

6.1 Foundry Processes & Capabilities Review

SPEC has reviewed the capabilities of a number of GaAs foundries, including site visits to McDonnell Douglas, Texas Instruments, Rockwell, GigaBit Logic, and Vitesse. The capabilities reviewed include process technology, capability of design staff, facilities, and company orientation. A diagram of the foundries reviewed is presented in Figure 6-1.

At present, SPEC believes that Vitesse offers the best solution because of their standard cell design capability, process technology, process maturity, production capacity, staff experience, and willingness and enthusiasm to participate in the Phase II program.

SPEC expects to continue this capabilities review in the early stages of the Phase II program, to include a visit to TriQuint, and follow-up visits to Vitesse and GigaBit Logic.

6.1.1 GigaBit Logic, Inc.

GigaBit Logic, Inc. offers custom, standard cell, gate array, and MSI products. GigaBit currently offers 5000 and 10,000 gate standard cells, with 20,000 gate devices projected for 1989. GigaBit's standard cells are available in the following 1 μm processes:

- Depletion Mode MESFET (D)
- Low Power Depletion mode MESFET (LPD)
- High Margin Enhancement/Depletion mode MESFET (HME/D)
- Enhancement/Depletion mode MESFET (E/D)

GigaBit standard cells can be designed with ECL, TTL, or CMOS I/O compatibility. Power dissipation for the 10,000 gate, current steering logic, E/D device is on the order of 0.4 mWatt/gate. Loaded gate delays range from 50 to 150 psec. Flip Flops operating at 7 - 8 GHz have been realized. Up to three levels of metalization are available.

GigaBit offers a number of standard cell Macros for I/O and internal cells, including input, output and clock buffers, accumulators, flip flops, inverters, nor gates, and many other functions. RAM and ROM cells are also now available.

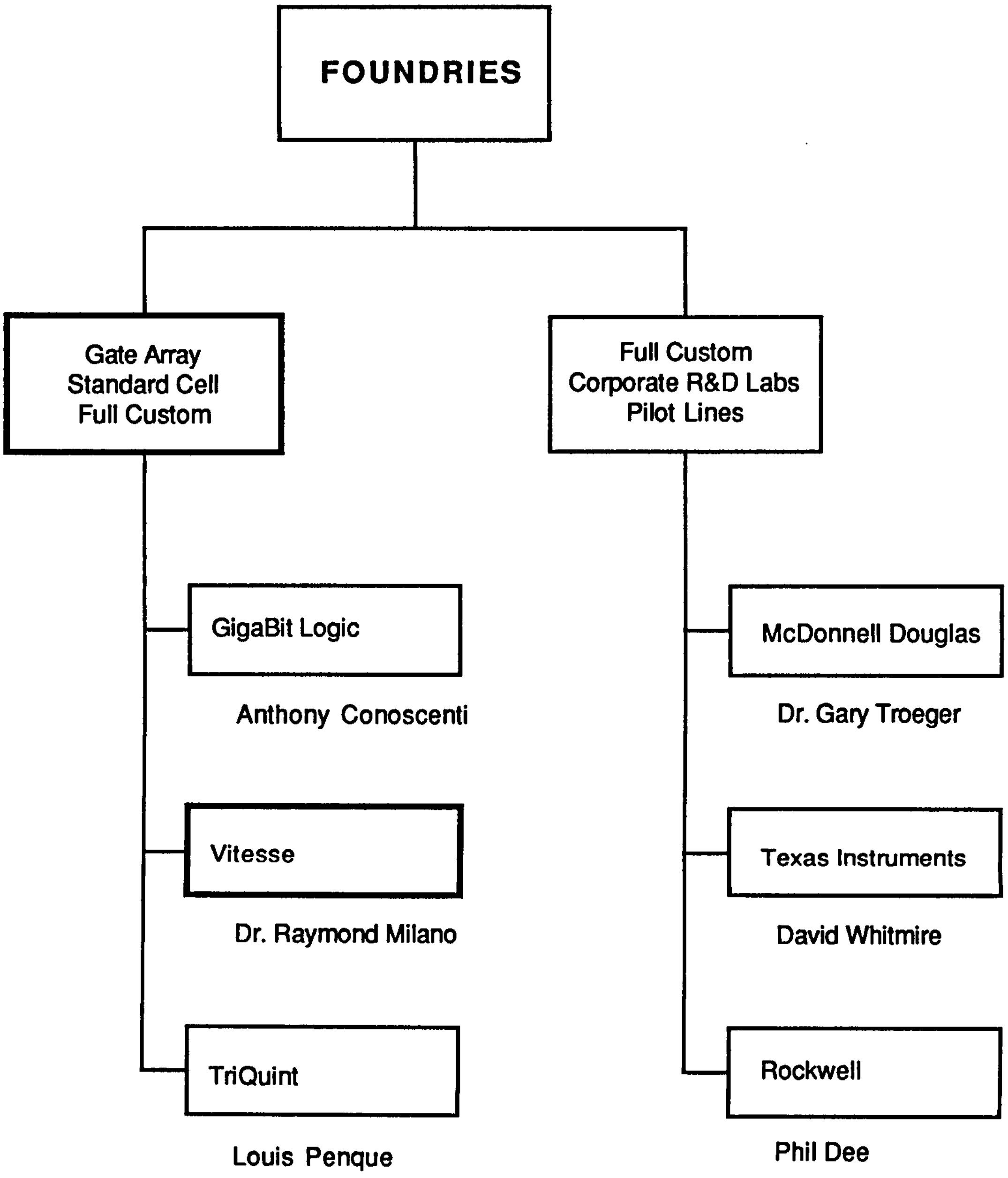


Figure 6-1 Foundries Surveyed.

Packaging options currently include 68 pin and 144 pin leaded chip carriers, with other package options in development.

Seattle Silicon and GigaBit Logic have jointly developed a GaAs compiler for custom circuit designs. The compiler has been used to develop a small ALU in Capacitor Diode FET Logic (CDFL). Discussions with Seattle Silicon indicate, however, that no additional development has occurred on this product.

Proprietary SPICE models of GigaBit processes and device characteristics are available. Daisy Systems, Mentor Graphics, and VLSI Technology workstations are supported.

GigaBit has a large production facility with a Class 10 cleanroom.

6.1.2 Vitesse Semiconductor Corporation

Vitesse Semiconductor Corporation offers complete foundry services, including wafer fabrication, packaging, testing, and quality assurance. The process is characterized by the following parameters:

- Enhancement/Depletion MESFET Technology
- NMOS-like Structure
- Simple - 9 Mask Steps
- Three levels of metal interconnect (Aluminum Metalization)
- 1.0 µm Process Lithography
- Self-aligned Gates
- Excellent Threshold Voltage Control
- High Level of Integration (20K Gates)
- High Speed Packages

The Vitesse process electrical parameters are as follows:

- Gate Delay = 120 psec
- Metal Line Delay = 60 psec/mm of metal
- Fan-Out Delay = 35 psec/f.o.
- Clock to Q Delay = 420 psec
- Power Dissipation = 0.4 mWatt/gate

Vitesse currently offers several tools for custom designs. By fourth quarter 1988 these tools will include a library of 45 Macro Cells (SSI/MSI) and 6 Mega Cells (LSI components). By third quarter 1989 they project having 250 Macro Cells, 10 Mega Cells and Compiled Cells (data path unit, state machine, & RAM).

Vitesse has developed a custom SPICE model of their process to assist in device design. Vitesse supports Sun, DEC VAX, Apollo and other major engineering workstations.

Vitesse has a 45,000 square foot fabrication facility, a Class 10 clean room, and over $1M in high speed test equipment.

6.1.3 TriQuint Semiconductor, Inc.

TriQuint Semiconductor, Inc. offers a wide range of both digital and analog GaAs foundry services. TriQuint offers a 1 µm Enhancement/Depletion Mode process for high level integration.

TriQuint offers a Q-LOGIC Standard Cell Design capability, including a large library of macros. TriQuint standard cells are capable of speeds of up to 2 - 3 GHz. Input and output cells are available for ECL, TTL, and CMOS compatibility.

Two logic families are available, ZFL (Zero-diode FET Logic) and SCFL (Source Coupled FET Logic). Currently, 6000 equivalent gates are available in the standard cell. With SCFL cells, gate delays are 65 psec, and with ZFL cells, gate delays are 150 psec. Loading delays for SCFL and ZFL are 13 psec/fan out (or 40 psec/mm of wire) and 7 psec/fan out (or 50 psec/mm of wire), respectively. The maximum toggle rate for SCFL Flip Flops is 2 GHz, and 1 GHz for ZFL Flip Flops.

SCFL dissipates 2.7 mWatt/gate and ZFL dissipates 0.8 mWatt/gate (2-input NOR).

Multilayer ceramic packages, of up to 132 pins are available.

Other TriQuint services include layout and verification, testing, and packaging.

6.1.4 Other Foundry Sources

6.1.4.1 Texas Instruments

Texas Instruments is currently developing a 32-bit GaAs microprocessor for the Defense Advanced Research Project Agency (DARPA). The processor has separate instruction and data memory (Harvard architecture) and a floating point coprocessor (now approximately 80% complete). The processor is designed to execute the MIPS standard instruction set.

Texas Instrument's processor is a 12,000 gate, full custom design implemented in HI^2L. Texas Instruments also offers GaAs gate arrays of up to 13,000 gates.

6.1.4.2 McDonnell Douglas Corporation

Like Texas Instruments, McDonnell Douglas is also a DARPA pilot line for GaAs production. McDonnell Douglas uses a low power Junction FET (JFET) GaAs process. This process does not have the speed capability of TI's HI^2L process, but is much more suitable for implementation of on-board RAM.

McDonnell Douglas has also implemented a MIPS Instruction Set Architecture (ISA) compatible RISC microprocessor.

6.1.4.3 Rockwell International Corporation

Rockwell recently developed an 8-bit slice using depletion mode technology. This slice includes 32 registers, and was implemented in approximately 1200 gates. Discussions with Rockwell indicate that they have not yet developed an E/D process.

6.2 Custom vs Standard Cell vs Gate Array Approach

After assessment of the design requirements, particulary gate count, SPEC has determined that a standard cell approach is the most cost effective and realizable design approach. Integration densities to 20,000 gates are currently realizable using Vitesse standard cell integrated circuits.

SPEC has developed estimates of the number of gates and pins required to implement the processor, floating point coprocessor, and the communications coprocessor. The estimates are detailed in Table 6-1. The RISC processor should require approximately 18,000 gates in a 128+ pin package. The floating point coprocessor should require approximately 19,000 gates in a 112+ pin package,

Device Element	Gates	Pins
Processor		
Register File	13312	
Status Registers	512	
Instruction Decode & Operation	2000	
Program Counters	256	
Arithmetic & Logic Unit	1000	
Shift Unit	200	
Address Bus	128	32
Instruction/Data Bus	256	32
Misc. Control, Power, & Ground Signals	256	64
Total	**17920**	**128**
Floating Point Coprocessor		
Register File	4096	
State Register	128	
Instruction Decode & Operation	2500	
Floating Point Address/Instruction Queue	1024	
Bit-Serial Multiplication Unit	2484	
Bit-Serial Division Unit	3508	
Bit-Serial Addition Unit	2228	
Bit-Serial Subtraction Unit	2228	
Address Bus	128	32
Instruction/Data Bus	256	32
Misc. Control, Power, & Ground Signals	256	48
Total	**18836**	**112**
Array Communications Coprocessor		
Register File	4096	
Status Registers	640	
Instruction Decode & Operation	2000	
Communications Address/Instruction Queue	2048	
Bit-Serial Communication Units	6048	32
Node Address	64	16
Address Bus	128	32
Instruction/Data Bus	256	32
Misc. Control, Power, & Ground Signals	256	32
Total	**15536**	**144**

Table 6-1 Device Gate Count and Pin Count Estimates.

while the communications coprocessor should require approximately 15,500 gates in a 144+ pin package.

6.3 Demonstration Unit Design

As delineated in Figure 6-2, the Phase II development program will consist of the parallel development of the three SPARC units, the processor, FPC, and ACC. Development will proceed in parallel on all three units, with priority initially on the basic RISC processor, followed by the communications coprocessor (required to implement the array), and subsequently the floating point coprocessor.

6.3.1 Hardware Implementation

A hardware demonstration unit will be designed on a 9-U VMEbus card, compatible with a Sun Microsystems workstation. The demonstration hardware will consist of 16 processor nodes, organized as two sets of eight nodes. The nodes will be interconnected to execute a length-64 Block DFT. A diagram of the proposed hardware is shown in Figure 6-3.

Each processor will have a minimum of 4 Kbytes of high speed dual-ported RAM for communication with the control processor (the workstation CPU). The demonstration unit will be designed to facilitate downloading of software to the RISC processors.

The demonstration unit will be designed to allow reconfiguration of the array by jumpers on the printed circuit board.

6.3.2 Software Development

Software development for the SPARC systolic array processor will occur on a Sun Series-4 workstation, which is based on the SPARC architecture. Sun offers a wide range of software development tools, including compilers, symbolic debuggers, and other utilities. Array processor code can be partially debugged on the workstation.

SPEC will develop a length-64 DFT software package to demonstrate the capabilities of the array processor. Other signal processing algorithms (as described in this report) will be implemented on the SPARC systolic array processor, per customer direction.

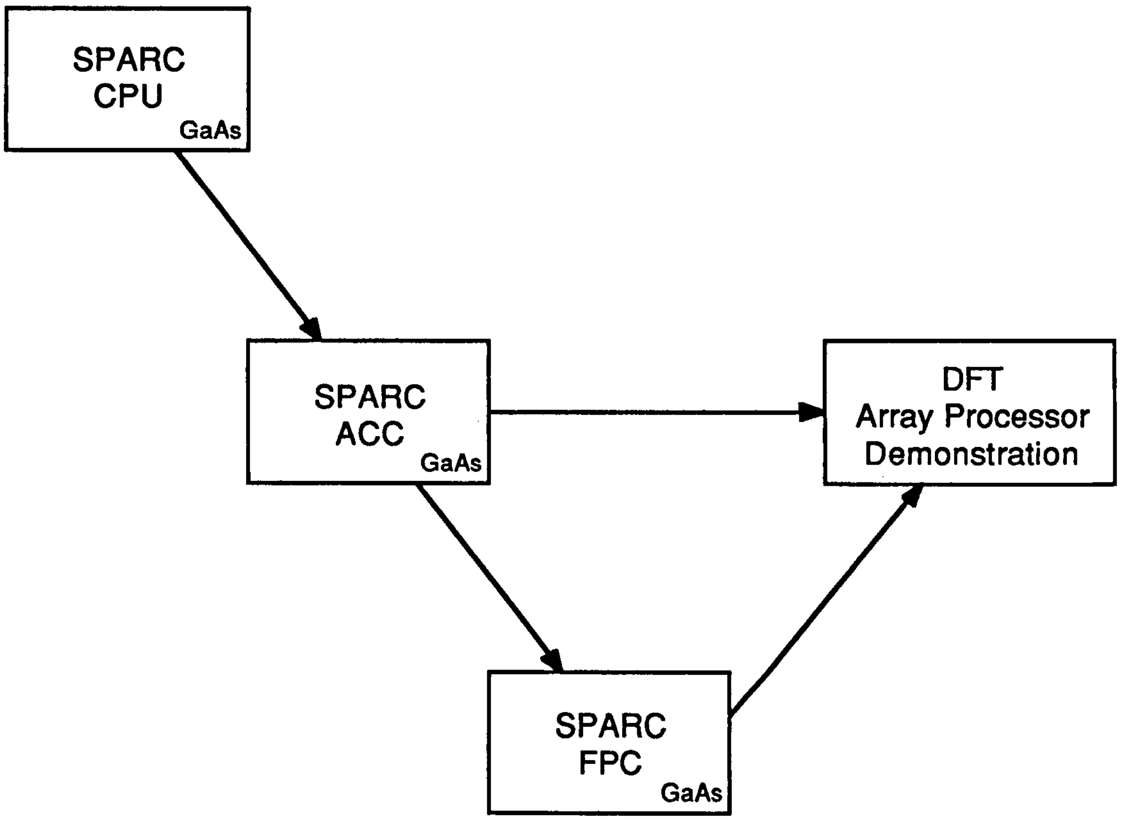


Figure 6-2 Phase II Development Program Objectives.

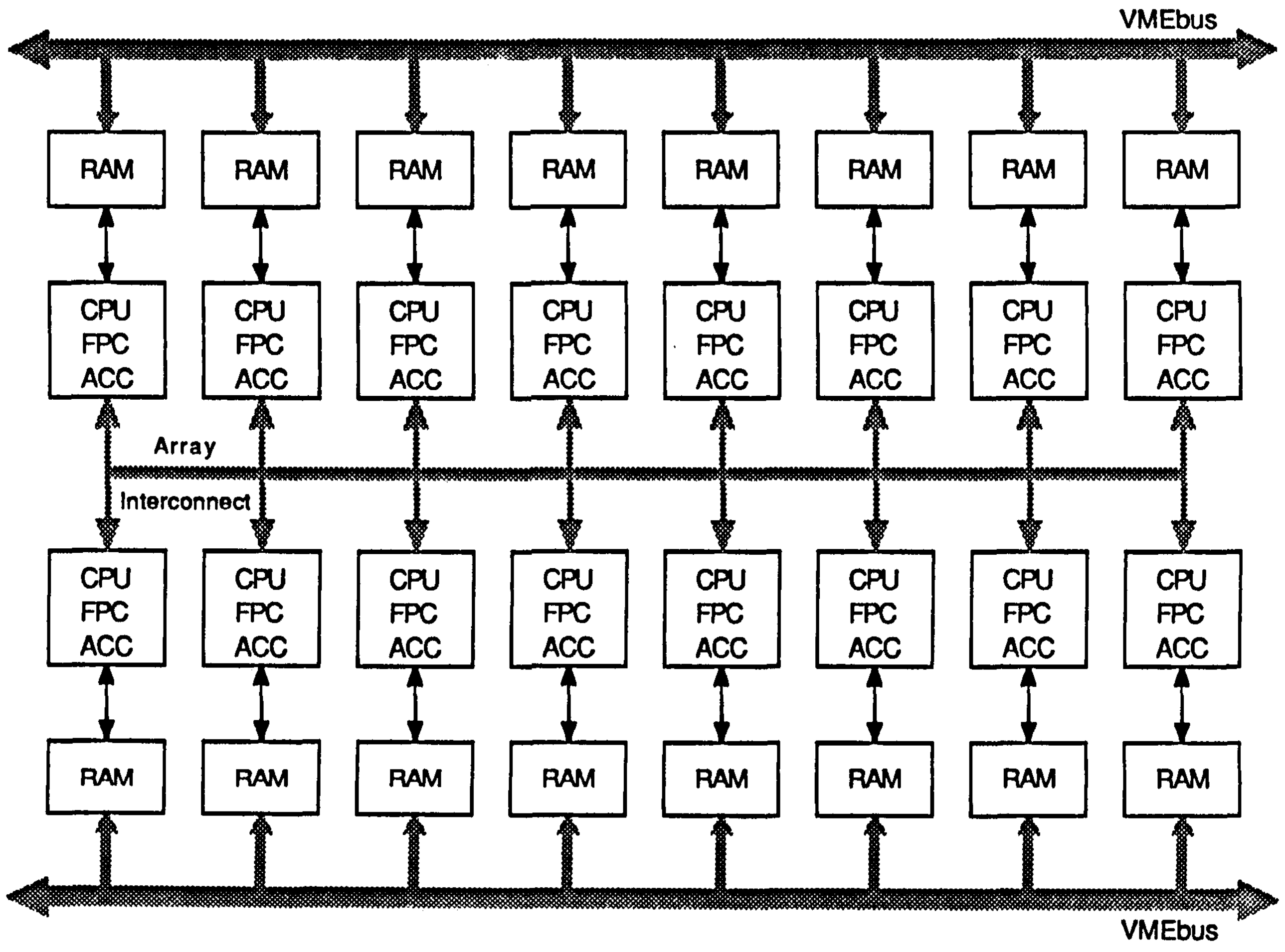


Figure 6-3 VMEbus (9U) Compatible GaAs Systolic Array DFT Processor.

7.0 Conclusions

SPEC has developed an innovative array processor architecture for computing Fourier transforms and other commonly used signal processing algorithms. The architecture described in this report is designed to extract the highest possible array performance from state-of-the-art GaAs technology. SPEC's architectural design includes a high performance RISC processor implemented in GaAs, along with a Floating Point Coprocessor and a unique Array Communications Coprocessor, also implemented in GaAs technology. Together, these data processors represent the latest in technology, both from an architectural and implementation viewpoint.

SPEC has examined numerous algorithms and parallel processing architectures to determine the optimum array processor architecture. SPEC has developed an array processor architecture with *integral* communications ability to provide maximum node connectivity. The Array Communications Coprocessor embeds communications operations directly in the core of the processor architecture.

A Floating Point Coprocessor architecture has been defined that utilizes Bit-Serial arithmetic units, operating at very high frequency, to perform floating point operations. These Bit-Serial devices reduce the device integration level and complexity to a level compatible with state-of-the-art GaAs device technology. Operating at clock frequencies in excess of 1 GHz, these Bit-Serial units compare favorably to parallel units implemented in silicon technology, while providing inherent radiation hardness and superior speed-power product of GaAs.

SPEC has selected Sun Microsystems' Scalable Processor ARChitecture (SPARC) as a basis for the high speed RISC processor. The SPARC is ideally suited for array processor applications, with a large register set, efficient instruction set, and simple implementation. The SPARC RISC processor has previously been implemented in a silicon gate array, with the design requiring less than 20,000 gates. This compares very favorable to other RISC processor implementations, which have required many times the device complexity.

SPEC has selected Vitesse Semiconductor's enhancement/depletion mode process for design implementation. Vitesse's GaAs foundry is now offering Standard Cell design capability up to 20,000 gates, which offers the best cost and performance alternative, and ensures success in a Phase II development activity.

In selecting SPARC basis for the processor, SPEC has ensured a high level of software support and design activity for successful commercialization of the product in Phase III. At the end of Phase II, SPEC will have demonstrated both a high performance DFT array processor architecture and GaAs RISC design.

References

1. J. W. Cooley and J. W. Tukey, "An Algorithm for the machine calculation of complex Fourier Series" *Math. Comput.* 19 297 (1965).

2. C. M. Rader and N. M. Brenner, "A new principle for fast Fourier transformation" *IEEE Trans. Acoust., Speech, Signal Processing* 24 264 (1976).

3. R. D. Preuss, "Very fast computation of the radix-2, discrete Fourier transform" *IEEE Trans. Acoust., Speech, Signal Processing* 30 595 (1982).

4. P. Duhamel and H. Hollmann, "Existence of a 2^n FFT algorithm with a number of multiplications lower than 2^{n+1}" *Electron. Lett.* 20 690 (1984).

5. M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations" *Signal Processing* 6 267 (1984).

6. M. T. Heideman and C. S. Burrus, "Multiply/add tradeoffs in length-2^n FFT algorithms" in *Proc. IEEE Int. Conf. ASSP*, Tampa, FL, (Apr. 1985).

7. ---, "On the number of multiplications necessary to compute a length-2^n DFT" *IEEE Trans. Acoust., Speech, Signal Processing* 34 91 (1986).

8. S. Winograd, "On computing the discrete Fourier transform" *Math. Comput.* 32 175 (1978).

9. ---, "On the multiplicative complexity of the discrete Fourier transform" *Adv. Math.* 32 83 (1979).

10. C.S. Burrus and P. W. Eschenbacher, "An in-place in-order prime factor FFT algorithm" *IEEE Trans. Acoust., Speech, Signal Processing* 29 806 (1981).

11. R. Kanaresan and P. K. Gupta, "A prime factor FFT algorithm with real valued arithmetic" *Proc. IEEE* 73 1241 (1985).

12. K. M. Cho and G. C. Temes, "Real-factor FFT algorithms" in *Proc. IEEE ICASSP* 634 (1982).

13. Z. Wang, "Fast algorithms for the discrete W transforms and the discrete Fourier Transform" *IEEE Trans. Acoust., Speech, Signal Processing* 32 803 (1984).

14. J. B. Martens, "Recursive cyclotomic factorization - A new algorithm for calculating the discrete Fourier transform" *IEEE Trans. Acoust., Speech, Signal Processing* 32 750 (1984).

15. ---, "Discrete Fourier transform algorithms for real valued sequences" *IEEE Trans. Acoust., Speech, Signal Processing* 32 390 (1984).

16. P. Duhamel and H. Hollmann, "Split-Radix FFT algorithm", *Electron. Lett.* 20 14 (1984).

17. H. V. Sorensen, M. T. Heideman, and C. S. Burrus, " On Computing the Split-Radix FFT" *IEEE Trans. Acoust., Speech, Signal Processing* 34, 152 (1986).

18. P. Duhamel, "Implementation of 'split-radix' FFT algorithms for complex, real, and real-symmetric data" *IEEE Trans. Acoust., Speech, Signal Processing* 34 285 (1986).

19. H. Johnson and C. S. Burrus, "Twiddle factors in the radix-2 FFT" in *Proc. 1982 Asilomar Conf. Circuits Syst., Comput.* 413 (1982).

20. F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform" *Proc. IEEE* 66 51 (1978).

21. K. M. M. Prabhu and H. Renganathan, "Optimum binary windows for discrete Fourier transforms" *IEEE Trans. Acoust., Speech, Signal Processing* 34 216 (1986).

22. R. N. Bracewell, "Discrete Hartley transform" *J. Opt. Soc. Amer.* 73 1832 (1983).

23. ---, "The fast Hartley transform" *Proc. IEEE* 72 1010 (1984).

24. H. J. Meckelburg and D. Lipka, "Fast Hartley transform algorithm" *Electron. Lett.* 21 341 (1985).

25. H. V. Sorensen, D. L. Jones, C. S. Burrus, and M. T. Heideman, "On computing the discrete Hartley transform" *IEEE Trans. Acoust., Speech, Signal Processing* 33 1231 (1985).

26. J. Prado, "Comments on the fast Hartley transform" *Proc. IEEE* 73 1862 (1985).

27. G. E. J. Bold, "A comparison of the time involved in computing fast Hartley and fast Fourier transforms" *Proc. IEEE* 73 1863 (1985).

28. K. H. Tzou and T. R. Hsing, "A study of the discrete Hartley transform for image compression applications" *Proc. SPIE* 534 1985.

29. H. S. Hou, "The fast Hartley transform algorithm" *IEEE Trans. Comput.* 36 147 (1987).

30. S. C. Pei and J. L. Wu, "Split-radix fast Hartley transform" *Electron. Lett.* 22 26 (1986).

31. O. Buneman, "Conversion of FFT's to fast Hartley transforms" *SIAM J. Sci. Stat. Comput.* 7 624 (1986).

32. H. Malvar, "Fast computation of discrete cosine transform through fast Hartley transform" *Electron. Lett.* 22 352 (1986).

33. N. Ahmid, T. Natarajan, and K. R. Rao, "Discrete cosine transform" *IEEE Trans. Comput.* C-23 88 (1974).

34. J. M. Jover and T. Kailath, "A Parallel Architecture for Kalman Filter Measurement Update" *IFAC 9th World Congress, Budapest, Hungary* 1005 (1984).

35. E. E. Swartzlander, Jr. and G. Hallnor, "Fast Transform Processor Implementation" *Proc. IEEE ICASSP*, 25 (1984).

36. H. Renganathan and K. M. M. Prabhu, "Improving FFT Efficiency in High Speed Applications" *Digital Signal Processing - 84*, North-Holland 101 (1984).

37. P. Chow, A. G. Vranesic, and J. L. Yen, "A Pipelined Distributed Arithmetic PFFT Processor" *IEEE Trans. Comp.*, C-32 1128 (1983).

38. M. G. X. Fernando, A. G. Constantinides, and T. E. Curtis, "Considerations for the Hardware Implementation of Fast DFT Algorithms" *Digital Signal Processing - 84*, North-Holland 207 (1984).

39. D. J. Spreadbury and T.M. Rees-Roberts, " Prime Radix Fourier Transform - From Algorithm to Silicon Implementation" *Digital Signal Processing - 84*, North-Holland 279 (1984).

40. J. S. Ward, J. V. McCanny, D. Phil, and J. G. McWhirter "Bit-Level Systolic Array Implementation of the Winograd Fourier Transform Algorithm" *IEE Proc.* 132 473 (1985).

41. G. Bongiovanni, "Two VLSI Structures for the Discrete Fourier Transform" *IEEE Trans. Comp.* C-32 750 (1983).

42. G. Bongiovanni, "A VLSI Network for Variable Size FFTs" *IEEE Trans. Comp.* C-32 756 (1983).

43. W. Siu, M. Phil, and A. G. Constantinides, "Very Fast Discrete Fourier Transform using Number Theoretic Transform" *IEE Proc.* 130 201 (1983).

44. B. Arambepola, "Discrete Fourier Transform Processor based on the Prime Factor Algorithm" *IEE Proc.* 130 138 (1983).

45. T. Wiley, R. Chapman, H. Yoho, T. S. Durrani, and D. Preis, "Systolic Implementations for Deconvolution, DFT and FFT" *IEE Proc.* 132 466 (1985).

46. C. D. Thompson, "Fourier Transforms in VLSI" *IEEE Trans. Comp.* C-32 1047 (1983).

47. G. A. Doodlesack, M. Gray, B. L. Johnson, C. L. Nowacki, and J. J. Vaccaro, "VLSI Chip Design for QRNS-Based DFTs" *IEEE* 1118 (1987).

48. J. G. Delgado-Frias and I. Contreras, "WASP - A WSI Bit-Serial Processor" *IEEE* 164 (1987).

49. F. F. Yassa, J. R. Jasica, R. I. Hartley, and S. E. Noujaim, "A Silicon Compiler for Digital Signal Processing: Methodology, Implementation, and Applications" *Proc. IEEE* 75 1272 (1987).

50. J. T. Burkley, "MPP VLSI Multiprocessor Integrated Circuit Design" *PE Circuit Design* (unknown).

51. S. G. Morton and E. Abreu, "The Dynamically Reconfigurable CAP Array Chip I" IEEE J. Solid-State Circuits SC-21 820 (1986).

52. S. G. Smith, M. S. McGregor, and P. B. Denyer, "Techniques to Increase the Computational Throughput of Bit-Serial Architectures" *IEEE* 543 (1987).

53. J. V. McCanny, D. Phil, and J. G. McWhirter, "Bit-Level Systolic Array Circuit for Matrix Vector Multiplication" *IEE Proc.* 130 125 (1983).

54. R. Gnanasekaran, "On a Bit-Serial Input and Bit-Serial Output Multiplier" *IEEE Trans. Comp.* C-32 878 (1983).

55. P. M. Chau, K. C. Chew, and W. H. Ku, "A Bit-Serial Floating-Point Complex Multiplier-Accumulator for Fault-Tolerant Digital Signal Processing Arrays" *IEEE* 483 (1987).