BIOACOUSTICS FAST FOURIER TRANSFORM PROCESSOR

BY

DAVID CRAIG RHODES

B.S., University of Illinois, 1979

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1980

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

May, 1980

WE HEREBY RECOMMEND THAT THE THESIS BY

DAVID CRAIG RHODES

ENTITLED BIOACOUSTICS FAST FOURIER TRANSFORM PROCESSOR

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

Michael Schlansker
Director of Thesis Research

A. W. Deerman, Jr
Head of Department

Committee on Final Examination†

Chairman

† Required for doctor's degree but not for master's.

O-517

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The goal of this project has been to design a processor, capable of efficiently executing the discrete Fourier transform (DFT) via the fast Fourier transform computation (FFT) [1]. From the computer systems viewpoint, the fast Fourier transform processor (henceforth referred to as the FFTP) is an auxiliary processor which takes commands and data from the system host computer, performs the FFT, and returns the data to the host. The target system is the Interdata 7/32 computer system at the Bioacoustics Research Laboratory in the Electrical Engineering Annex at the University of Illinois. This system is expected to be utilized in digital picture processing of data from a scanning laser acoustic microscope and from an acoustical scanner (UCATS) which produce acoustic images of biological material. Many of the computations involved in processing these images, take the form of DFT's, inverse DFT's (IDFT), or digital convolutions. In practice, these computations, which are performed via the FFT, can consume large amounts (perhaps hours) of computer time. The goal is to utilize the FFTP to perform these computational burdens, thereby reducing the time spent doing FFT's by a factor of one thousand to two thousand.

II. BACKGROUND AND GENERAL DESCRIPTION

A. FFT Introduction

A very brief introduction to the FFT is given here in order to clarify notation. A complete treatment can be found by Brigham [2]. The DFT is defined by:

$$F(k) = \sum_{n=0}^{N-1} f(n)\, e^{-j2(pi/N)nk} \quad ,k = 0, 1, \ldots, N-1.$$

Using the notation

$$W_N^{nk} = e^{-j2(pi/N)nk} .$$

we have:

$$F(k) = \sum_{n=0}^{N-1} f(n)\, W_N^{nk} \quad ,k = 0, 1, \ldots, N-1,$$

where k is the digital frequency, n is the discrete time at which sampling occurs, and N (sequence length) is the number of data samples (complex in general). This defines the transformation of f(n) into F(k). The DFT is typically computed via the FFT. The FFT performs $(N/2)\log_2 N$ computational entities called "butterflies" of the form:

$$ASUM = (A + BW_N^k) \quad \text{and} \quad BSUM = (A - BW_N^k)$$

The functional diagram is shown in figure 1, and the equivalent notation is shown in figure 2. The FFT computation of a complete 8-point "decimation-in-time"

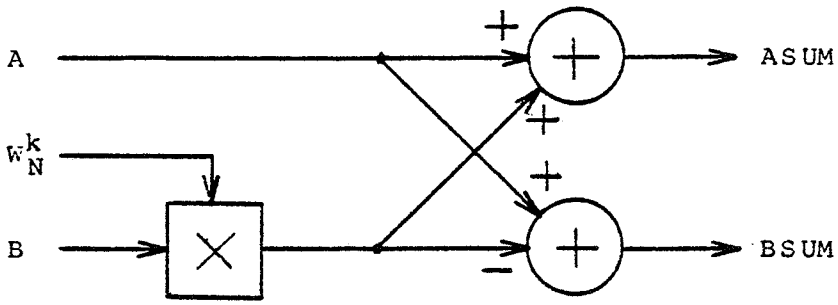


Figure 1. Butterfly Functional Diagram

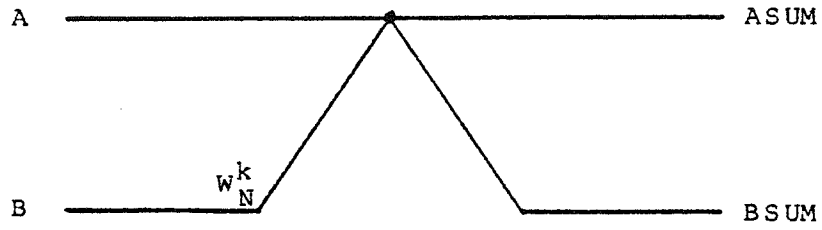


Figure 2. Butterfly Symbolic Notation

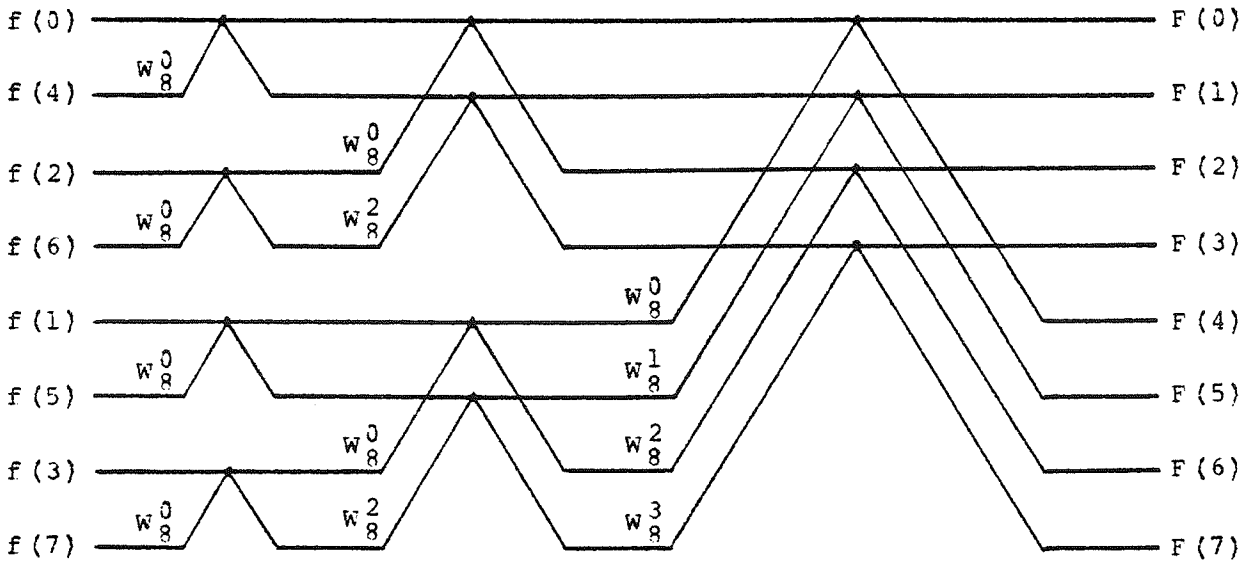


Figure 3. Eight-Point Decimation-in-Time FFT

transform is shown in figure 3. Each column contains N/2 butterflies and will be referred to as a "log slice". Each log slice is composed of blocks of butterflies which will be referred to as "skip slices". Thus the first log slice of figure 3 contains four skip slices, the second log slice contains two skip slices, and the last log slice contains one skip slice. In general, there are N/2 skip slices in the first log slice, N/4 skip slices in the second log slice, and so forth until the last log slice which is composed of a single skip slice. Note that the indices of f(n) are not in natural order in figure 3. This phenomenon is referred to as "bit reversal" because the proper index can be obtained by reversing the binary representation of the naturally ordered index.

In order to transform from F(k) back to f(n) we utilize the IDFT which is defined by:

$$f(n) = 1/N \sum_{k=0}^{N-1} F(k)\ e^{+j2(pi/N)nk} \qquad ,n = 0,\ 1,\ \ldots,\ N-1.$$

$$= 1/N \sum_{k=0}^{N-1} F(k)\ W_N^{-nk} \qquad ,n = 0,\ 1,\ \ldots,\ N-1.$$

We have a useful identity which allows computation of the IDFT via the DFT, with the exception of the 1/N factor.

$$f(n) = 1/N\ [\sum_{k=0}^{N-1} F^*(k)\ W_N^{nk}\ ]^* = 1/N \sum_{k=0}^{N-1} F(k)\ W_N^{-nk}\ ,$$

$$n = 0,\ 1,\ \ldots,\ N-1,$$

where "*" indicates the complex conjugate. This allows the same hardware which performs the DFT to be used for computing the IDFT, without additional logic.

B. System Design

The FFTP can be a general purpose, stand-alone processor, or it can be a slave unit which is dependent on a host computer. The general purpose processor has the advantage that it can perform data manipulations and decision-making tasks, independently from the rest of the system. This allows for flexibility in terms of allowable operations that can be performed. The second alternative is to provide an FFTP that requires a host computer to provide commands and decision-making capability. This general configuration has been discussed by Peled [3] and Kobylinski, et. al. [4] in a microprocessor environment. The advantage is that the FFTP is much simpler (thus cheaper). The tasks in signal processing are divided, rather naturally, into the "number crunching" portion performed by the FFTP, and the decision-making performed by the host computer. Computers that fall into the catagory of the general purpose signal processor include the LX-1 [5] and the LSP/2 [6]. Computers that fall into the catagory of host-driven processors include the FDP [7] and the SPS of Peled. Several of the commercially available signal processors, such as Floating Point System's AP120B, Signal Processing System's SPS-21, CSPI's MAP, and Honeywell's XAP, can function in either mode

of operation. Another trend is the use of a single, highly parallel-pipelined butterfly unit as opposed to several butterfly units operating in parallel. That is, the expense and effort is concentrated in producing a single, very fast unit which performs each butterfly in turn. Other parallel and serial trade offs in FFT hardware are discussed by Bergland [8].

The general design for the system will follow the suggestions of Peled and Kobylinski, et. al., for a host computer which performs the decision making tasks, and a high speed auxiliary processor to perform the cumbersome algebraic calculations of the FFT. This design concept fits well with the existing system at the Bioacoustics laboratory. The Interdata 7/32 provides a natural host processor since it is being interfaced for data gathering from the scanning laser acoustic microscope and the UCAT scanner. The addition of the FFTP (fast Fourier transform processor) will greatly enhance the speed in performing FFT's and will be less expensive than an additional general purpose processor (perhaps an array processor) which would duplicate some of the features of the Interdata 7/32. The remainder of this paper will discuss the design and verification of the FFTP.

C. Design Objectives

The primary objective in this design is the speed of computation, though we must necessarily be aware of the trade offs in performance versus cost. Fixed point, two's

complement, fractional representation of data will be assumed, since this is well suited to the FFT computation and the design goal of high speed. The work of Aiso, et. al. [9] indicates that a fixed point calculation utilizing a 16 bit word is sufficient for most calculations when eight bits of precision are available for the input sequence (as is our case). These results hold for applications where transform length is up to 8K data points. In this laboratory, transforms of up to 2K (2048) data points of eight bit words are currently envisioned, as well as two dimensional transforms. With this in mind, the initial design will utilize a 16 bit word width as the "median" word width, with the simulation providing the capability of varying the word width up to a maximum of 32 bits. One of the goals of this simulation is to determine the proper bit width necessary for this application. An initial maximum transform length of 4K (4096) data points will be utilized. This provides sufficient length for current use with some room for future expansion. The initial design goal for computational speed is one butterfly per 250 nanoseconds, which provides good compatibility with the Interdata 7/32 interface as will be discussed later. Finally, for ease of construction, we would prefer to use stock components rather than exotic arrangements of logic elements. (That is, multiplier chips rather than large arrays of adders.)

D. Architectural Block Design

The FFTP is divided into three architectural blocks. The control block is responsible for generating the memory addresses of operands for use in computing butterflies. The control block may have varying degrees of microprogramming complexity depending on the implementation. The second architectural block is the fast Fourier transform unit (FFTU) which performs the algebraic computations (namely butterflies) to produce the proper transform. The design will utilize a single, very fast, internally parallel butterfly unit, as opposed to implementing several separate butterfly units which operate in parallel. This substantially reduces the complexity of control as well as improving the performance versus cost ratio. The notation AR, AI, BR, BI, will refer to the real and imaginary components of the A operand and B operand. ARSUM, AISUM, BRSUM, BISUM will refer to the real and imaginary results of the butterfly. COS and SIN refer to the real and imaginary components of W, respectively (the subscript and superscript have been dropped). In order to calculate a single butterfly, the FFTU must compute:

$$ASUM = (A + (B)(W)) \text{ and } BSUM = (A - (B)(W))$$

with

$$(B)(W) = (BR + j\ BI)(COS + j\ SIN)$$

$$= [(BR)(COS) - (BI)(SIN)] + j [(BR)(SIN) + (BI)(COS)]$$

$$= BRP + j\ BIP$$

where BRP and BIP refer to the real and imaginary components of (B)(W). Finally, the memory block provides data storage of the input sequence, storage of partial results, storage of the final transformed data, and storage of trigonometric coefficients used in the calculation. Interfacing between these architectural blocks is provided via address busses and data busses.

Four general architectures were considered for the FFTP, based on the implementation complexity of the various architectural blocks. The first architecture (figure 4) is a microprogrammed processor with a high speed multiplier which performs a multiply in a single microinstruction cycle. In particular, consideration was given to the Advance Micro Devices AMD 2900 family of microprocessors [10], with a TRW multiplier [11]. The FFT computation would be accomplished by initiating a microinstruction sequence through a single instruction. The computation would proceed sequentially, much like a highly tuned general processor. A microinstruction cycle time of 200 nanoseconds is reasonable, with a single operation (add, multiply, shift, etc.) occuring at each cycle. The microprogram sequencing becomes the control block, and the FFTU block becomes incorporated in the data manipulation. For each butterfly, there are four

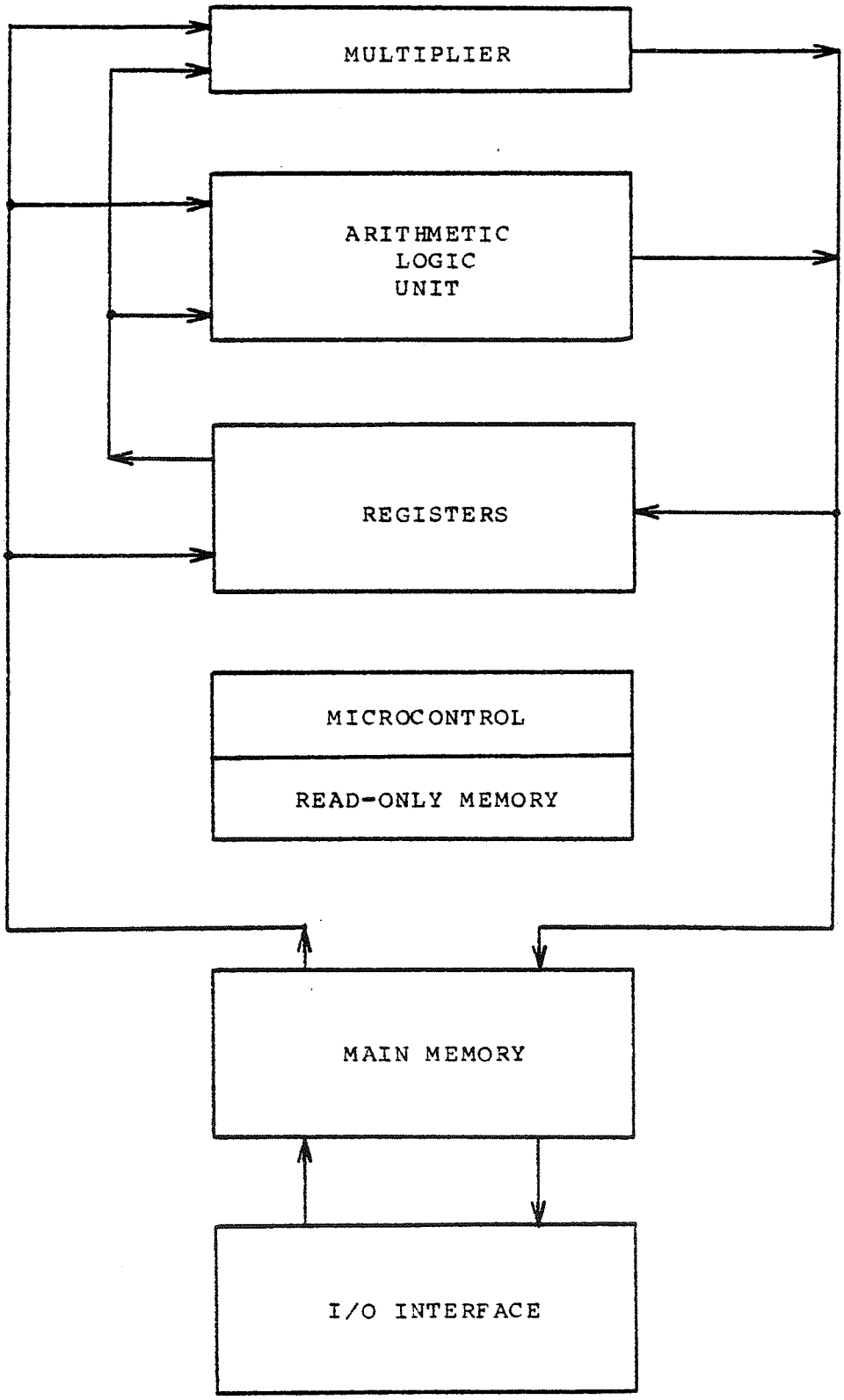


Figure 4. General Purpose Microprogrammable FFTP

real multiplications and six additions (or subtractions) to be accomplished. If only computational overhead is considered (neglecting the overhead for bit reversal), we could reasonably expect a butterfly every 10 microinstruction cycles, or one butterfly every 2 microseconds. (This assumes that trigonometric coefficients are pre-stored so that they need not be computed.) While this is a "reasonable" speed, it does not meet our design goal of one butterfly per 250 nanoseconds. If, in addition to the microprogrammed FFT's, we add microprogramming for general instructions, the architecture becomes a truly general purpose processor. The advantage would be flexibility (other operations such as raw sums of products could be computed). The disadvantages are the lack of speed and the redundance of operations which can already be performed by the Interdata 7/32.

The second architecture considered (figure 5) uses a "half butterfly" calculation in the FFTU. The real and imaginary parts of a full butterfly can be split into two computationally identical halves, each requiring two multiplications and three additions. The control section consists of a microsequencer with several hardware registers, and the FFTU is the "half butterfly" unit. The control section must generate six addresses for each full butterfly (two operands, two trigonometric coefficients, and two resultant locations). This requires four memory references per cycle (assuming real and imaginary data are stored in two-way interleaved memory). In order to provide good

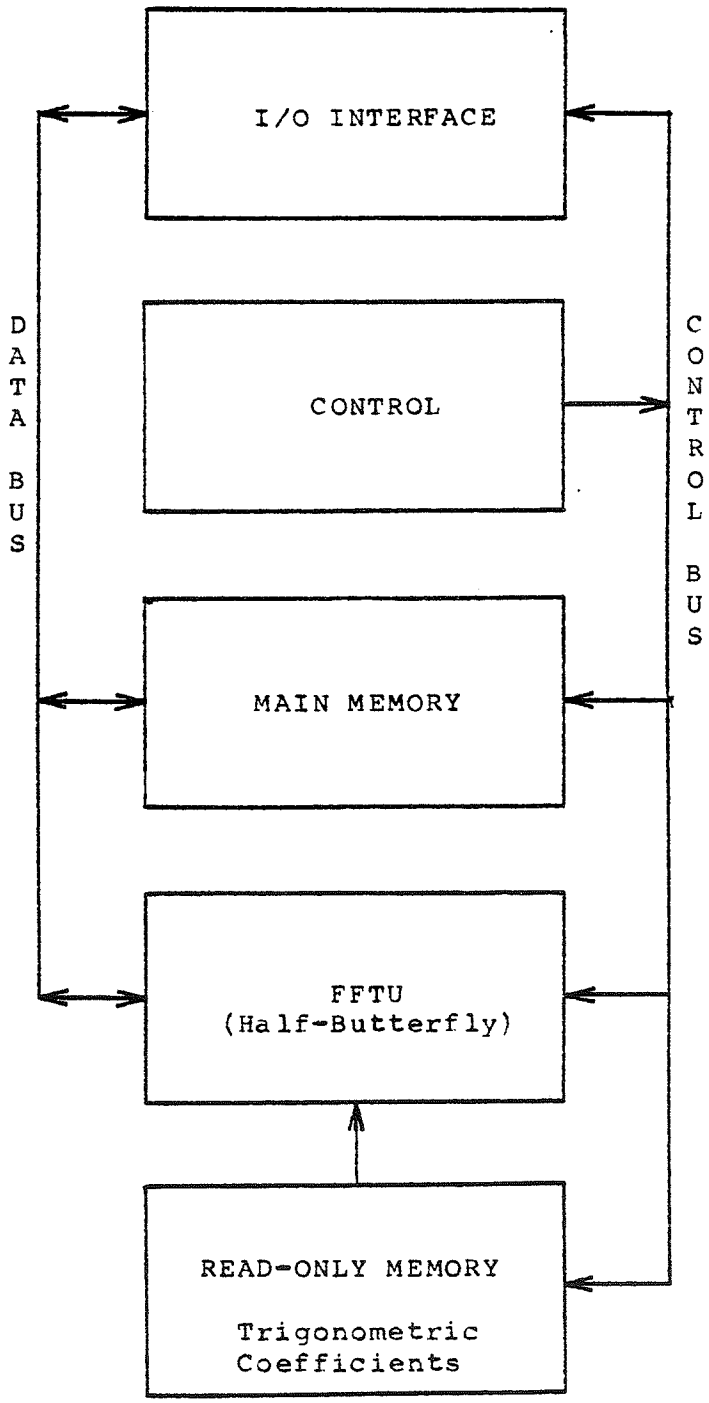


Figure 5. Half-Butterfly Architecture

compatibility we need the FFTU (half butterfly) to produce two passes for every four main memory cycles. Later, in the analysis of the full butterfly architecture, we will see that four memory cycles for each pass of the FFTU provides a better timing match.

The third general architecture utilizes a full butterfly calculation in the FFTU. As previously shown, this can be accomplished with four multipliers and six adders. Figure 6 shows the functional elements of the FFTU. The control section is similar to the half-butterfly implementation in that six addresses must be generated for each full butterfly, though in this case a savings results because no provision need be made to accommodate partial results. With this arrangement, a single pass of the FFTU must coincide with four main memory references, (two operand retrievals and two resultant stores), and two trigonometric coefficient retrievals.

The fourth general architecture considered, utilizes a full butterfly unit for the FFTU, but with three multipliers and nine adders. The computational elements are shown in figure 7. This is accomplished by algebraically manipulating the derivation of BRP and BIP:

$$T = (BR + BI)(COS)$$

$$BRP = T - BI(SIN + COS) = (BR)(COS) - (BI)(SIN)$$

$$BIP = T + BR(SIN - COS) = (BR)(SIN) + (BI)(COS)$$

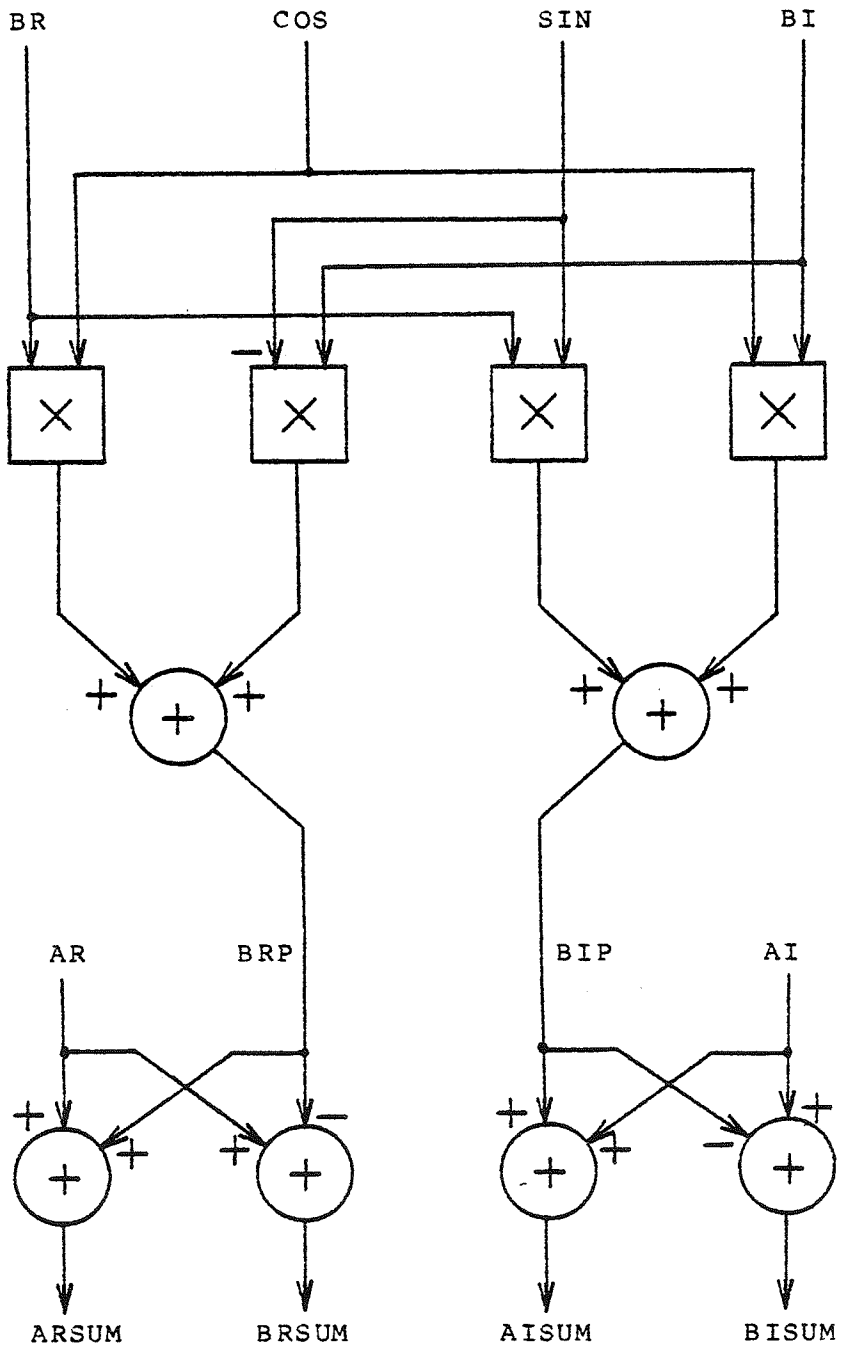


Figure 6. Four-Multiplier Full Butterfly

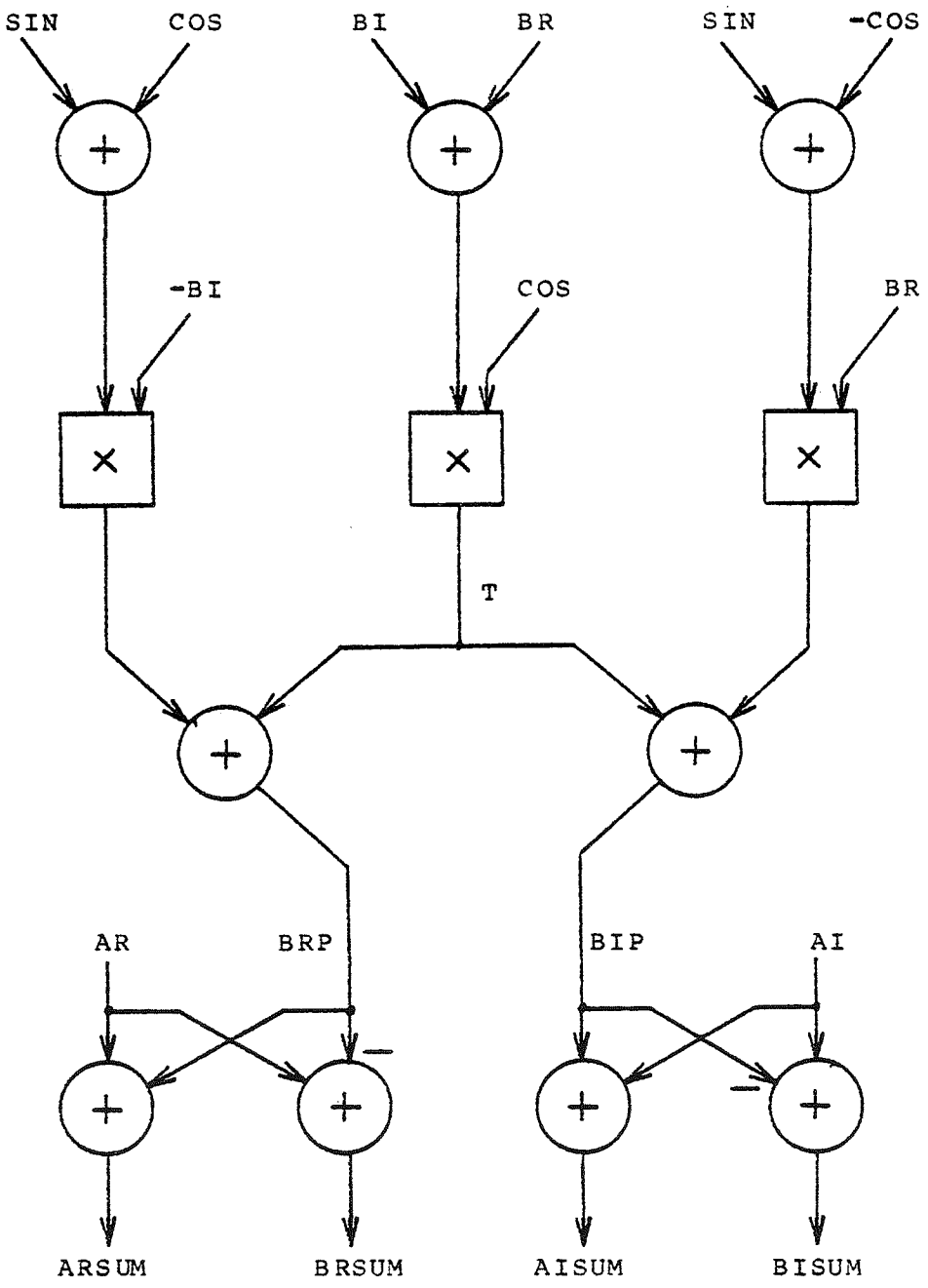


Figure 7. Three-Multiplier Full Butterfly

The reason for utilizing this method is that a full butterfly can be computed with three multipliers rather than four. This can be important since multipliers are usually expensive relative to the cost of adders. The penalty, as can be seen by comparing figure 6 with figure 7, is that fewer multipliers results in another stage of additions (vertical distance). This results in an increased time for the operands to propagate through the FFTU. Fixed point fractional representation of data also presents a problem, since the initial addition of SIN + COS, or BR + BI, sometimes results in overflow of the adders.

By comparing the propagation delays of commercially available TTL logic chips, it appears that the FFTU (rather than the memory) will be the bottleneck in achieving a high speed butterfly. For this reason, the full butterfly with four multipliers has been chosen for implementation of the FFTU. At this point, it also appears promising to consider a 200 nanosecond butterfly, which exceeds the initial goal of 250 nanoseconds. Having made this choice for the general architecture, we now proceed with the more detailed design of the control, memory, and FFTU blocks.

III. DETAILED HARDWARE DESIGN

A. Memory

The memory must have sufficient space to store the maximum allowable data sequence length. We have assumed a maximum of 4K complex data points, which requires 8K (8192) memory words of storage. It is natural to use two-way interleaving, thus the real and imaginary components are stored in identical addresses in separate memory banks. The fetching and storing of real and imaginary components can then occur simultaneously, utilizing the same address bus. The memory used to store trigonometric coefficients can be read-only memory (ROM). In computing the FFT, we require N/2 sine coefficients and N/2 cosine coefficients. We need, however, store only N/4 total coefficients, since there are N/4 distinct magnitudes of sine and cosine components combined. This is accomplished in a manner similar to the method of Agrawal and Ninan [12]. In order to use the reduced storage requirement, we must compute the proper index for the sine and cosine, as well as determine the sign of the cosine. (The sine is always negative.) Due to the constraints of representing two's complement fractions (as will be explained in the FFTU design), the negative rounded value of the sine (true value) will be stored, with addresses zero through 1023 corresponding to SIN(-0) through SIN(-(pi)1023/1024) respectively. (That is, SIN(-k(pi)/1024) where k = 0,1,2,...,1023 is the memory address.)

The memories are interfaced to the rest of the FFTP via address busses and data busses. Figure 8 shows the memories and their connections to the various busses. The interleaving of the real and imaginary data memories is straight forward and requires no further explanation. The multiplexors on DRBUS and DIBUS are used for scaling the data by one-half at each log slice to provide 1/N scaling over the complete FFT computation. This feature (provided as an externally controlled command option) is provided for use in preventing overflow, as well as allowing for normal scaling in the IDFT computation. To choose 1/N scaling, the LSB of COMND is set true when the command is sent. The trigonometric coefficient ROM is modelled as a single memory with double addressability, though this is not required in the actual implementation. The multiplexors on TSBUS and TCBUS are used to place the value "minus one" on the bus when the out-of-bounds address is detected by the address generation in the control section. Table 1 shows the timing of events for storage and retrieval of data during the computation of the FFT.

B. Control

The control section must generate the addresses for the operand retrieval and storage as well as the addresses for trigonometric coefficients. These two sections can be considered as independent units which are tied together by the timing mechanisms. Timing is derived from an eight phase

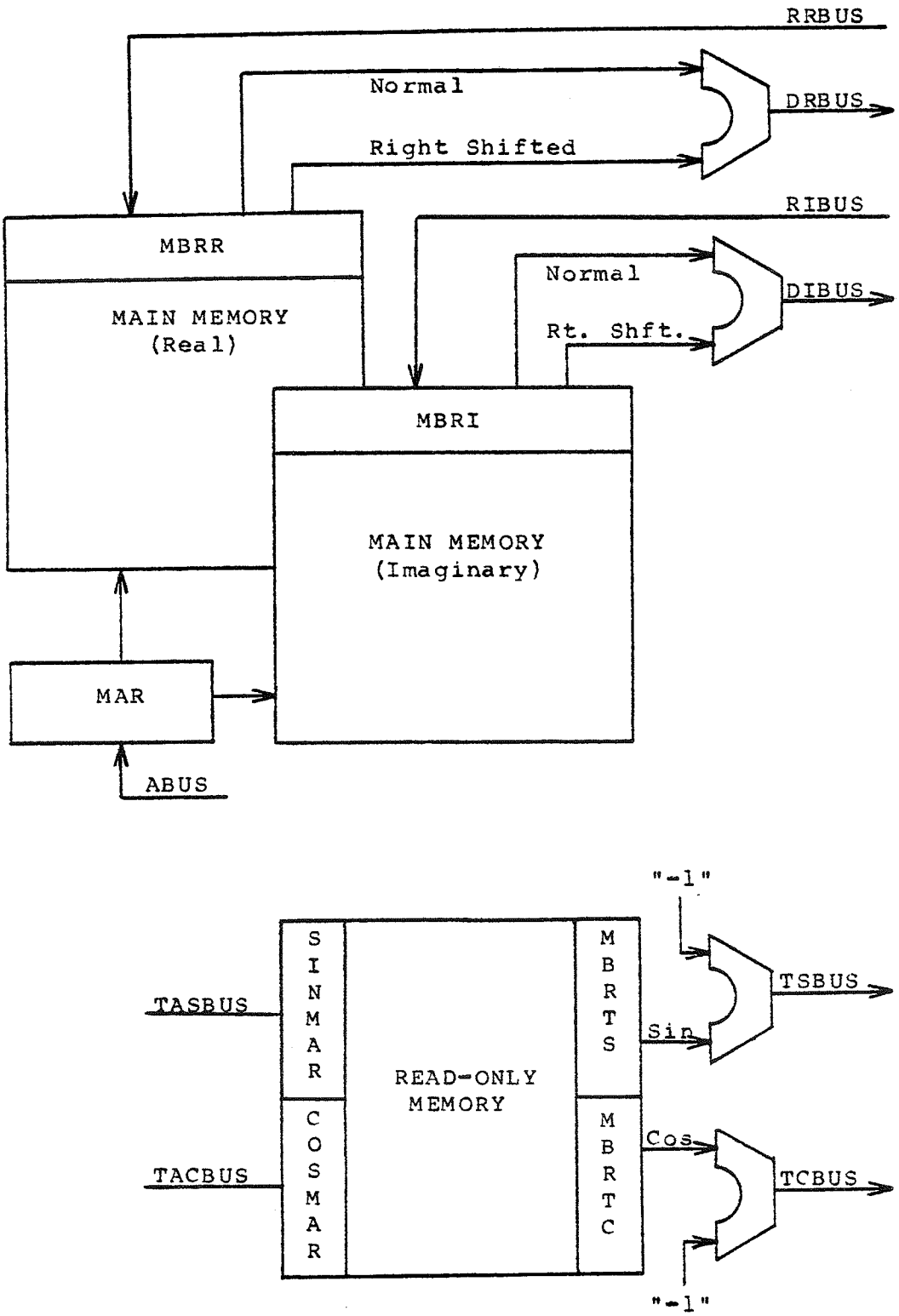


Figure 8. Memory Section

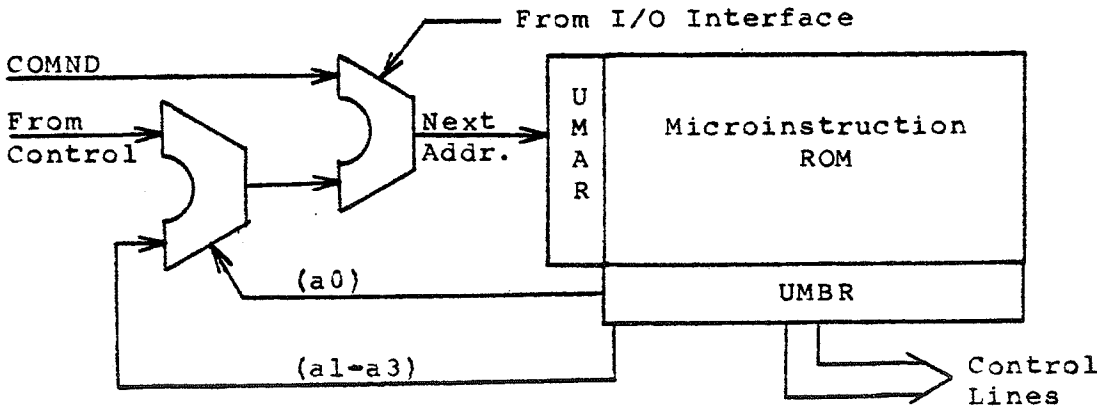
Table 1. Memory Timing Cycle

TIME	DATA RETRIEVAL	DATA STORAGE
t7	R2 to ABUS	MBR to MEM
t0	ABUS to MAR	
t1	MBR to DRBUS,DIBUS	R2P to ABUS
t2		ABUS to MAR RRBUS,RIBUS to MBR
t3	R1 to ABUS	MBR to MEM
t4	ABUS to MAR	
t5	MBR to DRBUS,DIBUS	R1P to ABUS
t6		ABUS to MAR RRBUS,RIBUS to MBR
t7	R2 to ABUS	MBR to MEM

clock (or perhaps a four phase clock where positive and negative edges are considered as timing signals). The times are labelled t0 through t7. The occurrence of a timing mark coupled with a microcontrol signal from the microsequencer causes execution and synchronization of events in control, memory, and the FFTU.

1. Microsequencer

In order to meet speed requirements, the microsequencer must produce operand addresses for one complete butterfly each clock cycle. With this speed constraint, the microsequencer could utilize one microinstruction for each of the butterflies. The simplicity of this approach bears a high cost for large microcontrol memory as well as the need for a separate microsequence for each data sequence length to be computed (roughly 45000 words of 60 bits each). The advantage is that all addresses are precomputed, so that no additional control logic is required. By introducing registers to indicate the current disposition of the FFT computation (that is, the log slice and skip slice) and using this information to compute the operand addresses, the microcontrol memory size can be drastically reduced. After several refinements, where registers replace redundant microinstructions and other registers hold necessary information to compute the addresses of operands, the microcontrol memory has been reduced to 8 words. The microsequencer as shown in figure 9, produces the control signals for proper operation of the FFTP. When idle, the IDLE microinstruction is present in the microcontrol memory buffer register (UMBR). The COMND register (command), is a 16 bit register which receives a 3 bit command and a 13 bit operand indicating sequence length. Upon receipt of a new command from the external I/O bus, the 3 bit command field is loaded into the microcontrol memory address register (UMAR)



Microsequencer

	ADDR	CONTROL														
	a0-a3	c1-c30														
INIT	0XXX	0	1	1	10	10	11	10	10	10	10	10	01	00000	10	10
NORMAL	0XXX	0	0	0	01	01	01	01	00	01	00	01	10	11110	00	01
SKIPSLICE	0XXX	0	0	0	01	01	01	01	00	10	00	01	11	11110	00	10
LOGSLICE	0XXX	0	0	0	10	01	11	01	01	10	01	10	01	11110	01	10
FINI	1100	0	0	0	01	01	01	01	00	00	00	00	00	11110	00	00

	MEMORY	FFTU	I/O INTERFACE
	m1-m7	f1-f5	i1-i4
INIT	1100111	11011	0000
NORMAL	1111111	11111	0000
SKIPSLICE	1111111	11111	0000
LOGSLICE	1111111	11111	0000
FINI	1011111	00111	0000

Figure 9. Microsequencer and Microinstruction Format.

to cause the fetch of the first instruction. This instruction will be one of three possible choices. RECEIVE will load a new sequence from the host computer, SEND will return the data to the host computer, or INIT will initiate performance of the FFT calculation. RECEIVE and SEND are single microinstruction commands, and remain in the UMBR until the completion of the operation is indicated. For the FFT computation, INIT is the first instruction called. This sets up initialization of the control unit and proceeds with the calculation by calling SKIPSLICE, LOGSLICE, and NORMAL microinstructions. At the end of the FFT's computation, the FINI microinstruction is called to complete the storage of the final butterfly results. The function of these microinstructions will become more clear in the explanation of the control, memory, and FFTU sections. The microinstruction word is 50 bits wide, consisting of several fields as shown in figure 9. The leftmost bit, a0, controls the source of the next microinstruction address. If a0 is set, the next group of three bits, labelled a1, a2, and a3, is the source for the next microinstruction address. If a0 is clear, the address comes from the COMND register to initiate a command, or from the hardware controller if a computation is in progress. The fields labelled c1 through c30, m1 through m7, and f1 through f5, are used in the control section, memory section, and FFTU section, respectively. The last field, labelled i1 through i4, is used to control the external I/O operations. Table 2 shows

Table 2. Microcontrol Word Definition

MICROCONTROL BIT	TIME	CONTROL
c1	t0	Load COMND register.
c2	t1	Load R0.
c3	t1	Load SCALE.
c4	t7	Clear R1.
c5	t7	Load R1.
c6	t7	Clear R1P.
c7	t7	Load R1P.
c8	t3	Clear R2.
c9	t5	Load R2.
c10	t5	Clear R2P.
c11	t5	Load R2P.
c12	t2	Load R3.
c13	t2	Left Shift R3.
c14	t4	Load R4.
c15	t4	Decrement R4.
c16	t2	Clear R5.
c17	t2	Left Shift R5.
c18	t4	Load R6.
c19	t4	Decrement R6.
c20,not c21	t3	RADD = R0.
not c20,c21	t3	RADD = R3.
c20, c21	t3	RADD = R0 + R3.
c22	t3	R1 enable to ABUS.
c23	t7	R2 enable to ABUS.
c24	t5	R1P enable to ABUS.
c25	t1	R2P enable to ABUS.
c26	t3	Bit rev. R1 to ABUS.
c27	t2	Load R5P.
c28	t2	Right Shift R5P.
c29	t4	Clear RTRIG.
c30	t4	Load RTRIG.
m1	t0,2,4,6	ABUS to Memory MAR.
m2	t1,t5	MBR to DRBUS,DIBUS.
m3	t2,t6	RRBUS,RIBUS to MBR.
m4	t3,t7	Store MBR.
m5	t1	ROM MBR's to TSBUS,TCBUS.
m6	t6	Load ROM MAR.
m7	t7	Load ROM MBR.
f1	t2	Load BR,BI,COSR,SINR.
f2	t6	Load AR, AI.
f3	t1	Ld ARSUM,BRSUM,AISUM,BISUM.
f4	t2	BRSUM,BISUM to RRBUS,RIBUS.
f5	t6	ARSUM,AISUM to RRBUS,RIBUS.

the microinstruction bits and their associated control functions and activation times, for the control, memory, and FFTU sections. A list of the five microinstruction words used to perform the FFT is included in figure 9.

2. Hardware Controller (Data)

The data addresses for FFTU operands are generated by using the COMND register (Command), R0 through R7, R1P, R2P, and adders RADD, R1ADD, and R2ADD. RF7 is a fictitious register that is not used in the actual implementation, but is used in the simulation. All registers except COMND and R5 are 12 bits. The configuration is shown in figure 10. COMND is a 16 bit register which receives a 3 bit command and a 13 bit operand indicating the length of the sequence (always a power of two). The data sequence is initially stored in memory in bit reversed order (binary representation of the address is reversed). The storage address is generated by storing sequential data in the bit reversed address from a 12 bit up-counter. The counter is implemented by loading R0 = 1 which passes through RADD to R1ADD. R1 will then increment by one and place its bit reversed value on the address bus. This results in performing the bit reversal completely transparently, but also results in gaps between data if the sequence is less than 4K in length. R0 receives the 12 bit right shifted, then bit reversed operand of COMND. R0 then contains the distance of the gaps between the elements of the data sequence. The value of R3 is initially loaded from R0,

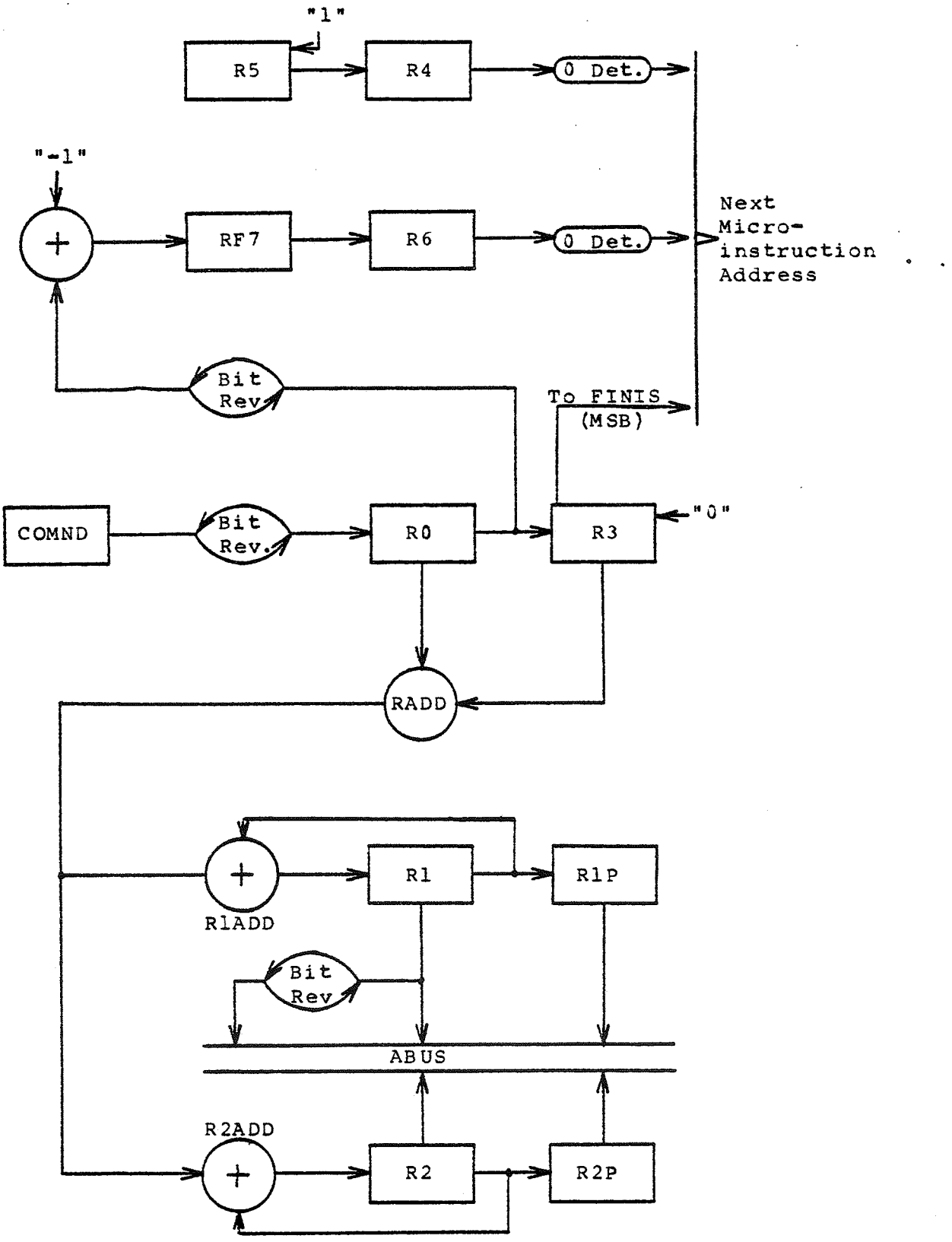


Figure 10. Control Section (Data Addresses)

and then is left shifted (doubled) for each log slice. R3, then, contains the initial "B" operand address at the beginning of each log slice. RADD has the ability to output R0, R3, or their sum, under microinstruction control. This value is sent to R1ADD and R2ADD for updating of R1 and R2. R1 and R2 contain the addresses of the "A" and "B" operands, respectively (real and imaginary components). These addresses are then loaded to R1P and R2P so that the resultant data can be stored upon completion of the butterfly. While executing butterflies within a single skip slice (NORMAL microinstruction), the addresses (R1 and R2) increment by R0 to pick sequentially stored data points. R5 contains the number of butterflies which make up a skip slice. This number remains constant for each log slice. R4, which is loaded from R5 at the beginning of each skip slice, is a decrementing counter which counts the butterflies. The end of a skip slice is indicated when R4 = 0 is detected. This condition forms a partial address for the next microinstruction fetch. This microinstruction (a SKIPSLICE microinstruction) causes RADD to output the sum of R0 and R3 which is subsequently added to the current values of R1 and R2 to form the addresses of the first operands in a new skip slice. R6 is initially loaded with the bit reversed minus one value of R0. This is (N/2) - 1 and is the number of butterflies remaining to complete a log slice. When R6 = 0 is detected, this forms the partial address for the next microinstruction (LOGSLICE microinstruction), which will

initiate a new log slice. A new log slice requires reloading R6 (same as initial value), left shifting R3, and loading R2 from R3 to form the first "B" operand address of the new log slice. Note that a new log slice is also a new skip slice, so that R4 (the remaining butterflies to complete a skip slice), must be loaded from the new value of R5. The end of the computation is indicated when R3 overflows. This condition is detected by the FINIS flip-flop (essentially the thirteenth bit of R3) which causes the next microinstruction cycle to fetch the FINI microinstruction.

3. Hardware Controller (Trigonometric)

The trigonometric coefficient addresses are generated by the 12 bit register R5P, the 11 bit register RTRIG, the adders TADD, and COSADD, and a negator, as shown in figure 11. Referring to figure 3, we see that the trigonometric coefficients form a repeated pattern in each skip slice within a given log slice. This information is computed from R5P which contains the distance between addresses of the trigonometric coefficients for a given log slice. R5P is initially loaded to 2048 (MSB set) and right shifted at the end of each log slice. RTRIG is cleared at the beginning of each skip slice so that it contains the address of the first sine coefficient. The lower 10 bits of COSADD form the cosine address which is always the sine address minus 1024. The first cosine coefficient address will be 1024 which does not fall within the range of addressability of the

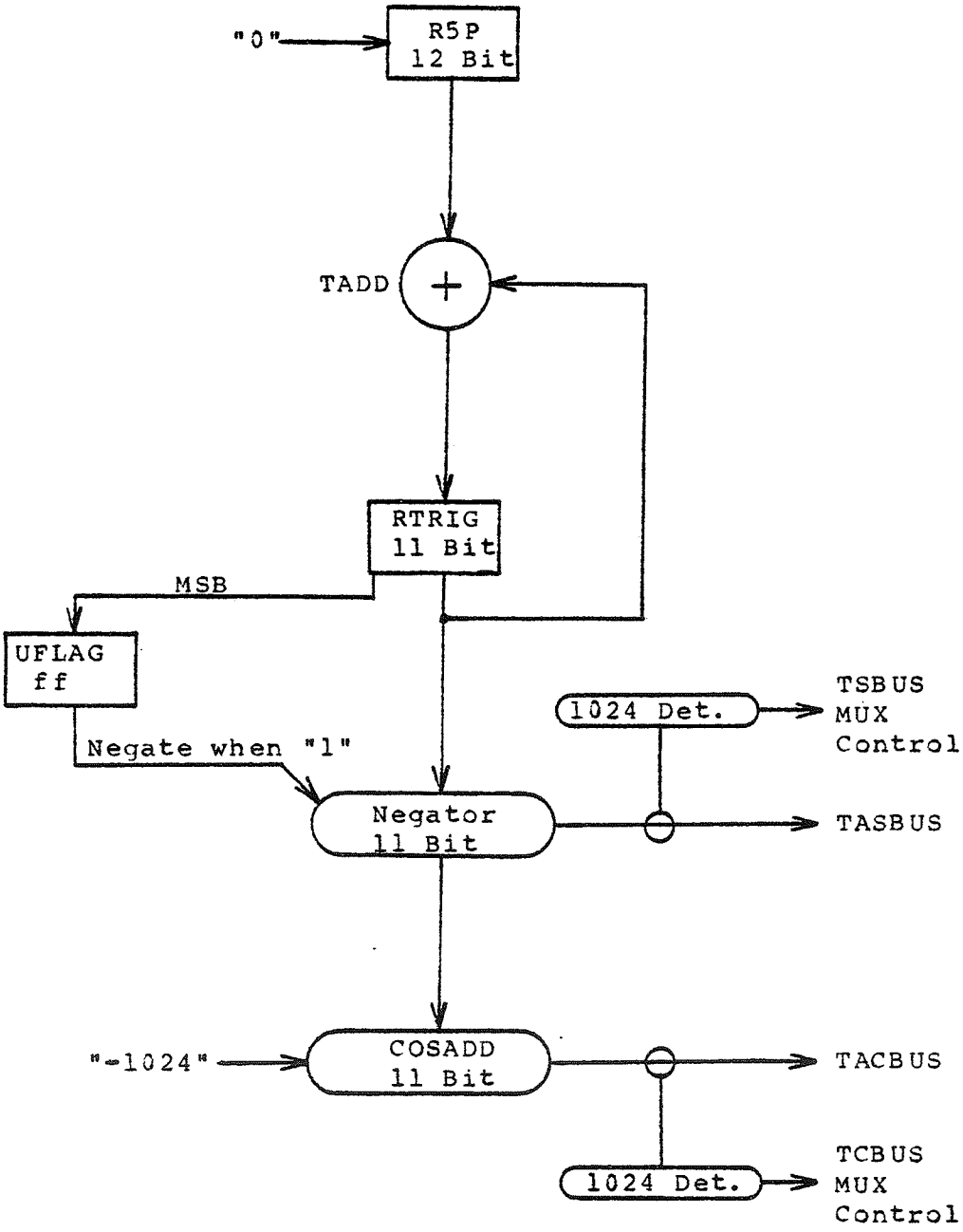


Figure 11. Control Section (Trigonometric Addresses)

coefficient ROM. This condition must be detected (for sine and cosine) and must cause the value "minus one" to be placed on the data line for the appropriate coefficient. The value of RTRIG may be greater than 1024. This condition is detected by monitoring the MSB of RTRIG, which sets UFLAG when true. When this occurs, the proper sine coefficient address is generated by negating RTRIG and using the lower 10 bits to address the coefficient ROM. The cosine coefficient is still generated by taking the sine address minus 1024 and using the lower 10 bits for the address. The actions and times at which updating occurs in the control section are shown in table 3, for the INIT, NORMAL, SKIPSLICE, and LOGSLICE microinstructions.

C. Fast Fourier Transform Unit

The FFTU performs a full butterfly calculation utilizing registers BR, BI, COSR, SINR, AR, AI, ARSUM, AISUM, BRSUM, BISUM, multipliers MP1, MP2, MP3, and MP4, with their associated output registers MR1, MR2, MR3, and MR4, and adders ADD1, ADD2, ADD3, ADD4, ADD5, and ADD6. BRP and BIP are fictitious registers that are not used in the actual implementation, but are used in the simulation. Figure 12 shows the flow of data and the times at which data must be present at various levels in order to complete the butterfly in one clock cycle. Initially, data enters BR, BI, COSR, and SINR at time t2. Recall that the trigonometric coefficient is a negative number. This is always the correct value for

Table 3. Control Section Timing

	MICROINSTRUCTION TYPE			
TIME	INIT	NORMAL	SKIPSLICE	LOGSLICE
t1	Set UFLAG	Set UFLAG	Set UFLAG	Set UFLAG
t2	Set R3 Clr R5 Load R5P (Set RF7)			L Shift R3 L Shift R5 R Shift R5P
t3	Clr R2 RADD=R3	RADD=R0	RADD=R0+R3	Clr R2 RADD=R3
t4	Load R4 Load R6 Clr RTRIG	Dec R4 Dec R6 Load RTRIG	Load R4 Dec R6 Clr RTRIG	Load R4 Load R6 Clr RTRIG
t5	Load R2P Load R2	Load R2P Load R2	Load R2P Load R2	Load R2P Load R2
t6	(Address for next Microinstruction is valid.)			
t7	Load R1P Clr R1	Load R1P Load R1	Load R1P Load R1	Load R1P Clr R1

the sine, but is correct for the cosine only when RTRIG is greater than or equal to 1024, which is indicated by UFLAG true. UFLAG controls the addition or subtraction at ADD1 and ADD2 to compensate for the sign of the cosine. The "A" operand enters register AR and AI at time t6 for use in the final sum and difference operations. The design utilizes multipliers that reflect the capabilities of TRW's HJ series,

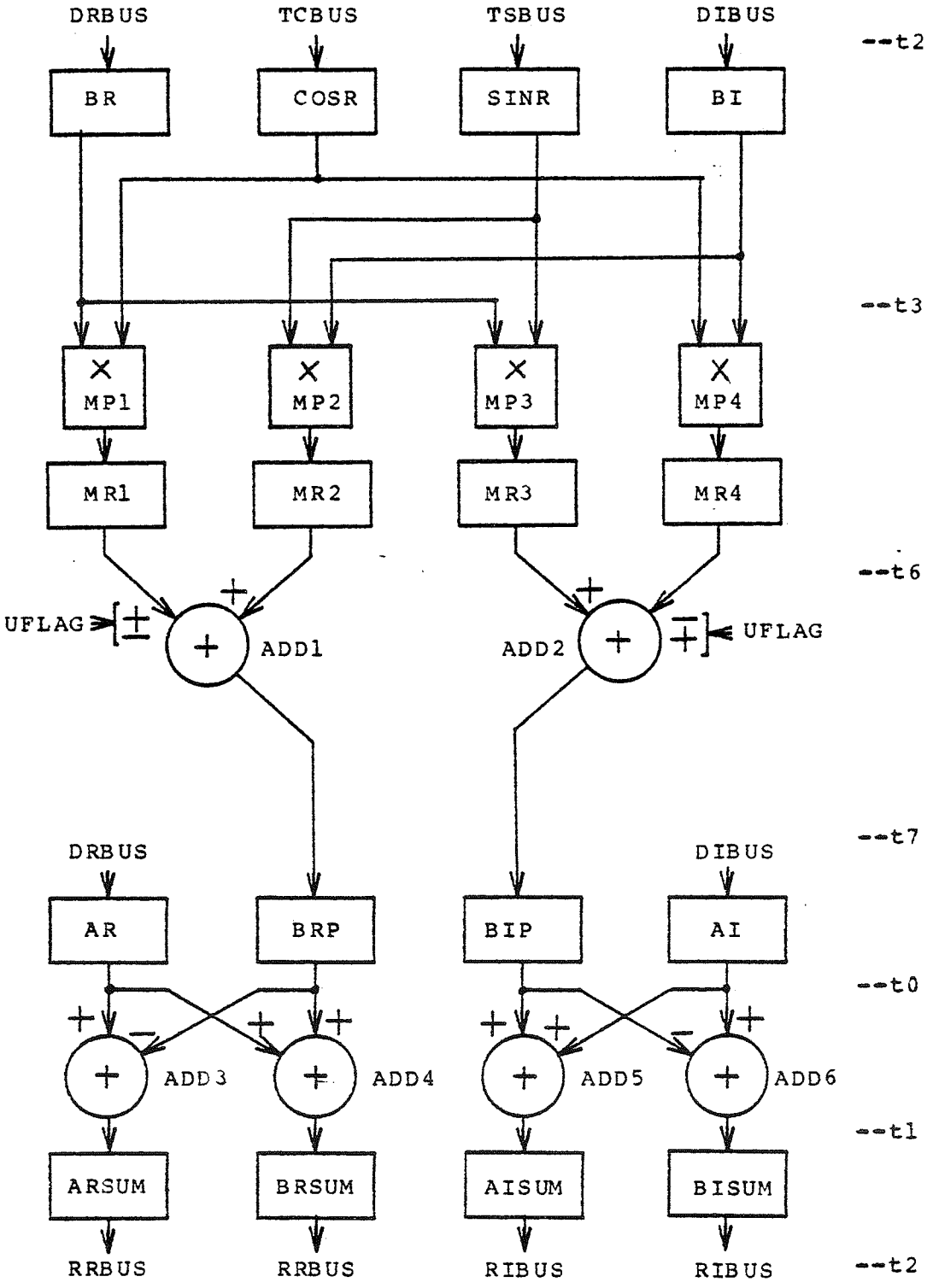


Figure 12. Full-Butterfly FFTU Implementation

but the implementation need not be restricted to this hardware. The two's complement fractions are represented as:

$$(-1)(\text{sign bit}) + \sum_{i=1}^{V} p2^{-i}$$

where V is the register length (number of bits) minus 1, and p is the value of the bit representing 2^{-i}. This representation precludes the use of multiplication by +1 (needed when sine = -1 or cosine = 1) but allows the use of -1, thus storing all trigonometric coefficients as negative numbers allows for consistency of control. The multiplier can produce a half-width rounded result which is useful since the data will subsequently be truncated by the register width. The result of the negative trigonometric coefficients is that register BRP contains the negative of the algebraic equation for BRP. This is corrected by appropriate choice of the sign at the final sum and difference stage (ADD3 and ADD4). Multilevel pipelining is difficult to implement in the FFTU because approximately half of the cycle is used for the multiplication. Thus the data in BR, BI, COSR, and SINR, propagates combinationally until it is latched into the output registers ARSUM, AISUM, BRSUM, and BISUM, at time t1. This places a restriction only on the total time to propagate data from input to output, rather than the modelled propagation through each register. Finally, at time t2, the result registers are ready to place their data on the data busses (RRBUS and RIBUS) for storage to the main memory.

IV. DESIGN OF THE SIMULATOR

A. General Design

A software simulation of the FFTP has been written in FORTRAN and run on the Interdata 7/32 computer system. The program listing is included in Appendix A, and a sample run for N = 8 is included in Appendix B. The goal of the simulation is to prove the correctness of the hardware design (hardware algorithm), check the propagation time through each logic element, indicate the total time to perform an FFT computation, and provide the numerical computation of the FFT to the desired precision (bit width) in order to evaluate the precision of the result and the error due to truncation and round-off.

Variable names correspond to the notation used in the diagrams of the hardware design. A variable is used to represent each register, adder, multiplier, bus, and zero detector. A memory is similarly modelled by an array variable. Associated with each variable, is a tag (suffix TAG), which represents the maximum propagation time through that particular element. Other variables are used to implement time, microinstruction type, bit width, data masks, and other system parameters. All variables, with the exception of a few temporary variables, appear at the beginning of the program with a brief description of their use. These variables are allocated COMMON storage, and are copied in each subroutine in which they are used.

B. Subroutine Specification

The function ZERODT (see Appendix A at line 174 of the program) is used to model the detection of zero in a register (R6 and R4 in particular). The routine checks for sufficient time delay, then returns true if the register contains zero. The subroutine BITREV (line 495) performs a bit reversal of an operand of width $\log_2$MAXN. At line 950 of the program is the subroutine REGSTR which models a register transfer in the FFTU. The register output (ROUT) is set equal to the register input (RIN) after checking the input tag (RINTAG) to insure that the input data is actually ready. If data is not ready, an error message is printed and the program stops. The output data is truncated to the specified bit width (MSB's are saved) and the output tag (ROUTAG) is set to the current time plus the register delay time (REGDLY). That is, the output becomes valid at the time indicated by ROUT. At line 976, the entry CTLSTR is exactly like REGSTR, except that data is truncated so that the LSB's are saved. This is used to model register transfers in the control section. BUSSTR, at line 991, is used similarly to place data on a bus. The bus is combinational, so the time at which the data becomes ready is the bus delay (BUSDLY) added to the maximum of current time and the input tag. Subroutine RECALL and STORE at lines 1003 and 1028, respectively, model main memory references. The memory tag (MBRTAG) is set to the time at which the memory reference will be complete, and is checked prior to each new reference. The subroutine TGCALL (line

1056), used to retrieve trigonometric coefficients, is identical to RECALL except for the memory size (array size). The ADDER subroutine at line 1081 represents a combinational adder. The operands A1 and A2 are added (subtracted) if OP is true (false) to produce the result A3. The result is ready one add delay (ADDDLY) following the availability of the last operand. The MPY subroutine (line 1108) performs a two's complement fractional multiplication. The scaling in the computation is due to the integer representation of operands on the simulating computer. Rounding to the specified bit width is performed to model a multiplier similar to the TRW HJ series.

C. Program Execution

The program begins execution at line 86 by inputting the user controlled variables. That is, the delay times of the logic elements, whether or not to scale data by 1/N, and the length of the current FFT computation. Next, the subroutine SETMSK (line 336) is called to set program constants according to the input variables. Next, TRGINT (line 199) is called to store the trigonometric coefficients to memory with the specified bit width. At line 244, DATSTR is called to read in the initial data sequence from a file. The raw sequence is printed, stored in bit reversed order, then printed again, after truncation, indicating the location in memory and the proper precision (see Appendix B).

The actual computation begins when UINIT is called (line 384). This models the INIT microinstruction and carries out the initialization of the FFTP in preparation for the first butterfly computation. After initialization the program proceeds with the main loop which calls in turn, control, memory, and FFTU simulation subroutines at each time frame. When the end of the computation is indicated, the FINI microinstruction is simulated by the subroutine FINI (line 459). In the main loop, the execution of the microinstructions NORMAL, SKIPSLICE, and LOGSLICE is indicated by the variable UTYPE. These three microinstructions perform the bulk of the computation. Subroutines CTLT-, MEMT-, and FFTUT-, perform the simulated realization of the control block, memory block, and FFTU, as indicated in the timing diagrams of tables 1 and 3, and figure 12, respectively. At the completion of the computation, the statistics and the final data sequence are printed.

V. RESULTS OF THE SIMULATION

A. Dealing with Overflow

Oppenheim and Weinstein [13] have derived a model to approximate the error due to finite register length in computing an FFT. This error is due to truncation of the multiplier outputs as well as quantization of trigonometric coefficients and initial data. Oppenheim and Weinstein model the product truncation noise statistically with a white noise generator at each multiplier. The effects of quantization of trigonometric coefficients and data are ignored. In the Bioacoustics Laboratory environment, the data is 8 bits while the trigonometric coefficients are stored to the bitwidth of the FFTP. Ignoring the quantization of trigonometric coefficients seems reasonable, since the precision of the data is less. Ignoring the quantization of data can be justified by assuming that the 8 bits of data are exact. The result of Oppenheim and Weinstein should be slightly pessimistic since the multiplications by one and minus one are performed noiselessly. The result which concerns this design is the method of dealing with overflow. If overflow occurs in the computation of an FFT, it can be dealt with in two ways with this design of the FFTP. The initial data can be scaled down small enough so that overflow does not occur, or initial data can be scaled up and division by 2 at each log slice will prevent overflow. Oppenheim and Weinstein show that the latter method is superior in terms of reducing

the noise-to-signal ratio. The former method produces a noise-to-signal ratio proportional to N-squared, whereas the latter method produces a noise-to-signal ratio proportional to N. In this design, the occurrence of overflow is detected by observing the I/O STATUS register. Correction can then be made utilizing either of the two methods described above.

B. Precision and Error Results

In order to evaluate the precision of the results, a number of test runs were performed. The initial data sequence used is |cos(n(pi)/N)| for real data, and zero for imaginary data. The results, when normalized to one and multiplied by pi/N, (except for the first point which is multiplied by 2(pi)/N), is a high peak at k = 0, and a monotonically decreasing sequence over the first N/2 points of F(k). The initial function is real and even, so the DFT is also real and even. The precision of the result is checked by comparing the real result to a more accurate computation performed on a Hewlett Packard desk-top calculator. The results of this experimental data for N = 128 are shown in table 4. The worst case precision is indicated by the minimum number of significant bits which were observed to be correct in the data resulting from an FFT computation. The precision does not change drastically for changes in N. For N in the range of 64 to 4K, the worst case precision falls in the range of b - 4 (b minus 4) bits to b - 6 bits when 1/N scaling is used, and b - 6 bits to b - 10

Table 4. Precision and Error Results for N = 128

BITWIDTH	WORST CASE PRECISION	SCALED BY 1/N	ABSOLUTE ERROR
24	17 bits	no	mean 1.273 var. 1.214 s. d. 1.106
24	20 bits	yes	mean 0.430 var. 0.308 s. d. 0.557
20	10 bits	no	mean 1.258 var. 1.706 s. d. 1.316
20	15 bits	yes	mean 0.400 var. 0.271 s. d. 0.524
16	8 bits	no	mean 1.203 var. 1.495 s. d. 1.232
16	10 bits	yes	mean 0.492 var. 0.250 s. d. 0.504
12	4 bits	no	mean 1.231 var. 1.531 s. d. 1.247
12	7 bits	yes	mean 0.338 var. 0.224 s. d. 0.477
8	1 bit	no	mean 0.576 var. 3.002 s. d. 1.746
8	2 bits	yes	mean 0.092 var. 0.115 s. d. 0.341

bits when 1/N scaling is not used, where b is the number of bits used to represent a fixed point word. The magnitude of the absolute error is checked by observing the imaginary result which should ideally be zero. Any deviation from zero indicates error due to finite register length. The mean, variance, and standard deviation of the absolute error over the first 64 points is shown. The absolute error for a given b, when 1/N scaling is used, does not vary drastically from the values indicated in the table. This is expected since the error is scaled by one half at each log slice, so the error generated in the early stages is reduced in significance. If 1/N scaling is not used, the mean absolute error is as high as 10 for b = 20 and N = 4K. These figures indicate that a 12 bit machine may be adequate for current intended use, but a 16 bit machine should be considered if more than 8 bits of input data precision are to be used.

C. Simulated Run Times

The time to compute an FFT of a given length N, is a function of the time delays through various hardware elements. The requirements are based on one eighth of a full clock cycle, which is referred to as a time-tick. The delay through a register, adder, or bus must be less than or equal to one time-tick, the memory access time must occur within two time-ticks, and a multiply must occur within four time-ticks. The multiplication time is the restricting factor, so that a quick estimate of speed is one butterfly

per clock cycle, where one clock cycle is twice the multiplication time. Using TRW multipliers, one butterfly can be accomplished in 160 nanoseconds for a bitwidth of 12, 200 nanoseconds for a bitwidth of 16, and 400 nanoseconds for a bitwidth of 24. For a bitwidth of 16, this results in a complete FFT computation in 90.0 microseconds for N = 128, 1.02 milliseconds for N = 1024, and 4.92 milliseconds for N = 4096. Note that a data transfer of 128 complex points to and from the Interdata 7/32 in burst mode, occurs at 6.4 mega-words per second (16 bit words). This takes a total of 80 microseconds, which is close to the time to compute the 128-point FFT, and thus provides good compatibility with the host computer.

VI. PERFORMANCE AND COST EVALUATION

A. Cost

The cost is an important tool in evaluating the capital outlay necessary to construct the FFTP, as well as its use in evaluating the performance of the design. To arrive at a cost, the following factors were utilized: First the cost of the memory and multiplier chips is used, since the cost per chip is high relative to the cost of surrounding hardware. Secondly, 200 percent of the cost of the chips is used to approximate the cost of the boards, wiring, and chips. Thirdly, the total power consumption was estimated at 100 watts by summing the power dissipation of individual chips. The cost of a power supply with appropriate rating was used. Fourthly, the estimated cost of additional hardware such as card cages, fans, mounting brackets, and containment hardware was added. Lastly, the cost of a graduate student for a full semester was used to approximate labor costs. The total cost figure arrived at using this method, for a 16 bit FFTU, was 4620 dollars, or about 5000 dollars.

B. Performance Measures

In order to obtain a figure of merit, I have used performance versus cost to compare this and other architectures. For a 16 bit FFTP, the computation proceeds at a rate of one butterfly per 200 nanoseconds, or 5000 butterflies per millisecond. This gives a performance/cost

figure of 5000/5000 or 1 butterfly per millisecond per dollar. The second architecture considered, utilizing a half-butterfly, reduces the cost of the FFTU by roughly half, while maintaining the same memory cost, and adding slightly to the control cost in order to account for the two cycles necessary to produce a butterfly. The FFTU is approximately 1/5 the entire cost of the FFTP, thus the cost would be about 4500 dollars but the computation rate is reduced to one butterfly per 400 nanoseconds, or 2500 butterflies per millisecond. This results in a performance/cost of 2500/4500 or 0.5556 butterflies per millisecond per dollar. This lower performance result agrees with Larson [14] who claims that performance/cost is maximum for a maximally parallel-pipelined device. The fourth architecture considered utilizes a 3-multiplier FFTU. Without considering implementation problems, the cost of the FFTU would be reduced by about 12 percent, and the performance would be reduced to one butterfly per 225 nanoseconds, or 4444 butterflies per millisecond. This results in a performance/cost of 4444/4880 or 0.91 butterflies per millisecond per dollar.

C. Commercial Processors

Any comparison with commercially available processors must necessarily account for the differences in the two processors. The commercial processors considered are Floating Point System's AP120B, Signal Processing System's SPS-21, CSPI's MAP, and Honeywell's XAP. These processors, as previously mentioned, can operate as slave peripherals or as stand-alone processors. Each is able to perform a variety of signal processing tasks, whereas the FFTP is limited to the FFT computation. The AP120B and MAP processors are able to obtain much higher numerical precision through the use of floating point arithmetic. For these two machines, the performance versus cost figure is approximately 1/45 butterflies per millisecond per dollar. That is, the speed of the FFTP is about five times that of the commercial processor, while the cost is about 1/10 that of the commercial processors. If we rate versatility with a weighting factor of 10, and precision with a weighting factor of 5, the AP120B, MAP, and FFTP would compare similarly.

VII. SUMMARY AND CONCLUSION

The FFTP has been designed to perform variable length FFT computations on data supplied by a host computer. Commands from the host computer initiate computation or data transfer. The detailed design of the FFTP has been presented, and a software simulation has been used to verify the design and obtain performance results for various machine configurations of bitwidth, component delay, and transform length. Results of the simulations indicate that a minimum word width of 12 bits should be used, though 16 bits is preferable for the Bioacoustics Laboratory environment where 8 bit input data words are used for transforms of up to 4K in length. The speed and cost make the FFTP a viable alternative to the more flexible and more expensive commercial processors.

REFERENCES

1. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, pp. 297-301, April 1965.

2. E. O. Brigham, The Fast Fourier Transform. Englewood Cliffs, N.J.: Prentice-Hall, 1974.

3. Abraham Peled, "A Digital Signal Processing System," International Conference on Acoustics, Speech, and Signal Processing, New York: IEEE Press, pp. 636-639, 1976.

4. R. A. Kobylinski, P. D. Stigall, and R. E. Ziemer, "A Microcomputer-Based Data Acquisition System with Hardware Capabilities to Calculate a Fast Fourier Transform," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-27, pp. 202-203, April 1979.

5. Gary D. Hornbuckle and Enrico I. Ancona, "The LX-1 Microprocessor and Its Application to Real-Time Signal Processing," IEEE Transactions on Computers, vol. C-19, pp. 710-720, August 1970.

6. P. E. Blankenship, A. H. Huntoon, and V. J. Sferrino, "LSP/2 Programmable Signal Processor," Proceedings of the National Electronics Conference, vol. XXIX, pp. 416-421, October 1974.

7. Bernard Gold, Irwin L. Lebow, Paul G. McHugh, and Charles M. Rader, "The FDP, a Fast Programmable Signal Processor," IEEE Transactions on Computers, vol. C-20, pp. 33-38, January 1971.

8. Glenn D. Bergland, "Fast Fourier Transform Hardware Implementations- An Overview," IEEE Transactions on Audio Electroacoustics, vol. AU-17, pp. 104-108, June 1969.

9. Hideo Aiso, Mario Tokoro, Shun-ichi Uchida, Hideki Mori, Noriyuki Kaneko, and Motoo Shimada, "A Very High-Speed Microprogrammable Pipeline Signal Processor," Digital Signal Computers and Processors, New York: IEEE Press, Ed. Andres C. Salazar, pp. 37-41, 1977.

10. Advanced Micro Devices: The AM2900 Family Data Book, Sunnyvale, Calif., 1978.

11. TRW, Inc.: LSI Multipliers; HJ Series, TRW LSI Products, Redondo Beach, Calif., 1978.

12. J. P. Agrawal and Jacob Ninan, "Hardware Considerations in FFT Processors," IEEE International Conference on Acoustics Speech, and Signal Processing, New York: IEEE Press, pp. 618-621, 1976.

13. Alan V. Oppenheim and Clifford J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proceedings of the IEEE, vol. 60, pp. 957-976, August 1972.

14. Arvid G. Larson, "Cost-Effective Processor Design with Application to Fast Fourier Transform Computers," Technical Report No. 73, Digital Systems Laboratory, Stanford University, Stanford Calif., August 1973.

APPENDIX A

Program Listing

```
$BATCH
C       ********  DRSIM  ********
C       THIS PROGRAM IS A TIME DRIVEN SIMULATOR FOR SIMULATING THE EVENTS
C       OF A HARDWARE DESIGN AT THE REGISTER LEVEL, WHERE TIME DELAYS ARE
C       SPECIFIED FOR REGISTERS, ADDERS, MULTIPLIERS, ETC.
C       THIS SIMULATION IS A HIGH SPEED FAST FOURIER TRANSFORM
C       PROCESSOR, WHICH IS ATTACHED TO A HOST COMPUTER.
C
C
C       THE R PREFIX DENOTES CONTROL REGISTERS USED IN THE GENERATION
C       OF OPERAND ADDRESSES.
        INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
       >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
       >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
       >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
        COMMON /CTL/R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
       >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
       >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
       >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
C
C       THE FFTU REGISTERS ARE DESIGNATED AS FOLLOWS:
        INTEGER*4 AR,AI,BR,BI,BRP,BIP,COSR,SINR,
       >ARSUM,AISUM,BRSUM,BISUM,
       >MP1,MP2,MP3,MP4,MR1,MR2,MR3,MR4,
       >ADD1,ADD2,ADD3,ADD4,ADD5,ADD6,
       >ARTAG,AITAG,BRTAG,BITAG,BRPTAG,BIPTAG,
       >COSTAG,SINTAG,ARSTAG,AISTAG,BRSTAG,BISTAG,
       >MP1TAG,MP2TAG,MP3TAG,MP4TAG,MR1TAG,MR2TAG,MR3TAG,MR4TAG,
       >AD1TAG,AD2TAG,AD3TAG,AD4TAG,AD5TAG,AD6TAG
        COMMON /FFTU/AR,AI,BR,BI,BRP,BIP,COSR,SINR,
       >ARSUM,AISUM,BRSUM,BISUM,
       >MP1,MP2,MP3,MP4,MR1,MR2,MR3,MR4,
       >ADD1,ADD2,ADD3,ADD4,ADD5,ADD6,
       >ARTAG,AITAG,BRTAG,BITAG,BRPTAG,BIPTAG,
       >COSTAG,SINTAG,ARSTAG,AISTAG,BRSTAG,BISTAG,
       >MP1TAG,MP2TAG,MP3TAG,MP4TAG,MR1TAG,MR2TAG,MR3TAG,MR4TAG,
       >AD1TAG,AD2TAG,AD3TAG,AD4TAG,AD5TAG,AD6TAG
C
C       THE FOLLOWING VARIABLES ARE CONTROLLED BY THE PROGRAM:
C       ---MSK  -THE VARIOUS MASKS USED TO TRUNCATE OR ROUND
```

```
41    C                VALUES TO THE PROPER BITWIDTH.
42    C      INITIM  -INITIAL SIMULATED TIME.
43    C      TIME    -CURRENT SIMULATED TIME.
44    C      UTYPE   -THE CURRENT MICRO INSTRUCTION BEING EXECUTED.
45    C      BINIT   -THE STARTING INDEX VARIABLE TO PERFORM BIT REVERSAL.
46    C      UFLAG   -THE BINARY FLAG USED TO MODEL THE ADDITION/SUBTRACTION
47    C                CONTROL IN THE FFTU.
48    C      FINIS   -THE FLAG REPRESENTING THE FETCH OF THE LAST MICRO-
49    C                INSTRUCTION SEQUENCE IN THE COMPUTATION.
50           INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
51           LOGICAL UFLAG,FINIS
52           COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
53          >UFLAG,FINIS
54    C
55    C      THE MEMORIES ARE DENOTED BY THE FOLLOWING ARRAYS AND REGISTERS.
56           INTEGER*4 RMEM(4096),IMEM(4096),TMEM(1025),
57          >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
58           COMMON /MEM/RMEM,IMEM,TMEM,
59          >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
60    C
61    C      THE FOLLOWING ARE USED TO REPRESENT BUSSES AND REGISTERS TO
62    C      AND FROM MEMORY AND THE FFTU.
63           INTEGER*4 DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
64          >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
65           COMMON/BUS/DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
66          >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
67    C
68    C      THE FOLLOWING 'CONSTANTS' CAN BE SELECTED TO CONTROL THE DISPOSITION
69    C      OF THE SIMULATION.
70    C      BTWDTH -THE BITWIDTH OF FIXED POINT COMPUTATIONS.
71    C      MAXN   -THE MAXIMUM LENGTH TRANSFORM THAT CAN BE HANDLED.
72    C      COMND  -THE COMMAND SPECIFYING THE LENGTH OF THE SEQUENCE
73    C                CURRENTLY BEING TRANSFORMED.
74    C      TIMTCK -THE LENGTH (IN NANOSECONDS) OF ONE-EIGHTH OF A MACHINE CYCLE.
75    C      ---DLY -THE MAXIMUM TIME DELAY (IN NANOSECONDS) FOR DATA TO
76    C                PROPOGATE THROUGH A REGISTER, ADDER, MULTIPLIER,
77    C                MEMORY, BUS, OR ZERO DETECTOR.
78           INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
79          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
80           LOGICAL SCALE
```

```
      COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
C
C     INPUT THE PROGRAM VARIABLES.
15    FORMAT(I5)
      READ(1,15)COMND
      READ(1,15)BTWDTH
      READ(1,15)MAXN
      READ(1,15)TIMTCK
C     IS DATA TO BE SCALED (NOT 0) OR NOT SCALED (= 0)?
      READ(1,15)REGDLY
        SCALE = .FALSE.
        IF (REGDLY.NE.0) SCALE = .TRUE.
      READ(1,15)REGDLY
      READ(1,15)ADDDLY
      READ(1,15)MPYDLY
C     SET MPYDLY TO COMPENSATE FOR THE METHOD OF SIMULATION.
      MPYDLY = MPYDLY - REGDLY
      READ(1,15)MEMDLY
C     SET MEMDLY TO COMPENSATE FOR THE METHOD OF SIMULATION.
      MEMDLY = MEMDLY - REGDLY
      READ(1,15)BUSDLY
      READ(1,15)ZRODLY
C
C
C     INITIATE PROGRAM VARIABLES.
      CALL SETMSK
C     STORE THE TRIGONOMETRIC COEFFICIENTS.
      CALL TRGINT
C
C
C     STORE DATA SEQUENCE IN MEMORY.
      CALL DATSTR
C
C
C     PERFORM INITIAL PASS OF COMPUTATION.
C     THIS IS ANALOGUS TO THE 'INIT' MICRO INSTRUCTION.
      CALL UINIT
C
C
```

```
C     **********************
C     MAIN COMPUTATION LOOP.
C     **********************
C
100   CALL TIMER
C     TIME T0.
      CALL CTLT0
      CALL MEMT0
      CALL FFTUT0
      CALL TIMER
C     TIME T1
      CALL CTLT1
      CALL MEMT1
      CALL FFTUT1
      CALL TIMER
C     TIME T2.
      CALL CTLT2
      CALL MEMT2
      CALL FFTUT2
      CALL TIMER
C     TIME T3
      CALL CTLT3
      CALL MEMT3
      CALL FFTUT3
      CALL TIMER
C     TIME T4.
      CALL CTLT4
      CALL MEMT4
      CALL FFTUT4
      CALL TIMER
C     TIME T5
      CALL CTLT5
      CALL MEMT5
      CALL FFTUT5
      CALL TIMER
C     TIME T6.
      CALL CTLT6
      CALL MEMT6
      CALL FFTUT6
      CALL TIMER
```

```
161     C       TIME T7
162             CALL CTLT7
163             CALL MEMT7
164             CALL FFTUT7
165             GO TO 100
166     C
167             STOP
168             END
169     C
170     C
171     C       THE FOLLOWING FUNCTION RETURNS TRUE IF THE NUMBER IS ZERO.
172     C       THIS MODELS THE DETECTION OF ZERO IN A REGISTER.
173     C       ************************
174             LOGICAL FUNCTION ZERODT(Z,ZTAG)
175     C       ************************
176             INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
177             LOGICAL UFLAG,FINIS
178             COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
179            >UFLAG,FINIS
180             INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
181            >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
182             LOGICAL SCALE
183             COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
184            >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
185             INTEGER*4 Z,ZTAG
186     C
187     C       CHECK PROPOGATION DELAY.
188             IF (ZTAG+ZRODLY.GT.TIME) CALL ERROR(1)
189             ZERODT = .FALSE.
190             IF (Z.EQ.0) ZERODT = .TRUE.
191             RETURN
192             END
193     C
194     C
195     C       TRGINT IS USED TO STORE THE TRIG TABLE IN TMEM.
196     C       MAXN/4 TERMS ARE STORED.  STORED COEFFICIENTS ARE:
197     C            -SIN((2*PI)X/MAXN),  X=0,1,2,...,(MAXN/4)-1
198     C       ******************
199             SUBROUTINE TRGINT
200     C       ******************
```

```
201           INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
202          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
203           LOGICAL SCALE
204           COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
205          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
206           INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
207           LOGICAL UFLAG,FINIS
208           COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
209          >UFLAG,FINIS
210           INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
211          >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
212          >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
213          >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
214           COMMON /CTL/R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
215          >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
216          >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
217          >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
218           INTEGER*4 RMEM(4096),IMEM(4096),TMEM(1025),
219          >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
220           COMMON /MEM/RMEM,IMEM,TMEM,
221          >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
222           COMMON /TG/PI
223           INTEGER*4 L,M,N,TEMP
224           DOUBLE PRECISION PI,X,Y,Z
225     C
226           PI = 3.14159265358979323846D0
227           K = (MAXN/4) + 1
228           Z = MAXN/2.0D0
229           L = 0
230           CALL BSET(L,0)
231           DO 1000 I = 1,K
232             J = I-1
233             Y = J/Z
234             X = DSIN(PI * Y)
235             Y = (X * L) - 5.0D-1
236             TMEM(I) = Y
237             CALL ROUND(TMEM(I))
238     1000    CONTINUE
239           RETURN
240     C
```

```
241     C
242     C       DATSTR IS USED TO STORE THE INITIAL DATA SEQUENCE TO MEMORY.
243     C       ************
244             ENTRY DATSTR
245     C       ************
246     C
247     35      FORMAT(2(I20))
248             DO 800 I = 1,MAXN
249               RMEM(I) = 0
250               IMEM(I) = 0
251     800       CONTINUE
252             WRITE(3,30)
253             WRITE(3,30)
254             WRITE(3,41)
255             WRITE(3,42)
256     C       READ DATA SEQUENCE AND STORE IN BIT REVERSED ORDER.
257     C       DATA IS ROUNDED TO SPECIFIED BIT WIDTH.
258             DO 810 I = 1,COMND
259               K = I - 1
260               J = K
261               CALL BITREV(J)
262               J = J + 1
263               READ(2,35)RMEM(J),IMEM(J)
264               WRITE(3,50)K,RMEM(J),IMEM(J)
265               CALL ROUND(RMEM(J))
266               CALL ROUND(IMEM(J))
267     810       CONTINUE
268             GO TO 820
269             RETURN
270     C
271     C
272     C       THE FOLLOWING SUBROUTINE PRINTS THE RESULTS OF THE SIMULATION.
273     C       ************
274             ENTRY RESULT
275     C       ************
276     10      FORMAT(('TOTAL TIME IS'),(2X,I11),('  NANOSECONDS.'))
277     20      FORMAT(('SEQUENCE LENGTH IS'),(1X,I6),('  DATA POINTS.'))
278     24      FORMAT(('COMPUTATIONS MODELLING A '),(I4),(' BIT MACHINE.'))
279     33      FORMAT('THE DATA IS SCALED BY 1/N.')
280     25      FORMAT(('ONE EIGHTH OF A MACHINE CYCLE IS ')
```

```
281           >,(I5),(' NANOSECONDS.'))
282      26    FORMAT('REGDLY  ADDDLY  MPYDLY  MEMDLY  BUSDLY  ZRODLY')
283      27    FORMAT((I3),6(5X,I3))
284      30    FORMAT(' ')
285      39    FORMAT('RESULTANT SEQUENCE:')
286      40    FORMAT((2X,'K'),(17X,'REAL DATA'),(13X,'IMAGINARY DATA'))
287      41    FORMAT('ORIGINAL DATA SEQUENCE:')
288      42    FORMAT((2X,'N'),(17X,'REAL DATA'),(13X,'IMAGINARY DATA'))
289      44    FORMAT('SEQUENCE STORED IN MEMORY:')
290      45    FORMAT((2X,'M'),(17X,'REAL DATA'),(13X,'IMAGINARY DATA'))
291      50    FORMAT((I4),(2(5X,I20)))
292      C
293            WRITE(3,30)
294            WRITE(3,30)
295            TEMP = TIME - INITIM
296            WRITE(3,10)TEMP
297            WRITE(3,20)COMND
298            WRITE(3,24)BTWDTH
299            IF (SCALE) WRITE(3,33)
300            WRITE(3,25)TIMTCK
301            WRITE(3,26)
302            MPYDLY = MPYDLY + REGDLY
303            MEMDLY = MEMDLY + REGDLY
304            WRITE(3,27)REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
305            WRITE(3,30)
306            WRITE(3,39)
307            WRITE(3,40)
308            GO TO 830
309      C     THIS ENTRY IS USED TO PRINT INITIAL DATA.
310      C     **********
311      820   ENTRY ORIG
312      C     **********
313            WRITE(3,30)
314            WRITE(3,44)
315            WRITE(3,45)
316            N = (2 ** (32-BTWDTH))
317      830   DO 110 I = 1,MAXN,R0
318              TEMP = I - 1
319              L = RMEM(I) / N
320              M = IMEM(I) / N
```

```
321             WRITE(3,50)TEMP,L,M
322     110     CONTINUE
323           RETURN
324     C
325     C
326     C     UPDATE TIME (INCREASING).  EACH UNIT REPRESENTS 25 NANOSECONDS.
327     C     ***********
328           ENTRY TIMER
329     C     ***********
330           TIME = TIME + TIMTCK
331           RETURN
332     C
333     C
334     C     SETMSK INITIALIZES THE MASKS USED FOR TRUNCATING TO PROPER BIT WIDTH.
335     C     ************
336           ENTRY SETMSK
337     C     ************
338     C     SET BINIT FOR USE AS THE INITIAL INDEX FOR BIT REVERSAL.
339           BINIT = 0
340           DO 840 I = 1,31
341             IF (BTEST(MAXN,I))  BINIT = I + 1
342     840   CONTINUE
343     C     CLEAR END-OF-COMPUTATION INDICATOR.
344           FINIS = .FALSE.
345     C     SET INITIAL TIME.
346           TIME = 0
347           INITIM = TIME
348     C     SET R0 = MAXN / COMND.
349           R0 = ISHFT(COMND,-1)
350           CALL BITREV(R0)
351     C     SET MASKS.
352           BITMSK = -1
353           TEMP = 32 - BTWDTH
354           DO 900 I = 1,TEMP
355             RNDMSK = BITMSK
356             BITMSK = ISHFT(RNDMSK,1)
357     900     CONTINUE
358           RNDMSK = IEOR(BITMSK,RNDMSK)
359           CTLMSK = MAXN - 1
360           RETURN
```

```
C
C
C     THE FOLLOWING SUBROUTINE DETERMINES THE PROPER ADDRESSES FOR
C     RETRIEVAL OF TRIGONOMETRIC COEFFICIENTS, BASED ON THE VALUE
C     OF REGISTER RTRIG.
C     ************
      ENTRY TRIGIT
C     ************
C
      RTRIGS = RTRIG
      IF (BTEST(RTRIG,21)) RTRIGS = -1 * RTRIG
      CALL CTRUNC(RTRIGS)
      CALL BCLR(RTRIGS,20)
      RTRIGC = (MAXN / 4) - RTRIGS
      CALL CTRUNC(RTRIGC)
      CALL BCLR(RTRIGC,20)
      RETURN
      END
C
C
C     THE FOLLOWING SUBROUTINE PERFORMS THE FIRST CYCLE OF COMPUTATION.
C     THIS IS ANALOGUS TO THE 'INIT' MICRO-INSTRUCTION.
C     ****************
      SUBROUTINE UINIT
C     ****************
      INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      LOGICAL SCALE
      COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
      LOGICAL UFLAG,FINIS
      COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
     >UFLAG,FINIS
      INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
     >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
     >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
     >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
      COMMON /CTL/R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
     >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
```

```
401           >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
402           >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
403            INTEGER*4 DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
404           >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
405            COMMON/BUS/DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
406           >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
407            INTEGER*4 RMEM(4096),IMEM(4096),TMEM(1025),
408           >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
409            COMMON /MEM/RMEM,IMEM,TMEM,
410           >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
411     C
412     C      TIME T0.  INITIALIZE VARIABLES.
413            R0TAG = INITIM
414            CALL TIMER
415            CALL TIMER
416     C      TIME T2.  SET RF7.
417            RF7 = R0
418            CALL BITREV(RF7)
419            RF7 = RF7 - 1
420     C      SET RF7TAG.
421            CALL CTLSTR(RF7,R0TAG,RF7,RF7TAG)
422            CALL CTLSTR(R0,R0TAG,R3,R3TAG)
423            CALL CTLSTR(0,R0TAG,R5,R5TAG)
424            CALL CTLSTR(MAXN/2,R0TAG,R5P,R5PTAG)
425            CALL TIMER
426     C      TIME T3.
427            CALL ADDER(R5P,R5PTAG,RTRIG,RTTAG,TADD,TADTAG,.TRUE.)
428            CALL ADDER(0,R0TAG,R3,R3TAG,RADD,RADTAG,.TRUE.)
429            CALL TIMER
430     C      TIME T4.
431            CALL ADDER(0,R1TAG,0,RADTAG,R1ADD,R1ATAG,.TRUE.)
432            CALL ADDER(0,R1TAG,RADD,RADTAG,R2ADD,R2ATAG,.TRUE.)
433            CALL CTLSTR(R5,R5TAG,R4,R4TAG)
434            CALL CTLSTR(RF7,RF7TAG,R6,R6TAG)
435            CALL CTLSTR(0,R0TAG,RTRIG,RTTAG)
436            CALL TRIGIT
437            CALL TIMER
438     C      TIME T5.
439            CALL CTLSTR(R0,R0TAG,R2P,R2PTAG)
440            CALL CTLSTR(R2ADD,R2ATAG,R2,R2TAG)
```

```
441     49    FORMAT(I15)
442           CALL TIMER
443     C     TIME T6.  SET SKIPSLICE INDICATOR.
444           UTYPE= 0
445           CALL MEMT6
446           CALL TIMER
447     C     TIME T7.
448           CALL CTLSTR(R1,R1TAG,R1P,R1PTAG)
449           CALL CTLSTR(R1ADD,R1ATAG,R1,R1TAG)
450           CALL BUSSTR(R2,R2TAG,ABUS,ABSTAG)
451           CALL TGCALL(TMEM,TASBUS,TASTAG,MBRTS,TMSTAG)
452           CALL TGCALL(TMEM,TACBUS,TACTAG,MBRTC,TMCTAG)
453           RETURN
454           END
455     C
456     C
457     C     THE FOLLOWING SUBROUTINE PERFORMS THE LAST CYCLE OF THE COMPUTATION.
458     C     ***************
459           SUBROUTINE FINI
460     C     ***************
461     C     TIME T2.
462           CALL MEMT2
463           CALL FFTUT2
464           CALL TIMER
465     C     TIME T3.
466           CALL MEMT3
467           CALL FFTUT3
468           CALL TIMER
469     C     TIME T4.
470           CALL CTLT4
471           CALL MEMT4
472           CALL FFTUT4
473           CALL TIMER
474     C     TIME T5.
475           CALL CTLT5
476           CALL MEMT5
477           CALL FFTUT5
478           CALL TIMER
479     C     TIME T6.
480           CALL CTLT6
```

```
      CALL MEMT6
      CALL FFTUT6
      CALL TIMER
C     TIME T7.
      CALL CTLT7
      CALL MEMT7
C     PRINT RESULTS OF SIMULATION.
      CALL RESULT
      STOP
      END
C
C
C     THE FOLLOWING SUBROUTINE PERFORMS A BIT REVERSAL OF A 12-BIT QUANTITY.
C     **********************
      SUBROUTINE BITREV(BRV)
C     **********************
      INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
      LOGICAL UFLAG,FINIS
      COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
     >UFLAG,FINIS
      INTEGER*4 BRV,TEMP
      TEMP = 0
      DO 1010 I = BINIT,31
        IF (BTEST(BRV,I)) CALL BSET(TEMP,31+BINIT-I)
1010    CONTINUE
      BRV = TEMP
      RETURN
      END
C
C
C     THE FOLLOWING SUBROUTINES (CTL PREFIX) ARE USED TO UPDATE THE
C     REGISTERS WHICH MAKE UP THE CONTROL SECTION OF THE SIMULATED
C     PROCESSOR.  THE REGISTERS ARE AS FOLLOWS:
C       R0   -COMMAND REGISTER
C       R1   -ADDRESS OF THE NEXT 'A' OPERANDS
C       R1P  -ADDRESS OF THE PREVIOUSLY COMPUTED 'A' OPERANDS
C       R2   -ADDRESS OF THE NEXT 'B' OPERANDS
C       R2P  -ADDRESS OF THE PREVIOUSLY COMPUTED 'B' OPERANDS
C       R3   -SKIP DISTANCE FOR UPDATING R1 AND R2
C       R4   -DECREMENTED COUNTER REPRESENTING REMAINING COUNT BEFORE A 'SKIP'
```

```
C        R5    -CURRENT 'SKIPCOUNT' FOR RELOADING R4
C        R6    -DECREMENTED COUNTER REPRESENTING REMAINING COUNT BEFORE 'LOGSLICE'
C        RF7   -A FICTITIOUS REGISTER CONTAINING BIT-REVERSED R0 MINUS ONE
C        RADD -THE COMBINATIONAL OUTPUT OF THE R0 + R3 ADDER
C        RTRIG-THE ADDRESS OF THE TRIG COEFFICIENT
C
C      TIME T0.
C      ****************
       SUBROUTINE CTLT0
C      ****************
       INTEGER*4 TEMP
       INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
       LOGICAL UFLAG,FINIS,ZERODT
       COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
      >UFLAG,FINIS
       INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
      >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
       LOGICAL SCALE
       COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
      >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
       INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
      >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
      >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
      >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
       COMMON /CTL/R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
      >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
      >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
      >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
C
C      TIME T0.  SET UFLAG TO CONTROL ADDERS IN FFTU.
       UFLAG = .FALSE.
       IF (BTEST(RTRIG,21))  UFLAG = .TRUE.
       RETURN
C
C
C      TIME T1.  NO ACTION.
C      ***********
       ENTRY CTLT1
C      ***********
       RETURN
```

```
C
C
C     TIME T2.  SET R5, R5P, AND R3 IF THE MICROINSTRUCTION IS LOGSLICE.
C     ***********
      ENTRY CTLT2
C     ***********
      IF (FINIS) CALL FINI
      IF (UTYPE) 2010,2010,2000
C   LOGSLICE.
2000    TEMP = ISHFT(R5,1)
        CALL BSET(TEMP,31)
        CALL CTLSTR(TEMP,R5TAG,R5,R5TAG)
        TEMP = ISHFT(R5P,-1)
        CALL CTLSTR(TEMP,R5PTAG,R5P,R5PTAG)
        TEMP = ISHFT(R3,1)
C     CHECK TERMINATING CONDITION.
        IF (TEMP.EQ.MAXN) FINIS = .TRUE.
        CALL CTLSTR(TEMP,R3TAG,R3,R3TAG)
2010  RETURN
C
C
C     TIME T3.  ESTABLISH THE OUTPUT OF THE ADDER FOR
C     FUTURE UPDATING OF R1 AND R2.
C     ***********
      ENTRY CTLT3
C     ***********
C
      CALL ADDER(R5P,R5PTAG,RTRIG,RTTAG,TADD,TADTAG,.TRUE.)
      IF (UTYPE) 2050,2060,2070
C     NORMAL.
2050    CALL ADDER(R0,R0TAG,0,R0TAG,RADD,RADTAG,.TRUE.)
        RETURN
C     SKIPSLICE.
2060    CALL ADDER(R0,R0TAG,R3,R3TAG,RADD,RADTAG,.TRUE.)
        RETURN
C     LOGSLICE.
2070    CALL ADDER(0,R0TAG,R3,R3TAG,RADD,RADTAG,.TRUE.)
        RETURN
C
C
```

```
C     TIME T4.  R4 AND R6 ARE UPDATED.
C     ***********
      ENTRY CTLT4
C     ***********
C
      IF (UTYPE) 2100,2110,2120
C     NORMAL.  DEC R4, DEC R6.  PROPOGATE R1ADD,R2ADD,TADD.
2100    CALL ADDER(R1,R1TAG,RADD,RADTAG,R1ADD,R1ATAG,.TRUE.)
        CALL ADDER(R2,R2TAG,RADD,RADTAG,R2ADD,R2ATAG,.TRUE.)
        CALL CTLSTR(R4-1,R4TAG,R4,R4TAG)
        CALL CTLSTR(R6-1,R6TAG,R6,R6TAG)
        CALL CTLSTR(TADD,TADTAG,RTRIG,RTTAG)
        CALL BCLR(RTRIG,20)
C     DETERMINE THE ADDRESSES FOR THE NEXT TRIG COEFFICIENTS.
        CALL TRIGIT
        RETURN
C     SKIPSLICE.  LOAD R4, DEC R6.  PROPOGATE R1ADD,R2ADD.
2110    CALL ADDER(R1,R1TAG,RADD,RADTAG,R1ADD,R1ATAG,.TRUE.)
        CALL ADDER(R2,R2TAG,RADD,RADTAG,R2ADD,R2ATAG,.TRUE.)
        CALL CTLSTR(R5,R5TAG,R4,R4TAG)
        CALL CTLSTR(R6-1,R6TAG,R6,R6TAG)
        CALL CTLSTR(0,RTTAG,RTRIG,RTTAG)
C     DETERMINE THE TRIG COEFFICIENTS BASED ON RTRIG.
        CALL TRIGIT
        RETURN
C     LOGSLICE.  LOAD R4, LOAD R6.  ADDERS SET FOR RESETTING R1,R2.
2120    CALL ADDER(0,R1TAG,0,RADTAG,R1ADD,R1ATAG,.TRUE.)
        CALL ADDER(0,R2TAG,RADD,RADTAG,R2ADD,R2ATAG,.TRUE.)
        CALL CTLSTR(R5,R5TAG,R4,R4TAG)
        CALL CTLSTR(RF7,RF7TAG,R6,R6TAG)
        CALL CTLSTR(0,RTTAG,RTRIG,RTTAG)
C     DETERMINE TRIG COEFFICIENTS BASED ON RTRIG.
        CALL TRIGIT
      RETURN
C
C
C     TIME T5.  UPDATE R2, R2P.
C     ***********
      ENTRY CTLT5
C     ***********
```

```
641           CALL CTLSTR(R2,R2TAG,R2P,R2PTAG)
642           CALL CTLSTR(R2ADD,R2ATAG,R2,R2TAG)
643           RETURN
644     C
645     C
646     C     TIME T6.  SET ADDRESS FOR NEXT MICROINSTRUCTION.
647     C     **********
648           ENTRY CTLT6
649     C     **********
650     C   NORMAL.
651           UTYPE= -1
652     C   SKIPSLICE.
653           IF (ZERODT(R4,R4TAG)) UTYPE= 0
654     C   LOGSLICE.
655           IF (ZERODT(R6,R6TAG)) UTYPE= 1
656           RETURN
657     C
658     C
659     C     TIME T7.  UPDATE R1 AND R2P.
660     C     **********
661           ENTRY CTLT7
662     C     **********
663     C
664           CALL CTLSTR(R1,R1TAG,R1P,R1PTAG)
665           CALL CTLSTR(R1ADD,R1ATAG,R1,R1TAG)
666           RETURN
667           END
668     C
669     C
670     C
671     C     THE FOLLOWING SUBROUTINES (PREFIX MEM) MODEL THE TRANSFERS OF
672     C     ADDRESSES AND DATA TO AND FROM MEMORY.
673     C
674     C
675     C     TIME T0.
676     C
677     C     ****************
678           SUBROUTINE MEMT0
679     C     ****************
680           INTEGER*4 RMEM(4096),IMEM(4096),TMEM(1025),
```

```
     >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
      COMMON /MEM/RMEM,IMEM,TMEM,
     >RMTAG,IMTAG,TMSTAG,TMCTAG,MBRR,MBRI,MBRTS,MBRTC
      INTEGER*4 DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
     >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
      COMMON/BUS/DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
     >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
      INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
     >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
     >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
     >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
      COMMON /CTL/R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
     >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
     >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
     >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
      INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      LOGICAL SCALE
      COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
C
C     RECALL 'B'.
      CALL RECALL(RMEM,ABUS,ABSTAG,MBRR,RMTAG)
      CALL RECALL(IMEM,ABUS,ABSTAG,MBRI,IMTAG)
      RETURN
C
C
C     TIME T1.  MEMORY DATA TO DATA BUSSES; R2P TO ADDRESS BUS.
C     TRIG DATA TO TRIG DATA BUSSES.
C
C     ***********
      ENTRY MEMT1
C     ***********
C
C     DIVIDE THE OPERANDS BY 2 IF SCALING IS INDICATED.
      IF (.NOT.SCALE) GO TO 700
        MBRR = ISHFT(MBRR,-1)
        IF (BTEST(MBRR,1)) CALL BSET(MBRR,0)
        MBRI = ISHFT(MBRI,-1)
        IF (BTEST(MBRI,1)) CALL BSET(MBRI,0)
```

```
721     700   CALL BUSSTR(MBRR,RMTAG,DRBUS,DRTAG)
722           CALL BUSSTR(MBRI,IMTAG,DIBUS,DITAG)
723           CALL BUSSTR(R2P,R2PTAG,ABUS,ABSTAG)
724           CALL BUSSTR(MBRTS,TMTAG,TSBUS,TSTAG)
725           CALL BUSSTR(MBRTC,TMTAG,TCBUS,TCTAG)
726           RETURN
727     C
728     C
729     C     TIME T2.  NO ACTION.
730     C
731     C     ***********
732           ENTRY MEMT2
733     C     ***********
734     C
735           RETURN
736     C
737     C
738     C     TIME T3.  R1 TO ADDRESS BUS; STORE 'B' RESULT.
739     C
740     C     ***********
741           ENTRY MEMT3
742     C     ***********
743     C
744     C     STORE 'B SUM'.
745           CALL STORE(RRBUS,RRTAG,RMEM,RMTAG,ABUS,ABSTAG)
746           CALL STORE(RIBUS,RITAG,IMEM,IMTAG,ABUS,ABSTAG)
747           CALL BUSSTR(R1,R1TAG,ABUS,ABSTAG)
748           RETURN
749     C
750     C
751     C     TIME T4.  MEMORY DATA TO 'A'.
752     C
753     C     ***********
754           ENTRY MEMT4
755     C     ***********
756     C
757     C     RECALL 'A'.
758           CALL RECALL(RMEM,ABUS,ABSTAG,MBRR,RMTAG)
759           CALL RECALL(IMEM,ABUS,ABSTAG,MBRI,IMTAG)
760           RETURN
```

```
761     C
762     C
763     C       TIME T5.  MEMORY DATA TO DATA BUSSES; R1P TO ADDRESS BUS.
764     C
765     C       ***********
766             ENTRY MEMT5
767     C       ***********
768     C
769     C       DIVIDE THE OPERANDS BY 2 IF SCALING IS INDICATED.
770             IF (.NOT.SCALE) GO TO 710
771               MBRR = ISHFT(MBRR,-1)
772               IF (BTEST(MBRR,1)) CALL BSET(MBRR,0)
773               MBRI = ISHFT(MBRI,-1)
774               IF (BTEST(MBRI,1)) CALL BSET(MBRI,0)
775     710     CALL BUSSTR(MBRR,RMTAG,DRBUS,DRTAG)
776             CALL BUSSTR(MBRI,IMTAG,DIBUS,DITAG)
777             CALL BUSSTR(R1P,R1PTAG,ABUS,ABSTAG)
778             RETURN
779     C
780     C
781     C       TIME T6.  TRIG ADDRESS TO TRIG ADDRESS BUS.
782     C
783     C       ***********
784             ENTRY MEMT6
785     C       ***********
786     C
787             CALL BUSSTR(RTRIGS,RTTAG,TASBUS,TASTAG)
788             CALL BUSSTR(RTRIGC,RTTAG,TACBUS,TACTAG)
789             RETURN
790     C
791     C
792     C       TIME T7.  RESULT 'A' (DATA BUS) TO MEMORY; R2 TO ADDRESS BUS;
793     C       RECALL TRIG DATA.
794     C
795     C       ***********
796             ENTRY MEMT7
797     C       ***********
798     C
799     C       STORE 'A SUM'
800             CALL STORE(RRBUS,RRTAG,RMEM,RMTAG,ABUS,ABSTAG)
```

```
801           CALL STORE(RIBUS,RITAG,IMEM,IMTAG,ABUS,ABSTAG)
802           CALL BUSSTR(R2,R2TAG,ABUS,ABSTAG)
803           CALL TGCALL(TMEM,TASBUS,TASTAG,MBRTS,TMSTAG)
804           CALL TGCALL(TMEM,TACBUS,TACTAG,MBRTC,TMCTAG)
805           RETURN
806           END
807     C
808     C
809     C
810     C     THE FOLLOWING SUBROUTINES (PREFIX FFTU) MODEL THE OPERATIONS
811     C     OF THE 'FAST FOURIER TRANSFORM UNIT' WHICH PERFORMS
812     C     ONE FULL BUTTERFLY PER CYCLE.
813     C
814     C
815     C     TIME T0.  PROPOGATE 'A' TO THE ADDERS
816     C     TO FORM THE SUM AND DIFFERENCE.
817     C
818     C     ******************
819           SUBROUTINE FFTUT0
820     C     ******************
821           INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
822           LOGICAL UFLAG,FINIS
823           COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
824          >UFLAG,FINIS
825           INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
826          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
827           LOGICAL SCALE
828           COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
829          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
830           INTEGER*4 DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
831          >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
832           COMMON/BUS/DRBUS,DIBUS,RRBUS,RIBUS,ABUS,TASBUS,TACBUS,TSBUS,TCBUS,
833          >DRTAG,DITAG,RRTAG,RITAG,ABSTAG,TASTAG,TACTAG,TSTAG,TCTAG
834           INTEGER*4 AR,AI,BR,BI,BRP,BIP,COSR,SINR,
835          >ARSUM,AISUM,BRSUM,BISUM,
836          >MP1,MP2,MP3,MP4,MR1,MR2,MR3,MR4,
837          >ADD1,ADD2,ADD3,ADD4,ADD5,ADD6,
838          >ARTAG,AITAG,BRTAG,BITAG,BRPTAG,BIPTAG,
839          >COSTAG,SINTAG,ARSTAG,AISTAG,BRSTAG,BISTAG,
840          >MP1TAG,MP2TAG,MP3TAG,MP4TAG,MR1TAG,MR2TAG,MR3TAG,MR4TAG,
```

```
841           >AD1TAG,AD2TAG,AD3TAG,AD4TAG,AD5TAG,AD6TAG
842            COMMON /FFTU/AR,AI,BR,BI,BRP,BIP,COSR,SINR,
843           >ARSUM,AISUM,BRSUM,BISUM,
844           >MP1,MP2,MP3,MP4,MR1,MR2,MR3,MR4,
845           >ADD1,ADD2,ADD3,ADD4,ADD5,ADD6,
846           >ARTAG,AITAG,BRTAG,BITAG,BRPTAG,BIPTAG,
847           >COSTAG,SINTAG,ARSTAG,AISTAG,BRSTAG,BISTAG,
848           >MP1TAG,MP2TAG,MP3TAG,MP4TAG,MR1TAG,MR2TAG,MR3TAG,MR4TAG,
849           >AD1TAG,AD2TAG,AD3TAG,AD4TAG,AD5TAG,AD6TAG
850     C
851     C      PROPOGATE 'A' TO THE ADDERS TO FORM THE SUM AND DIFFERENCE.
852            CALL ADDER(AR,ARTAG,BRP,BRPTAG,ADD3,AD3TAG,.FALSE.)
853            CALL ADDER(AR,ARTAG,BRP,BRPTAG,ADD4,AD4TAG,.TRUE.)
854            CALL ADDER(AI,AITAG,BIP,BIPTAG,ADD5,AD5TAG,.TRUE.)
855            CALL ADDER(AI,AITAG,BIP,BIPTAG,ADD6,AD6TAG,.FALSE.)
856            RETURN
857     C
858     C
859     C      TIME T1.  PROPOGATE SUM AND DIFFERENCE TO THE OUTPUT REGISTERS.
860     C
861     C      ************
862            ENTRY FFTUT1
863     C      ************
864     C
865            CALL REGSTR(ADD3,AD3TAG,ARSUM,ARSTAG)
866            CALL REGSTR(ADD4,AD4TAG,BRSUM,BRSTAG)
867            CALL REGSTR(ADD5,AD5TAG,AISUM,AISTAG)
868            CALL REGSTR(ADD6,AD6TAG,BISUM,BISTAG)
869            RETURN
870     C
871     C
872     C      TIME T2.  CLOCK IN 'B' AND TRIG OPERANDS FROM MEMORY, AND PLACE
873     C      'B' OUTPUTS OF THE FFTU ON THE MEMORY BUS FOR STORAGE.
874     C
875     C      ************
876            ENTRY FFTUT2
877     C      ************
878     C
879            CALL REGSTR(DRBUS,DRTAG,BR,BRTAG)
880            CALL REGSTR(DIBUS,DITAG,BI,BITAG)
```

```
      CALL REGSTR(TSBUS,TSTAG,SINR,SINTAG)
      CALL REGSTR(TCBUS,TCTAG,COSR,COSTAG)
      CALL BUSSTR(BRSUM,BRSTAG,RRBUS,RRTAG)
      CALL BUSSTR(BISUM,BISTAG,RIBUS,RITAG)
      RETURN
C
C
C     TIME T3.  PROPOGATE INITIAL OPERANDS TO THE MULTIPLIER INPUTS.
C
C     ************
      ENTRY FFTUT3
C     ************
C
      CALL MPY(BR,BRTAG,COSR,COSTAG,MR1,MR1TAG)
      CALL MPY(BI,BITAG,SINR,SINTAG,MR2,MR2TAG)
      CALL MPY(BR,BRTAG,SINR,SINTAG,MR3,MR3TAG)
      CALL MPY(BI,BITAG,COSR,COSTAG,MR4,MR4TAG)
      RETURN
C
C
C     TIME T4.  NO OPERATION.
C
C     ************
      ENTRY FFTUT4
C     ************
      RETURN
C
C
C     TIME T5.  NO OPERATION.
C
C     ************
      ENTRY FFTUT5
C     ************
      RETURN
C
C
C     TIME T6.  INPUT 'A' OPERANDS TO THE REGISTERS.
C     SUM THE MULTIPLIER RESULTS TO FORM 'B PRIME'.
C     'A' RESULTS OF FFTU TO MEMORY BUS.
C
```

```
C      ************
       ENTRY FFTUT6
C      ************
C
       CALL REGSTR(DRBUS,DRTAG,AR,ARTAG)
       CALL REGSTR(DIBUS,DITAG,AI,AITAG)
       CALL ADDER(MR2,MR2TAG,MR1,MR1TAG,ADD1,AD1TAG,.NOT.UFLAG)
       CALL ADDER(MR3,MR3TAG,MR4,MR4TAG,ADD2,AD2TAG,UFLAG)
       CALL BUSSTR(ARSUM,ARSTAG,RRBUS,RRTAG)
       CALL BUSSTR(AISUM,AISTAG,RIBUS,RITAG)
       RETURN
C
C
C      TIME T7.  PROPOGATE 'B PRIME' RESULTS TO THE LAST ADDER STAGE.
C
C      ************
       ENTRY FFTUT7
C      ************
C
       CALL REGSTR(ADD1,AD1TAG,BRP,BRPTAG)
       CALL REGSTR(ADD2,AD2TAG,BIP,BIPTAG)
       RETURN
       END
C
C
C
C      THE FOLLOWING SUBROUTINE MODELS A REGISTER RECEIVING  DATA
C      RIN WITH THE APPROPRIATE DELAY TO PRODUCE ROUT.
C      *****************
       SUBROUTINE REGSTR(RIN,RINTAG,ROUT,ROUTAG)
C      *****************
       INTEGER*4 RIN,RINTAG,ROUT,ROUTAG
       INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
      >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
       LOGICAL SCALE
       COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
      >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
       INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
       LOGICAL UFLAG,FINIS
       COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
```

```
      >UFLAG,FINIS
C
C      CHECK PROPOGATION TIME.
       IF (RINTAG.GT.TIME) CALL ERROR(2)
       ROUT = RIN
C      SAVE UPPER ORDER BITS.
       CALL TRUNC(ROUT)
       ROUTAG = TIME + REGDLY
       RETURN
C
C
C      THE FOLLOWING SUBROUTINE PERFORMS REGISTER TRANSFERS FOR THE
C      CONTROL SECTION OF THE SIMULATION. (LOWER ORDER MASK)
C
C      ************
       ENTRY CTLSTR
C      ************
C
C      CHECK PROPOGATION DELAY.
       IF (RINTAG.GT.TIME) CALL ERROR(3)
       ROUT = RIN
C      SAVE LOWER ORDER BITS.
       CALL CTRUNC(ROUT)
       ROUTAG = TIME + REGDLY
       RETURN
C
C
C      THE FOLLOWING SUBROUTINE TAKES DATA FROM A REGISTER AND TRANSFERS
C      IT TO THE DESIGNATED BUS.
C      ************
       ENTRY BUSSTR
C      ************
C
       ROUTAG = MAX0(TIME,RINTAG) + BUSDLY
       ROUT = RIN
       RETURN
       END
C
C
C      THE FOLLOWING SUBROUTINE READS MEMORY AT THE ADDRESS SPECIFIED
```

```
1001    C      BY  ADDR.
1002    C      *****************
1003           SUBROUTINE RECALL(MEM,ADDR,ADRTAG,MBR,MBRTAG)
1004    C      *****************
1005           INTEGER*4 MEM(4096),ADDR,ADRTAG,MBR,MBRTAG
1006           INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
1007          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
1008           LOGICAL SCALE
1009           COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
1010          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
1011           INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
1012           LOGICAL UFLAG,FINIS
1013           COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
1014          >UFLAG,FINIS
1015    C
1016    C      CHECK THAT PREVIOUS MEMORY REFERENCE IS COMPLETE.
1017           IF (MBRTAG.GT.TIME) CALL ERROR(4)
1018    C      SET THE TAG TO INDICATE TIME-OF-COMPLETION.
1019           MBRTAG = MAX0(TIME,ADRTAG) + MEMDLY
1020           MBR = MEM(ADDR + 1)
1021           RETURN
1022           END
1023    C
1024    C
1025    C      THE FOLLOWING SUBROUTINE STORES DATA TO MEMORY FROM THE DESIGNATED
1026    C      BUS, TO THE ADDRESS SPECIFIED BY ADDR.
1027    C      ****************
1028           SUBROUTINE STORE(BUS,BUSTAG,MEM,MBRTAG,ADDR,ADRTAG)
1029    C      ****************
1030           INTEGER*4 BUS,BUSTAG,MEM(4096),MBRTAG,ADDR,ADRTAG
1031           INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
1032          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
1033           LOGICAL SCALE
1034           COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
1035          >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
1036           INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
1037           LOGICAL UFLAG,FINIS
1038           COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
1039          >UFLAG,FINIS
1040    C
```

```
1041    C       CHECK THAT THE PREVIOUS MEMORY REFERENCE IS COMPLETE.
1042              IF (MBRTAG.GT.TIME) CALL ERROR(5)
1043    C       CHECK THAT ADDRESS AND DATA ARE READY IN TIME FOR STORE.
1044              IF (ADRTAG.GT.TIME) CALL ERROR(6)
1045              IF (BUSTAG.GT.TIME) CALL ERROR(7)
1046             MEM(ADDR + 1) = BUS
1047    C        SET TAG TO ALLOW FOR STORAGE OF DATA.
1048             MBRTAG = TIME + MEMDLY
1049             RETURN
1050             END
1051    C
1052    C
1053    C        THE FOLLOWING SUBROUTINE READS TRIG MEMORY AT THE ADDRESS SPECIFIED
1054    C        BY ADDR.
1055    C        ******************
1056             SUBROUTINE TGCALL(MEM,ADDR,ADRTAG,MBR,MBRTAG)
1057    C        ******************
1058             INTEGER*4 MEM(1025),ADDR,ADRTAG,MBR,MBRTAG
1059             INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
1060            >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
1061             LOGICAL SCALE
1062             COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
1063            >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
1064             INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
1065             LOGICAL UFLAG,FINIS
1066             COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
1067            >UFLAG,FINIS
1068    C
1069    C        CHECK THAT PREVIOUS MEMORY REFERENCE IS COMPLETE.
1070             IF (MBRTAG.GT.TIME) CALL ERROR(8)
1071    C        SET TAG TO ALLOW TIME FOR MEMORY REFERENCE.
1072             MBRTAG = MAX0(TIME,ADRTAG) + MEMDLY
1073             MBR = MEM(ADDR + 1)
1074             RETURN
1075             END
1076    C
1077    C
1078    C        THE FOLLOWING SUBROUTINE TAKES A1 PLUS(MINUS) A2 TO
1079    C        PRODUCE RESULT A3, IF OP IS .TRUE. (.FALSE.).
1080    C        ****************
```

```
      SUBROUTINE ADDER(A1,A1TAG,A2,A2TAG,A3,A3TAG,OP)
C     ***************
      INTEGER*4 A1,A1TAG,A2,A2TAG,A3,A3TAG
      LOGICAL OP
      INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      LOGICAL SCALE
      COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
      LOGICAL UFLAG,FINIS
      COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
     >UFLAG,FINIS
C
C     SET TAG TO INDICATE A COMBINATIONAL DELAY THROUGH THE ADDER.
      A3TAG = MAX0(A1TAG,A2TAG)
      A3TAG = MAX0(A3TAG,TIME)
      IF (OP) A3 = A1 +A2
      IF (.NOT.OP) A3 = A1 - A2
      A3TAG = A3TAG + ADDDLY
      RETURN
      END
C
C
C     MPY FORMS THE PRODUCT M3 FROM OPERANDS M1 AND M2.
C     ALL NUMBERS REPRESENT 2'S COMPLEMENT FRACTIONS.
C     **************
      SUBROUTINE MPY(M1,M1TAG,M2,M2TAG,M3,M3TAG)
C     **************
      INTEGER*4 M1,M1TAG,M2,M2TAG,M3,M3TAG
      INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      LOGICAL SCALE
      COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
     >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
      INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
      LOGICAL UFLAG,FINIS
      COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
     >UFLAG,FINIS
      DOUBLE PRECISION Y
```

```
C
C       SET TAG TO INDICATE A COMBINATIONAL DELAY.
        M3TAG = MAX0(M1TAG,M2TAG)
        M3TAG = MAX0(M3TAG,TIME)
        Y = (2.0D0 ** 31)
        Y = M2 / Y
        Y = M1 * Y
        M3 = Y
C       ROUND THE HIGHER ORDER RESULT.
        CALL ROUND(M3)
        M3TAG = M3TAG + MPYDLY
        RETURN
        END
C
C
C       THE FOLLOWING SUBROUTINE ROUNDS THE NUMBER TO THE SPECIFIED BIT WIDTH.
C       **********************
        SUBROUTINE ROUND(RND)
C       **********************
        INTEGER*4 RND
        INTEGER*4 BTWDTH,MAXN,COMND,TIMTCK,
       >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
        LOGICAL SCALE
        COMMON /VAR/BTWDTH,MAXN,COMND,TIMTCK,SCALE,
       >REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
        INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
        LOGICAL UFLAG,FINIS
        COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
       >UFLAG,FINIS
        INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
       >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
       >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
       >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
        COMMON /CTL/R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
       >RADD,TADD,R1ADD,R2ADD,RTRIGS,RTRIGC,
       >R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
       >R5PTAG,R6TAG,RF7TAG,RTTAG,RADTAG,TADTAG,R1ATAG,R2ATAG
C
        RND = RND + RNDMSK
        RND = IAND(RND,BITMSK)
```

```
1161            RETURN
1162    C
1163    C
1164    C       THE FOLLOWING SUBROUTINE TRUNCATES THE NUMBER TO THE SPECIFIED BIT WIDTH.
1165    C       ***********
1166            ENTRY TRUNC
1167    C       ***********
1168    C
1169            RND = IAND(RND,BITMSK)
1170            RETURN
1171    C
1172    C
1173    C       THE FOLLOWING SUBROUTINE MASKS UPPER BITS TO FORM A WORDWIDTH
1174    C       OF LOG2(MAXN).
1175    C       ************
1176            ENTRY CTRUNC
1177    C       ************
1178    C
1179            RND = IAND(RND,CTLMSK)
1180            RETURN
1181            END
1182    C
1183    C
1184    C       THE ERROR ROUTINE PRINTS THE SOURCE LOCATION OF THE ERROR,
1185    C       AND STOPS THE PROGRAM.
1186    C
1187    C       ****************
1188            SUBROUTINE ERROR(ER)
1189    C       ****************
1190            INTEGER*4 ER
1191    77      FORMAT (('ERROR OCCURED AT LOCATION '),(2X,I5))
1192    C
1193            WRITE(5,77)ER
1194            STOP
1195            RETURN
1196            END
1197    $BEND
```

```
      DRSIM ASSIGNMENT FILE
 1    ASSIGN 1,DRVAR.DTA
 2    ASSIGN 2,DRDTA.DTA
 3    ASSIGN 3,PR:
 4    ASSIGN 5,CON:
 5    $EXIT
```

```
      DRVAR.DTA FILE
 1    00008
 2    00016
 3    04096
 4    00025
 5    00001
 6    00025
 7    00025
 8    00100
 9    00050
10    00025
11    00025
```

APPENDIX B

Sample Program Execution

ORIGINAL DATA SEQUENCE:

N	REAL DATA	IMAGINARY DATA
0	1310720000	0
1	1210947380	0
2	926819000	0
3	501590828	0
4	0	0
5	501590828	0
6	926819000	0
7	1210947380	0

SEQUENCE STORED IN MEMORY:

M	REAL DATA	IMAGINARY DATA
0	20000	0
512	0	0
1024	14142	0
1536	14142	0
2048	18478	0
2560	7654	0
3072	7654	0
3584	18478	0

TOTAL TIME IS 2775 NANOSECONDS.
SEQUENCE LENGTH IS 8 DATA POINTS.
COMPUTATIONS MODELLING A 16 BIT MACHINE.
THE DATA IS SCALED BY 1/N.
ONE EIGHTH OF A MACHINE CYCLE IS 25 NANOSECONDS.

REGDLY	ADDDLY	MPYDLY	MEMDLY	BUSDLY	ZRODLY
25	25	100	50	25	25

RESULTANT SEQUENCE:

K	REAL DATA	IMAGINARY DATA
0	12568	0
512	4414	0
1024	-1036	0
1536	586	0
2048	-498	0
2560	586	0
3072	-1036	0
3584	4414	0