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 Schlaer
Director of Thesis Research

J. W. Seelenson, Jr.
Head of Department

Committee on Final Examination†

Chairman

† Required for doctor's degree but not for master's.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I. INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>II. BACKGROUND AND GENERAL DESCRIPTION</strong></td>
<td></td>
</tr>
<tr>
<td>A. FFT Introduction</td>
<td>2</td>
</tr>
<tr>
<td>B. System Design</td>
<td>5</td>
</tr>
<tr>
<td>C. Design Objectives</td>
<td>6</td>
</tr>
<tr>
<td>D. Architectural Block Design</td>
<td>8</td>
</tr>
<tr>
<td><strong>III. DETAILED HARDWARE DESIGN</strong></td>
<td></td>
</tr>
<tr>
<td>A. Memory</td>
<td>17</td>
</tr>
<tr>
<td>B. Control</td>
<td>18</td>
</tr>
<tr>
<td>1. Microsequencer</td>
<td>21</td>
</tr>
<tr>
<td>2. Hardware Controller (Data)</td>
<td>25</td>
</tr>
<tr>
<td>3. Hardware Controller (Trigonometric)</td>
<td>28</td>
</tr>
<tr>
<td>C. Fast Fourier Transform Unit</td>
<td>30</td>
</tr>
<tr>
<td><strong>IV. DESIGN OF THE SIMULATOR</strong></td>
<td></td>
</tr>
<tr>
<td>A. General Design</td>
<td>34</td>
</tr>
<tr>
<td>B. Subroutine Specification</td>
<td>35</td>
</tr>
<tr>
<td>C. Program Execution</td>
<td>36</td>
</tr>
<tr>
<td><strong>V. RESULTS OF THE SIMULATION</strong></td>
<td></td>
</tr>
<tr>
<td>A. Dealing with Overflow</td>
<td>38</td>
</tr>
<tr>
<td>B. Precision and Error Results</td>
<td>39</td>
</tr>
<tr>
<td>C. Simulated Run Times</td>
<td>41</td>
</tr>
<tr>
<td><strong>VI. PERFORMANCE AND COST EVALUATION</strong></td>
<td></td>
</tr>
<tr>
<td>A. Cost</td>
<td>43</td>
</tr>
<tr>
<td>B. Performance Measures</td>
<td>43</td>
</tr>
<tr>
<td>C. Commercial Processors</td>
<td>45</td>
</tr>
<tr>
<td><strong>VII. SUMMARY AND CONCLUSION</strong></td>
<td>46</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>47</td>
</tr>
<tr>
<td>APPENDIX A. Program Listing</td>
<td>49</td>
</tr>
<tr>
<td>APPENDIX B. Sample Program Execution</td>
<td>81</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1. Memory Timing Cycle</td>
<td>20</td>
</tr>
<tr>
<td>Table 2. Microcontrol Word Definition</td>
<td>24</td>
</tr>
<tr>
<td>Table 3. Control Section Timing</td>
<td>31</td>
</tr>
<tr>
<td>Table 4. Precision and Error Results for N=128</td>
<td>40</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Butterfly Functional Diagram</td>
</tr>
<tr>
<td>2</td>
<td>Butterfly Symbolic Notation</td>
</tr>
<tr>
<td>3</td>
<td>Eight-Point Decimation-in-Time FFT</td>
</tr>
<tr>
<td>4</td>
<td>General Purpose Microprogrammable FFT</td>
</tr>
<tr>
<td>5</td>
<td>Half-Butterfly Architecture</td>
</tr>
<tr>
<td>6</td>
<td>Four-Multiplier Full Butterfly</td>
</tr>
<tr>
<td>7</td>
<td>Three-Multiplier Full Butterfly</td>
</tr>
<tr>
<td>8</td>
<td>Memory Section</td>
</tr>
<tr>
<td>9</td>
<td>Microsequencer and Microinstruction Format</td>
</tr>
<tr>
<td>10</td>
<td>Control Section (Data Addresses)</td>
</tr>
<tr>
<td>11</td>
<td>Control Section (Trig Addresses)</td>
</tr>
<tr>
<td>12</td>
<td>Full-Butterfly FFTU Implementation</td>
</tr>
</tbody>
</table>
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:

\[ F(k) = \sum_{n=0}^{N-1} f(n) e^{-j2\pi n k} \text{ , } k = 0, 1, \ldots, N-1. \]

Using the notation

\[ \frac{w_{nk}}{N} = e^{-j2\pi n k} \text{ , } \]

we have:

\[ F(k) = \sum_{n=0}^{N-1} f(n) w_{nk} \text{ , } k = 0, 1, \ldots, N-1, \]

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:

\[ \text{ASUM} = (A + Bw_N^k) \text{ and } \text{BSUM} = (A - Bw_N^k) \]

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:

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

\[
= \frac{1}{N} \sum_{k=0}^{N-1} F(k) W_n^{-nk}, \quad n = 0, 1, \ldots, N-1.
\]

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

\[
f(n) = \frac{1}{N} \left[ \sum_{k=0}^{N-1} F(k) W_n^{nk} \right] = \frac{1}{N} \sum_{k=0}^{N-1} F(k) W_n^{-nk},
\]

\[
n = 0, 1, \ldots, N-1,
\]
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 Kobylniski, 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 category of the general purpose signal processor include the LX-1 [5] and the LSP/2 [6]. Computers that fall into the category 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 Kobylnski, 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 FFTU 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:

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

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.) occurring 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 redundancy 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 FPTU (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 FPTU provides a better timing match.

The third general architecture utilizes a full butterfly calculation in the FPTU. As previously shown, this can be accomplished with four multipliers and six adders. Figure 6 shows the functional elements of the FPTU. 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 FPTU 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 FPTU, 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 straightforward 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 \( l/N \) 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 \( l/N \) 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

<table>
<thead>
<tr>
<th>TIME</th>
<th>DATA RETRIEVAL</th>
<th>DATA STORAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t7</td>
<td>R2 to ABUS</td>
<td>MBR to MEM</td>
</tr>
<tr>
<td>t0</td>
<td>ABUS to MAR</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>MBR to DRBUS,DIBUS</td>
<td>R2P to ABUS</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td>ABUS to MAR RRBUS,RIBUS to MBR</td>
</tr>
<tr>
<td>t3</td>
<td>R1 to ABUS</td>
<td>MBR to MEM</td>
</tr>
<tr>
<td>t4</td>
<td>ABUS to MAR</td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td>MBR to DRBUS,DIBUS</td>
<td>R1P to ABUS</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>ABUS to MAR RRBUS,RIBUS to MBR</td>
</tr>
<tr>
<td>t7</td>
<td>R2 to ABUS</td>
<td>MBR to MEM</td>
</tr>
</tbody>
</table>

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)
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 UMIR 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

<table>
<thead>
<tr>
<th>MICROCONTROL</th>
<th>BIT</th>
<th>TIME</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>t0</td>
<td></td>
<td>Load COMND register.</td>
</tr>
<tr>
<td>c2</td>
<td>t1</td>
<td></td>
<td>Load R0.</td>
</tr>
<tr>
<td>c3</td>
<td>t1</td>
<td></td>
<td>Load SCALE.</td>
</tr>
<tr>
<td>c4</td>
<td>t7</td>
<td></td>
<td>Clear R1.</td>
</tr>
<tr>
<td>c5</td>
<td>t7</td>
<td></td>
<td>Load R1.</td>
</tr>
<tr>
<td>c6</td>
<td>t7</td>
<td></td>
<td>Clear R1P.</td>
</tr>
<tr>
<td>c7</td>
<td>t7</td>
<td></td>
<td>Load R1P.</td>
</tr>
<tr>
<td>c8</td>
<td>t3</td>
<td></td>
<td>Clear R2.</td>
</tr>
<tr>
<td>c9</td>
<td>t5</td>
<td></td>
<td>Load R2.</td>
</tr>
<tr>
<td>c10</td>
<td>t5</td>
<td></td>
<td>Clear R2P.</td>
</tr>
<tr>
<td>c11</td>
<td>t5</td>
<td></td>
<td>Load R2P.</td>
</tr>
<tr>
<td>c12</td>
<td>t2</td>
<td></td>
<td>Load R3.</td>
</tr>
<tr>
<td>c13</td>
<td>t2</td>
<td></td>
<td>Left Shift R3.</td>
</tr>
<tr>
<td>c14</td>
<td>t4</td>
<td></td>
<td>Load R4.</td>
</tr>
<tr>
<td>c15</td>
<td>t4</td>
<td></td>
<td>Decrement R4.</td>
</tr>
<tr>
<td>c16</td>
<td>t2</td>
<td></td>
<td>Clear R5.</td>
</tr>
<tr>
<td>c17</td>
<td>t2</td>
<td></td>
<td>Left Shift R5.</td>
</tr>
<tr>
<td>c18</td>
<td>t4</td>
<td></td>
<td>Load R6.</td>
</tr>
<tr>
<td>c19</td>
<td>t4</td>
<td></td>
<td>Decrement R6.</td>
</tr>
<tr>
<td>c20, not c21</td>
<td>t3</td>
<td></td>
<td>RADD = R0.</td>
</tr>
<tr>
<td>not c20, c21</td>
<td>t3</td>
<td></td>
<td>RADD = R3.</td>
</tr>
<tr>
<td>c20, c21</td>
<td>t3</td>
<td></td>
<td>RADD = R0 + R3.</td>
</tr>
<tr>
<td>c22</td>
<td>t3</td>
<td></td>
<td>R1 enable to ABUS.</td>
</tr>
<tr>
<td>c23</td>
<td>t7</td>
<td></td>
<td>R2 enable to ABUS.</td>
</tr>
<tr>
<td>c24</td>
<td>t5</td>
<td></td>
<td>R1P enable to ABUS.</td>
</tr>
<tr>
<td>c25</td>
<td>t1</td>
<td></td>
<td>R2P enable to ABUS.</td>
</tr>
<tr>
<td>c26</td>
<td>t3</td>
<td></td>
<td>Bit rev. R1 to ABUS.</td>
</tr>
<tr>
<td>c27</td>
<td>t2</td>
<td></td>
<td>Load R5P.</td>
</tr>
<tr>
<td>c28</td>
<td>t2</td>
<td></td>
<td>Right Shift R5P.</td>
</tr>
<tr>
<td>c29</td>
<td>t4</td>
<td></td>
<td>Clear RTRIG.</td>
</tr>
<tr>
<td>c30</td>
<td>t4</td>
<td></td>
<td>Load RTRIG.</td>
</tr>
</tbody>
</table>

| 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, RLP, R2P, and adders RADD, RLADD, 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 RLADD. 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 SKIPSICE 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, ARSUM, 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

<table>
<thead>
<tr>
<th>TIME</th>
<th>INIT</th>
<th>NORMAL</th>
<th>SKIPSLICE</th>
<th>LOGSLICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Set UFLAG</td>
<td>Set UFLAG</td>
<td>Set UFLAG</td>
<td>Set UFLAG</td>
</tr>
<tr>
<td>t2</td>
<td>Set R3, Clr R5, Load R5P (Set RF7)</td>
<td>L Shift R3, L Shift R5</td>
<td>R Shift R5P</td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>Clr R2, RADD=R3</td>
<td>RADD=R0</td>
<td>RADD=R0+R3</td>
<td>Clr R2, RADD=R3</td>
</tr>
<tr>
<td>t4</td>
<td>Load R4, Load R6, Clr RTRIG</td>
<td>Dec R4, Dec R6</td>
<td>Load R4, Dec R6</td>
<td>Load R4, Clr RTRIG</td>
</tr>
<tr>
<td>t5</td>
<td>Load R2P, Load R2</td>
<td>Load R2P</td>
<td>Load R2P</td>
<td>Load R2</td>
</tr>
<tr>
<td>t6</td>
<td>(Address for next Microinstruction is valid.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td>Load R1P, Clr R1</td>
<td>Load R1P</td>
<td>Load R1P</td>
<td>Load R1P</td>
</tr>
</tbody>
</table>

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 \( \sin = -1 \) or \( \cos = 1 \)) 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 \( \log_2 \text{MAXN} \). 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, TRGIN (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 FFTP 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 FFTP, 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 FFT. 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 FFT. 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 $N^2$, 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 $N = 128$

<table>
<thead>
<tr>
<th>BITWIDTH</th>
<th>WORST CASE PRECISION</th>
<th>SCALED BY $1/N$</th>
<th>ABSOLUTE ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>17 bits</td>
<td>no</td>
<td>mean 1.273</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 1.214</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 1.106</td>
</tr>
<tr>
<td>24</td>
<td>20 bits</td>
<td>yes</td>
<td>mean 0.430</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 0.308</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 0.557</td>
</tr>
<tr>
<td>20</td>
<td>10 bits</td>
<td>no</td>
<td>mean 1.258</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 1.706</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 1.316</td>
</tr>
<tr>
<td>20</td>
<td>15 bits</td>
<td>yes</td>
<td>mean 0.400</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 0.271</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 0.524</td>
</tr>
<tr>
<td>16</td>
<td>8 bits</td>
<td>no</td>
<td>mean 1.203</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 1.495</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 1.232</td>
</tr>
<tr>
<td>16</td>
<td>10 bits</td>
<td>yes</td>
<td>mean 0.492</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 0.250</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 0.504</td>
</tr>
<tr>
<td>12</td>
<td>4 bits</td>
<td>no</td>
<td>mean 1.231</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 1.531</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 1.247</td>
</tr>
<tr>
<td>12</td>
<td>7 bits</td>
<td>yes</td>
<td>mean 0.338</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 0.224</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 0.477</td>
</tr>
<tr>
<td>8</td>
<td>1 bit</td>
<td>no</td>
<td>mean 0.576</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 3.002</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 1.746</td>
</tr>
<tr>
<td>8</td>
<td>2 bits</td>
<td>yes</td>
<td>mean 0.092</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>var. 0.115</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s. d. 0.341</td>
</tr>
</tbody>
</table>
bits when \( \frac{1}{N} \) 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 \( \frac{1}{N} \) 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 \( \frac{1}{N} \) 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 FFTP, 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


APPENDIX A

Program Listing
**SBATCH**

C  * * * * * * * D R S I M  * * * * * *

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  THE R PREFIX DENOTES CONTROL REGISTERS USED IN THE GENERATION
C  OF OPERAND ADDRESSES.

C

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, R7TAG, RADTAG, RDTAG, R1ATAG, R2ATAG
C
  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, R7TAG, RADTAG, RDTAG, R1ATAG, R2ATAG

C

C  THE FFTU REGISTERS ARE DESIGNATED AS FOLLOWS:

C

INTEGER*4  AR, AI, BR, BI, BRP, BIP, COSR, SINR,
  > ARSUM, AISM, 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
  COMMON /FFTU/AR, AI, BR, BI, BRP, BIP, COSR, SINR,
  > ARSUM, AISM, 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
VALUES TO THE PROPER BITWIDTH.

INITIM = INITIAL SIMULATED TIME.
TIME = CURRENT SIMULATED TIME.
UTYPE = THE CURRENT MICRO INSTRUCTION BEING EXECUTED.
BINIT = THE STARTING INDEX VARIABLE TO PERFORM BIT REVERSAL.
UFLAG = THE BINARY FLAG USED TO MODEL THE ADDITION/SUBTRACTION
CONTROL IN THE FFTU.
FINIS = THE FLAG REPRESENTING THE FETCH OF THE LAST MICRO-
INSTRUCTION SEQUENCE IN THE COMPUTATION.

INTEGER*4 BITMSK, RNDMSK, CTLMSK, INITIM, TIME, UTYPE, BINIT
LOGICAL UFLAG, FINIS
COMMON /CONST/BITMSK, RNDMSK, CTLMSK, INITIM, TIME, UTYPE, BINIT,
>UFLAG, FINIS

THE MEMORIES ARE DENOTED BY THE FOLLOWING ARRAYS AND REGISTERS.
INTEGER*4 RMEM(4096), IMEM(4096), TMEM(1025),
>RMTAG, IMTAG, TMSTAG, TMC TAG, MBRR, MBRI, MBRTS, MBRTC
COMMON /MEM/RMEM, IMEM, TMEM,
>RMTAG, IMTAG, TMSTAG, TMC TAG, MBRR, MBRI, MBRTS, MBRTC

THE FOLLOWING ARE USED TO REPRESENT BUSSES AND REGISTERS TO
AND FROM MEMORY AND THE FFTU.
INTEGER*4 DBBUS, DIBUS, RRBUS, RIBUS, ABUS, TASBUS, TACBUS, TCBUS,
>DRTAG, DRITAG, RRTAG, RITAG, ABSTAG, TASTAG, TACTAG, TSTAG, TCTAG
COMMON/ BUS/DBBUS, DIBUS, RRBUS, RIBUS, ABUS, TASBUS, TACBUS, TCBUS,
>DRTAG, DRITAG, RRTAG, RITAG, ABSTAG, TASTAG, TACTAG, TSTAG, TCTAG

THE FOLLOWING 'CONSTANTS' CAN BE SELECTED TO CONTROL THE DISPOSITION
OF THE SIMULATION.
BITWDTH = THE BITWIDTH OF FIXED POINT COMPUTATIONS.
MAXN = THE MAXIMUM LENGTH TRANSFORM THAT CAN BE HANDLED.
COMND = THE COMMAND SPECIFYING THE LENGTH OF THE SEQUENCE
CURRENTLY BEING TRANSFORMED.
TIMTCK = THE LENGTH (IN NANOSECONDS) OF ONE-EIGHTH OF A MACHINE CYCLE.
DLY = THE MAXIMUM TIME DELAY (IN NANOSECONDS) FOR DATA TO
PROPAGATE THROUGH A REGISTER, ADDER, MULTIPLIER,
MEMORY, BUS, OR ZERO DETECTOR.
INTEGER*4 BITWDTH, MAXN, COMND, TIMTCK,
>REGLY, ADDDLY, MPYDLY, MEMDLY, BUSDLY, ZRODLY
LOGICAL SCALE
COMMON /VAR/BTWIDTH,MAXN,COMND,TIMTCK,SCALE,
>REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY

C
C INPUT THE PROGRAM VARIABLES.
15 FORMAT(15)
READ(1,15)COMND
READ(1,15)BTWIDTH
READ(1,15)MAXN
READ(1,15)TIMTCK
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
SET MPYDLY TO COMPENSATE FOR THE METHOD OF SIMULATION.
MPYDLY = MPYDLY - REGDLY
READ(1,15)MEMDLY
SET MEMDLY TO COMPENSATE FOR THE METHOD OF SIMULATION.
MEMDLY = MEMDLY - REGDLY
READ(1,15)BUSDLY
READ(1,15)ZRODLY

C
C INITIATE PROGRAM VARIABLES.
CALL SETMSK
C STORE THE TRIGONOMETRIC COEFFICIENTS.
CALL TRGINT
C
C STORE DATA SEQUENCE IN MEMORY.
CALL DATSTR
C
C PERFORM INITIAL PASS OF COMPUTATION.
C THIS IS ANALOGOUS TO THE 'INIT' MICRO INSTRUCTION.
CALL UNIT
C
C
**MAIN COMPUTATION LOOP.**

```
125 100 CALL TIMER
126 CALL CTLT0
127 CALL MEMT0
128 CALL FFTUT0
129 CALL TIMER
130 CALL T1
131 CALL CTLT1
132 CALL MEMT1
133 CALL FFTUT1
134 CALL TIMER
135 CALL T2
136 CALL CTLT2
137 CALL MEMT2
138 CALL FFTUT2
139 CALL TIMER
140 CALL T3
141 CALL CTLT3
142 CALL MEMT3
143 CALL FFTUT3
144 CALL TIMER
145 CALL T4
146 CALL CTLT4
147 CALL MEMT4
148 CALL FFTUT4
149 CALL TIMER
150 CALL T5
151 CALL CTLT5
152 CALL MEMT5
153 CALL FFTUT5
154 CALL TIMER
155 CALL T6
156 CALL CTLT6
157 CALL MEMT6
158 CALL FFTUT6
159 CALL TIMER
160 CALL TIMER
```
C TIME T7
162 CALL CTLT7
163 CALL MEMT7
164 CALL FFTUT7
165 GO TO 100
166 C STOP
168 END
169 C
170 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 C **********************
175 C LOGICAL FUNCTION ZERODT(Z,ZTAG)
176 C **********************
177 INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
178 LOGICAL UFLAG,FINIS
179 COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
180 UFLAG,FINIS
181 INTEGER*4 BTWDT,CXN,COMND,TIMTCK,
182 >REGLY,ADDL1,MPYL1,MEML1,BUSL1,ZRODL1
183 LOGICAL SCALE
184 COMMON /VAR/BTWDT,CXN,COMND,TIMTCK,SCALE,
185 >REGLY,ADDL1,MPYL1,MEML1,BUSL1,ZRODL1
186 INTEGER*4 Z,ZTAG
187 C CHECK PROPAGATION 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 TRGINT IS USED TO STORE THE TRIG TABLE IN TMEM.
195 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 **********************
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
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
COMMON /TG/PI
INTEGER*4 L, M, N, TEMP
DOUBLE PRECISION PI, X, Y, Z
C
PI = 3.14159265358979323846D0
K = (MAXN/4) + 1
Z = MAXN/2.0D0
L = 0
CALL BSET(L, 0)
DO 1000 I = L, K
   J = I-1
   Y = J/Z
   X = DSIN(PI * Y)
   Y = (X * L) - 5.0D-1
   TMEM(I) = Y
   CALL ROUND(TMEM(I))
1000 CONTINUE
RETURN
C
C DATSTR IS USED TO STORE THE INITIAL DATA SEQUENCE TO MEMORY.
C ************
C ENTRY DATSTR
C ************
C
35 FORMAT(2(I20))
DO 800 I = 1, MAXN
   RMEM(I) = 0
   IMEM(I) = 0
800 CONTINUE
WRITE(3, 30)
WRITE(3, 30)
WRITE(3, 41)
WRITE(3, 42)
C READ DATA SEQUENCE AND STORE IN BIT REVERSED ORDER.
C DATA IS ROUNDED TO SPECIFIED BIT WIDTH.
DO 810 I = 1, COMMD
   K = I - 1
   J = K
   CALL BITREV(J)
   J = J + 1
   READ(2, 35) RMEM(J), IMEM(J)
   WRITE(3, 50) K, RMEM(J), IMEM(J)
   CALL ROUND(RMEN(J))
   CALL ROUND(IMEM(J))
810 CONTINUE
GO TO 820
RETURN
C
C THE FOLLOWING SUBROUTINE PRINTS THE RESULTS OF THE SIMULATION.
C ************
C ENTRY RESULT
C ************
10 FORMAT(´TOTAL TIME IS´),(2X,I11),´ NANOSECONDS.´))
20 FORMAT(´SEQUENCE LENGTH IS´),(1X,I6),´ DATA POINTS.´)
24 FORMAT(´COMPUTATIONS MODELLING A´),(I4),´ BIT MACHINE.´)
33 FORMAT(´THE DATA IS SCALED BY 1/N.´)
25 FORMAT(´ONE EIGHTH OF A MACHINE CYCLE IS´)
>,(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) BTWIDTH
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-BTWIDTH))
317 830 DO 110 I = 1, MAXN, R0
318 TEMP = I - 1
319 L = RMEM(I) / N
320 M = IMEM(I) / N
WRITE(3,50)TEMP,L,M
CONTINUE
RETURN
C
C UPDATE TIME (INCREASING). EACH UNIT REPRESENTS 25 NANOSECONDS.
C **********
ENTRY TIMER
C **********
TIME = TIME + TIMTCK
RETURN
C
C SETMSK Initializes the masks used for truncating to proper bit width.
C **********
ENTRY SETMSK
C **********
C SET BINIT for use as the initial index for bit reversal.
BINIT = 0
DO 840 I = 1,31
   IF (BTEST(MAXI,I)) BINIT = I + 1
840 CONTINUE
C CLEAR END-OF-COMPUTATION INDICATOR.
FINIS = .FALSE.
C SET INITIAL TIME.
TIME = 0
INITIM = TIME
C SET R0 = MAXN / COMND.
R0 = ISHFT(COMND,-1)
CALL BITREV(R0)
C SET MASKS.
BITMSK = ~1
TEMP = 32 ~ BTWDTH
DO 900 I = 1,TEMP
   RNDMSK = BITMSK
   BITMSK = ISHFT(RNDMSK,1)
900 CONTINUE
RNDMSK = IEO(BITMSK,RNDMSK)
CTLMSK = MAXN ~ 1
RETURN
THE FOLLOWING SUBROUTINE DETERMINES THE PROPER ADDRESSES FOR
RETRIEVAL OF TRIGONOMETRIC COEFFICIENTS, BASED ON THE VALUE
OF REGISTER RTRIG.

************
ENTRY TRIGIT
************

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

THE FOLLOWING SUBROUTINE PERFORMS THE FIRST CYCLE OF COMPUTATION.
THIS IS ANALOGOUS TO THE 'INIT' MICRO-INSTRUCTION.

************
SUBROUTINE UNIT
************

INTEGER*4 BTWDTIM,MAXN,COMND,TIMTCK,
>REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
LOGICAL SCALE
COMMON /VAR/BTWDTIM,MAXN,COMND,TIMTCK,SCALE,
>REGDLY,ADDDLY,MPYDLY,MEMDLY,BUSDLY,ZRODLY
INTEGER*4 BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,UNIT,
LOGICAL UFLAG,FINIS
COMMON /CONST/BITMSK,RNDMSK,CTLMSK,INITIM,TIME,UTYPE,UNIT,
>UFLAG,FINIS
INTEGER*4 R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
>RADD,TADD,RIADD,R2ADD,RTRIGS,RTRIGC,
>R0TAG,R1TAG,R1PTAG,R2TAG,R2PTAG,R3TAG,R4TAG,R5TAG,
>R5PTAG,R6TAG,R7TAG,R8TAG,R9TAG,R10TAG,R11TAG,R21TAG
COMMON /R0,R1,R1P,R2,R2P,R3,R4,R5,R5P,R6,RF7,RTRIG,
>RADD,TADD,RIADD,R2ADD,RTRIGS,RTRIGC,
C TIME T0. INITIALIZE VARIABLES.
R0TAG = INITIM
CALL TIMER
CALL TIMER

C TIME T2. SET RF7.
RF7 = R0
CALL BITREV(RF7)
RF7 = RF7 - 1

C SET RF7TAG.
CALL CTLSTR(RF7, R0TAG, RF7, RF7TAG)
CALL CTLSTR(R0, R0TAG, R3, R3TAG)
CALL CTLSTR(0, R0TAG, R5, R5TAG)
CALL CTLSTR(MAXN/2, R0TAG, R5P, R5PTAG)
CALL TIMER

C TIME T3.
CALL ADDER(R5P, R5PTAG, RTRIG, RRTAG, TADD, TADTAG, .TRUE.)
CALL ADDER(0, R0TAG, R3, R3TAG, RADD, RADTAG, .TRUE.)
CALL TIMER

C TIME T4.
CALL ADDER(0, R1TAG, 0, RADTAG, R1ADD, R1ATAG, .TRUE.)
CALL ADDER(0, R1TAG, RADD, RADTAG, R2ADD, R2ATAG, .TRUE.)
CALL CTLSTR(R5, R5TAG, R4, R4TAG)
CALL CTLSTR(RF7, RF7TAG, R6, R6TAG)
CALL CTLSTR(0, R0TAG, RTRIG, RRTAG)
CALL TRIGIT
CALL TIMER

C TIME T5.
CALL CTLSTR(R0, R0TAG, R2P, R2PTAG)
CALL CTLSTR(R2ADD, R2ATAG, R2, R2TAG)
FORMAT(I15)

CALL TIMER

C TIME T6. SET SKIPSICE INDICATOR.

UTYPE= 0

CALL MEMT6

CALL TIMER

C TIME T7.

CALL CTSTR(R1, R1TAG, R1P, R1PTAG)

CALL CTSTR(R1ADD, R1ATAG, R1, R1TAG)

CALL BUSSTR(R2, R2TAG, ABUS, ABSTAG)

CALL TGCALL(TMEM, TASBUS, TASTAG, MBRTS, TMSTAG)

CALL TGCALL(TMEM, TACBUS, TACTAG, MBRTC, TMCTAG)

RETURN

END

C

C THE FOLLOWING SUBROUTINE PERFORMS THE LAST CYCLE OF THE COMPUTATION.

C ************

C SUBROUTINE FINI

C ************

C TIME T2.

CALL MEMT2

CALL FFTUT2

CALL TIMER

C TIME T3.

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
C TIME T7.
CALL CTLT7
CALL MEMT7
C PRINT RESULTS OF SIMULATION.
CALL RESULT
STOP
END

C
C THE FOLLOWING SUBROUTINE PERFORMS A BIT REVERSAL OF A 12-BIT QUANTITY.
C
C **********************
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 10 I = BINIT,31
   IF (BTEST(BRV,I)) CALL BSET(TEMP,31+BINIT-I)
10 CONTINUE
BRV = TEMP
RETURN
END

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'
R5 = CURRENT 'SKIPCOUNT' FOR RELOADING R4
R6 = DECREMENTED COUNTER REPRESENTING REMAINING COUNT BEFORE 'LOGSLICE'
RF7 = A FICTITIOUS REGISTER CONTAINING BIT-REVERSED R0 MINUS ONE
RADD = THE COMBINATIONAL OUTPUT OF THE R0 + R3 ADDER
RTRIG = THE ADDRESS OF THE TRIG COEFFICIENT

TIME T0.

******************

SUBROUTINE CTLT0

******************

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, TRIGS, TRIGC,
R0TAG, R1TAG, R1PTAG, R2TAG, R2PTAG, R3TAG, R4TAG, R5TAG,
R5PTAG, R6TAG, RF7TAG, R7TAG, R8TAG, TRIGS, TRIGC,
COMMON /CTL/R0, R1, R1P, R2, R2P, R3, R4, R5, R5P, R6, RF7, RTRIG,
RADD, TADD, R1ADD, R2ADD, TRIGS, TRIGC,
R0TAG, R1TAG, R1PTAG, R2TAG, R2PTAG, R3TAG, R4TAG, R5TAG,
R5PTAG, R6TAG, RF7TAG, R7TAG, R8TAG, R9TAG

TIME T0. SET UFLAG TO CONTROL ADDERS IN FFTU.
UFLAG = .FALSE.
IF (BTEST(RTRIG, 21)) UFLAG = .TRUE.
RETURN

TIME T1. NO ACTION.

 ENTRY CTLT1

RETURN
**TIME T2. SET R5, R5P, AND R3 IF THE MICROINSTRUCTION IS LOGSLICE.**

**ENTRY CTLT2**

**TIME T2. CHECK TERMINATING CONDITION.**

**ENTRY CTLT3**

**TIME T3. ESTABLISH THE OUTPUT OF THE ADDER FOR FUTURE UPDATING OF R1 AND R2.**

**ENTRY CTLT1**
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.)
2101 CALL ADDER (R2, R2TAG, RADD, RADTAG, R2ADD, R2ATAG, .TRUE.)
2102 CALL CTLSTR (R4-1, R4TAG, R4, R4TAG)
2103 CALL CTLSTR (R6-1, R6TAG, R6, R6TAG)
2104 CALL CTLSTR (TADD, TADTAG, RTRIG, RTTAG)
2105 CALL BCLR (RTRIG, 20)
C DETERMINE THE ADDRESSES FOR THE NEXT TRIG COEFFICIENTS.
2106 CALL TRIGIT
2107 RETURN
C SKIPSICE. LOAD R4, DEC R6. PROPOGATE R1ADD, R2ADD.
2110 CALL ADDER (R1, R1TAG, RADD, RADTAG, R1ADD, R1ATAG, .TRUE.)
2111 CALL ADDER (R2, R2TAG, RADD, RADTAG, R2ADD, R2ATAG, .TRUE.)
2112 CALL CTLSTR (R5, R5TAG, R4, R4TAG)
2113 CALL CTLSTR (R6-1, R6TAG, R6, R6TAG)
2114 CALL CTLSTR (0, RTTAG, RTRIG, RTTAG)
C DETERMINE THE TRIG COEFFICIENTS BASED ON RTRIG.
2115 CALL TRIGIT
2116 RETURN
C LOGSLICE. LOAD R4, LOAD R6. ADDERS SET FOR RESETTING R1, R2.
2120 CALL ADDER (0, R1TAG, 0, RADTAG, R1ADD, R1ATAG, .TRUE.)
2121 CALL ADDER (0, R2TAG, RADD, RADTAG, R2ADD, R2ATAG, .TRUE.)
2122 CALL CTLSTR (R5, R5TAG, R4, R4TAG)
2123 CALL CTLSTR (RF7, RF7TAG, R6, R6TAG)
2124 CALL CTLSTR (0, RTTAG, RTRIG, RTTAG)
C DETERMINE TRIG COEFFICIENTS BASED ON RTRIG.
2125 CALL TRIGIT
2126 RETURN
C
C TIME T5. UPDATE R2, R2P.
C
ENTRY CTLT5
C
C
CALL CTLSSTR(R2, R2TAG, R2P, R2PTAG)
CALL CTLSSTR(R2ADD, R2ATAG, R2, R2TAG)
RETURN

C
C
TIME T6. SET ADDRESS FOR NEXT MICROINSTRUCTION.
C
**********
ENTRY CTLT6
C
**********
NORMAL.
UTYPE = -1
SKIPSlice.
IF (ZERODT(R4, R4TAG)) UTYPE = 0
C
LOGSLICE.
IF (ZERODT(R6, R6TAG)) UTYPE = 1
RETURN

C
TIME T7. UPDATE R1 AND R2P.
C
**********
ENTRY CTLT7
C
**********
CALL CTLSSTR(R1, R1TAG, R1P, R1PTAG)
CALL CTLSSTR(R1ADD, R1ATAG, R1, R1TAG)
RETURN
END

C

THE FOLLOWING SUBROUTINES (PREFIX MEM) MODEL THE TRANSFERS OF
ADDRESSES AND DATA TO AND FROM MEMORY.

C
TIME T0.
C
**********
SUBROUTINE MEMT0
C
**********
INTEGER*4 RMEM(4096), IMEM(4096), TMEM(1025),
>RMTAG, IMTAG, TMSTAG, TMCTAG, MBRR, MBRI, MBRTS, MBRTC

   COMMON /MEM/RMEM, IMEM,

   >RMTAG, IMTAG, TMSTAG, TMCTAG, MBRR, MBRI, MBRTS, MBRTC
   INTEGER*4 DBUS, 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, R7TAG, RADTAG, TADTAG, T1ATAG, 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, R7TAG, 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)
C
C CALL RECALL (IMEM, ABUS, ABSTAG, MBRI, IMTAG)
C
RETURN
C
C TIME T1. MEMORY DATA TO DATA BUSSES; R2P TO ADDRESS BUS.
C TRIG DATA TO TRIG DATA BUSSES.
C
C ************
C ENTRY MENT1
C ************
C
C DIVIDE THE OPERANDS BY 2 IF SCALING IS INDICATED.
C
C IF (.NOT.SCALE) GO TO 700
C
MBRR = IŞHFT(MBRR, -1)
C
IF (BTEST(MBRR, 1)) CALL BSET(MBRR, 0)
MBRI = IŞHFT(MBRI, -1)
C
IF (BTEST(MBRI, 1)) CALL BSET(MBRI, 0)
CALL BUSSTR(MBRR,RMTAG,DRBUS,DRTAG)
CALL BUSSTR(MBRI,IMTAG,DIBUS,DITAG)
CALL BUSSTR(R2P,R2PTAG,ABUS,ABSTAG)
CALL BUSSTR(MBRTS,TMTAG,TSBUS,TSTAG)
CALL BUSSTR(MBRTC,TMTAG,TCBUS,TCTAG)
RETURN

C
TIME T2. NO ACTION.

C
*************
ENTRY MEMT2
*************
RETURN

C
TIME T3. R1 TO ADDRESS BUS; STORE 'B' RESULT.

C
*************
ENTRY MEMT3
*************
STORE 'B SUM'.
CALL STORE(RRBUS,RRTAG,RMEM,RMTAG,ABUS,ABSTAG)
CALL STORE(RIBUS,RITAG,IMEM,IMTAG,ABUS,ABSTAG)
CALL BUSSTR(R1,R1TAG,ABUS,ABSTAG)
RETURN

C
TIME T4. MEMORY DATA TO 'A'.

C
*************
ENTRY MEMT4
*************
RECALL 'A'.
CALL RECALL(RMEM,ABUS,ABSTAG,MBRR,RMTAG)
CALL RECALL(IMEM,ABUS,ABSTAG,MBRI,IMTAG)
RETURN
C
  TIME T5. MEMORY DATA TO DATA BUSSES; R1P TO ADDRESS BUS.
C
C
  **********
C  ENTRY MEMT5
C  **********
  DIVIDE THE OPERANDS BY 2 IF SCALING IS INDICATED.
  IF (.NOT.SCALE) GO TO 710
    MBRR = ISHFT(MBRR,-1)
    IF (BTEST(MBRR,1)) CALL BSET(MBRR,0)
    MBRI = ISHFT(MBRI,-1)
    IF (BTEST(MBRI,1)) CALL BSET(MBRI,0)
    CALL BUSSTR(MBRR,RMTAG,DRBUS,DRTAG)
    CALL BUSSTR(MBRI,IMTAG,DIBUS,DITAG)
    CALL BUSSTR(R1P,R1PTAG,ABUS,ABSTAG)
  RETURN
C
  TIME T6. TRIG ADDRESS TO TRIG ADDRESS BUS.
C
C
  **********
C  ENTRY MEMT6
C  **********
C
C  CALL BUSSTR(RTRIGS,RRTAG,TASBUS,TASTAG)
C  CALL BUSSTR(RTRIGC,RRTAG,TACBUS,TACTAG)
  RETURN
C
  TIME T7. RESULT 'A' (DATA BUS) TO MEMORY; R2 TO ADDRESS BUS;
C  RECALL TRIG DATA.
C
C
  **********
C  ENTRY MEMT7
C  **********
C
C
C  STORE 'A SUM'
C
  CALL STORE(RRBUS,RRTAG,RMEM,RMTAG,ABUS,ABSTAG)
CALL STORE (RIBUS, RITAG, IMEM, IMTAG, ABUS, ABTAG)
CALL BUSSTR (R2, R2TAG, ABUS, ABTAG)
CALL TGCALL (TMEM, TASBUS, TASTAG, MBRTS, TMSTAG)
CALL TGCALL (TMEM, TACBUS, TACTAG, MBRTC, TMCTAG)
RETURN
END

C

THE FOLLOWING SUBROUTINES (PREFIX FFTU) MODEL THE OPERATIONS
OF THE 'FAST FOURIER TRANSFORM UNIT' WHICH PERFORMS
ONE FULL BUTTERFLY PER CYCLE.

C

TIME T0. PROPAGATE A TO THE ADDERS
TO FORM THE SUM AND DIFFERENCE.

C

***************

SUBROUTINE FFTU T0

***************

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 BTWDTCH, 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 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 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,
PROPAGATE 'A' TO THE ADDERS TO FORM THE SUM AND DIFFERENCE.

CALL ADDER(AI, ARTAG, BIP, BIPTAG, ADD5, AD5TAG, .TRUE.)
CALL ADDER(AI, ARTAG, BIP, BIPTAG, ADD6, AD6TAG, .FALSE.)
RETURN

TIME T1. PROPAGATE SUM AND DIFFERENCE TO THE OUTPUT REGISTERS.

ENTRY FFTUT1

CALL REGSTR(ADD3, AD3TAG, ARSUM, ARSTAG)
CALL REGSTR(ADD4, AD4TAG, BRSUM, BRSTAG)
CALL REGSTR(ADD5, AD5TAG, ARSUM, ARSTAG)
CALL REGSTR(ADD6, AD6TAG, BRSUM, BRSTAG)
RETURN

TIME T2. CLOCK IN 'B' AND TRIG OPERANDS FROM MEMORY, AND PLACE 'B' OUTPUTS OF THE FFTU ON THE MEMORY BUS FOR STORAGE.

ENTRY FFTUT2

CALL REGSTR(DRBUS, DRTAG, BR, BRTAG)
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
TIME T3. PROPOGATE INITIAL OPERANDS TO THE MULTIPLIER INPUTS.

C
************
ENTRY FFTUT3
************
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
TIME T4. NO OPERATION.

C
************
ENTRY FFTUT4
************
RETURN

C
TIME T5. NO OPERATION.

C
************
ENTRY FFTUT5
************
RETURN

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 ************
ENTRY FFTUT6
C ************
CALL REGSTR(DRBUS,DRTAG,AR,ARTAG)
CALL REGSTR(DIBUS,DITAG,AL,AILTAG)
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 TIME T7. PROPOGATE 'B PRIME' RESULTS TO THE LAST ADDER STAGE.
C ************
ENTRY FFTUT7
C ************
CALL REGSTR(ADD1,AD1TAG,BRP,BRPTAG)
CALL REGSTR(ADD2,AD2TAG,BIP,BIPTAG)
RETURN
END

C THE FOLLOWING SUBROUTINE MODELS A REGISTER RECEIVING DATA
C 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,RNDSK,CTLMSK,INITIM,TIME,UTYPE,BINIT
LOGICAL UFLAG,FINIS
COMMON /CONST/BITMSK,RNDSK,CTLMSK,INITIM,TIME,UTYPE,BINIT,
>UFLAG,FINIS

C CHECK PROPOGATION TIME.
964 IF (RINTAG.GT.TIME) CALL ERROR(2)
965 ROUT = RIN
966 C SAVE UPPER ORDER BITS.
967 CALL TRUNC(ROUT)
968 ROUTAG = TIME + REGDLY
969 RETURN

C THE FOLLOWING SUBROUTINE PERFORMS REGISTER TRANSFERS FOR THE
971 C CONTROL SECTION OF THE SIMULATION. (LOWER ORDER MASK)

C **************
975 ENTRY CTLSTR
976 C **************
977 C CHECK PROPOGATION DELAY.
979 IF (RINTAG.GT.TIME) CALL ERROR(3)
981 ROUT = RIN
982 C SAVE LOWER ORDER BITS.
983 CALL CTRUNC(ROUT)
984 ROUTAG = TIME + REGDLY
985 RETURN

C THE FOLLOWING SUBROUTINE TAKES DATA FROM A REGISTER AND TRANSFERS
988 C IT TO THE DESIGNATED BUS.

C **************
991 ENTRY BUSSTR
992 C **************
994 ROUTAG = MAX0(TIME,RINTAG) + BUSDLY
995 ROUT = RIN
996 RETURN
997 END

C THE FOLLOWING SUBROUTINE READS MEMORY AT THE ADDRESS SPECIFIED
C BY ADDR.

********************************************************************
SUBROUTINE RECALL (MEM, ADDR, ADRTAG, MBR, MBRTAG)
********************************************************************
INTEGER*4 MEM(4096), ADDR, ADRTAG, MBR, MBRTAG
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 CHECK THAT PREVIOUS MEMORY REFERENCE IS COMPLETE.
IF (MBRTAG.GT.TIME) CALL ERROR(4)
C SET THE TAG TO INDICATE TIME-OF-COMPLETION.
MBRTAG = MAX0(TIME, ADRTAG) + MEMDLY
MBR = MEM(ADDR + 1)
RETURN
END

C THE FOLLOWING SUBROUTINE STORES DATA TO MEMORY FROM THE DESIGNATED
C BUS, TO THE ADDRESS SPECIFIED BY ADDR.
********************************************************************
SUBROUTINE STORE(BUS, BUSTAG, MEM, MBRTAG, ADDR, ADRTAG)
********************************************************************
INTEGER*4 BUS, BUSTAG, MEM(4096), MBRTAG, ADDR, ADRTAG
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
CHECK THAT THE PREVIOUS MEMORY REFERENCE IS COMPLETE.

IF (MBR_TAG.GT.TIME) CALL ERROR(5)

CHECK THAT ADDRESS AND DATA ARE READY IN TIME FOR STORE.

IF (ADR_TAG.GT.TIME) CALL ERROR(6)

IF (BUSTAG.GT.TIME) CALL ERROR(7)

MEM(ADDR + 1) = BUS

SET TAG TO ALLOW FOR STORAGE OF DATA.

MBR_TAG = TIME + MEMDLY

RETURN

END

THE FOLLOWING SUBROUTINE READS TRIG MEMORY AT THE ADDRESS SPECIFIED

BY ADDR.

***************

SUBROUTINE TGCALL(MEM, ADDR, ADRTAG, MBR, MBRTAG)

***************

INTEGER*4 MEM(1025), ADDR, ADRTAG, MBR, MBRTAG

INTEGER*4 BTWDTH, MAXN, COMND, TIMTCK,

>REGLDY, ADDDLY, MPYDLY, MEMDLY, BUSDLY, ZRODLY

LOGICAL SCALE

COMMON /VAR/BTWDTH, MAXN, COMND, TIMTCK, SCALE,

>REGLDY, 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

CHECK THAT PREVIOUS MEMORY REFERENCE IS COMPLETE.

IF (MBR_TAG.GT.TIME) CALL ERROR(8)

SET TAG TO ALLOW TIME FOR MEMORY REFERENCE.

MBR_TAG = MAX(0, TIME, ADRTAG) + MEMDLY

MBR = MEM(ADDR + 1)

RETURN

END

THE FOLLOWING SUBROUTINE TAKES A1 PLUS(MINUS) A2 TO

PRODUCE RESULT A3, IF OP IS .TRUE. (.FALSE.).

***************
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, TMTCK,
>REGDLY, ADDDLY, MPYDLY, MEMDLY, BUSDLY, ZRODLY
LOGICAL SCALE
COMMON /VAR/BTWDTH, MAXN, COMND, TMTCK, 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

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

MPY FORMS THE PRODUCT M3 FROM OPERANDS M1 AND M2.
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, TMTCK,
>REGDLY, ADDDLY, MPYDLY, MEMDLY, BUSDLY, ZRODLY
LOGICAL SCALE
COMMON /VAR/BTWDTH, MAXN, COMND, TMTCK, 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 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 THE FOLLOWING SUBROUTINE RoundS THE NUMBER TO THE SPECIFIED Bit WIDTH.
C
C ******************
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)
RETURN

C
THE FOLLOWING SUBROUTINE TRUNCATES THE NUMBER TO THE SPECIFIED BIT WIDTH.
C
************
C
ENTRY TRUNC
C
************
C
RND = IAND(RND,BITMSK)
RETURN
C
C
THE FOLLOWING SUBROUTINE MASKS UPPER BITS TO FORM A WORDWIDTH
C
OF LOG2(MAXN).
C
************
C
ENTRY CTRUNC
C
************
C
RND = IAND(RND,CTLMSK)
RETURN
END
C
C
THE ERROR ROUTINE PRINTS THE SOURCE LOCATION OF THE ERROR,
C
AND STOPS THE PROGRAM.
C
************
C
SUBROUTINE ERROR(ER)
C
************
INTEGER*4 ER
C
77 FORMAT (,'#ERROR OCCURED AT LOCATION #'),(2X,I5))
C
WRITE(5,77)ER
C
STOP
C
RETURN
C
END
C
$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:

<table>
<thead>
<tr>
<th>( n )</th>
<th>REAL DATA</th>
<th>IMAGINARY DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1310720000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1210947380</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>926819000</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>501590828</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>501590828</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>926819000</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1210947380</td>
<td>0</td>
</tr>
</tbody>
</table>

### SEQUENCE STORED IN MEMORY:

<table>
<thead>
<tr>
<th>( m )</th>
<th>REAL DATA</th>
<th>IMAGINARY DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20000</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>14142</td>
<td>0</td>
</tr>
<tr>
<td>1536</td>
<td>14142</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>18478</td>
<td>0</td>
</tr>
<tr>
<td>2560</td>
<td>7654</td>
<td>0</td>
</tr>
<tr>
<td>3072</td>
<td>7654</td>
<td>0</td>
</tr>
<tr>
<td>3584</td>
<td>18478</td>
<td>0</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>( k )</th>
<th>REAL DATA</th>
<th>IMAGINARY DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12568</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>4414</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>-1036</td>
<td>0</td>
</tr>
<tr>
<td>1536</td>
<td>586</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>-498</td>
<td>0</td>
</tr>
<tr>
<td>2560</td>
<td>586</td>
<td>0</td>
</tr>
<tr>
<td>3072</td>
<td>-1036</td>
<td>0</td>
</tr>
<tr>
<td>3584</td>
<td>4414</td>
<td>0</td>
</tr>
</tbody>
</table>