Architectural considerations of a signal processor under microprogram control

by Y. S. Wu

International Business Machines Corporation
Gaithersburg, Maryland

INTRODUCTION

The application of microprogramming to seismic, acoustic and radar processing is well-known. The system architecture required to address wide bandwidth signal processing problems is of a general form shown in Figure 1. In order to provide the high throughput which is required by digital signal processing, parallelism is generally accepted as the proper design concept for the signal processing arithmetic element. A microprogram processor (or a host processor) could be used to control the arithmetic element which can be either an associative processor, functional memory, parallel ensemble, or a vector processor. The efficient design of the microprogram processor is the key to insure the high duty cycle utilization of the expensive arithmetic element hardware. Parallel architectures fall far short of their expectations because of the control problem associated with keeping all the hardware usefully occupied all the time.

This paper surveys basic digital signal processing algorithms. It proposes a signal processing architecture consisting of a microprogrammed control processor (MCP), a highly efficient sequential signal processing arithmetic unit (SPAU) and necessary buffer memories. Emphasis is placed on the MCP architecture because of its importance in enhancing the system performance. An application example, optimum processing in frequency domain, is given to verify the applicability of the architecture.

Hardware technology surveys, and fabrication and packaging considerations are beyond the scope of this paper. The firmware and microprogramming support software discussions are also omitted.

BASIC DIGITAL SIGNAL PROCESSING ALGORITHMS

A wealth of literature, including several very good text books is available in signal processing and digital signal processing. A brief description of some of the processing algorithms is included here. In general, linear filter theory and spectrum analysis techniques form the basis of digital signal processing. Time-domain and frequency domain equivalence of processing is widely assumed. In practice, time-domain digital processing is only used in real time applications with relatively short sample data block. The fast Fourier transform (FFT) algorithm provides the efficient digital processing link between time and frequency domains, and frequency domain processing is preferred for narrow-band analysis with fine resolution and large quantities of input data.

Time-domain processing

Consider a sampled signal \( x(t_n) \), where \( t_n = nT \), \( n = 0, 1, \ldots, n \) and \( T \) is the sample period. A linear digital filter is written as:

\[
 y(t_n) = \sum_{m=0}^{N} h(t_m) x(t_{n-m})
\]

where \( y(t_n) \) are filtered outputs of \( x(t_n) \) and \( h(t_n) \) sampled impulse responses of the filter.

Similarly, the correlation function of \( f(t_n) \) and \( g(t_n) \) can be written as:

\[
 l(t_n) = \sum_{m=-N}^{N} f(t_{n+m}) g(t_m)
\]

Another type of time-domain processing, which is known as recursive filter, is commonly used for limited number of samples to compute successive value of out-

* Presently employed at Naval Research Laboratory, Washington, D.C.
puts. Let \( \{x(t_n)\} \) be the input signal samples, then the filtered outputs \( \{y(t_n)\} \) are expressed by the linear difference equation:

\[
y(t_n) = \sum_{k=0}^{N} W_k y(t_{n-k}) + \sum_{k=0}^{r} \lambda_k x(t_{n-k})
\]

(3)

where \( \{w_k\} \) and \( \{\lambda_k\} \) are weighting or filter coefficients of the recursive filter.

For \( N \) sample points, time-domain processing requires \( N^2 \) multiply-sum operations. Therefore, real time application of time-domain processing is limited to short sample data blocks only. Typically, \( N \) equals 64 or less.

**Frequency domain processing**

Let \( \{X(f_j)\} \) be the discrete Fourier transform of a signal \( \{x(t_n)\} \) that

\[
X(f_j) = \sum_{n=0}^{N-1} x(n) \exp(-i2\pi nf_j)
\]

(4)

where \( i = \sqrt{-1} \), \( j = 0, 1, 2, \ldots N - 1 \) and \( f_j = j(1/NT) \).

This discrete transform involves a couple of assumptions. Equally spaced time samples are assumed. Also the sampling rate must be above the Nyquist rate, twice the frequency of the highest frequency in the waveform being sampled. When these criteria are met, the discrete Fourier transform has parallel properties to the continuous transform. The original waveform can be completely recreated from the samples. Transformations between the time and frequency domains are performed by using the discrete transform and its inverse. It can be shown that the equivalent frequency domain digital filter in equation (1) can be written as:

\[
Y(f_j) = X(f_j) H(f_j)
\]

(5)

where \( \{Y(f_j)\} \) and \( \{H(f_j)\} \) are discrete Fourier transforms of \( \{y(t_n)\} \) and \( \{h(t_n)\} \) respectively.

Similarly, the equivalent correlation function in frequency domain can be shown as:

\[
\Phi(f_j) = \tilde{\Phi}(f_j) G(f_j)
\]

(6)

where \( \{\tilde{\Phi}(f_j)\} \) is the complex conjugate of the discrete Fourier transform of \( \{f(t_n)\} \).

As a special case, the power spectrum density function is:

\[
|F(f_j)|^2 = \tilde{\Phi}(f_j) F(f_j)
\]

(7)

It is seen that for \( N \) frequency domain samples, the equivalent digital filtering or correlation function requires \( N \) complex multiplications instead of \( N^2 \) product-sum operations in the time-domain. Tremendous computational savings for large \( N \) can be achieved if an efficient processing link between time domain and frequency domain is established. The fast Fourier transform algorithm is this missing link.

**Fast fourier transform**

Since 1965, a great deal of attention has been given to the FFT algorithm by the digital signal processing community. Interested readers can find detailed derivations and variations of the algorithm in references listed in the Bibliography. A simple derivation is included below.

Let's rewrite the discrete Fourier transform expression shown in equation (4).

\[
A_r = \sum_{k=0}^{N-1} x_k \exp(2\pi ikr/N) = \sum_{k=0}^{N-1} x_k W^r_k
\]

(8)

where: \( i = \sqrt{-1} \)

\[
W = \exp(2\pi i/N)
\]

\( N = \) number of samples

\( r = \) harmonic number = 0, 1, . . . , \( N - 1 \)

\( k = \) time-sample number = 0, 1, . . . , \( N - 1 \)

Thus \( A_r \) is the \( r \)th coefficient of the Fourier transform and \( x_k \) is the \( k \)th sample of the time series.

The samples, \( x_k \), may be complex, and the coefficients, \( A_r \), are almost always complex.

Working through an example in which \( N = 8 \) will illustrate some of the calculation short cuts.

In this case: \( j = 0, 1, \ldots, 7 \)

\( k = 0, 1, \ldots, 7 \)
To put these into binary form:

\[ j = j_0(2^2) + j_1(2^1) + j_0 \]

\[ k = k_0(2^2) + k_1(2^1) + k_0 \]

where: \( j_0, j_1, k_0, k_1, k_2 = 0, 1 \)

Thus:

\[ A(j_0, j_1, j_0) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \chi(k_0, k_1, k_0) [W^{(j_0+j_1+j_2)}(k_0+k_1+k_2)] \]  (10)

Now the \( W \) can be broken down further:

\[ W^{(j_0+j_1+j_2+k_0)} = [W^{(j_0+j_1+j_2)}]W^{k_0} \]

\[ W^{(j_0+j_1+j_2+k_0+k_1)} = [W^{(j_0+j_1+j_2)}]W^{(k_0+k_1)} \]

\[ W^{(j_0+j_1+j_2+k_0+k_1+k_2)} = W^{(j_0+j_1+j_2+k_0+k_1+k_2)} \]  (11)

Since \( W^8 = [\exp(2\pi i/8)]^8 = \exp(2\pi i) = 1 \), the bracketed terms equal one, and can be dropped from the computation. (Note that \( [\exp(2\pi i)]^P = 1^P = 1 \).) This saves many calculations.

Then \( A(j_0, j_1, j_0) \) can be obtained by sequentially calculating the \( x_3 \) as follows:

\[ x_3(j_0, j_1, j_0) = \sum_{k_0=0}^{1} (k_0, k_1, k_0) W^{j_0+k_0} \]

\[ x_3(j_0, j_1, j_0) = \sum_{k_1=0}^{1} 1 (j_0, k_1, k_0) W^{j_0+j_1+k_0} \]

\[ x_3(j_0, j_1, j_0) = \sum_{k_2=0}^{1} x_3(j_0, j_1, k_0) W^{(j_0+j_1+j_2+k_0)} \]

\[ A(j_0, j_1, j_0) = x_3(j_0, j_1, j_0) \]

Once these computation savings were found, one may generalize that for \( N \) point fast Fourier transform \( \frac{1}{2} \log_2 N \) complex multiplications and summations are required. For the equivalent digital filter operation in equation (1), \( N \log_2 N \) complex multiplications and summations are required. For the equivalent digital filter operation in equation (1), \( N \log_2 N \) complex multiplications are performed including the fast Fourier transform on input samples and the inverse transform to obtain time domain filtered outputs. Comparing with \( N^2 \) operations required for (1), this is a worthwhile saving in processing load, when \( N \) is large.

What are the basic operations?

Product-sum and complex multiplications!

A PROPOSED SIGNAL PROCESSOR ORGANIZATION

The basic arithmetic operation performed by a signal processor is the high speed multiplication in the form of product-sum for time-domain processing and complex multiply for the frequency-domain computations and the FFT algorithm. These operations are always performed on arrays or blocks of sensor data. In other words, signal processing deals exclusively with 'structured' data. A system architect faces:

1. The design of a high speed Signal Processing Arithmetic Unit (SPAU).

2. The problem of how to keep this arithmetic unit efficiently and usefully busy.

The latter poses a bigger challenge because it dictates the system through-put by collecting and controlling sensor inputs, and structuring the input data in a manner that can be most efficiently accepted by the SPAU. Typical functions are:

- I/O control
- Multiplexing or Demultiplexing
- Data conditioning
- Scaling
- Orthogonal addressing
- Format conversion
- Data buffering, blocking and packing.

The above listed preprocessing requirements for a SPAU are characterized by:

- Relatively simple algorithms
- Highly iterative operations
- Low precision

The advantages of using a Microprogrammed Control Processor (MCP) rather than special purpose hardware for these interface functions are the lower cost of such a general purpose architecture and the flexibility provided by the ability to change the microprogram. Furthermore, microprogramming implementation of these functions offers 5-10 times performance gain over a conventional general purpose computer of comparable technology.\(^1,2,8\) In addition, macro signal processing functions can be provided by properly sequencing the SPAU under microprogram control. Some of these
macros could be:

- Convolution Filter
- Recursive Filter
- Beam Forming
- FFT
- Inverse FFT
- Correlations
- Power Spectrum
- Filter
- Unpack
- Matrix Operations

By requiring the system to be under microprogrammed control, the designer is permitting a single piece of hardware to be specialized to a particular type of signal processing calculation by allowing for the design of an optimum ‘instruction set’ for that calculation to be loaded into the control store of the MCP.

Figure 2 depicts the functional diagram of a signal processor under microprogram control. The major components are System Storage, MCP, and SPAU.

**System storage hierarchy**

The structured nature of signal processing requires a block (or page) of data to be operated upon by the SPAU. Therefore, SPAU performance specifications define the buffer speed requirements for each ‘page’ and the system through-put requirement determines the transfer rate between the system bulk store and the buffer memories. A system storage hierarchy is implied. As to the microprogrammed Control Store (CS), one may consider each signal processing kernel as a module (or page) with highly repetitive execution duty cycle.

If Writable Control Store (WCS) is considered, a dynamic paging hierarchy can again be established for the microprogram execution. Since both data and the programs are sequential and block in nature for signal processing, no cache requirement is foreseen. For relatively long data blocks, buffer paging with respect to the system bulk storage can be accomplished through the MCP I/O control unit. No additional paging hardware will be required.

**Buffer memories**

At least two independent buffer memories will be required because of the over taxing demands on buffer memory cycles by the pipe-lined SPAU operations while MCP ALU and IOCU are preparing the next block of data for SPAU consumption. Two buffer memories in conjunction with MCP can only support a SPAU with a 4-cycle basic operation. If a 2-cycle SPAU is required, four independent buffer memories will be needed to achieve the desired performance.

A 64-bit buffer memory interface is proposed for the purpose of increasing real time instantaneous signal input bandwidth as well as enhancing the paging interface efficiency with the bulk system storage. Experience indicated that each buffer memory should be expandable vertically in 256 by 64-bit words increments up to 4K words. 1K to 2K 64-bit words buffer size is commonly seen. It is intended that the buffer memory is the same monolithic storage used for the microprogram control store for commonality and logistic simplicity. The speed of the buffer memory is defined by the operational requirement of SPAU.

**Control store**

A 64-bit wide control store compatible with the buffer memory is used for the microprogram storage. Each micro-instruction is capable of executing the following operations in parallel:

- Access the Buffer Memories for One or More Operands
- Manipulate Local Registers and Arithmetic Registers
- Perform Arithmetic and Logic Functions
- Decode and Status Sensing
- Decision Making
- Form Next Micro-Instruction Address
- Other Special Controls
Allowing multiple micro-instruction formats, one can easily reduce the micro-instruction width to 32-bit with a degradation of MCP performance by less than 15 percent. Double fetch of micro-instructions can be considered; however, microprogramming will be more difficult in this case.

Since writable control store is used in the system, dynamic paging of microprograms will be considered. Tight kernels are characteristic for MCP microprograms in the signal processing environment. Small page size (i.e., 64 micro-instructions) may be adequate for a dynamic control store size of not exceeding 1K 64-bit words.

**Bulk system storage**

Bulk System Storage can be provided as an I/O attachment to the MCP-SPAU signal processing system in the stand alone case. Or, the signal processing sub-system can be considered as interfacing through the bulk system storage with the central computer complex. In this case, bulk system storage can be a part of the shared CPU main memory or auxiliary large core storage (LCS).

The speed requirement of the bulk system storage is dictated by the type of processing performed in the MCP-SPAU. Assume a block of data with \( N \) points are first transformed into \( N \) spectral elements in the frequency domain and then filtered by \( N \) corresponding filter coefficients; the following are observed:

- **Data Transfers**
  - Bulk System Storage to MCP-SPAU
    - \( N \) Input Data Points
    - \( N \) Filter Coefficients
  - MCP-SPAU to Bulk System Storage
    - \( N \) Filtered Outputs

- **Computations**
  - \( N + \frac{1}{2}N \log_2 N \) Complex Multiplications

If single cycle data transfer and 2-cycle complex multiply are assumed, the speed ratio between the bulk system storage and the buffer memories is obtained as \( (2 + \log_2 N) / 3 \). When \( N = 1024 \), the speed ratio equals 4. This allows bulk system storage bandwidth to be 4 times slower than the buffer memory speed.

**Local stores**

Local storages will be provided in MCP and SPAU data flows for internal arithmetic operations. Further discussions are deferred until later sections.

**Figure 3—Microprogrammed control processor data flow (MCP)**

**Microprogrammed control processor (MCP) architecture**

The architecture of the MCP is oriented toward its signal processing application as an interface and control processor. The salient features of the MCP required by this application are:

- a. High-speed buffer and channels
- b. Efficient interrupt scheme
- c. Simple external interfaces
- d. Ease of microprogramming

These design goals were achieved by:

- a. Using two independent monolithic storage units as buffer memories and linking them with a wide 64-bit channel interface.
- b. Using only three registers in the data flow and matching the storage cycle time with the internal processing time in order to permit the interrupt overhead operation (save and restore) to be performed in three micro-instructions.
- c. Using a monolithic read-write storage as a microprogram control store. A 64-bit microprogram instruction provides a control format that is 5 to 10 times more powerful than typical 16-bit instructions found in minicomputers and provides the computing power necessary to perform the interface tasks. A read-write control store provides the ease of program modification required to efficiently debug the operational microprograms. It also offers dynamic paging of microprograms through the system storage hierarchy.

The basic data flow is shown in Figure 3. The func-
tional units include:

a. Sequence Unit. The sequence unit is that portion of the machine which controls the sequence of logical operations by determining the order in which control words are addressed from the control store. The sequence unit operations are controlled by the control store word, storage bus condition, data flow conditions, machine status, and channel conditions. The address for each control store word access is assembled by the sequence unit from the above controlling sources in the next address assembly register (NAAR). The save address latch (SAL) holds assembled addresses for storage during interrupt processing. The last address register (LAR) holds the previous cycle control store address register (CSAR) when a machine stop occurs.

b. Buffer Memory Control and Bussing. Each of the basic buffer memories has a word width of 64 bits, used as four 16-bit words in the data flow. The buffer control unit (BCU) serves as an interface between the buffer memory and the various devices that use storage, such as I/O channels, the MCP data flow, and any other direct access devices such as SPAU that may be connected to it. The BCU includes a priority determination unit, an addressing unit, a storage bus, and fetch busses.

c. Data Flow. The data flow includes three 16-bit registers that provide the necessary buffering between the storage bus and the arithmetic and logic unit (ALU). They are destination registers for data fetch and ALU operations. Input selection is under direct control of the control store. All three registers have connections to the storage bus to allow them to be saved following an interrupt. Selection of operands to the ALU is controlled by a control store field. The ALU is a conventional 16-bit parallel adder and logic unit that performs a function on the selected left (L) and right (R) operands. 16-bits represents a 96 db dynamic range. Physical measurements or control signals seldom require more than 16-bit precision. Microprogrammed double-precision can be used for rare exceptions. The ALU output (Z) is latched when the function is completed and held through the late part of the control cycle, when the result is transferred to the destination register. The registers are set late in the control cycle and are held for an ALU operation in the next cycle. The ALU functions include adding, shifting, and logical operations.

d. Local Storage. A 16 word Local Storage is contained in the data flow. The local store provides 16-bit inputs to the ALU. The local store write register has an input from the ALU latch. Local store is addressed from the Local Store Address Register (LSAR). The LSAR has inputs from the X register, the CD field, and the LSAR decrementer. The Local Store can be accessed and written in one MCP cycle. It can be expanded to 32 words. Or, a second local store can be added to achieve the operations of two independent register stacks within one MCP ALU cycle.

e. Literals. Two 16-bit literals are available to the ALU for generation of constants and buffer memory addresses.

f. Interrupt Processing. Interrupt processing consists of saving program status data that would be destroyed in processing the interrupt, and restoring conditions so that the interrupted program may be continued after the interrupt has been processed. The save operation is performed in one micro-instruction:

```
SCXYS(O)
```

This micro-instruction stores the CSAR, X register, Y register, S register, stats, and masks in four consecutive locations in both buffer memories beginning at address 0000. The restore operation requires two micro-instructions:

1. \( A(O) = X, B(1) = Y \)
   This loads the Y register with the value to be restored into S and places the CSAR value to be restored into the X register.
2. \( A(2) = X, B(3) = Y, Y = S, RTN \)
   This restores X, Y, and S registers; RTN forces CSAR to be loaded from prior X register value returning to the next address of the routine that was interrupted and also restores stats and masks.

The restore operation is interruptable. Interrupts of MCP can be generated by I/O channels, program stats and SPAU completion. There are four programmable interrupt priority levels.

Signal processing arithmetic unit (SPAU)

In preprocessing, it is observed that there is no requirement for a hardware multiplier in interface or control functions. However, an extremely high duty cycle multiplier is necessary to satisfy signal processing re-
quirements. Furthermore, a ‘structured’ multiplier will be needed in this case.

In order to accommodate both time domain and frequency domain operations, the SPAU is designed to execute the following basic operation with implicit addressing:

\[ D_i \leftarrow A_i * B_i + C_i \]

where \( i = 1, 2, 3, \ldots 4096 \) and \( A, B, \) and \( C \) are all complex numbers

Although the SPAU provides a basic 16×16 complex multiplier, it is hardware equivalent to four 16×16 real multipliers or one 32×32 fixed point multiplier. Under the MCP microprogram control, the SPAU can function as desired in various configurations when application arises. For special case FFT operations, block floating point format may be assumed with 12-bit mantissa and 4-bit scaling.

The SPAU buffered operations can be interrupted by the MCP when MCP ALU registers require a single multiply operation, i.e., MCP register mode has priority over the normal buffered mode of SPAU. The SPAU buffered operations are initiated by MCP microprograms and SPAU interrupts the MCP on completion.

The SPAU design includes the necessary pipeline to assume an asynchronous operational speed. The basic SPAU operation requires two to four buffer memory cycles dependent upon number of independent buffer memories used in the system. Some special functions are included in the SPAU design such as an address generation and conjugate multiply.

The parallel matrix multiplier logic of SPAU is very straightforward. However, the amount of hardware in terms of logic gate counts is probably twice the MCP ALU/10CU combined. It is almost anti-climactic to state again the importance of the ‘lean’ MCP design in order to keep the ‘fat’ SPAU usefully busy.

AN APPLICATION EXAMPLE—ADAPTIVE SPATIAL PROCESSING

The function flow of the application example is shown in Figure 4. The mathematical computations required to achieve the ‘optimum’ processing are described below.

The sensor input signals \( X_k(t) \) are first transformed into spectral elements in frequency domain that

\[ \{X_k(f_i)\} = FFT\{x_k(i)\} \quad (13) \]

Then the frequency domain inputs are filtered to obtain the desired beam outputs

\[ Y_j(f_i) = \sum_{k=1}^{b} \theta_{kj}(f_i) X_k(f_i) \quad (14) \]

Where \( Y_j(f_i) \) is the jth beam output and \( \theta_{kj}(f_i) \) are optimum filter coefficients including spatial processing in the frequency domain. These filter coefficients are updated through iterative gradient search process to minimize the output noise power. The output power spectrum is then computed

\[ |P_j(f_i)| = \left| Y_j(f_i) \right|^2 \quad (15) \]

Notice the block array forms of data which are efficiently processed by the SPAU. The MCP will control the paging and I/O for the system.

The optimum gradient search algorithm first computes the gradient for minimization as an input-output cross correlation function \( Z_{kj}(f_i) \) that

\[ Z_{kj}(f_i) = Y_j(f_{km}) \exp(j2\pi f_{m} \tau_{jk}) X_k(f_{N-m}) \]

\[ -X_j(f_{N-m}) \quad (16) \]

where \( X_j(f_{N-m}) \) are average beam outputs computed much less frequency. Notice the spatial term \( \exp(j2\pi f_{m} \tau_{jk}) \) in the equation and \( j = \sqrt{-1} \) in the exponent instead of subscripted \( j \). In order to maximize the output noise power change between \( m \)th and \( (m-1) \)th iteration, an iteration step size \( a_j(m) \) is chosen that

\[ a_j(m) = \frac{\sum_{i=1}^{N} Y_j(f_{N-m};m) Q_j(f_i;m)}{\sum_{i=1}^{N} |Q_j(f_i;m)|^2} \quad (17) \]

where

\[ Q_j(f_i;m) = \sum_{k=1}^{b} X_k(f_i) \exp(j2\pi f_{m} \tau_{jk}) Z_{kj}(f_i) \quad (18) \]

Figure 4—Adaptive spatial processing
then the iterative processing is completed by

\[ \theta_{k+1}(f_i; m) = \theta_k(f_i; m-1) - a_j(m)Q_j(f_i; m) \] (19)

and

\[ Y_j(f_i; m) = Y_j(f_i; m-1) - a_j(m)Q_j(f_i; m) \]

In addition to loop controls for the gradient search algorithm, the MCP will handle orthogonal addressing and organize the optimum filter coefficients in such a manner which can be efficiently retrieved from the bulk system storage.

The distributed processing load of this application to MCP and SPAU is normalized to MCP total processing load and tabulated below:

<table>
<thead>
<tr>
<th>Percent</th>
<th>MCP Load 4-cycle</th>
<th>SPAU Load 2-cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>9.7</td>
<td>4.9</td>
</tr>
<tr>
<td>Filter</td>
<td>9.8</td>
<td>69.7</td>
</tr>
<tr>
<td>Iterative Search</td>
<td>86.0</td>
<td>17.0</td>
</tr>
<tr>
<td>Power Spectrum</td>
<td>4.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>96.8</td>
</tr>
<tr>
<td></td>
<td>48.7</td>
<td></td>
</tr>
</tbody>
</table>

It is noted that for the adaptive spatial processing 2-cycle SPAU will not be needed unless an additional MCP unit is included in the system. The 2-buffer memory configuration as indicated in Figure 2 will be applicable to this problem. The load balancing between MCP and SPAU is accomplished by the heavy involvement of MCP microprograms in the various processing loop controls of the gradient search algorithm to compute the optimum filter coefficients. For less demanding MCP control role in other signal processing applications, the 2-cycle SPAU with four buffer memories can handle higher through-put when needed. Additional MCP processing can also be applied to post detection computations which are not described in this example.

CONCLUSION

The Microprogrammed Control Processor and the Signal Processing Arithmetic Unit architecture presented in this paper blends economy, flexibility and performance in one design. It can truly be a general purpose signal processor. If ECL and bi-polar memories are used for implementation, 50 nanoseconds micro-instruction time and 100 nanoseconds complex multiplication speed can be achieved for ground based applications. The hardware size in terms of logic gate counts is approximately one tenth of that of a commercially available general purpose computer with equivalent performance.

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to many people who participated in applying microprogramming to signal processing problems. In particular, he would like to thank R. G. Baron and S. G. Francisco of IBM for their pioneer work in seismic and acoustic processing, R. A. Rieth of Teledyne-Brown and J. W. Fisher of Bell Telephone Laboratories for their active involvement in radar processing and G. L. Kratz of IBM for general microprogrammed processor architecture work during the past five years.

REFERENCES

1 R G BARON W VANDERKULK S D LORENZ
   A LASA signal processing system
   IBM Federal Systems Division internal report Bethesda
   Maryland November 1965

2 Large aperture seismic array signal processing study
   IBM Final Report—Prepared for the Advance Research
   Project Agency Washington D C Contract Number SD-196
   July 15 1965

3 G D HORNBUCKLE E I ANCONA
   The LX-1 microprocessor and its application to real time
   signal processing
   IEEE Transactions on Computers August 1970 Vol C-19
   Number 8

4 Y S WU G L KRATZ
   Microprogrammed interface processor (MIP) and its
   application to phased array radar
   Tech Note Spring Joint Computer Conference May 1971

5 J E SHORE F A POLKINGHORN JR
   A fast, flexible, highly parallel associative processor
   Naval Research Laboratory Report Number 6961
   Washington D C November 28 1969

6 P L GARDNER
   Functional memory and its microprogramming implications
   IEEE Transactions on Computers July 1971 Vol C-20
   Number 7

7 J A GITHENS
   An associative, highly-parallel computer for radar data
   processing
   In L C Hobbs et al eds “Parallel Processing Systems
   Technologies and Applications” Spartan 1970 pp 71-87

8 R S ENTNER
   The advanced avionic digital computer
   Ibid pp 203-214

9 IBM 2938 array processor
   IBM Systems Reference Library 1967

10 J E SHORE
   Second thoughts on parallel processing
   To be published IEEE International Convention March
   1972
11 B WIDROW  P MANTEY  L GRIFFITHS
B GOODE
Adaptive antenna systems
IEEE Proceedings Vol 55 Number 12 December 1967
12 Y W LEE
Statistical theory of communication
Wiley N Y 1960
13 B GOLD  C M RADER
Digital processing of signals
McGraw-Hill N Y 1969
14 F K KUO  J F KAISER Eds
System analysis by digital computer
Wiley N Y 1967
15 G D DANIELSON  C LANCZOS
Some improvements in practical Fourier analysis and their
application to X-ray scattering from liquids
J Franklin Inst Vol 233 pp 365-380 and 435-452 April
1942
16 J W COOLEY  J W TUKEY
An algorithm for the machine calculation of complex Fourier
series
Math of Comput Vol 19 pp 297-301 April 1965
17 W T COCHRAN et al
What is the fast Fourier transform?
IEEE Proceedings Vol 55 Number 10 October 1967
18 R W COOK  M J FLYNN
System design of a dynamic microprocessor
IEEE Transactions on Computers Vol C-19 Number 3
March 1970
19 W R SMITH
Simulation of AADC system operation with an E2-B
program workload
Naval Research Laboratory Report Number 7259
Washington D C April 22 1971
20 S S HUSSON
Microprogramming principles and practices