

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Research Laboratory Washington, D.C. 20390		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE DESIGN CONSIDERATIONS OF A PROGRAMMABLE PREDETECTION DIGITAL SIGNAL PROCESSOR FOR RADAR APPLICATIONS			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) An interim report on the problem; work on this and other phases is continuing.			
5. AUTHOR(S) (First name, middle initial, last name) Barry P. Shay			
6. REPORT DATE December 13, 1972		7a. TOTAL NO. OF PAGES 56	7b. NO. OF REFS 7
8a. CONTRACT OR GRANT NO. NRL Problem B02-10		9a. ORIGINATOR'S REPORT NUMBER(S) NRL Report 7455	
b. PROJECT NO. Project No. XF 53-241-003 Task K032		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.			
d.			
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Naval Electronics Systems Command Washington, D.C. 20360	
13. ABSTRACT Signal processing tasks normally encountered in a radar system, such as pulse compression and moving target indication, are defined in terms of recursive and nonrecursive digital filtering and the discrete Fourier transform. Analysis of these algorithms reveals that the general second-order recursive filter and the Fast Fourier Transform butterfly are kernels which should be efficiently executed by a Radar Arithmetic Processing Element (RAPE). This element would be the arithmetic unit of an envisioned programmable signal processor. With associated memory and a small microprogrammed control unit, this element could be an independent, general-purpose signal processor. As such, it may represent an "off-the-shelf" module capable of replacing a wide range of existing special-purpose hardware. Computation schemata are used to display potential parallelism within the computational kernels. By exploiting parallelism in hardware and by using state-of-the-art components, the output of a second-order digital filter may be computed in approximately 400 nsec. This is sufficient as an MTI filter for a 2000-range-gate system. Similarly, an FFT butterfly may be computed in approximately 150 to 200 nsec. This processing rate is sufficient to compress pulses of widths from 100 to 0.4 μ sec (200 ft resolution) within a 1-nsec pulse repetition period. Some consideration is given to incorporating the RAPE into a generalized signal processing system containing parallelism at many levels. This system would contain many RAPEs, each individually controlled, operating concurrently.			

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Digital signal processing Digital filtering Fast Fourier transform Parallel processing Programmable signal processor Program schemata Radar signal processing						

CONTENTS

Abstract	ii
Problem Status	ii
Authorization	ii
I. INTRODUCTION	1
II. COMPUTATIONAL IMPLICATIONS	2
III. DESIGN IMPLICATIONS	9
Timing Considerations	10
System Configuration	23
IV. CONCLUSIONS	25
REFERENCES	27
BIBLIOGRAPHY	28
APPENDIX A — Radar Signal Processing	30
APPENDIX B — General Digital Signal Processing	37
APPENDIX C — Recursive Filter Design	41
APPENDIX D — Fast Fourier Transform — Spectrum Generation	49

ABSTRACT

Signal processing tasks normally encountered in a radar system, such as pulse compression and moving target indication, are defined in terms of recursive and nonrecursive digital filtering and the discrete Fourier transform. Analysis of these algorithms reveals that the general second-order recursive filter and the Fast Fourier Transform butterfly are kernels which should be efficiently executed by a Radar Arithmetic Processing Element (RAPE). This element would be the arithmetic unit of an envisioned programmable signal processor. With associated memory and a small micro-programmed control unit, this element could be an independent, general-purpose signal processor. As such, it may represent an "off-the-shelf" module capable of replacing a wide range of existing special-purpose hardware.

Computation schemata are used to display potential parallelism within the computational kernels. By exploiting parallelism in hardware and by using state-of-the-art components, the output of a second-order digital filter may be computed in approximately 400 nsec. This is sufficient as an MTI filter for a 2000-range-gate system. Similarly, an FFT butterfly may be computed in approximately 150 to 200 nsec. This processing rate is sufficient to compress pulses of widths from 100 to 0.4 μ sec (200 ft resolution) within a 1-msec pulse repetition period.

Some consideration is given to incorporating the RAPE into a generalized signal processing system containing parallelism at many levels. This system would contain many RAPEs, each individually controlled, operating concurrently.

PROBLEM STATUS

This is an interim report; work on this and other phases of the problem is continuing.

AUTHORIZATION

NRL Problem B02-10
Project No. XF 53-241-003 Task K032

Manuscript submitted June 23, 1972.

DESIGN CONSIDERATIONS OF A PROGRAMMABLE PREDETECTION DIGITAL SIGNAL PROCESSOR FOR RADAR APPLICATIONS

I. INTRODUCTION

Major signal processing tasks in a contemporary tactical radar system (see Appendix A) include waveform generation, moving target indication (MTI), pulse compression (PC), and video integration (1). Early radar systems executed these functions using special purpose analog devices. As digital technology progressed, it became apparent that some of these functions could be performed by sampling the analog signals and then performing the transformations on the sampled digital data (Appendix B) and that system advantages accrued when these functions were performed in the digital domain. The large bandwidths characterizing radar signal returns necessitate high sampling rates which, in turn, require high computation speeds. General purpose computer technology could not support such speed requirements and so special purpose hardwired digital devices were used despite the unfavorable effects on development time and risk and on life cycle costs resulting from the design of a new processor for each new radar. The development of large-scale integrated (LSI) circuit technology, along with the evolution of ultrahigh-speed memories, allows one to consider the possibilities of performing these tasks via a programmable computing structure optimized for signal processing.

A complete signal processing task can be thought of as composed of a number of levels, each level containing "resources" which may be called upon by the preceding level. In this layered representation, shown in Fig. 1, each element of a particular level can be considered a resource of the next higher level. Thus any particular signal processing task may be defined in a nested manner. As an example, the statement

$$[SP [Radar [MTI [Recursive Filter [2-pole, 2-zero]]]]]$$

means that the signal processing task under consideration is a radar function, specifically moving target indication (MTI). The MTI is implemented in the time domain via recursive filtering (Appendix C) with the 2-pole, 2-zero filter as the basic element. If the system contains elements from which the second-order filter can be composed, then further nesting is possible. Real-time computing constraints dictate which level should be available as a hardwired device. Analysis of various radar predetection algorithms reveals that the second-order recursive filter and the FFT butterfly (Appendix D) should be directly available as resources to higher level elements. Additional external control of the manner in which these resources are used by elements of higher levels is required.

The apparent similarity between the sequencing and structure of the second-order filter and the FFT butterfly suggests "merging" the two kernels into a single arithmetic element. When this element is enhanced with data memories and an internal control unit, possibly microprogrammed, it may represent an "off-the-shelf" module capable of replacing a wide range of existing special-purpose hardware. It should be noted, however, that external control or supervision will be required to incorporate this module into the total system.

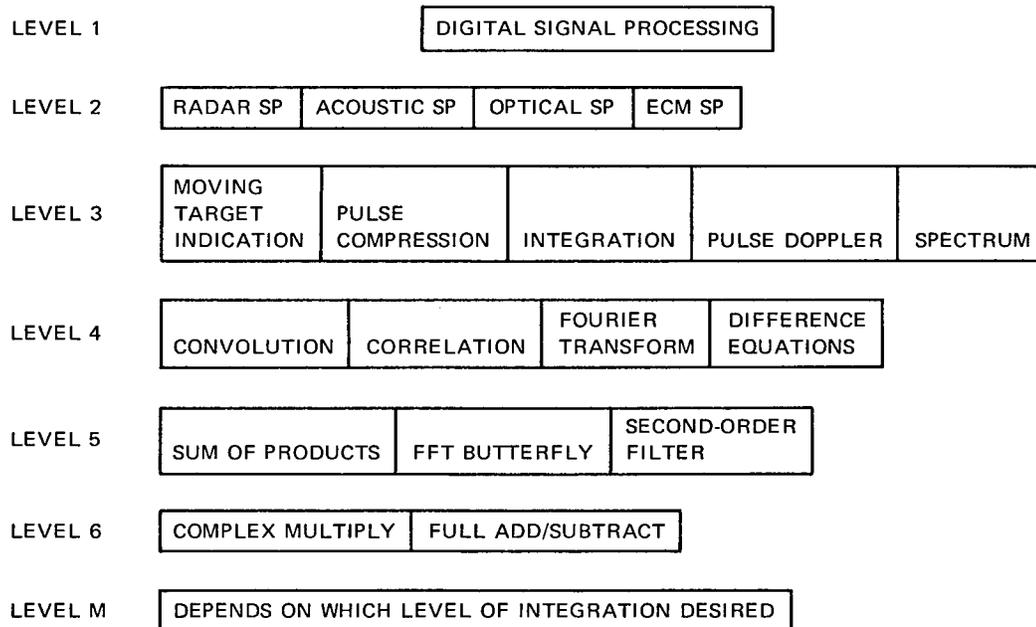


Fig. 1 — Signal processing structure

Work is currently under way in the design and fabrication of a prototype of an efficient Radar Arithmetic Processing Element (RAPE) with the above properties. The design of an external supervisor is also under way. These will be addressed in detail in subsequent reports.

II. COMPUTATIONAL IMPLICATIONS

A programmable computing structure for real-time signal processing must be extremely fast. To achieve this speed the structure should be reconfigurable to match the algorithm to be executed. In addition, the basic arithmetic processing element of the structure must be capable of executing, very efficiently, the computing kernel of the algorithm. In other words, the processing element should, if at all possible, look like a hardwired version of the kernel. The algorithm could then be realized (executed) by iterating the kernel, either by hardware or by software, to match the algorithm. To maximize speed of computation, all the parallelism inherent in the algorithm and in the kernel should be apparent in the structure. This is an important point, for there are many levels of parallelism within digital signal processing algorithms and not taking advantage of it simply increases the execution time. For example, the basic 2-pole, 2-zero filter element, from which all filters may be realized (Appendix B), allows four multiplications and then two additions to be performed concurrently. The basic FFT butterfly also allows concurrent arithmetic operations. At a higher level, sophisticated filters may be realized using parallel and cascade forms. These lend themselves either to parallel computations directly or to pipelining techniques for cascaded filters. Similarly, the FFT computation contains enough parallelism to be executed in the time it takes to compute one butterfly, if the hardware is available (2). In summary: the parallelism is there; the problem is to exploit it!

One of the reasons the capability to perform several arithmetic operations concurrently within a computing structure has not been realized is that there has been no systematic means for identifying parallelism within computational algorithms. Indeed, the range of algorithms normally encountered makes this task no easy matter. Consequently, almost all general-purpose computers have been built to execute algorithms in a single sequence determined entirely by the program. The single-sequence execution scheme not only obviates possible speed increases due to parallelism, but also precludes more efficient code generation which may be obtainable by performing the same computation in another order.

Recently a good deal of effort has been directed toward the representation and modeling of algorithms in their "most parallel" form (3-5). While much of this work is still in the embryonic stage, a few models have been devised which can be directly applied to existing algorithms. One of these models, called a computation or program schema, is a suitable vehicle for representing signal processing algorithms in their most parallel form. Once the algorithms are represented in this form, computing structures may more easily be devised which exploit the exhibited parallelism. Signal processing is an ideal vehicle upon which to use these techniques since the workload contains a small class of highly structured algorithms which lend themselves very well to decomposition and parallel processing.

While each level in Fig. 1 may be represented via schema, only the second-order recursive filter and the FFT butterfly will be so represented here. These models display the data transformations within the kernel, along with a control structure, or precedence graph, which orders the sequence of data transformations. These representations exhibit parallelism at an arithmetic level and will be used to indicate possible realizations of an arithmetic signal processing element.

The concept of computation or program schemata to represent algorithms and structures is new enough to warrant a brief description at this time. (For a more detailed explanation, the reader is advised to consult Ref. 3.)

Every digital system, whether programmable or not, is comprised of a set of storage cells and operators. Storage cells include such things as memory and arithmetic registers. Operators include the logic blocks capable of performing transformations on information contained in storage cells. The sequencing of such transformations is determined by the control structure of the system and additional information contained in storage cells (e.g., the program). Within this framework a simplified computation schema will now be defined.

A *computation schema* is a finite set M of storage cells, a finite set of operators A , a directed data flow graph, and a directed precedence graph. The nodes of the data flow graph represent the operators and storage cells. The operator nodes transform information contained in its incident nodes and place the results in the nodes to which emanating branches are directly connected. The nodes of the precedence graph represent the instance of occurrence of the operation associated with that node. An *execution sequence* of a computation schema is a sequence of operator occurrences in an order consistent with the precedence graph. The values associated with the storage cells depends upon which operation the operator nodes of the schema represent in addition to the execution sequence. Hence, an *interpretation* of a computation schema is defined to be an association of the particular transformation (e.g., add, multiply, shift left, etc.) with the operator nodes of the schema.

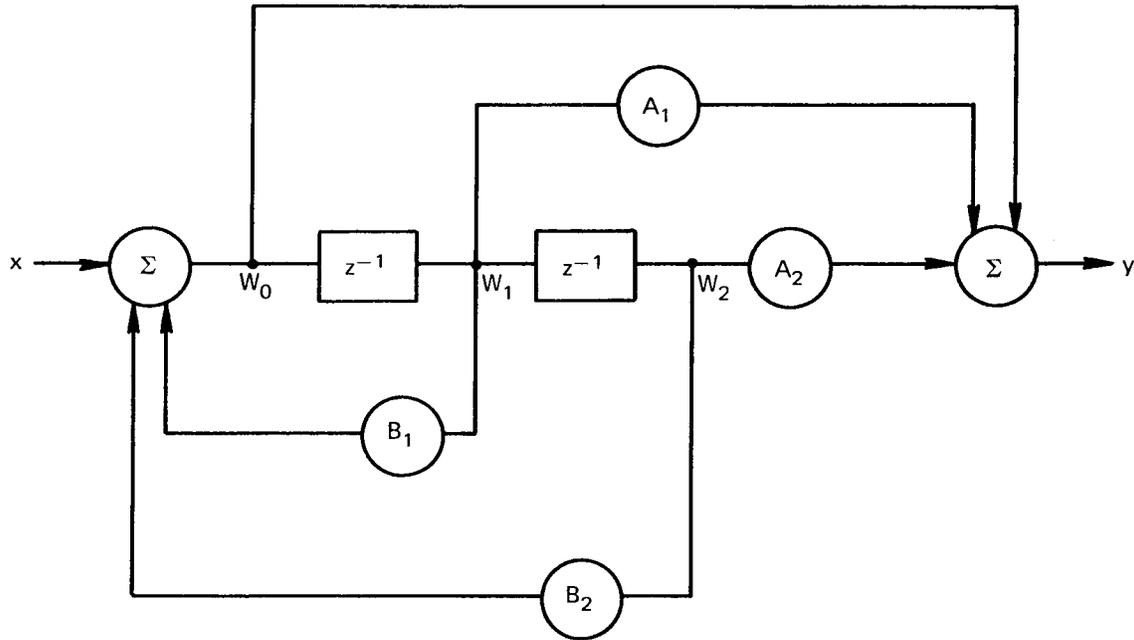


Fig. 2 — Second-order filter

Second-Order Filter — With this introduction, we represent the computation required for the general second-order filter and the FFT butterfly in the form of a computation schema. Consider the second-order filter represented in Fig. 2.

The output y at any instant k can be computed in a variety of ways. For our example, we will use a 2-step computation, defined in terms of the labels in Fig. 2

$$W_0 = X - B_1 W_1 - B_2 W_2 \quad (1)$$

$$y = W_0 + A_1 W_1 + A_2 W_2. \quad (2)$$

Assuming a unity sampling interval (i.e., $y^{-1} = \exp(-j\omega\tau)$, $\tau = 1$), a new output is computed after each transfer of the contents stored in the delay elements. According to our definition, we construct a computation schema for the filter in terms of the actual storage cells and operators required:

$$\text{Memory cells } M = \{A_1, A_2, B_1, B_2, W_0, W_1, W_2, p, q, r, s, x, y\}$$

$$\text{Operators } A = \{X_1, X_2, X_3, X_4, +_1, +_2, T_1, T_2\}.$$

It should be noted that the names of cells which store parameters (coefficients) will be designated by the parameters themselves. For example, A_1 will be the value of the parameter stored in cell A_1 . Memory cells W_1 and W_2 represent the cells corresponding to the delay elements; x and y represent the primary input and output cell of the filter, respectively, and could be thought of as source and sink for input and output samples. Intermediate storage cells are represented by W_0, p, q, r, s , and t .

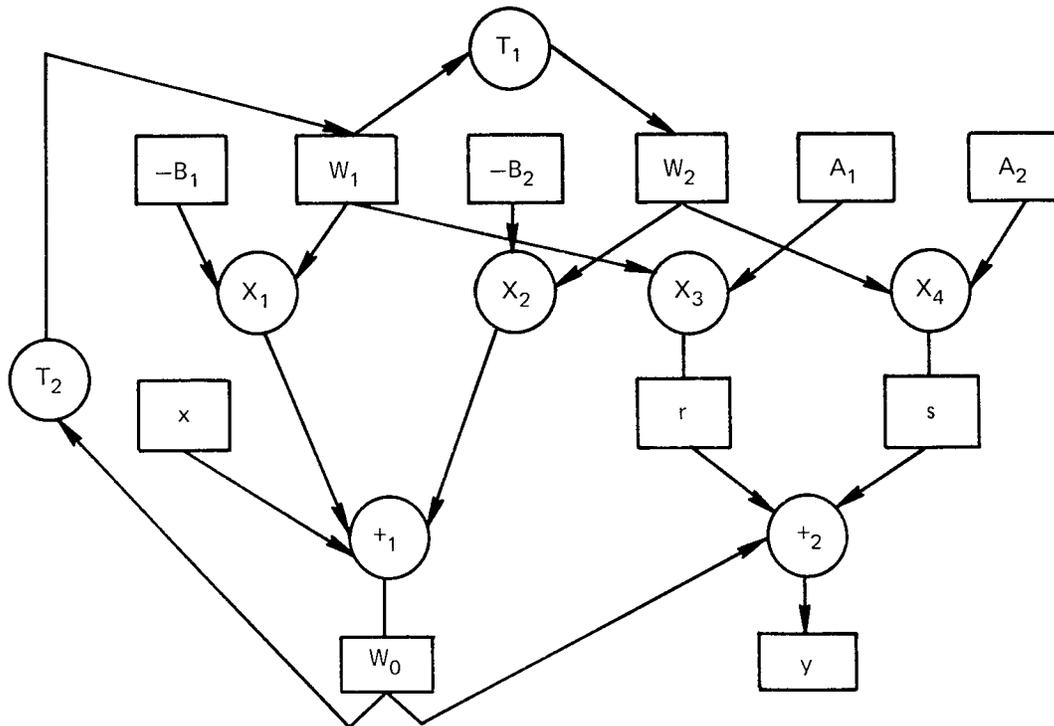
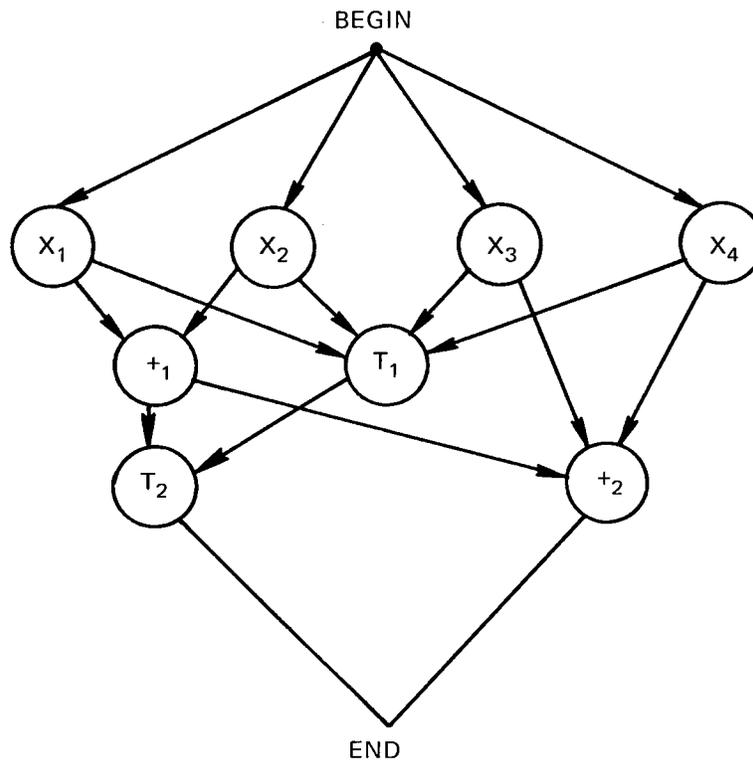


Fig. 3 — Data flow graph

The data flow graph representing the computation defined by Eqs. (1) and (2) is shown in Fig. 3. Intermediate cells are used to store results between dependent operators in order to avoid conflicts. Note that ordering of the two transfer operators is necessary to avoid destroying information before it is used.

The precedence graph in Fig. 4 represents all allowable sequences of execution consistent with the data flow graph. The partial ordering relations indicate dependent operations. Operations for which no ordering relation exists can be performed in parallel, assuming that hardware facilities are available. For example, one particular execution sequence is (X_1, X_2, X_3, X_4) performed concurrently, followed by (A_1, T_1) concurrently, and finally (A_2, T_2) concurrently. This sequence can be completed in three execution cycles. Another possible execution sequence is $(X_1), (X_2), (X_3), (X_4), (+_1), (T_1), (+_2), (T_2)$ all performed sequentially. This computation requires eight cycles to complete. Many other legitimate execution sequences are made possible by exploiting various degrees of concurrency. It is obvious that the more parallelism exploited, the faster the total computation time.

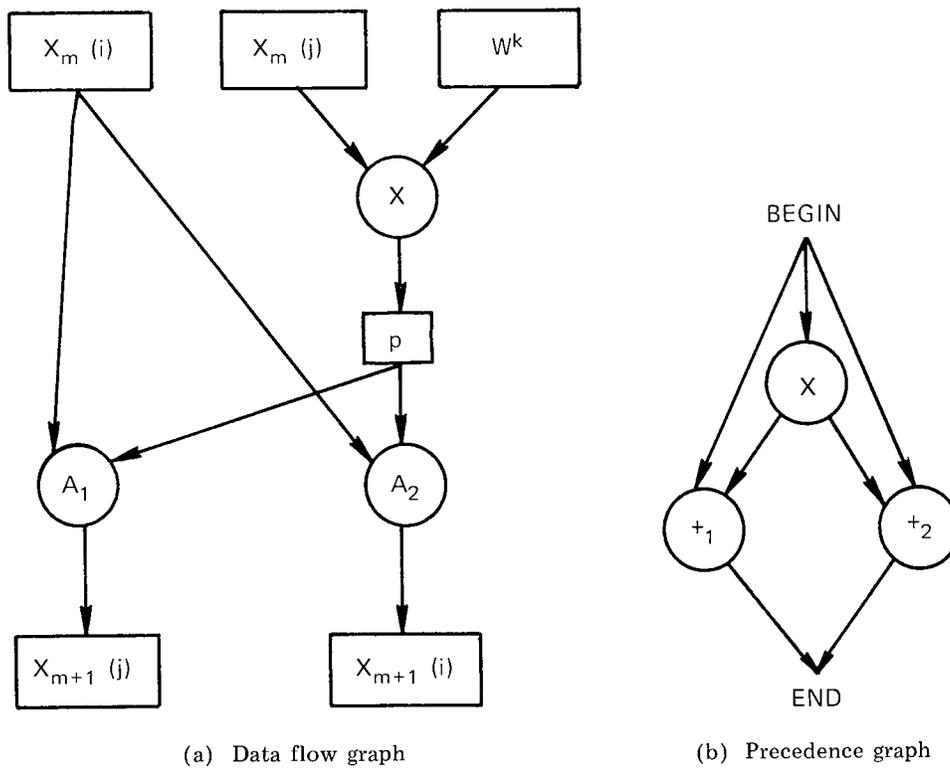
FFT Butterfly — The FFT butterfly, presented in Appendix D, may be computed in a variety of ways. The schemata representation of the computation displays all possible execution sequences, and so may be used to arrive at an optimum configuration. In addition, the segmentation of the computation into its real and imaginary parts is apparent. Figure 5 represents the FFT butterfly in its complex form; i.e., all operations and data involve complex quantities. Each of the operators X , $+$ ₁, and $+$ ₂ can be decomposed into operators whose domain and range consists of real quantities. When this is done, the configuration in Fig. 6 results.



INTERPRETATION:

$X_1: -B_1 \times W_1 \rightarrow p$	$+_1: p + q + x \rightarrow W_0$
$X_2: -B_2 \times W_2 \rightarrow q$	$+_2: r + s + W_0 \rightarrow y$
$X_3: W_1 \times A_1 \rightarrow r$	$T_1: W_1 \rightarrow W_2$
$X_4: W_2 \times A_2 \rightarrow s$	$T_2: W_0 \rightarrow W_1$

Fig. 4 — Precedence graph



$$M: X_m(j) \cdot W^k \rightarrow p$$

$$A_1: X_m(i) - p \rightarrow X_{m+1}(j)$$

$$A_2: X_m(i) + p \rightarrow X_{m+1}(i)$$

(c) Interpretation

Fig. 5 — FFT butterfly, complex

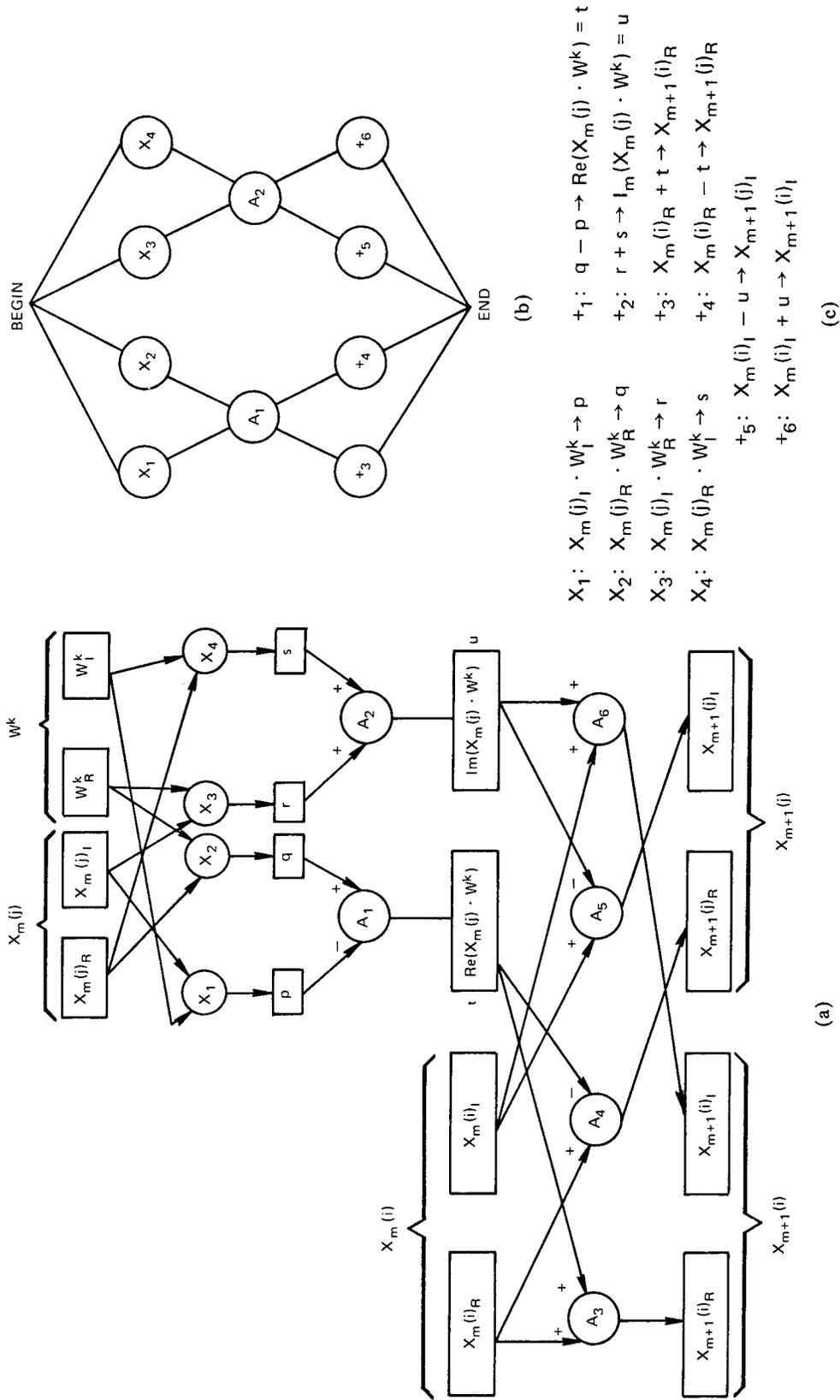


Fig. 6 — FFT butterfly, real-imaginary

The precedence graph in Fig. 6b defines those operations which may be performed concurrently as well as those which must be computed in sequence. For example, the execution sequence composed of (X_1, X_2, X_3, X_4) followed by $(+_1, +_2)$ and then $(+_3, +_4, +_5, +_6)$ allows the complete FFT butterfly to be computed in a minimum of three execution cycles. On the other hand, nine execution cycles are required for a completely sequential computation of the same butterfly. Obviously, there are execution sequences which fall somewhere between these two extremes.

Two other interesting observations become apparent from a consideration of the schema of the FFT butterfly. The first is that no memory cell in the data flow graph is reused by subsequent operators. This implies that the computation may be executed in a "pipelined" fashion. Accordingly, the throughput rate is determined solely by the slowest operation in the pipe; i.e., the multiply. This, of course, assumes that data are available when required by the operators. In Section III, a configuration based on the apparent parallel pipeline characteristics of the FFT butterfly will be specified. The throughput rate of this configuration is, indeed, equivalent to the cycle time of the multiplier. In addition, if each multiplier is configured by more than one sequential stage, then the multiplier itself may be considered a pipeline. In this situation, the effective throughput rate will be increased to that of the slowest stage in the pipe.

The second observation concerns the apparent left-right symmetry of the schema. This fact permits the effective throughput rate to be doubled when real data are used. In such a case, cells which normally hold imaginary data will now be supplied with real data. Separating the data at the completion of the butterfly results in four real data points. This idea will not be pursued in this study, although it will be discussed in future reports.

Each of the combined data flow and precedence graphs with which we have represented both the second-order filter and the FFT butterfly may be considered a hardwired algorithm. As such, they may be synthesized directly from the representation using registers, multipliers, and adders for the storage cells and operators. The control sequence could be generated either synchronously or asynchronously according to execution sequences consistent with the precedence graph. For maximum speed of operation, the execution sequence chosen should be one containing maximum parallelism. This results, however, in a significant increase of hardware. While a hardware implementation as just described may be useful as a second-order filter, it does not possess the necessary storage required to complete a full FFT of more than two points. By adding memory for working storage, we can develop similar schema which do possess the capabilities of performing useful computations. This will be discussed in the next section.

III. DESIGN IMPLICATIONS

In the previous section, the data flow and sequencing of the second-order filter and the FFT butterfly were presented purely in terms of registers and operators. A hardwired digital system could be built for each algorithm based on the schemata. However, our main goal is to develop a programmable system with more flexibility than a completely hardwired device. This implies that we incorporate memory for data storage and a control unit for decoding instructions and sequencing the arithmetic element. In this section, we incorporate data memories into the schema of the filter and butterfly and show that, by using state-of-the-art components judiciously, real-time operation is possible. At this time, we will not discuss in detail the elements of the control unit. However, some general comments with respect to controlling the arithmetic element will be made in the concluding section.

Timing Considerations

To arrive at approximate timing and sampling requirements, we propose a simplified radar system in terms of its range resolution and unambiguous range. We assume that the radar system performs the functions of MTI, pulse compression, and video integration on a variety of waveforms. The result of this processing will be a detection decision. We assume that the MTI filter is implemented as a digital filter in which the notches and pass-bands are determined by the coefficients. Two filters are considered: one a first-order, single-pulse canceler, and the other a general second-order filter. The pulse-compression filter is realized in the frequency domain by FFT's. The data aperture is chosen so that an aperiodic convolution results. The digital integrator is realized as a simple first-order recursive digital filter similar to the single-pulse MTI filter. While no specific structure is assumed, the proposed arithmetic unit contains the hardware necessary to realize the second-order filter and FFT butterfly operations. A specific design satisfying this requirement has been specified by the author and will be presented in a forthcoming report.

We generate the sampling requirements based upon a hypothetical radar system with an unambiguous range of about 85 mi and range resolution such that two targets separated by 200 ft or more will be distinguishable. Thus, the basic uncompressed pulse width required is 0.4 μ sec, with a pulse repetition period (PRP) of 1.024 msec. We divide the basic pulse repetition period into range gates corresponding to the 200-ft (0.4- μ sec) resolution. Thus, we have 2560 range gates spanned by the PRP. We assume that a single sample represents the return for a particular range gate. Hence the sampling rate is 2560 samples per 1024 μ sec or 2.5×10^6 samples per second (2.5 MHz). In addition to the uncoded pulse, we have available longer coded pulses to cover ranges up to 85 mi. By compressing these pulses to 0.4 μ sec after their return, the requirements for 200-ft resolution and maximum signal-to-noise ratio are met. Since we are concerned with digital processing, we assume coded pulses of widths which can be represented by 2^v samples (v an integer). We also assume an arbitrary coding scheme such that the time-bandwidth product of the pulse allows compression to 0.4 μ sec, i.e., the resolution of the radar. Based upon these waveforms, we can compute the required sampling rates. This information is contained in Table 1.

Table 1
Sampling Rates

Uncoded Pulse Width (μ sec)	Coded Pulse Width (μ sec)	Bandwidth (MHz)	Time \times Bandwidth (Pulse Compression Ratio)	Required Sampling Rate (MHz)	
				Single Channel	I&Q*
0.4	25.6	2.5	64	5	2.5
0.4	51.2	2.5	128	5	2.5
0.4	102.4	2.5	256	5	2.5
0.4	204.8	2.5	512	5	2.5
0.4	409.6	2.5	1024	5	2.5

* I = In-phase channel

Q = Quadrature channel

If we assume a quadrature system comprised of two identical channels, we can sample each component at half the rate (6). This is termed *bandpass sampling*. In either case, the total number of samples in a given time, with the same bandwidth, will be the same. The last column of Table 1 lists the sampling rates for both the in-phase (I) and quadrature (Q) channels (i.e., 2.5 MHz). With this sampling rate, the time-bandwidth product represents the number of samples per coded pulse since, by our original assumption, we are representing each 0.4- μ sec range gate with a single sample. By using a quadrature system, we can therefore process each specified waveform at a uniform sampling rate.

Moving-Target Indication — Consider the single-delay, feedback MTI filter shown in Figs. 7 and 8.

M_1 and M_2 represent separate and independent memories which feed the multiplier. M_1 contains the feedback factor and M_2 stores samples for each range gate (one word per gate) for a time equal to the pulse repetition period. X represents the input from the A/D converter (I or Q) and could be a buffer memory. Each word of M_2 holds the return for a particular range gate so that a particular address is always associated with the same range gate. We assume that all words are 8 bits and that either rounding or truncation occurs where appropriate. In a worst-case situation, all range gates will be interrogated within a PRP. Hence, according to Table 1, the throughput rate of the filter must correspond to the input sample rate of 2.5×10^6 samples/sec or equivalently 400 nsec/sample. The complete MTI operation, as deduced from Fig. 2, requires two memory cycles (M_1 and M_2 independent), a multiplication, and an addition. With state-of-the-art components as detailed in Table 2, the single-delay recursive MTI can be computed in approximately 305 msec. Thus, it is possible to maintain the required throughput rate for the single-delay MTI filter. By including an additional memory to store feedback terms for use during the next PRP, the computation may be pipelined, thereby reducing the effective completion time to about 150 nsec. This technique will be described by the author in a later report.

Let us now consider the general second-order filter, discussed in Appendix B, as a double-delay MTI filter. Consider Figs. 9 and 10. M_1 and M_2 again represent independent semiconductor memories, each with a 75-nsec access time and a 150-nsec cycle time. They each contain 2560 words, one word per range gate. B_1 , B_2 , A_1 , and A_2 represent storage cells for the coefficients and could be small separate read-only memories (ROMs). W_1 represents an intermediate register. Again all words are assumed to be 8 bits long and the associated multiply and add times are those in Table 2. X and Y represent input and output cells, respectively. For batch processing, these could be independent memory buffers. It is now possible to compute the time of a single pass through the second-order filter. The timing diagrams for both filters are shown in Fig. 11 and we see that a complete pass through the second-order filter takes approximately 325 nsec. This, again, falls within the throughput requirements of the hypothesized system. As was mentioned for the single-delay canceler, reuse of memory M_1 during the same PRP may be eliminated by incorporating an additional memory to store feedback terms for use during the next PRP. This allows the filter to be pipelined during each PRP, thereby almost doubling the completion rate.

It is also possible to accommodate higher PRF's by initiating the memory read and write cycles earlier. This is possible because of the high memory bandwidth and the equivalent cycle time of the multipliers. Varying PRF's can be accommodated easily by altering the basic clock rate; thus, a staggered MTI for eliminating blind speeds is easily realized.

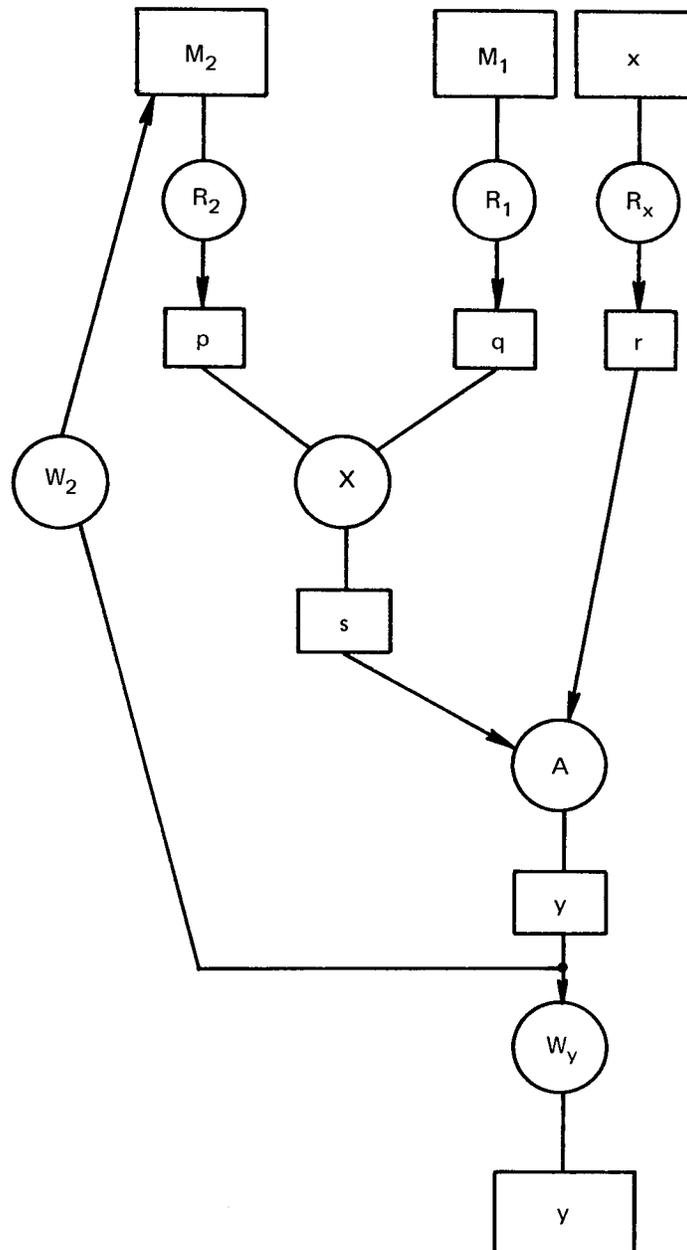


Fig. 7 — Data flow graph

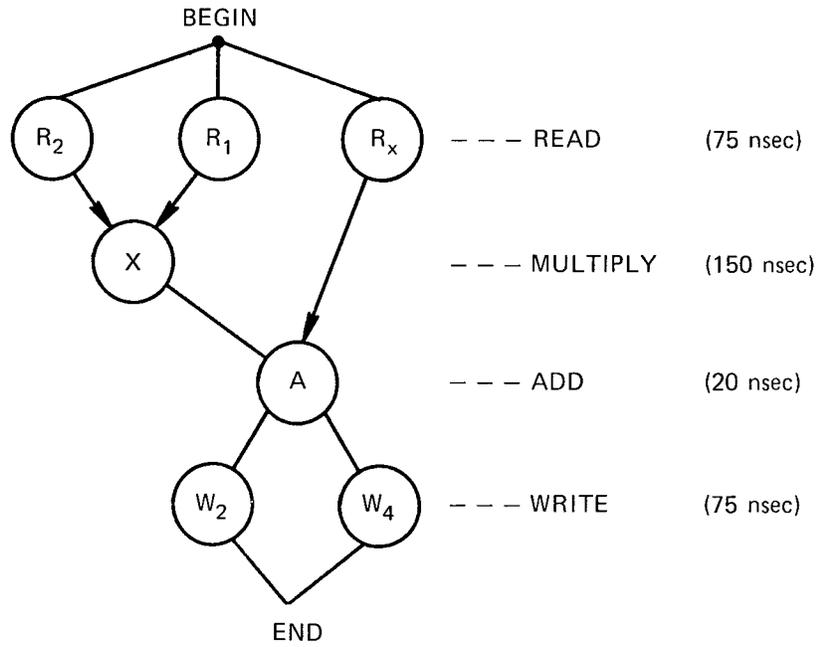


Fig. 8 — Precedence graph

Table 2
Component Operating Speeds

Operation	Time (nsec)
Memory Access	75
8 × 8-Bit Multiply	150
8 × 8-Bit Add	20
3-Input Add	40

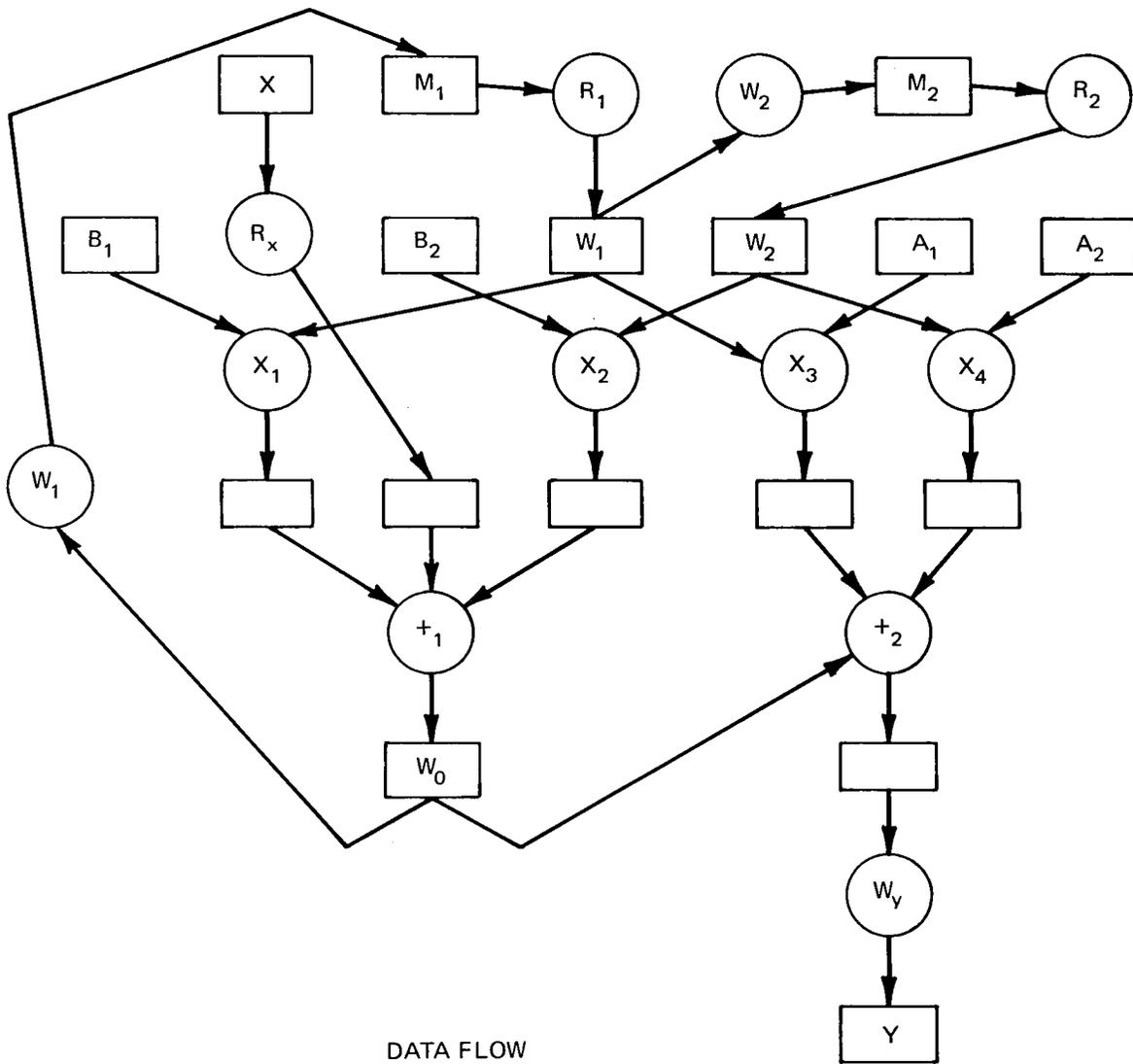


Fig. 9 — Second-order filter

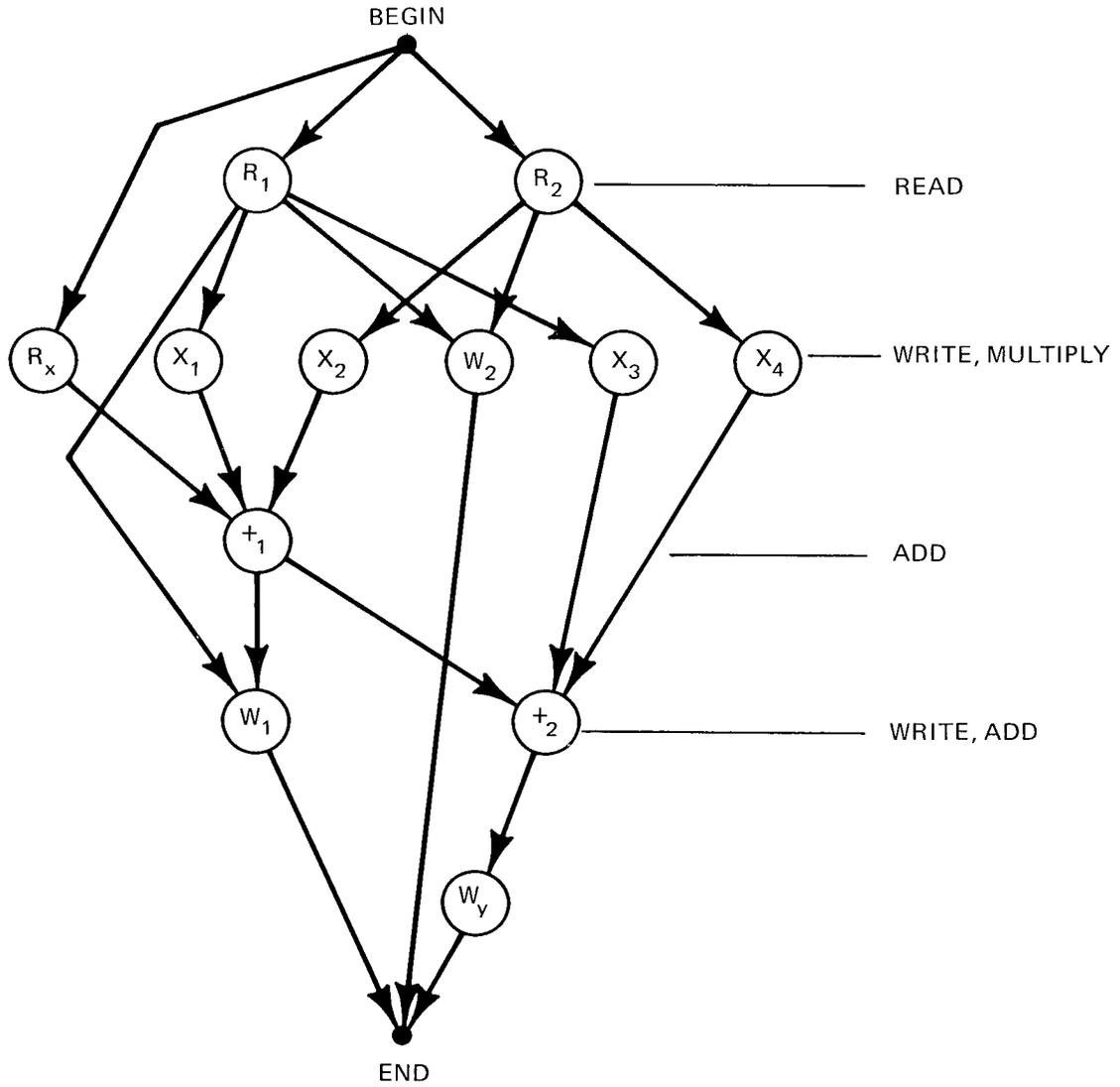
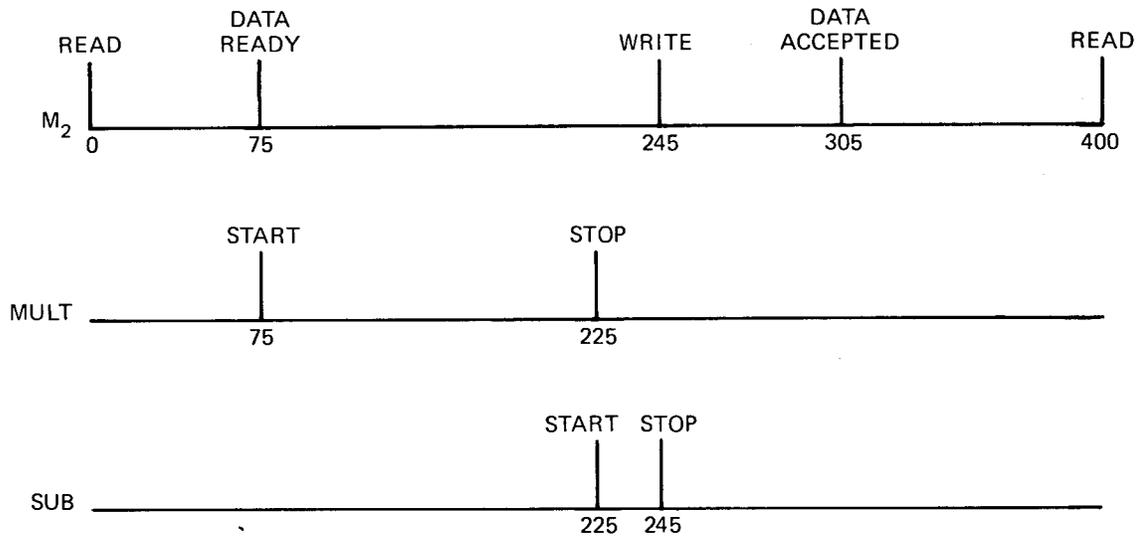
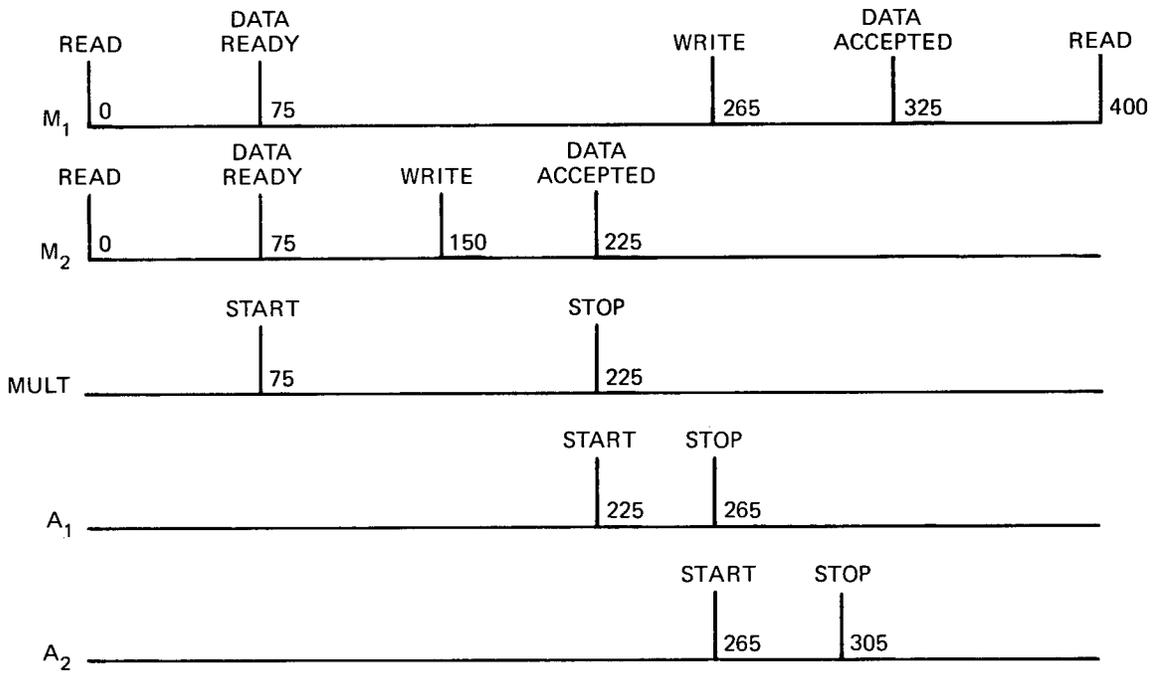


Fig. 10 - Precedence graph



(a) First order



(b) Second order

Fig. 11 — Timing diagram

While the second-order filter is used here as an MTI filter, it can serve equally well as the basic element of any digital filter (see Appendix C). Both cascade and parallel realizations are possible by just cycling through the unit some fixed number of times. For parallel realizations, results for each pass would be stored and added at the completion of the final pass or accumulated at each pass. For cascade realizations, intermediate results constitute the inputs for the next pass. While many filters can be represented by combinations of identical stages, it may be necessary to alter the coefficients at each pass. This configuration allows for such possibilities. These techniques can also be used to implement multiple-delay cancelers.

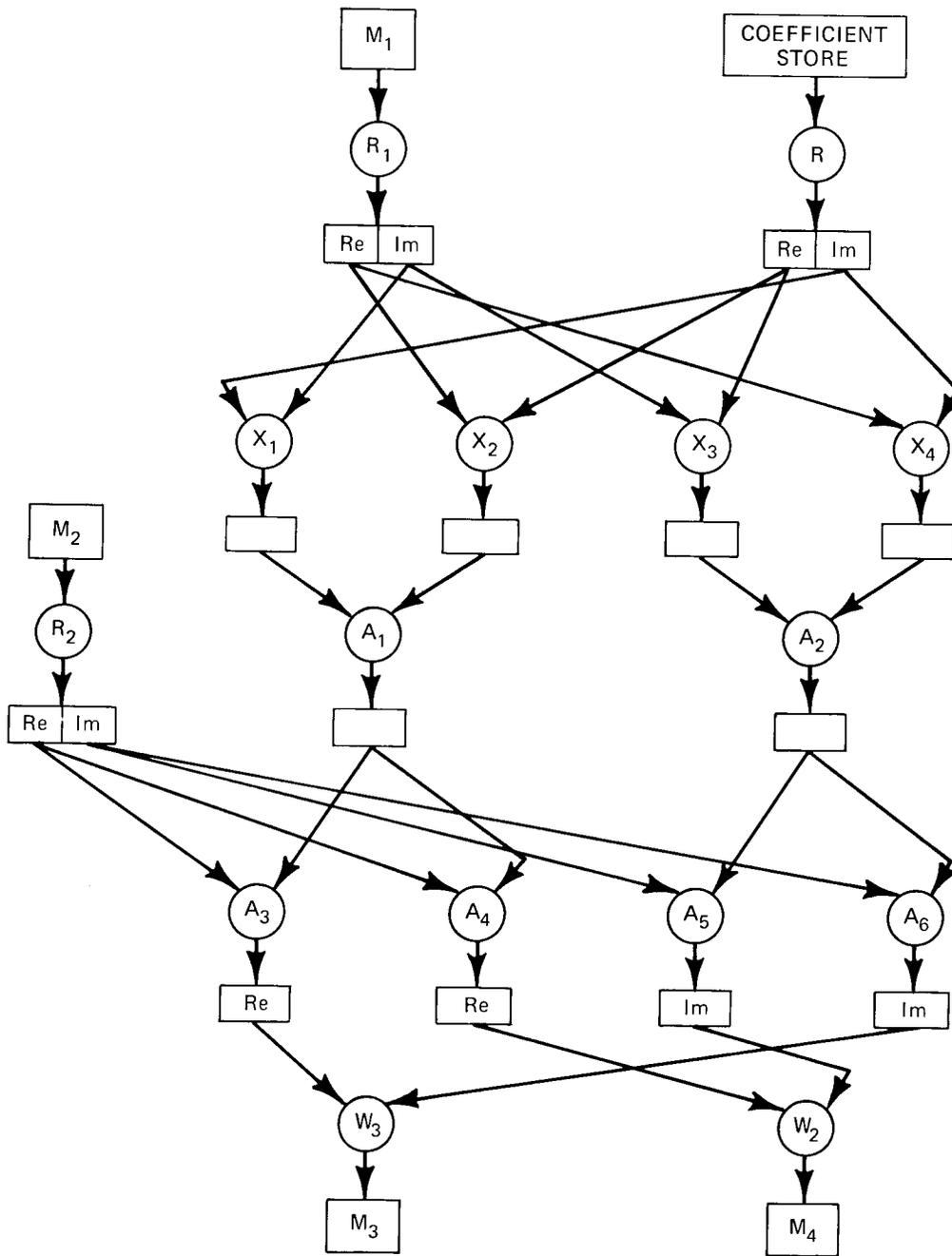
As far as MTI is concerned, it can be shown (7) that the second-order filter produces about a 10-dB improvement over the first-order filter, whereas a third-order MTI filter adds only about 1 dB further improvement. In addition, as shown above, the second-order filter requires only 20 nsec additional computing time over the first-order filter. However, the hardware increase is significant; twice as much memory and about three times as many gates.

Fast Fourier Transform — The FFT is the key to performing pulse compression in the frequency domain. We therefore propose a configuration for computing the FFT, and from this configuration we determine the implications of performing pulse compression in real time.

For an N -point transform, there are $\log_2 N$ arrays to be calculated, with N points in each array. Each new point in the next array is the result of a complex multiplication and a complex addition. A complex multiply requires four real multiplications and two real additions, and a complex addition requires two real additions. Thus, each new point requires four real multiplications and four real additions. However, one-half of the complex multiplies can be saved per pair of points, since $W^{q+N/2} = -W^q$ (see Appendix D). Therefore, each pair of points requires four multiplications and six additions. Equivalently, each new point requires two multiplications and three additions. Thus, the total number of real multiplies and real additions for an N -point transform are $2N\log_2 N$ multiplications and $3N\log_2 N$ additions. As is discussed in Appendix D, and as can be seen from the precedence graph in Fig. 12, many of these operations can be performed concurrently. Thus, each pair of new data points can be computed in one multiply cycle and two addition cycles. Hence, the total multiplication and addition cycles are $(N\log_2 N)/2$ and $N\log_2 N$, respectively. The configuration we propose incorporates this "horizontal" parallelism as well as the overlapping of multiplication and addition cycles (i.e. pipelining).

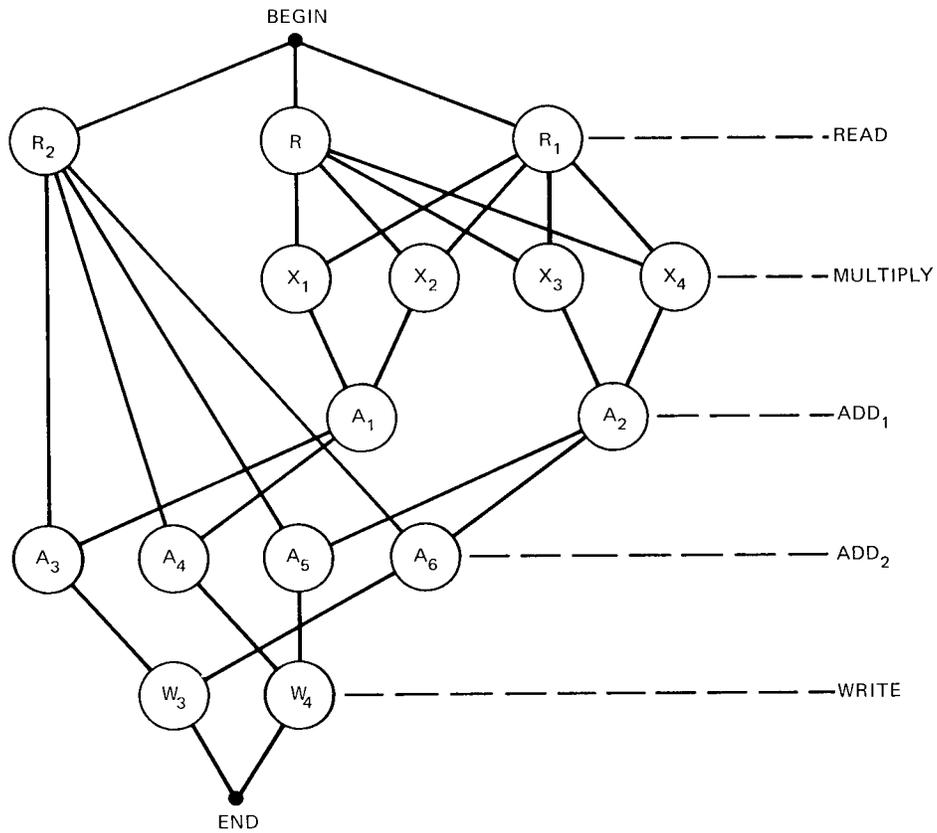
Consider the configuration represented in Fig. 12a. M_1 and M_2 represent data memories and each of them contains half the required number of complex data points. The coefficient store could be a separate ROM or a small hardware network to compute the proper coefficients. M_3 and M_4 represent data memories which store the results of computations on the data contained in M_1 and M_2 . Once N new points have been computed, the roles of the memories are interchanged. Now M_3 and M_4 contain the input data for the next stage, and the results of the computation are stored in M_1 and M_2 . (The initial loading of M_1 and M_2 , the address generation, and the memory switching will not be discussed at this time; however, they will be considered in a future report.)

The operators represent the required multipliers and adders (subtractors) each operating on 8-bit real data words (i.e., a 16-bit complex), to produce 8-bit data words. All components are considered to be current state-of-the-art. The operating speeds are as provided in Table 2.



PARALLEL-PIPELINED FFT BUTTERFLY

(a) Data flow graph



(b) Precedence graph

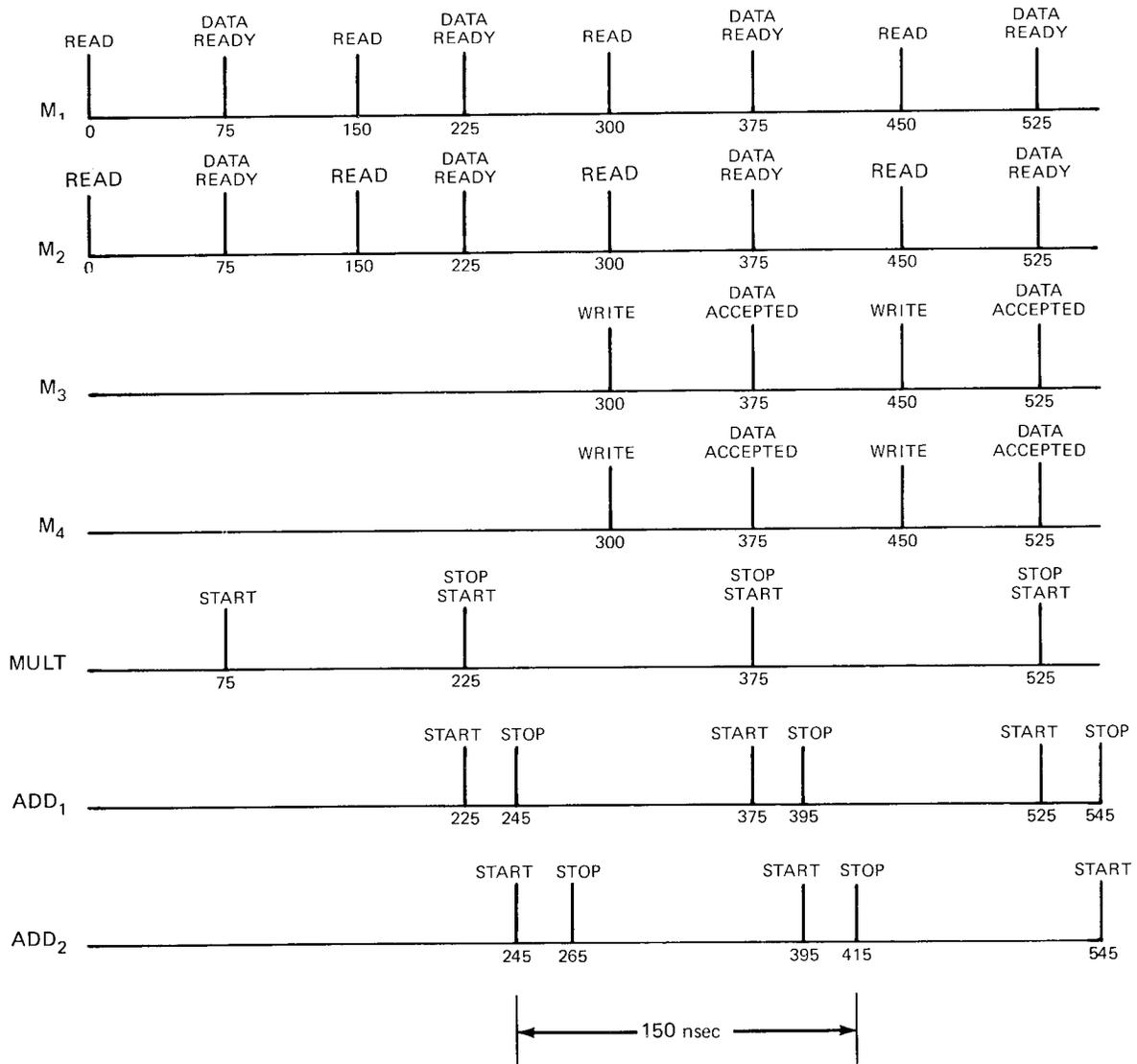
Fig. 12 — Parallel-pipelined FFT butterfly

The sequence of operations defined by the configuration will now be described. We assume that the input data have been loaded into M_1 and M_2 and that the addressing scheme, i.e., the particular FFT algorithm used, is wired into the design. (For this configuration an out-of-place algorithm is most appropriate.)

Two complex data points are read from M_1 and M_2 concurrently. As soon as the data are available, the multiplication cycle begins. At the completion of the multiplication cycle, the first addition cycle commences. This cycle is followed by the second addition cycle. The computed data points are now written into M_3 and M_4 . The timing diagram, in Fig. 13 describes the operation of the system in more detail. Since the memory cycle time and the multiply cycle time are matched, it is possible to pipeline the system at the multiply rate. Thus, the completion period of each FFT butterfly is 150 nsec. It should be noted that in this situation all memories and all multipliers are operating at 100% duty cycle. To achieve this duty cycle and throughput rate, parallelism was exploited in a variety of ways. For example, memories M_1 and M_2 are read concurrently, while memories M_3 and M_4 are written into concurrently. Thus, operations on four independent memories proceed simultaneously. The arithmetic element contains three horizontal levels of parallelism, all obvious from the precedence graph. It is apparent that four multiplies followed by two adds, and then four adds, are all performed concurrently. In addition, vertical parallelism is exploited by overlapping the next data fetch with the current multiply cycle, the next multiply cycle with current add cycles, and the current data storage with the next multiply cycle. The result of exploiting this maximal parallelism has been the realization of a throughput rate equal to that of the multiply rate.

The configuration we have just defined is capable of executing one stage of an FFT in $\tau N/2 \mu\text{sec}$, where τ is the time to compute an FFT butterfly on two complex data points. If we include gate transfer delays and address generation time, τ for this configuration is about 200 nsec. Thus, it is possible to compute a 1024-point transform in $512 \times 10 \times 0.2 \mu\text{sec}$, or 1.024 msec. This, of course, assumes that the data have been loaded into M_1 and M_2 in appropriate locations. The processing is then of the batch type. If the processing is to be done in real time with respect to the sampling clock, then $\log_2 N$ stages must be cascaded. In this case, a 1024-point transform will be completed in $512 \times 0.2 \mu\text{sec}$ or $102.4 \mu\text{sec}$. Typical FFT times for the batch and cascade modes are displayed in Table 3.

We now consider the timing constraints imposed by the desire to perform pulse compression in the frequency domain. The compression will be performed by first computing the FFT of the incoming signal (pulse), multiplying this by a set of reference Fourier coefficients, and then performing the inverse FFT on the product to obtain the compressed pulse. We assume that each pulse will be represented by N data points and N zeros so that the data aperture is essentially twice the uncompressed pulse width. This is necessary to avoid overlap caused by circular convolution. The excess zeros essentially allow us to transform a circular convolution into an aperiodic convolution. Since we have the capability of performing four multiplies simultaneously, the effective multiply time is assumed to be approximately $160/4$ or 40 nsec. We can now determine the time needed to perform pulse compression on pulses of various widths. For example, the $25.6\text{-}\mu\text{sec}$ pulse is represented by 64 data points, based on bandpass sampling (single channel). We now perform the FFT on 128 points (half are zeros) to obtain 64 useful coefficients. The coefficients are multiplied by the reference Fourier coefficients and then the inverse FFT is performed.



THROUGHPUT RATE \approx 7 MHz

1024-POINT TRANSFORM \sim 0.8 nsec (8-BIT REAL, 8-BIT IMAGINARY)

Fig. 13 — FFT butterfly timing

Table 3
Typical FFT Processing Times

Number of Bits/Word	Number of Points	FFT Processing Time (μsec)			
		Batch		Cascade	
		Number of Passes	Processing Time	Number of Stages	Processing Time
8 Real, 8 Im.	64	6	38.4	6	6.4
8 Real, 8 Im.	128	7	89.6	7	12.8
8 Real, 8 Im.	256	8	204.8	8	25.6
8 Real, 8 Im.	512	9	460.8	9	51.2
8 Real, 8 Im.	1024	10	1024	10	102.4
8 Real, 8 Im.	2048	11	2252	11	204.8

Thus, the time used to compress the 25.6- μsec pulse is

$$\left(2 \times \frac{128}{2} \log_2 128 \times 0.2 + 64 \times 0.04\right) \mu\text{sec},$$

or 184.3 μsec . This, of course, presumes a batch mode of operation requiring seven passes. If instead, we had cascaded seven stages for the FFT and another seven for the inverse, the computation time would have been

$$\left(2 \times \frac{128}{2} \times 0.2 + 64 \times 0.04\right) \mu\text{sec} \text{ or } 28.2 \mu\text{sec}.$$

The values in Table 4 were computed in a similar manner for various pulse widths.

Table 4
Pulse Compression Processing Time

Coded Pulse Width	Number of Points per Pulse	Pulse Compression Processing Time (μsec)			
		Batch		Cascade	
		Number of Passes*	Processing Time	Number of Stages	Processing Time
25.6	64	15	184.3	14	28.2
51.2	128	17	419.8	16	56.4
102.4	256	19	942.1	18	112.8
204.8	512	21	2089	20	225.6
409.6	1024	23	4588	22	451.2

*Assumes one pass per multiply.

Let us now refer to the specification of the radar system. The pulse repetition period T is 1024 μsec . Assuming that we wish to compress a single pulse within this time period, then with a single arithmetic unit in the batch mode, we are limited to pulses of duration up to 102.4 μsec , as seen from Table 4. This situation can be remedied by cascading a number of units so that the real time constraint is met. For example, consider the 204.8- μsec pulse represented by 512 data points. If we cascade k units, then the computation time for pulse compression is approximately

$$\frac{2N \log_2 2N}{k} \tau_B + N\tau_M, \quad k = 1, 2, \dots, \log_2 2N,$$

where τ_B is the FFT butterfly time and τ_M is the effective multiply time. Since this time must be less than T , we can determine the required k . For the 204.8- μsec pulse, then

$$\frac{(1024 \times 10)}{k} (0.2) + (512) (0.04) \leq 1024 \mu\text{sec}$$

$$\frac{2048}{k} + 20.48 \leq 1024 \mu\text{sec}$$

$$\frac{2048}{k} \leq 1024$$

$$k \geq 2 \quad (\text{neglecting multiply time}).$$

Therefore, we can conceivably meet the real-time constraint by cascading two processing elements. Similarly for the pulse width of 409.6 μsec , we have

$$\frac{(2048 \times 10) (0.2)}{k} + (1024) (0.04) \leq 1024$$

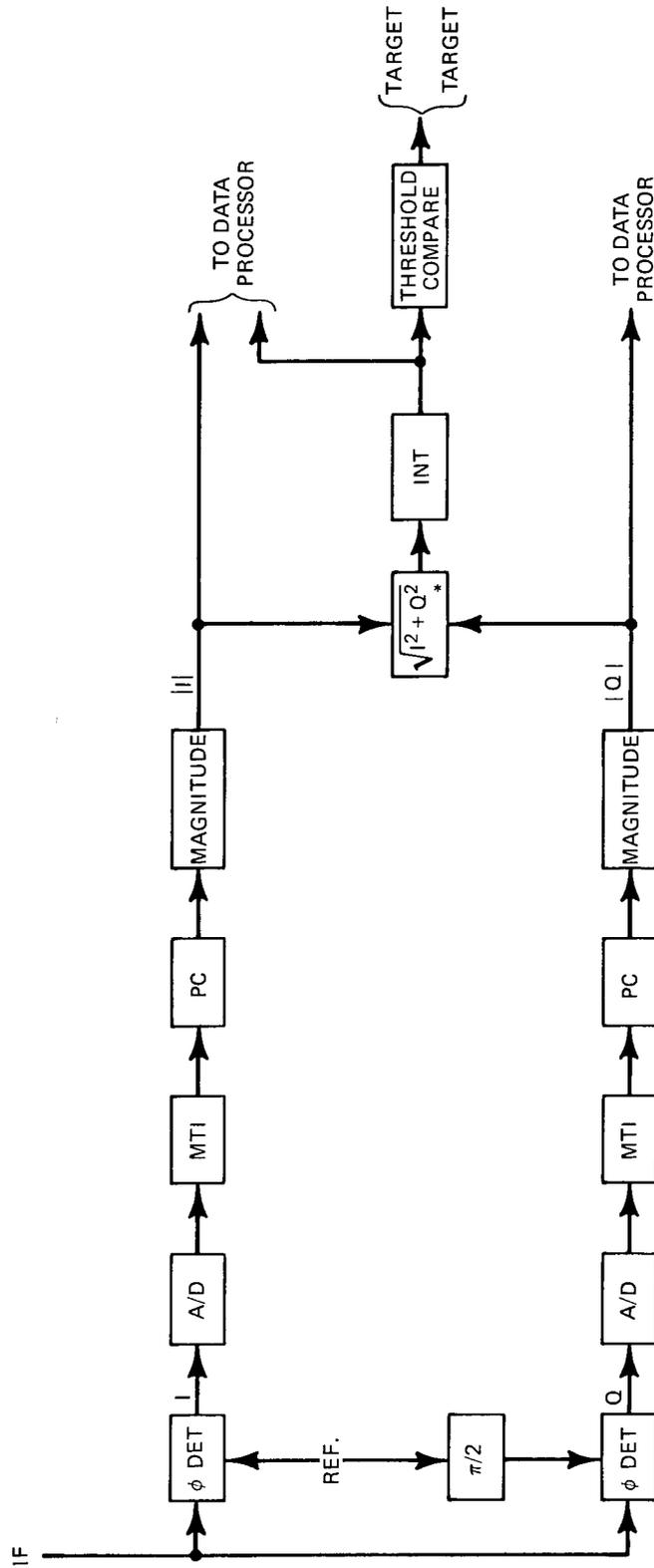
$$\frac{4096}{k} + 40.96 \leq 1024$$

$$k \geq 4 \quad (\text{neglecting multiply time}).$$

Thus, four cascaded processing elements will allow us to compress the 409.6- μsec pulse in the allowable 1024- μsec interval.

System Configuration

We have just demonstrated that an arithmetic element representing either the second-order filter or the FFT butterfly can perform various predetection radar signal processing algorithms in real time. As the real-time constraints increase because of higher signal bandwidths or more data points, more processing elements may be required per algorithm (e.g., pulse compression). In addition, the eventual decision as to whether a target is present at a particular range is made only after some set of operations has been performed on digitized signal returns. For example, a typical radar processing chain is illustrated in Fig. 14. (For



$$*\sqrt{I^2 + Q^2} \approx \text{MAX}(|I|, |Q|) + \frac{1}{2} \text{MIN}(|I|, |Q|)$$

Fig. 14 — Predetection processing

a monopulse system, each channel actually consists of three or four separate signals, i.e., Σ , Δ_{el} , Δ_{az} , each of which undergoes similar signal processing.) It is our contention that a single type of processing element, incorporating the commonality of the FFT butterfly and the second-order filter, may be used to execute each of the algorithms implied by the signal processing blocks in Fig. 15. This implies that the chain of processing will be composed of some number of identical processing elements, each of which will be controlled to execute a particular function within the chain. In addition, each function may be computed by one or more processing elements if the real-time constraints demand it.

The array represented in Fig. 15 is a generalization of the foregoing discussion. Each large block represents a particular signal processing task. Each small block contained within a task block represents a signal processing element. This element contains a small micro-program control unit which essentially configures (and possibly sequences) the processing element to perform a basic function such as a second-order filter, an FFT butterfly, or a sum of four products, etc. In addition, each processing element, when self-contained, includes four data memories as well as the arithmetic section. A hardware address generator may be included in each element for FFT applications (see Fig. 16). The particular function to be performed by the individual processing elements (e.g. FFT) and significant parameters will be passed to the individual elements from an external "supervisor." Note that the functions performed within a given column will be identical, so the external supervisor does not have to be too "smart."

While the array in Fig. 15, represents the signal processor in its most general form, there are many situations in which only a small number of processing elements will be required. In fact, for situations involving one channel and moderate bandwidths, one processing element may be sufficient.

IV. CONCLUSIONS

The bulk of radar signal processing prior to a detection decision can be performed digitally, and in real time, by a programmable computing structure. That is the general conclusion of this work; however, there are numerous conditions upon which that statement is based.

Moving target indication, pulse compression, doppler filtering, and video integration can all be decomposed into various combinations of basic algorithmic kernels. The kernels of particular interest are the general second-order recursive digital filter and the fast Fourier Transform butterfly. By exploiting maximal parallelism within the kernels, configurations have been defined which execute the filter computation in about 400 nsec and the butterfly in about 200 nsec. Current state-of-the-art technology is assumed for both arithmetic elements (multipliers and adders) and memory units. The second-order filter can be considered a "complete" MTI filter, whereas the butterfly is only a small part of a total Fourier transform. However, with this butterfly as a basic module, a 1024-point complex Fourier transform can be completed in approximately 1 msec.

By performing two FFT's and a multiply, pulse compression can be realized in real time with one processing element, for pulse widths up to 100 μ sec. Longer pulses and/or higher bandwidths can be accommodated by cascading some number of processing elements. Digital MTI filters of higher complexity can be synthesized by iterating the basic second-order filter.

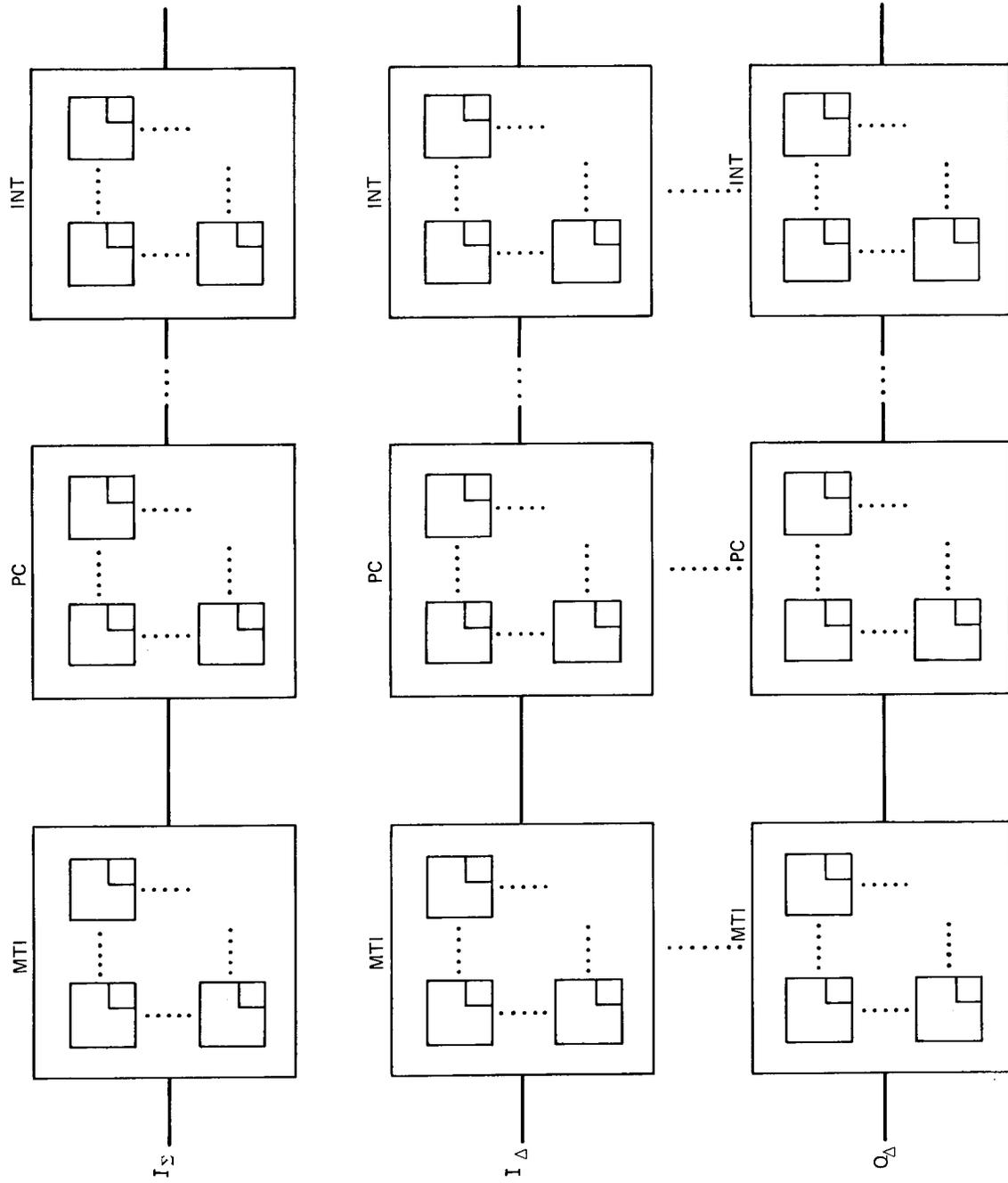


Fig. 15 — Generalized signal processor

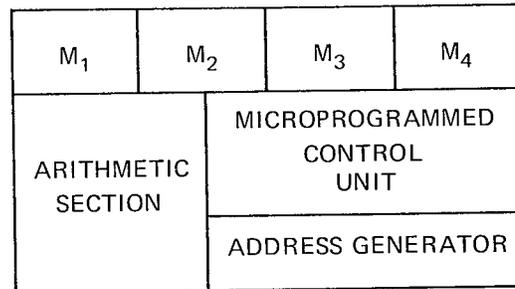


Fig. 16 — Signal processing element

Program schema have been used to display maximal parallelism within the kernels as well as to indicate the required control sequence for computing the two kernels of interest. These models have, in addition, shed light on the problem of controlling a single arithmetic element to behave either as a second-order filter or as an FFT butterfly. It should be noted that there is a large degree of commonality between the filter and butterfly computation. For example, both kernels require four multiplications which can be computed concurrently, four memories for data storage, a small coefficient store, and similar arithmetic sequencing. Differences arise primarily with respect to the data flow paths. A small microprogrammed control unit within a single arithmetic element may be used to direct the data flow. In addition, it can provide a means for utilizing the numerous subfunctions within the basic kernels. For example, the four multipliers may be used to compute the sum of four product terms simultaneously. The details of the foregoing approach will be reserved for future reports (see, for example, Ref. 8).

A general observation concerns the apparent hierarchy in which potential parallelism arises within signal processing algorithms. At the lowest level, for example, is the horizontal and vertical parallelism exploited within the basic computational kernels. This involves mainly the potential for concurrent arithmetic operations, overlapping of fetch and execute cycles, and pipelining where appropriate. The next level of parallelism is defined by the concurrent operation of a number of processing elements to satisfy real-time constraints (e.g., pulse compression). At a higher level, it may be required to process a number of signal channels simultaneously, as in the pulsed radar system. In addition, the chain of processing (i.e., MTI, PC, Integration) within each channel represents a possible pipeline situation, or a fourth level of parallelism. A total signal processing system, then, can be considered a generalized hierarchical parallel processor. However, there exists one major difference and that is with respect to the workload. The parallel processor we have defined is a direct outgrowth of the potential parallelism within signal processing algorithms and the environment in which they occur. In this sense, the problem of discovering applications for a given parallel architecture in order to justify the structure is nonexistent.

REFERENCES

1. Shay, B., and Shore, J.E., "Development of a General Purpose Signal Processor: Signal Processing Tasks of the AN/TPS-59," NRL Memorandum Report 2353 (Confidential Report, Unclassified Title)
2. Bergland, G.D., "Fast Fourier Transform Hardware Implementations — An Overview," IEEE Trans. Audio and Electroacoust. AU-17 (No. 2), 104-108 (June 1969)

3. Dennis, J.B., "Computation Structures," Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, copyright 1970
4. Karp, R.M., and Miller, R.E., "Parallel Program Schemata," *J. Comput. Syst. Sci.* **3** (No. 2), 147-195 (May 1969)
5. Keller, R.M., "On Maximally Parallel Schemata," *IEEE Conference Record of the Eleventh Annual Symposium on Switching and Automata Theory*, 1970, pp. 32-50
6. Thomas, J., "Introduction to Statistical Communication Theory," Ch. 7, New York, Wiley, 1969, pp. 400-402
7. Roecker, W., "The Application of Digital Filters for Moving Target Indication," *IEEE Trans. Audio Electroacoust.* **AU-19** (No. 1), 72-77 (Mar. 1971)
8. Smith, W.R. and Smith, H.H., "Signal Processing Element Functional Description (Part II, Preliminary) — Signal Processing Arithmetic Unit," *NRL Memorandum Report 2522*.

BIBLIOGRAPHY

Recursive Filter Design

- Gold, B., and Rader, C.M., "Digital Processing of Signals," New York, McGraw-Hill, 1969
- Kuo, F.F., and Kaiser, J.F., eds., "System Analysis by Digital Computer," New York, Wiley, 1966
- Oppenheim, A.V., ed., "Papers on Digital Signal Processing," Cambridge, M.I.T. Press, 1969

Z-Transform Theory

- Jury, E.I., "Theory and Application of the Z-Transform Method," New York, Wiley, 1964
- Gold, B., and Rader, C.M., "Digital Processing of Signals," New York, McGraw-Hill, 1969

Fast Fourier Transform

- Bergland, G.D., "A Guided Tour of the Fast Fourier Transform," *IEEE Spectrum* **6** (No. 7), 41-52 (July 1969)
- Gold, B., and Rader, C.M., "Digital Processing of Signals," New York, McGraw-Hill, 1969
- Cooley, J.W., and Tukey, J.W., "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput.* **19**, 297-301 (Apr. 1965)

Program Schemata

- Dennis, J.B., "Computation Structures," Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, copyright 1970

Karp, R.M., and Miller, R.E., "Parallel Program Schemata," J. Comput. Syst. Sci. 3 (No. 2) 147-195 (May 1969)

Keller, R.M., "On Maximally Parallel Schemata," IEEE Conference Record of the Eleventh Annual Symposium on Switching and Automata Theory, pp. 32-50, 1970

Radar

Berkowitz, R.S., ed., "Modern Radar," New York, Wiley, 1965

Nathanson, F.E., "Radar Design Principles," New York, McGraw-Hill, 1969

Skolnik, M.I., ed., "Radar Handbook," New York, McGraw-Hill, 1970

Appendix A
RADAR SIGNAL PROCESSING

MOVING TARGET INDICATION

MTI systems are used to detect moving targets in the presence of clutter. Basically, an MTI system is a filter which rejects energy at clutter frequencies and passes energy at the doppler shifted frequencies due to returns from moving targets. For a pulsed radar system, where T is the pulse repetition period (PRP), energy returned from stationary clutter will be concentrated at multiples of the pulse repetition frequency (PRF), $1/T$. Because of the doppler effect, energy returned from moving targets will be somewhat displaced from these frequencies. Accordingly, a filter whose response is indicated in Fig. A1 will pass signals due only to moving targets. In its simplest form, an MTI filter subtracts two successive returns (pulses) from the same location. Reflections from stationary clutter will then cancel, while those from moving targets will be passed. This form of filter is known as a single-delay canceler and is shown in Fig. A2.

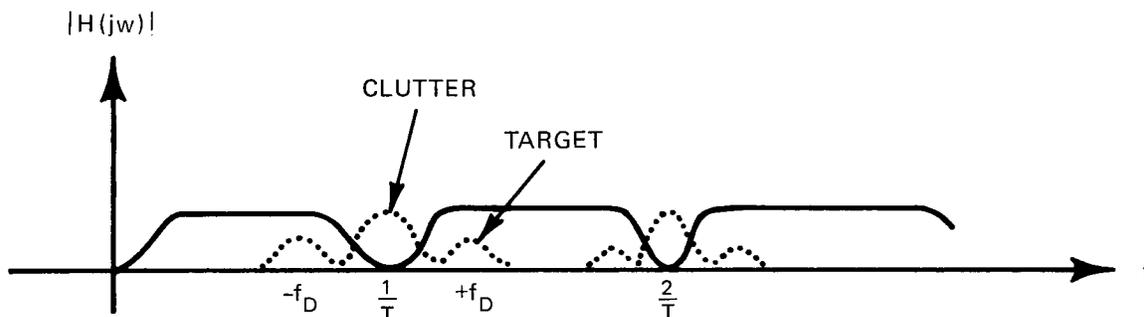


Fig. A1 — MTI filter response

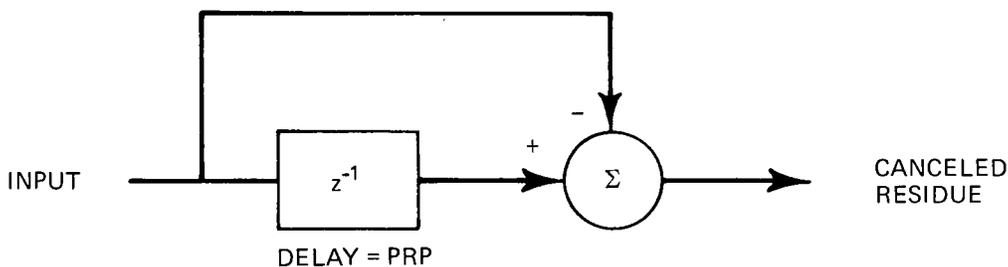


Fig. A2 — Single-delay canceler

Frequently, clutter is nonstationary; that is, it contains nonzero velocity components in the radial direction. To remove energy due to this velocity component, a technique known as clutter-locking can be used. In this technique, the phase difference between two successive returns due to the average phase change of moving clutter returns is fed back to a phase shifter which modifies the phase of the undelayed signal accordingly. Essentially, this feedback network allows the filter to lock onto the moving clutter and use that as a reference. Signals will then be passed whose concentration of energy is displaced from multiples of the average clutter frequencies. A simplified version of a clutter-locked filter is shown in Fig. A3.

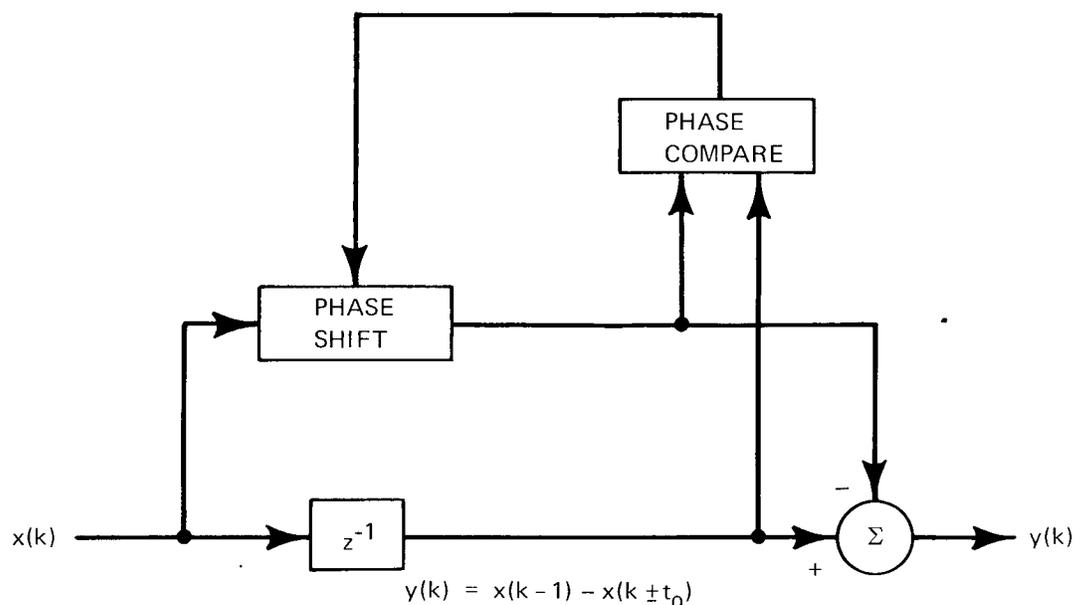


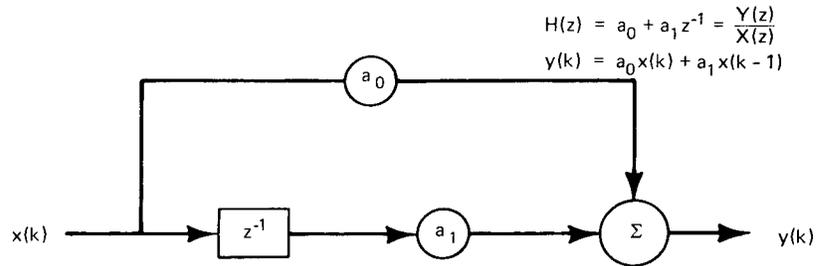
Fig. A3 — Clutter-locked MTI filter; t_0 is the relative phase shift due to moving clutter

More sophisticated MTI filters can be obtained by canceling more pulses, weighting pulses differently according to their delay, and using feedback paths. These variations define the notches and pass bands of the filter. In Figs. A4a through f, typical MTI filters are shown. Along with each filter is the difference equation which describes its behavior and the corresponding transfer function in the z -domain, where y^{-1} represents a delay of T_0 the PRP.

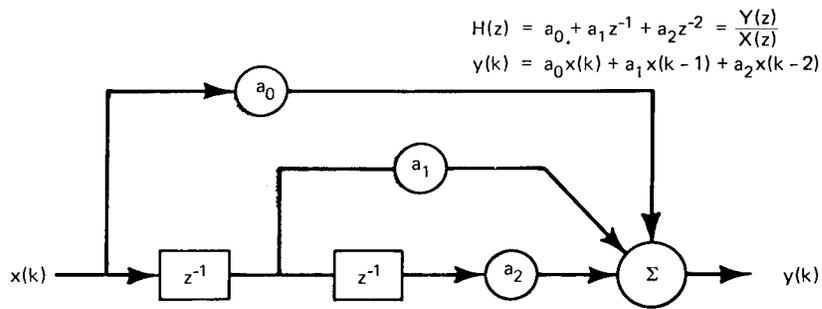
PULSE COMPRESSION

It has been shown* that if an optimum detection procedure is used, the sensitivity of the radar receiver depends only on the total energy of the received signal and not on its form. In order to avoid generating high peak power signals to achieve increased energy, it is necessary to generate long pulses. This not only increases the average power and hence

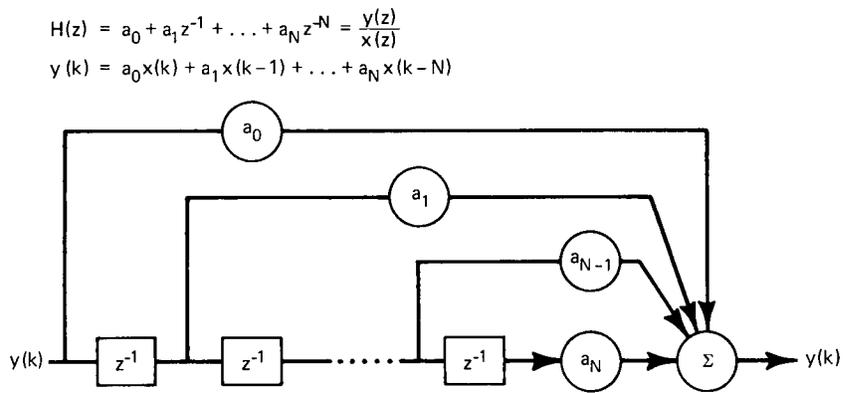
*Berkowitz, R.S., "Modern Radar," New York, Wiley, 1965



(a) Nonrecursive single delay



(b) Nonrecursive double delay



(c) Nonrecursive *n*-delay

$$H(z) = \prod_{i=1}^N (a_{0i} + a_{1i}z^{-1}) = \frac{Y(z)}{X(z)}$$

$$a_{0i} = -1, \quad a_{1i} = 1$$

$$H(z) = (1 - z^{-1})^N = \sum_{j=0}^N (-1)^{N-j} \binom{N}{j} z^{j-N}$$

$$y_1(k) = a_{01}x(k) + a_{11}x(k-1)$$

FOR N = 2

$$y_i(k) = a_{0i}y_{i-1}(k) + a_{1i}y_{i-1}(k); \quad i = 2, \dots, N$$

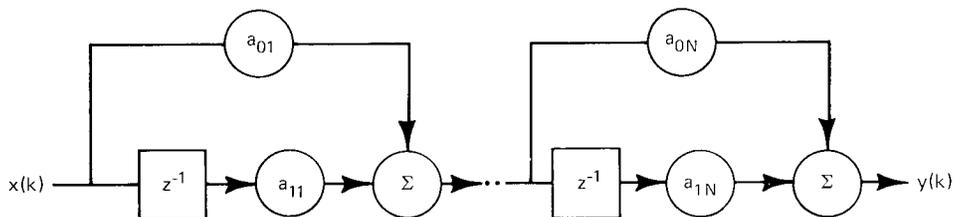
$$H(z) = (1 - z^{-1})^2 = 1 - 2z^{-1} + z^{-2}$$

$$y(k) = y_N(k)$$

$$y(k) = x(k) - 2x(k-1) + x(k-2)$$

CASCADE

TYPICAL CASE

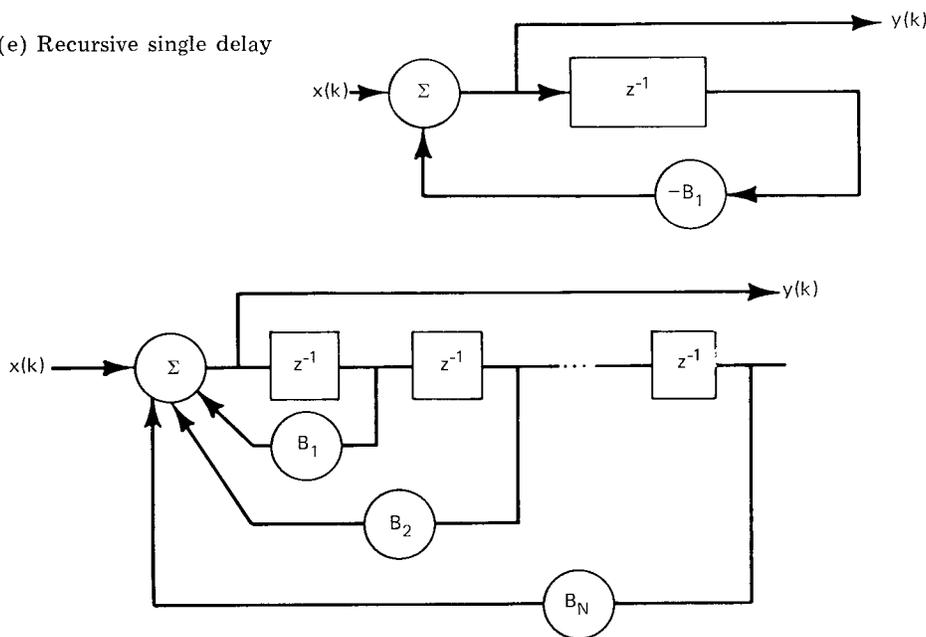


(d) Equivalent to cascade filter where the weights are $a_0 = 1, a_1 = -2, a_2 = 1$

$$H(z) = \frac{1}{1 + B_1z^{-1}} = \frac{Y(z)}{X(z)}$$

$$y(k) = x(k) - B_1x(k-1)$$

(e) Recursive single delay



$$H(z) = \frac{1}{1 + B_1z^{-1} + B_2z^{-2} + \dots + B_Nz^{-N}} = \frac{Y(z)}{X(z)}$$

$$y(k) = x(k) - \sum_{i=1}^N B_i y(k-i)$$

(f) Recursive multiple delay

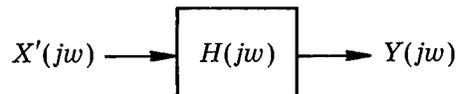
Fig. A4 - MTI filter forms

sensitivity, but also improves the velocity resolution of the receiver. On the other hand, the range resolution (i.e. the ability to distinguish multiple targets) suffers. By coding the transmitted signal the effective bandwidth is increased, thereby improving the range resolution capabilities of the radar. Transmitting this coded, long pulse and processing the received signal in a "matched" filter simultaneously improves the sensitivity, range, and velocity resolution of the radar. The "matched" filter, by taking advantage of the high time-bandwidth product of the incoming signal, essentially compresses the incoming pulse into a very narrow pulse, thereby obtaining the range resolution of a narrow pulse system. The process of filtering the coded signal to produce a narrow pulse is generally called pulse compression and, in this context, the matched filter is the pulse compression filter.

While the performance of a pulse compression system depends upon the type of coding selected (e.g., linear FM, phase coding, etc.), the actual implementation of the matched filter depends upon which mathematically equivalent formulation of the filter is chosen. The equivalent arises because the properties of the Fourier Transform permit the matched filter to be expressed in either the time or the frequency domain or both.

A filter is considered "matched" to its input signal when its transfer function is the complex conjugate of the Fourier transform of the signal. Since the transform of the output of the filter is the product of the input transform and the transfer function, the output time signal can be found by performing an inverse transform on this product. These operations are described here diagrammatically.

	<u>Time</u>	<u>Frequency</u>
Transmitted Waveform	$x(t)$	$X(j\omega)$
Received Waveform	$x'(t)$	$X'(j\omega)$
Filter Output	$y(t)$	$Y(j\omega)$
Transfer Function (Impulse Response)	$h(t)$	$H(j\omega)$



For a matched filter,

$$H(j\omega) = X^*(j\omega) \approx [X'(j\omega)]^*$$

$$y(t) = F^{-1} [X^*(j\omega)X'(j\omega)]$$

where X^* represents the complex conjugate of X .

This formulation can be implemented directly in hardware as shown in Fig. A5. In this situation, the reference waveform $x(t)$ and its transform would be locally generated.

From the theory of the Fourier transform, it can be shown that the inverse transform of a product of two transforms is equivalent to the convolution of the two time functions. For the matched filter then, the output can be computed by convolving the received signal

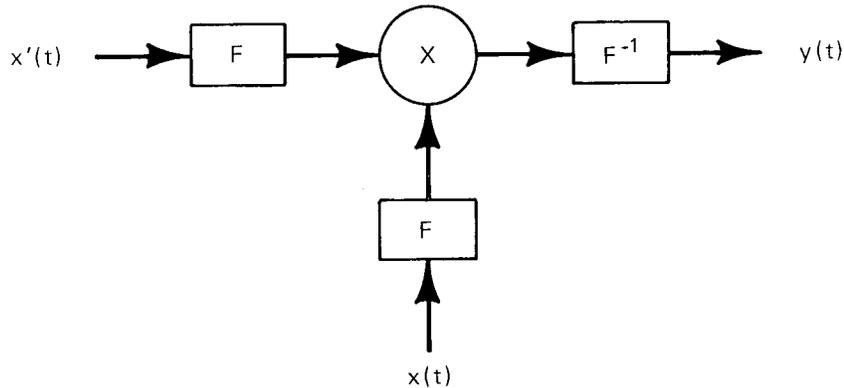


Fig. A5 — Pulse compression in the frequency domain

with the impulse response of the filter. Since the transfer function for the matched filter is the conjugate of the transmitted signal, the impulse response is the time inverse of the reference signal. The required convolution is therefore equivalent to the cross correlation of the received signal with the reference (transmitted) signal. This observation leads to a direct implementation in hardware as depicted in Fig. A6. Thus the matched filter can be implemented in either the time domain (Fig. A6) or the frequency domain (Fig. A5).

Since we are primarily concerned with performing radar signal processing numerically we now discuss digital implementations of the matched filter. Performing the Fourier transform on a sampled signal leads to what is called the Discrete Fourier Transform (DFT) of a sequence of numbers. The discrete version of the “matched” filter in the frequency domain is shown in Fig. A7. In this situation, the Fourier coefficients of the reference signal would be stored. As will be shown later, the advent of the Fast Fourier Transform (FFT) renders the frequency version a viable candidate for implementation as a pulse compression filter.

As has been discussed, convolution with the inverse time function is equivalent to correlation. This equivalence is directly applicable in the discrete case and is depicted in Fig. A8 as the time domain matched filter.

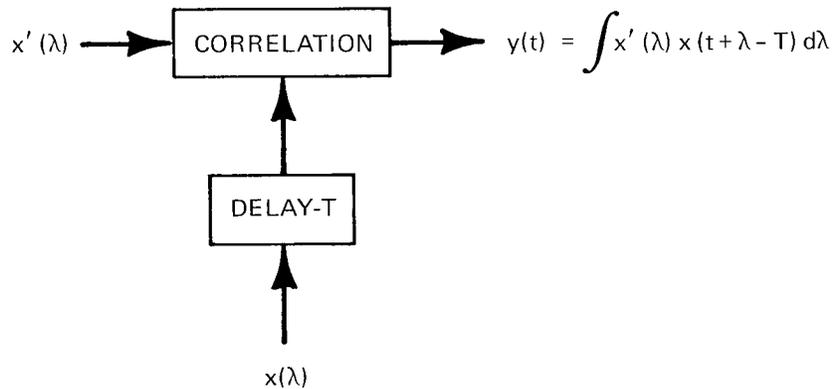


Fig. A6 — Pulse compression in the time domain

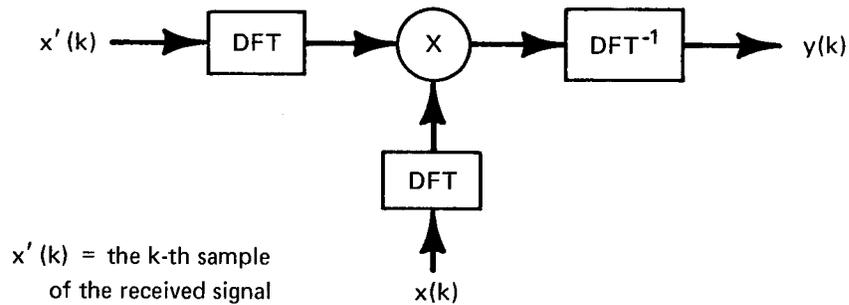


Fig. A7 — Digital pulse compression in the frequency domain

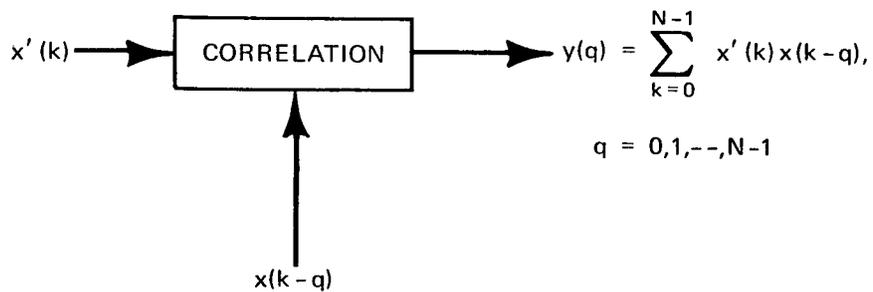


Fig. A8 — Digital pulse compression in the time domain

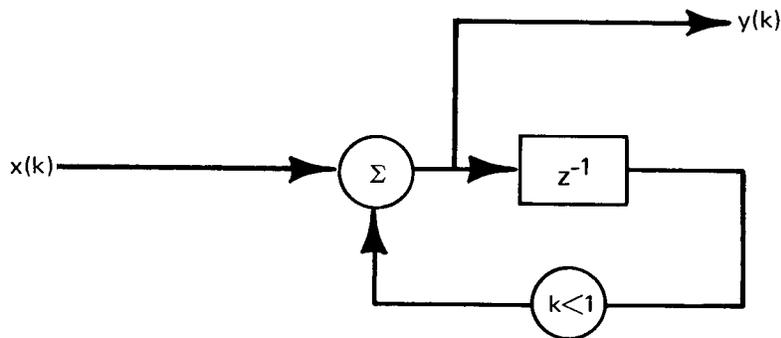


Fig. A9 — Digital integrator

VIDEO INTEGRATION

A digital integrator is a device which adds a delayed sample to an incoming sample and stores the result. This result is then delayed (stored) until the next sample arrives, at which time the two are added and the result is stored. This process continues until a threshold is reached, at which time a detection is made. Generally, the stored samples are first multiplied by a constant less than 1. The digital integrator is represented in Fig. A9. It should be noted that this integrator is equivalent to the single-delay recursive filter depicted in Fig. A4e.

Appendix B

GENERAL DIGITAL SIGNAL PROCESSING

In very general terms, the envisioned programmable signal processor will operate on samples of a high bandwidth demodulated (video) signal to produce a detection decision. The intermediate processes essentially enable the radar system to "look" at only what it is of interest (e.g., moving targets) while maximizing the chances of a correct detection decision. Accordingly, the entire signal processor can be thought of as a digital filter composed of various combinations of operations involving convolution, spectrum generation, and recursive and nonrecursive digital filtering. These operations are all variations of what is termed a general linear digital filter defined as follows.

A *digital filter* is a device, or operation, which transforms a sequence of numbers into another sequence of numbers possessing the properties of

1. Linearity: If $\{x_n\} \rightarrow \{y_n\}$ and $\{v_n\} \rightarrow \{w_n\}$,
then $a\{x_n\} + b\{v_n\} \rightarrow a\{y_n\} + b\{w_n\}$.
2. Time Invariance: If $\{x_n\} \rightarrow \{y_n\}$,
then $\{x_{n+\nu}\} \rightarrow \{y_{n+\nu}\}$.

CONVOLUTION

As with analog linear filters, a digital filter can be characterized either by its impulse response, frequency response, or by the linear equations relating its input to its output. In the digital or discrete case, the impulse response is the output of the filter due to an input

$$x_n = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}.$$

If we let $H(k\Omega)$ be the DFT of h_n , then it can be shown that the output y_n of a digital filter, characterized by $H(k\Omega)$, is

$$y_n = \text{IDFT} [X(k\Omega) H(k\Omega)]$$

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} X(k\Omega) H(k\Omega) \exp\left(j \frac{2\pi}{N} nk\right), \quad n = 0, 1, \dots, N-1.$$

Thus, the output of the filter can be computed when $H(k\Omega)$, the frequency response of the filter at the frequencies $(2\pi/N)k$ ($k = 0, 1, 2, \dots, N-1$), is known.

DIFFERENCE EQUATIONS (RECURSIVE AND NONRECURSIVE FILTERING)

The output of a digital filter can be computed directly from the difference equations which characterize its behavior. It can be shown that if x_n is the input sequence and y_n is the output sequence, then

$$y_n = \sum_{i=0}^m a_i x_{n-i} - \sum_{i=1}^p b_i y_{n-i}.$$

The filter is thus characterized by m , p , and the coefficients a_i , b_i .

The three distinct, but equivalent, characterizations of a digital filter are presented in Figs. B1, B2, and B3.

If we let h_n be the impulse response, then by the principle of superposition, the output $y(n)$ of the filter due to any input sequence x_n is

$$y_n = \sum_{j=-\infty}^{\infty} x_j h_{n-j} = \sum_{i=-\infty}^{\infty} x_{n-i} h_i.$$

Hence, the output sequence is the weighted sum over all previous values of the input sequence. The weights constitute the impulse response and thus characterize the filter. This computation is termed linear convolution.

FOURIER TRANSFORM (SPECTRUM GENERATION)

Corresponding to the Fourier transform of an analog signal, the discrete Fourier transform (DFT) of a sampled signal (sequence) x_n is defined as

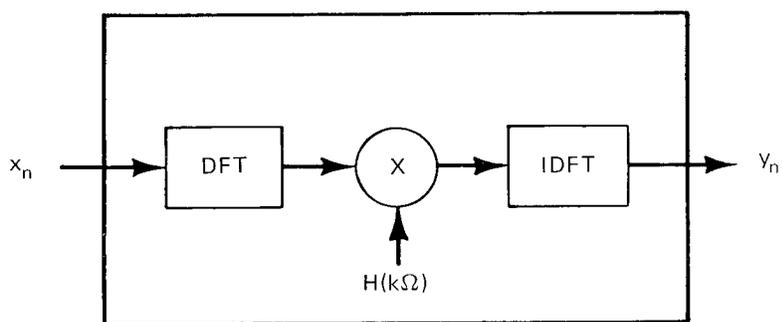
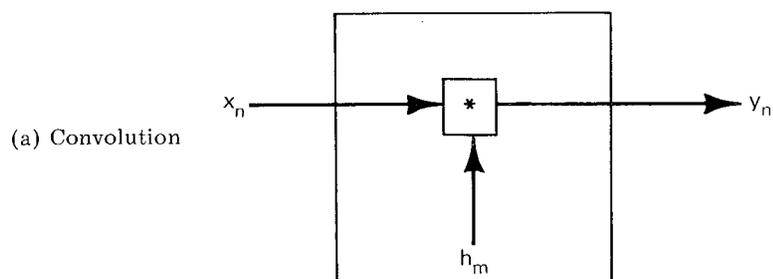
$$\text{DFT } [x_n] = \sum_{n=0}^{N-1} x_n \exp\left(-j \frac{2\pi}{N} nk\right) = X(k\Omega),$$

$$k = 0, 1, \dots, N - 1,$$

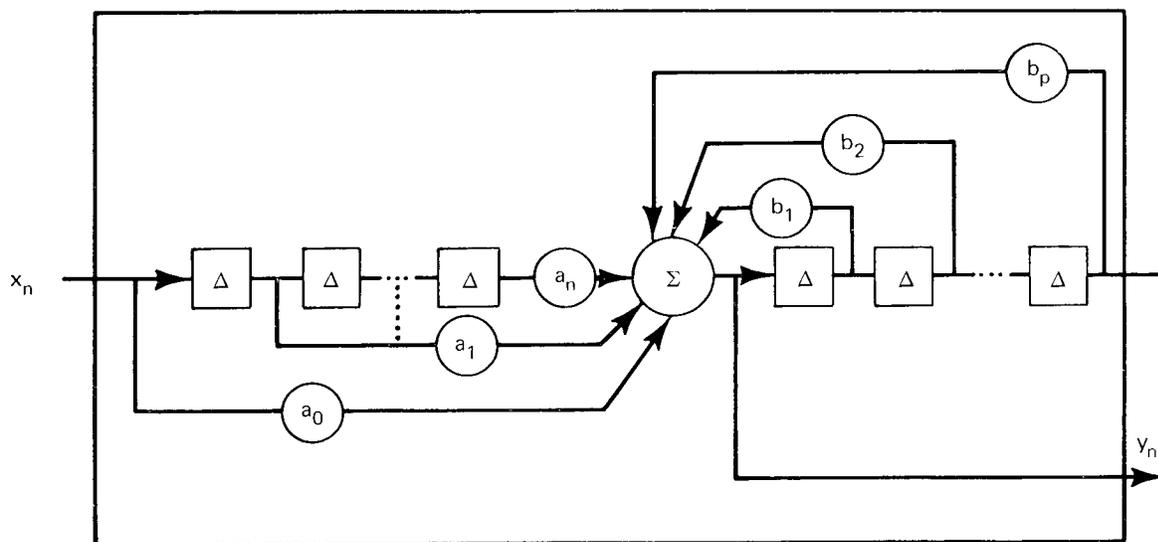
where $\Omega = (2\pi/N)$, and N is the number of samples to be transformed. Similarly, the Inverse Discrete Fourier Transform (IDFT) transforms $X(k\Omega)$ back to the original time sample and is defined by

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X(k\Omega) \exp\left(j \frac{2\pi}{N} kn\right), \quad n = 0, 1, \dots, N - 1.$$

In Fig. B1a, h_n , represents the impulse response of the filter. These values would be stored and read out in reverse order if the input sequence arrived in real time. In



(b) Fourier transform



(c) Difference equation

Fig. B1 — Digital filter characterizations

Fig. B1b, $H(k\Omega)$ is the DFT of h_m and represents the frequency response of the filter. These values would be stored and read out for each k . In Fig. B1c, Δ represents the delay corresponding to the sampling interval used for direct implementation of the filter via the difference equation.

It should be clear that the three implementations of the digital filter lead to three distinct computational algorithms for evaluating the output. Which algorithm or realization one chooses depends upon the speed requirements, hardware complexity and accuracy, and ease of programming. In addition, within each algorithm there are computational variations which can affect the accuracy and speed of the computations. For example, convolution is essentially a vector dot product for each n . Hence, the additions can be partitioned to incorporate parallelism or to minimize storage. For the case of direct evaluation of difference equations, there are many implementations which affect the accuracy and speed. These will be discussed in more detail later. In Fig. B1b, great increases in computational speed can be obtained by computing the DFT and IDFT using any of the FFT algorithms.

An MTI filter, in order to extract echoes from moving targets, places stopbands about all multiples of the PRP. This implies that the delay elements store samples for a duration T , at which time the sample is operated upon. For an MTI filter, however, one or more samples represent the return for a particular range of interest (range gate). In this situation, the storage medium must hold samples for all range gates. Thus for real-time operation, the MTI filter must be capable of processing data at the sampling rate, although the filter characteristics depend upon the pulse repetition period. Accordingly, the filter may be designed on a PRP basis (i.e., assuming one range gate) as long as the computations can be completed at the sampling rate. Thus, designing an MTI filter is equivalent to designing either a recursive or a nonrecursive digital filter of specified frequency characteristics.

It should be apparent that any implementation of the general matched filter can be used to realize a pulse compression filter. In particular, the reduction in the number of computations due to the FFT allows realistic signals to be compressed via the frequency domain.

The video integrator is a special case of the computation representing the solution to the general difference equation and can be considered a first-order digital low pass filter (see Fig. A4e).

The foregoing discussion indicates that the radar signal processing tasks of interest are essentially special cases of the general signal processing algorithms presented in this section. In particular, Table B1 displays the equivalences.

Table B1
Generalized Radar Signal Processing

	Moving Target Indication	Pulse Compression	Video Integration	Pulse Doppler	Waveform Generation
Spectrum Generation (Fourier Transform)		X		X	X
Convolution/Correlation		X			
Difference Equations (Recursive and Nonrecursive Filter)	X		X	X	

Appendix C

RECURSIVE FILTER DESIGN

Both MTI filters and digital integrators may be considered particular implementations of general recursive and nonrecursive digital filters. Therefore, the problem of designing an MTI filter is equivalent to designing a recursive or nonrecursive digital filter whose frequency response is determined by the PRP and the doppler shift. Recursive digital filters seem better suited to MTI applications than do nonrecursive filters since the filter characteristics can more easily be altered.

There are numerous methods for designing recursive digital filters specified in the frequency domain. One of the most convenient is to transform the digital filter specifications (critical frequencies, etc.) to equivalent analog filter specifications. The filter is then designed as an analog filter. Once the analog filter is designed, applying the bilinear transformation $s \rightarrow (z + 1)/(z - 1)$ yields the desired digital filter as a ratio of polynomials in z . The coefficients determine the locations of the poles and zeros in the z -plane and represent the feedforward and feedback multiplying factors in the realization.

Before discussing various implementations of digital recursive filters, we present some introductory material concerning the z -transform.

Z-TRANSFORM

A digital filter, when applied to a sampled signal, possesses frequency-selective properties akin to those of an analog filter applied to a continuous signal. While the DFT can be used to represent the frequency characteristics of the filter, the periodic frequency response caused by the sampling process renders this approach unattractive for analysis and synthesis. A transformation $z = \exp(j\omega t)$, which maps the $j\omega$ axis onto the unit circle and the left- and right-hand planes interior to and exterior to the unit circle, respectively, displays the redundancy as circular symmetry. The transfer function which can now be expressed in terms of z contains all the frequency-selective information about the filter. Since the filter can be represented as a linear difference equation, the z -transform technique can be used to reduce the solution of the difference equation to that of solving an algebraic equation in z . In the analysis to follow, it will be convenient to characterize digital filters in terms of both z and the difference or recursive equations. Accordingly, the z -transform will be defined more precisely.

Let $\{x_n\}$ be the sequence of numbers representing the sampled function $x(n\tau)$, where τ is a fixed positive integer (i.e., the sampling interval). Then the z -transform of the sequence is defined as

$$Z[x_n] = X(z) = \sum_{n=0}^{\infty} x_n z^{-n} = \sum_{n=0}^{\infty} x(n\tau) z^{-n} = Z[x(n\tau)].$$

Some properties of the z -transform which will be used implicitly in the analysis are listed below.

1. Linearity: $Z(ax_1 + bx_2) = aZ(x_1) + bZ(x_2)$
2. Time Displacement: (assuming zero initial condition)

If $Z(x) = X(z)$, then

- a. $Z[x(t + \tau)] = zX(z)$
- b. $Z[x(t + k\tau)] = z^k X(z)$
- c. $Z[x(t - k\tau)u(t - k\tau)] = z^{-k} X(z)$

$$\text{where } u(t - k\tau) = \begin{cases} 1, & t - k\tau \geq 0 \\ 0, & t - k\tau < 0 \end{cases}$$

From 2c it can be seen that multiplication by z^{-1} represents a delay in time of the underlying signal by an amount τ .

Consider the general representation of the difference equation.

$$y_n = \sum_{i=0}^m a_i x_{n-i} - \sum_{i=1}^p b_i y_{n-i}.$$

If we compute the z -transform of both sides of the equation, then by applying properties 1, 2, and 3 (with $\tau = 1$) we obtain

$$Z[y_n] = \sum_{i=0}^m a_i Z[X_{n-i}] - \sum_{i=1}^p b_i Z[Y_{n-i}]$$

$$Y(z) = \sum_{i=0}^m a_i z^{-i} X(z) - \sum_{i=1}^p b_i z^{-i} Y(z)$$

$$Y(z) = X(z) \sum_{i=0}^m a_i z^{-i} - Y(z) \sum_{i=1}^p b_i z^{-i}$$

$$Y(z) = \left[1 + \sum_{i=1}^p b_i z^{-i} \right]^{-1} X(z) \sum_{i=0}^m a_i z^{-i}$$

$$Y(z) = \frac{\sum_{i=0}^m a_i z^{-i}}{1 + \sum_{i=1}^p b_i z^{-i}} X(z) = H(z) X(z).$$

Thus $H(z)$, termed the system function, relates the input z -transform to the output z -transform. It is this function which defines the frequency-selective properties of the filter. This is so because the location of the poles and zeros in the z -plane uniquely determines the transfer function. Since the z -transform maps the imaginary axis of the s -plane onto the unit circle in the z -plane, frequency is now measured along the circumference of the unit circle.

FILTER FORMS

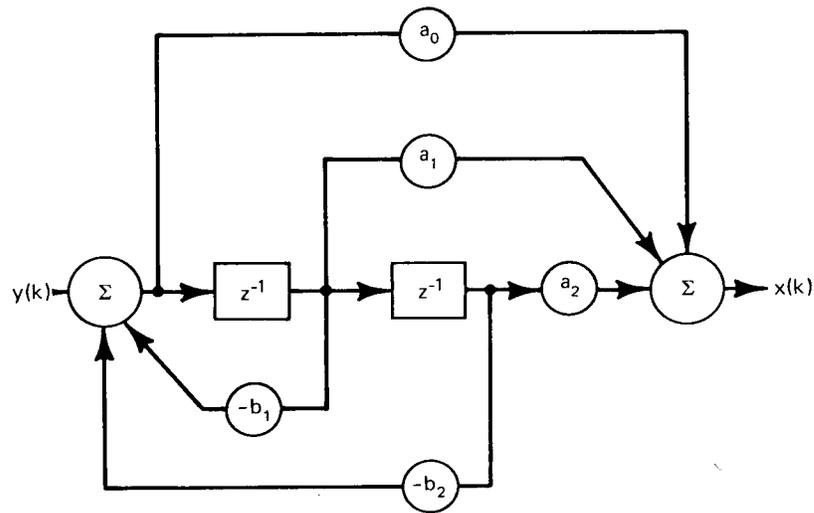
Realizations of digital filters can be implemented physically in terms of delay elements along with adders and multipliers. Filters composed of these elements are essentially direct representations of linear difference equations. Each filter has four basic forms or algorithms for its implementation. These are generally referred to as (a) direct, (b) direct canonic, (c) cascade, and (d) parallel implementations. Each realization, though structurally different, produces equivalent results when word length and speed of computation are of no consequence. However, since we will be dealing with a finite word-length machine, the particular realization we choose to implement is of major concern. Finite word-length requirements essentially affect the performance of the digital filter by adding noise to the system in three ways:

1. The quantization of the input, which is a function of the A/D converter used.
2. The rounding or truncation of products or sums of products which are fed back to be used in further computations.
3. The quantization of the coefficients of the difference equations, which can result in unstable operation of the filter because pole locations may be moved outside the unit circle.

It is generally agreed that both the direct and the direct canonic forms are the most sensitive to these errors, whereas the parallel and cascade combinations of second-order system are less sensitive. However, direct comparisons cannot be made without first deriving the scaling factors used to prevent machine overflow.

Parallel and cascade realizations contain, as the basic element, the general second-order filter depicted in Fig. C1. This stems from the fact that the system function $H(z)$ can be expressed either as a product or as a partial fraction expansion. The product form leads to the cascade realization and the partial fraction form to the parallel implementation. Since these realizations are less sensitive to noise, they are generally preferred.*

*Edwards, R., Bradley, J., and Knowles, J.B., "Comparison of Noise Performances of Programming Methods in the Realization of Digital Filters," Proceedings of Symposium on Computer Processing in Communication, Polytechnic Institute of Brooklyn, April 8-10, 1969; Polytechnic Press, Brooklyn, N.Y., 1970



$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} = \frac{Y(z)}{X(z)}$$

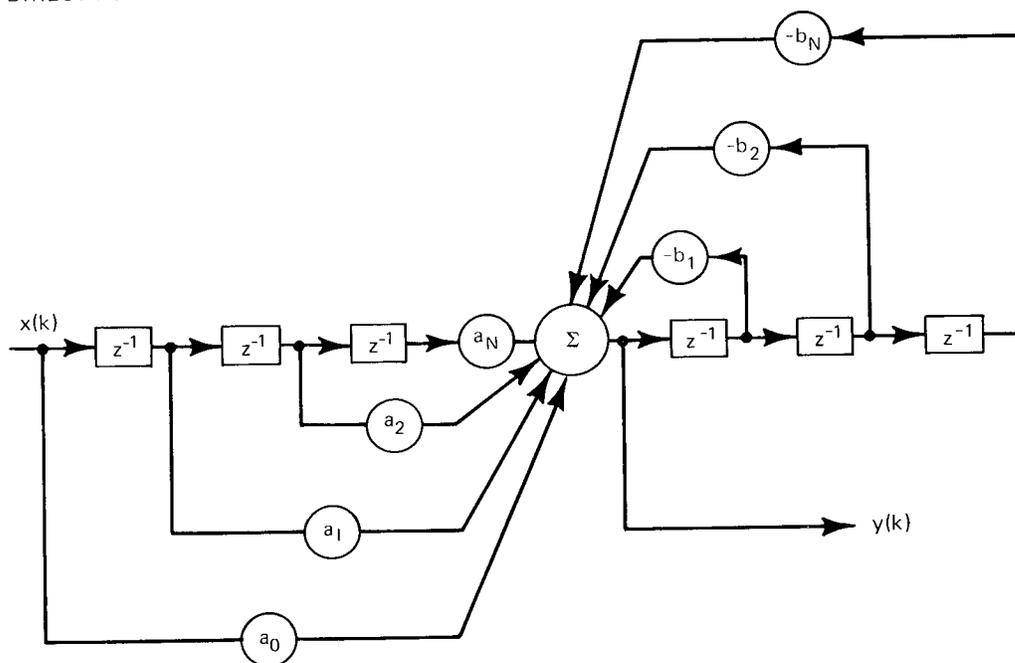
$$y(k) = a_0 x(k) + a_1 x(k-1) + a_2 x(k-2) - b_1 y(k-1) - b_2 y(k-2)$$

2 DELAYS, 5 MULTIPLICATIONS, 4 ADDITIONS

Fig. C1 — Second-order filter

In Figs. C2 through C5, the direct, direct canonic, cascade, and parallel implementations are displayed. Each figure contains the transfer function in the z -domain, the relevant difference equation, and the number of delays and arithmetic operations to be performed within a sampling interval.

DIRECT FORM



TRANSFER FUNCTION

$$H(z) = \frac{a_0 + a_1 z^{-1} + \dots + a_N z^{-N}}{1 + b_1 z^{-1} + \dots + b_N z^{-N}} = \frac{Y(z)}{X(z)}$$

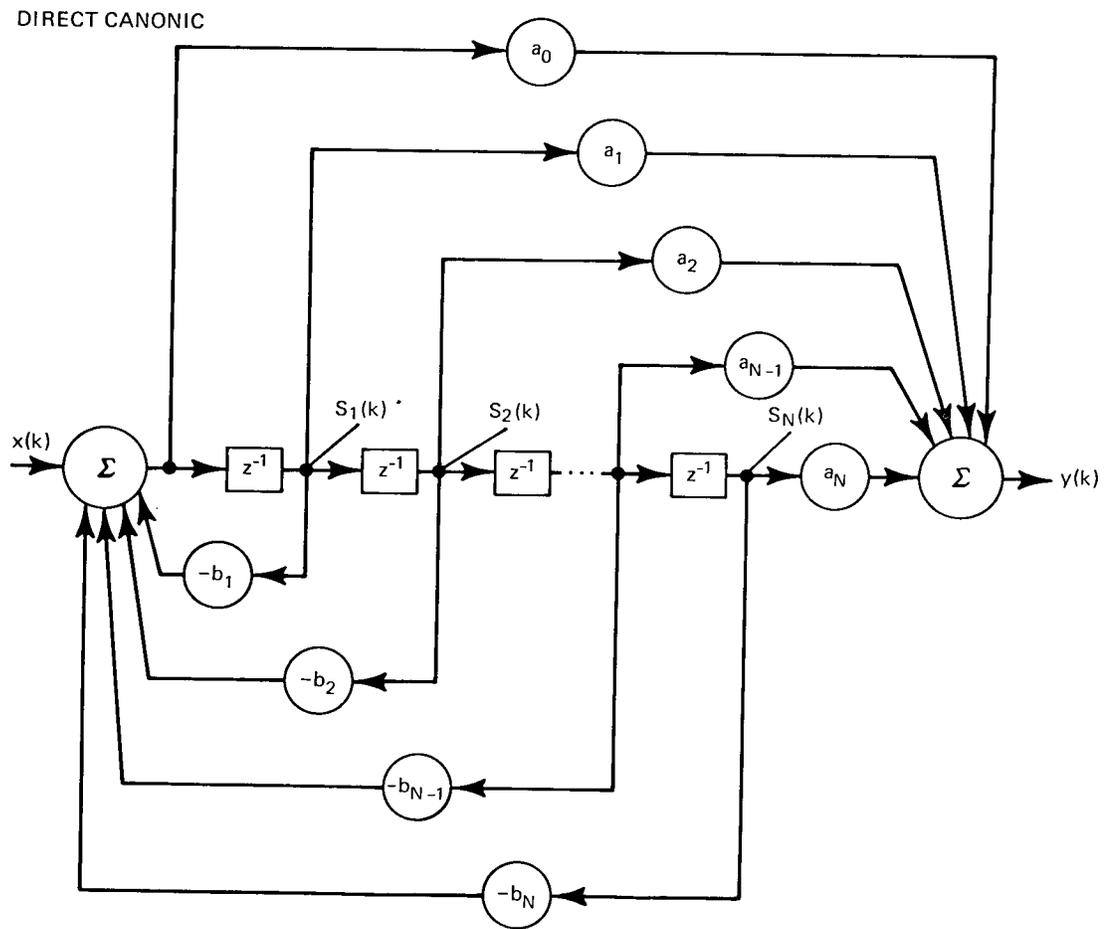
DIFFERENCE EQUATION

$$y(k) = \sum_{i=0}^N a_i x(k-i) - \sum_{i=1}^N b_i y(k-i)$$

REQUIREMENTS (PER SAMPLING INTERVAL)

2N DELAYS; 2N+1 MULTIPLICATIONS; 2N ADDITIONS

Fig. C2 — Direct implementation of second-order filter



TRANSFER FUNCTION

$$H(z) = \frac{a_0 + a_1 z^{-1} + \dots + a_N z^{-N}}{1 + b_1 z^{-1} + \dots + b_N z^{-N}} = \frac{Y(z)}{X(z)}$$

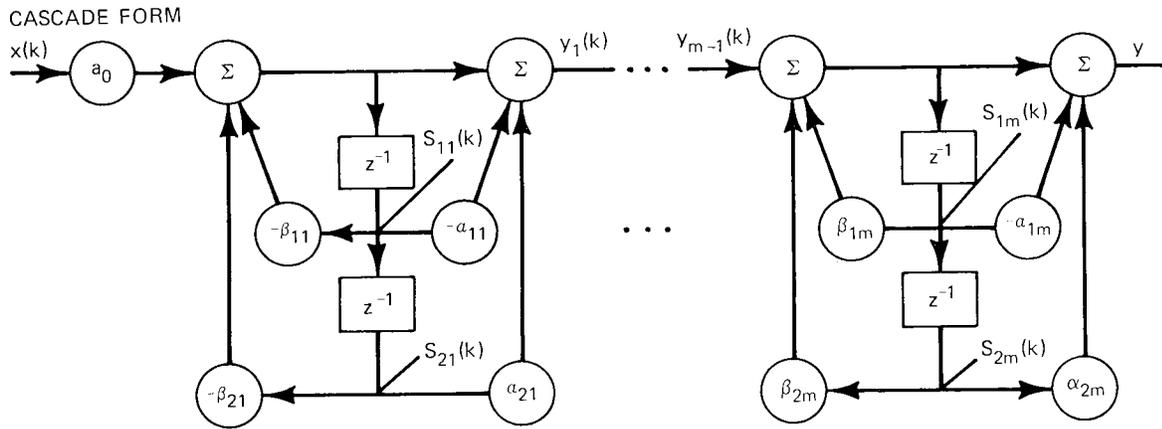
DIFFERENCE EQUATIONS

$$\begin{cases} \text{I} & S_1(k+1) = x(k) - \sum_{i=1}^N b_i S_i(k) \\ \text{II} & S_j(k+1) = S_{j-1}(k); \quad j = 2, 3, \dots, N \\ \text{III} & y(k) = a_0 x(k) + \sum_{i=1}^N (a_i - a_0 b_i) S_i(k) \\ \text{IV} & S_1(k+1) = x(k) - \sum_{i=1}^N b_i S_1(k+1-i) \end{cases}$$

REQUIREMENTS (PER SAMPLING PERIOD)

N DELAYS, 2N+1 MULTIPLICATIONS, 2N ADDITIONS

Fig. C3 — Direct canonic implementation of second-order filter



TRANSFER FUNCTION

$$H(z) = a_0 \prod_{i=1}^m \frac{1 + \alpha_{1i}z^{-1} + \alpha_{2i}z^{-2}}{1 + \beta_{1i}z^{-1} + \beta_{2i}z^{-2}} = \frac{Y(z)}{X(z)}; m = \left\lceil \frac{n+1}{2} \right\rceil$$

DIFFERENCE EQUATIONS ($a_0 = 1$)

$$y_1(k) = a_0x(k) + \alpha_{11}x(k-1) + \alpha_{21}x(k-2) - \beta_{11}y_1(k-1) - \beta_{21}y_1(k-2)$$

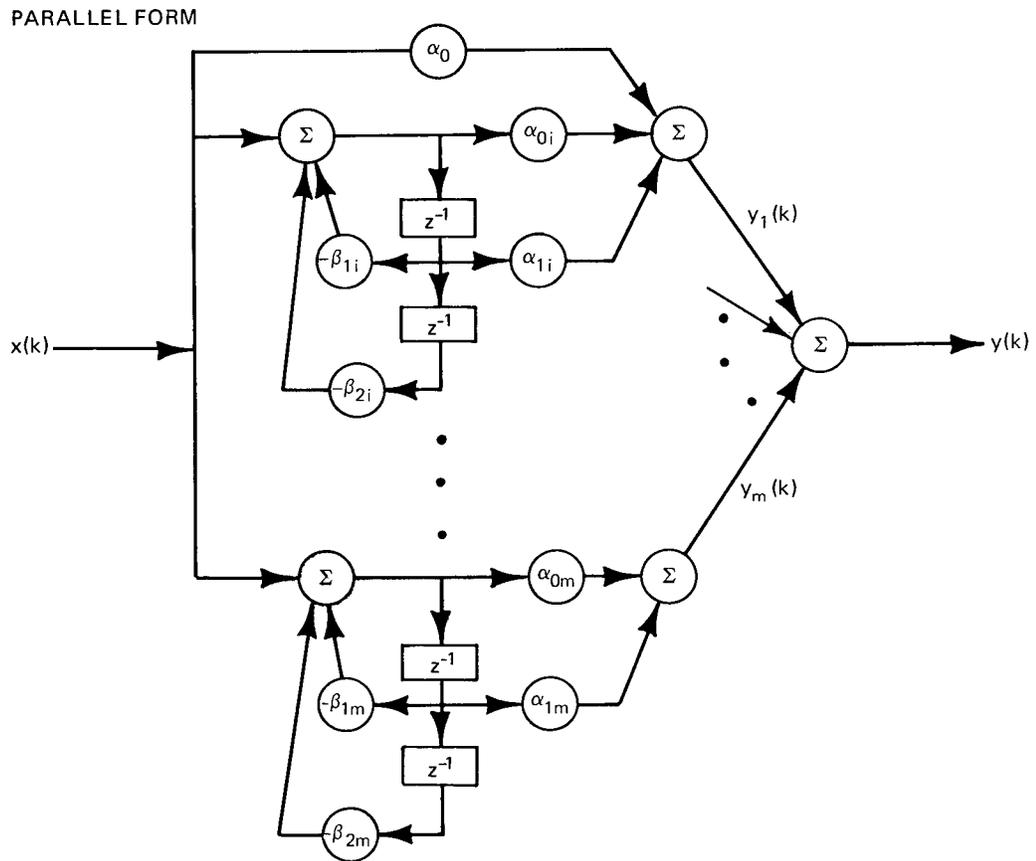
$$y_i(k) = y_{i-1}(k) + \alpha_{1i}y_{i-1}(k-1) + \alpha_{2i}y_{i-1}(k-2) - \beta_{1i}y_i(k-1) - \beta_{2i}y_i(k-2); i = 2, \dots, m$$

$$y(k) = y_m(k)$$

REQUIREMENTS

N DELAYS; 2N+1 MULTIPLICATIONS; 2N ADDITIONS

Fig. C4 — Cascade implementation of second-order filter



TRANSFER FUNCTION

$$H(z) = \alpha_0 + \sum_{i=1}^m \frac{\alpha_{0i} + \alpha_{1i}z^{-1}}{1 + \beta_{1i}z^{-1} + \beta_{2i}z^{-2}} = \frac{Y(z)}{X(z)}$$

DIFFERENCE EQUATIONS

$$y_i(k) = \alpha_{0i}x(k) + \alpha_{1i}x(k-1) - \beta_{1i}y(k-1) - \beta_{2i}y(k-2)$$

$$y(k) = \alpha_0x(k) + \sum_{i=1}^m y_i(k)$$

REQUIREMENTS

N DELAYS, 2N+1 MULTIPLICATIONS, 2N ADDITIONS

Fig. C5 — Parallel implementation of second-order filter

Appendix D

FAST FOURIER TRANSFORM — SPECTRUM GENERATION

The various realizations of digital filters operate on time domain input samples to produce time domain output samples directly. While the particular filter to be realized may be designed in the frequency domain, the actual operation of the filter is defined by its time delay elements and the coefficients which multiply the delayed samples. It is shown in Appendix B that equivalent digital filters may be realized by operating in the frequency domain and it is indicated that the Fast Fourier Transform (FFT) is the key to such a realization. This is based on the fact that the FFT algorithm enables one to compute the Fourier coefficients of the Discrete Fourier Transform (DFT) iteratively, thereby reducing the computation time considerably. For example, if the time sequence consists of $N = 2^m$ sample points, then $(1/2)N \log_2 N$ complex arithmetic operations are required to evaluate all N -associated Fourier coefficients. On the other hand, direct evaluation of the DFT requires N^2 similar complex arithmetic operations. For large N , this enormous savings in computational time renders frequency domain analysis in real time a distinct possibility.

While there are many versions of the FFT algorithm in the literature, they are all equivalent in the sense that they exploit the redundancies in information caused by the inherent periodicity of the complex exponential functions used in the DFT computations. The equations for the DFT are

$$X(k) = \sum_{n=0}^{N-1} x(n)W^{nk}, \quad k = 0, 1, \dots, N-1, \quad (\text{D1})$$

where

$X(k)$ = equally spaced spectral (Fourier) coefficients, $k = 0, 1, \dots, N-1$,

$x(n)$ = equally spaced time samples, $n = 0, 1, \dots, N-1$

$$W = e^{-j \frac{2\pi}{N}}$$

$N = 2^\nu$ = number of points in the transform (ν an integer).

It can be seen from Eq. (D1), that N^2 complex multiplications and additions are required to compute the spectrum at N frequencies, given N sample points. The FFT reduces the number of complex multiplications to $(N/2)\log_2 N$ and the number of complex additions to $N \log_2 N$.

The basic computations composing the FFT algorithm can best be described by referring to Fig. D1, a representation of the FFT for $\nu = 4$, $N = 2^4 = 16$. Let us assume

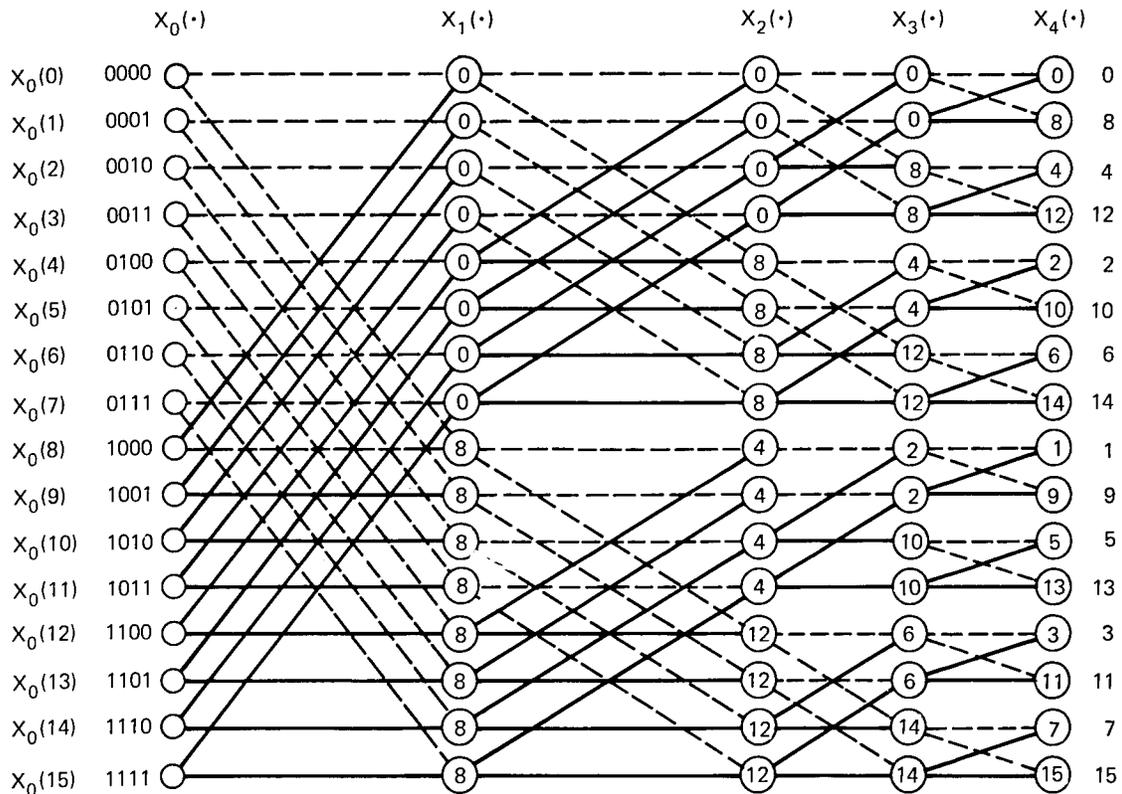


Fig. D1 — FFT (in-place algorithm)

that the original N data points are represented as elements of the vector $X_0(j)$, $j = 0, 1, \dots, N - 1$. The elements of this vector are in turn represented by the first column of nodes in the figure. The computation proceeds by calculating the next vector column of nodes $X_1(i)$ from $X_0(\cdot)$. The vector $X_2(\cdot)$ is then calculated from $X_1(\cdot)$, and so on for $X_3(\cdot)$, ..., $X_{\log_2 N}(\cdot)$. The resulting vector of Fourier coefficients will appear at the last iteration. However, the order in which the elements of the last vector occur must be changed to reflect the proper ordering of the coefficients. This is easily done by reversing the order of the binary representation of the j th spectral point.

To display the actual arithmetic operations involved, we pick data points $X_0(0)$ and $X_0(8)$ from the first column and, from these, compute $X_1(0)$ and $X_1(8)$ of the next column. This computation is depicted in Fig. D2.

The solid arrow brings a complex quantity from the node $X_0(8)$ and multiplies it by W^q , where q is the integer in the circle; in this case, $q \in \{0, 8\}$. These complex products are added to the complex quantity brought via the dashed arrow from the node $X_0(0)$. Thus, the two nodes in vector $X_1(\cdot)$ imply the computation

$$X_1(0) = X_0(0) + X_0(8) W^0$$

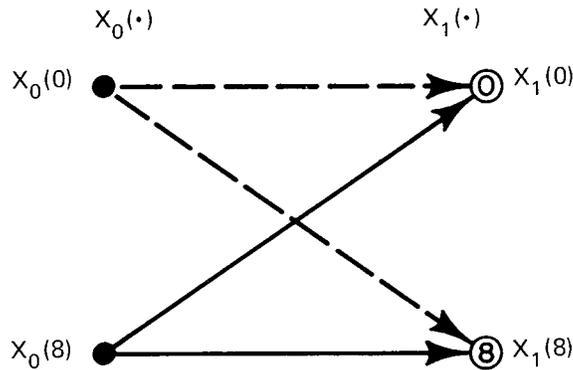


Fig. D2 — Actual arithmetic operations of Fig. D1

$$X_1(8) = X_0(0) + X_0(8) W^8.$$

In this manner, all the elements of the vector $X_1(\cdot)$ are computed from elements of the vector $X_0(\cdot)$. As was previously mentioned, vector $X_2(\cdot)$ is computed from vector $X_1(\cdot)$ in a similar manner and so on for the remainder of the vectors. It remains to be shown how the integers q , in the circles, are computed. Consider the number in the circle of the j th node of the m th vector. This number can be found by representing j in its binary form, shifting it by $(\log_2 N - m)$ places to the right with zero fill on the left, and reversing the order of the bits. For example, $X_3(13)$ has address $13 = 1101$. Scaling by $4 - 3 = 1$ gives (0110) and reversing the order of the bits gives $(0110) = 6$. This is seen to be the number in the appropriate node in Fig. D1.

An examination of the figure reveals that there are always two nodes in vector $m + 1$ which are affected by the same pair of nodes in vector m , and that no other nodes in vector $m + 1$ are affected by this pair of nodes in vector m . The two nodes in vector $m + 1$ and the two nodes in vector m form a rectangle. The computation represented by this rectangle is generally referred to as the FFT “butterfly” and can be written as

$$X_{m+1}(i) = X_m(i) + X_m(j)W^{q1}$$

$$X_{m+1}(j) = X_m(i) + X_m(j)W^{q2}.$$

It should be noted that the pair of nodes in the $(m + 1)$ th vector have solid arrows coming from the same node in the m th array and that the integers in the circles differ by $N/2$. Since $W^{q+N/2} = W^q W^{N/2} = W^q \exp(-j2\pi/N) \cdot N/2 = W^q(-1) = -W^q$, the multiplications required for these two nodes for the $(m + 1)$ th vector are negatives of each other. This implies that the computations for both nodes may be performed together, thus saving half the required multiplications. For this situation, the FFT “butterfly” can be written as

$$X_{m+1}(i) = X_m(i) + X_m(j)W^q$$

$$X_{m+1}(j) = X_m(i) - X_m(j)W^q.$$

It should now be apparent that the FFT algorithm can be computed by repeated use of this basic computational kernel. This of course, assumes that the location of data and coefficients to be accessed at any given time is known or can be computed. This problem of address generation can be considered the domain of an external control or address unit.