NASA CR-
141887

Systems Analysis Of The Space Shuttle

Final Report

April 17, 1974 - April 18, 1975

National Aeronautics and Space Administration

Johnson Space Center - Houston

under

NASA Contract NAS 9 - 13940

Department of Electrical Engineering

(NASA-CR-141887) SYSTEMS ANALYSIS OF THE SPACE SHUTTLE Final Report, 17 Apr. 1974 - 18 Apr. 1975 (City Coll. of the City Univ. of New York.) 196 p HC $7.00 CSCL 22B

N75-25997

Unclas
G3/18 27302

Donald L. Schilling Se Jeung Oh Fred Thau

Professors of Electrical Engineering; Co-Principal Investigators

THE CITY COLLEGE OF
THE CITY UNIVERSITY of NEW YORK

Systems Analysis Of The Space Shuttle

Final Report

April 18, 1974 - April 17, 1975

National Aeronautics and Space Administration
Johnson Space Center - Houston
under
NASA Contract NAS 9 - 13940

Department of Electrical Engineering

Donald L. Schilling Se Jeung Oh Fred Thau
Professors of Electrical Engineering; Co-Principal Investigators

Introduction

This report consists of three aspects of the Shuttle Systems Analysis problem: Communications, Computers and Power Distribution. Part I summarizes the Communications research directed by Professor D. L. Schilling; Part II summarizes the Computer research directed by Professor S. J. Oh and Part III summarizes the research on Power Distribution directed by Professor F. Thau.

In addition to the three coprincipal investigators this contract partially supported Dr. C. B. Park and the following Doctoral Students: T. Appelewicz, C. S. Chuang, E. Feria, R. Lei, J. LoCicero, G. S. Mersten, V. Rao, N. Scheinberg, D. Ucci, L. Weiss and C. Zeigler.

Several papers, summarizing particular aspects of the research conducted under this contract, were presented at conferences, and the paper, "Video Encoding Using an Adaptive Digital Delta Modulator" by L. Weiss, I. Paz, and D. L. Schilling has been accepted for publication as a Technical paper in the IEEE Transactions on Communications.

I. 1. Digital Processing of Video Signals

A. High Speed Frame to Frame Delta Modulation

Successive frames of a motion picture contain redundant information. The amount of redundancy depends upon the degree of change in the scene from frame to frame. We are currently building a high speed delta modulator which encodes television signals in such a way as to use the frame to frame redundancy of motion pictures to reduce the bit rate required to transmit a television signal.

Figure 1 shows the basic encoding scheme. Each large square represents a successive frame of a motion picture. Each dot on a frame represents a pixel (picture element). Note that each pixel has associated with it a Delta Modulator (ΔMOD) that follows the same pixel through successive frames. Thus the ΔMOD in the upper most left hand corner always encodes the pixel in the upper most left hand corner for every frame.

Bit rate compression is achieved in the following manner. From previous studies of delta modulators it has been observed that high sampling rates are required for a delta modulator to accurately encode rapidly changing signals and low sampling rates may be used on slowly varying signals. The usual method of encoding a picture by sampling successive pixels within the same frame results in rapidly changing signals which require a high delta modulator bit rate (typically, 6 bits per pixel). With the frame to frame encoding technique of Fig. 1 each delta modulator is associated with its own pixel. In general the pixel value will change slowly from frame to frame, allowing the delta modulator to accurately encode the slowly changing pixel value at a low bit rate. Using this scheme motion pictures will be encoded at 1 bit per pixel.

At first thought, hardware implementation of Fig. 1 may seem impossible since, at 1 delta modulator per pixel, a typical 200,000 pixels/frame 3 MHz bandwidth TV system would have to contain 200,000 delta modulators each operating at a 6MHz sampling rate. Fortunately only 1 6MHz delta modulator need be used to implement Fig. 1; however, this single delta modulator will have to contain enough shift register type memory to store an entire picture frame.

The reason that 1 delta modulator can replace all the delta modulators of Fig. 1 with no increase in operating speed is simple. If Fig. 1 were implemented as shown all 200,000 pixels could be encoded, decoded and displayed in parrallel, but a real time television system requires only one pixel at a time in serial. The retention time of the eye and screen give the appearance of a full picture. We may take advantage of this by starting a single delta modulator at the upper most left hand corner of a frame, let it encode that pixel based upon its previous estimate of the pixel from the previous frame, transmit a bit (E_k), store its new estimate (X_k) for that pixel, and then repeat the process for the next adjacent pixel in the same frame. The delta modulator will continue to encode, and transmit for each adjacent pixel in turn, until the delta modulator has been multiplexed through the entire frame (typically 1/30 sec). Then the delta modulator will return to the first pixel and repeat the process for the next frame.

Figure 2 shows the hardware implementation of the high speed video delta modulator. The arithmetic and logic portions are being assembled out of Schottky TTL devices. The 200,000 bit shift registers will either be assembled out of 4K dynamic Random Access Memories (RAM's) such as the Intel 2107 or Intel's new 16K Charge Coupled Device (CCD) serial memories. The CCD devices are superior to the RAM's but the RAM's may have to be used if the CCD devices cannot be obtained. If the 4K RAM's are used, special multiplexing and demultiplexing logic will have to be used with them to make the relatively slow 4K RAM's appear as high speed shift registers. The logic configuration for the 4K RAM's converted into shift registers is shown in Fig. 3. The addressing logic (not shown) consists of 8 twelve-stage counters counting the pulses of lines A, B, C, D, $\overline{A}$, $\overline{B}$, $\overline{C}$, and $\overline{D}$. Each counter output feeds the address lines of a 4K RAM.

Construction of the high speed delta modulator without the large shift register memory is complete. Testing of the delta modulator has just begun.

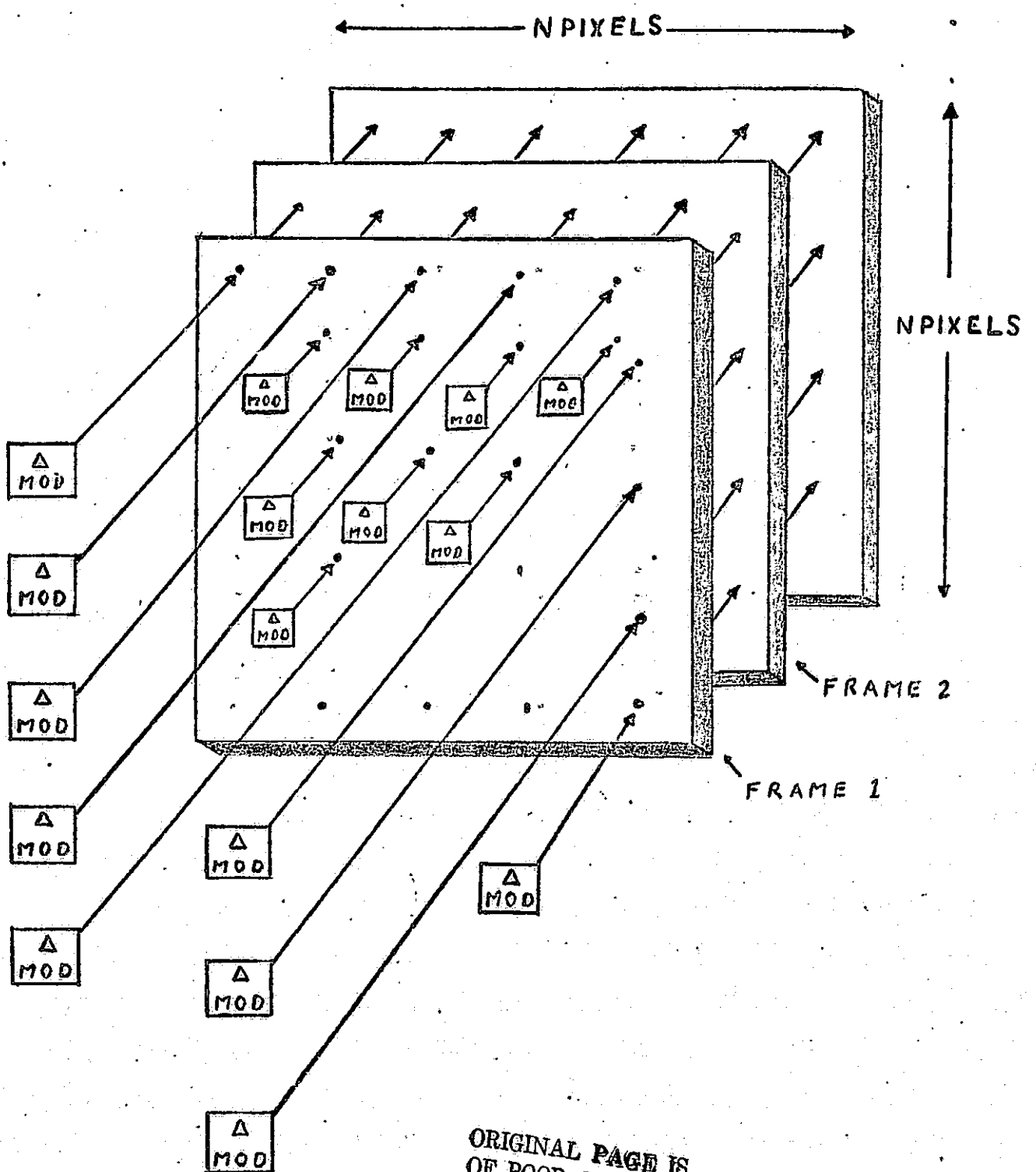
B. Computer Processor for Color Video Signals

We are assembling a computer processor for color video signals. The processor will enable us to feed color video signals from a color TV camera into a computer. The computer will be able to process the video signals and display a picture on a video monitor.

The basic system is shown in Fig. 4. An image originatinating in a 35 mm slide projector is focused on a color TV camera. The TV camera has three video outputs, one for the red component of the image, one for the green component of the image, and the third for the blue component of the image. Each of the three video outputs is put into a scan converter which stores a single picture frame in its electrostatic storage tube. Upon receiving commands from the computer the scan converter transfers the stored picture into the computer. The rate at which the scan converter sends information to the computer is controlled by the computer. The computer can also randomly access the information stored in the scan converter, i.e., the computer can read the picture elements into its memory in whatever order the computer program requires. After the computer processes the picture, the picture is read out of the computer and stored on three scan converters, one for the red component, one for the green component and one for the blue component of the picture. Again the computer controls the rate and order in which the picture elements are stored in the scan converter. The scan converters are read out onto a TV monitor which displays the picture. Since reading of the scan converters is nondestructive and occurs at 30 frames per second, the user of the system sees the picture frame frozen on the TV monitor. The picture can be kept frozen for several minutes and then photographed for a permanent record.

When the system described above is completed we will have a very flexable research tool. By simply changing computer programs the system can perform image enhancement, bandwidth compression or any other type of image processing one might want to perform on color pictures.

FIG. 1 FRAME TO FRAME ENCODING



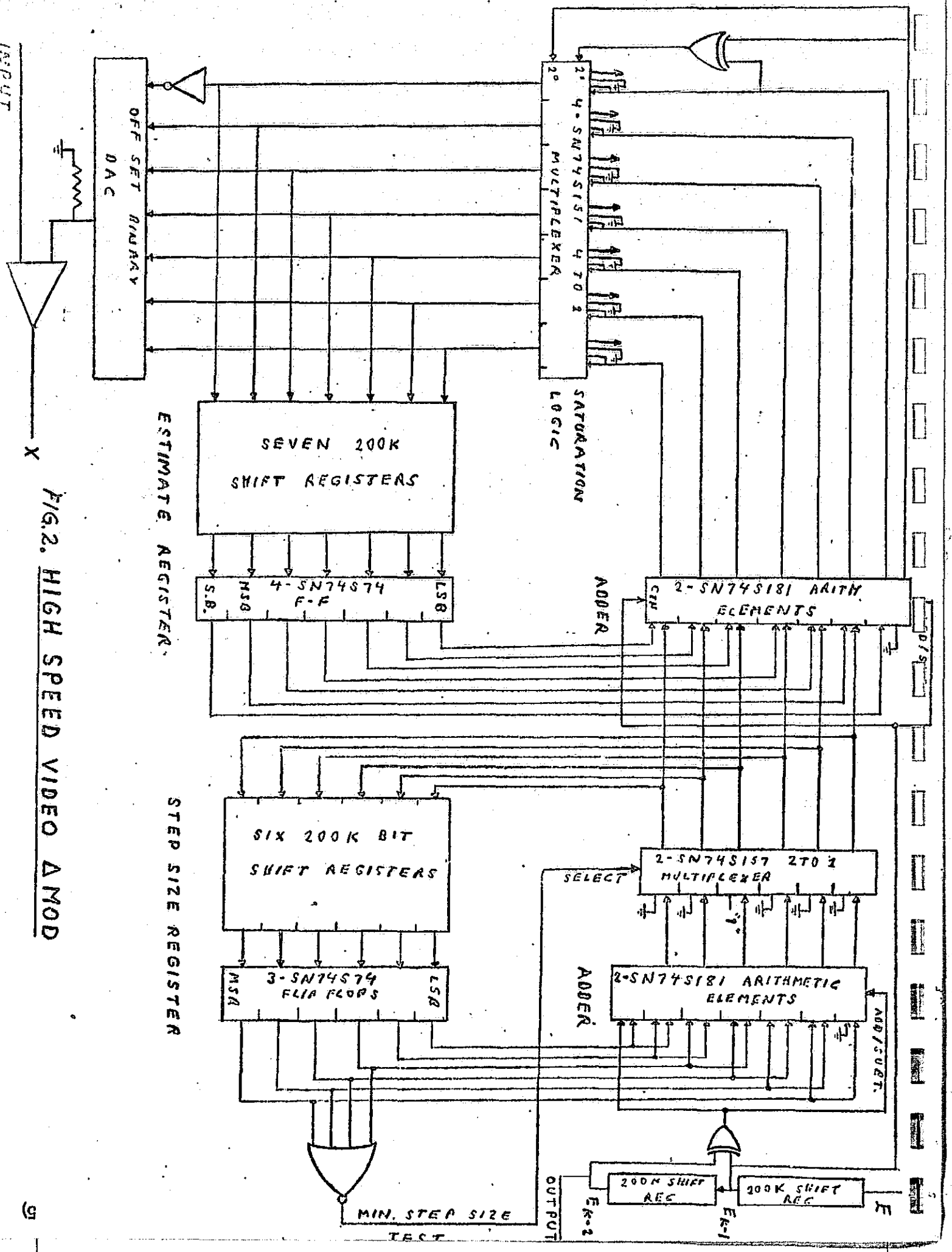
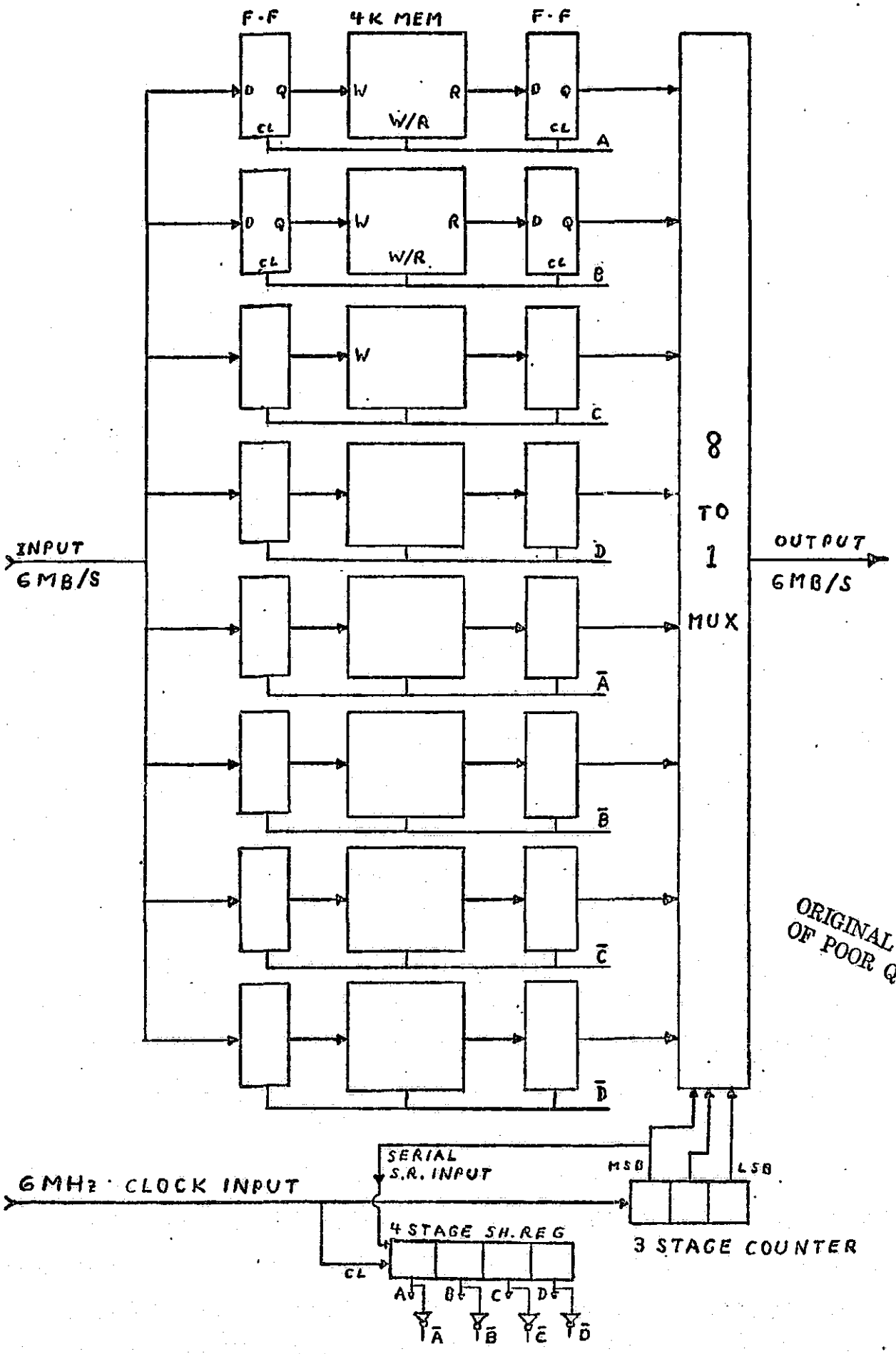


FIG.2. HIGH SPEED VIDEO ΔMOD

FIG. 3. 32K SHIFT REGISTER



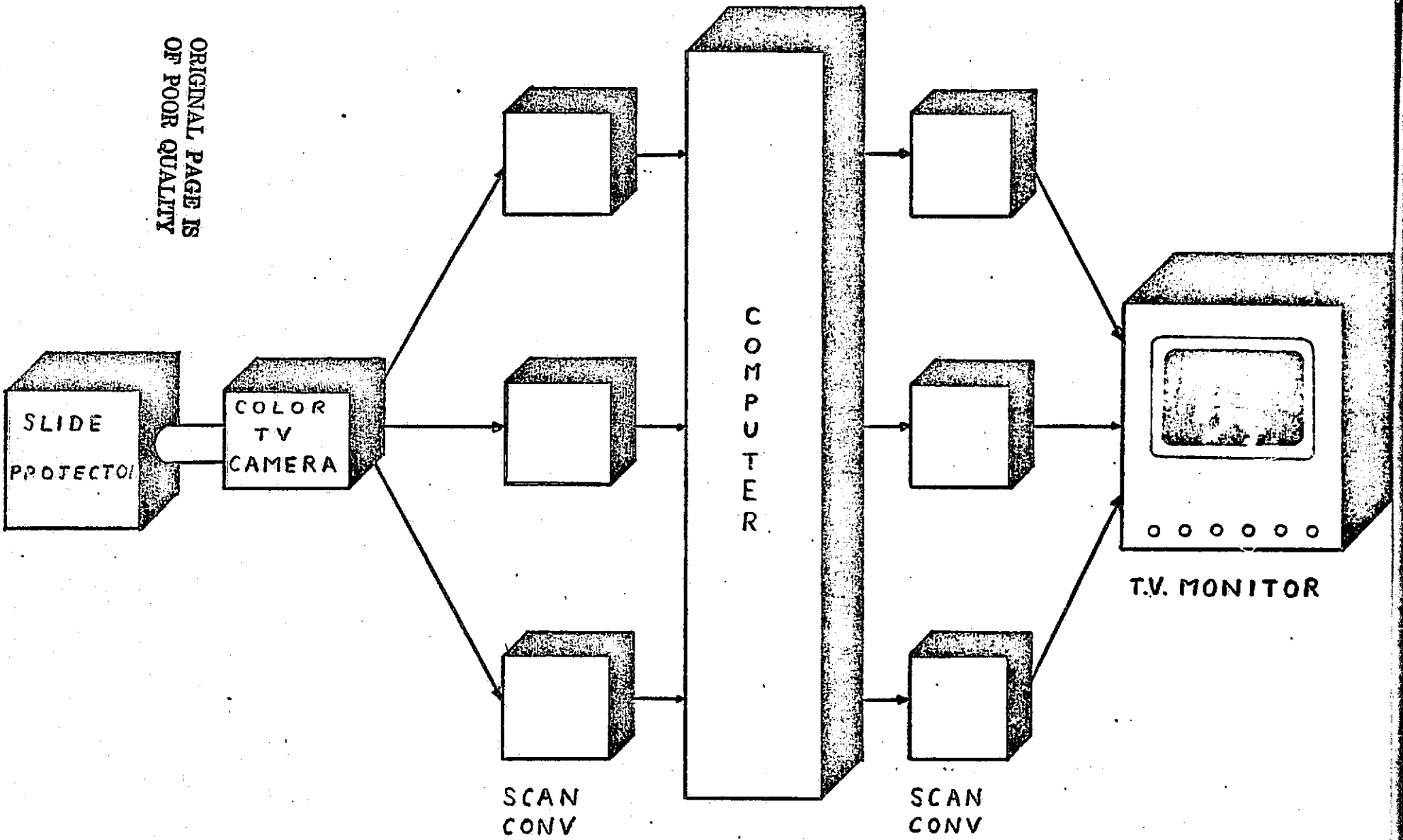


FIG. 4. COMPUTER PROCESSOR FOR COLOR VIDEO SIGNALS

I. 2. Entropy Encoding of a Delta Modulator Output for Bandwidth Compression

A. Abstract

The output bit stream of the Abate Mode Adaptive Delta Modulator with voice source input has been studied. It is found that about 50 per cent of the bits are in a steady state sequence, "1100", when the input voltages are relatively low. Hence, the output bit stream can be coded such that the final bit rate and bandwidth are reduced. Two kinds of coding techniques, the Huffman coding and run length coding, have been considered. The Huffman coding is found to be superior to the run length coding. However, the direct Huffman coding algorithm, which is an information preserving code, does not result in a high data compression ratio. The modified Huffman coding scheme, an information degrading process, is currently being studied.

B. Introduction

1. Historical Background

Source-encoding or data compression techniques are used to reduce the volume of data generated. As a result, the bandwidth of the channel over which the data is transmitted is reduced. There are two kinds of data compression techniques: information preserving and information degrading. When the information preserving process is used for coding, all of the information presented before coding can be regenerated. Wheras, when the information degrading process is used, only part of the original information can be regenerated. However, for the information degrading process, a fidelity criterion can be chosen such that certain bits of information can be ignored in order to obtain a better result in data compression and still be able to retain a relatively high intelligibility.

Source-encoding systems can be classified as either "variable to block" or "block to variable". A system accepting a variable-length sequence of digits from the source and generating a fixed-length block code word is called a "variable to block" encoder. A system accepting a fix-length block of digits

from the source and generating a variable-length block code word is called a "block to variable" encoder.

A binary memoryless system generating a statistically independent source alphabet, (0, 1), can be characterized by the probability, P and (1 - P). P is defined as the probability of a digit "1" being emitted, and (1 - P) is defined as the probability of a digit "0" being emitted. The entropy of the source is defined as

$$H'(p) = -P\log_2 P - (1 - P)\log_2(1 - P), \tag{1}$$

and is a measure of the information content of the source.

If a binary memoryless digital system has n possible values, $(x_1, x_2, \ldots, x_n)$, to be encoded with the probabilities, $(P_1, P_2, \ldots, P_n)$ respectively. Then, the entropy of the source is

$$H(p) = -\sum_{i=1}^{n} P_i \log_2 P_i. \tag{2}$$

For example, $x_1 = 00$, $x_2 = 01$, $x_3 = 10$, $x_4 = 11$, and $P_1 = 1/8$, $P_2 = 1/8$, $P_3 = 1/4$, $P_4 = 1/2$, Then using Eq. (2),

$$H(p) = 1.75$$

In the above example, code words are not assigned to single digits but to blocks of digits. In other words, the encoder waits for the source to produce a block of two digits, then assigns a code word to the two-digit block.

According to Shannon's noiseless coding theorem (1), the minimum average code-word length is equal to H(p). The theorem is stated as follows: Given a random variable, x, in a binary system with entropy H(x), there exists a code for x whose average code-word length, $\bar{n}$, satisfies

$$H(x) \leq \bar{n} < H(x) + 1. \qquad (3)$$

If a long source sequence of N blocks is the input to an optimal source encoder, the output will be a sequence of NH(p) bits. In the previous example, since each block consists of 2 digits, the total length of the input sequence is 2N bits. The output sequence length is 1.75N bits. The maximum data compression ratio is, then,

$$C = \frac{2N}{1.75N} = 1.143.$$

The maximum data compression ratio depends on the statistics of the encoding source. In other words, once the probabilities of occurrance of the different patterns of the source digits are known, the maximum data compression ratio is theoretically determined.

2. Huffman Coding

The first optimal coding scheme developed was the Huffman coding (2). This is a "block to variable" technique. When Huffman coding is applied to a binary memoryless source, the source sequence has to be broken up into blocks of N bits long. Then, each block contains one of $M = 2^N$ possible messages. The least probably message is assigned a code word containing the longest sequence of bits. While the more probable messages are assigned code words of shorter length.

3. Run Length Coding

Run length coding (3, 4) technique is "variable to block". It works well when probability P is either very low or very high. In the case of low probability P, the number of consecutive zeros are counted and transmitted as a block of binary digits. In the case of high probability P, the number of consecutive ones are counted and transmitted. The optimal compression ratio as a function of P is shown in Fig. 1.

C. Theoretical Considerations of Source Encoding of an Abate Mode Adaptive Delta Modulator

For a DC input voltage, the Abate Mode Adaptive Delta Modulator has a steady state output pattern as shown in Fig. 2. It is known that during speeches or telephone conversations, there are many short period of pauses. In the noiseless case, this pauses will be considered as DC voltage by the delta modulator and a stream of steady state pattern will be generated. Also when signal voltage and signal frequencies are low, the output bit stream of Adaptive Delta Modulator will be viewed as segments of steady state patterns corrupted by some noise bits, as shown in Fig. 3.

The length and frequency of the steady state pattern depend upon factors, such as input signal voltage, minimum step size of the Adaptive Delta Modulator, sampling rate, and signal bandwidth. The run length coding scheme will be used if the steady state pattern is very long. When the steady state pattern is not very long but its probability of occurrance out of the entire bit stream is very high, the Huffman coding scheme should be used.

The steady state pattern is as follows:

.....1100110011001100.....

If a block of 4 is chosen for the Huffman code, the steady state pattern can be broken up into four different steady state sequences, 1100, 1001, 0110, and 0011. If a block of 6 is chosen, then the sequence will be 110011, 100110, 001100, 011001.

In the next section, the experimental results will show that blocks of 4 are better than blocks of 6 in the sense that the maximum data compression ratio is higher for blocks of 4. The reason is obviously because the steady state pattern is a repetition of a 4 bit sequence. Higher bit blocks will drastically increase the number of sequences to be coded thus decreasing the ratio of steady state sequences to other sequences.

D. Summary of Results

1. Experimental Results

In order to determine whether the run length code or the Huffman code is better for the Abate Mode Adaptive Delta Modulator, we must know how

long a run of the steady state pattern is. The average run length of the steady state pattern for this experiment was calculated using a PDP-8 computer. Figure 4 shows the procedure employed to obtain this statistics. Since it is desired to have the computer simulated Adaptive Delta Modulator work in real time, the sampling frequency is, therefore, determined by the length of the computer program. In other words, after each sample value have been transferred to the computer from the A/D converter, the computer will go through the program's process of generation the output bit and analyzing the statistics before it goes back to take another sample. Since the PDP-8 is not a high speed computer, the fastest sampling rate we can manage to obtain is about 5KHz. However, a sampling rate of 32K is standard for Delta Modulators. Therefore, a process of slowing down the speed of the source tape is necessary. By alternatively playing back and recording at different speed with two high quality Ampex tape recorder, a voice tape of $1/8^{th}$ of the normal speed was successfully made. This $1/8^{th}$ speed voice tape was originally part of a magnetic open reel tape recorded by the Ampex Company with Ed Begley reading the story of Mark Twain. This tape was used as the voice source for all the experiments in this report.

All of the programs are written in proper length such that the sampling rate of 4KHz becomes equivalent to 32KHz at normal speed. The digital estimated signal, $\hat{x}(k)$, of the simulated Adaptive Delta Modulator was first fed into a D/A converter; then the analog estimated signal $\hat{x}(t)$ was generated and recorded into a tape recorder. This magnetic tape then went through the reverse process of slowing down, previously described, to return it to the normal speed. Finally, this tape's intelligibility was determined by playing it back from a recorder. While the estimated signal $\hat{x}(t)$ was being recorded, the computer was doing statistical computations and storing the results in its memory. The results are compiled for a total of 20 minutes, a time interval which we believe is long enough to collect unbiased statistics.

The average run length of the steady state pattern is listed in Table 1.

Note that the average run length is not very long. A run of 2 or 3 is definitely too short for a run length code; therefore, the use of a run length code is ruled out.

The output bit sequence of the Adaptive Delta Modulator is intrinsicaly dependent on the slope of the input signal. Increasing the minimum step size, S_0, will have the same effect as decreasing the input signal level. The following factors were chosen to perform the experiment:

(1) Voice signals are bandlimited to within the range of 300 Hz to 2.5KHz.

(2) Sampling frequency f_s, is adjusted to 32khz. Some other different sampling frequencies are also used for comparison purpose.

(3) The minimum step size, S_0 is set to be 40 mv.

(4) The maximum peak to peak input voltage level is varied in steps from $1\ v_{p-p}$ to $20\ v_{p-p}$.

In order to apply the Huffman coding technique to the output bit stream of the Adaptive Delta Modulator, a computer program is written to search for the steady state sequences. When blocks of 4 bits are used to form the code word, the '1100' sequence was being searched for from the entire output bit stream. Whereas, when blocks of 6 bits are used as in the Huffman coding block, the '110011' sequence was being searched for. The results are shown in Table 2 and Table 3 respectively. It is clear that blocks of 4 are better for coding than blocks of 6, and low input voltage level have shorter average code word length. It is also found that in the case of $f_s = 32KHz$, $S_0 = 40$ mv, the input voltage level of 2 volts peak-to-peak or higher results in very high intellegibility. An input voltage level of 1 volt peak-to-peak or lower results in poor performance due to unfiltered in-band gramular noise.

By referring to Table 2, it can be seen that at $2\ v_{p-p}$ input signal level, 53 percent of the total bit stream are in the steady state sequence and yet, we still maintain high intelligibility. These results encourage us to do further studies on the probability distributions of 16 different sequences. This data is listed in Table 4. In order to give a better visual comparison,

they are plotted in Fig. 5 through Fig. 9. From Fig. 5 and Fig. 6, it can be seen that the probability of occurrance of steady state sequences (0011, 0110, 1001, 1100) are much higher than the others. This fulfills the fundamental requirements of the Huffman coding scheme. It also can be seen, by referring to Fig. 7 and Fig. 8, that the probability of the steady state sequences are not exceedingly higher than the others when the input voltage level is increased. In the extreme case of 20 V_{p-p}, as shown in Fig. 9, the probability of steady state sequences are even less than some other sequences. Hence, a number of general conclusions can be drawn from these results:

(1) Blocks of 4 should be used to construct the Huffman code.

(2) As far as signal to noise ratio is concerned, the performance of the Delta Modulator is very good when the input voltage is 2 V_{p-p} or higher.

(3) At low input voltage, the resulting probability distributions are most suitable for coding.

(4) From statements (2) and (3) above, it is clear that the probability distribution for 2 V_{p-p} input voltage, as referred to in Fig. 6, should be used for the coding scheme.

2. Information Preserving Coding

The entropy of the coding source is a measure of the information content of the source and is the lower bound of the code word length. Table 5 shows the entropy of the source at different input voltages. It was found that at an input voltage of 2 V_{p-p} the minimum obtainable code word length is 3.56, which gives very little data compression. The actual code word length computed after the code has been constructed, as shown in Fig 10, is 3.598. The data compression ratio is $4/3.598 = 1.112$, which is certainly not satisfactory. The reasons are twofold:

(1) The four most probably steady state sequences are coded separately, consequently, three digits are needed to code each sequence. However, if these 4 most probably sequences can be represented by one sequence with the probability 4 times higher, then a single digit is enough to code this sequence and a tremendous savings in code word length can be expected.

(2) As it can be seen from Fig. 6 and Fig. 10, the probabilities of sequences,

'0000', '1111', '0101', and '1010' are much lower than the other sequences. Thus, they can be neglected without significantly degrading the performance. The 6 digit code word will not exist if these 4 sequences are neglected. Therefore, a further shortening in code word length can be again expected.

3. Information Degrading Coding

The probability of occurrence of the "0000" sequences is very low as shown in Fig. 6. The encoder can be designed to receive this sequence and to send out the more probable sequence, "0100". Similarly, the "1111", "1010" and "0101" sequences can be converted into the "1011" or "0100" sequences as shown in Fig. 11. Futhermore, if a sliding block coding algorithm is used, there would be only one steady state sequence "1100", with probability higher than 50%. If we assume that the 4 steady state sequences could be represented by one sequence, the final total number of sequences to be coded would then be reduced to 9. The average code word length would thus be reduced to 2.228, as shown in Fig. 11. The performance of this information degrading process is not yet determined. The complete algorithm describing this process is currently being researched. However, the theoretically computed value, 2.228, gives us some valuable indications that the coding of the Delta Modulator for bandwidth compression is feasible.

E. Conclusions

A thorough study of the statistics of the output bit stream of the Abate Mode Adaptive Delta Modulator has been performed. A few conclusions can be drawn; they are stated as follows:

(1) The "block-to-variable" coding scheme should be used for the Adaptive Delta Modulator. The optimum block length is four bits.

(2) The direct Huffman coding technique does not result in a high data compression ratio. Hence, a modified information-nonpreserving coding scheme based on the Huffman method should be derived to meet the theoretical bound.

Further studies are still to be done in this area. Some theoretical calculations for the information degrading scheme is needed. Transformation methods, such as, Fourier transformation, Hadamard transformation, Karhunen-Loeve transformation, and others, are alternate approaches to reduce the final output bit rate and channel bandwidth. The video signal source using the Song Video Mode Adaptive Delta Modulator has not been analized at the time this report is written. Further expansion from an audio source to a video source is the next planned research.

References

(1) C: E. Shannon and W. Weaver, "The Mathematical Theory of Communication" University of Illinois Press, Urbana, 1949.

(2) Robert Ash, "Information Theory" Interscience publishers, 1965.

(3) H. Meyr , H. G. Rosdolsky and T. T. Huang "Optimum Run Length Codes" IEEE Transactions on Communications, Vol. COM-22, No. 6, pp 826-835, June 1974.

(4) J. Pieter M. Schalkwijk "An Algorithm for Source Coding" IEEE Transactions on Information Theory, Vol, IT-18, No. 3 pp 395-399, May 1972.

Table 1

Average Number of Runs of Steady State Sequences at Different Input Voltages

Sequences	Input Voltages				
	1 V_{p-p}	2 V_{p-p}	3 V_{p-p}	4 V_{p-p}	6 V_{p-p}
0011	4.81	2.96	1.91	1.86	1.69
0110	4.13	2.25	2.24	1.73	1.53
1001	2.19	1.94	1.86	1.65	1.50
1100	2.56	2.21	2.36	1.97	1.74
Averge:	3.42	2.34	2.09	1.80	1.62

Table 2

The Number of "1100" Sequences Encountered by Shifting Along the Adaptive Delta Modulator Output Bit Stream (In Unites of Ten Thousands Of Bits)

	Input Voltages				
	1 V_{p-p}	2 V_{p-p}	3 V_{p-p}	4 V_{p-p}	6 V_{p-p}
Number of "1100" Sequence	89	66	55	50	41
Total Number Of Bits	500	500	500	500	500
Ratio of Bits in "1100" Sequence to The Total Number of Bits	71%	53%	44%	40%	33%

Table 3

The Number of "110011" Sequence Being Searched (Blocks of 6)

	Input Voltages				
	1 V_{p-p}	2 V_{p-p}	3 V_{p-p}	4 V_{p-p}	6 V_{p-p}
Number of "110011" Sequence	40	27	22	20	16
Total Number of bits	500	500	500	500	500
Ratio of Bits in "110011" Sequence to The Total Number of Bits	48%	32%	26%	24%	19%

Table 4

Probabilities of 16 Possible Sequences at Different Input Voltages

	Input Voltages					
Sequences	1 V_{p-p}	2 V_{p-p}	3 V_{p-p}	4 V_{p-p}	6 V_{p-p}	20 V_{p-p}
0000	0.006	0.020	0.022	0.028	0.033	0.035
0001	0.035	0.050	0.033	0.037	0.040	0.070
0010	0.018	0.042	0.077	0.081	0.080	0.088
0011	0.245	0.162	0.092	0.090	0.080	0.053
0100	0.047	0.068	0.048	0.050	0.054	0.105
0101	0.004	0.011	0.018	0.022	0.027	0.018
0110	0.243	0.158	0.143	0.128	0.111	0.088
0111	0.008	0.024	0.064	0.066	0.065	0.035
1000	0.008	0.022	0.064	0.061	0.084	0.053
1001	0.133	0.132	0.118	0.105	0.100	0.088
1010	0.004	0.011	0.018	0.024	0.027	0.018
1011	0.063	0.072	0.050	0.055	0.058	0.105
1100	0.125	0.116	0.114	0.103	0.088	0.053
1101	0.020	0.044	0.083	0.083	0.081	0.088
1110	0.037	0.050	0.037	0.039	0.042	0.070
1111	0.004	0.018	0.020	0.026	0.031	0.035

Table 5

Entropies of the Output Bit Stream of This Abate Mode Adaptive Delta Modulator When Blocks of 4 Are being Used For Coding

	Input Voltages				
	1 V_{p-p}	2 V_{p-p}	3 V_{p-p}	4 V_{p-p}	6 V_{p-p}
Entropy	3.03	3.56	3.78	3.82	3.86

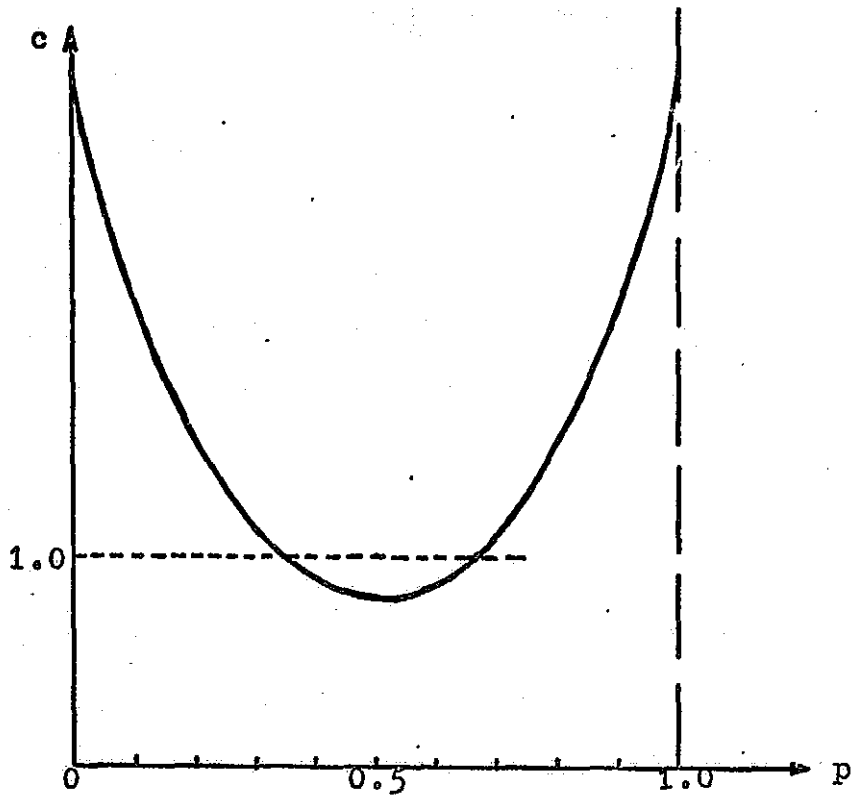


Figure 1. Optimal compression ratio as a function of p

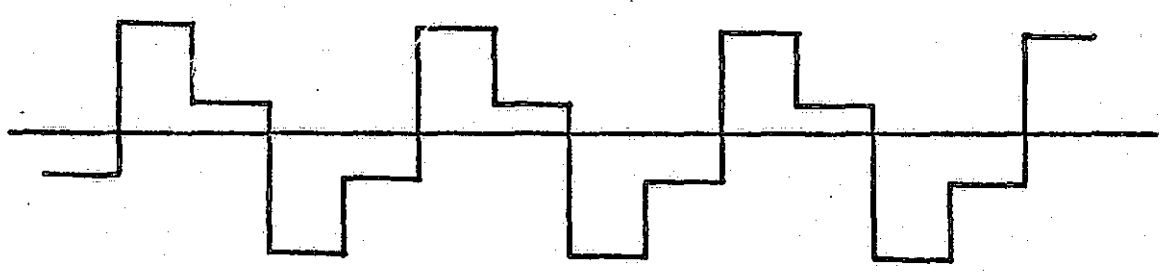


Figure 2. Steady state output bit pattern of Abate Mode Adaptive Delta Modulator

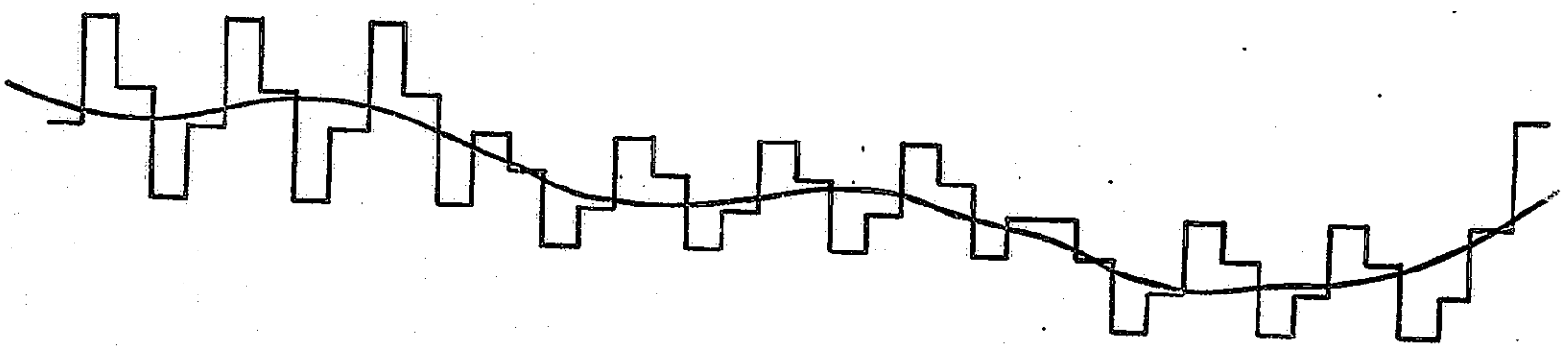


Figure 3. When signal voltage and frequencies are low, the Abate Mode Adaptive Delta Modulator responds with segments of steady state patterns.

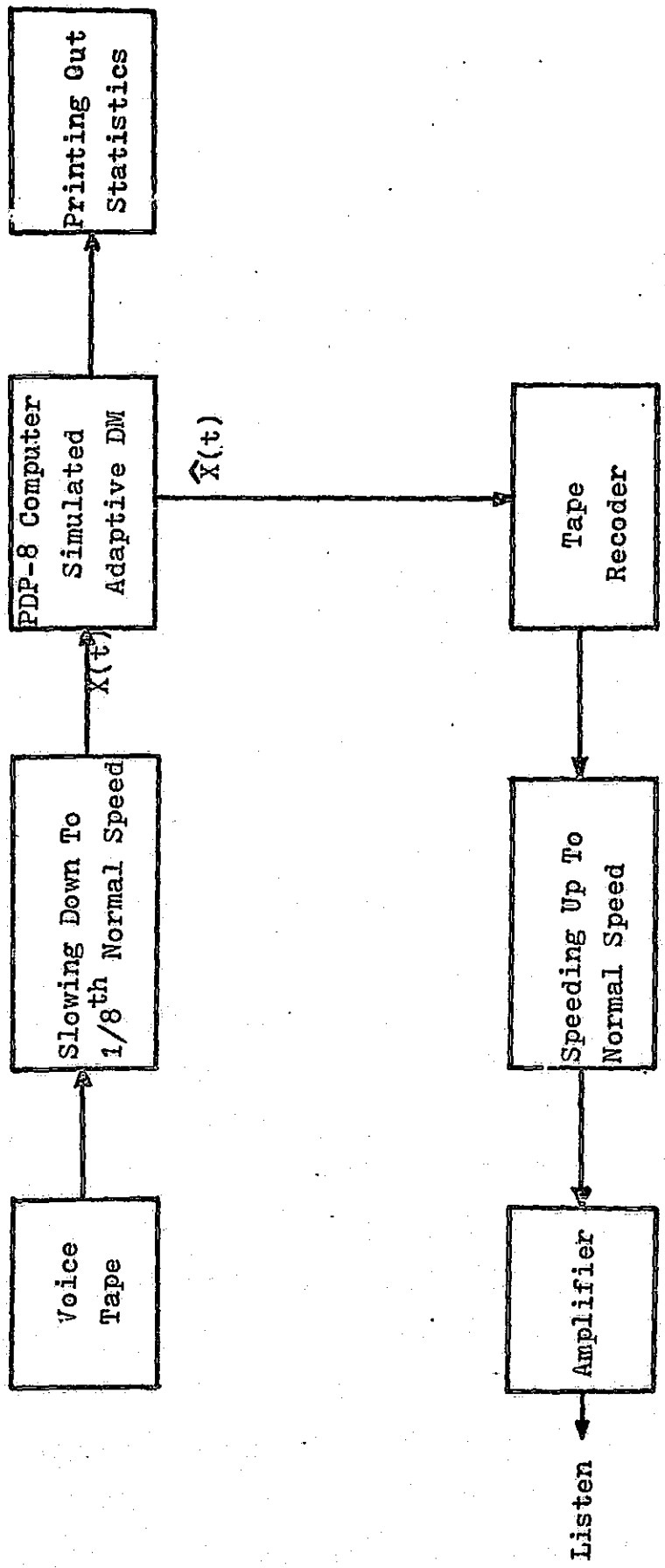


Figure 4. Block diagram of experimental procedure

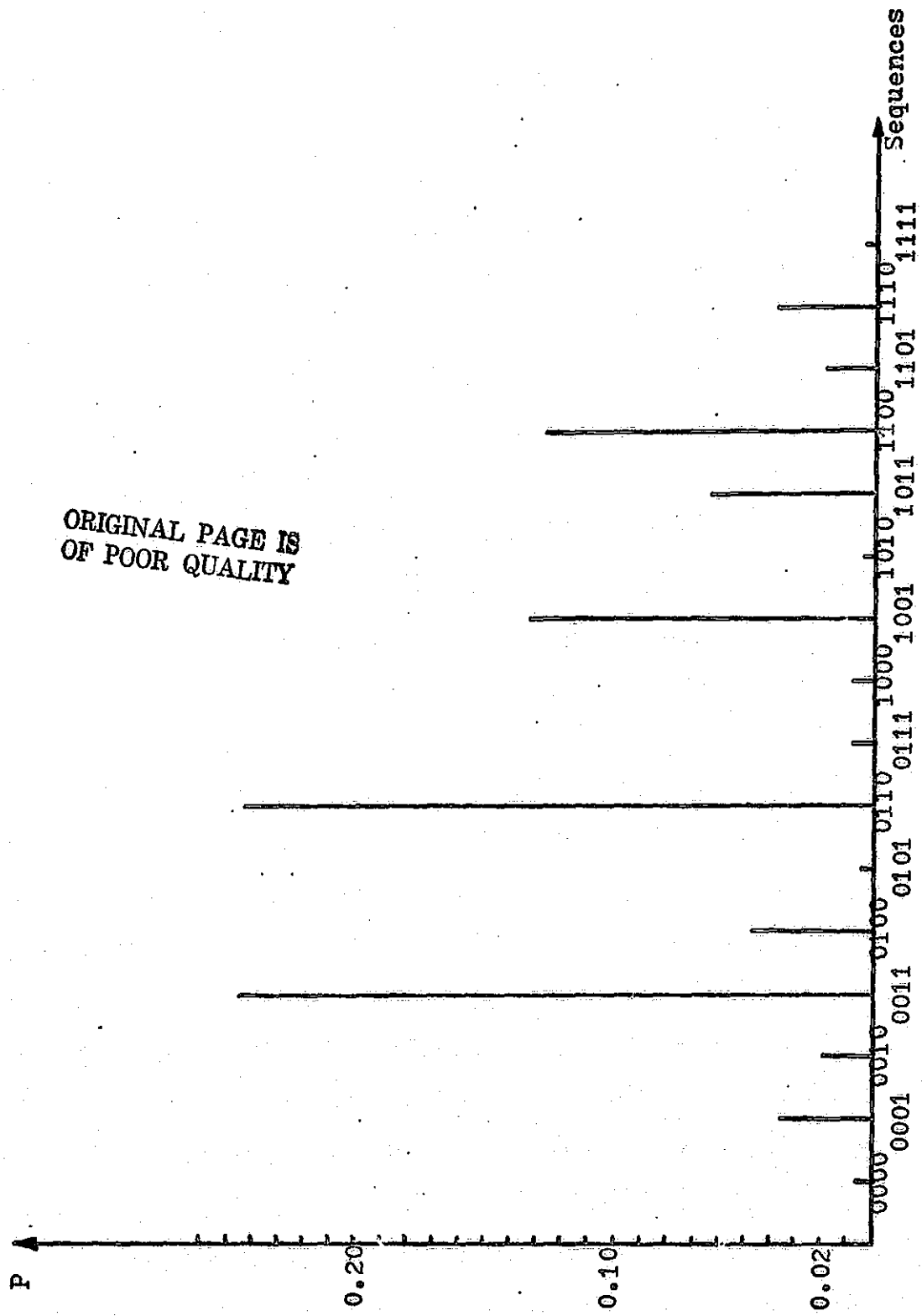


Figure 5. The probability distributions of sequences with V_{in}=1 V_{p-p}, f_s=32 KHz, S_o=40 mV.

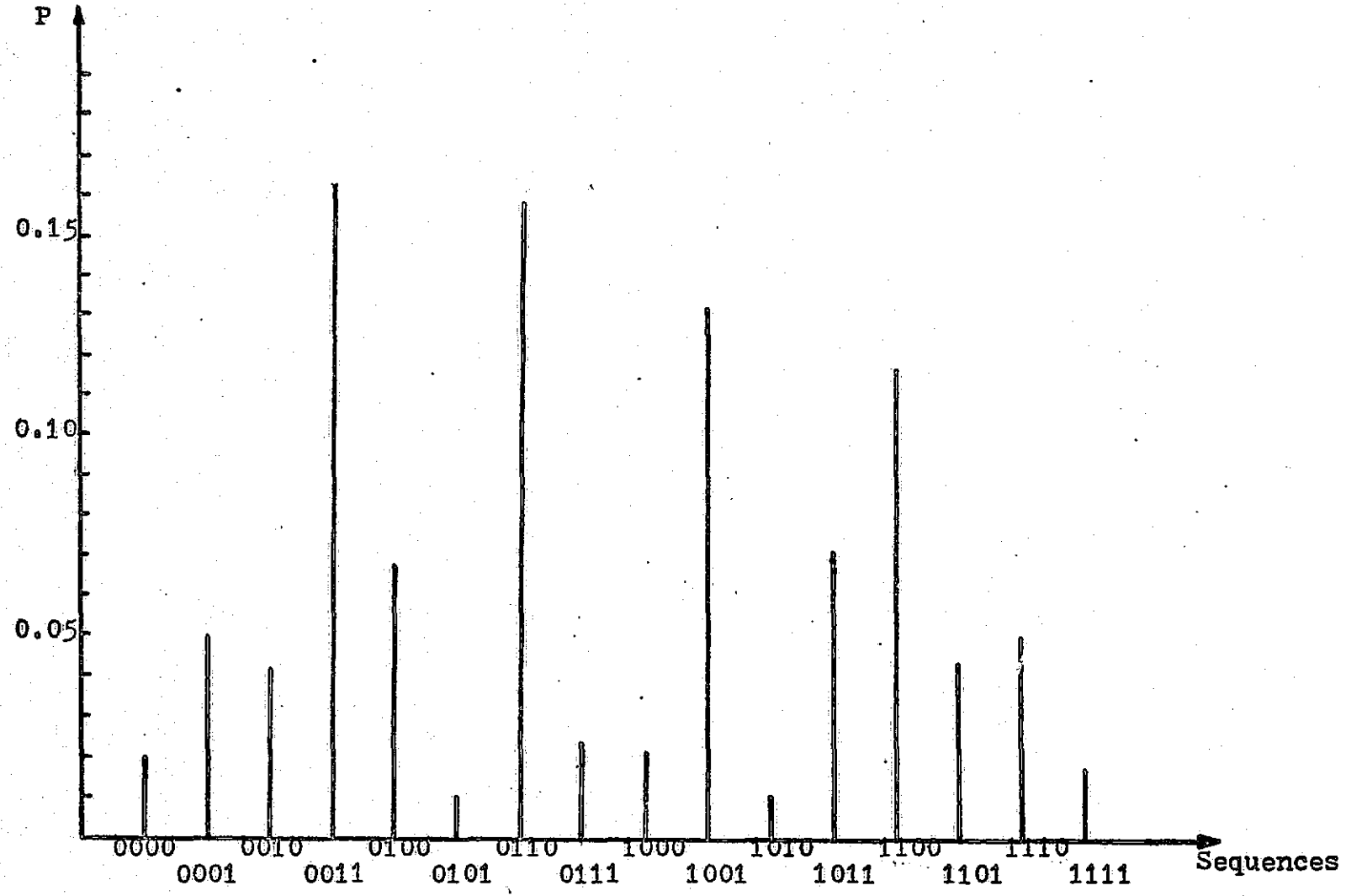


Figure 6. Probability distributions of sequences with V_{in}=2 V_{p-p}, f_s=32 KHz, S_o=40 mV.

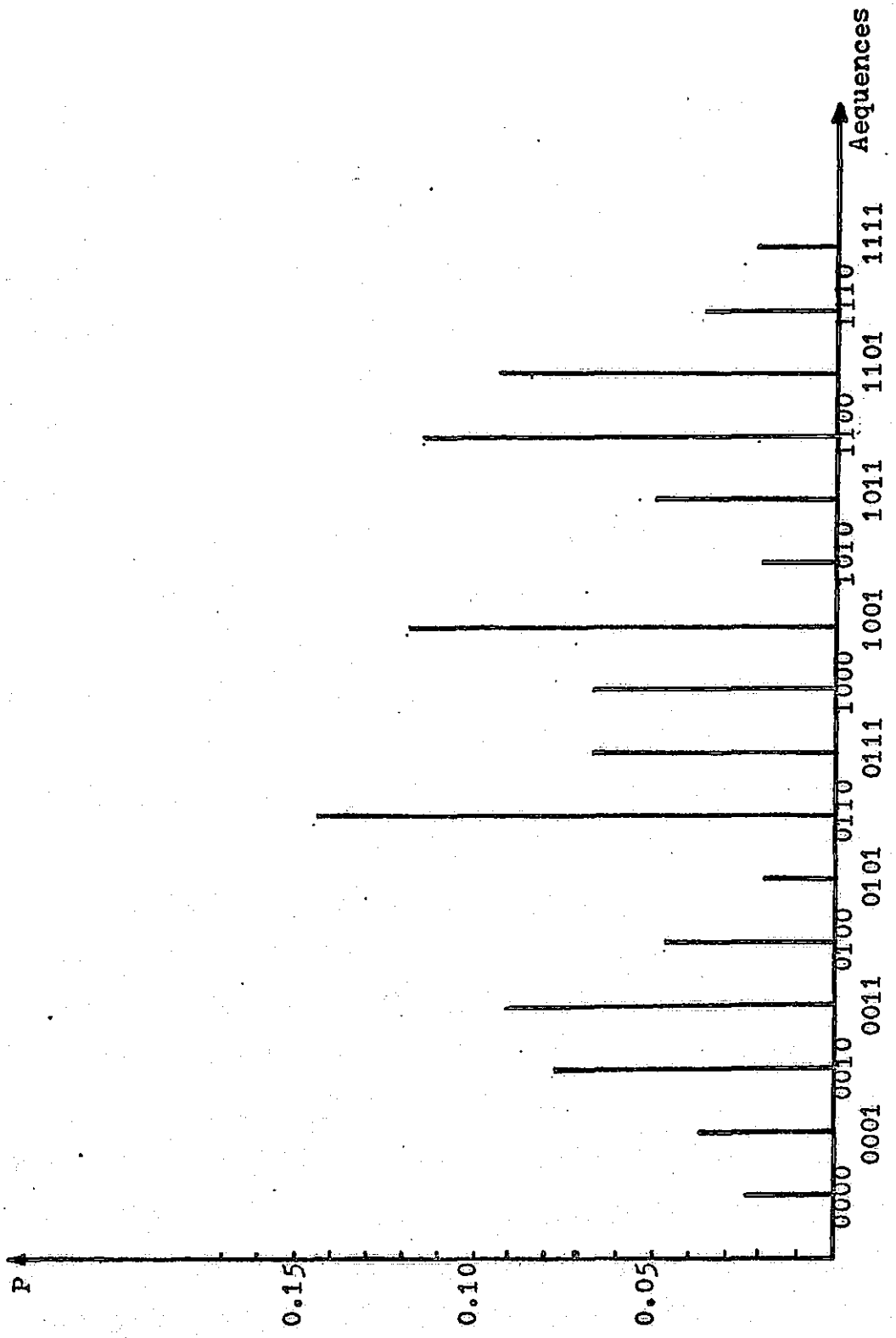


Figure 7. Probability distributions of sequences with $V_{in}=3\ V_{p-p}$, $f_s=32$ KHz, $S_o=40$ mV.

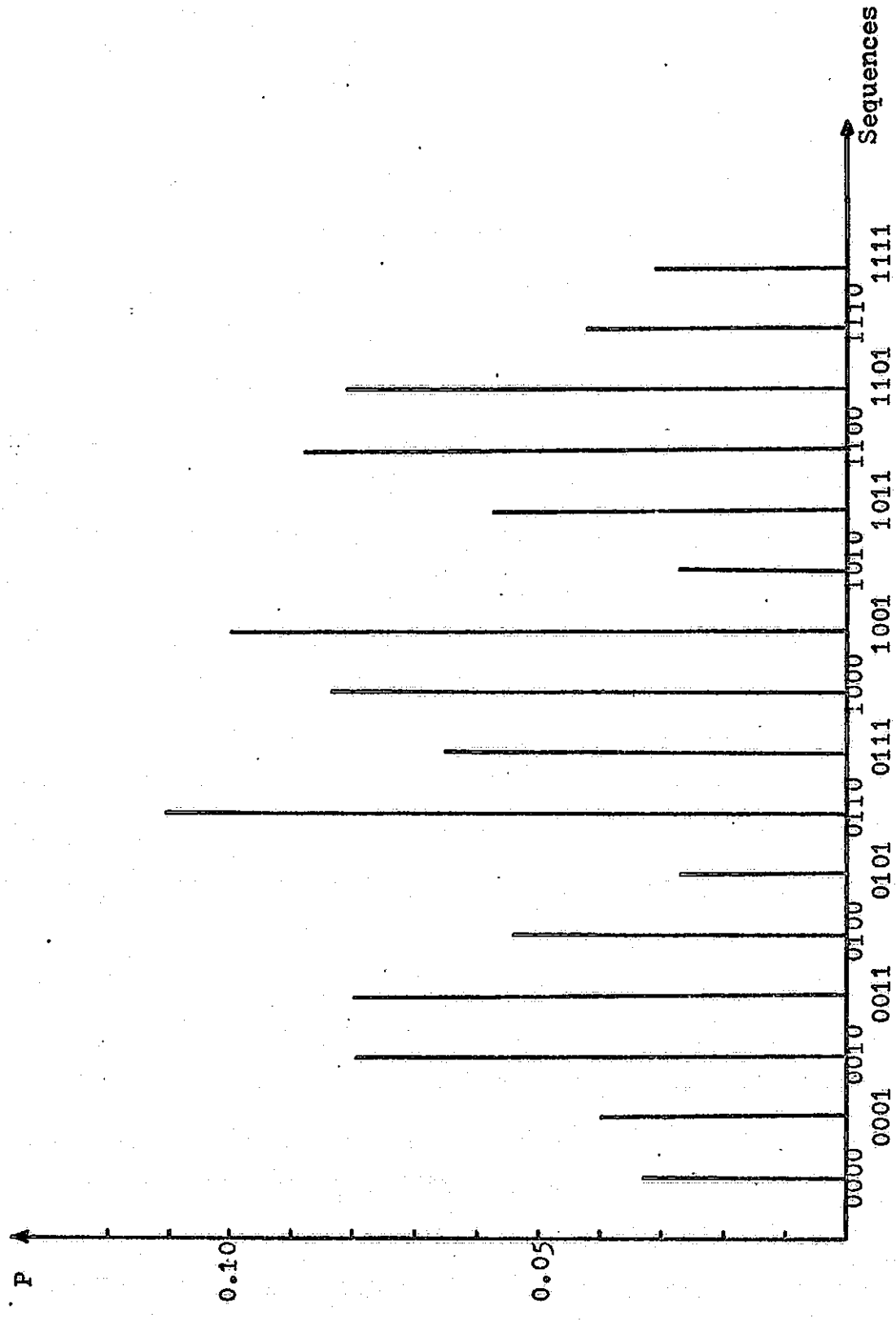


Figure 8. Probability distributions of sequences with V_{in}=6 V_{p-p}

f_s=32 KHz, S_o = 40 mV.

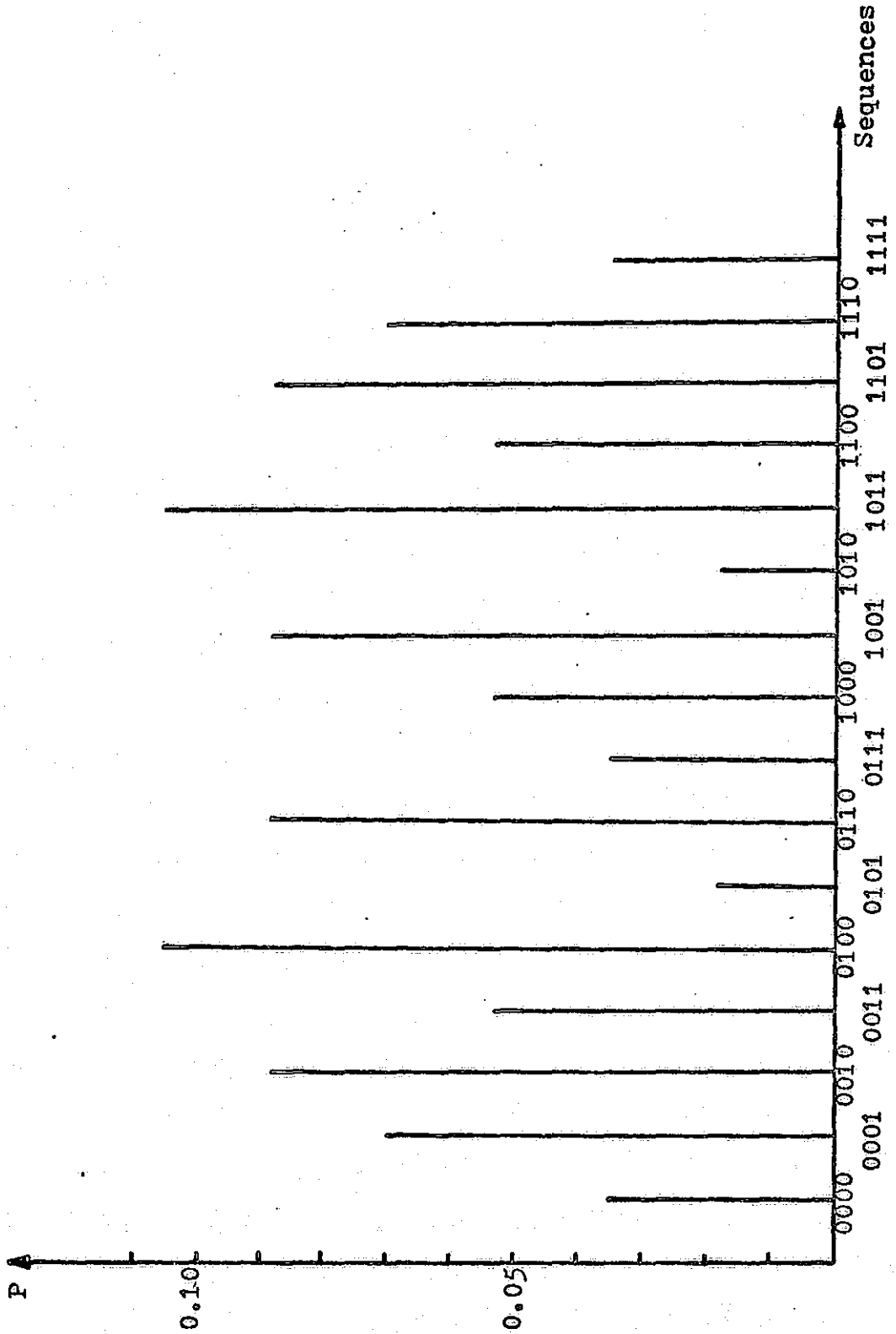


Figure 9. Probability distributions of sequences with V_{in}=20 V_{p-p}, f_s=32 KHz, S_o=40 mV.

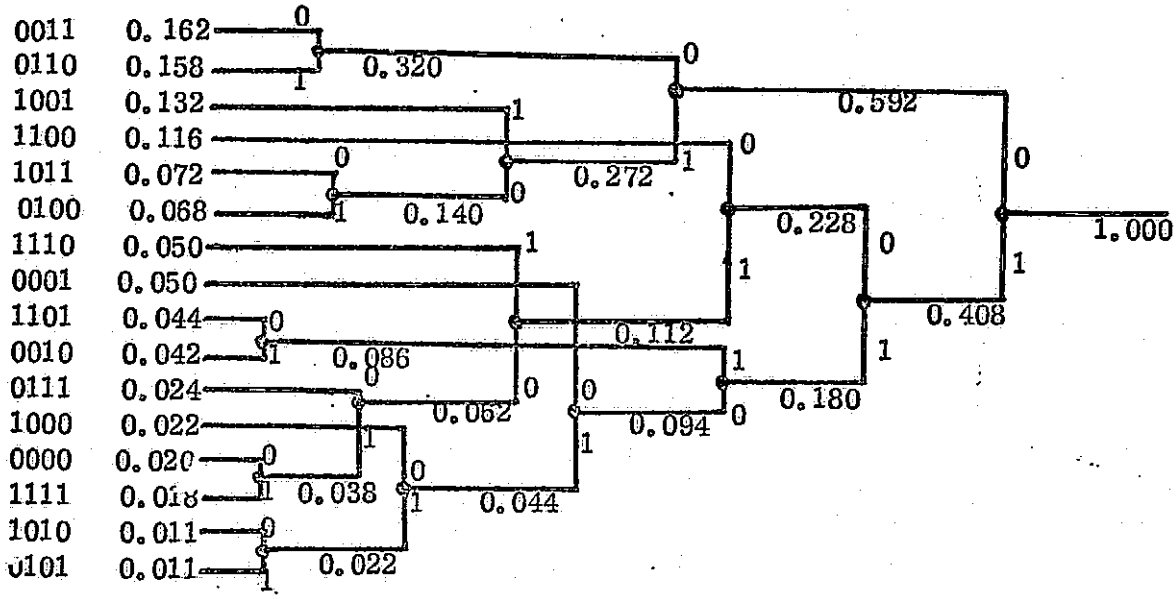


Figure 10(a). Constructing the Huffman code

0011 : 000
0110 : 001
1001 : 011
1100 : 100
1011 : 0100
0100 : 0101
1110 : 1011
0001 : 1100
1101 : 1110
0010 : 1111
0111 : 10100
1000 : 11010
0000 : 101010
1111 : 101011
1011 : 110110
0101 : 110111

Figure 10(b). The code words

The average code word length , $N = 3 \times (0.162+0.158+0.132+0.116) + 4 \times (0.072+0.068+0.050+0.050+0.044+0.042) + 5 \times (0.024+0.022) + 6 \times (0.020+0.018+0.011+0.011) = 3.598$

Received Sequence	0000	0101	1010	1111
Output Sequence	0100	0100	1011	1011

Figure 11(a). Converting the least probable sequences to more probable sequences

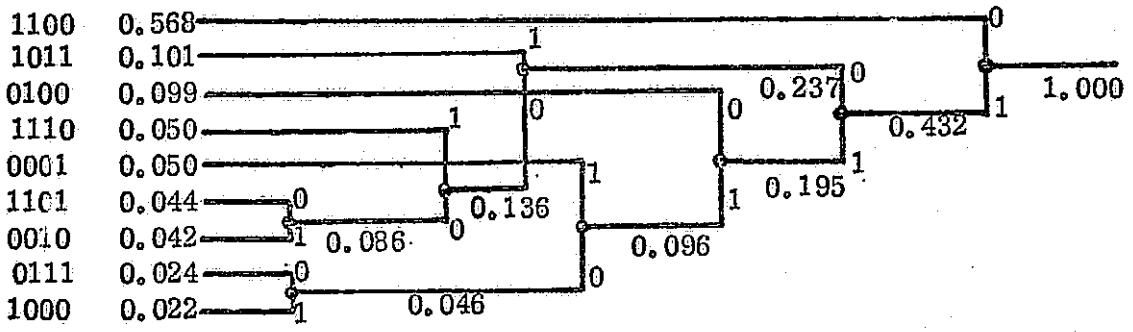


Figure 11(b). Constructing the Huffman code

1100: 0
1011: 101
0100: 110
1110: 1001
0001: 1111
1101: 10100
0010: 10101
0111: 11100
1000: 11101

Figure 11(c). The code words

The average code word length, $N = 1x0.568 + 3x(0.099 + 0.101) + 4x(0.050 + 0.050) + 5x(0.044 + 0.042 + 0.024 + 0.022) = 2.228$

I. 3. Processing and Conversion of Delta Modulation Encoded Signals

A. Performance of a Digital Adaptive Delta Modulator

1. Introduction

The most general form of a digital Delta Modulator (DM) is depicted in Fig. 1. Here we have assumed that the input signal, $x(t)$, is bandlimited to f_m and is sampled well above the Nyquist rate, i.e., $f_s >> 2f_m$. A general mathematical description of a digital DM is given by the following set of equations:

$$e_x(k) = \text{sgn}\left[\xi_x(k)\right], \tag{1a}$$

$$\xi_x(k) = x(k) - \hat{x}(k) \tag{1b}$$

and

$$\hat{x}(k) = \hat{x}(k-1) + S_x(k), \tag{1c}$$

where

$S_x(k)$ = step size at the k^{th} interval.

The particular type of DM is specified by the step size algorithm used to formulate $S_x(k)$. We shall be concerned with the Song Algorithm (1) that is used with audio signals. For this type of adaptive DM the step size increases linearly and is given by:

$$S_x(k) = \left| S_x(k-1) \right| e_x(k-1) + S\, e_x(k-2),$$

where

$S \equiv$ magnitude of the minimum step size.

In order to evaluate the performance of the Song audio mode DM, we shall let x (t) be a sinusoid of frequency f_m and let f_s be an integral multiple of f_m. Whenever the sampling frequency is an integral multiple of the sinusoid frequency, the DM estimate, $\hat{x}$ (k), will assume a periodic sinusoidal steady state pattern. Since the estimate is periodic, it can be expressed as a Fourier series and the fundamental as well as the higher harmonics can be calculated. We can then pass each frequency component thru a realistic low pass filter and use the resulting filtered estimate to calculate the output signal-to-noise ratio.

2. Fourier Series Representation of the DM Estimate

In order to facilitate the derivation of the Fourier series for $\hat{x}$ (k) we shall assume that its fundamental frequency is f_m and not a submultiple of it. If we let $f_s = P f_m$ and $T = 1/f_m$, then this means that $\hat{x}$ (k) periodically takes on P discrete values every T seconds. Using the continuous notation, $\hat{x}$ (t), the Fourier series for the DM estimate is

$$\hat{x}(t) = C_o + \sum_{n=1}^{\infty} C_n \cos(2\pi n f_m t + \varphi_n), \tag{2a}$$

where

$$C_o = (1/T) \int_o^T \hat{x}(t)\, dt, \tag{2b}$$

$$C_n = \sqrt{A_n^2 + B_n^2}, \tag{2c}$$

$$\varphi_n = -\arctan(B_n / A_n), \tag{2d}$$

and

$$A_n = (2/T) \int_o^T \hat{x}(t) \cos(2\pi n f_m t)\, dt, \tag{2e}$$

$$B_n = (2/T) \int_o^T \hat{x}(t) \sin(2\pi n f_m t)\, dt. \tag{2f}$$

Since $\hat{x}(t)$ takes on P discrete values in a period of T seconds, if we represent these values as $\hat{x}_j$, then Eqs. (2e) and (2f) can be written as

$$A_n = (2/T) \sum_{j=1}^{N} \hat{x}_j \int_{(j-1)T/N}^{jT/N} \cos(2\pi n f_m t) dt \tag{3a}$$

and

$$B_n = (2/T) \sum_{j=1}^{N} \hat{x}_j \int_{(j-1)T/N}^{jT/N} \sin(2\pi n f_m t) dt. \tag{3b}$$

Using the fact that $f_m = 1/T$ and some trignometric identities, A_n and B_n reduce to

$$A_n = \frac{2 \sin(n\pi/N)}{n\pi} \sum_{j=1}^{N} \hat{x}_j \cos[n\pi (2j-1)/N] \tag{4a}$$

and

$$B_n = \frac{2 \sin(n\pi/N)}{n\pi} \sum_{j=1}^{N} \hat{x}_j \sin[n\pi (2j-1)/N]. \tag{4b}$$

Equations (4a) and (4b) represent the simplest expressions obtainable to determines the strength of the Fourier components, C_n. It is easy to see that the digital adaptive DM as well as this method of obtaining the Fourier series for the DM estimate are both readily adaptable for computer simulations on almost any digital computer.

3. Output Signal-to-Noise Ratio

Since we are concerned with an audio mode DM it would seem reasonable to choose a low pass filter that is applicable to voice signals. A common low pass filter is a fourth order Butterworth type whose magnitude-squared transfer function is given as

$$|H(s)|^2 = 1/[1 + (s/\omega_c)^8] \tag{5}$$

where

$\omega_c \equiv$ the radian cutoff frequency.

We are interested in the frequency characteristics of this low pass filter and find them to be

$$|H(f)| = 1/\sqrt{1 + (f/f_c)^8}, \tag{6}$$

where

$$f_c = \omega_c/2\pi.$$

To realisticly represent a voice signal we shall choose $f_m = 600$Hz and $f_c = 4f_m = 2400$Hz. From Eq. (6) we now obtain the attenuation factor, α_n, that must be applied to the Fourier components of $\hat{x}(t)$ in order to simulate low pass filtering,

$$\alpha_n = [1 + (n/4)^8]^{-\frac{1}{2}}. \tag{7}$$

Since all harmonics are orthognal we will only be concerned with the attenuation produced by the low pass filter and not consider the phase shift which arises.

After final low pass filtering, the output signal power is seen to be

$$S_o = \tfrac{1}{2}(\alpha_1 C_1)^2. \tag{8}$$

The output noise power comes from all the frequency harmonics other than the fundamental. After the low pass filter the output noise power is expressable as

$$N_o = \tfrac{1}{2} \sum_{n=2}^{\infty} (\alpha_n C_n)^2 \tag{9}$$

Thus the output signal-to-noise ratio (SNR) is given as

$$SNR_o = \frac{S_o}{N_o} = \frac{(\alpha_1 C_1)^2}{\sum_{n=2}^{\infty} (\alpha_n C_n)^2} \quad . \tag{10}$$

4. Computer Simulation

Thus far we have successfully simulated the Song audio mode DM on a PDP 8/L computer employing 8K of memory. We have observed the response to an input sinudoid and have verified that the DM estimate is in fact periodic. In addition, the Fourier components have been calculated using Eqs. (4a); (4b) and (2c) and the resulting output signal-to-noise ratio has been determined. Naturally we did not use an infinite number of harmonics to calculate the noise power as required by Eq. (9). Instead we truncated after the ninth harmonic because the term $(\alpha_{10} C_{10})^2$ was negligible in comparison to the total noise due to the second thru ninth harmonics.

At this time we are constructing a family of curves depicting output signal-to-noise ratio in dB versus relative input signal power also in dB for various ratios of f_s/f_m and for the minimum step size set to unity. We have found that, for the same input signal amplitude, the periodic pattern that the estimate assumes and consequently the output signal-to-noise ratio is very dependent upon the starting point of the input sinusoid. In Fig. 2 we show the DM response to a constant input. Since the DM estimate is periodic with a period of $4T_s = 4/f_s$, the input sinusoid can start at any point within this period with equal probability. In Fig. 2 we also show a number of possible starting points of the input sinusoid. In order to obtain a truly representative value of output signal-to-noise ratio we are currently averaging the output signal-to-noise ratios obtained for 20 different starting points of the input sinusoid.

B. Direct Arithmetic Processing of Delta Modulation Encoded Signals

A technique for adding and multiplying signals that are DM encoded without first converting them to a Pulse Code Modulation (PCM) format has successfully been developed. The results of this investigation were presented as a

paper, which is included in the appendix, at the National Telecommunications Conference, December 1-3, 1974, in San Diego, California.

In this paper we develope both an addition/subtraction algorithm and a multiplication algorithm for DM encoded signals. We present the systems to realize these algorithms and show that for constant inputs the performance of these systems is identical to the performance of PCM adders and multipliers. In addition we display experimental results when elementary signals are used as inputs which verify the theory that has been developed.

At the present time we are attempting to apply the Fourier series theory developed above to DM encoded signals which have been subjected to direct arithmetic processing. By forming the direct sumand product of sinusoidal input signals which have first been DM encoded, we will be able to produce output signal-to-noise ratios for both the direct sum and the direct product.

C. DM to PCM Conversion

1. Basic Philosophy

Frequently the situation arises where a signal has been DM encoded and transmitted but at the receiver we wish to use the signal information in a system that requires a PCM encoded signal. Since both DM and PCM are digital encoding techniques, we would like to avoid demodulating the DM signal into analog form and then recoding it in PCM form. Thus the need arises for a direct, all digital method of conversion from DM to PCM.

The most obvious method to convert from DM to PCM would be to pass the DM bits, $e_x(k)$, thru the DM digital feedback circuit, as shown in Fig. 1, and produce the DM estimate, $\hat{x}(k)$. Since the DM operates at a rate much higher than the Nyquist rate, it would be necessary to gate $\hat{x}(k)$ at the Nyquist rate so as to obtain PCM format. The problem with merely using DM estimate values that occur at the Nyquist rate for our PCM sample values is that we run the risk of obtaining a "poor" value of $\hat{x}(k)$. By a "poor" value of $\hat{x}(k)$ we mean one which exhibits a large variation from the true PCM sample value, $x(k)$. A "poor" DM estimate is frequently produced when the

DM step size has grown too quickly causing the estimate to overshoot the true signal. If the true signals continues to increase, the estimate will reverse direction for one period, due to the overshoot, and then continue to increase. During the one period when the DM has reversed direction, the $\hat{x}(t)$ value is generally a "poor" estimate. When the DM is demodulated to an analog signal, the " poor" estimate values are easily averaged out by a low pass filter because the DM operating frequency is much higher than the Nyquist rate. However, since the PCM samples occur at the Nyquist rate, a "poor" value will give rise to a considerable error even after final low pass filtering.

In order to eliminate these "poor" values of $\hat{x}(k)$, and still maintain a completely discrete system, we can insert a digital filter after the DM estimate, just before the gating device operating at the Nyquist rate. The digital filter may be viewed as a device which filters in the frequency domain, produces a statistical estimate, or performs a digital interpolation. In all cases the result is to decrease the out of band noise and make our estimate values more accurate.

2. Nonrecursive Filtering Technique

The use of a nonrecursive filter to achieve DM to PCM conversion was first invertigated by D. Goodman (2) who only considered the case of a linear DM and used a minimun mean square error design criterion to determine the filter coefficients. In order to complete his design, it was necessary to assume input signal statistics. We have dealt with the more general case of any digital adaptive DM. In additon, we use a method to determine filter coefficients which is completely independent of input signal statistics.

The basic DM to PCM conversion scheme for any digital adaptive DM first forms the step size, $S_x(k)$, from the DM bits, $e_x(k)$; then accumulates the step sizes to produce the DM signal estimate, $\hat{x}(k)$. These operations are performed at the DM operating frequency,

$$f_s = 2N f_m,$$

where N is a positive integer and the input signal, $x(t)$, is assumed to be bandlimited to f_m. Finally we gate $\hat{x}(k)$ at the PCM or Nyquist frequency,

$$f_N = 2f_m ,$$

to produce the PCM samples, $\hat{x}(Nk)$. This scheme is shown in Fig. 3.

In order to eliminate the poor values of $\hat{x}(k)$, we can insert a low pass filter after the accumulator. Now our converter takes the form of the DM step size, $S_x(k)$, feeding two cascaded linear filters. The accumulator can be represented as an ideal integrator whose impulse response is given as

$$\begin{aligned} a(t) &= 1 \quad , \quad t \geq 0 \\ &= 0 \quad , \quad t < 0. \end{aligned} \tag{11}$$

Let us designate the impulse response of the nonanticipatory low pass filter as

$$\begin{aligned} h(t) \quad &, \quad t \geq 0 \\ h(t) = 0 \quad &, \quad t < 0. \end{aligned}$$

In order to complete this converter we must gate the output of the low pass filter to yield the improved PCM sample, $\tilde{x}(Nk)$. In Fig. 4 we present this system before the two cascaded filters, $a(t)$ and $h(t)$, have been transformed to a nonrecursive digital filter.

Since both $a(t)$ and $h(t)$ represent linear filters, they can be combined into one linear filter, $g(t)$, where

$$\begin{aligned} g(t) &= a(t) * h(t), \\ &= \int_{-\infty}^{\infty} a(t-\lambda)\, h(\lambda)\, d\lambda . \end{aligned} \tag{12}$$

The upper limit in Eq. (12) becomes t because the accumulator is nonanticipatory, i.e., a $(t-\lambda) = 0$ for $\lambda > t$, and the lower limit becomes zero since the low pass filter is causal, i.e., $h(\lambda) = 0$ for $\lambda < 0$. We also notice that for these limits of integration $a(t-\lambda) = 1$. Therefore Eq. (12) reduces to

$$g(t) = \int_0^t h(\lambda)\, d\lambda, \tag{13}$$

and this is merely the unit step response of the low pass filter.

Now we can express the filtered DM estimate, $\tilde{x}(k)$, by the following discrete convolution:

$$\tilde{x}(k) = \sum_{j=-\infty}^{\infty} S_x(k-j)\, g(j), \tag{14}$$

where

$$g(j) = g(j T_s)$$

and

$$T_s = 1/f_s = \text{the DM sampling period.}$$

The lower limit of the sum in Eq. (14) becomes zero because the filter g(t) is causal, i.e., $g(j) = 0$ for $j < 0$. Thus we have

$$\tilde{x}(k) = \sum_{j=0}^{\infty} S_x(k-j)\, g(j), \tag{15}$$

Since g(t) represents the unit step response of a low pass filter, we know that

$$\lim_{t\to\infty} g(t) = 1 \qquad \text{or} \lim_{j\to\infty} g(j) = 1, \tag{16}$$

and that there exists a value of j (or t) for which g(j) is arbitrarily close to 1. If we call this value Q, then

$$g(j) \simeq 1 \qquad \text{for all } j \geq Q. \tag{17}$$

Using this fact, the filtered DM estimate can be ap proximated very closely by

$$\tilde{x}(k) \simeq \sum_{j=0}^{N-1} S_x(k-j)\, g(j) + \sum_{j=N}^{\infty} S_x(k-j). \tag{18}$$

If we notice that the second sum can be rewritten, letting $k - j = i$, as

$$\sum_{j=N}^{\infty} S_x(k-j) = \sum_{i=k-N}^{j-\infty} S_x(i) = \sum_{i=-\infty}^{k-N} S_x(i), \tag{19}$$

and recall that the DM estimate is given by

$$\hat{x}(k) = \sum_{i=-\infty}^{k} S_x(i), \tag{20}$$

then we can ultimately express the filtered DM estimate as

$$\tilde{x}(k) \simeq \sum_{j=0}^{N-1} S_x(k-j)\, g(j) + \hat{x}(k-N). \tag{21}$$

In Fig. 5, we give the block diagram of this nonrecursive digital filter.

Our design is now complete except for the choice of the filter coefficients,

g(j). In order to achieve the best out of band noise rejection without any in band signal deterioration, we choose g(j) to be samples of the unit step response of an ideal low pass filter. Although an analog low pass filter is not physically realizable, when a nonrecursive sigital filter is constructed we can choose any coefficients desired to simulate a given characteristic. In Fig. 6 we show the unit step response of an ideal low pass filter plotted on a normalized abscissa. Analytically g(t) is expressed as

$$g(t) = \tfrac{1}{2} + (1/\pi)\ \mathrm{Si}\ (\omega_c t - K_o) \tag{22}$$

where

$$\mathrm{Si}\ (\alpha) \equiv \int_0^{\alpha} \frac{\sin x}{x}\ dx,$$

ω_c ≡ the radian cutoff frequency,

$K_o = \omega_c T_d$,

and

T_d ≡ the time delay of the filter.

Since g(0) = 0 and Si (α) is an odd function,

$$\mathrm{Si}\ (K_o) = \pi/2 \quad \text{or} \quad K_o = 1.926 \text{ radians.}$$

In order to obtain the filter coefficients, we select N points of g(t) in the interval $2\omega_c T_d$. Thus we set

$$T_s = 2T_d/N, \tag{23}$$

and write g(k) as

$$g(k) = \tfrac{1}{2} + (1/\pi)\ \mathrm{Si}\ (\omega_c k T_s - K_o)\,. \tag{24}$$

Using Eq. (23) and the fact that $T_d = K_o/\omega_c$, we obtain a final expression for the filter coefficients:

$$g(k) = \tfrac{1}{2} + (1/\pi)\ \mathrm{Si}\ [\ K_o\ (\ (2k/N) - 1)\ ]\,. \tag{25}$$

What should be pointed out at this point is that the coefficients obtained from Eq. (25) are within 1% of the values obtained by D. Goodman (2) for the case N = 5 which is documented in this reference. More significantly we stress that these coefficients were derived independent of input signals statistics. Presently we are undertaking a computer simulation of this system in order to obtain signal-to-noise ratio curves.

3. Recursive Filtering Technique

If we approach the problem of achieving DM to PCM conversion from a strictly digital system point of view, the solution lends itself to the use of recursive digital filters. The objection is to relieve $\hat{x}(k)$ of its "poor" estimate values before the PCM samples are gated out. This can be achieved by a sharp cutoff low pass filter. These characteristics can be best obtained with minimum hardware by using a recursive digital low pass filter. The filter is inserted after the accumulator which produces $\hat{x}(k)$ and before the gating device which renders the PCM samples, $\tilde{x}(Nk)$.

The recursive filter design techniques which have been used are the impulse invariant method and the squared-magnitude method. Both design procedures are well document by Gold and Rader (3). Currently we are undertaking computer simulations to obtain the performance of this system with various different types of recursive digital low pass filters.

D. PCM to DM Conversion

1. Statement of the Problem

Consider the case where a signal is encoded in PCM format but we wish to use a digital processing technique that requires the signal to be DM encoded. Now we wish to convert from information arriving at the Nyquist

rate, $2f_m$, to a DM form which has a frequency of occurence which is N times faster. That is, the DM operating frequency is $f_s = 2Nf_m$. In addition, we would like to confine our conversion technique to all digital hardware.

It is evident that this problem is much more complicated than the DM to PCM conversion problem considered in the previous section. In PCM to DM conversion we do not have information about the signal excursion between PCM samples and a DM estimate can follow several paths and still pass thru the given PCM samples. Thus we must address ourselves to a method which first obtains additional sample values between the PCM samples and then uses these to choose the correct DM estimate path. Since the DM estimate is directly related to its output bit stream for a prespecified digital adaptive DM, we will have them achieved the desired conversion.

2. Submultiple Sampling Technique

Although there are many approaches to this problem, the solution presented here is consistent with previously developed conversion methods in that it appeals to a basic theoretical principle and results in a system which is easily physically realizable with the current state of the art digital hardware. We normally expect that the best way to demodulate the PCM signal would be to pass it thru an analog low pass filter. By doing this, we extract all the information between PCM samples. However, we seek only a finite number of additional sample values between the PCM points. In order to achieve this, we use a low pass filter which is not only digital but which operates in a submultiple sampling mode (4).

When a digital system operates in the submultiple sampling mode it means that the system produces outputs at a frequency which is an integer multiple of the input frequency. This is exactly the circumstances that exist when we derive additional sample values from the PCM samples. To facilitate the entire PCM to DM conversion process, we choose to operate the submultiple digital low pass filter at the desired DM frequency, $f_s = 2Nf_m$. Then we can use these values as the input to the digital adaptive DM that we are employing and therefore automatically generate the necessary DM bits, $e_x(k)$. The only

restriction to this technique is that the DM must operate at a high enough frequency so that there is negligible error between the output of the filter and the resulting DM estimate. The general block diagram of the PCM to DM converter is given in Fig. 7.

In order to thoroughly describe the submultiple sampling technique that we are investigating, it becomes necessary to dichotomize this technique into two types. The first type uses the PCM samples as inputs to the digital low pass filter for only $T_s = 1/f_s$ seconds. In the time between PCM samples, (N-1) T_s seconds, we insert zeros or no input. This is analygous to the situation where the filter input is a series of impulses. The second type employs the PCM samples as filter inputs for NT_s seconds. This is similar to the PCM samples being held for the Nyquist period.

1. PCM Samples with Zeros Inserted

We shall describe the theory for both cases in a general fashion omitting the particular characteristics of the low pass filter. Furthermore let us start with the ordinary Z - transform of a digital low pass filter represented by G(z) and the input and output of this filter specified by X(z) and Y(z) respectively. Then as we normally expect,

$$Y(z) = G(z)\ X(z). \tag{26}$$

In addition, we require that the input occurs once every Nyquist sampling period, $T_N = 1/2f_m$.

If we want the output to occur n times in the Nyquist sampling period and the input, which are PCM samples of X(z), to have zeros between these PCM samples, then the output is expressed as

$$Y(z)_n = G(z)_n\ X(z), \tag{27}$$

where

$$G(z)_n = [\, G(z)\,]_{z=z^{1/n}} \qquad T_N = T_N/n \tag{27a}$$

$G(z)_n$ is referred to as the submultiple sampling Z-transform of the digital low pass filter and is realized exactly the same as the ordinary digital filter except delay elements are reduced by a factor n and scalars with the constant T_N included are attenuated by a factor n. This implies that the digital filter operates at a frequency $2nf_m$ and for only one out of every n cycles is there an input which is a PCM sample and not an inserted zero. The filter operates on these inputs to produce estimates of the signal between PCM samples.

2. PCM Samples Held

In order to obtain n outputs per Nyquist sampling period for the case when the input PCM sample is maintained as the filter input of all n cycles of the Nyquist period, it is necessary to employ a digital hold circuit, H(z), after X(z). A digital circuit to hold one value as its output for J periods when the held value is the input during the first period and there is zero input for the second thru the J^{th} periods has the transfer function:

$$H(z) = \frac{B(z)}{A(z)} = \sum_{i=0}^{J-1} z^{-1} \tag{28}$$

A block diagram of this J period digital hold circuit is given in Fig. 8.

If we incorporate the hold circuit into our digital low pass filter operating in the submultiple sampling mode then we must hold the PCM sample for n filter cycles. The desired PCM-held output is given by

$$Y_H(z)_n = G(z)_n\, H(z)_n\, X(z), \tag{29}$$

where

$$H(z)_n = \sum_{i=0}^{n-1} z^{-i/n} .$$

This completes the general theory for both types of the submultiple sampling PCM to DM conversion technique leaving the option to choose whatever digital low pass filter characteristics that may be desired and to set the number of filter cycles per Nyquist period, n, to be a specific number, N, which is large enough so that there is negligible error between the output of the filter and the resulting DM estimate.

3. Normalization

Because the DM can suffer from both slope overload noise and granular noise due to an input which is too large or too small, respectively, we should take the precaution to insure that our digital low pass filter does not contribute to either of these degradations by amplifying or attenuating the input signal. This can be accomplished if we normalize the filter transfer function in the frequency domain. Since $z = \exp(j\omega T_N)$, we normalize G(z) so that at $\omega=0$ it will be unity. This normalization applies to the first type considered were zeros are inserted between PCM samples. If we call the normalized low pass filter transfer function, $G'(z)$, then

$$G'(z) = G(z)/G(\omega = 0) \tag{30}$$

and

$$G'(z)_n = G(z)_n/G(\omega = 0). \tag{30a}$$

Using the normalized transfer function, the true filter output is

$$Y'(z)_n = G'(z)_n X(z). \tag{31}$$

For the type of converter when the PCM samples are held we seek to normalize $G(z)_n$ so that at $\omega = 0$ this will be unity. Let us denote the normalized held transfer function as $G''(z)_n$ which is then given by

$$G''(z)_n = G(z)_n / G(\omega = 0)_n. \tag{32}$$

Now the true output becomes the expression

$$Y_H''(z)_n = G''(z)_n H(z)_n X(z). \tag{33}$$

4. System Evaluation

In order to determine the performance of both types of PCM to DM converter we are undertaking two different approaches. One will be purely analytical in which we let $X(z)$ equal the Z-transform of a sinusoid of frequency f_m which is sampled at the Nyquist rate. After $G(z)$ and n are chosen, we can find the Z-transform of the filter output; take the inverse Z-transform; and then evaluate the mean square error.

An alternate approach will be to completely simulate each type of converter and the desired sinusoidal input on a digital computer. We can use the Fourier analysis technique described earlier to determine the signal-to-noise ratio of the DM estimate. This will give us an even more accurate measure of the quality of the conversion system. In both approaches we can vary the filter characteristics and observe the effect of filter hardware complexity upon performance. The latter approach also allows us to vary n and observe the different system performance in terms of a change in DM estimate signal-to-noise ratio.

References

(1) C. L. Song, J. Garodnick and D. L. Schilling, "A Variable Step Size Robust Delta Modulator", IEEE Trans. Commun. Tech., vol. COM-19, pp. 1033-1044, Dec. 1971.

(2) D. J. Goodman, "The Application of Delta Modulation to Analog-to-PCM Encoding", Bell Sys. Tech. J., vol. 48, pp. 321-343, Feb. 1969.

(3) B. Gold and C. M. Rader, "Digital Processing of Signals", N. Y.: McGraw Hill, 1969, pp. 51-70

(4) B. C, Kuo, "Analysis and Synthesis of Sampled-Data Control Systems", Englewood Cliffs, N. J.: Prentice Hall, 1963, pp. 82-86.

APPENDIX

Direct Arithmetic Processing Of Delta Modulation Encoding Signals

J. L. LO CICERO, D. L. SCHILLING and J. GARODNICK

Dept. of Electrical Engineering
The City College of the City
University of New York

Goldmark Communications Corp.
Stamford, Connecticut

Abstract

In this paper we show that two or more Delta Modulation (DM) encoded signals can be added or multiplied without first converting them to a Pulse Code Modulation (PCM) format. These results are obtained for a large class of all digital adaptive DMs as well as for the basic all-digital linear DM. The addition and multiplication is performed by operating directly on the DM bit stream and the sum or product signal is presented as either a DM bit stream or in a PCM format.

Bounds are given on the Signal-to-Quantization Noise Ratio (SNR) for these DM arithmetic operations and compared with the results obtained in PCM. Experimental results are presented which verify the theory that has been developed.

Introduction

The conventional approach to arithmetic signal processing is to first encode the signal in Pulse Code Modulation (PCM) format and then digitally process the PCM signal via standard parallel processing techniques. It is becoming increasingly popular to encode signals in Delta Modulation (DM) format, where digital data is presented in a serial rather than a parallel fashion. If we wish to arithmetically process DM encoded signals, it becomes necessary to perform the additional operation of conversion from DM to PCM before processing is initiated. It is, however, possible to avoid this additional operation of conversion and to produce the sum, difference and even the product of DM encoded signals by direct arithmetic processing of the serial data.

In Fig. 1, we show the basic form of a digital DM. Here we have assumed that the input signal, $x(t)$, is bandlimited to f_m and is sampled well above the Nyquist rate, i.e., $f_s >> 2f_m$. The most general mathematical description of a digital DM is given by the following set of equations:

$$e_x(k) = \text{sgn}[\xi_x(k)], \tag{1a}$$

$$\xi_x(k) = x(k) - \hat{x}(k) \tag{1b}$$

and

$$\hat{x}(k) = \hat{x}(k-1) + S_x(k), \tag{1c}$$

where

$S_x(k)$ = step size at the kth interval.

The particular type of DM is specified by the step size algorithm used to formulate $S_x(k)$. For the linear DM,

$$S_x(k) = Se_x(k-1), \tag{2}$$

where

S ≡ magnitude of the minimum step size.

Although there are many step size algorithms for adaptive DMs, where the step size adapts to the input signal power, we shall be concerned with the class of DMs derived by minimizing a mean square cost function, i.e., those described by the Song Algorithm [1]. For the case of the Song audio mode DM, where the step size increases linearly,

$$S_x(k) = |S_x(k-1)|e_x(k-1) + Se_x(k-2), \tag{3}$$

and for the Song video mode DM, where the step size increases exponentially,

$$\begin{aligned} S_x(k) &= |S_x(k-1)|[e_x(k-1)+\tfrac{1}{2}e_x(k-1)], && |S_x(k-1)| \geq 2S, \\ &= 2Se_x(k-1), && |S_x(k-1)| < 2S. \end{aligned} \tag{4}$$

The special region, $|S_x(k-1)| < 2S$, is needed to prevent a dead zone where $S_x(k)$ will be driven to zero.

Addition/Subtraction of DM Encoded Signals

Consider two signals, x(t) and y(t), both bandlimited to f_m and both DM encoded so that we only have the sequences $\{e_x(k)\}$ and $\{e_y(k)\}$ available. Then consider the problem of obtaining the sum of these two signals. We can perform arithmetic processing on $\{e_x(k)\}$ and $\{e_y(k)\}$ and form the direct sum, $a_D(k)$, as the sum of the individual signal estimates, i.e.,

$$a_D(k) = \hat{x}(k) + \hat{y}(k). \qquad (5)$$

Using the DM expression for the estimate of x(t) and y(t), Eq.(1c), we can form a recursive relationship for the direct sum,

$$a_D(k) = a_D(k-1) + S_x(k) + S_y(k), \qquad (6)$$

where $S_x(k)$ and $S_y(k)$ are formed directly from $\{e_x(k)\}$ and $\{e_y(k)\}$.

The direct sum, $a_D(k)$, is available in PCM format, because of the recursive relationship by which it was formulated, i.e., Eq. (6). To obtain the DM bit stream of the sum, $\{e_a(k)\}$, we merely pass $a_D(k)$ through a digital DM. A block diagram showing the structure used to obtain the sum of the DM encoded signals is presented in Fig. 2. The DM digital feedback circuit shown in this figure is, in general, constructed with the appropriate step size network followed by an accumulator. With this in mind, we notice that in order to physically realize the entire DM direct sum system, it requires only a full adder, an accumulator and the necessary step size network, all time shared.

To subtract DM encoded signals, we just add the negative of the subtrahend signal. If we wished to form x(t)-y(t), all we need do is change +Sy(k) to -Sy(k) in Eq. (6) and the result would be the direct difference. Thus, subtraction takes the same structure as the addition shown in Fig. 2.

It is to be noted that the structure derived is completely independent of the type of DM and is therefore universal for any digital DM. In order to realize the direct sum of signals encoded by a particular type of DM, we must construct the circuitry used to formulate the step size algorithm employed in the original DM encoder. For the modes cited above, the step size circuitry can be constructed using the standard digital hardware i.e., full adders, delays, scalers and Exclusive - OR gates. The DM adder for the linear mode and the Song audio mode is shown in Figs. 3 and 4, respectively.

Averaging Filter

The steady state response of all types of digital DMs to a constant input is an estimate signal which exhibits a periodic pattern. For a linear DM the estimate is a simple square wave with a period of two sampling intervals. For any Song mode DM the period is four sampling intervals. In Fig. 5 we show a typical steady state estimate signal for the Song audio mode. In this figure, the constant input, x, has been quantized to x_q and m is a non-negative integer.

Since the direct sum, $a_D(k)$, produced by a DM adder was formulated as the sum of the individual signal estimates, Eq. (5), we naturally expect the direct sum to also exhibit a periodic pattern when responding to constant inputs. For the Song audio mode, there are four possible steady state direct sum waveforms; one of which is given in Fig. 6. In this figure, n and r are both non-negative integers. The important property of the four possible steady state direct sum patterns is that the arithmetic average of any four consecutive values of $a_D(k)$ is always equal to $x_q + y_q$. This fact inspired the use of a four term non-recursive filter after $a_D(k)$.

The four term averaging filter employed is described by the following equation:

$$A(k) = \tfrac{1}{4}[a_D(k) + a_D(k-1) + a_D(k-2) + a_D(k-3)]. \qquad (7a)$$

If we apply Eq. (7a) to the waveform given in Fig. 6, the result is

$$A(k) = x_q + y_q \qquad (7b)$$

for all k, as long as $a_D(k)$ has reached the steady state. Thus it is seen that after a four term averaging filter, the DM sum produces the same result obtainable by PCM addition.

In order to illustrate the function of the four term averaging filter, we form the digital transfer function, H(z). Assuming zero initial conditions and taking the Z-transform of Eq. (7a), we obtain

$$H(z) = \frac{A(z)}{a_D(z)} = \tfrac{1}{4}(1 + z^{-1} + z^{-2} + z^{-3}). \qquad (8)$$

To display the frequency characteristics of this filter, we let $z = \exp(j\omega T)$ and form

$$H(\omega) = \exp(-j3\omega T/2)\cos(\omega T/2)\cos(\omega T), \qquad (9)$$

where

$$T = 1/f_s.$$

In Fig. 7 we plot $|H(\omega)| = |\cos(\omega T/2)\cos(\omega T)|$ on an abscissa normalized to f_s.

From Fig. 7, we observe that H(ω) has a zero at 1/4 f_s, 1/2 f_s and 3/4 f_s. It is

precisely the zero at 1/4 f_s that eliminates the four sampling interval periodic component produced by the Song mode DM. We also note that the four term averaging filter exhibits low pass filter characteristics since it slightly attenuates even baseband frequencies. Thus care must be taken in utilizing this filter, because it can introduce some distortion to baseband signals.

Multiplication of DM-Encoded Signals

If we wish to multiply x(t) and y(t) and if we only have the sequences $[e_x(k)]$ and $[e_y(k)]$ available, then we can form the direct product, $p_D(k)$, by using the product of the individual signal estimates, i.e.,

$$p_D(k) = \hat{x}(k)\hat{y}(k). \tag{10}$$

As is the case of the direct sum, we can develop a recursive relationship for the direct product,

$$p_D(k) = p_D(k-1) + S_y(k)\hat{x}(k-1) + S_x(k)\hat{y}(k-1) + S_x(k)S_y(k). \tag{11}$$

The basic block diagram showing the direct product, both in PCM format $[p_D(k)]$ and in DM format $[e_p(k)]$, is given in Fig. 8.

Since the direct product, $p_D(k)$, is formulated as the product of the individual signal estimates, as in the case of the direct sum, we expect the direct product to exhibit a periodic pattern when responding to constant inputs. Similar to the direct sum, the Song audio mode exhibits four possible steady state waveforms for the direct product. In Fig. 9 we show a typical steady state waveform. The constants c_1 and c_2 depend upon x_q and y_q and the amplitude of the steady state error pattern ($\hat{x}-x_q$ and $\hat{y}-y_q$), while d_1 and d_2 depend only on the latter of these two. Notice that the arithmetic average of any four consecutive values of $p_D(k)$ is always equal to $x_q y_q$ plus a second order term depending on S^2. This warrants the use of the following four term non-recursive filter after $p_D(k)$:

$$P(k) = \tfrac{1}{4}[p_D(k) + p_D(k-1) + p_D(k-2) + p_D(k-3)]. \tag{12a}$$

Applying Eq. (12a) to the waveform shown in Fig. 9, we see that

$$P(k) = x_q y_q + \tfrac{1}{2}(d_1+d_2)S^2 \tag{12b}$$

for all k, as long as $p_D(k)$ has reached the steady state. For any reasonably small value of step size, the term $1/2(d_1+d_2)S^2$ is negligible and thus, after the four term averaging filter, the DM product yields a result almost identical to the PCM product.

Returning to Fig. 8, we observe that although the structure derived for the direct product is again seen to be universal for any digital DM, it will only be useful if the step size algorithm is such that we can recursively realize the partial products, that is, the last three terms in Eq. (11). For the linear mode this is not difficult, and the direct product is seen to be

$$p_D(k) = p_D(k-1) + Se_y(k-1)\hat{x}(k-1) + Se_x(k-1)\hat{y}(k-1) + S^2e_x(k-1)e_y(k-1). \tag{13}$$

In Fig. 10 we show the block diagram for the DM multiplier for the linear mode. It is important to point out that the ease in realization of the direct product for this mode stems from the fact that there are no non-linear operations involved, only simple scaling and multiplication by +1 or -1.

In order to recursively realize all three partial products for the Song audio mode, we must use the following step size relationship which is a property of all types of DMs:

$$S_x(k) = |S_x(k)|e_x(k-1). \tag{14}$$

From Eq. (3), we see that this property is applicable for the Song audio mode as long as $|S_x(k-1)| \geq S$. Employing the step size relationship, the partial products for the Song audio mode can be expressed as:

$$S_y(k)\hat{x}(k-1) = S_y(k-1)\hat{x}(k-2)e_y(k-1)e_y(k-2) + S_x(k-1)S_y(k-1)e_y(k-1)e_y(k-2) + \hat{x}(k-1)Se_y(k-2), \tag{15}$$

$$S_x(k)\hat{y}(k-1) = S_x(k-1)\hat{y}(k-2)e_x(k-1)e_x(k-2) + S_y(k-1)S_x(k-1)e_x(k-1)e_x(k-2) + \hat{y}(k-1)Se_x(k-2), \tag{16}$$

$$S_x(k)S_y(k) = |S_x(k-1)S_y(k-1)|e_x(k-1)e_y(k-1) + S|S_x(k-1)|e_x(k-1)e_y(k-2) + S|S_y(k-1)|e_y(k-1)e_x(k-2) + S^2e_x(k-2)e_y(k-2). \tag{17}$$

It is easy to see that Eqs. (15), (16) and (17) can be constructed using adders, delays, scalers and Exclusive-OR gates. In Figs. 11, 12 and 13 we show the block diagrams of the partial products for the Song audio mode as specified in Eqs. (15), (16) and (17) respectively.

We can also construct a DM adder and multiplier for the Song video mode, as defined by Eq. (4). They are similar to the Song audio mode devices, since the step size algorithm takes on a comparable structure. Likewise, the step size relationship given by Eq. (14) is also applicable as can readily be seen from Eq. (4).

It is important to realize that all of the systems previously mentioned are accumulator or positive feedback type. For both the adder and the multiplier, for all DM modes, the present output is equal to the past output plus additional terms. Thus it is significant to begin with the correct initial condition for the past output, or else suffer a constant offset error. It is convenient to start with both signals, x(t) and y(t), at zero so that we can employ a zero initial condition for the past output.

Signal-To-Noise Ratio

The signal-to-noise ratio for the sum of two PCM encoded signals is well-known [2], and is given by

$$SNR_a(PCM) = 6(\sigma_x^2+\sigma_y^2)/S^2. \quad (18)$$

After nonrecursive averaging, the error in a DM encoded signal can be shown to be equal to that of PCM, and hence for a constant input,

$$SNR_a(DM) = SNR_a(PCM). \quad (19)$$

The signal-to-noise ratio of the product of two PCM signals is also well-known and is given by

$$SNR_p(PCM) = \frac{144\sigma_x^2\sigma_y^2}{12(\sigma_x^2+\sigma_y^2)S^2 + S^4}. \quad (20a)$$

If the two signals have equal power and the step size is small ($\sigma_x^2=\sigma_y^2=\sigma^2 \gg S^2$), the SNR becomes

$$SNR_p(PCM) \cong 6\sigma^2/S^2. \quad (20b)$$

For the direct product of DM encoded signals we can develop an expression for the signal-to-noise ratio. Again, we include the averaging filter introduced in Eq. (12a) and the DM error is formulated as

$$\xi_p = p - P, \quad (21a)$$

where

$$P = P(k) = x_q y_q + \delta S^2. \quad (21b)$$

Let us now define the product signal-to-noise ratio for DM signals to be

$$SNR_p(DM) \equiv \overline{p^2}/Var(\xi_p). \quad (22)$$

Since $\overline{p^2} = \sigma_x^2\sigma_y^2$, we evaluate

$$\overline{\xi_p} = \delta S^2, \quad (23a)$$

$$\overline{\xi_p^2} = \overline{\epsilon_p^2} + \delta^2 S^4, \quad (23b)$$

and

$$Var(\xi_p) = \overline{\xi_p^2} - \overline{\xi_p}^2 = \overline{\epsilon_p^2}. \quad (23c)$$

But this means that

$$Var(\xi_p) = Var(\epsilon_p), \quad (23d)$$

and

$$SNR_p(DM) = \frac{144\sigma_x^2\sigma_y^2}{12(\sigma_x^2+\sigma_y^2)S^2 + S^4}. \quad (24a)$$

Again taking $\sigma_x^2 = \sigma_y^2 = \sigma^2$ and $\sigma^2 \gg S^2$, this result becomes

$$SNR_p(DM) \cong 6\sigma^2/S^2. \quad (24b)$$

As in the case of addition, the conclusion is that the performance of the direct PCM multiplier followed by the averaging filter is identical to the PCM multiplier performance for constant input signals.

Simulation Results

A computer simulation has been completed for the direct arithmetic processing of DM encoded signals, and results have been obtained for both the direct sum and the direct product employing the Song audio mode DM. Both the direct sum and the direct product were passed through the nonrecursive four term averaging filter mentioned previously. The sum results are shown in Figs. 14 thru 16 and the product results are shown in Figs. 17 thru 19. In all simulation results both the step size and the sampling period have been normalized to unity.

Fig. 14 shows the sum of a step function and a pulse; Fig. 15 displays the result of adding a step function and a sinusoid; and Fig. 16 gives the addition of two sinusoids with the same amplitudes and frequencies that are in phase. In Figs. 17, 18 and 19 we show the product of the signals that were added in Figs. 14, 15 and 16 respectively. These simulation results verify the theory developed for both the DM Direct Sum and the DM direct product. We must particularly emphasize the role played by the four term averaging filter to achieve both sum and product results that are so accurate. To fully appreciate the effect of this four term averaging, we show in Fig. 20 the direct product of a step and a pulse without the four-term averaging. Comparing this with Fig. 17, which is the result after four term averaging, clear demonstrates the important role played by this filter.

As a concluding remark, we observe an interesting by-product of the DM multiplier. In Fig. 21 we show the response of a Song audio mode DM to a step of amplitude 150. The response time, needed to reach at least 150, is seen to be 17 sampling periods. Notice, from Fig. 18, that for the multiplier to reach an amplitude of 150 it takes only 8 sampling periods. Thus we have expanded the bandwidth by a factor of two, consistent with the previous assumption of the multiplication process. This type of system may be useful when processing video signals that are characterized by abrupt, step-like amplitude variations.

References

[1] C. L. Song, J. Garodnick and D. L. Schilling, "A Variable Step Size Robust Delta Modulator," IEEE Trans. Commun. Tech., Vol. Com.-19, pp. 1033-1044, Dec. 1971.

[2] J. M. Wozencraft and I. M. Jacobs, Principles of Communication Engineering, N. Y.: John Wiley & Sons, Inc., 1967, pp. 674-676.

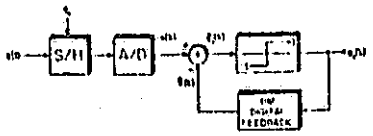


Fig 1 Basic Form of a Digital DM

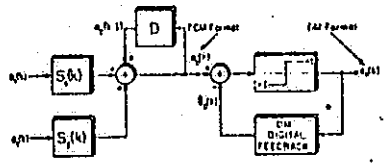


Fig 2 Direct Sum of DM Encoded Signals

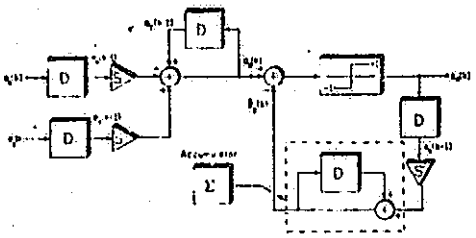


Fig 3 DM Adder for Linear Mode

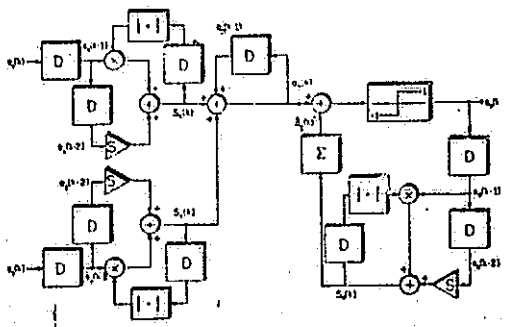


Fig 4 DM Adder for Song Audio Mode

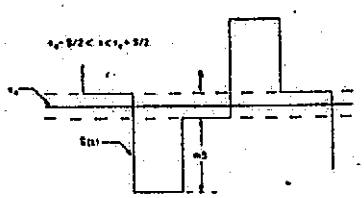


Fig 5 Steady State Estimate Signals for the Song Audio Mode DM for a Constant Input U

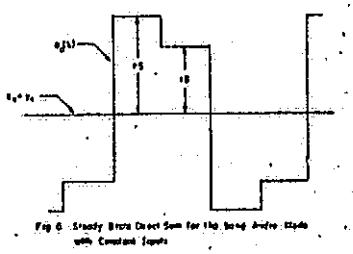


Fig 6 Steady State Direct Sum for the Song Audio Mode with Constant Inputs

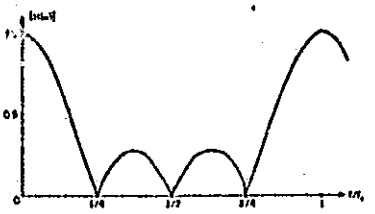


Fig 7. Amplitude-Frequency Characteristics of the Four Term Non-recursive Averaging Filter

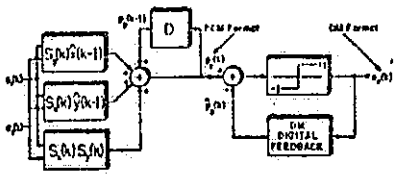


Fig 8 Direct Product of DM Encoded Signals

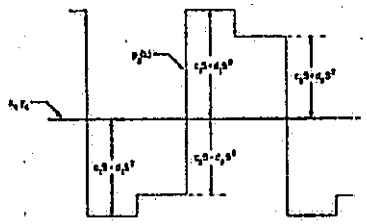


Fig 9. Steady State Direct Product for the Song Audio Mode with Constant Inputs

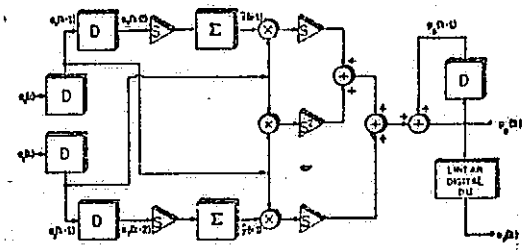


Fig 10. - DM Multiplier for Linear Mode

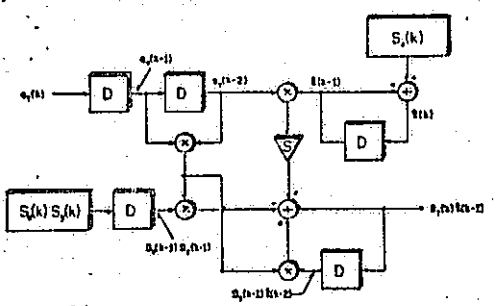


Fig 11. Realization of $S_y(k)\hat{y}(k-1)$ for Song Audio Mode

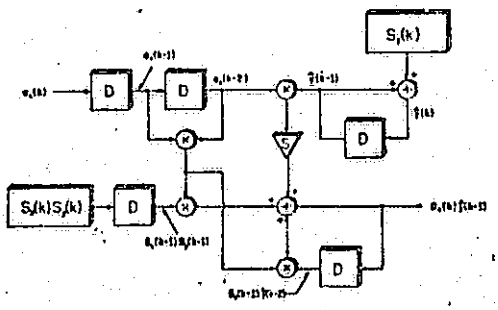
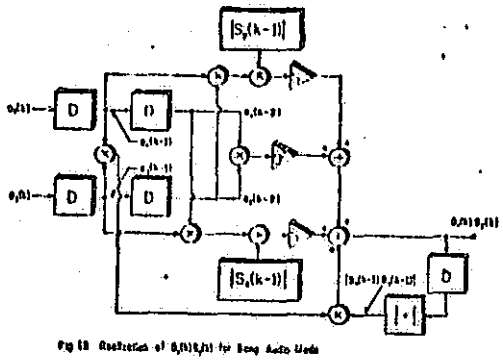
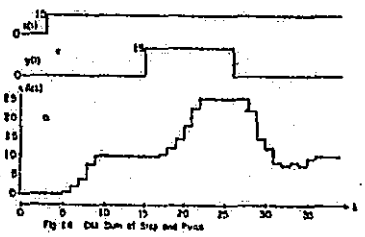
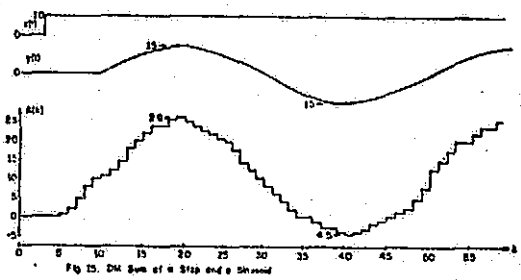
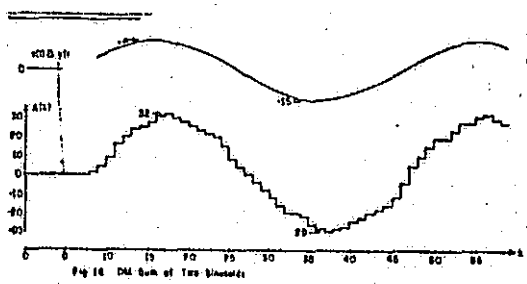


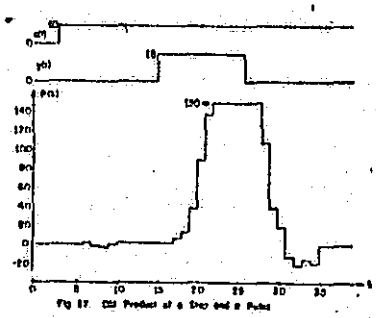
Fig 12. Realization of $S_x(k)\hat{x}(k-1)$ for Song Audio Mode

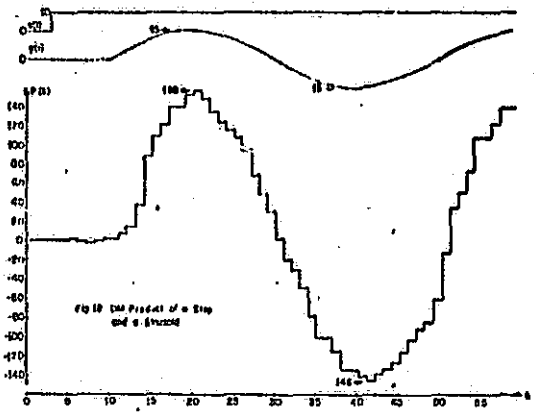

D
D
D
D
D

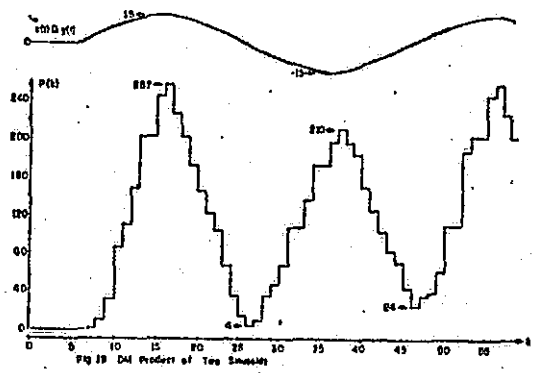


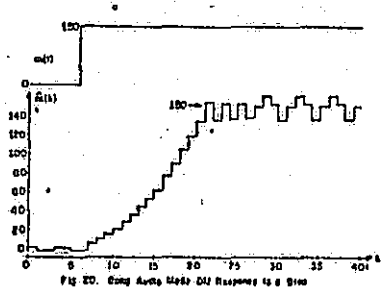


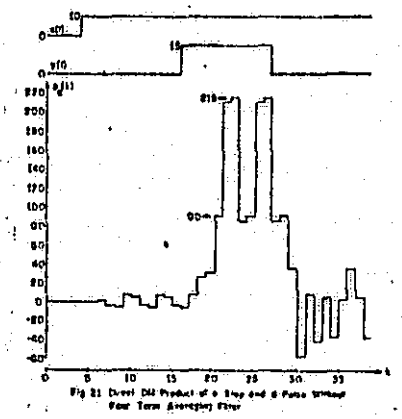












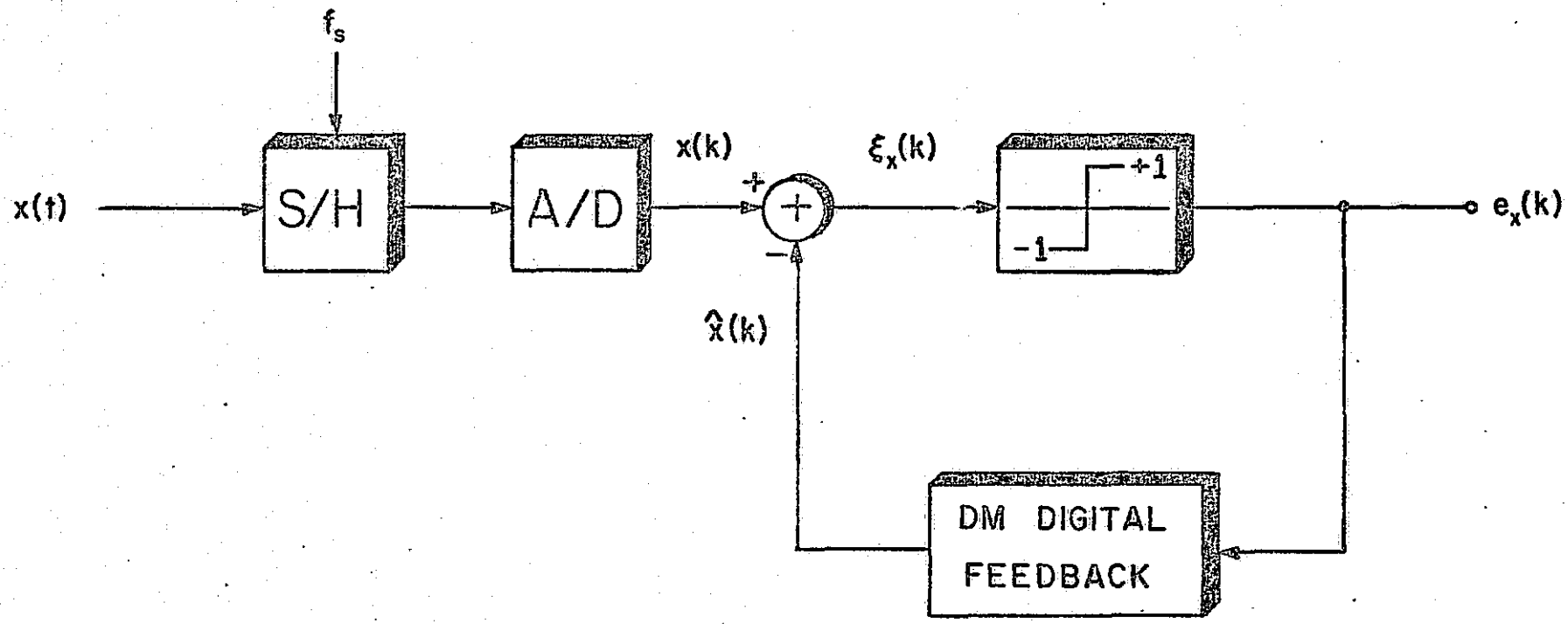


Fig. 1. General Form of an All-Digital DM

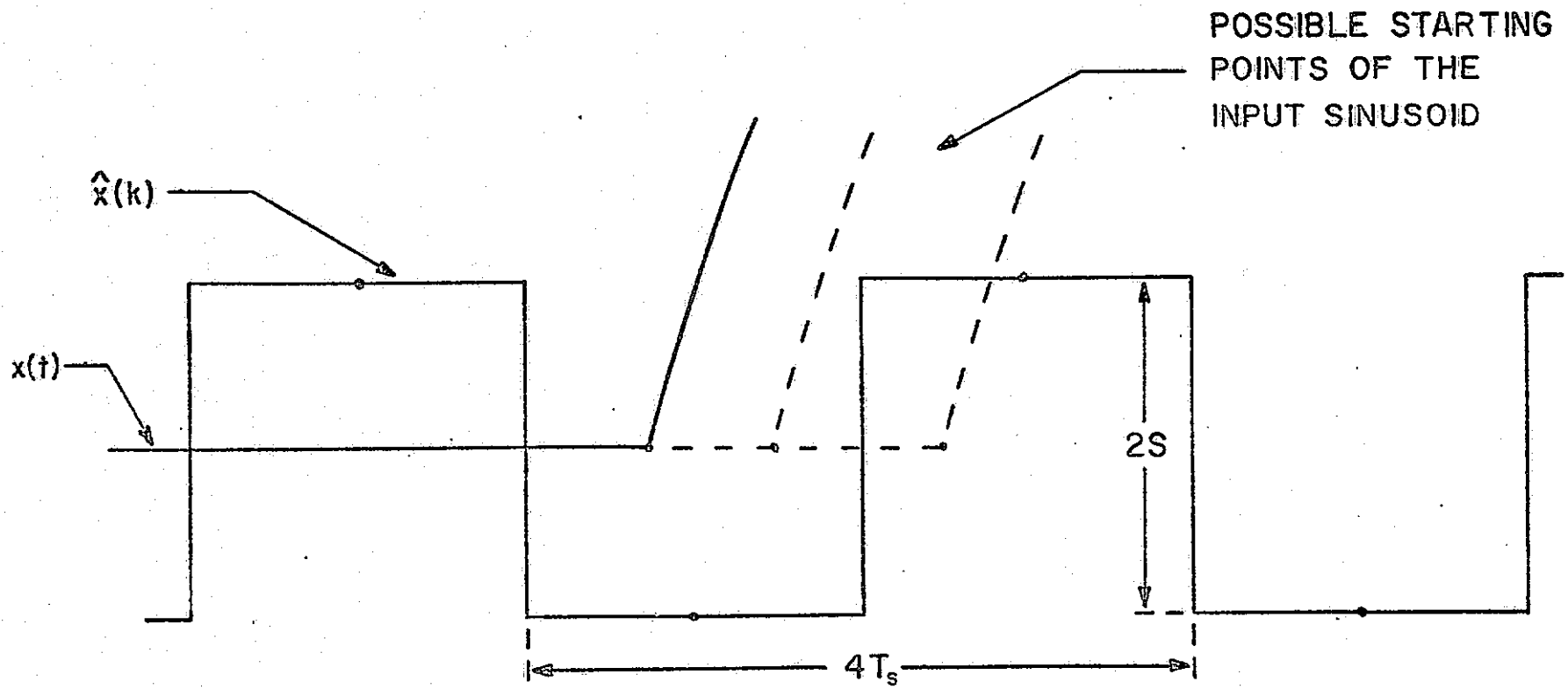


Fig. 2. DM Response To A Constant

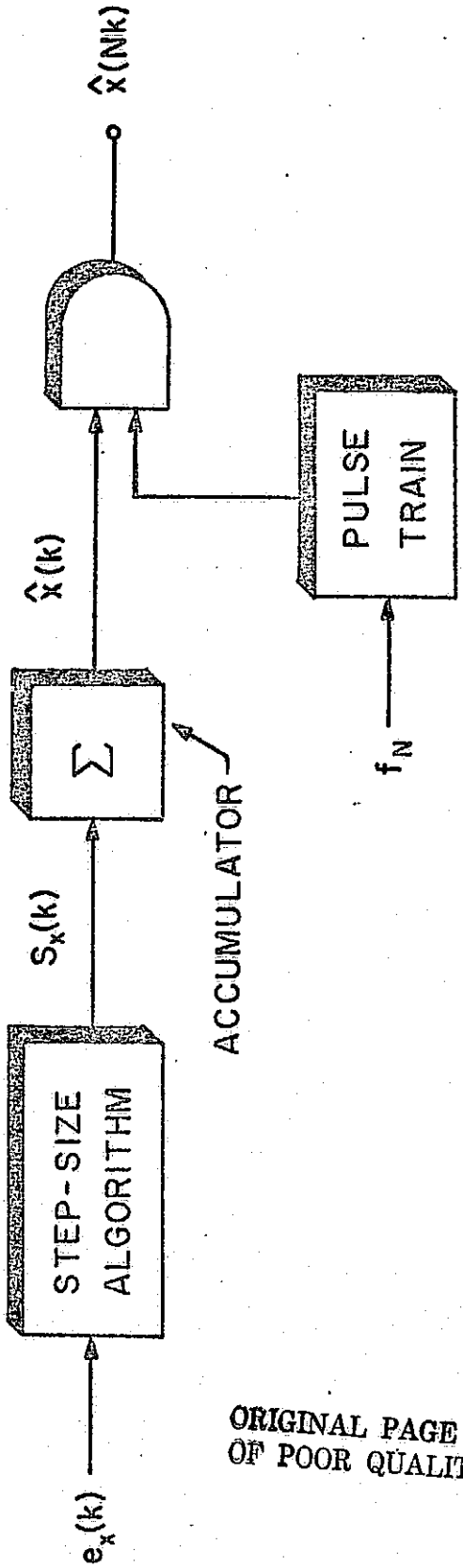


Fig. 3. Basic DM To PCM Converter

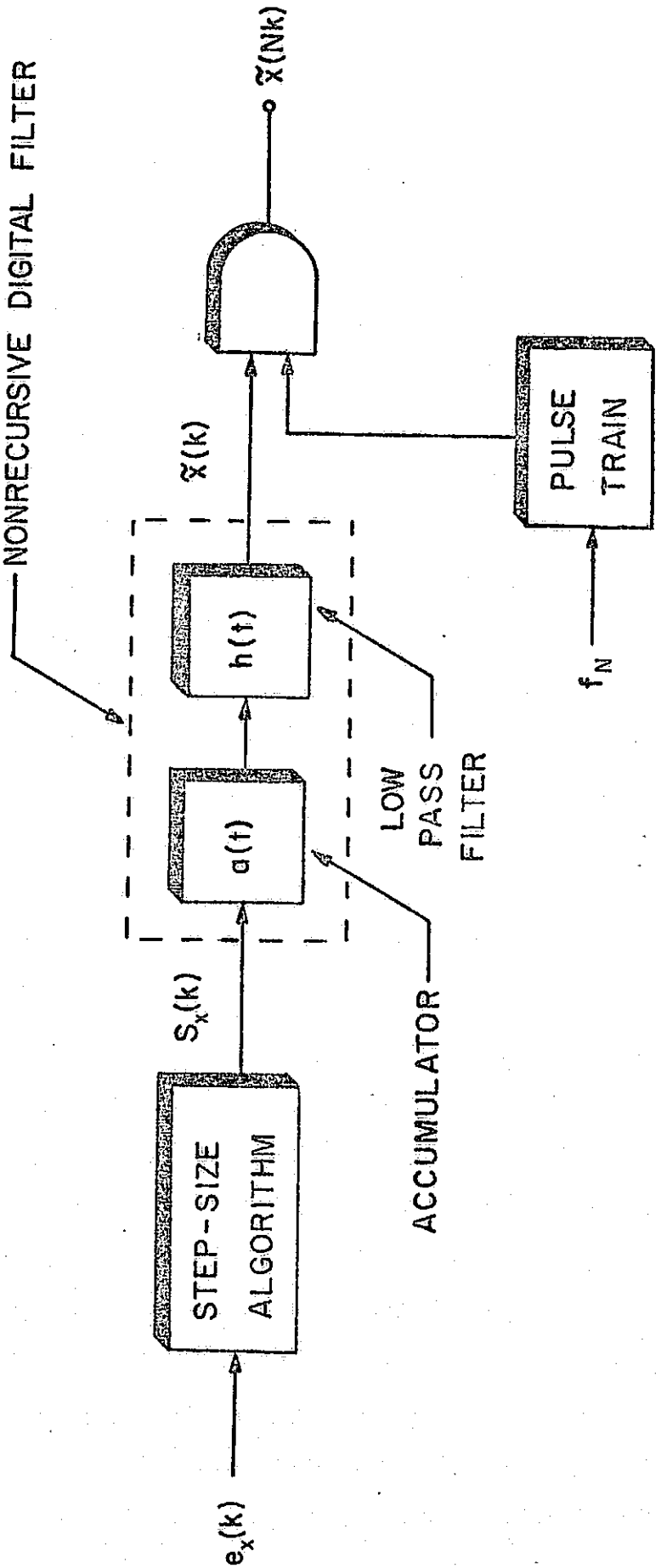


Fig. 4. DM To PCM Converter With Filter

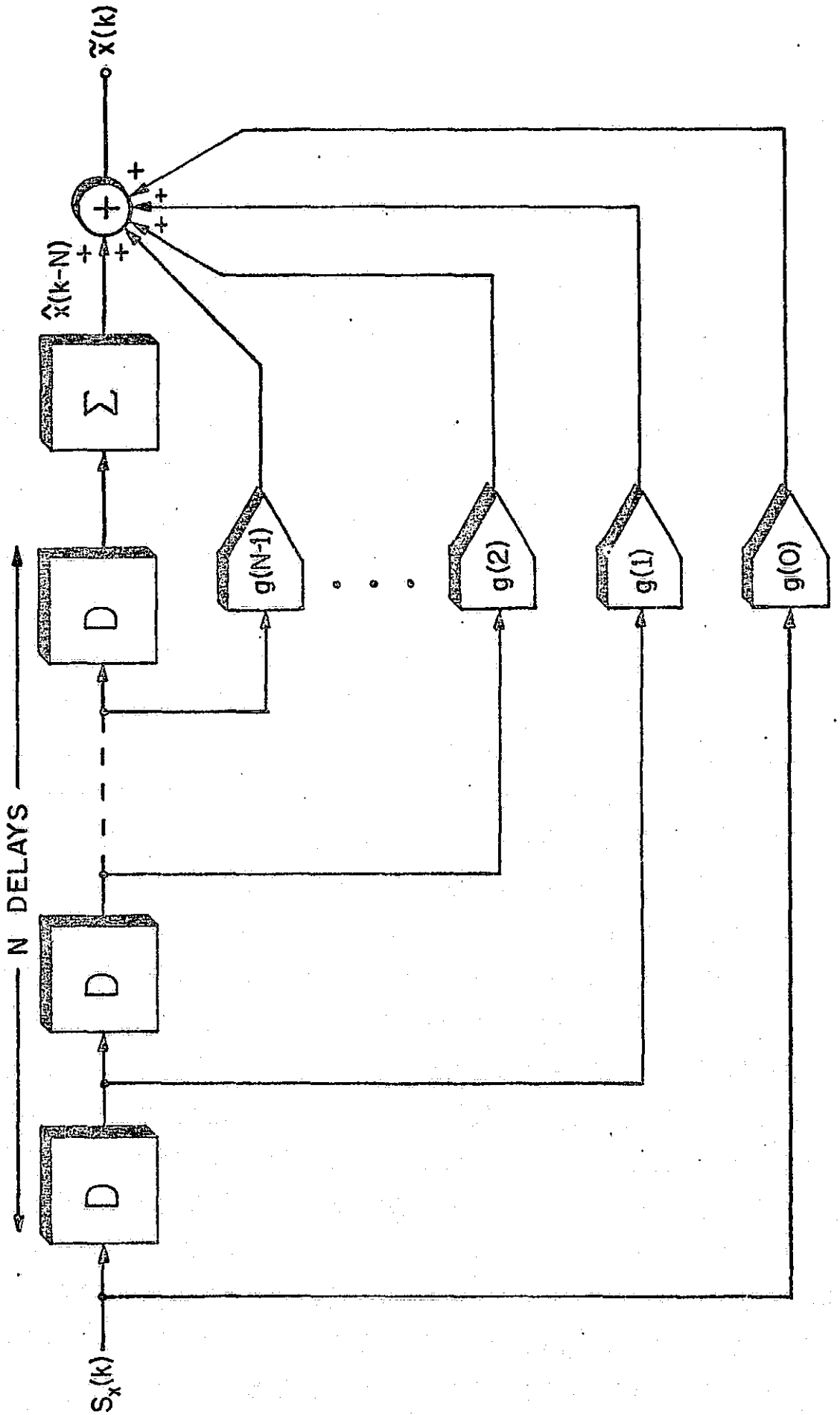


Fig. 5. Nonrecursive Digital Filter

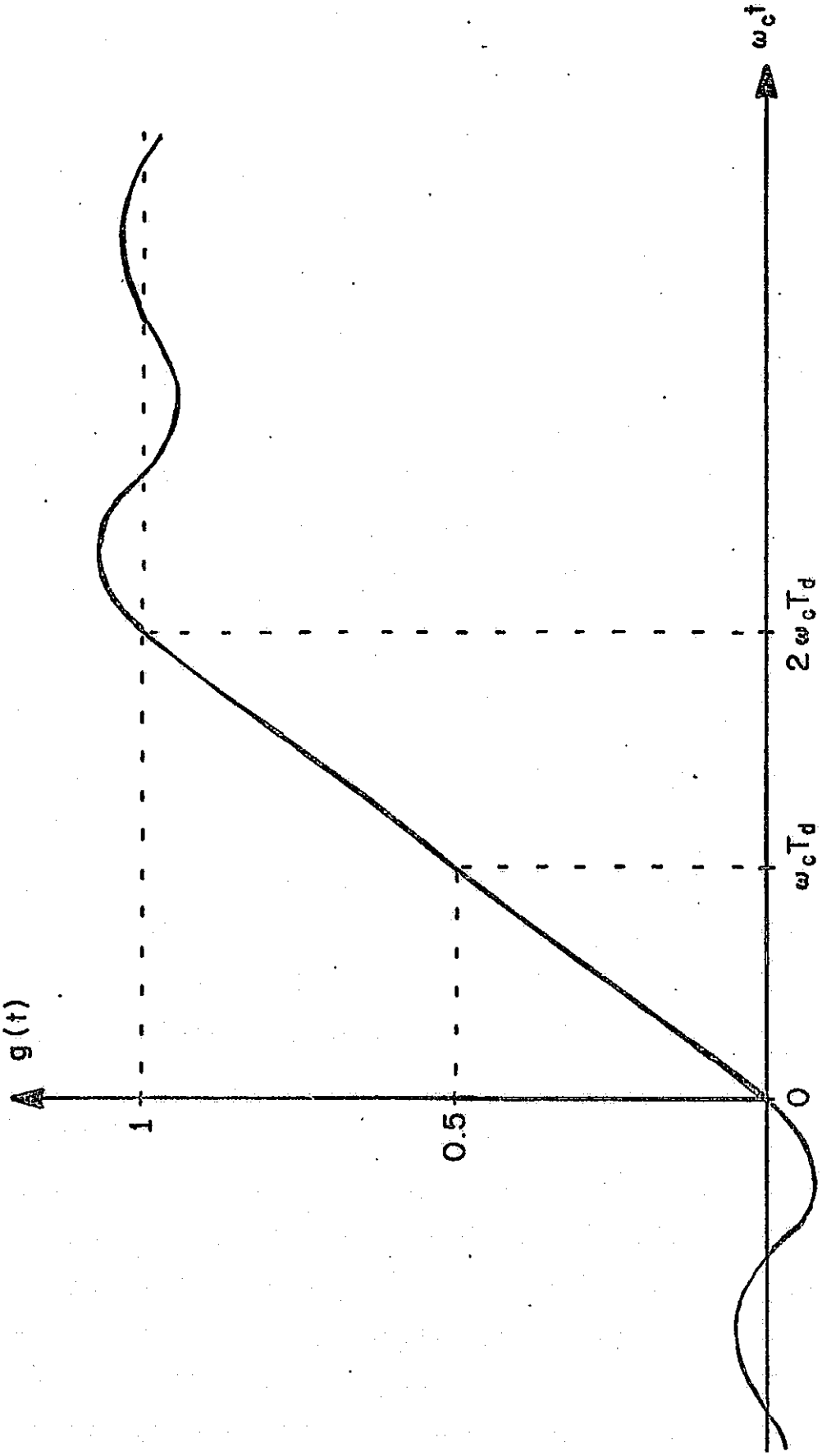


Fig. 6. Unit Step Response Of An Ideal Low Pass Filter

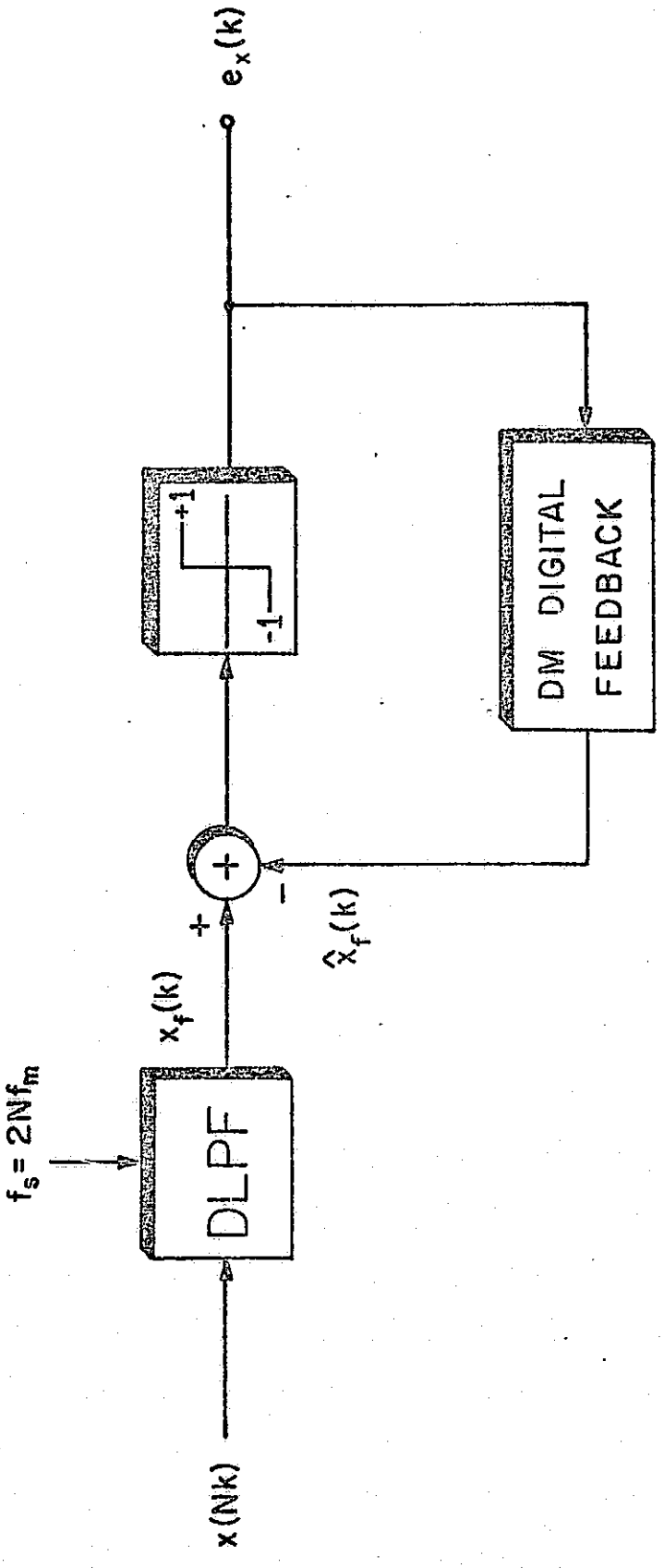


Fig. 7. PCM To DM Converter

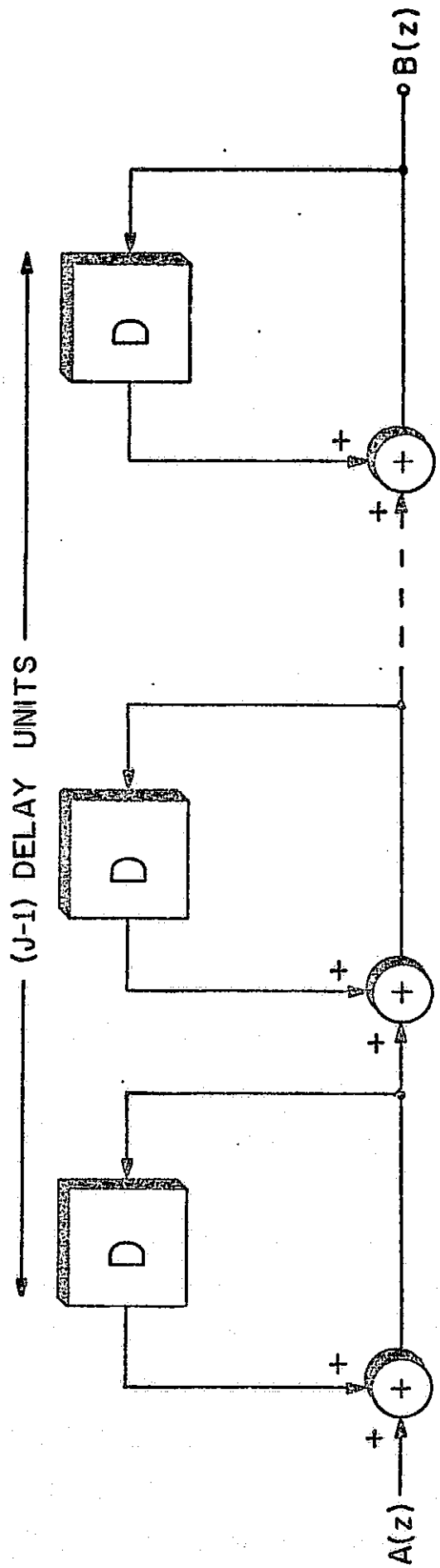


Fig. 8. J Period Digital Hold Circuit

I. 4. A Phase Locked Loop with Nonlinear Processor

A. Introduction

A general configuration for a Digital Phase Locked Loop (DPLL) is shown in Fig. 1. The input signal, f(t), is a bandlimited (B Hz) angle modulated waveform (i.e., FM, PM, FSK, PSK, etc.) of nominal carrier frequency f_o, plus an additive noise signal, n(t). $f(kT_s)$ is the digitized version of f(t), where T_s is the sampling period, and $f_v(kT_s)$ is the Voltage Controlled Oscillator (VCO) waveform in digital format. The Phase Detector (PD) extracts the difference frequency term from the product $f(kT_s) \cdot f_v(kT_s)$. This signal is then hard limited, scaled, and then processed to produce the loop output signal, $y(kT_s)$.

Since in a Phase Locked Loop (PLL) we are interested in extracting the phase of the input signal, we can consider the use of a Phase Controlled Oscillator (PCO) as a feedback device. The frequency information would then be the phase difference $[\varphi_v(kT_s) - \varphi_v(kT_s - T_s)] / \Delta T$.

Let us assume that the form of the input signal is some phase modulated signal. The digital version of this signal (assuming no additive noise is present) can be shown to be of the form

$$f(kT_s) = -2 \cos [(k\pi T_s/2j) + \varphi_m(kT_s)] \tag{1}$$

where j is an integer greater than zero and $\varphi_m(kT_s)$ is the modulating signal.

The VCO waveform will be of the form

$$f_v(kT_s) = g [(kT_s \pi/2j) + \varphi_v(kT_s)], \tag{2}$$

where $\varphi_v(kT_s)$ is the estimate of $\varphi_m(kT_s)$ and $g(\cdot)$ is a function defined as follows:

$$g(x) = Sq(x) = \begin{cases} +1 & 0 \leq x < \pi \\ -1 & \pi \leq x < \pi \end{cases} \tag{3a}$$

and

$$g(x + 2\pi) = g(x). \tag{3b}$$

Also, the VCO phase is given by

$$\varphi_v(kT_s) = G_{VCO} \cdot \sum_{i=-\infty}^{k-1} y(iT_s), \tag{4}$$

where $y(iT_s)$ is the output of the algorithmic processor at the time instant $t = iT_s$. At this point we will normalize T_s to unity for simplicity unless otherwise noted.

B. Linear Delta Modulator (DM) Type Processer

1. System Characteristics

Consider the following digital filter to be used as the algorithmic processor. The new estimate, $y(k)$, will be equal to the old estimate plus some update data, $b(k)$, as in a linear Delta Modulator (DM),

$$y(k) = y(k-1) + b(k) \tag{5a}$$

$$b(k) = g_d \, Sgn\,[\, e(k) \,] \tag{5b}$$

$$e(k+1) = (f(k) \cdot g(k))_{LPF}. \tag{5c}$$

In the above, g_d is some scaling factor and $(x)_{LPF}$ has the following connotation. Since $g(x) = Sq(x)$ is a square wave, it can be expanded in a Fourier series to obtain

$$Sq(x) = \frac{4}{\pi} \left(\sin x + \frac{1}{3} \sin 3x + \cdots + \frac{1}{(2n+1)} \sin (2n+1)\, x + \cdots\right), \tag{6}$$

where n is a positive integer. Therefore the product

$2 \cos [\, (k\pi T_s/2j) + \varphi_m(k) \,] \cdot Sq [\, (k\pi T_s/2j) + \varphi_v(k) \,]$ generates

harmonics at $f/f_s = 0,\ 2\pi/2j,\ 4\pi/2j,\ \cdots\ 2l\pi/2j$, where l is an integer. If we extract the signal content around f = 0, i.e., we Low Pass Filter (LPF) this signal, we obtain the desired error signal. This is precisely what the PD does.

From Eq. (4) we have

$$\varphi_v(k) - \varphi_v(k-1) = G_{VCO}\, y(k-1) \tag{7}$$

and so

$$\varphi_v(k+1) - 2\varphi_v(k) + \varphi_v(k-1) = G\ \mathrm{sgn}\left[\ \sin\left(\varphi_m(k-1) - \varphi_v(k-1)\right)\ \right] \tag{8}$$

where $G = (4/\pi)\, g_d\, G_{VCO}$. Equation (8) is a second order difference equation of the loop phase estimate $\varphi_v(k)$. Notice that at any sampling instant, k, the Right Hand Side (RHS) is $\pm G$.

Let us assume for a moment that $\varphi_m = \varphi_o$, a constant phase offset. Then, of course, one would like $\varphi_v(k)$ to be a constant approximately equal to φ_o since $\varphi_v(k)$ is an estimate of $\varphi_m(k)$. In particular one would want $\varphi_v(k) = \varphi_o$ (modulo 2π). Consider Eq. (8) without the sgn function present. If $\varphi_v(k)$ is a constant then the Left Hand Side (LHS) of Eq. (8) is zero which implies that $\sin\left[\ \varphi_m(k) - \varphi_v(k)\ \right] = 0$ or that $\varphi_m(k) - \varphi_v(k) = 0$ (modulo 2π) in steady state. However, in the actual equation one notices that if $\varphi_v(k)$ is a constant then the LHS of Eq. (8) is zero while the RHS is $\pm G$. This is clearly a contradiction. Instead what happens is the following. In the steady state lock condition the phase estimate $\varphi_v(k)$ will oscillate by a fixed amount, $\Delta\varphi$, about some quiescent value (close to φ_o). This can be shown as follows.

Let the difference between the input and VCO phase be the phase error $\phi_e(k)$, i.e.,

$$\phi_e(k) = \varphi_m(k) - \varphi_v(k). \tag{9}$$

We can then rewrite Eq. (8) as follows:

$$\phi_e(k+1) - 2\phi_e(k) + \phi_e(k-1) = -G \operatorname{sgn}\{\sin \phi_e(k-1)\} + [\varphi_m(k+1) - 2\varphi_m(k) + \varphi_m(k-1)] \tag{10a}$$

If $\varphi_m = \varphi_o$ as above, the bracketed term in Eq. (10a) is zero, therefore we have

$$\phi_e(k+1) - 2\phi_e(k) + \phi_e(k-1) = -G \operatorname{sgn}\{\sin \phi_e(k-1)\}. \tag{10b}$$

Let us assume that a possible solution to the above is

$$\phi_e(k) = -G_o \operatorname{sgn}[\sin \phi_e(k-1)], \tag{10c}$$

where $0 < G_o < \pi/2$. Also assume that $\phi_e(k-1) = G_o$, so $\phi_e(k) = -G_o$, and $\phi_e(k+1) = +G_o$. Using the above for $\phi_e(k-1)$ we see from Eq. (10b) that $\phi_e(k+1) = -G - 3G_o$. For $\phi_e(k+1)$ to be $+G_o$ and thus satisfy Eq. (10c), G must be equal to $-4G_o$ or $G_o = -G/4$. So $\phi_e(k)$ will oscillate about zero by $\pm G/4$.

2. Spike Suppression

Consider the above system operating with a PCO. The PD will have a binary characteristic taking on values of ± 1. Let us assume we want to track a phase signal or demodulate a phase modulated signal which is varying linearly with time (i.e., a frequency offset). Since we are operating in a noisy environment, we would like to investigate the loop's spike suppression capabilities. To do this we will look at the loop's response to a ramp input (a crude approximation of a noise spike).

For a PCO in the feedback path the phase error difference equation becomes,

$$\phi_e(k) - \phi_e(k-1) = -G_p \operatorname{sgn}[\sin\phi_e(k-1)] + \varphi_m(k) - \varphi_m(k-1). \quad (11)$$

where $G_p = (4/\pi)\, g_d\, G_{pco}$ and G_{pco} is the PCO gain. Furthermore, let us set $G_p = S$, the normalized step size of the DM ($S = S''/T_s$).

Let us assume we wish to track a constant phase, $\varphi_m(k) = \varphi_o = 0$, and that at a time instant $k = 0$ a spike of noise occurs which changes $\varphi_m(k) = \varphi_o + 2\pi = 2\pi$ in a linear manner as shown in Fig. 3. (Notice: this is an analog representation of the quantized input and is used for clarity). Assume initially that the system is in steady state, i.e., the loop is tracking the carrier with zero frequency error. The PCO phase will approximate the input phase and differ by a value less than $|S|$. At time instant $k = 0$, the input phase begins to rise at a rate of T_R rad/sec until it reaches 2π radians, that is,

$$\varphi_m(k) = \begin{cases} T_R k & , \quad k \le 2\pi / T_R \\ 2\pi & , \quad k \le \lceil 2\pi / T_R \rceil \end{cases}, \quad (12)$$

where $\lceil x \rceil$ denotes the least integer greater than x.

Since we do not wish to follow the spike, the PCO phase should not increase by 2π radians. Instead, since we wish to follow the input signal, there must be a time (denoted as the "Turnaround Time"), K_o, for which the estimate will decrease and eventually relock to $\varphi_o = 0$. Due to the binary nature of the PD and the use of a linear DM, $\varphi_v(k)$ will increase by one step size each time instant until $\phi_e(k)$ is greater than π (but less than 2π), at which time $\varphi_v(k)$ will decrease ($b(k) = -1$). Thus it is seen that the PCO phase is a spike and so the time derivative of the phase is a doublet thereby suppresing the spike and improving the SNR (1). For $k \ge K'$ (see Fig. 3) the signal estimate becomes periodic and tracks the

original signal level, $\varphi_o = 0$.

The situation indicated in Fig. 3 will occur for $\pi \leq \phi_e(k) < 2\pi$ when $k > K_o$ so we have

$$\frac{\pi + \varphi_v(0) - S}{T_R - S} \leq K_o < \frac{2\pi + \varphi_v(0) - S}{T_R - S}, \tag{13}$$

where $T_R > S$ and $\varphi_v(0)$ is the phase estimate at $k = 0$. (The reason for $T_R > S$ is obvious from Fig. 3, and is equivalent to a slope overload condition in an ordinary linear DM. If this condition were not so, turn-around would not occur.) The above equation specifies the range of turnaround time as a function of step size and spike rise time.

It has been shown (2) that T_R (spike) $\leq 1/B_{IF}$ where $B_{IF} = 2(\beta+1)f_m$ is the bandwidth of the IF filter in the receiver front end. This factor gives the designer an idea of T_R (i.e., and upper limit) and so in conjunction with some other criteria (e.g., a minimum step size or turn-around time) he can find the best parameters for his system.

C. Song Audio and Video Mode DM

Two other types of DM processors are presently being investigated for use as the algorithmic processors. The Song Audio Mode is governed by the following set of equations

$$y(k) = y(k-1) + \Delta y(k) \tag{14a}$$

and

$$\Delta y(k) = |\Delta y(k-1)| \, b(k-1) + S\, b(k-2) \tag{14b}$$

and the Song Video Mode by

$$y(k) = y(k-1) + \Delta y(k) \tag{15a}$$

and

$$\Delta y(k) = \begin{cases} |\Delta y(k-1)|\, b(k-1) + \frac{1}{2} b(k-2); & |\Delta y(k-1)| \geq 2S \\ 2S\, b(k-1) \quad ; & |\Delta y(k-1)| < 2S \end{cases} \tag{15b}$$

The Song Audio mode is characterized by a quadratic increase in step size and the Song Video mode by an exponential rise. Proceeding in a manner similar to the linear DM, we find, for the Song Audio mode, that

$$S K_o^2 + (S - 2T_R) K_o + 2\pi + 2\varphi_v(0) \leq 0 \tag{16a}$$

and

$$S K_o^2 + (S - 2T_R) K_o + 4\pi + 2\varphi_v(0) > 0. \tag{16b}$$

Similarly for the Video mode we have

$$\pi < T_R K_o + \{ 4S [\, 1 - (1.5)^{K_o} \,] - \varphi_v(0) \} < 2\pi. \tag{17}$$

D. Signal Plus Noise

Let us now consider receiving an FM signal plus additive white Gaussian noise as in Fig. 1. After IF bandpassing the input signal and obtaining the digital format we have

$$f_n(k) = -2 \cos [\, (k\pi/2j) + \varphi_m(k) \,] + n(k), \tag{18}$$

where $f_n(k)$ is the digital information signal plus digitized noise, $n(k)$.

If we write n (k) in its quadrature form as

$$n(k) = n_1(k) \cos(k\pi/2j) - n_2(k) \sin(k\pi/2j) \tag{19}$$

and assume a high input SNR, we can write

$$f_n(k) = -2 \cos[(k\pi/2j) + \varphi_m(k) - n_2'(k)] \tag{20}$$

where $n_2'(k) = n_2(k)/2$.

Using a PCO and a linear DM we obtain

$$\phi_e(k) = \phi_e(k-1) + G_p \operatorname{sgn}\{\sin \phi_e(k-1) - n_2'(k-1)\} + \varphi_m(k) - \varphi_m(k-1). \tag{21}$$

Letting $H(k) = \phi_e(k) - n_2'(k)$ (22a)

we have

$$H(k) - H(k-1) = -G_p \operatorname{Sgn}[\sin H(k-1)] - n_g(k) + \varphi_m(k) - \varphi_m(k-1), \tag{22b}$$

where $n_g(k) = n_2'(k) - n_2'(k-1)$ and is a Gaussian noise process. For $\varphi_m(k) = \varphi_o$, $\Delta\varphi_m(k) = \varphi_m(k) - \varphi_m(k-1) = 0$ for all k. If we assume that the noise samples are independent, then under the above conditions we see that Eq. (22b) becomes a first order Gauss Markov (discrete) process, and so we can apply a discrete version of the Chapman - Kolmogorov equation to determine a steady state pdf, i.e., the pdf of H (k) conditioned on H (k-1) (3). The equation for the pdf is

$$p_{k+1}(H \mid H_o) = \int_{-\infty}^{\infty} q_k(H \mid z)\, p_k(z \mid H_o)\, dz, \tag{23}$$

where

$H_o = \overline{H(0)} = \bar{\phi}_e(0)$

$p_k(H \mid H_o)$ = pdf of H (k) conditioned on H_o

$q_k(H \mid z)$ = transition pdf of H (k+1) given H (k) = z.

Noting that $n_g(k)$ is independent of H (k), we see from Eq. (22b) that the transition pdf $q_k(H \mid z)$ is Gaussian with

$$\text{mean} = E_k(H \mid z) = z - G_p \operatorname{sgn}(\sin z) \tag{24a}$$

and

$$\text{variance} = \text{Var}(H \mid z) = 2\sigma^2_{n_2'} = \sigma^2/2 = \sigma^2_g, \tag{24b}$$

where $\sigma^2_{n_2'}$ is the variance of n_2' and σ^2 is the variance of the original noise process n(k). Notice that the mean and variance are independent of the time parameter k. Thus we have

$$p_{k+1}(H \mid H_o) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2_g}} \cdot \exp\left[-(H - z + G_p \operatorname{sgn}(\sin z))^2/2\sigma^2_g\right] \cdot p_k(z \mid H_o)\, dz. \tag{25}$$

The range of ϕ_e, and therefore of H, that we are interested in is generally $[-\pi, \pi]$. In Eq. (25) the range is $(-\infty, \infty)$. A simple adjustment (4) results in the following

$$P_{k+1}(H \mid H_o) = \int_{-\pi}^{\pi} K(H, z)\, P_k(z \mid H_o)\, dz, \tag{26}$$

where

$$P_k(H \mid H_o) = \sum_{n=-\infty}^{\infty} p_k(H + 2n\pi \mid H_o), \tag{27a}$$

$$P_o(H \mid H_o) = \delta(H - H_o) \tag{27b}$$

and

$$K(H, z) = \sum_{n=-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_g^2}} \exp\left[-(H + 2n\pi - z + G_p \operatorname{sgn}(\sin z)^2/2\sigma_g^2\right] \tag{28}$$

so that $P_k(H \mid H_o)$ is periodic modulo 2π.

As $k \to \infty$, we look for a steady state pdf P(H). P(H) would then be obtained as the solution to the integral equation

$$P(H) = \int_{-\pi}^{\pi} K(H, z)\, P(z)\, dz. \tag{29}$$

The search for an analytic solution of the above as well as numerical analysis on a digital computer are presently under way. Other items of interest to investigate are mean time to slip a cycle, location of threshold, and threshold extension. These problems are being investigated presently. Similar studies are proposed for the Song Audio and Video processor algorithms.

References

(1) H. Taub and D. L. Schilling, Principles of Communication Systems, New York: McGraw Hill, 1971 pp 326-329.

(2) I bid, pp 329.

(3) J. L. Doob, Stochastic Processes, New York: Wiley, 1953, pp 195-198.

(4) A. Viterbi, Principles of Coherent Communication, New York: McGraw-Hill, 1956 pp 88-89.

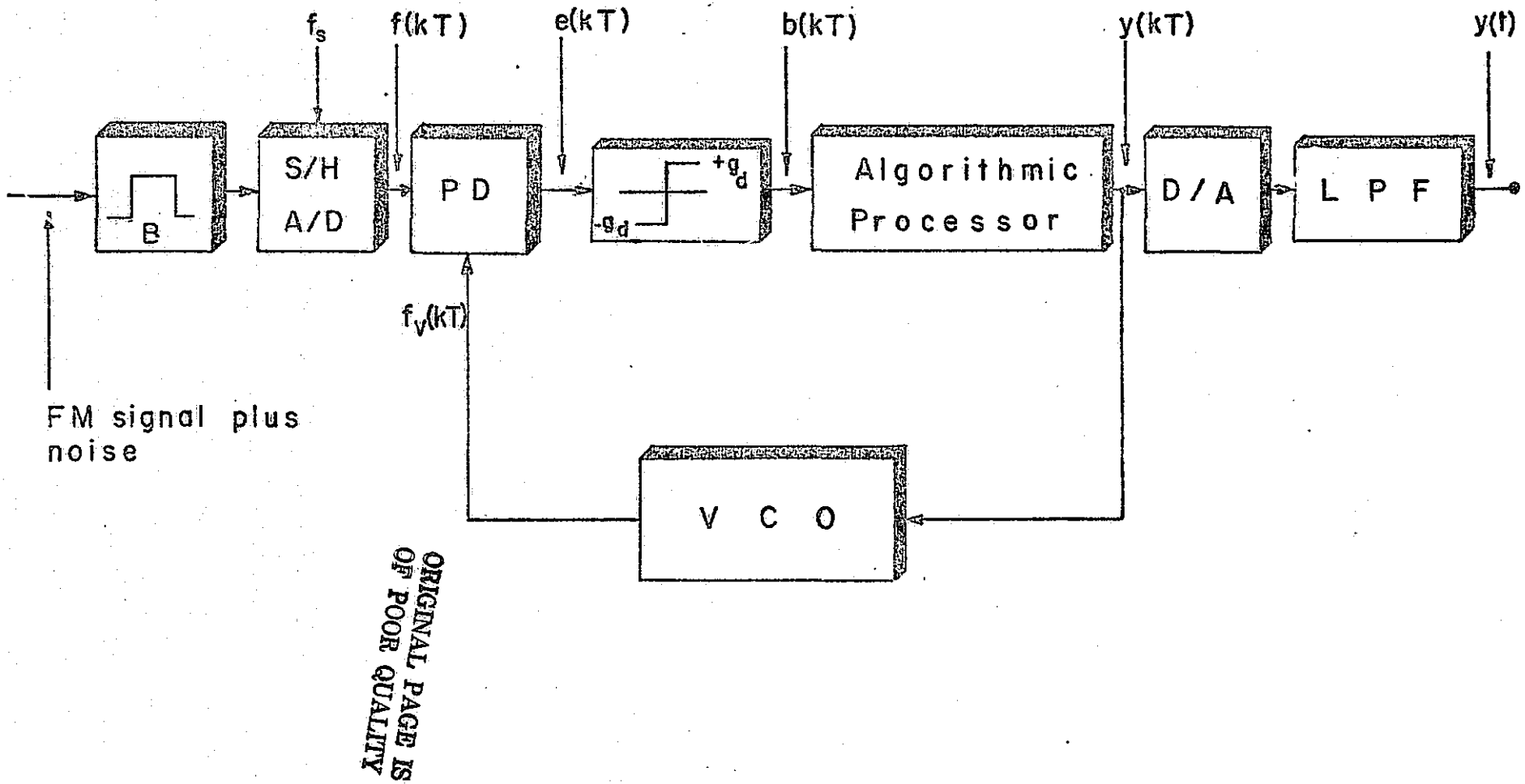


Figure 1. A General DPLL with Nonlinear Processor

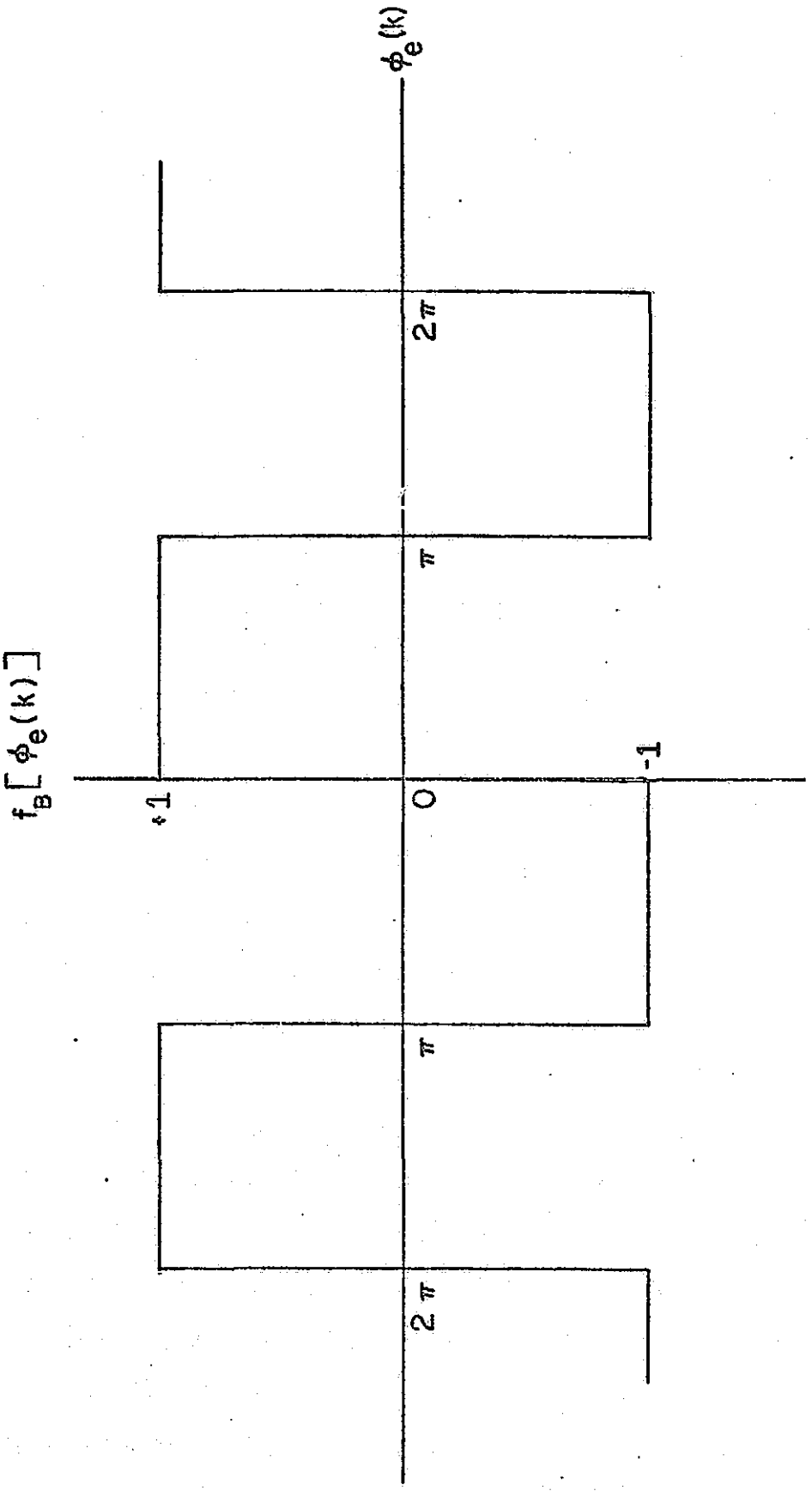


Figure 2. Binary Phase Characteristic

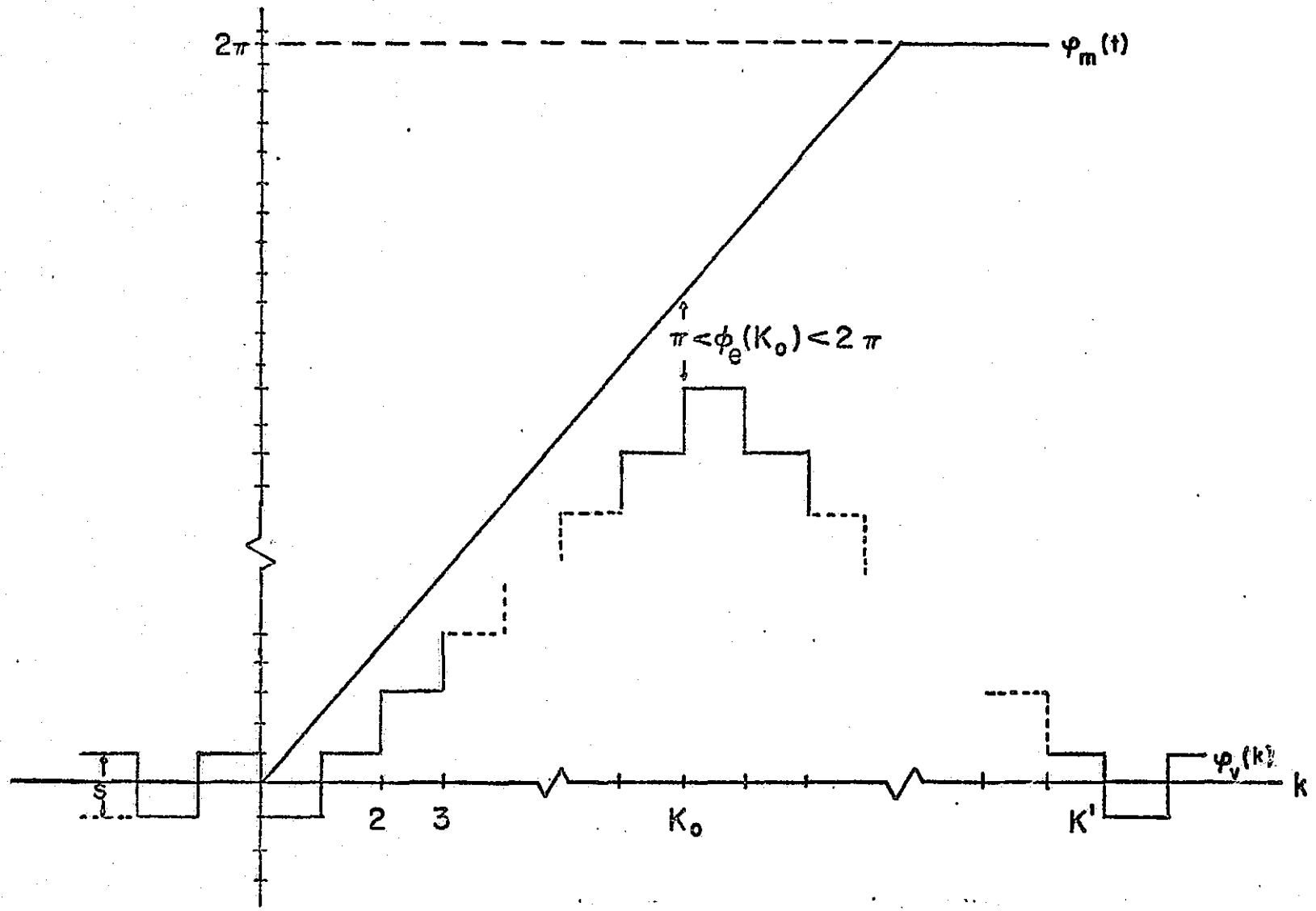


Figure 3. Illustrating Spike Suppression

I. 5. The Measurement of Light Intensity of Computer Generated Pictures

A. Abstract

We describe a method of measuring the total light energy in a computer generated picture. This method uses Cadmium Sulfide (Cd S) photocells as transducers which convert the light intensity to an electrical signal. This signal is then processed and the result is displayed on a DC meter. This new measurement technique will provide an efficient tool to aid us in optimally recording computer generated pictures.

B. Introduction

The equipment used in conducting this research is related to both communications and imaging systems. The apparatus gives a measure of the screen image light energy, which is the output of the video communications system. The image is generally discrete in nature, the shape of which is approximately known. However we want to know the light energy so that we can optimally record the image. In Fig. 1 we show a block diagram of the overall system. Figure 2 presents a block diagram of the electrical system that was designed and constructed.

C. Optical System

A series of experiments were conducted to determine the optimum location of the photocells. It was apparent that the best performance, i.e., signal-to-noise ratio, would be obtained by directing the maximum amount of light from the Cathode Ray Tube (CRT) on to all the photodiodes used. The light had to be directed to the photocells by a method that would not interfere with photographing the image on the CRT screen. We also had to insure that the amount of illumination reaching the photocells was independent of the position of the CRT dot and depended only on the intensity of the dot. It was found that placing eight photodiodes in a circle around the edge of the Polaroid camera lens gave satisfactory results. A sketch of this is given in Fig. 3.

It was found that when the eight photodiodes were connected in parallel their total resistance varied from 500K with maximum illumination from the CRT to 5M in total darkness. This implies that each photocell has a resistance of 8 (500K) = 4M with maximum illumination. Employing the photocell manufacturer's curve displaying cell resistance versus the illumination, we were able to extrapolate to find that the illumination per cell was 0.005 footcandles.

D. The Detection and Measurement System

The optical information obtained from the photodetectors is processed as follows. Operational Amplifier (OA) 1 (see Fig. 4) is a buffer to transform the light intensity, which is proportional to the photocell resistance, into an electrical signal, m(t). During the time interval T_I (see Fig. 6), OA 2 integrates m(t) as follows:

$$v(t) = \frac{1}{T_o} \int_{o}^{T_I < T_S} m(\tau)\, d\tau,$$

where

$T_I \equiv$ the integration period (switch S_o open),

$T_o \equiv 1/R_o C_o$ = the integration time constant

and

$T_S \equiv$ the scanning period for one video frame.

Just prior to the dump time, T_D (see Fig. 6), switch S_1 closes and a final reading of v(t) is taken and presented to the input of OA6. The output of OA6 is then displayed on a DC meter. When S_1 opens, S_0 closes and dumps the accumulated value of v(t). S_0 then opens and the cycle repeats during the next scanning interval. The synchronization circuit of Fig. 5 generates the required control signals to operate switches S_0 and S_1.

E. Overall Performance

The designed circuits were found to function reliably under different screen intensities and complex illumination patterns. In addition, the frequency characteristics of our processing and synchronization circuits are such that we can operate over a wide range of frame scanning time with little change in performance. Thus we are able to obtain a sufficiently good measure of the light energy in a video frame as it appears on our scanner. Using this information to aid in setting the exposure of the Polaroid camera we are able to more precisely record a video frame.

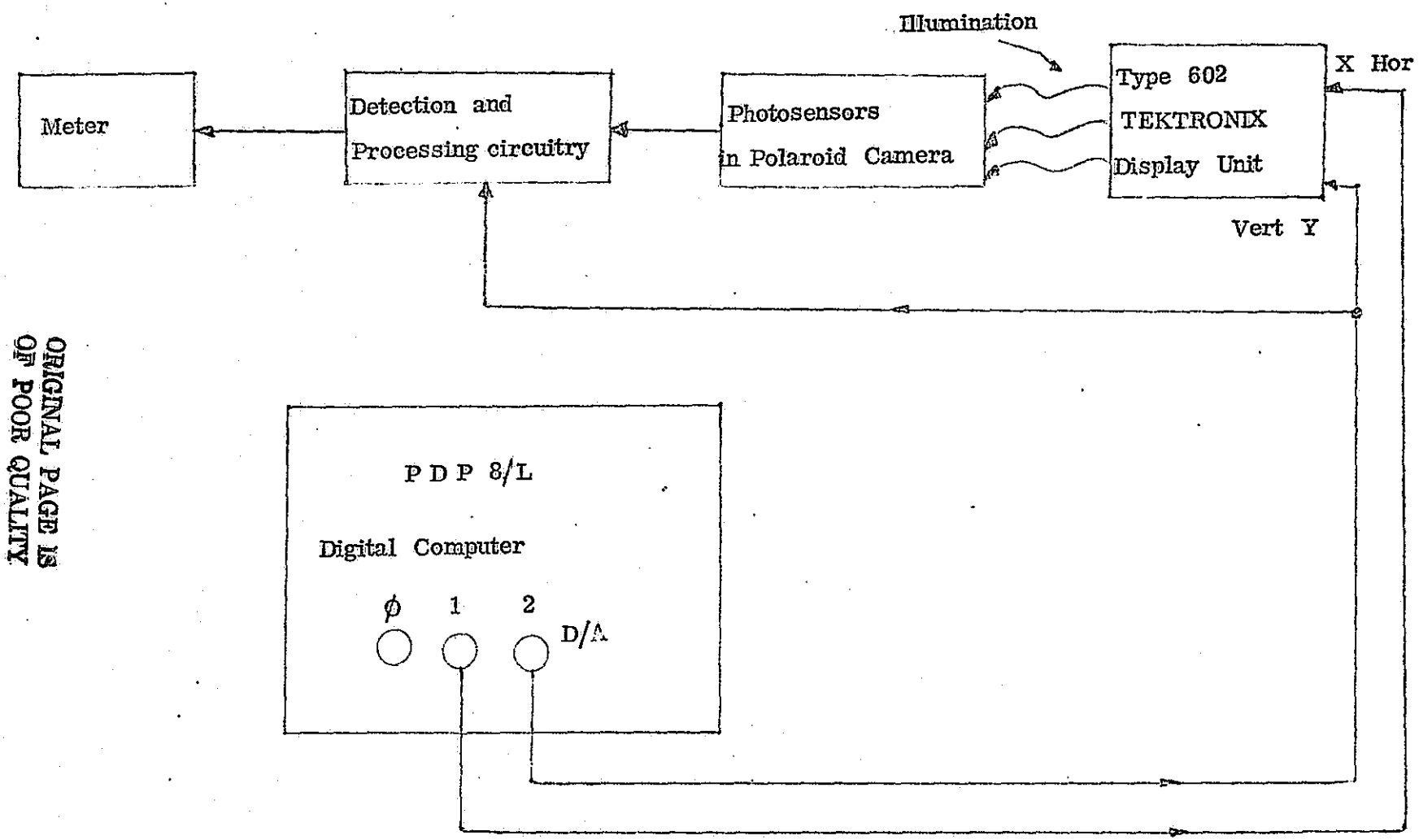


Fig. 1. The overall system.

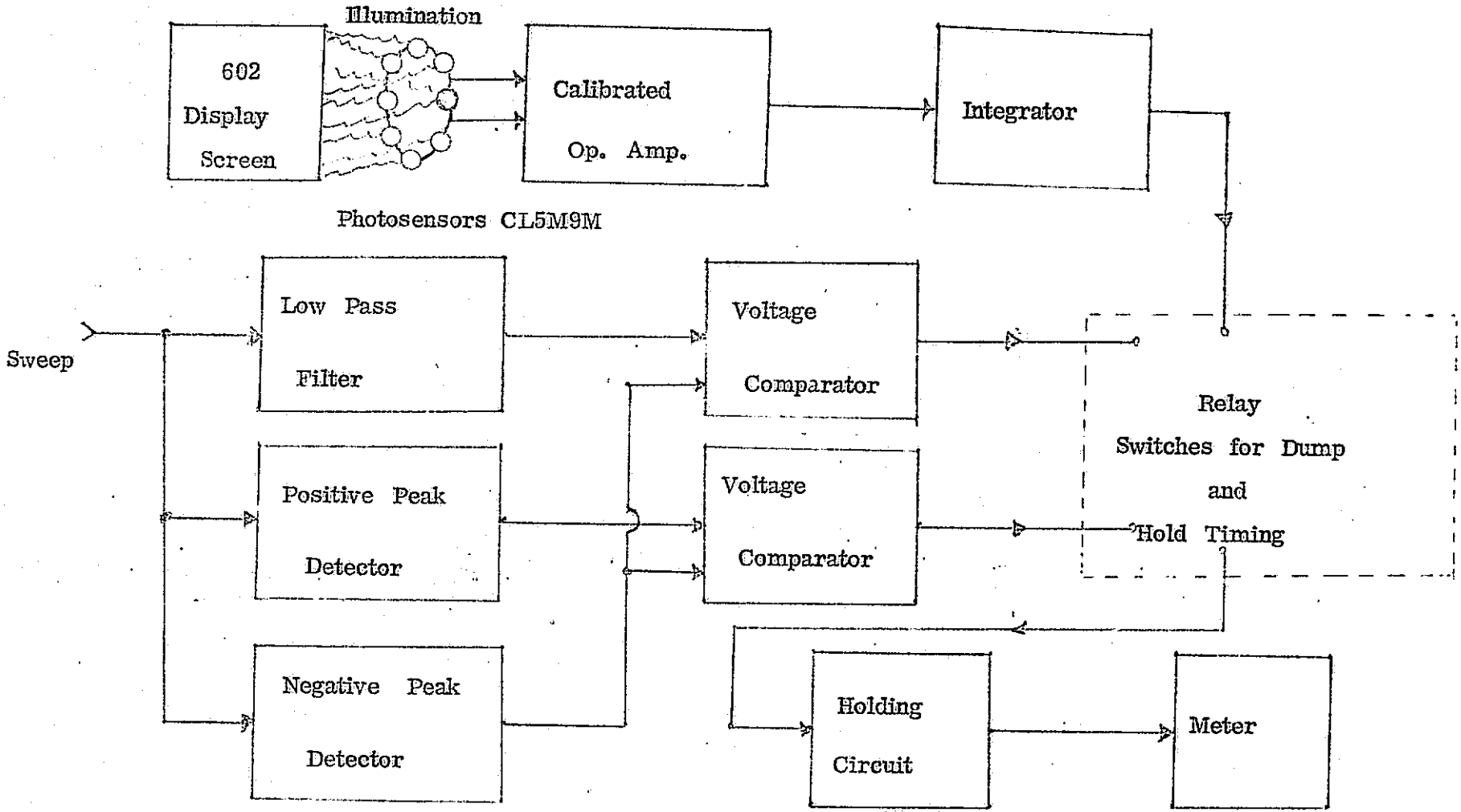


Fig. 2. Block diagram of the electrical system.

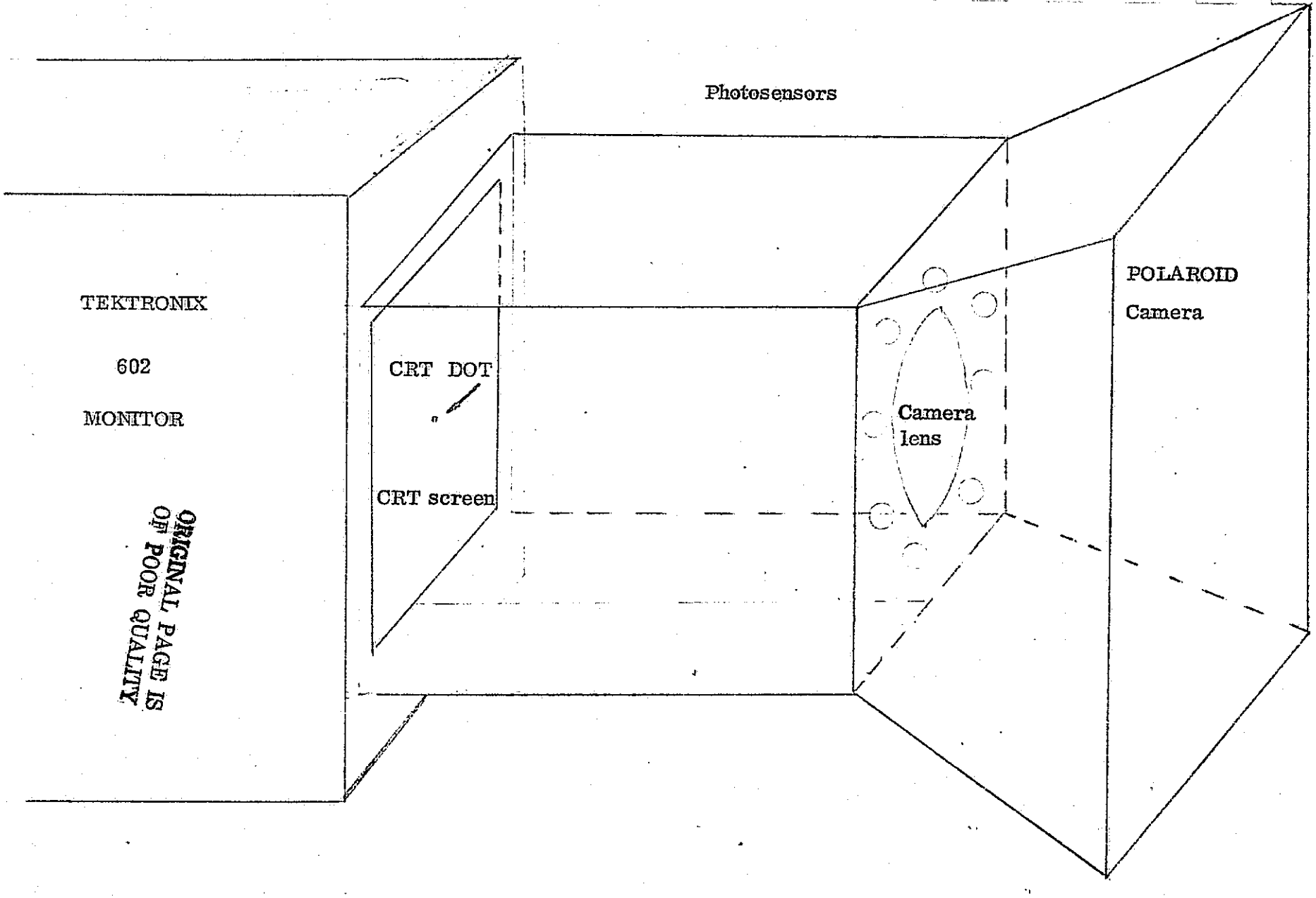


Fig. 3. Placement of photosensors around the camera lens.

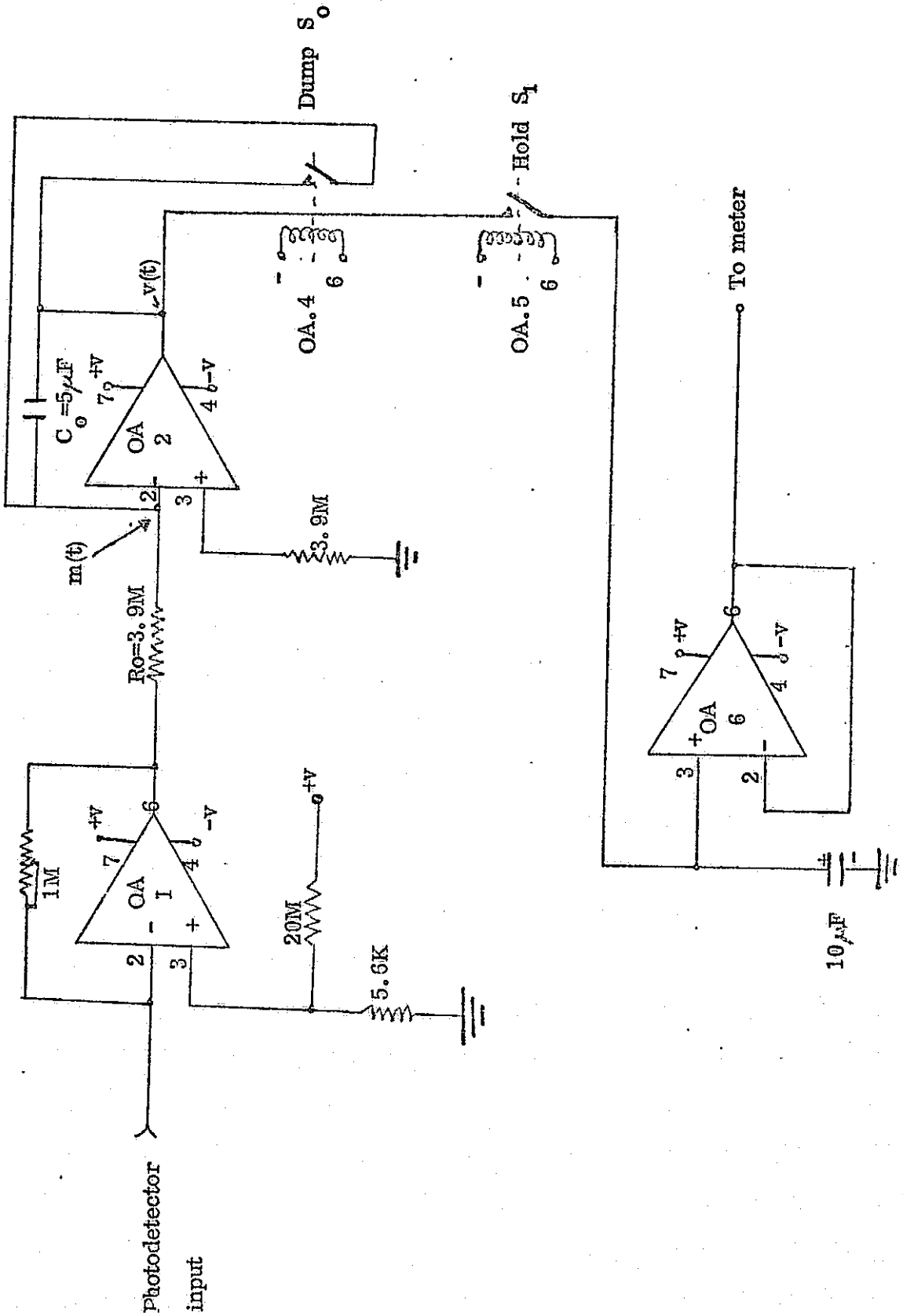


Fig.4. The processing circuit.

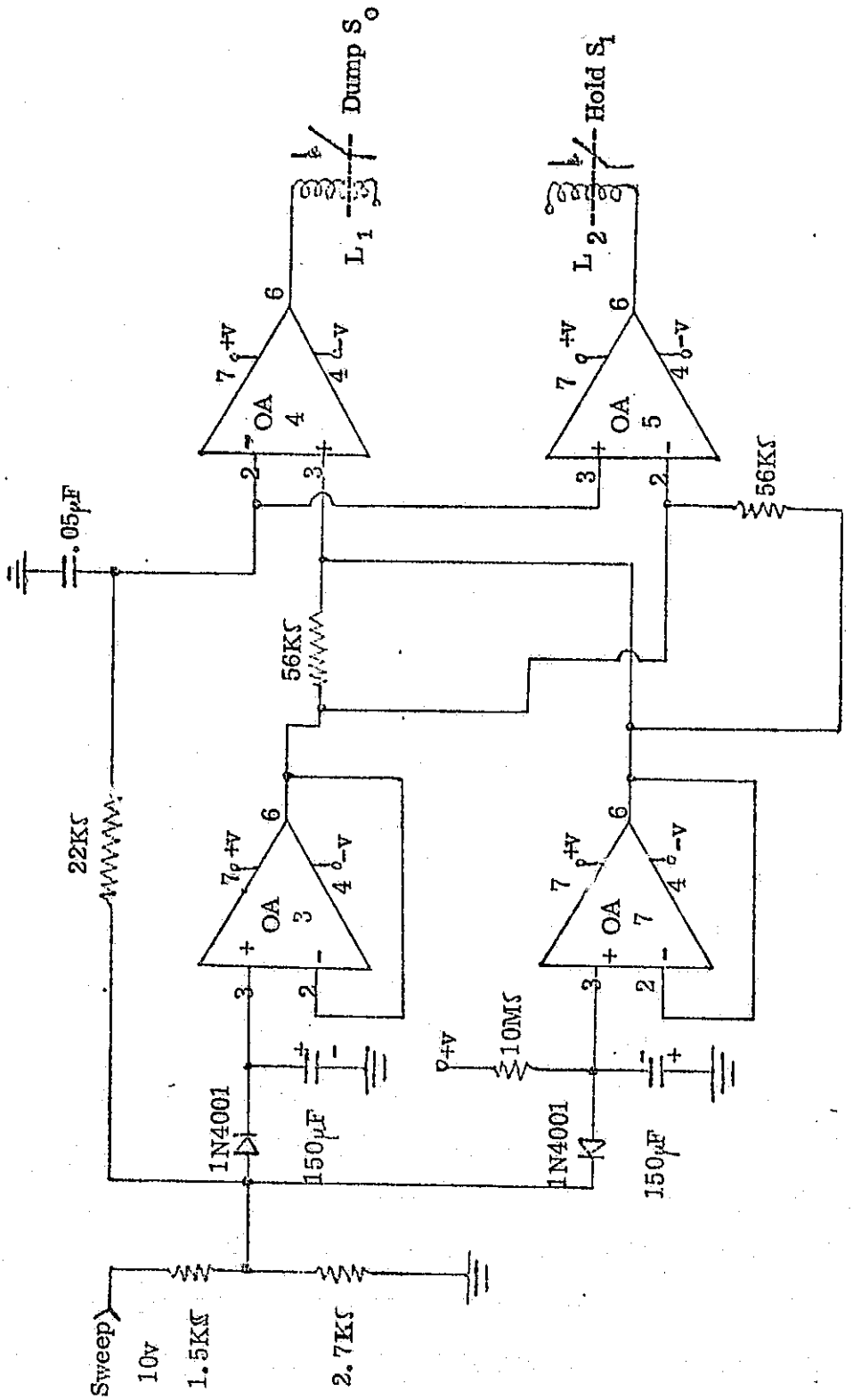


Fig.5. Synchronization circuit.

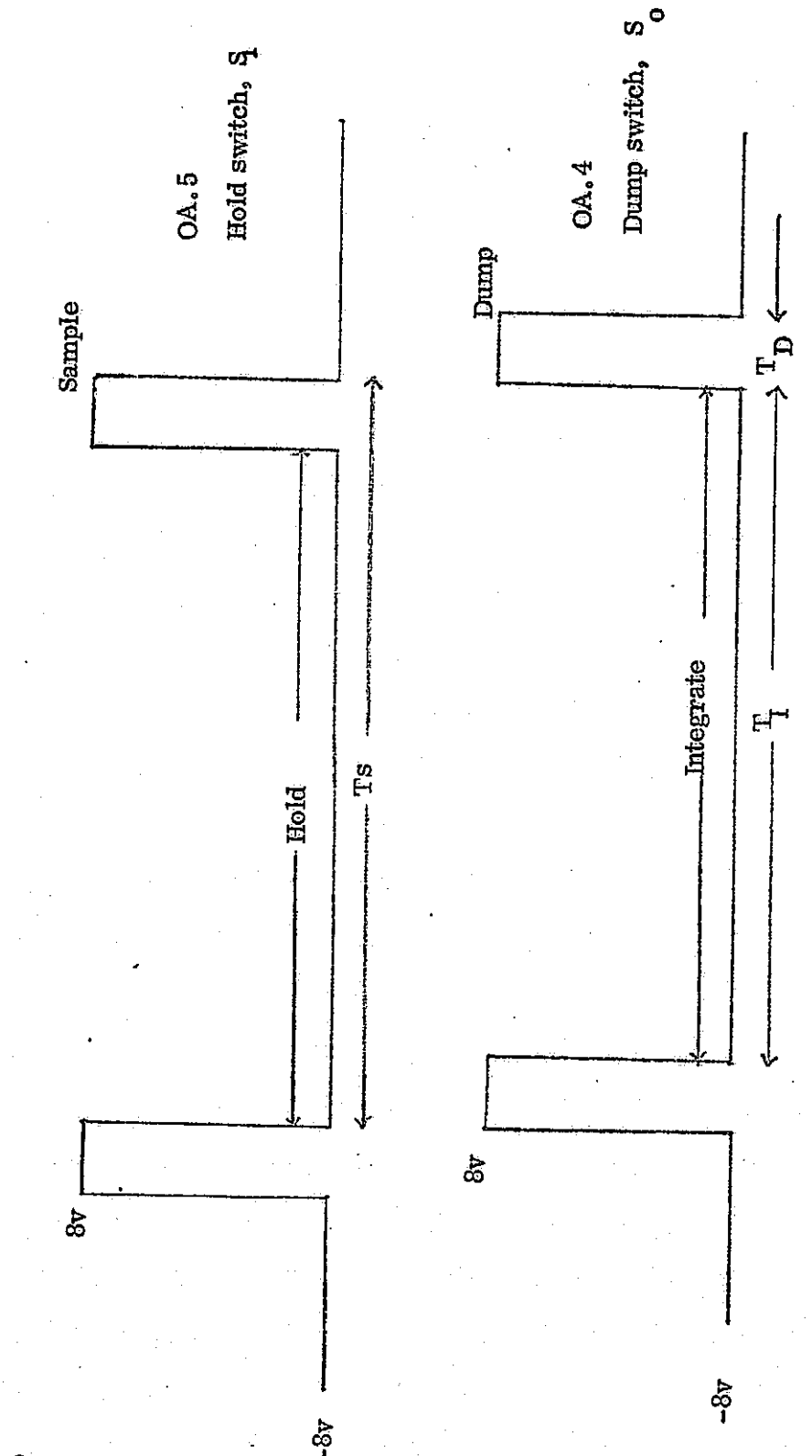


Fig. 6. Control signals for hold switch, S_1, and dump switch S_o.

II. 1 Testability Enhancement in Digital System Design

C. S. Chuang and S. J. Oh

Digital system applications demand high capability to insure the correct operation of a system. Rapid real-time fault detection is essential to satisfy this aspect of the system performance criteria.

Most of the existing techniques to detect the stuck type failures require, in general, an application of long input sequences generated by a hard core computer according to a complex algorithm. On the other hand, with the recent advances in semiconductor technology, the complexity of circuits fabricated on a single LSI chip tends to increase rapidly. The diagnosis of systems utilizing LSI components poses an extremely difficult problem because of the structural complexity of the components and their limited accessibility.

The problem of augmenting the testability of a digital system becomes the important design criteria. Several approaches to enchance the testability of digital system by means of redundant hardware have been developed recently. But, they are still not efficient enough to provide real time fault detection for stuck at fault model.

One approach to satisfy this requirement is to include the fault detecting capability of the system from the initial stage of the system design. New techniques of employing hardware redundancy to reduce the number of tests to detect the stuck at faults are proposed, so that the criteria of real time fault detection will be possible for a digital system.

Some works have been done for this effort, two papers have been published recently. The results show two tests are enough for stuck at fault detection of any combinational circuit implemented with

the proposed PLM's and CSI's, and four tests are sufficient for detecting stuck at fault of the combinational logic portion and single permanent malfunction of the flip-flops of a synchronous sequential circuit implemented with the proposed PLM's, CSI's and CMM's. Also, the test patterns are automatically generated by the systematic design procedure.

Eighth Asilomar Conference on Circuits, Systems and Computers,
Dec. 3-5, 1974, Pacific Grove, California.
Reprint from the Conference Record.

DESIGN OF EASILY DETECTABLE COMBINATIONAL AND SYNCHRONOUS SEQUENTIAL CIRCUITS*

Chin Sheng Chuang and Se Jeung Oh
Department of Electrical Engineering
The City College of The City University of New York
New York, New York

Abstract

The problem to reduce the number of tests for fault detection in both combinational logic circuits and synchronous sequential circuits is investigated. Utilizing programmable logic module and controllable memory module, systematic design and detection procedures are described. Two tests are sufficient to detect any stuck type faults of combinational circuit, and four tests are necessary and sufficient to detect any stuck type faults of elementary logic gates and the malfunction of flip-flops of synchronous sequential circuit using D flip-flops.

I. INTRODUCTION

Digital system applications demand high reliability which implies an increased capability to assure the correct system operation. Rapid real time detection of stuck faults is often essential to insure the integrity of the system performance. One aspect of the system to meet these performance criteria is to imbed the fault detecting capability within the system from the initial stage of system design.

The existing techniques [1]-[3] to detect the stuck faults require, in general, application of long input sequences generated by a hard core computer according to a complex algorithm. On the other hand, due to the recent advances in semiconductor technology, the complexity of circuits fabricated on single LSI chip tends to increase rapidly. The functional diagnosis of systems utilizing LSI components poses a difficult problem.

Techniques [4], [5] were investigated in various directions to simplify the fault detection by using hardware redundancy. A new approach is proposed here which will allow us to perform fault detection in real time.

The following definitions are introduced:

(1) Input Sensitivity: The sensitivity of an input pattern for a logic gate is defined as the number of individual inputs in the pattern capable of sensing failures in the gate. For example, (00) is the most sensitive input pattern for two-input OR gate since any one of the input stuck-at-1 will affect its output. The most sensitive input pattern for the N-input OR gate is $00\ldots0$ (N 0's). By extending this idea, the most sensitive input patterns for the ordinary combinational logic gates are established as shown in Table-1.

(2) Sensitizing Path: A path which propagates the fault information to the observable outputs.

The stuck type faults can be partitioned into two classes by the logic function nature of the elementary gates.

(3) Type I Fault: The fault set consisting of stuck-at-1 (s-a-1) faults of the inputs of OR, NOR and the output of NAND, and the stuck-at-0 (s-a-0) faults of the inputs of AND, NAND and the output of NOR.

The complement of Type I fault is defined as Type II fault and they form the complete class of stuck type faults.

It is found that the Type I fault information in a combinational logic circuit can always propagate to the observable outputs of the circuit if and only if the inputs for all gates are the most sensitive input patterns for the respective gates.

(4) Programmable Logic Module (PLM): A multiple input-single output combinational circuit is defined to be a programmable logic module if each of the logic functions performed by the module can be uniquely specified

* This work was supported in part by NASA under the Contract NAS9-13940.

by a set of control signals.

(5) Control Signal Inverter (CSI): An Exclusive OR gate in which one of inputs is used as level control signal. The specific logic function of PLM for AND, OR, NAND and NOR gates is described in Table-II. There are many ways to implement a desired PLM. Fig. 1 shows one of the implementation of PLMs for two-input gates. The general representation of PLMs is shown in Fig. 2. The circuit constructed by the use of PLMs and CSIs has three modes of operation: NORMAL mode, TEST 1 mode and TEST 2 mode. If we denote one class of stuck type faults which can be detected by TEST 1 mode as Type I fault, then this class is equivalent to the Type II fault of TEST 2 mode. Similarly, the Type II fault of TEST 1 mode is equivalent to the Type I fault of TEST 2 mode and it can be detected by TEST 2 mode. So the complete class of stuck type faults can be detected by setting the circuit to TEST 1 and TEST 2 modes. For single stuck type fault assumption (s-a-1, s-a-0), any combinational circuit with primary output observable can be constructed with PLMs and CSIs so as to detect the stuck type faults with two tests input patterns [4].

(6) Controllable Memory Module (CMM): A memory element with direct set and reset capability whose outputs are convertable by two external control signals. A general model of CMM is shown in Fig. 3. Its outputs are q_i and $\bar{q}_i$ (where $q_i = y_i \cdot C_{y_i}$, $\bar{q}_i = y_i \cdot C_{\bar{y}_i}$), and $C_{y_i}/C_{\bar{y}_i}$ depends on the control signals of its successor PLM and direct set and reset signals of the flip-flop. Consider a Delay flip-flop CMM as shown in Fig. 4. Type I malfunction is defined as the failure of a Delay CMM which fails to propagate 0 when D=0. Fig. 4(a) shows that the s-a-1 of D, y, q and C_y are equivalent to Type I malfunction. Type II malfunction is defined as the failure of a Delay CMM which fails to propagate 1 when D=1. Fig. 4(b) shows that the s-a-0 of D, y, q and C_y are equivalent to Type II malfunction. It is sufficient to detect the malfunction of CMM for fault detection of the flip-flops portion of a sequential circuit. The relation of control signals between the consecutive gates are shown in Table-III. $C_{y_i}/C_{\bar{y}_i}$ of CMM is set according to Table-IV. Systematic design and detection procedures are developed. Two examples are used to illustrate the procedures.

II. COMBINATIONAL CIRCUIT DESIGN

Consider the boolean function $f=b(a'+c'd)+(bd)'(a'+c'd)'$ as shown in Fig. 5 (a). Its controllable circuit is constructed as shown in Fig. 5 (b). All the control signals C_1, C_2, C_3 are set to be 0 during the normal operation. For test operation, we apply the two tests according to the detection procedure as follows:

TEST 1: Set $C_1C_2C_3$=011 (TEST 1 mode) and apply input test pattern abcd=0101. If the observable output= f=1, then the circuit does not have stuck fault of Type I. Otherwise, it has at least one stuck fault of Type I.

TEST 2: Set $C_1C_2C_3$=101 (TEST 2 mode) and apply input test pattern abcd=1010. If the observable output f=0, then the circuit does not have stuck fault of Type II. Otherwise, it has at least one stuck fault of Type II. Once the circuit pass these two tests, the circuit is guaranteed to be free of stuck-at-1 and stuck-at-0 faults.

III. SYNCHRONOUS SEQUENTIAL CIRCUIT WITH D-FLIP-FLOPS

Consider a two bit shift right/shift left register as shown in Fig. 6(a) in which X_1 is the serial input for shift left, X_2 is the serial input for shift right and X_3 is the shift control. It is converted to an easily testable circuit as shown in Fig. 6(b), where Z_m is the monitoring output. For normal operation we set all the control signals to be 0. For testing, four tests are applied according to the detection procedure as follows:

TEST 1: (S_dR_d=10): Set $C_1C_2C_3C_{y_1}C_{y_2}$=01100 and apply input test pattern $X_1X_2X_3$=110. Observe the output Z_m. If Z_m=1, then the combinational logic does not have stuck fault of Type I. Otherwise, it has at least one.

TEST 2: (S_dR_d=00): Observe the output $Z_{y_1}Z_{y_2}$, if $Z_{y_1}Z_{y_2}$=00, then the flip-flops do not have the TYPE II malfunction. The disagreement implies the corresponding flip-flop has the Type II malfunction.

TEST 3: (S_dR_d=01): Set $C_1C_2C_3C_{y_1}C_{y_2}$=10100, and apply input test pattern $X_1X_2X_3$=001. Observe the output Z_m, if Z_m=0, then the combinational logic does not have stuck fault of Type II. Otherwise, it has at least one.

TEST 4: (S_dR_d=00): Observe the output $Z_{y_1}Z_{y_2}$, if $Z_{y_1}Z_{y_2}$=11, then the flip-flops do not have the Type I malfunction. The disagreement implies the corresponding flip-flop has the Type I malfunction. Once the circuit passes these tests, it is guaranteed to be free of stuck at 1 and 0 faults of logic gates and the malfunction of flip-flops. Notice that the Exclusive OR unit in the CMM is not necessary for this special circuit.

IV. CONCLUSION

The utilization of PLMs, CMMs and CSIs assures that two tests can detect any combinational circuit and four tests, for any synchronous sequential circuit using D-flip-flops. In addition, the test pattern can be easily generated by a systematic procedure so as to allow fault detection in real time.

REFERENCES

1. Kautz, W.H., "Fault Testing and Diagnosis in Combinational Digital Circuits", IEEE Trans. on Computers, Vol. C-17, 1968, pp. 352-366.
2. Roth, J.P., "Diagnosis of Automata Failure: A Calculus and A Method", IBM Journal R and D, Vol. 10, 1966, pp. 278-291.

3. Su, S.Y.H. and Cho, Y.C., "A New Approach to the Fault Location of Combinational Networks", IEEE Trans. on Computers, Vol. C-21, 1972, pp. 21-30.

4. Sakauchi, M. and Inose, H., "Synthesis of Fault Diagnosable Logical Circuits by Function Conversion Method", Trans. C of JIEC, Vol. 56-D, No. 1, Jan. 1973, pp. 47-54.

5. Hayes, J. P., "On Modifying Logic Networks to Improve Their Diagnosability", IEEE Trans. on Computers, Vol. C-23, 1974, pp. 56-62.

N-Input Gate	The Most Sensitive Input Pattern
OR	00...0 (N 0's)
AND	11...1 (N 1's)
NOR	00...0 (N 0's)
NAND	11...1 (N 1's)

Table-I. The most sensitive input pattern of N-input elementary gates.

Module	Desired Logic Function When : C=0	C=1
PLM_1	AND	OR
PLM_2	OR	AND
PLM_3	NAND	NOR
PLM_4	NOR	NAND

Table-II. Four basic programmable logic modules.

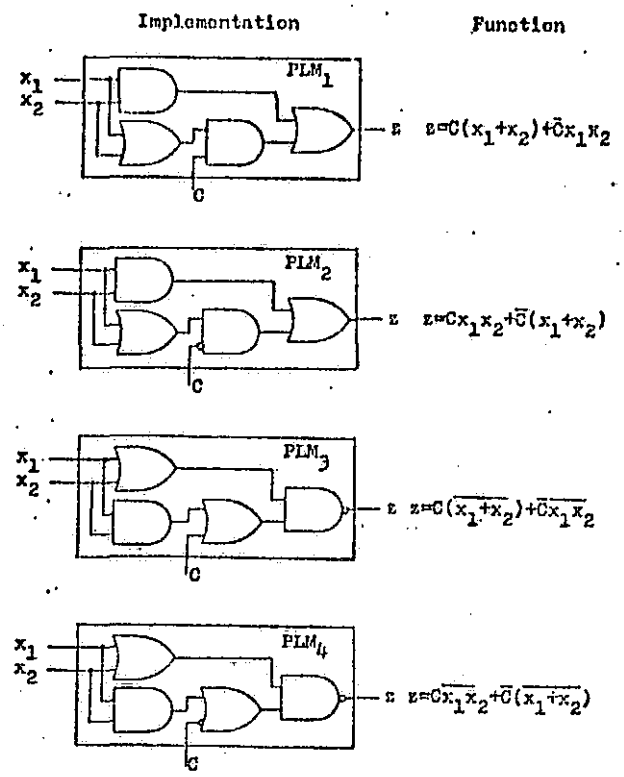


Fig. 1. An implementation of four basic 2-input PLMs.

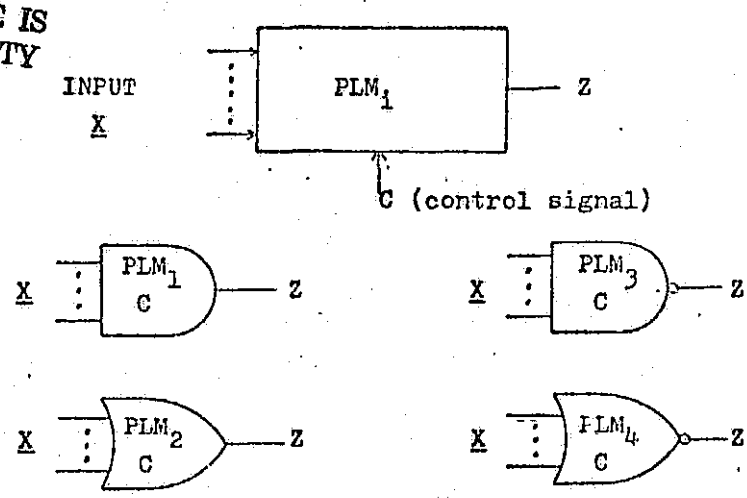


Fig. 2. The general model of PLM and symbols.

Gate	Its Successor	The Relation of Their Control Signal
OR	OR/NOR	SAME
	AND/NAND	DIFFERENT
AND	OR/NOR	DIFFERENT
	AND/NAND	SAME
NOR	OR/NOR	DIFFERENT
	AND/NAND	SAME
NAND	OR/NOR	SAME
	AND/NAND	DIFFERENT

Table-III. The general rule of determining the PLM control signal settings.

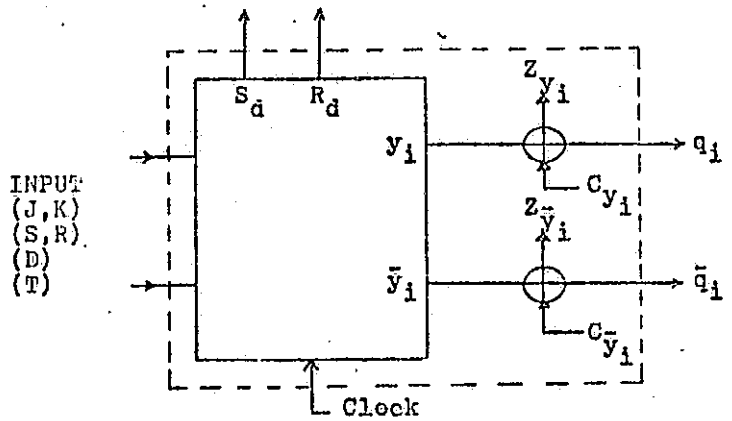


Fig. 3. The general model of CMM.

The Successor of The State Variable of The Flip-Flop (q_i or $\tilde{q}_i$)	S_dR_d= 10	S_dR_d= 01
PLM_1 or PLM_3	$C_{y_i} = C_j$ *	$C_{y_i} = \bar{C}_j$
	$C_{\bar{y}_i} = \bar{C}_j$	$C_{\bar{y}_i} = C_j$
PLM_2 or PLM_4	$C_{y_i} = \bar{C}_j$	$C_{y_i} = C_j$
	$C_{\bar{y}_i} = C_j$	$C_{\bar{y}_i} = \bar{C}_j$

* C_j is the control signal of PLMs

Table-IV. Control signal of CMM with respect to the pre-determined control signal of its successor PLMs.

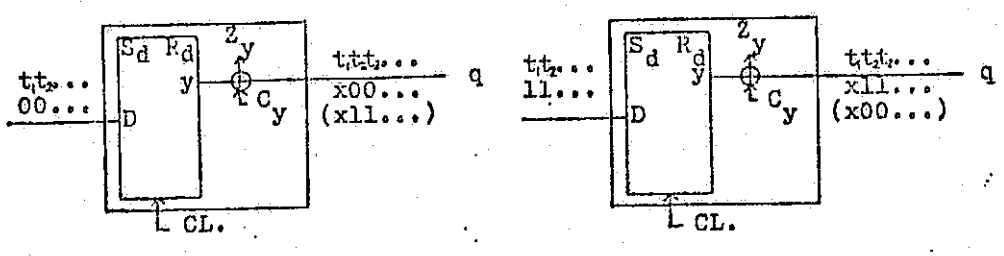
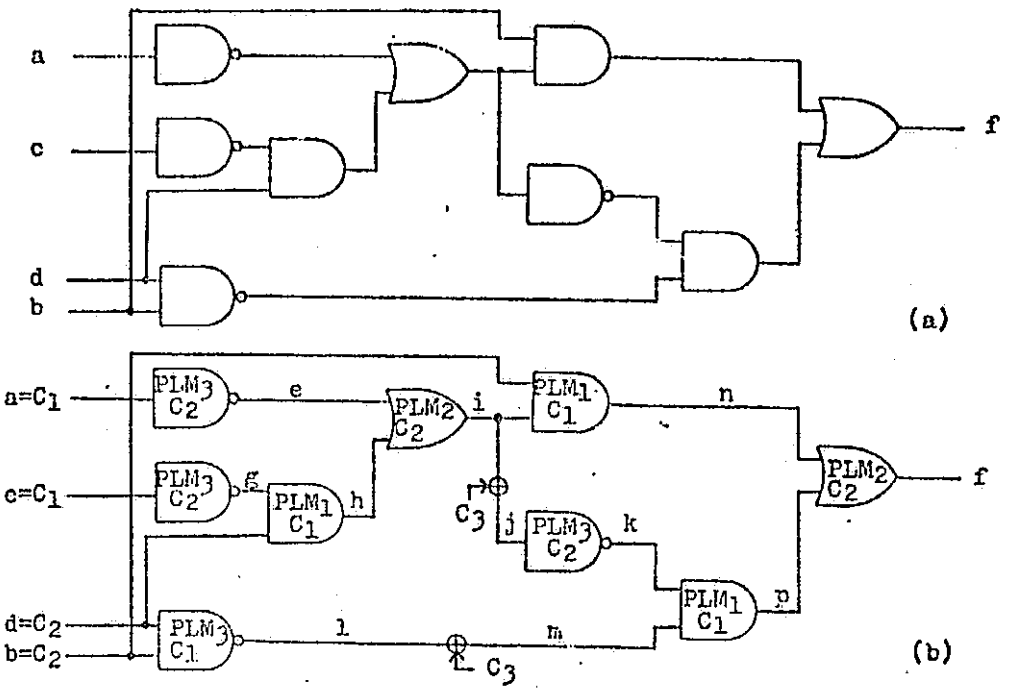
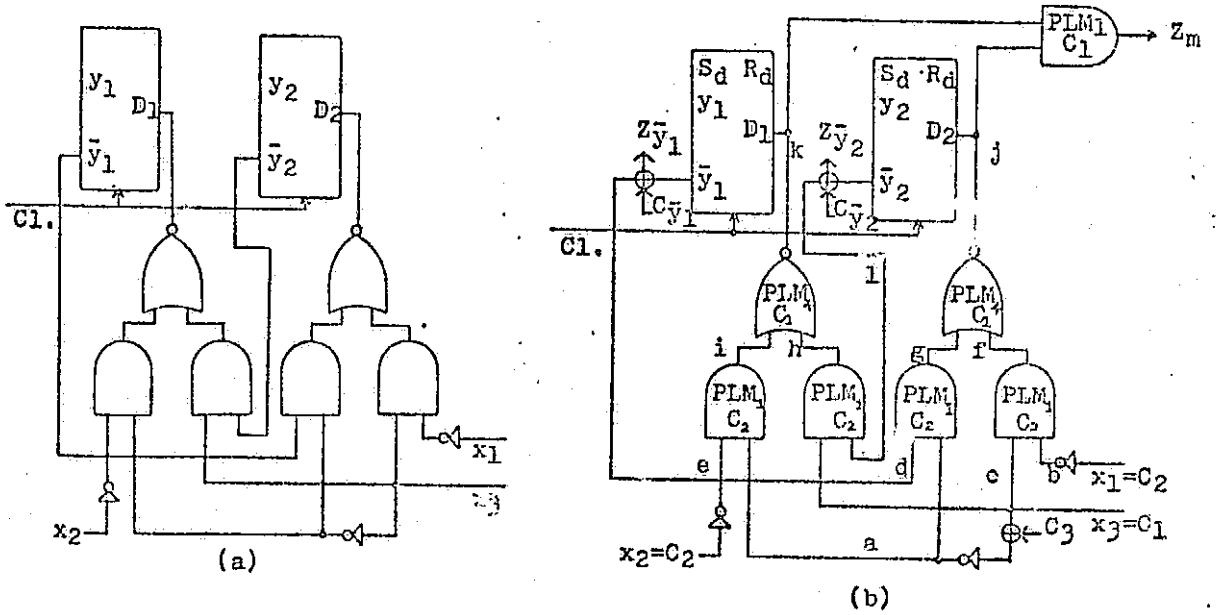


Fig. 4. (a) Type I malfunction of Delay CMM.
(b) Type II malfunction of Delay CMM.



Type I faults=$[a_1, b_0, c_1, d_0, e_0, g_0, h_0, i_0, j_1, k_0, l_1, m_0, n_0, p_0, f_0]$
Type II faults=$[a_0, b_1, c_0, d_1, e_1, g_1, h_1, i_1, j_0, k_1, l_0, m_1, n_1, p_1, f_1]$
Where x_k = the line x stuck-at-k (k= 0 or 1)

Fig. 5. Combinational circuit design example.
(a) Original circuit. (b) Easily testable circuit.



Type I faults=$[a_1, b_1, c_1, d_1, e_1, f_1, g_1, h_1, i_1, j_0, k_0, l_1, Zm_0]$
Type II faults=$[a_0, b_0, c_0, d_0, e_0, f_0, g_0, h_0, i_0, j_1, k_1, l_0, Zm_1]$

Fig. 6. Synchronous sequential circuit design example.
(a) Original circuit. (b) Easily testable circuit.

Reprint from IEEE Intercon 75 Session 11

TESTABILITY ENHANCEMENT IN DIGITAL SYSTEM DESIGN*

Chin Sheng Chuang and Se Jeung Oh
Department of Electrical Engineering
The City College of The City University of New York
New York, New York

ABSTRACT

Several approaches to enhance the testability of digital system by means of redundant hardware have been proposed recently.[1,2,3] A new way of employing hardware redundancy to reduce the number of tests for fault detection in both combinational and synchronous sequential circuits is investigated. An approach is presented for utilizing systematic redundancy to simplify design work. Models for PLM (programmable logic module) and CMM (controllable memory module) are depicted. Systematic design and detection procedures are described. Using these procedures, an easily testable circuit (for stuck faults) can be designed. In contrast to the earlier results on fault detection in logic circuit,[4,5,6] two tests are needed to detect any stuck faults of combinational logic circuit. Four tests are necessary and sufficient to detect any stuck faults of elementary logic gates and the malfunction of flip-flops of synchronous sequential circuit using Delay flip-flops or Trigger flip-flops.

I. COMBINATIONAL NETWORK

If a logic gate can not perform the desired logic function with respect to the input pattern applied, then there exists some permanent faults in the logic gate. These permanent faults are defined as stuck fault. The stuck fault can be categorized into two types, that is, stuck at logic 1 (s-a-1) and stuck at logic 0 (s-a-0).

Consider a two-input AND gate. There are four possible binary input patterns (00, 01, 10, 11). Some input failures will not cause the output to change, i.e., they are masked. For example, if 00 is used as the inputs to an AND gate, the output will be affected only when both inputs stuck at 1, and all the other possible stuck failures will be masked. We introduce the sensitivity of an input pattern to a gate to classify the pattern according to its ability to avoid fault masking.

The sensitivity of an input pattern to a gate is defined as the number of individual inputs in the pattern capable of sensing failures in the gate.

If we apply the input pattern 01 to an AND gate, then only the input "0" stuck-at-1 will affect the output. If we apply the input pattern 11 to the AND gate, then any or both "1" stuck-at-0 will cause the change of output. Hence the most sensitive input pattern for the AND gate is 11. We find that the most sensitive input pattern for an N-input AND gate is 11...1 (N 1's) by extending the definition. The most sensitive input patterns for ordinary OR and NOR gates are all zero input, and for ordinary AND and NAND gate are all one input. For Exclusive-OR and Equivalence gates, all the possible input patterns have equal sensitivity.

* This work was supported in part by NASA-Houston under the NASA Contract NAS 9-13940.

A path is defined as sensitizing path if the fault information can propagate along the path to the observable outputs. The stuck type fault of any combinational circuit can be partitioned into two distinct classes according to the logic function performed by the elementary gate. One class is the stuck-at-1 (s-a-1) faults of fan-in of an OR or NOR gate and fan-out of a NAND gate and the stuck-at-0 (s-a-0) faults of fan-in of an AND or NAND gate and fan-out of a NOR gate. This class is denoted as Type I faults. Another class is the complement of Type I faults and is denoted as Type II faults. We can then conclude the following theorem.

Theorem I

The Type I fault information in a combinational logic circuit always propagates to the observable outputs of the circuit if and only if the circuit topology is arranged in such a way that the inputs for all gates are the most sensitive input patterns for the respective gates.

Proof

1. Sufficient Condition If we use the most sensitive input patterns as the test inputs, then any stuck at 1 or 0 fault of Type I faults will alter the output of the gate. This will in turn affect the output of the subsequent gates connected to it by chain process, and the fault information of Type I will eventually propagate to the observable outputs.

2. Necessary Condition Since the most sensitive input patterns will not mask any set of fault information in the Type I faults, the necessity is obvious.

In the following discussion, let the line x stuck at the logic value k be denoted as x_k (k=0 or 1).

Example I

Consider the circuit shown in Fig. 1. The circuit satisfies the conditions of theorem I because it is possible to find a test input pattern such that inputs to every gate are most sensitive. When we apply an input pattern abcdefg=1101001 to this circuit and observe z=1 in response to the input, then the circuit does not have Type I faults, where Type I faults= $\{a_0, b_0, c_1, d_0, e_1, f_1, g_0, h_1, i_0, j_1, k_1, l_0, z_0\}$. On the other hand, if z=0, there exists at least one Type I fault. Similarly, if a new circuit is generated by interchanging OR to AND, and NOR to NAND, or vice versa in the circuit of Fig. 1, we can detect the complement of Type I faults of the original circuit from the new circuit with the complement of the input pattern used for detecting the Type I faults in the original circuit. This structural conversion for testing can be performed by means of the programmable logic modules (PLM).

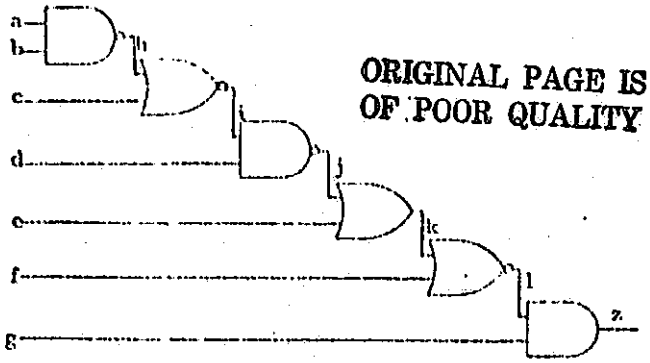


Fig. 1

A multiple input-single output combinational circuit is defined to be a programmable logic module if each of the logic functions performed by the module can be uniquely specified by a set of control signals.

The specific logic functions of PLM for AND, OR, NAND and NOR gates are described in Table I. There are numerous ways to implement the desired PLM. Fig. 2 shows one of the implementation of PLM's for two-input and one-output gates. The general representations of PLM's are shown in Fig. 3.

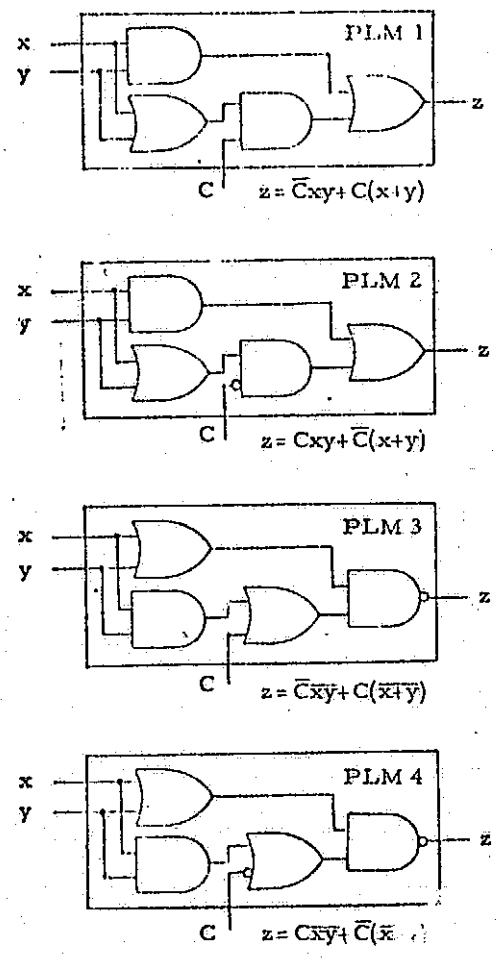


Fig. 2

Module	Desired Logic Function When: C = 0	C = 1
PLM 1	AND	OR
PLM 2	OR	AND
PLM 3	NAND	NOR
PLM 4	NOR	NAND

Table I

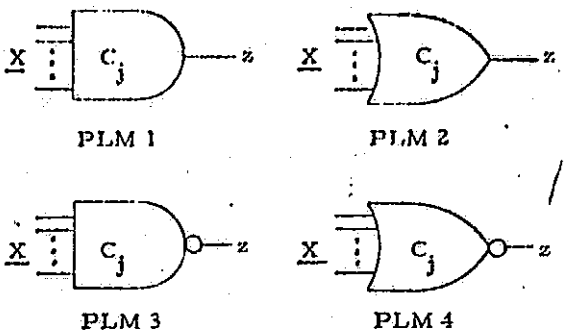


Fig. 3

Corollary

Any stuck fault information in the combinational logic network will propagate to the observable outputs if and only if the structure of the network satisfies theorem I and all the gates are interchanged with the suitable types of programmable logic modules.

Proof

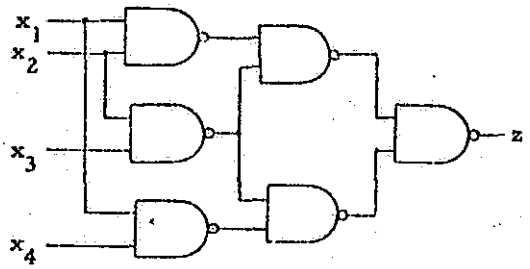
The detection of Type I faults is obvious. The utilization of programmable logic modules allows us to derive a simple test pattern for the detection of the complement of Type I faults by setting the control signals for the PLM's. In other words, any stuck type faults can be detected by a single input pattern and its complement.

The circuit constructed by the PLMs has three circuit modes, i. e., NORMAL mode, TEST 1 mode, TEST 2 mode. If we denote the class of stuck type fault that can be detected by TEST 1 mode as Type I faults, then it is the Type II faults of TEST 2 mode, so it can be detected under TEST 2 mode. This implies that the complete stuck fault class will be detected under TEST 1 mode and TEST 2 mode. In the following, we let C_1C_2=00 indicate NORMAL mode, C_1C_2=01 indicate TEST 1 mode and C_1C_2=10 indicate TEST 2 mode, where C_1C_2 are the control signal of its corresponding PLMs. Notice that the faults at control signal are equivalent to the corresponding fault class at testing.

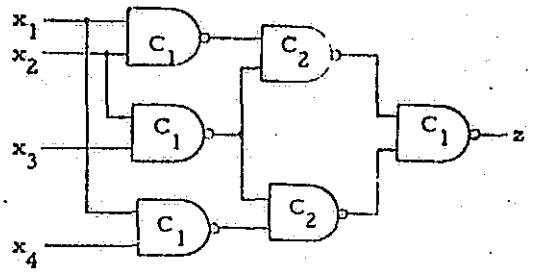
Example II

Consider the combinational circuit for $z = (x_1x_2)'(x_2x_3)' + (x_1x_4)'(x_2x_3)'$ as shown in Fig. 4 (a). We use the PLM3 in place of each NAND gate and connect the control signals as shown in Fig. 4 (b). For normal operation, setting C_1C_2=00 (normal mode), all the PLM3's function as NAND gates.

For testing, we set C_1C_2=10 (TEST 2 mode) and $(x_1x_2x_3x_4)$=(0000), consequently, all the gates in odd level become NOR and all the gates in even level become NAND. If z=0, we then know that there exists at least one Type II fault. Then we set C_1C_2=01 (TEST 1 mode) and $(x_1x_2x_3x_4)$=(1111). Now all the gates in odd level become NAND's and all the gates in even level NOR's. If z=1, there exists at least one Type I fault. We observe that the test input sequence is short and easy to produce if the combinational network is constructed to satisfy the conditions of Theorem I.



(a)



(b)

Fig. 4

Except for a restricted class of circuits such as the one illustrated in Example III, we can convert a circuit to a specified structure which will satisfy the requirement of Theorem I during the testing by properly setting the control signals of PLMs'. The choice of control signal settings for a given circuit is therefore important. The relation of control signals between the consecutive gates are established as shown in Table II based on Theorem I and its corollary.

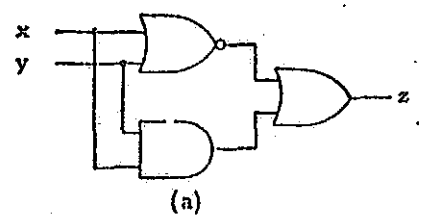
Gate	Its Successor	Relation of Their Control Signal
OR	OR/NOR	Same
	AND/NAND	Different
AND	OR/NOR	Different
	AND/NAND	Same
NOR	OR/NOR	Different
	AND/NAND	Same
NAND	OR/NOR	Same
	AND/NAND	Different

Table II

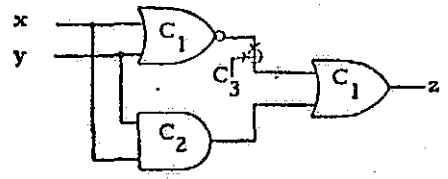
Example III

Consider the Boolean expression z=x'y'+xy as shown in Fig. 5 (a). It is impossible to construct the circuit to satisfy the connection requirement by the technique of setting control signals. The only way to solve this problem is to apply a controlled signal inverter (CSI) at the proper points in the circuit to make the circuit satisfy the conditions of Theorem I. The CSI is actually an Exclusive OR gate in which one of the input is used as level control signal. So the circuit can be reconstructed by using PLM's and CSI as shown in Fig. 5 (b).

For normal operation, we set $C_1C_2C_3$=000. For testing, we set $C_1C_2C_3$=011 with (xy)=(00), and then set $C_1C_2C_3$=101 with (xy)=(11).



(a)



(b)

Fig. 5

In our subsequent discussion, an input variable of the circuit will be said to be a primary input if the variable is included in the minimal expression of at least one observable circuit output. We then conclude the following corollary.

Corollary II

Given any combinational circuit, if each observable output of the circuit can be represented as an explicit switching function of circuit inputs, and each circuit input variable is a primary input, then the circuit can be constructed with PLM's and CSI so as to detect the stuck faults with two test input patterns.

II. SYSTEMATIC DESIGN PROCEDURE FOR COMBINATIONAL CIRCUIT

The following definitions are introduced to facilitate the discussion of the systematic design procedure.

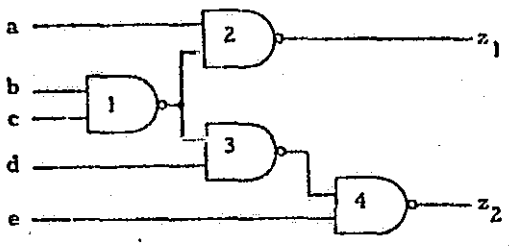
Predecessor gate

A gate G_i is called the predecessor of a gate G_j if the gate output of G_i is one of the inputs of the gate G_j. On the other hand, G_j is the successor of G_i.

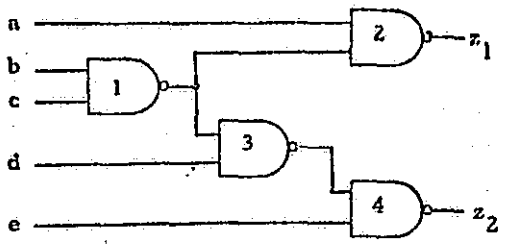
Level of gate G

Let the level of gate G be denoted as L(G). Then L(G)=1 if the gate output of G is the primary output (observable). Otherwise, L(G)= the maximum level of the successors of G + 1.

Consider the circuit shown in Fig. 6, where Z_1Z_2 are primary output. Then $L(G_2)=L(G_4)=1$ because their gate outputs are primary outputs. $L(G_3)=L(G_4)+1=2$ and $L(G_1)=\max\left[L(G_2), L(G_3)\right]+1=2+1=3$ according to the definition.



(a)



level 3 level 2 level 1

(b)

Fig. 6

An algorithm of the systematic design procedure for the combinational circuit is generated from the property of the proposed PLM's and the introduced definitions. The flow chart is shown in Fig. 7 and the details of the procedure are described below.

Systematic design procedure

Step 1 Find out the level of each gate of a given combinational circuit and reconstruct the circuit to a controllable circuit by using PLM1, PLM2, PLM3 and PLM4 to replace the AND, OR, NAND and NOR gate respectively. Denote the control signal of the predecessor as CP and the logic value of the primary input as PI and the reference control signal as RS.

Step 2 Let I=1 and J=the maximum level of the circuit and the control signal of PLM1/PLM3 in level-1 be C_1 and that of PLM2/PLM4 in level-1 be C_2, where C_2 is the complement of C_1.

Step 3 Pick one input of level-I which is not picked yet and trace back to its predecessor. If the module from which the input is picked is PLM1/PLM3, go to step 4. If the module from which the input is picked is PLM2/PLM4, go to step 10.

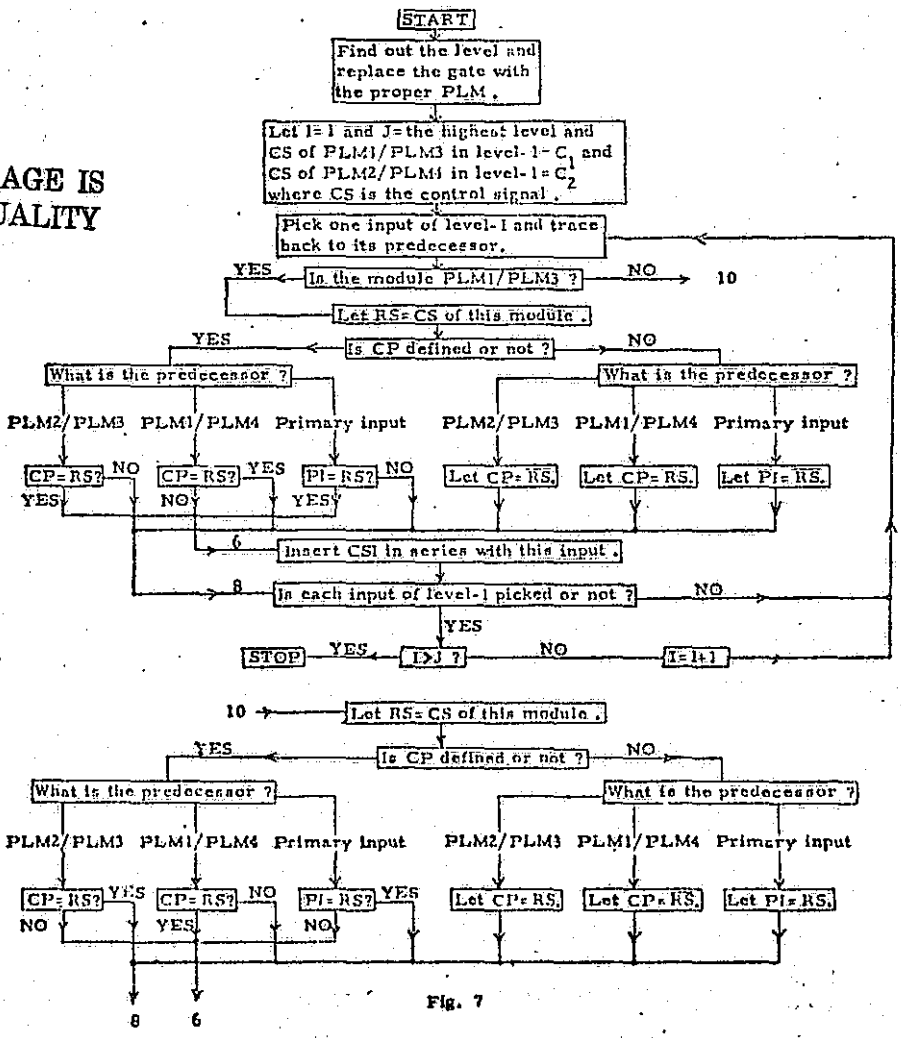


Fig. 7

Step 4 Let RS be equal to the control signal of this PLM1/PLM3 module. Then check to see whether the control signal of its predecessor is defined or not. If yes, go to step 5. Otherwise, go to step 7.

Step 5 If the predecessor is:

(1) PLM2/PLM3, check "Is CP=RS?" If yes, go to step 6; otherwise go to step 5.

(2) PLM1/PLM4, check "Is CP=RS?" If yes, go to step 8; otherwise go to step 6.

(3) Primary input, check "Is PI=RS?" If yes, go to step 6; otherwise, go to step 8.

Step 6 Insert a CSI in series with this input. Then go to step 8.

Step 7 If the predecessor is:

(1) PLM2/PLM3, set CP to be the complement of RS.

(2) PLM1/PLM4, set CP to be equal to RS.

(3) Primary input, set PI to be the complement of RS. Then go to step 8.

Step 8 Check to see whether each input of the lebel-I is picked or not. If yes, go to step 9; otherwise go back to step 3.

Step 9 Check "Is I larger than J?" If yes, the procedures are completed; otherwise let I=I+1 then go back to step 3.

Step 10 Let RS be equal to the control signal of this PLM2/PLM4 module. Then check to see whether the control signal of its predecessor is defined or not. If yes, go to step 11. Otherwise, go to step 12.

Step 11 If the predecessor is:

(1) PLM2/PLM3, check "Is CP=RS?" If yes, go to step 8; otherwise go to step 6.

(2) PLM1/PLM4, check "Is CP=RS?" If yes, go to step 6; otherwise go to step 8.

(3) Primary input, check "Is PI=RS?" If yes, go to step 8; otherwise go to step 6.

Step 12 If the predecessor is:

(1) PLM2/PLM3, set CP to be equal to RS.

(2) PLM1/PLM4, set CP to be the complement of RS.

(3) Primary input, set PI to be equal to RS. Then go to step 8.

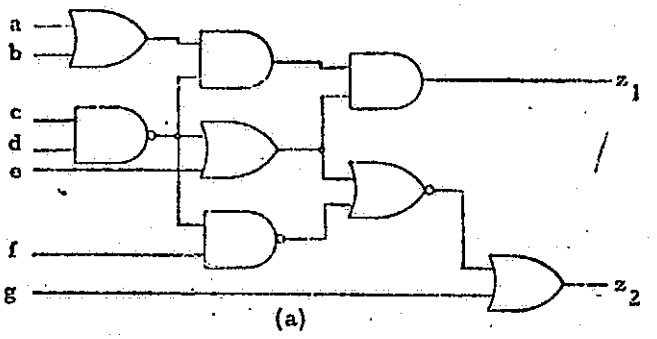
Fault detection procedure

Fault detection process for the translated circuit is easy to follow. The test input pattern is already generated by the design procedure. Two tests can detect all stuck faults (Type I and Type II):

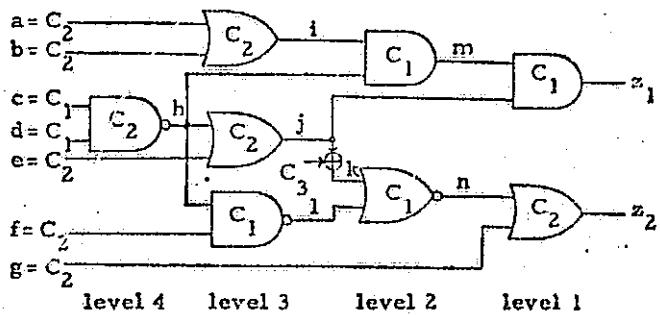
Test 1 Set $C_1C_2C_3$=011 and apply the corresponding test input pattern. If the observable output of PLM1/PLM2 is 1 and that of PLM3/PLM4 is 0, then the circuit does not have any Type I fault. Otherwise, it has at least one.

Test 2 Set $C_1C_2C_3$=101 and apply the corresponding test input pattern. If the observable output of PLM1/PLM2 is 0 and that of PLM3/PLM4 is 1, then the circuit does not have any Type II fault. Otherwise, it has at least one.

Once the circuit passes these two tests, it has guaranteed to be free of stuck faults, then set $C_1C_2C_3$=000 to perform its normal operation.



(a)



(b)

Fig. 8

Example IV

Consider the circuit shown in Fig. 8 (a), where z_1z_2 are the observable primary outputs. We first define the level of each gate according to the definition, then the circuit is converted to a easily detectable circuit as shown in Fig. 8 (b) by applying the systematic procedure. The label for each line is used to define the Type I fault and Type II fault. The input test pattern (abcdefg)=($C_2C_2C_1C_1C_2C_2C_2$) is generated. Two tests are applied as follows:

Test 1 Set $C_1C_2C_3$=011 and apply the test pattern (abcdefg)=(1100111). If z_1z_2=11, then the circuit does not have any Type I fault, where Type I fault = $[a_0, b_0, c_1, d_1, e_0, f_0, g_0, h_0, i_0, j_0, k_1, l_1, m_0, n_0, z_{10}, z_{20}]$. Otherwise, it has at least one.

Test 2 Set $C_1C_2C_3$=101 and apply the test pattern (abcdefg)=(0011000). If z_1z_2=00, then the circuit does not have any Type II fault, where Type II fault= $[a_1, b_1, c_0, d_0,$

$e_1, f_1, g_1, h_1, i_1, j_1, k_0, l_0, m_1, n_1, z_{11}, z_{21}$]. Otherwise, it has at least one.

III. FAULT MODEL OF MEMORY ELEMENT (FLIP-FLOP)

Each flip-flop has its own truth table. To determine the accuracy of flip-flop, it is not necessary to find out where is wrong inside the flip-flop. The important aspect is to know whether the flip-flop does function properly according to its desired truth table or not. If the flip-flop can not function correctly for certain excitation, then we say that the flip-flop has malfunction. The malfunction of flip-flop can be classified, in general, into four categories:

1. Type A malfunction is the flip-flop fails to remain at its previous state.

2. Type B malfunction is the flip-flop fails to change its previous state.

3. Type C malfunction is the flip-flop fails to reset to 0 state.

4. Type D malfunction is the flip-flop fails to set to 1 state.

Consider a D-flip-flop, it will only have Type C and D malfunctions. Type C malfunction is caused by the failure of failing to reset to 0 state when D=0. Type D malfunction is caused by the failure of failing to set to 1 state when D=1. The T-flip-flop will only have Type A and B malfunctions according to its truth table. The JK-flip-flop will have all of the four possible malfunctions. If the flip-flop does not have the malfunctions described above, then it is ensured to function properly under all the possible inputs unless there exists a intermittent (transient) fault inside the flip-flop.

IV. SYNCHRONOUS SEQUENTIAL NETWORK

The difference between sequential circuit and combinational circuit is that the sequential circuit contains memory elements in addition to the combinational logic. In a sequential circuit, the present outputs depend on the previous states and/or present primary inputs, and the state transitions depend on the previous states and primary inputs of the circuit. Consequently, present fault may not affect the present outputs. Hence we cannot in general detect the faults by a single test pattern. It requires a sequence of test pattern to detect the faults in a sequential circuit and the length of the test sequence is proportional to the number of memory elements in the circuit. There are several techniques to reduce the length of the test sequence by means of additional control logic circuitry. In the following discussion, we introduce a new aspect of control circuitry to detect the faults in synchronous sequential circuits. In addition to the PLM's, we propose the controllable memory modules for the memory elements as follows:

A memory element with direct set and reset capability whose outputs are convertible by two external control signals is called the controllable memory module (CMM).

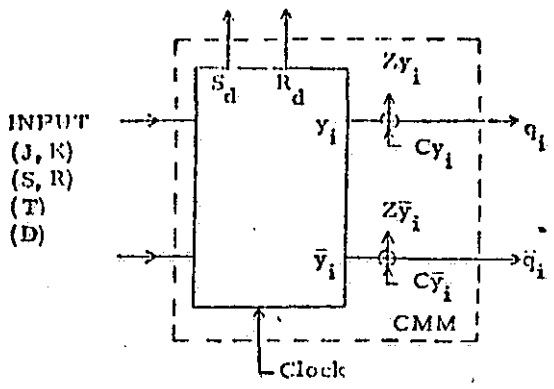


Fig. 9

A general model of CMM is shown in Fig. 9. If the flip-flop is DFF (of TFF, JKFF, SRFF, etc.), the input is D/ (or T, JK, SR, etc.) and the module is denoted as DCMM (or TCMM, JKCMM, SRCMM, etc.). Its outputs are q_i and $\bar{q}_i$ (where $q_i = y_i \oplus Cy_i$, $\bar{q}_i = \bar{y}_i \oplus C\bar{y}_i$) and $Cy_i/C\bar{y}_i$ depends on the control signals of its successor PLM and direct set and reset signals of the flip-flop, and Zy_i and $Z\bar{y}_i$ are observable outputs.

The CMM will operate like the pure flip-flop unit in the CMM when the control signals are set to o. For testing, we set its control signals according to Table III. The failure of CMM will be contributed by the faults of its control portion and the flip-flop unit. In the following discussion, the fault in CMM is restricted to the single permanent fault.

Successor of the Flip-Flop	$S_dR_d = 10$	$S_dR_d = 01$
PLM 1/PLM 3	$Cy_i = C_j$	$Cy_i = \bar{C}_j$
	$C\bar{y}_i = \bar{C}_j$	$Cy_i = C_j$
PLM 2/PLM 4	$Cy_i = \bar{C}_j$	$Cy_i = C_j$
	$Cy_i = C_j$	$C\bar{y}_i = \bar{C}_j$

Table III

C_j is the control signal of PLM.

Consider the TCMM as shown in Fig. 10. The Type A malfunction can be caused by T_1 or y_1 (when the module is previously reset) or Cy_1 (Cy_0) or q_1 (q_0) or the equivalent fault. It can be detected by first directly reset the module and then apply T=0 and Cy=0 (Cy=1). Type B malfunction can be caused by T_0 or y_0 (when the module is previously reset) or Cy_1 (Cy_0) or q_0 (q_1) or the equivalent fault. It can be detected by first directly reset the flip-flop and then apply T=1 and Cy=0 (Cy=1). The malfunctions of JKCMM can be detected by the similar concept. So the malfunctions of CMM can be detected by certain input pattern. The application of using CMM to enhance the testability of synchronous sequential circuit is illustrated by the following example.

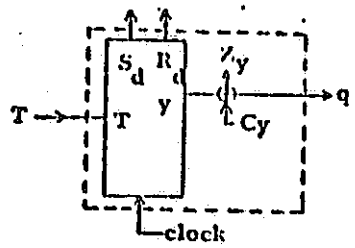


Fig. 10

Example V

Consider a four-bit synchronous binary counter implemented by T-flip-flop as shown in Fig. 11 (a). It is transformed to the circuit as shown in Fig. 11 (b) by substituting the ordinary gates and T-flip-flops with the proper PLMs and TCMMs to enhance its testability, where an additional module is inserted to monitor the behavior of combinational logic portion and z_m is the monitoring output. All the control signals are set to 0 for normal operation. For testing, four tests are applied as follows:

Test 1 (S_dR_d=10) Set C_i=0 and Cy_i=0 for i=0,1,2,3. If z_m=1, then there exists no Type I fault, where Type I faults= $[a_0, b_0, c_0, d_0, e_0]$. Otherwise the fault information will be sensed and drives z_m to 0.

Test 2 (S_dR_d=00) Let x=1 and keep the control signals unchanged. If $q_0q_1q_2q_3$=0000, then no TCMM has type B malfunction. Otherwise, at least one TCMM has Type B malfunction.

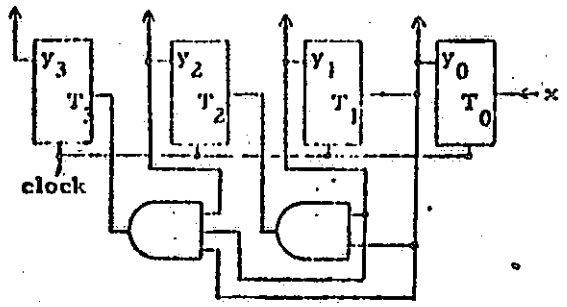
Test 3 (S_dR_d=01) Set C_i=1 and Cy_i=0 for i=0,1,2,3. If z_m=0, then there exists no Type II fault, where Type II faults= $[a_1, b_1, c_1, d_1, e_1]$. Otherwise the fault information of Type II will force z_m to 1.

Test 4 (S_dR_d=00) Let x=0 and keep the control signals unchanged. If $q_0q_1q_2q_3$= 0000, then no TCMM has Type A malfunction. Otherwise, at least one TCMM has Type A malfunction.

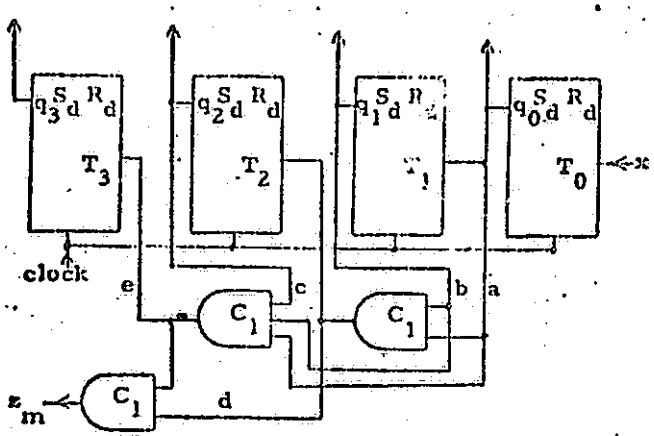
As long as the circuit passes these four tests, then the logic gates are guaranteed to be free of stuck faults, and the TCMMs are guaranteed to function properly. It is worth to point out the fact that the Exclusive OR unit in TCMM is not required in most of the shift register and counter circuits which are implemented by using TCMM. It is also true for DCMM. It is important to note that the procedure proposed here is independent of the number of flip-flops and gates in the circuit under test.

V. CONCLUSION

The application of PLMs, CMMs are proposed which will transform a circuit to certain topological structure to augment the testability of the circuit. Two tests can detect any combinational circuit with observable outputs. Systematic design and detection procedure for combinational network are presented. Four tests are sufficient for fault detection of synchronous sequential circuit implemented with T-flip-flops or D-flip-flops. The same concept can be extended for synchronous sequential circuit using JK-flip-flop or SR-flip-flop, and for synchronous sequential circuit using mixed type of flip-flops. Systematic design and detection procedure of any synchronous sequential circuit is more complicated than that of combinational circuit and requires further investigation.



(a)



(b)

Fig. 11

REFERENCES

[1] Williams, M. and Angell, J., "Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic," IEEE Trans. on Computers, Vol. C-22, Jan. 1973, pp. 46-60.

[2] Sakauchi, M. and Inose, H., "Synthesis of Fault Diagnosable Logical Circuits by Function Conversion Method," JIEC Trans. C, Jan. 1973, pp. 47-54.

[3] Hayes, J., "On Modifying Logic Networks to Improve Their Diagnosability," IEEE Trans. on Computers, Vol. C-23, 1974, pp. 56-62.

[4] Kautz, W., "Fault Testing and Diagnosis in Combinational Digital Circuits," IEEE Trans. on Computers, Vol. C-17, 1968, pp. 352-366.

[5] Roth, J., "Diagnosis of Automata Failure: A Calculus and A Method," IBM Jour. R and D, Vol. 10, 1966, pp. 278-291.

[6] Su, S. Y. H. and Cho, Y. C., "A New Approach to the Fault Location of Combinational Networks," IEEE Trans. on Computers, Vol. C-21, 1972, pp. 21-30.

II. 2 Parallelism Exploitation in Parallel Computing Systems

T. Hsu and S. J. Oh

II. 2-1. Introduction

This report summarizes the result of an investigation on parallelism exploitation in parallel computing systems. The basic need for using such a system is to increase the processing speed for real-time application, without excessively increasing the cost or control complexity of the system.

The discussions will center on three major topics,

1) Parallelism detection
2) Task scheduling
3) System configuration

II. 2-2. Parallelism Detection

Parallelism refers to the fact that two or more subprocesses of a computing task may be done simultaneously. A necessary and sufficient condition for this to be true is that the input parameters of one process are not functions of the output parameters of another, and vice versa. Under this definition, although not always separable, there are two basic types of parallelism in most computing problems:

1) Algorithmic parallelism: This is the type of parallelism where many independent but similar or identical computations are done as required by the computing algorithms. The parallel algorithm could be a direct adaptation of a serial algorithm, or could be one created mainly for efficient parallel processing. Typical examples are matrix multiplication, array data processing, computation of fast Fourier transforms, etc. Algorithmic parallelism is reflexed in sequential programs (such as Fortran) by DO loops, although careful distinction should be made between the recursive type (where input data are updated as indices are changed), and the non-recursive (or array) type.

2) Operational parallelism: This is the type of parallelism in which, although not always evident, exist several subcomputations which either bear no data transfer relations, or are not affected by the precedence order of their executions. It is therefore possible to detect these subcomputations and make them concurrent operations. This type of parallelism generally exists in any sequential programs, both between statements and within a statement. Good detection algorithms are essential for a). designing efficient compilers for conventional uniprocessor systems, and b). generating parallel subtasks suitable for concurrent operations in parallel processing systems.

II. 2-3. Task Scheduling

While the parallelism detection algorithm may generate all possible concurrent processes regardless of how they can actually be utilized by the system, it is the function of the scheduling mechanism, either by software or hardware, to properly match these two together and result in an optimal or near-optimal performance in terms of computation time, which is usually the criterion for the measurement of scheduling strategies.

Principle of scheduling can be applied at different levels of a computational process, from program segmentation in a multiprocessor system, to the execution of microinstructions by several arithmetic units. With slight variations, each process can be properly modeled. In a parallel processing system, we are specially interested in the scheduling of tasks at the machine instruction level, and a deterministic model in the form of a job graph is generally used. Each task node represents an instruction execution, all linked by the precedence

relations, and the node weight corresponds to its execution time-length.

Several formal scheduling techniques exist, although most of them are both complicated and lengthy for practical application except for some simpler special cases like:

1) when the graph is a tree and all nodes have equal weights.

2) when there are only two processors to be scheduled and all tasks have equal weights.

For more general problems, experimental results have shown that optimal or near-optimal results can often be obtained by using much simpler heuristic algorithms. Two of the most prominent ones are:

1) Critical path scheduling: for the nodes waiting to be processed, those on the critical path are scheduled first in accordance with the number of processors available.

2) Largest-processing-time algorithm for independent tasks: tasks are processed in the order of decreasing task weights.

II. 2-4 System Configuration

While the conventional uniprocessor system operates on single-instruction single-data stream (SISD), parallel processing and multiprocessor systems have structures to exploit single-instruction stream multiple-data stream (MIMD). Specialized systems under the names of array processors, pipeline processors, and associative processors, etc. are variations of the SIMD type, while a number of uniprocessors interconnected under various schemes belong to the MIMD type. Data processing, data access, and data alignment play major roles in the determination of system configuration. Existing SIMD systems have up to hundreds of processing units while MIMD systems are limited to a parallelism of 16 or so.

II. 2-5 Conclusions

We conclude this report with some comments:

1) Efficient parallel algorithms, which in many cases may not

be just simple adaptation of the serial counterparts, have the gross impact on parallel computing, and operational parallelism aids to speed up the computation at the lower level.

2) In practical and real-time applications, simple scheduling techniques prove to be satisfactory. Formal algorithms can be used to establish the optimal standard for comparison.

3) It is desirable to have a system's capability to reconfigure for computing purposes, but the increased problems seem quite prohibitive. Since a close relation between the computing algorithm and the system capability is vital for efficient processing, it appears that a basic need exists to identify the class of problems for application and base the system configuration on the most applicable and efficient algorithms. The advancement of microprocessor application may also point to this direction.

II. 3 An Experimental Simultaneous Multiprocessor Organization

G. S. Mersten and S. J. Oh

The purpose of this research effort is to develop the basic requirements of a highly parallel information processing system and the specific development and implementation of an advanced experimental computer organization concept entitled, "Simultaneous Multiprocessor Organization", abreviated "SAMSON". Some of the areas to be studied encompass:

Parallelism and parallel processors,
SISD stream processors,
SIMD stream processors,
MIMD stream processors and,
MISD stream processors.

The primary effort to date has been the development and implementation of one of the uniprocessors of the SAMSON computer system. This uniprocessor will be used at this time to collect data to determine the preliminary SAMSON configuration.

The SAMSON uniprocessor is a general purpose, parallel digital central processor providing a full parallel sixteen bit arithmetic structure utilizing multiple general purpose accumulators and a microprogrammed control unit.

The Arithmetic Unit of the SAMSON uniprocessor provides the capability to perform arithmetic and logical operations on the various machine registers and memory. The information from the general purpose accumulators, memory, program counter and input bus are routed through the arithmetic unit under control of the Micro-Control Unit. The Arithmetic Unit has been built and is fully operational.

The Micro-Control Unit of the SAMSON uniprocessor is the heart of all the processor control and timing signals. The major element of this unit is a micro-control memory implemented with

LSI programable read only memories (PROM,'s). The content of this memory specifies which of the many machine operations is to be performed during every phase of every instruction. The Micro-Control Unit has been built and is fully operational. It is anticipated that the instruction set presently microprogrammed will be altered as this research effort progresses.

The Memory Interface Unit consists of the control and interface logic required to interface the uniprocessor with a commercial core memory. This interface is built and is fully operational; it interfaces with a 4K core memory.

The status of the uniprocessor is displayed by means of the SAMSON Control Panel which provides display, control and loading capability; thus enabling the operator to control and observe the operation of the various uniprocessors. Presently the control panel interfaces with only one uniprocess. As additional uniprocessors are added, the control panel will expand to accommodate the additional uniprocessors. The Control Panel has been built and is operational.

A commercial paper tape reader has been interfaced via the Control Panel and paper tape reader interface. This interface is fully operational. In addition, a teletype interface has been built; however the teletype has not been received yet.

A Micro-Memory Monitor has been built and is fully operational which allows monitoring the sequence of micro-memory executions.

Present efforts are being directed toward the development of data collection software to determine the desireability of specific features of the SAMSON architecture.

Appendix for II.2.

Compilation and Schduling of FORTRAN Programs for Parallel Processing Systems*

Terry T. Hsu
and
S. J. Oh

Department of Electrical Engineering
The City College of The City University of New York
New York, New York

*This work was supported in part by NASA-Houston under the NASA Contract NAS9-13940.

Compilation and Scheduling of FORTRAN Programs for Parallel Processing Systems

I. Introduction

Two phases of preprocessing (i.e. compilation and scheduling) of Fortran programs are discussed. The purpose is to make the program more efficiently executable on a parallel processing system by exploiting as many parallel-processable computations as possible. It is hoped that this will increase the throughput as well as system hardware utilization.

Since only the speedup due to concurrent arithmetic operations is concerned, it is necessary to examine the whole program and sort out those statements which involve direct arithmetic computations. Namely, we shall deal only with the following three types:

1. Assignment Statements
2. IF Statements involving arithmetic operations
3. DO Statements.

II. Compilation and Scheduling of assignment Statements.

We call a Block of Assignment Statements (BAS) as a group of Statements consisting of only assignment Statements. They are not necessarily continuous Statements in the original program, but should preserve the order that they appear in the program. The order is important because a variable may be updated before or after it is used.

1. Parsing of statements.

A common way to show the syntactic structure of an arithmetic expression for the purpose of compilation is to use a syntactic tree where each node represents a binary operation on nodes or operands of previous levels. Many different techniques have been developed to construct a systactic tree [1], [2] among which, Baer and Bovet's technique [3] using operator precedence order is in general considered the best to produce a tree with minimum height. Other algorithms using distribution law as developed by Muraoka, Kraska, and others [4], [5], [6], [7] can sometimes be used to further reduce the tree-height. But the complexity of their algorithms prohibits a practical application. Therefore, only operator precedence orders will be considered, and they are, in decreasing order:

$(\times, \div)$, $(+, -)$, parentheses ( ).

We first define a number of terms:

Def.: A tree is a set of nodes and arcs (edges) in a leveled structure such that each node at level j has two or more inward arcs emanating from nodes at level i, where $i \leq j-1$, and each node at level j has only one outgoing arc to a certain node at level k, where $k \geq j+1$. Nodes at level 0 have no inward arcs, and are called leaves or initial nodes.

There is only one node at the bottom level of the tree. It has no outgoing arcs, and is called the root of the tree, or terminal node.

For sake of brevety, unless otherwise specified, a node will generally refer to one of those between the initial and terminal levels.

Def.: If a node N_i has a chain of arcs leading to another node N_j, N_j is then called a successor of N_i, and N_i a predecessor of N_j, denoted by $N_i < N_j$. If there is only one arc between N_i and N_j, they are called immediate predecesor and immediate successor to each other.

Def.: A binary tree is a tree where each node has exactly two inward arcs.

Def.: A syntactic tree is a binary tree representing an arithmetic statement, where each node represents a binary operation and its result on the data carried from the two immediate predecessors by the inward arcs. Sometimes the terms "nodes" and "operands" (which the nodes represent) are used interchangeably.

It is apparent from above definitions that a tree can not form a loop among its nodes.

The following is an algorithm for generating a syntactic tree of an assignment statement.

1). Place all the right-hand-side (RHS) variables and their associated operators at level 0.

2). Scan through the RHS of the Statement from left to right. If two operands can be joined by the operator between them without violating the operator precedence order, a new node

representing each one operation is created and placed at level 1.

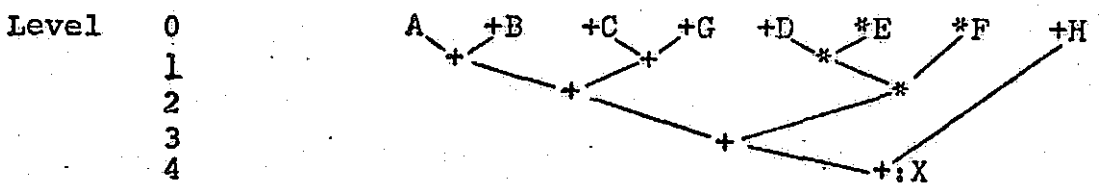
If an operand can be joined with more than one operand, the one scanned first has higher priority.

3). Scan through the nodes in level 1 and those not yet combined in level 0 from left to right. Again combine node pairs where operator precedence orders permit, and place them as new nodes in level 2.

4). Proceed in the same way to combine nodes in level 2 and so on until all operands of the RHS are combined and a single node results at the root which is the desired left-hand-side (LHS) variable of the assignment statement. The syntactic tree is now completed.

An example of parsing is shown in the following.

X=A+B+C+D*E*F+G+H



It is obvious that a syntactic tree of n initial nodes must have at least $\lceil \log_2 n \rceil$ levels, where $\lceil x \rceil$ denotes the least integer such that $\lceil x \rceil \geq x$.

2. The BAS graph

We now extend the structure of a syntactic tree to include all statements in a BAS. Again a few terms are first defined.

Def.: Let S_i be the i^{th} statement in a BAS, then $I(S_i)$ is the set of input variables contained in the RHS of S_i, and $O(S_i)$ is the output variable of S_i, i.e. the LHS of S_i.

We observe the following:

If i) $O(S_i) \in I(S_j)$, or

ii) $O(S_j) \in I(S_i)$, where $i<j$,

then S_j can be completed only after S_i is executed. Otherwise S_i and S_j can be executed simultaneously.

In above, condition i) indicates that S_j depends directly on the outcome of S_i, and condition ii) indicates that $I(S_i)$ will be updated by S_j at a later time, hence S_i must be executed before S_j.

The statements in a BAS are parsed in the following way.

1). Construct the syntactic tree of the first statement according to the procedures described in Sec. 1.

2). Go to the next statement. If $I(S_2) \cap I(S_1)=0$, and $I(S_2) \cap O(S_1)=0$, construct the syntactic tree for S_2 as before, with all its initial nodes at level 0.

Otherwise, S_1 contains one or more input variable or subexpression (part of the arithmetic expression) which has been encountered in S_1 and represented by some nodes (initial, terminal, or in between) in the first tree. Hence, in constructing tree of S_2, those operands are taken from appropriate nodes of the tree of S_1. For any operation that combines N_i and N_j, where i and j indicate the levels of the nodes, and $j \geq i$, the resulting new node is placed in level j+1.

3). Go to the next statement and repeat the same process except that the input variables are now checked against nodes of all previously generated trees to determine if any dependency exists. The process continues until all statements are parsed.

We now have a "tree complex" of the BAS. Since some nodes may now have more than one outgoing arc and the whole tree complex can have several terminal nodes, we shall more properly call it a BAS graph.

Def.: A BAS graph is a directed, acyclic graph consisted of interconnected syntactic trees. It shows all the intra-statement and inter-statement operations.

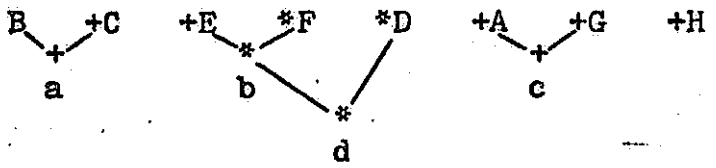
A few points are to be noted:

1). All nodes at same level in a BAS graph are operationally independent and hence can be executed simultaneously.

2). If a node has more than one outgoing arc, its information either has to be shared by more than one PE at same time (if immediate successors are at same level), or must be stored for later use (immediate successors at different levels).

3). Since the parsing of statements is done sequentially in order, it may not produce a tree whose intermediate nodes can be directly used for subsequent statements, although such a structurally different but functionally equivalent tree is possible and more desired. For example, for a BAS which contains a statement used in the last example, namely, X=A+B+C+D*E*F+G+H and another statement Y=B+C+E*F*K, if RHS

of first statement is parsed as follows(shown partially) instead of what was shown previously,



then nodes a and b are readily usable by the second statement while make no difference to the first one.

This shows that to make even more use of the inter-statement parallelism, other parsing algorithms are possible. But its value being offset by the added complexity in searching for identical suboperations among different statements is questionable for practical purpose, except in some simple programs where parsing is done by inspection.

3. Principle of Scheduling

To facilitate later discussions, we define the following terms.

Def.: A graph G is called a relaxed graph, denoted by G_R, if all the terminal nodes are placed in the bottom level (which has the largest level number in G_R, assuming to be q), and all other nodes are placed in a level such that the distances (i.e. the difference between level numbers) between each and its immediate successors are minimized (the minimum being 1).

Def.: A node weight w_i is the number of time units required to perform the operation of node n_i.

Def.: The largest backward path value W_i of a node n_i is defined by $W_i=w_i+\max\left[W_j \mid n_j>n_i\right]$

It is the largest sum of node-weights between n_i and its terminal nodes over all possible paths.

Def.: The largest forward path value D_i of a node n_i is defined by $D_i=w_i+\max\left[D_h \mid n_h<n_i\right]$.

It is the largest sum of node-weights between n_i and its initial nodes.

Def.: The critical path value C_q of G_R having q levels is defined as $C_q=\max\left[W_i \mid n_i \in G_R\right]$.

The chain of arcs resulting in C_q is the critical path.

Def.: Let i be the level number in a G_R, $1\leq i\leq q$, and there be m nodes contained in G_R from level 1 to level i. The partial node-weight sum, P_i, is defined as

$$P_i=\sum_{1}^{m} W_j.$$

The scheduling of nodal operations is based on a relaxed graph with added information defined above. The resulting graph is commonly called a job graph. We describe the generation of a job graph from a BAS graph next.

1). Place all terminal nodes of G in the bottom level q of the graph.

2). All nodes in G whose successors are only found in level q are placed in level q-1.

3). Repeat this process upward in decreasing level order so that all nodes which have successors only in levels greater than i

are placed in level i of G_R. The process terminates only when all nodes are adjusted. All arcs in G_R retain their relations with nodes as they have in G.

4) Include node weights w_i for all nodes $n_i \in G_R$.

5) Compute the largest backward path value W_i for all $n_i \in G_R$. This is easily done starting with nodes in level q and going upward. In general, W_i is found by adding w_i to the largest of W_j's where n_j's are the immediate successors of n_i. The largest W_i in the graph is the critical path value, which also determines the critical path.

6) Compute the largest forward path value D_i for all nodes. This is done starting with nodes in level 1 and going downward. D_j is found by adding w_j to the largest of D_i's where n_i's are the immediate predecessors of n_j.

The following theorem by Kraska [5] determines a lower bound on the number of PE's required to execute the job represented by G_R in critical time C_q.

Theorem: If $\max\left[P_i/D_i \mid i \leq q\right] > m-1$, at least m PE's are required to process all nodes of G_R in C_q time.

Using a job graph, there exist at least two typical optimal scheduling techniques using either forward tracing [8] or backward tracing [5] to schedule the operations of nodes on a fixed number of PE's so that the total execution time is minimized. However, both techniques are too complicated and time-consuming for real-time applications. It has been tested and shown

that a rather simple heuristic approach can in many cases be as good as an optimal scheduling. The principle is:

i). Schedule first those nodes which are in the next higher level and/or with highest backward path values.

ii). Keep as many PE's busy as possible by assigning nodes which are processable next.

4. Scheduling and Execution with Operational Stacks

We now apply the principle of heuristic scheduling to store and transfer information from a job graph to a number of stacks so that they can be directly executed. These stacks then define the order and concurrence of operations and are called operational stacks. The number of stacks is set equal to the smaller of the number of PE's available in the system or the number determined from Kraska's theorem cited above. In this way, one PE will be executing instructions from one stack and no redundent PE's are used.

Using the job graph, the stacks are loaded in the following way.

1). Starting from all initial nodes of G_R, $\{N\}$, load operation instructions defined by each node on top of stacks. Two situations may occur:

i). If $\#\{N\} \leq \#$(stacks), some top stacks may be left empty.

ii). Otherwise, load stacks first from those initial nodes which have highest W_i's.

In either case, denote those loaded nodes in this step by $\{n\}$.

2). Examine the immediate successors of $\{n\}$ and those left in $\{N\}$ (i.e. $\{N\}-\{n\}$). Consider those which are parallel

processable (i.e. none are predecessors or successors of each other) as new {N} and load the nodes which have highest W_i's in the next level of stacks until either condition similar to step 1 has occurred. (If a new stack has an empty top level, then the top level is loaded first). In general, when a node n_i spawns to two immediate successors (or more) n_j and n_k, n_j is put (directly) under n_i in the same stack, and n_k in another stack with a predecessor indicator (i.e. (i)K) and n_i has a successor indicator (i.e. i(K)). Similar indicators are used when two nodes n_i and n_j merge into one node n_k, and one of the predecessors (say n_j) is not in the same stack as n_k.

3). Repeat step 2) throughout G_R until all nodes are loaded.

The execution phase of instructions is straight forward. Each stack supplies an instruction sequence to one PE, which is executed serially. All PE's normally run in parallel, independent of one another except when a predecessor indicator occurs before an instruction. This sets up a flag and stops the PE from executing the instruction, unless its predecessors are executed and the successor indicators clearing the flag.

An example of the parsing, operational stacks, and simulated execution of a BAS is shown next.

Example: Assume a BAS as follows:

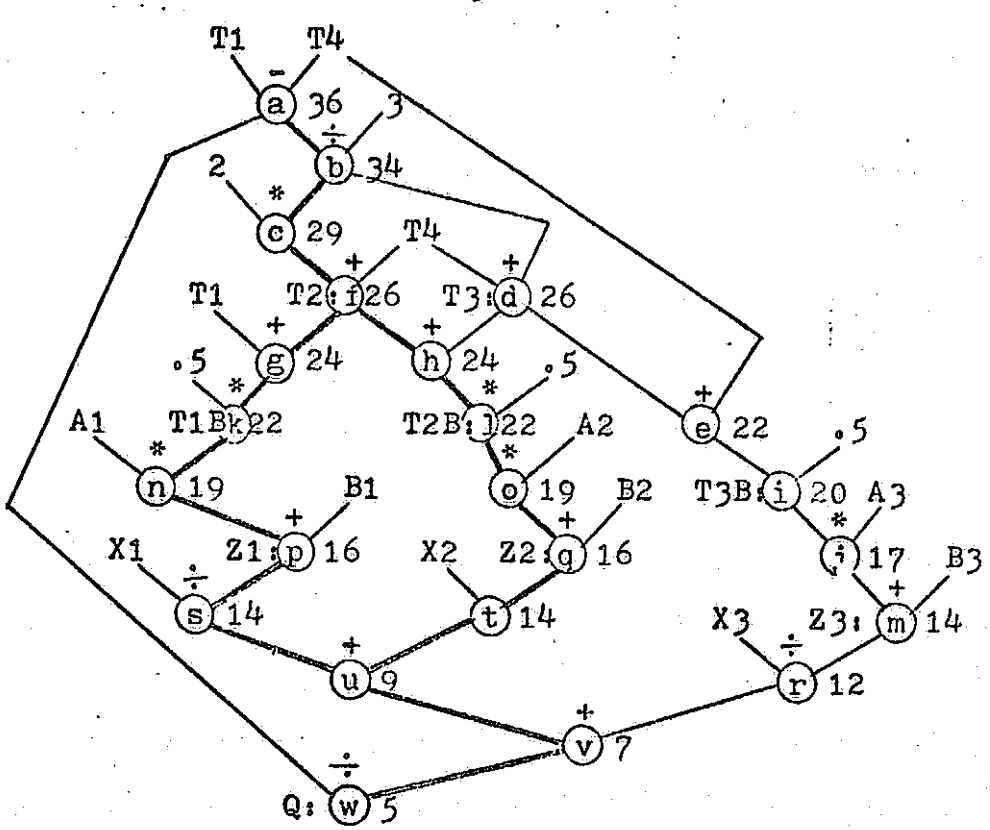
Data: T1, T4, B1, B2, B3, x1, x2, x3.

```
T2=2*(T1-T4)/3+T4
T3=(T1-T4)/3+T4
T1B=.5*(T1+T2)
T2B=.5*(T2+T3)
```

```
T3B=.5*(T3+T4)
Z1=A1*T1B+B1
Z2=A2*T2B+B2
Z3=A3*T3B+B3
Q=(T1-T4)/(X1/Z1+X2/Z2+X3/Z3).
```

The job graph is as follows.

i	Pi/Di
0	0/0
1	2/2
2	7/7
3	10/10
4	14/12
5	18/14
6	26/17
7	35/20
8	42/22
9	54/27
10	61/29
11	63/31
12	68/36



Notes: 1). Assume node weights for $\frac{\circ}{\circ}=5$

$x=3$

$+, -=2$

2). Numbers outside nodes are largest backward path values W_i.

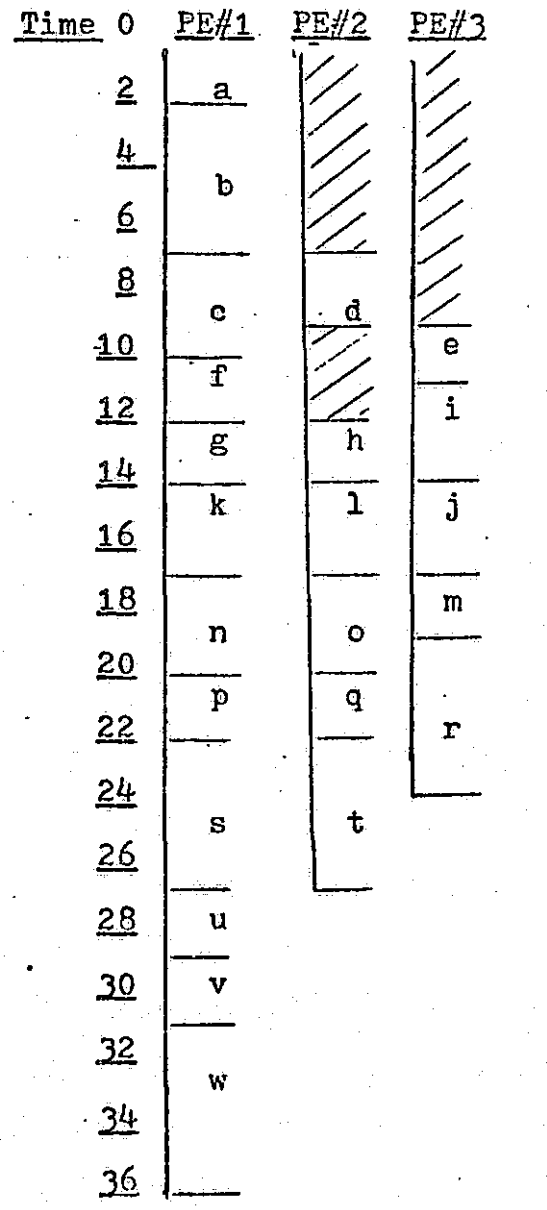
3). Critical path value=36, critical path in heavy lines.

4). Max $\lceil P_i/D_i \rceil = P_{10}/D_{10}=2.1$, hence 3 PE's is the lower bound for achieving $C_q=36$.

Operational Stacks

#1	#2	#3
a	(b)d(e)	(d)e
b(d)	(f)h	i
c	l	j
f(h)	o	m
g	q	r(v)
k	t(u)	
n		
p		
s		
(t)u		
(r)v		
w		

Simulated Execution



III. Preprocessing of IF Statements

Given an IF Statement of the type

IF (s) 1,2,3

where s is an arithmetic or logic expression, and 1, 2, 3 are branching instructions, the most obvious way to make use of parallelism is to parse s for maximum parallel suboperations. For example,

IF (B*B-4*A*C) 6, 8, 9

is converted to X=B*B-4*A*C

IF (X) 6, 8, 9

With added circuitry it is possible to start processing all the branching instructions 6, 8, 9 simultaniously, with the final result determined by the outcome of X. However, since most IF statements have only simple expression as argument, the concurrent operations of branching instructions on one IF statement alone does not cause considerable speedup. More sophiscated algorithms can be used to detect a block of statements with high occurrence rate of IF statements and converts the IF statements into a binary decision tree which is then processed by a decision processor. More assignment statements associated with the IF statements are also collected and processed simultaneously. Details on this subject are discussed by Davis [9].

IV. Preprocessing of DO Statements.

The principle in treating DO loops is to separate the loops into several completely independent DO loops and process them

simultaneously. In order to do this, precedence relations among the statements in the loop are examined. But, in addition to the ways encountered in the treatment of a BAS, since data can be used first and updated next and then again used in the next iteration, input variables have to compare with outputs of not only statements before it but also after it.

1). Compare $I(S_j)$ with $O(S_i)$, where $i<j$. If a match in variables with same index is found, then place an arc from node i to node j.

Also compare $I(S_j)$ with $O(S_k)$, where $k>j$. If a match in variables is found whose index is higher in S_k, there is an arc from node k to node j.

2). If some subgraphs are completely disconnected at the completion of the graph, each subgraph becomes a new DO loop, and they are all parallel processable.

Before looking into each new DO loop, we first define two terms.

Def.: A DO statement is called recursive if it has the form $X_i=f(X_{i-1}, \ldots, X_{i-m}, a_i)$ where m is any positive integer and a_i some vector of parameters. Otherwise the DO statement is non-recursive and called of array type.

We now continue the steps.

3). Within each new loop:

i). If statements are of array type, one statement can be generated for each index value, resulting in many parallel statements. For example,

```
      DO 1  I=1, 3, 1
    1 A(I)=A(I+1)+C(I)*D(I)
```

becomes A(1)=A(2)+C(1)*D(1)

A(2)=A(3)+C(2)*D(2)

A(3)=A(4)+C(3)*D(3)

Three PE's (or groups of PE's) can work on A(1), A(2), and A(3) separately and simultaneously.

The assignment and scheduling then are same as in the case of a BAS.

ii). If statements are of recursive type, backward substitution is used to generate statements which allow simultaneous operations. For example,

```
    DO 1  I=1,5
1   A(I)=A(I-1)+B(I)
    A(1)=A(0)+B(1)
    A(2)=B(1)+B(2)+A(o)
    .
    .
    .
    A(5)=A(0)+B(1)+B(2)+B(3)+B(4)+B(5).
```

which may be computed in only 3 addition times instead of 5, with 3 participating PE's.

It is also to be noted that in this case, A(1) through A(4) are actually included in A(5). Hence it suffices to parse only the RHS of A(5) and in the process appropriate intermediate results are obtained for A(1) through A(4).

An example of decomposing a DO loop is shown next.

```
      DO  6  I=1, 3, 1
1     T(I)=G(I)+M
2     G(I)=T(I)+D(I)
3     E(I)=F(I-1)+B(I)
4     F(I)=E(I)+G(I)
5     H(I)=A(I-1)+H(I-1)
6     A(I)=C(I)+N
```

In the accompanying graph, we observe:

1 → 2 because of T(I) in both.

2 → 4 because of G(I) in both.

3 → 4 because of E(I) in both.

4 → 3 because of F(I) in 4 and F(I-1) in 3 .

6 → 5 because of A(I) in 6 and A(I-1) in 5 .

And since there are 2 completely disconnected subgraphs, 2 new DO loops are created:

```
      DO  4  I=1, 3, 1
1     T(I)=G(I)+M
2     G(I)=T(I)+D(I)
3     E(I)=F(I-1)+B(I)
4     F(I)=E(I)+G(I)
```

```
      DO  6 I=1, 3, 1
5     H(I)=A(I-1)+H(I-1)
6     A(I)=C(I)+N
```

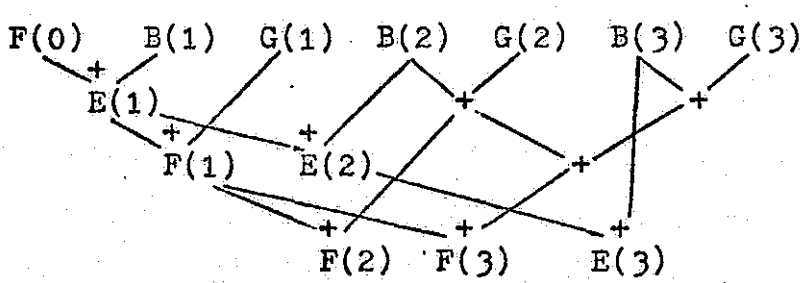
We further decompose each new DO loop as follows:

```
For first subgraph:                          (comment)
    DO  1  I=1, 3, 1
1   T(I)=G(I)+M              generate 3 array equations T(1),T(2),T(3)
    DO 2 I=1, 3, 1
2   G(I)=T(I)+D(I)           generate 3 array equations G(1),G(2),G(3)
    DO 4 I=1, 3, 1
3   E(I)=F(I-1)+B(I)         generate 3 recursive equations E(1),E(2),E(3)
4   F(I)=E(I)+G(I)           generate 3 recursive equations F(1),F(2),F(3)
```

By back-substitution, the recursive equations for E and F are

```
    E(1)=F(0)+B(1)
    E(2)=F(0)+B(1)+B(2)
    E(3)=F(0)+B(1)+B(2)+B(3)
    F(1)=F(0)+B(1)+G(1)
    F(2)=F(0)+B(1)+G(1)+B(2)+G(2)
    F(3)=F(0)+B(1)+G(1)+B(2)+B(3)+G(2)+G(3)
```

Parsing on F(3) produces all the answers for above 6 equations:



For the second loop, 3 array equations and 3 recursive equations are generated in similar ways for statements 6 and 5, respectively. We also observe that all three array equations of S_6 can be executed simultaneously and before S_5.

VI. Conclusions

Some compilation and scheduling aspects of computation-oriented statements for parallel processing have been discussed. The treatments have been confined to rather general cases, so that the results are applicable to most parallel processing systems. Although other theoretically superior algorithms or techniques are available, it is felt that their real-time applications are not so readily justifiable due to the amount of software/hardware sophistication, except possibly in some rather special applications.

References

1. C.V. Ramamoorthy and M.J. Gonzalez, "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs, "Proc. Fall Joint Computer Conference, 1969.
2. C.V. Ramamoorthy, etal., "Compilation Techniques for Recognition of Parallel Processable Tasks in Arithmetic Expressions," IEEE Trans. Comp., Vol. c-22, no. 11, Nov. 1973.
3. J.L. Baer and D.P. Bovet, "Compilation of Arithmetic Expressions for Parallel Computations," Proc. IFIPS, 1968.
4. y. Muraoka, "Parallelism Exposure and Exploitation in Programs," Ph. D. Dissertation, Univ. Illinois, Urbana, 1971.
5. P. Krasks, "Parallelism Exploitation and Scheduling," Ph. D. Dissertation, Univ. Illinois, Urbana, 1972.
6. D. Kuck, etal., "On the number of Operations Simultaneously Executable in Fortran-like Programs and Their Resulting Speedup," IEEE Trans. Comp., vol. c-21, Dec. 1972.
7. D. Kuck, etal., "Measurements of Parallelism in Ordinary Fortran Programs," COMPUTER, Jan. 1974.
8. C.V. Ramamoorthy, etal., "Optimal Scheduling Strategies in a Multiprocessor System," IEEE Trans. Comp., vol. c-21, no. 2, Feb. 1972.
9. E. Davis, "Concurrent Processing of Conditional Jump-trees" COMPCON Digest, 1972.

Appendix for II.3.

An Experimental Simultaneous Multiprocessor Organization*

G. S. Mersten
S. J. Oh

Department of Electrical Engineering
The City College of The City University of New York
New York, New York

October, 1974

*This work was supported in part by NASA-Houston under the NASA Contract NAS9-13940.

1.0 INTRODUCTION

The purpose of this paper is to present a general overview of an ongoing research program in the area of highly parallel information processing system organization and the specific development and implementation of an advanced experimental computer organizational concept entitled, "Simultaneous Multiprocessor Organization", abbreviated "SAMSON". This work is broad in scope, encompassing:

- o parallelism and parallel processors,*
- o completely independent multiple processors or multiple independent single instruction single data (SISD) stream processors,
- o concurrent processors or single instruction multiple data (SIMD) stream processors,
- o task sharing computational processors+ or multiple instruction multiple data (MIMD) stream processors,
- o loosely and strongly-connected multiprocessors or multiple instruction single data (MISD) stream processors,
- o multiprogrammed multiprocessors,
- o real-time multiprocessors,
- o memory organization(s),
- o communications and data routing and,
- o path building and supervisory control.

It is useful to classify the requirements and problems associated with a multiprocessor computer system into two classes, as suggested by Saltzer[1], intrinsic and technological. Intrinsic

---

* This paper uses the term "processor" rather than computer here, to distinguish the structure of the computational processing from the overall computer system which includes input/output, I/O, processing; peripherals; etc.

\+ As distinguished from task sharing with separate processors for I/O, computation, etc.

requirements and problems are inherent in the problem itself whereas technological requirements and problems are transitory in nature, being inherent to the technology at a given point in time. The important intrinsic problems being studied in this research effort are:

- o providing an easy ability to access, transform and share information within a multiprocessor in a controlled manner.
- o to provide protection of this information, and
- o implement an experimental multiprocessor, SAMSON, with these features.

In addition, the technological limitations which impose bottlenecks within a multiprocessor being investigated as a part of this work are:

- o the memory contention problem and,
- o the communication problem.

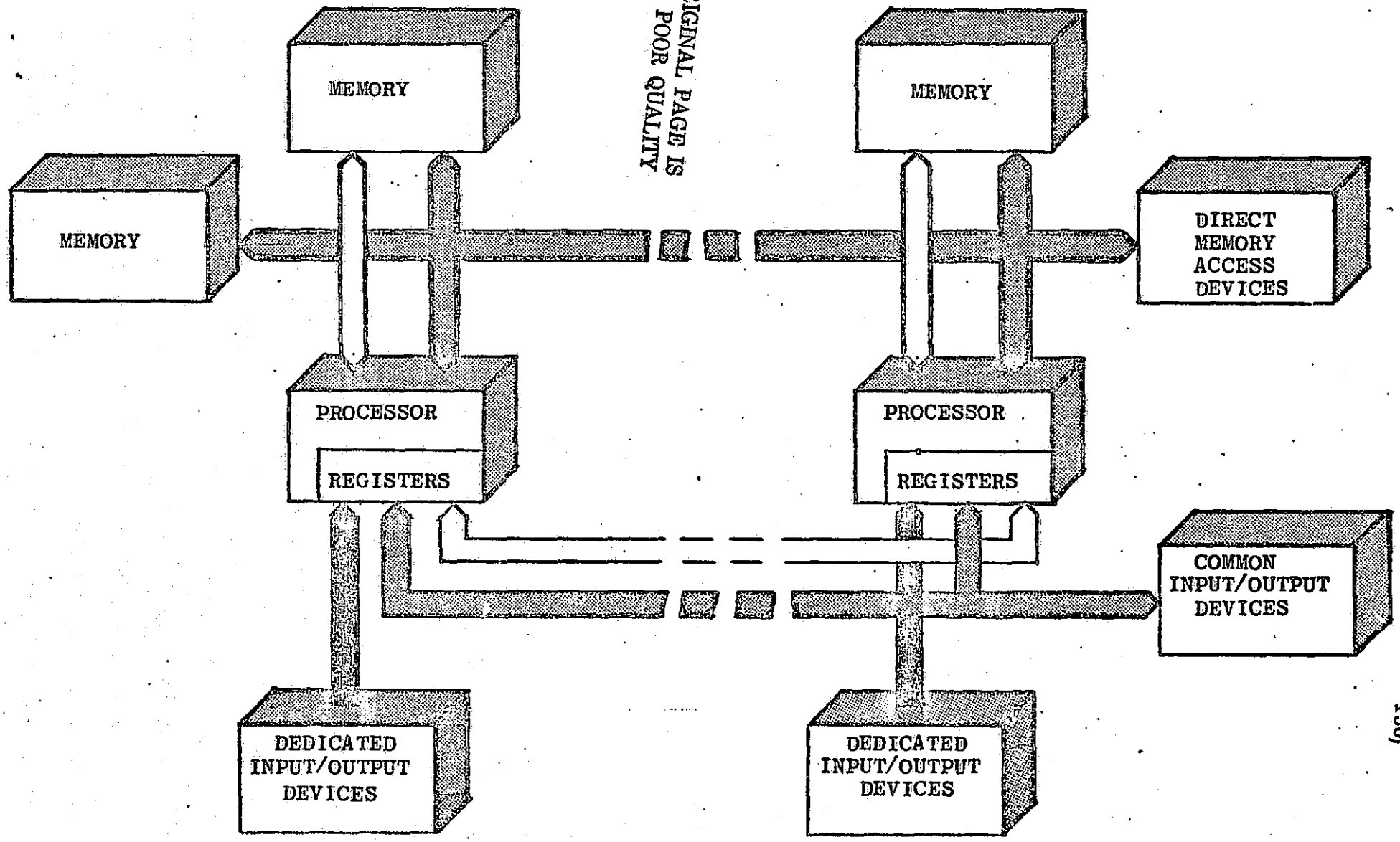
In addition, the SAMSON computer, Figure 1, must be a viable system providing a basis for, and a vehicle upon which, research of advanced multiprocessor organizational and architectural concepts can evolve; that is, a research vehicle for multiprocessor computer system development, capable of supporting a wide range of investigations in computer structures and computer science. SAMSON must be capable of possessing the necessary flexibility and computing capability to assure its applicability to a wide variety of present and future computational applications and to assure the system's continuing evolution and viability.

To meet these goals and to satisfy the widely varying nature of the computational requirements imposed on a simultaneous multiprocessor, the system demands a general-purpose computer organization capable of a certain degree of parallelism.

In this research work a strongly-connected multiprocessor capable of SISD, SIMD and MIMD stream processing (which are special

FIGURE 1

SAMSON BLOCK DIAGRAM



cases of MISD stream processing) as well as being capable of MISD stream processing will be developed and an experimental machine implemented, the SAMSON machine. Particular attention will be given to memory organization and communications in order to eliminate the classical memory contention and path building problems peculiar to multiprocessors. Furthermore, the requirements for multiprogramming and real-time multiprocessing will be investigated and, if practical, one or both of these features, incorporated into the SAMSON architecture.

1.1 MOTIVATION FOR PARALLELISM AND MULTIPROCESSING

There are several reasons which exist as motivation for doing research in the area of multiprocessors. Three of the most obvious reasons being improved reliability, improved maintainability and ease of expandability. However, these motivations, in and of themselves, are not adequate justification for pursuing this research, since duplex systems* (which are not here considered as multiprocessors) offer improved reliability and maintainability while providing an expansion capability through the use of the back-up processor. Thus, additional motivation must exist to justify pursuing this research; this additional motivation is indeed the prime motivation for this research.

This additional motivation results from the ever increasing complexity of present day computational requirements, coupled with

---

* A duplex system usually consists of two identical processors operating so that, in the event of shut downs due to maintenance, checkout, improvements, etc., of one processor, the second can operate without a reduction in capability of the system. In addition, the back-up processor may be utilized to run lower priority tasks or as a slave to the primary processor (in which case a shut down of either will reduce system capability).

the necessity of ever increasing: data processing rates, response time, and capacity. These requirements, in specific application have exceeded and, in general, are gradually exceeding the processing capability of some of our present generation computer facilities.[2]

Furthermore, recognition of the fact that generally much of a large computing task is repeated execution (on similar or dissimilar pieces of data) of the same procedure, leads one to the observation that utilization of this inherent parallelism in a simultaneous multiprocessor organization might well provide a dramatic improvement in computing power.[3]

Consequently, it is the ultimate objective of this research effort to develop a multiprocessor organization capable of a significant increase in computing power through computational parallelism, with increased flexibility, reliability and maintainability, without exceeding the existing limitations of the component technology.

2.0 HISTORICAL BACKGROUND

The concept of parallelism is very broad and includes a wide variety of notions. Intuitively these notions include: many devices working together in some way or other on the same task, devices working together on different but related tasks, and one device performing more than one task at a time. This broad notion of parallelism is, in part, responsible for the wide diversity possible within the area of parallel processing, i.e., SISD, SIMD, MISD and MIMD stream processing.

Prior to a discussion of the historical background of parallel processors, the concepts of global and local control[4] need be introduced.

Global control implies the existence of one or more central control units, either centrally or decentrally located, having

common control over a number of processing elements. Local control implies control of a processing element by its own control unit (either locally or remotely located).

The SOLOMON computer[5] is a classical machine concept consisting of an array of 32 x 32 processing elements, each connected to its four nearest neighbors, under the global control of one central control unit. These processing elements have limited processing power; the central control unit processes a single program with each participating processor executing the same instruction stream on its own data stream, SIMD stream processing. Depending upon the internal state and mode of each of these processing elements, they either participate or do not participate during an instruction execution. The internal state of the processing element is a function of the data it is processing while the mode of the processing element is determined by the control unit. The principle means of control for each processing element is the mode commands. The major disadvantage of this machine organization and of global control structures in general, is its low efficiency when applied to typical general-purpose computations.[6] These computations require sequential execution; consequently, the central control unit utilizes only one (or relatively few) processing elements while all others stand idle.

The ILLIAC IV system[7,8] alleviates the problem of the SOLOMON machine to some degree by: replacing the single global control unit with four such units, by providing I/O access to each processing element, and increasing the power of the processing elements. Each of the four global control units directly governs the operation of an 8 x 8 array of processing units. These global control units may function independently (64 processors per array) or in groups of two, three, or four arrays. These processing elements are all but devoid of local control, mode status and data dependent conditions being the only exceptions.

The HOLLAND machine[9] is a distributed control machine consisting of an array of modules, each of which possesses some measure of local control and a limited capability for independent instruction execution. Within this structure the capability exists (to a limited extent) to execute independent programs simultaneously. HOLLAND proposed the organization to provide a basis for theoretical investigations, not as a practical device. Each module contains a storage register, operand registers and communication paths to its four nearest neighbors. An instruction stream is stored one instruction per module with indicators to denote the predecessor and successor modules (one of the four neighbors). The instructions are executed in time sequence following the above predetermined path. The address of the operand is also stored in the module. The storage register can be loaded from an external source during the first machine cycle. During the second cycle the active module determines the operand location and establishes communications between the operand and the accumulation module. In the final phase the instruction is executed. The proposed attribute of local control structures is the high degree of hardware utilization that can be achieved when many (small) computations are being executed concurrently; however, the HOLLAND machine architecture has fallen far short of its goal. The main problem associated with the local control organization of the HOLLAND machine is the spacial structure, of both the array and of the neighborhood communications, which results in serious path building problems.

In addition to the above mentioned array multiprocessors there have been some noteworthy multiple computer systems which have been developed. These include the PILOT, LARC and STAR Systems.

The PILOT system[10] consists of three independent processors. The basic computer functions are divided, among these three different processors, upon the basis of the processing task. Each processor is optimized to perform one of the following basic functions:

arithmetic and logical operations, control and housekeeping operations, or input-output operations. The processors are independently programmed to perform their own tasks. The PILOT is not here considered a multiprocessor.

The LARC system[11] is a dual computing system consisting of two main computers, a totally time shared memory bus (with a maximum of 39 independent memories), and an input-output processor. The main feature of the LARC system is its memory organization. The memory system consists of a single time multiplexed memory bus and includes such features as overlapping (which makes possible the processing of several instructions concurrently), priority determination and interlocking memory protection. The LARC is, here considered, a limited multiprocessor; the two main computers are capable of operating either independently or jointly to a limited degree.

The STAR computer[12] is a replacement system utilizing a redundant machine organization having one or more (unpowered) spares, which are switched on-line to replace units. To detect and replace these failures, the system utilizes: error detecting codes, software diagnostics, software recovery procedures, transient fault identification (by repetition of program segments), and redundant special purpose processors to monitor, vote and replace subsystems. The STAR is not here considered a multiprocessor.

Other than the ILLIAC IV* none of the above are contemporary processors. The only major contemporary processor under construction is the C.mmp (Carnegie Mellon multi-mini-processor).[13,14] The C.mmp is in its early stages and little can be said in the way of operating features, performance, etc. at this time.

---

*ILLIAC IV is still in the construction and debugging phase; only one of the four quadrants will be constructed.

3.0 DEFINITIONS

The concepts of parallelism and of multiprocessors are both very broad in scope. It is the intent of this section to attempt to define,[15] or at least to place some bounds, on these concepts.

3.1 DEFINITION OF PARALLELISM

Previously the intuitive notions associated with the term parallelism were introduced. The concept of parallelism can best be formally defined by dividing it into two types, applied and natural parallelism. Applied parallelism is the property that enables two or more identical operations within a set of computations to be processed concurrently on the same or distinct data bases. Natural parallelism is the property that enables two or more operations within a set of computations to be processed concurrently and possibly independently on the same or distinct data bases.

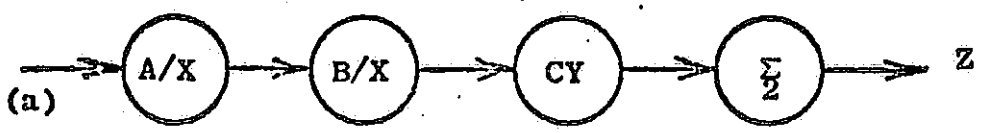
From these definitions it is clear that applied parallelism is just a special case of natural parallelism. However, the distinction is made because of the important impact it has on computer organizations. Applied parallelism is efficiently handled by global control techniques since it provides common control for identical operations. Natural parallelism is efficiently handled by local control techniques since it provides independent processing for different operations. Figure 2 illustrates how the expression

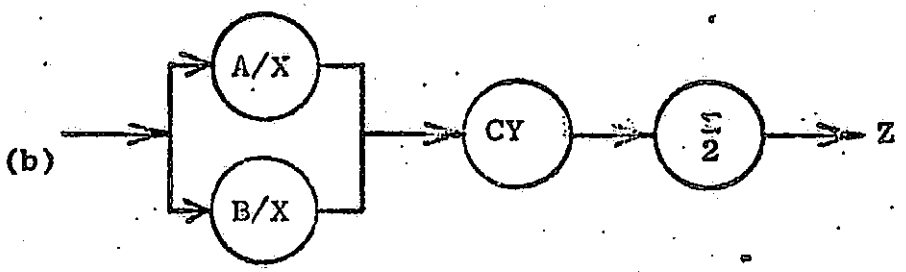
$$A/X + B/X + CY = Z$$

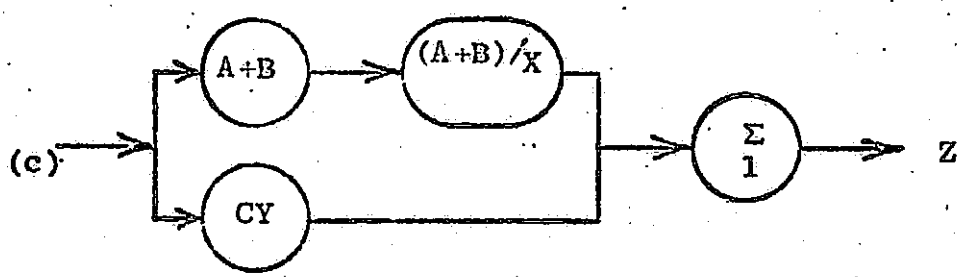
is computed with and without the use of applied and natural parallelism.

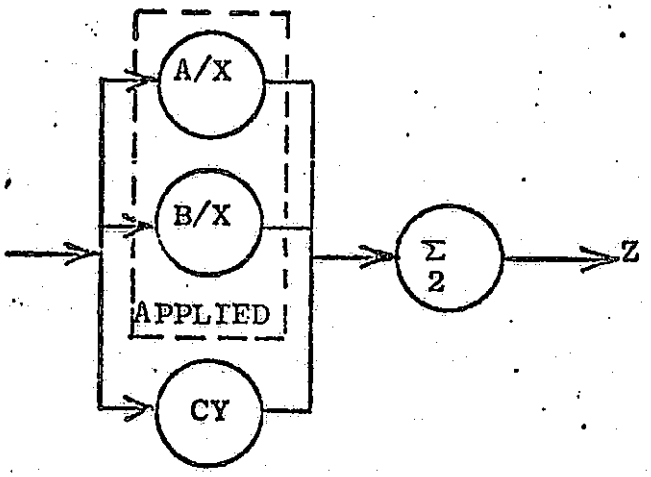
The SOLOMON machine discussed above is based upon the principles of applied parallelism while the Holland machine is based upon the principles of natural parallelism.

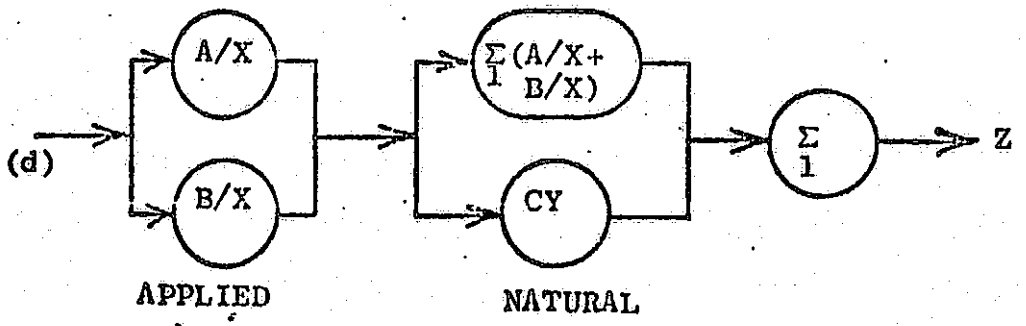
FIGURE 2 Computation of the Expression A/X + B/X + CY = Z in (a) sequential steps, (b) utilizing applied parallelism, (c) utilizing natural parallelism, and (d) utilizing applied and natural parallelism.











3.2 DEFINITION OF MULTIPROCESSOR

Much controversy exists in regard to the definitions of multiple computer systems. Frequently, the terms multiple computer, multiple processors, parallel processors, multicomputer and multiprocessor are used interchangeably; in this work a distinction is made and their broad definitions presented.

MULTIPLE COMPUTERS, MULTIPLE PROCESSORS or PARALLEL PROCESSORS as defined in this work are general terms which are used interchangeably and apply to any system containing two or more, not necessarily identical digital computers. These digital computers may be elementary processing units or full scale digital computers; they may be conventional or unconventional, general purpose or special purpose, and they may operate jointly or independently of one another.

MULTICOMPUTERS as defined here is a group of two or more, not necessarily identical, digital computers interconnected via their input-output system (i.e. obtaining intercomputer communication by considering other computer(s) as peripheral devices). These digital computers may be elementary processing units or full scale digital computers; they may be conventional or unconventional, and they may be general purpose or special purpose.

MULTIPROCESSORS as defined here is a group of two or more, not necessarily identical, digital computers interconnected on an integrated basis (to effect intercomputer communications) capable of operating concurrently on the same or similar phases of a processing task. Similarly, these digital computers may be elementary processors or full scale digital computers, conventional or unconventional, and/or general purpose or special purpose. (The requirement of operating on similar phases of the processing task simultaneously eliminates the class of multicomputers having independent processors for arithmetic functions, control functions, and

I/O functions from being here considered as multiprocessors.)

Bauer's[16] definition of a multiprocessor (which he referred to as a multi-computer system) required that it satisfy the following four criteria:

> "1. There are two or more separate arithmetic control units capable of operating simultaneously and with special hardware facility for operating two or more programs simultaneously.
>
> 2. There are two or more indepently operating primary random access memories.
>
> 3. Communications among major elements of the system is of the memory-to-memory type at memory access speeds.
>
> 4. All major components of the system are in use during normal operations."

In this work Bauer's criteria for defining a multiprocessor are amended as follows: Criterion 2, is intended to imply that two or more quantities can be accessed simultaneously; however, Bauer used this criterion to eliminate from contention systems having one large random access memory. Systems having one such large random access memory, capable of being accessed by more than one processor simultaneously (i.e. by interleaving), as well as, systems having multiport memories should not be, and are not here, elimated from being classified multiprocessors. Therefore, criterion 2 should read: Two or more independent quantities can be accessed simultaneously.

Criterion 3 does not apply to systems which provide for access to a common memory. Furthermore, register to register type communications should be respected as a possible means of intermachine communications. In fact, since minimum communication time is frequently a major multiprocessor goal, register to register communications is an extremely desirable feature and indeed should be a sought after objective in a multiprocessor architecture.

The frequency of use of all or most of the major components of the system should be sufficiently high to justify a multiprocessor. (Unfortunately, whether or not criterion 4 is satisfied, assuming a reasonable architecture, is largely a function of the programmer's ability and ingenuity, his familiarity with the system, and the nature of the programs for which the system is utilized.) Alternatively, the extent to which a multiprocessor utilizes its hardware efficiently could be considered secondary to the speed advantage hoped to be gained by a multiprocessor organization.

4.0 SAMSON CONFIGURATION

Throughout the development of such an experimental multiprocessor, evaluations of various design alternatives and a determination of the effects of various parameters on total system performance will be studied. Through this effort we will be advancing our knowledge of the type of architectural and programming techniques required of future generation parallel processors to achieve increased effectiveness, performance and computational capabilities.*

No one machine organization can handle widely divergent tasks with equal capability. However, the multiprocessor should be capable of handling a broad range of tasks more efficiently, at least time wise, than a uniprocessor. These tasks should not be so broad that the efficiency of any particular task be seriously impaired. In the worst case a multiprocessor should be no worse

---

* No specific programming development is intended as a part of this research; however, insight will be gained regarding programming techniques during architectural development.

than a general purpose uniprocessor.

Hardware utilization efficiency must be secondary to computing power if system versatility is sought. If the tasks to be handled are highly restricted and well defined in advance, then special purpose hardware can be utilized to optimize the processing tasks. However, use of such special purpose machines, on tasks not originally included in the limited set of tasks considered, would be highly restrictive and inefficient.

Consequently, if versatility is a goal, a general-purpose architecture is called for and the related-inefficiencies associated with particular tasks must be accepted. A general purpose architecture tends to increase total execution time; this is equally true of uniprocessors and multiprocessors.

4.1 OPERATIONAL FEATURES

The operational features of the SAMSON machine must include the facility for data manipulation and computation as well as the capability of controlling the sequencing of instructions.

SAMSON will include all of the conventional facilities for (local) instruction execution and it will provide global instruction features. In addition, SAMSON will possess a high degree of concurrency in that simultaneous operation of several processors jointly on one task or separately on different tasks will be possible.

The basic processor (central processor) elements of the SAMSON computer are 16 bit general purpose microprogrammed processors having multiple general purpose registers operating on 16 bit data words. Their instruction repertoire will include special multiprocessor instructions (global instructions), such as FORK, JOIN, etc.

The major characteristics of the SAMSON processor elements are:

- o Type
  - --general purpose, digital, binary
  - --full parallel organization
  - --microprogrammable
- o Arithmetic
  - --binary, fixed point
  - --16 bit data word
  - --negative numbers in 2's complement form
  - --add time: 1 μ sec (register-to register)* 2 μ sec (memory to register)
  - --multiply time: 21 μ sec (Average, register to register)
  - --divide time: 27.5 μ sec (Average, register to register)
- o Memory
  - --DRO core
  - --1 μ sec cycle time
  - --each memory is expandable to 32K words in 4K or 8K increments
- o Addressing Modes
  - --direct addressing to 512 words of memory
  - --indexed addressing (execution time unaffected)
  - --multilevel indirect addressing
- o Input/Output
  - --program interrupts expandable
  - --power fail and power-restore interrupts can be provided
  - --I/O communicates with CPU and memory by means of separate common data and address bus; I/O devices can be added as required
  - --standard peripherals - teletype and/or CRT display and paper tape reader

---

* Execution time based on a memory cycle time of 1 μ sec.

- Logic

  --implemented with LSI and MSI arrays of T^2L

4.2 MEMORY ORGANIZATION

Multimemory accessability is required at the processor level in a multiprocessor since, in general, it is impossible to operate solely out of one memory unless swapping procedures, anticipative procedures, or mapping techniques are utilized. Although these features are desirable in a uniprocessor memory hierarchy to obtain an economical configuration, their usefulness as the primary memory for a multiprocessor is questionable. Nevertheless, these techniques could be utilized to supplement the primary memory. SAMSON'S present memory organization will not include these features; however, provisions can be provided for their addition at a later date, should the research in this area so indicate.

A modular memory organization consisting of several small singleport memories, rather than one large common multiaccess or multiport memory, will be utilized in the present SAMSON architecture. It should be noted that this was one of Bauer's criteria for a multiprocessor and although exception was taken previously, the exception was based upon this criteria being a necessary condition (and not on it being a desirable condition). Each memory module will be accessible by all processing elements. This scheme provides for parallel transfer of individual words or entire memory modules using present day switching technology; thus providing the capability of transferring large quantities of information at full memory rates. This multi-memory organization allows programs and data to be shared by several processing elements on a high speed basis while providing a means of insuring privacy of data. When independently operating, the processors which comprise the SAMSON computer system will be able to communicate via memory to memory transfers under program control. Memory transfers will be on a

request basis, thus avoiding catastrophic memory destruction. Careful consideration must be given with regard to providing a means of insuring privacy and protection of data without unduly limiting the capability of sharing.

It is intended that at any given time each processor can have access to several memories, but only those assigned to it. Furthermore, any processor (if assigned) can have accessibility to all memory modules (memory stacking) which would be in excess of its inherent addressing capability; thus providing an expanded high speed memory capacity.

In a multiprocessor, it may well be desirable to be able to write identical, duplicate copies of data and/or instructions into several memories without any additional expenditure of time. This feature is presently being considered for the SAMSON machine. Research in this area is required.

The above memory organization enables the total computing power of SAMSON to be applied to those tasks requiring the full computing power of the machine.

4.3 COMMUNICATIONS STRUCTURE

The communication requirements of a multiprocessor can be divided into two classes of communications: intermachine communications and input/output (I/O) communications. In either event the time investment made in establishing the connection should be minimal, or alternatively the connection should be retained for a relatively long period of time in order to justify the time expenditure in establishing the communication link. Furthermore, the data rate should be sufficiently high to satisfy system operating requirements.

Three basic communications techniques exist where by the major system components and peripherals can be interconnected, namely,

space-division, time-division and frequency division.

The space-division techniques, including completely dedicated communications as well as multilevel switching techniques (crossbar switching), are conceptually the simplest. However, an excessive amount of hardware is required and/or operational difficulties such as path building, conflicting communications requirements, and extended periods of time required to establish connections may occur.

Time-division or time multiplexing techniques are a viable alternative approach. Here queuing and priority schemes are utilized to insure that one and only one device pair can have access to the data bus during any one time slot. With this technique time slots can be either preassigned or a dynamic allocation procedure utilized. The drawback of this technique is the possibility of one device pair (with proper priority) completely tying up the communication link.

A frequency-division (frequency-multiplexing) scheme appears to offer the greatest flexibility and potential. The major drawbacks to this approach are: carrier frequencies in the range of 100's of megahertz are required (to meet the processors data rate[17] requirements) and furthermore, the necessary hardware for modulators and demodulators (which are not standard computer components) needed for transmitting large amounts of information would be excessive.

4.3.1 INTERMACHINE COMMUNICATIONS

SAMSON'S intermachine communications, as presently viewed, will be a combination of both time multiplexing and spacial communication techniques. It is felt that this approach is the most amenable to present state of the art computer techniques.

Memory communications will use both of these techniques, in that each memory will be assigned a dedicated communication path to its associated processor as well as a connection to the common intermachine memory communication network.

Intermachine register to register communications is also being considered for SAMSON. However, no decision has been made whether such communications will be included in the present SAMSON architecture.

4.3.2 INPUT/OUTPUT FACILITIES

The purpose of an input-output (I/O) system is to efficiently, reliably, and in an orderly fashion transfer the maximum amount of information in a minimal amount of time and with minimal interference to other subsystems. The I/O philosophy should provide a general purpose interface with all the necessary control and communications paths to allow data transfers between the various peripheral devices and the main processing system.

I/O communication requirements differ from intermachine communication requirements since the peripheral devices have widely varying characteristics, because of the multiplicity of the peripheral devices, and the unrestricted physical length over which the communication may take place. An additional fundamental difference in the type of communications results from the assignment of peripherals. Whereas memories are assigned to computers under program control (either by request or otherwise) I/O peripherals belong to a common pool of devices which, although available to all, are only "loaned" to a particular processor. The peripheral remains on loan to the assigned processor until the request is fulfilled or in some instances until an interrupt occurs from a higher priority source.

Reasoning as above, SAMSON's I/O communication philosophy will be based upon a time multiplexed system. (Fixed time slots will

not be utilized here due to the pseudo random nature of the required interconnections). Furthermore, the I/O system will be a request-response system, thus allowing maximum transmission rates over various physically different length paths.

In addition, the SAMSON I/O facility includes both internally and externally controlled data transfers.

5.0 RESEARCH STATUS

It is the objective of this work to physically implement the SAMSON machine, as described herein, with appropriate modifications reflecting the results of the ensuing research.

The SAMSON machine will be a useful tool for performing precise mathematical operations across a broad spectrum of applications while providing a multiprocessor both suitable for, and with the necessary capabilities to enable research in such areas as:

- Multiprocessor architecture (few multiprocessors have been constructed; thus each represents a major research milestone)[18]
- The processor-memory interconnection
- The memory contention problems
- The communication and control of a multiprocessor memory
- The communication structure of multiprocessors
- The instruction repertoire mandated by a multiprocessor
- Applied and natural parallelism
- Global and local control
- The hardware operating system
- The effect of a multiple accumulator and/or multiple index register structure

- o Interrupt handling, DMA operation, cycle stealing and priority assignments
- o The effect of a hierarchical memory organization on a multiprocessor
- o The effects of overlapping, prefetching instructions and operands, multiple instruction decoding and conditional branching
- o Data flow efficiency and input/output flexibility
- o Diagnostic procedures, reconfiguration and degraded operation
- o The optimum sequencing problem
- o Software application versatility, decomposition of computations and overall executive efficiency

It is beyond the scope of any one research effort to study all or even most of the above mentioned subject matter. Some of these topics will be covered as part of this research in the development of the SAMSON machine; i.e., multiprocessor architecture, processor-memory interconnection, memory-contention, memory communication and control, communications, instruction repertoire, parallelism, and global and local control. It is felt that the subjects covered by this work, the additional subject matters listed above, as well as any additional topics uncovered as a result of this and other research efforts will be continued in future research activity. It is felt that SAMSON will be an invaluable aid to future research efforts in the area of multiprocessors since research in this area has a strong experimental and empirical component requiring research, design and construction of many systems.[19]

The SAMSON machine is in the initial research stage of development and will consist of four (4) uniprocessors capable of concurrent processing (as a strongly-connected multiprocessor).

The first SAMSON processor element is presently being debugged. Similarly, the SAMSON test set is also in the debugging stage. One piece of support test equipment has been completed and is being used to assist debugging of the SAMSON processor element.

References

1. Saltzer, J. H.; Traffic Control in a Multiplexed Computer System, MAC-TR-30, Thesis, MIT, Cambridge, Mass., July 1966.

2. Curtin, W. A.; Multiple Computer Systems Advances in Computers, 1963, 4, 245-303.

3. Aschenbrenner, R. A.; Intrinsic Multiprocessing, Spring Joint Computer Conf., 1967, 81-86.

4. Koczela, L. J.; The Distributed Processor Organization, Advances in Computers, 1968, 9, 286-353.

5. Slotnick, D. L.; The SOLOMON Computer, Fall Joint Computer Conference, 1962, 97-107.

6. Murtha, J. C.; Highly Parallel Information Processing Systems, Advances in Computers, 1966, 7, 1-226.

7. Burroughs Corp.; ILLIAC IV Systems Characteristics and Programming Manual, NASA CR-2159, 1973, Feb.

8. Bouknight, W. J.; The ILLIAC IV System, Proceedings of the IEEE, Vol. 60, 4, April 1972, 369-388.

9. Comfort, W. T.; A Modified HOLLAND Machine, Fall Joint Computer Conference, 1963, 481-488.

10. Leiner, A. L.; PILOT - A New Multiple Computer System, Journal, ACM, 6, 1959, 315-35.

11. Eckert, J. P.; Design of UNIVAC - LARC System I., Proc. of the Eastern Joint Computer Conference, 1959, Dec. 1959.

12. Avizienis, A.; et al., The STAR Computer, Jet Propulsion Laboratory, California Institute of Technology.

13. Bell, G. C. and Wulf, W. A.; C.mmp - A Multi-mini-processor, Fall Joint Computer Conference, 1972, 765-777.

14. Bell, G. C. and Freeman, P.; C.ai - A Computer Architecture for AI Research, Fall Joint Computer Conference, 1972, 779-790.

15. Koczela; op. cit., Distributed Processor; 289.

16. Bauer, W. F.; Why Multicomputers? Datamation 6, Sept. 1962, 51-55.

17. Curtin; op. cit., Multiple Computer; 256.

18. Bell; op. cit., C.mmp; 766.

19. Ibid., 765.

III. Digital Computer Simulation and Automatic Fault Detection for Space Shuttle Electric Power Distribution System

by F. E. Thau and C. B. Park

One aspect of our analysis of the Space Shuttle power distribution system has been concerned with the development of a flexible digital computer program, comprising a fixed main routine and a number of user-supplied subroutines, which simulates the dynamic performance of the distribution system given a specification of a sequence of mission modes and the load distribution in each of these modes. The program, previously delivered to NASA, has been tested successfully on a network of about the size of one third of the proposed power distribution system. This program can be applied for the evaluation of competing proposed modifications in the electric distribution system structure or in the ratings of elements of the distribution system. A description of the program is contained in section III. 1 .

A second aspect of our analysis has been the development of techniques for automatic fault detection for application in future space shuttle designs. Two approaches have been taken: a reduced search procedure was developed to use a limited number of physical measurements together with a listing of nominal conditions to automatically specify which loads or cables are faulted. This approach is outlined in section III. 2. a. The second approach uses a dynamic model of the distribution system in the design of a sampled - data observer or filter whose inputs are the available measurements of voltage and current and whose outputs are the state of the power distribution system. This approach is outlined in section III. 2. b.

A number of problems remain to be studied in order to apply automatic fault detection for future space shuttle power distribution systems. First, the automatic fault detection schemes outlined above should be tested on networks of full size to determine the computational requirements that each technique would impose on the on-board computers. Second, automatic fault correction or reconfiguration techniques should be developed to fully implement an automatic fault detection and correction sub-system.

III. 1

SIMULATION PROGRAM DESCRIPTION FOR SPACE-SHUTTLE ELECTRIC POWER DISTRIBUTION SYSTEM

SEPDIS PROGRAM

A configuration of the electric power distribution system of the space shuttle is assumed. A sequence of distribution modes of the power system associated with relay states is also given in real time. The operation of the power system follows the schedule specified by the mode sequence. Section I gives the analytic basis for the simulation program. Section II describes the program in detail and Section III contains an application example.

A. Simulation Program Background

Assumptions

Assume that the electric power distribution of the shuttle spaceship (EPDS) is represented by a schematic diagram in Fig. 1.

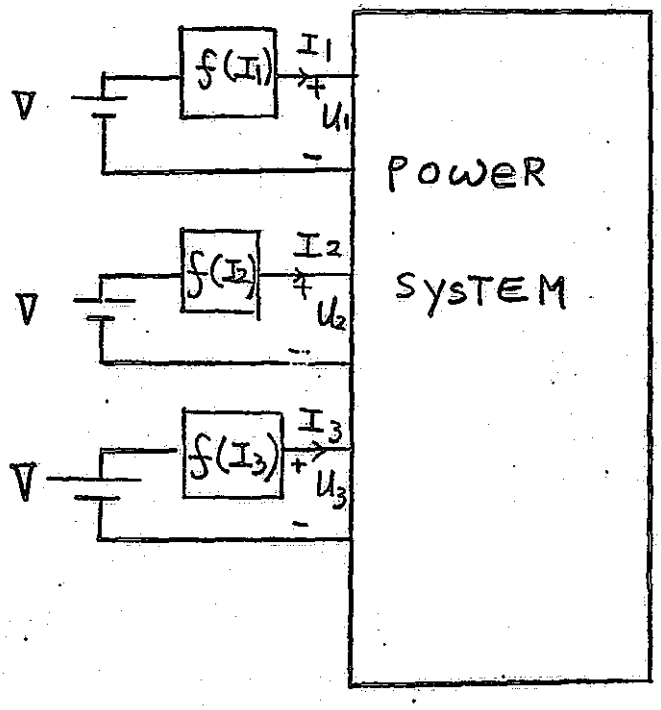


Fig-1

In Fig. 1 the non-linear function of I, f(I) depends on the nature of the V-I characteristics of the given Fuel Cell (D.C. source). Assume that all the elements of the power system can be approximated as linear, lumped, and are given as circuit elements for a given mode, i.e.,

1) Dynamic of each load is described by a linear differential eq or by a circuit description
2) For simplicity each relay is simulated by a Resistor R

$R = 0.01\ \Omega$ when closed

$R = 10^8\ \Omega$ when open

with a delay time given.

ii) Determination of System Differential Equation

One can describe the dynamic of the given power system under the assumptions made by

$$\begin{bmatrix} O \\ \hline E\dot{\underline{q}} \end{bmatrix} = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \begin{bmatrix} \underline{Z} \\ \hline \underline{q} \end{bmatrix} + \begin{bmatrix} G_1 \\ \hline G_2 \end{bmatrix} \underline{u} \tag{1}$$

$$\underline{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} V - \begin{bmatrix} f(I_2) \\ f(I_2) \\ f(I_3) \end{bmatrix}, \quad \underline{I} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

$$\underline{I} = C \begin{bmatrix} \underline{Z} \\ \hline \underline{q} \end{bmatrix}$$

Where

$$\underline{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \qquad u_1, u_2, u_3, \quad \text{input variables}$$

$$\underline{q} = \begin{bmatrix} q_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ q_{NSV} \end{bmatrix} \qquad \text{state variables of the system}$$

$$\underline{I} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$ output variables of the system

$$\underline{Z} = \begin{bmatrix} Z_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Z_{NAUX} \end{bmatrix}$$ auxiliary variables which connect some states, input and output variables

A determination of the submatrices A_{11}, A_{12}, A_{21}, A_{22}, E, G_1, G_2 depends on how to choose the auxiliary variables and state variables as physical elements.

From the standpoint of computer analysis of the complex circuit Ref. 1 the state variables are taken to be all the inductive currents and all the capacitive voltages. Moreover, let the auxiliary variables be all the node voltages and all the branch currents except for those node voltages and branch currents corresponding to state variables, i.e.,

$$\underline{Z} = \begin{bmatrix} \underline{V}_N \\ \hline \underline{I}b \end{bmatrix}$$

Where

$$\underline{V}_N = \begin{bmatrix} V_{N1} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ V_{NNNode} \end{bmatrix} \qquad \underline{I}b = \begin{bmatrix} Ib_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Ib_{Ibr} \end{bmatrix}$$

V_i indicates the i-th node voltage (i = 1,2,..., NNode)

Ib_i indicates the i-th branch current (i = 1,2,..., Ibr)

Rewriting (1) in terms of physical variables $\underline{V}_N$, $\underline{I}b$, $\underline{q}$ gives

$$\begin{bmatrix} O \\ \hline O \\ \hline \dot{E}\underline{q} \end{bmatrix} = \left[\begin{array}{c|c|c} F_{11} & F_{12} & F_{13} \\ \hline F_{21} & F_{22} & O \\ \hline F_{31} & O & A_{22} \end{array}\right] \begin{bmatrix} \underline{V}_N \\ \hline \underline{I}b \\ \hline \underline{q} \end{bmatrix} + \begin{bmatrix} B_1 \\ \hline B_2 \\ \hline O \end{bmatrix} \underline{u}$$

and

$$\underline{I} = \left[\begin{array}{c|c|c} O & C_2 & C_3 \end{array}\right] \left[\begin{array}{c} \underline{V_N} \\ \hline \underline{I_b} \\ \hline \underline{q} \end{array}\right] \qquad (2)$$

Where the submatrices F_{11}, F_{12}, F_{13}, B_1 are determined by applying KCL law at each node. Therefore, the NNode x NNode matrix F_{11}, NNode x3 matrix B1 are null and all the elements of the matrices F_{12}, F_{13} are 1 if entering into the node, -1 if flowing out of the node, otherwise zero. And the submatrices F_{21}, F_{22}, F_{31}, A_{22}, B_2, and E are determined so that all the branch elements, resistors, inductances, capacitances and independent sources are defined.

iii) Formulation of Normal Form of State Equation

To formulate the normal form of the state equation from the system differential equations (2), one has to eliminate the auxiliary variables in eq(2). For the elimination of the auxiliary variables a gauss-elimination method has been used, i.e., from (2) one can have

$$\left[\begin{array}{c} \underline{V_N} \\ \hline \underline{I_b} \end{array}\right] = -\left[\begin{array}{c|c} O & F_{12} \\ \hline F_{21} & F_{22} \end{array}\right]^{-1} \left[\left[\begin{array}{c} F_{13} \\ \hline O \end{array}\right] \underline{q} + \left[\begin{array}{c} O \\ \hline B_2 \end{array}\right] \underline{u}\right] \qquad (3)$$

Where the indicated inversion of the matrix is assumed to exist.

From (2), (3) one can obtain state equation

$$E\dot{\underline{q}} = -\left[\begin{array}{c|c} F_{31} & O \end{array}\right] \left[\begin{array}{c|c} O & F_{12} \\ \hline F_{21} & F_{22} \end{array}\right]^{-1} \left[\left[\begin{array}{c} F_{13} \\ \hline O \end{array}\right] \underline{q} + \left[\begin{array}{c} O \\ \hline B_2 \end{array}\right] \underline{u}\right] + A_{22}\underline{q}$$

Therefore, the normal form of state equation is

$$\dot{\underline{q}} = E^{-1}\left[A_{22} - \left[\begin{array}{c|c} F_{31} & O \end{array}\right] \left[\begin{array}{c|c} O & F_{12} \\ \hline F_{21} & F_{22} \end{array}\right]^{-1} \left[\begin{array}{c} F_{13} \\ \hline O \end{array}\right]\right] \underline{q} + E^{-1}\left[\left[\begin{array}{c|c} F_{31} & O \end{array}\right] \left[\begin{array}{c|c} O & F_{12} \\ \hline F_{21} & F_{22} \end{array}\right]^{-1} \left[\begin{array}{c} O \\ \hline B_2 \end{array}\right]\right] \underline{u} \qquad (4)$$

and

$$\underline{I} = C_3\underline{q} \qquad -\left[O \mid C_2\right]\left[\begin{array}{c|c} F_{11} & F_{12} \\ \hline F_{21} & F_{22} \end{array}\right]\left[\left[\begin{array}{c} F_{13} \\ \hline O \end{array}\right]\underline{q} + \left[\begin{array}{c} O \\ \hline B_2 \end{array}\right]\right]\underline{u}$$

iv) Solution of Non-Linear Differential Equation

The dynamic of the EPDS can be described by a set of non-linear differential eqs,

$$\dot{\underline{q}} = A\underline{q} + B\underline{u}$$

$$\underline{u} = \begin{bmatrix} V \\ V \\ V \end{bmatrix} - \begin{bmatrix} f(I_1) \\ f(I_2) \\ f(I_3) \end{bmatrix} \qquad (5)$$

where all the elements of A, B are given.

A solution of the above non-linear differential equation becomes very much unrealistic when some of eigenvalues of A are very large. When some relays are open that may give this case since the relay is simulated by a resistor whose value is 0.01 if closed, $10^8\ \Omega$ if it is open. To integrate the system with some eigenvalues large, one has to take a very small integration step size which may take prohibitive large integration time. One may overcome such a difficulty by finding steady state solution of some state variables corresponding to very large eigenvalues as a problem of singular perturbation.

B Computer Program of Simulation of EPDS

Computer program of simulation of EPDS consists of four main steps, i.e.,

(1) Read data according to the specification of a circuit's branches and their connection and formulate a set of system differential equations. Read data for mode sequence associated relay states.

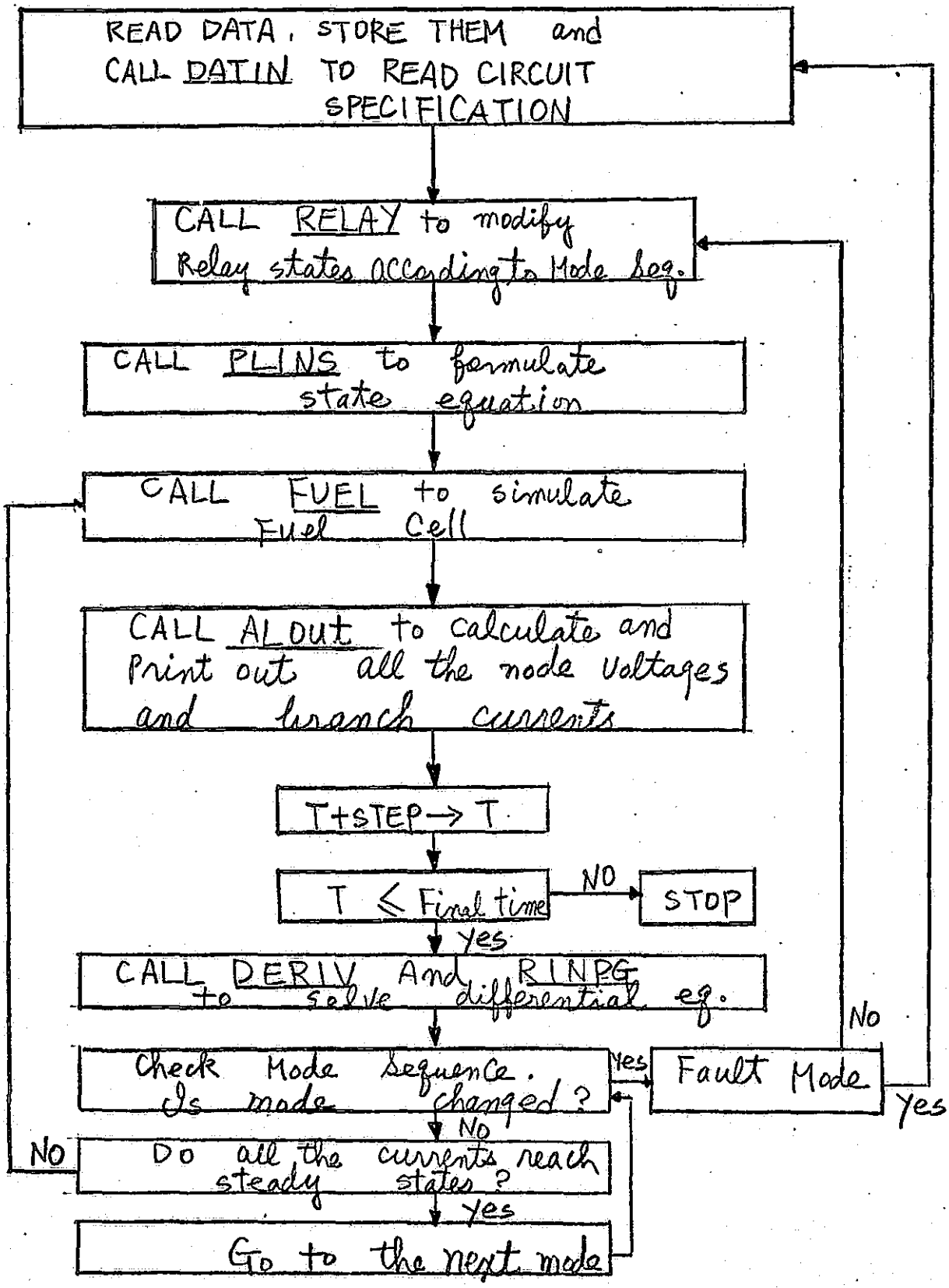
(2) Reduce the system differential equations to the state normal form.

(3) Solve non-linear differential eq.

(4) Check all the relay states according to a mode sequence and modify system differential equation.

To see these steps refer to flow chart.

FLOW CHART


READ DATA, STORE THEM and CALL DATIN TO READ CIRCUIT SPECIFICATION
CALL RELAY to modify Relay states according to Mode seq.
CALL PLINS to formulate state equation
CALL FUEL to simulate Fuel Cell
CALL ALOut to calculate and Print out all the node voltages and branch currents
T+STEP → T
T ≤ Final time
NO
STOP
yes
CALL DERIV And RINPG to solve differential eq.
Check Mode sequence. Is mode changed ?
Yes
Fault Mode
No
Yes
No
Do all the currents reach steady states ?
No
yes
Go to the next mode

As shown in the accompanying flow chart, the SEPDIS program consists of six major subroutines, DATIN, RELAY, PLINS, FUEL, ALOUT, RINPG.

SUBroutine DATIN (KECT, VALUI, NR, NL, NC, NV, NI, M, NCV, NCI, NEQN, NVAR, NOU, NN, NZ, NAUX, NK, K11, NO)

DATIN reads all the circuit elements, resistors, inductors, voltage sources associated with two nodes and store them in KECT and in VALUI (corresponding numerical value of element), i.e.,

Outputs: KECT, VALUI

NR:	number of	resistive elements
NL:	" "	inductive "
NC:	" "	capacitive "
NV:	" "	independent voltage sources
NI:	" "	independent current sources
M:	" "	mutual inductances
NCV:	" "	capacitance voltages
NCI:	" "	capacitance currents
NAUX:	" "	auxiliary variables
NVAR:	" "	the order of system
NEQN:	" "	NAUX + NVAR
NOU:	" "	number of outputs
NINP:	" "	number of inputs

NN, NZ, NK, K11, NO are working variables

SUBroutine Relay (NRLAY, R, IRY, NR, T, TSEQ, NMODE, IJK, KECT, VALUI, VN)

Inputs: NRLAY, R, IRY, NR, T, TSEQ, NMODE (READ), IJK (Main), VN (see note)

Outputs: KECT, VALUI

Note

Note that a diode can be modeled by a relay associated with two adjacent node voltages. However, in this subroutine simulation of diode has been neglected at the present program.

SUBroutine PLINS (A, B, KECT, VALUI, NR, NL, NC, NV, NI, M, NCI, F21, NK, K11, NCV, NEQN, NINp, NVAR, NOV, NN, NSVR, NO, NZ, NAUX, GI, NRPT)

Inputs: KECT (DATIN), VALUI (DATIN), NR, NL, NC, NV, NI, M, NCI, NK, K11, NCV, NEQN, NINP, NVAR, NOV, NN, NO, NZ, NAUX, (all from DATIN), NPRIN (READ)

Outputs: A, B, F21, G1, NSVR

A: System matrix

B: Input " (see eq.5)

$$F21 = \left[A_{11} \,\vdots\, A_{12} \right], \; G_1$$

(See eq.1)

NSVR: number of state variables

SUBroutine Fuel (YIN, X, NDeg, NVAR)

Input: X (from read initially and from integration subroutine RINPC)
NDeg (read), NVAR (DATIN)

Output: YIN

$$YIN = \begin{bmatrix} 32 \\ 32 \\ 32 \end{bmatrix} - \begin{bmatrix} f(I_1) \\ f(I_2) \\ f(I_3) \end{bmatrix}$$ (see fig. 1)

where the non-linearality must be programmed in the subroutine fuel.

SUBroutine ALout (F21, YIN, X, PN, VN, NSV, ITK, T, Numbr, NDeg, NLoad, NNode, NAUX, BRC, W, G1, NINP)

Input: F21 (PLINS), YIN (Fuel), X (RINPC), NSV (NSVR), IJK, T, Numbr (Main), NDeg, NLoad, NNode, NAUX (Read), G1 (PLINS), NINP (DATIN)

Output: VN, PN

VN (°): all the node voltages, branch currents

PN (°): instantaneous powers at all the loads. At the present time, PN (°) has not been programmed

SUBroutine RINPC (X, DX, T, Step, ISW, Rmax, Emax, NDeg, NLoad, IJK, YIN, A, B, NSV, Numbr)

Subroutine RINPC is followed by subroutine DERIV (DX, X, YIN, A, B, NDeg, NSV); In the Subroutine Deriv Input-Output specification is as follows:

Input: X (RINPC and initially read). YIN (Fuel), A, B (PLINS), NDeg (Read), NSV (=NVAR)

Output: DX describing the given dynamic structure of the system, i.e.,

$$DX = A X + B U$$

The arguments of the subroutine RINPC is as follows:

Input: DX (Deriv), Step (Read), Rmax (=TSEQ (NNode) , NDeg (Read), NLoad (Read), YIN (Fuel), A, B (PLINS), ISW, IJK, Numbr (working variables)

Output: X

An integration routine in RINPC is based on the fourth order R-K method.

Restrictions on the Present Program

1. The dynamic system is assumed to be at rest initially
2. Only circuits having 60 system variables (= number of auxiliary variables + number of state variables). 10 state variables, 3 inputs, are allowed.

Input to Program

The present program accepts circuits having up to 60 system variables (= number of state variables + number of auxiliary variables), 10 state variables, 3 inputs, no restrictions on output variables.

(1) Data-input to the Program

Block 1

i) The first card provides the number of relays, the number of modes, the number of loads, the number of nodes and should be punched in the format

I3, I3, I3, I3

ii) The second card gives an initial integration step, initial time, and frequency of print out in the following format

E15.5, E15.5, I5

Block 2

i) The first set of cards in Block 2 provides the mode sequence in real time and is punched in the format

4E15.5

ii) The second set of cards gives the locations of relays in terms of node label, i.e., IRY (I, J), J=1, 2 will identify the I-th relay by two nodes. The format is

2I3

iii) The third set of data cards in Block 2 provides all the relay states, i.e., R(I,J) indicates the i-th relay state between the j-th mode and the (j+1)th mode. The third set of data cards should be punched in the format

4E15.5

Block 3

Block 3 defines all the branches of the circuit to be analyzed. For each branch a single card should be punched

EL (I,J) = $\pm$A

in the format

1A1, 1X, I3, 1X, I3, 2X, E16.5

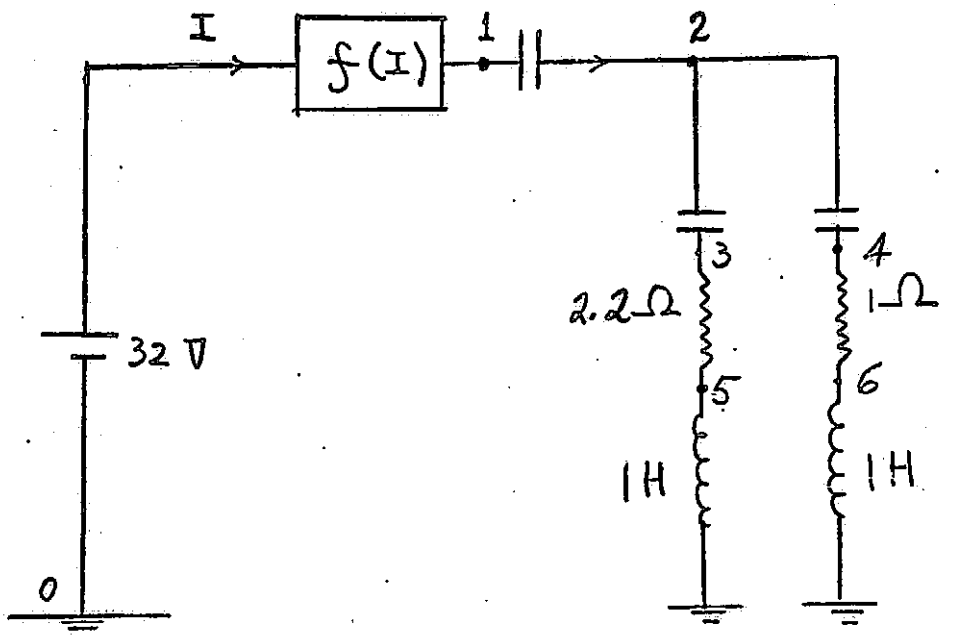
The integer i and j are the initial and terminal node labels of the branch, respectively, and EL defines the kind of branch

EL=R if branch element is resistor
EL=C " " " " capacitor
EL=L " " " " inductor
EL=V " " " " voltage source
EL=I " " " " current source

The number A, the numerical value of the branch element EL, the number A may be any real number if EL=V. The last four cards in Block 3 should punch in the first column for each card.

C Example 1. Example of Input Data Format

Consider a following two loads problem with one input as a test problem.



Assume that the system is rest initially. Analyze the transient response of the system from 0 to 1 second when the relay between the node 1 and 2 is open at time 0.5 sec.

$$f(I) = \text{Exp.}\ (-0.01*I)$$

Input Data

Block 1

i) The first card 3 3 2 6 (in the format 4I3)

ii) The second card 0.01 0. 10 (in the format 2E15.5,I5)

Block 2

i) The first card 0. 0.5 1. (4E15.5)

ii) The second card 1 2 (2I3)

The third card 2 3 (2I3)

iii) The fourth card 1 1 1 (4E15.5)

The fifth card 1 1 1 (4E15.5)

The sixth card 1 0 0 (4E15.5)

Block 3

R 1 2 1. E -2

R 2 3 1. E-2
R 2 4 1. E-2
R 3 5 2 1. E O

R 4 6 1. E O

L 6 O 1. E O

L 5 O 1. E O

Example 1 PRINT-OUT

State Equations for Each Mode

Mode 1

STATE EQUATION PRINT OUT
****

Q VECTOR OF STATE VARIABLES

INDUCTUVE BRANCH CURRENT 5 0

INDUCTUVE BRANCH CURRENT 6 0

X VECTOR OF INPUT VARIABLES

VOLTAGE SOURCE 0 1

E MATRIX = IDENTITY MATRIX

A MATRIX

I	J	A(I,J) (NON-ZERO)
1	1	-2.22000E 00
1	2	-1.00000E-02
2	1	-9.99998E-03
2	2	-1.02000E 00

B MATRIX

I	J	B(I,J) (NON-ZERO)
1	1	1.00000E 00
2	1	1.00000E 00

Mode 2

STATE EQUATION PRINT OUT
****

Q VECTOR OF STATE VARIABLES

INDUCTUVE BRANCH CURRENT 5 0

INDUCTUVE BRANCH CURRENT 6 0

X VECTOR OF INPUT VARIABLES

VOLTAGE SOURCE 0 1

E MATRIX = IDENTITY MATRIX

A MATRIX

I	J	A(I,J) (NON-ZERO)
1	1	-2.22000E 00
1	2	-9.99999E-03
2	1	-1.00000E-02
2	2	-1.00000E 08

B MATRIX

I	J	B(I,J) (NON-ZERO)
1	1	1.00000E 00
2	1	1.00000E 00

NODE VOLTAGES AT TIME = 0.0

1 0.31000E 02 2 0.31000E 02 3 0.31000E 02 4 0.31000E 02
5 0.31000E 02 6 0.31000E 02

ALL THE RESISTIVE CURRENTS AT TIME T= 0.0

1 2 0.11086E-04 2 3 0.36955E-05 2 4 0.0 3 5 0.0
4 6 0.0

NODE VOLTAGES AT TIME = 0.10000E 00

1 0.31056E 02 2 0.30998E 02 3 0.30971E 02 4 0.30969E 02
5 0.24852E 02 6 0.28021E 02

ALL THE RESISTIVE CURRENTS AT TIME T= 0.10000E 00

1 2 0.57293E 01 2 3 0.27810E 01 2 4 0.29483E 01 3 5 0.27810E 01
4 6 0.29483E 01

NODE VOLTAGES AT TIME = 0.20000E 00

1 0.31101E 02 2 0.30995E 02 3 0.30944E 02 4 0.30938E 02
5 0.19921E 02 6 0.25325E 02

ALL THE RESISTIVE CURRENTS AT TIME T= 0.20000E 00

1 2 0.10624E 02 2 3 0.50104E 01 2 4 0.56131E 01 3 5 0.50104E 01
4 6 0.56131E 01

NODE VOLTAGES AT TIME = 0.30000E 00

1 0.31138E 02 2 0.30990E 02 3 0.30922E 02 4 0.30909E 02
5 0.15967E 02 6 0.22888E 02

ALL THE RESISTIVE CURRENTS AT TIME T= 0.30000E 00

1 2 0.14819E 02 2 3 0.67974E 01 2 4 0.80215E 01 3 5 0.67974E 01
4 6 0.80215E 01

NODE VOLTAGES AT TIME = 0.40000E 00

1 0.31168E 02 2 0.30984E 02 3 0.30902E 02 4 0.30882E 02
5 0.12797E 02 6 0.20684E 02

ALL THE RESISTIVE CURRENTS AT TIME T= 0.40000E 00

1 2 0.18428E 02 2 3 0.82296E 01 2 4 0.10198E 02 3 5 0.82296E 01
4 6 0.10198E 02

NODE VOLTAGES AT TIME = 0.50000E 00

1 0.31194E 02 2 0.30978E 02 3 0.30885E 02 4 0.30857E 02
5 0.10254E 02 6 0.18692E 02

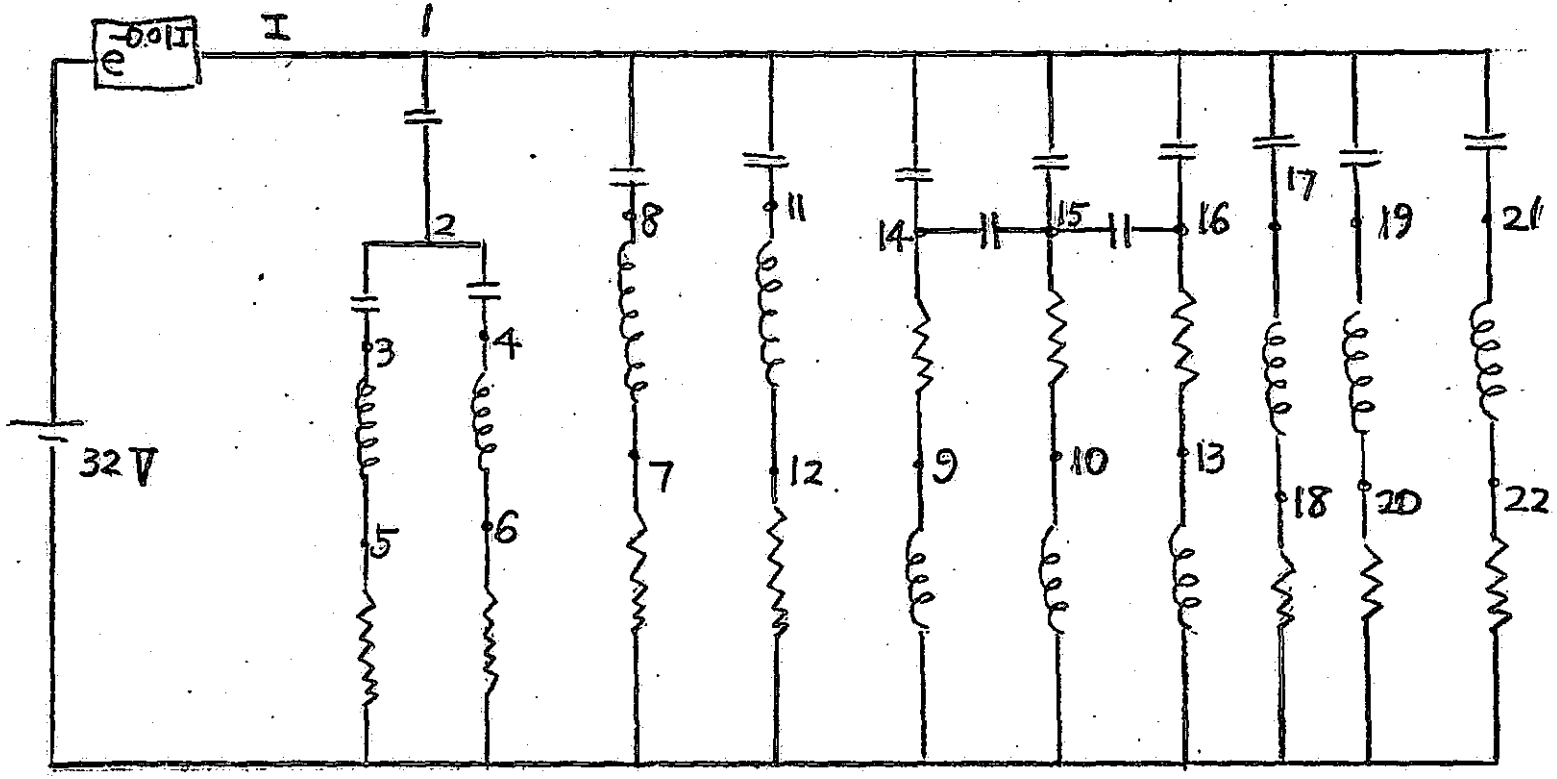
ALL THE RESISTIVE CURRENTS AT TIME T= 0.50000E 00

1 2 0.21542E 02 2 3 0.93774E 01 2 4 0.12165E 02 3 5 0.93774E 01
4 6 0.12165E 02

ALL THE BRANCH CURRENTS AND NODE VOLTAGES OF EXAMPLE-1

```
NODE VOLTAGES AT TIME =     0.60000E 00
    1    0.31098E 02     2    0.30995E 02     3    0.30892E 02     4    0.79346E-03
    5    0.82334E 01     6    0.79346E-03

ALL THE RESISTIVE CURRENTS AT TIME T=     0.60000E 00
    1    2    0.10299E 02     2    3    0.10299E 02     2    4    0.30994E-06     3    5    0.10299E 02
    4    6    0.30994E-06

NODE VOLTAGES AT TIME =     0.70000E 00
    1    0.31104E 02     2    0.30994E 02     3    0.30884E 02     4    0.64087E-03
    5    0.66003E 01     6    0.64087E-03

ALL THE RESISTIVE CURRENTS AT TIME T=     0.70000E 00
    1    2    0.11038E 02     2    3    0.11038E 02     2    4    0.30993E-06     3    5    0.11038E 02
    4    6    0.30993E-06

NODE VOLTAGES AT TIME =     0.80000E 00
    1    0.31110E 02     2    0.30993E 02     3    0.30877E 02     4    0.51880E-03
    5    0.52911E 01     6    0.51880E-03

ALL THE RESISTIVE CURRENTS AT TIME T=     0.80000E 00
    1    2    0.11630E 02     2    3    0.11630E 02     2    4    0.30993E-06     3    5    0.11630E 02
    4    6    0.30993E-06

NODE VOLTAGES AT TIME =     0.90000E 00
    1    0.31114E 02     2    0.30993E 02     3    0.30872E 02     4    0.42725E-03
    5    0.42415E 01     6    0.42725E-03

ALL THE RESISTIVE CURRENTS AT TIME T=     0.90000E 00
    1    2    0.12105E 02     2    3    0.12105E 02     2    4    0.30993E-06     3    5    0.12105E 02
    4    6    0.30993E-06

NODE VOLTAGES AT TIME =     0.10000E 01
    1    0.31117E 02     2    0.30993E 02     3    0.30868E 02     4    0.35095E-03
    5    0.34001E 01     6    0.35095E-03

ALL THE RESISTIVE CURRENTS AT TIME T=     0.10000E 01
    1    2    0.12485E 02     2    3    0.12485E 02     2    4    0.30992E-06     3    5    0.12485E 02
    4    6    0.30992E-06
```

Example-2 APPLICATION TO ⅓ DISTRIBUTION NETWORK



$R_{1,2} = 10^8$ [Ω]
$R_{2,3} = 10^{-2}$ "
$R_{2,4} = 10^{-2}$ "
$R_{5,0} = 10$ "
$R_{6,0} = 12$ "
$R_{1,8} = 10^{-2}$ "
$R_{7,0} = 10^2$ "
$R_{1,11} = 10^8$ "
$R_{12,0} = 10^8$ "

$R_{14,1} = 10^8$ [Ω]
$R_{14,15} = 10^{-2}$ "
$R_{15,16} = 10^{-2}$ "
$R_{6,15} = 10^{-2}$ "
$R_{6,16} = 10^{-2}$ "
$R_{14,9} = 22$ "
$R_{15,10} = 44$ "

$R_{16,13} = 40$ [Ω]
$R_{1,17} = 0.01$ "
$R_{1,19} = 10^{-2}$ "
$R_{1,21} = 10^{-2}$ "
$R_{18,0} = 30$ "
$R_{20,0} = 20$ "
$R_{22,0} = 1$ "

$L_{3,5} = 1$ [H]
$L_{4,6} = 1$ "
$L_{8,7} = 0.1$ "
$L_{11,12} = 1.2$ "
$L_{9,0} = 1$ "
$L_{10,0} = 1$ "
$L_{13,0} = 1$ "

$L_{17,18} = 1.1$ [H]
$L_{19,20} = 0.8$ "
$L_{21,22} = 0.1$ "

MODE DESCRIPTION

MODE 1 ($0 \leq t < 0.5$)

$R_{1,2}$, $R_{1,8}$, $R_{14,15}$, $R_{15,16}$, $R_{1,16}$, $R_{1,17}$, $R_{1,19}$, $R_{1,21}$ $R_{1,15}$ are closed

$R_{2,3}$; $R_{1,11}$, $R_{1,14}$ are open

MODE 2 ($0.5 \leq t < 1$)

$R_{1,2}$, $R_{1,23}$, $R_{2,4}$, $R_{1,14}$, $R_{14,15}$, $R_{15,16}$, $R_{1,17}$, $R_{15,16}$ $R_{1,15}$, $R_{1,16}$, $R_{1,19}$, $R_{1,21}$ are closed

$R_{1,8}$, $R_{1,11}$ are open

MODE 3 ($t = 1$)

STOP OPERATION

FIND ALL THE Branch CURRENTS, NODE VOLTAGES.

Example-2 State Equations For Each Mode

MODE 1

VOLTAGE SOURCE 0 1

E MATRIX = IDENTITY MATRIX

A MATRIX

I	J	A(I,J) (NON-ZERO)
1	1	-1.00200E 01
1	2	-1.00000E-02
2	1	-1.00000E-02
2	2	-1.20200E 01
2	3	-1.38778E-09
2	4	-1.38778E-09
2	5	4.65661E-10
2	6	9.31322E-10
2	7	-1.39698E-09
3	3	-9.09092E 07
4	4	-8.33334E 07
5	5	-2.20062E 01
5	6	-2.50002E-03
5	7	-1.25001E-03
6	5	-2.50001E-03
6	6	-4.40050E 01
6	7	-2.50002E-03
7	5	-1.56251E-03
7	6	-3.12502E-03
7	7	-5.00078E 01
8	8	-2.72818E 01
9	9	-2.50125E 01
10	10	-1.00100E 02

B MATRIX

I	J	B(I,J) (NON-ZERO)
1	1	1.00000E 00
2	1	9.99999E-01
3	1	9.09091E-01
4	1	8.33333E-01
5	1	1.00000E 00
6	1	1.00000E 00
7	1	1.25000E 00
8	1	9.09091E-01
9	1	1.25000E 00
10	1	1.00000E 01

MODE2

VOLTAGE SOURCE 0 1

E MATRIX = IDENTITY MATRIX

A MATRIX

I	J	A(I,J) (NON-ZERO)
1	1	-1.00000E 08
1	2	-9.99999E-03
2	1	-9.99999E-03
2	2	-1.00000E 08
3	3	-9.09182E 01
4	4	-8.33334E 07
5	5	-2.20166E 01
5	6	-6.66666E-03
5	7	-3.33334E-03
6	5	-5.66667E-03
6	6	-4.40017E 01
6	7	-3.33335E-03
7	5	-4.16667E-03
7	6	-4.16667E-03
7	7	-5.00083E 01
8	8	-2.72818E 01
9	9	-2.50125E 01
10	10	-1.00100E 02

B MATRIX

I	J	B(I,J) (NON-ZERO)
1	1	1.00000E 00
2	1	1.00000E 00
3	1	9.09091E-01
4	1	8.33333E-01
5	1	9.99999E-01
6	1	1.00000E 00
7	1	1.25000E 00
8	1	9.09091E-01
9	1	1.25000E 00
10	1	1.00000E 01

NODE VOLTAGES AT TIME = 0.0

1	0.31000E 02	2	0.31000E 02	3	0.31000E 02	4	0.31000E 02
5	0.0	6	0.0	7	0.0	8	0.31000E 02
9	0.31000E 02	10	0.31000E 02	11	0.31000E 02	12	0.0
13	0.31000E 02	14	0.31000E 02	15	0.31000E 02	16	0.31000E 02
17	0.31000E 02	18	0.0	19	0.31000E 02	20	0.0
21	0.31000E 02	22	0.0				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.0

1	2	0.0	2	3	0.0	2	4	0.0	5	0	0.0
6	0	0.0	1	8	0.0	7	0	0.0	1	11	0.0
12	0	0.0	1	14	-0.21244E-12	14	15	0.19640E-05	15	16	0.40708E-03
1	15	0.40315E-03	1	16	-0.40720E-03	14	9	0.0	15	10	0.0
16	13	0.0	1	17	0.0	1	19	0.0	1	21	0.0
18	0	0.0	20	0	0.0	22	0	0.0			

NODE VOLTAGES AT TIME = 0.10000E 00

1	0.31082E 02	2	0.31082E 02	3	0.99182E-03	4	0.99182E-03
5	0.31081E-05	6	0.37297E-05	7	0.31073E 02	8	0.31079E 02
9	0.34465E 01	10	0.38617E 00	11	0.10223E-02	12	0.27973E-04
13	0.21399E 00	14	0.31054E 02	15	0.31066E 02	16	0.31070E 02
17	0.31072E 02	18	0.29033E 02	19	0.31068E 02	20	0.28510E 02
21	0.31051E 02	22	0.31047E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.10000E 00

1	2	0.52162E-05	2	3	0.31081E-06	2	4	0.31081E-06	5	0	0.31081E-06
6	0	0.31081E-05	1	8	0.31073E 00	7	0	0.31073E 00	1	11	0.31081E-06
12	0	0.31081E-06	1	14	0.28113E-09	14	15	-0.12549E 01	15	16	-0.39317E 00
1	15	0.15590E 01	1	16	0.11646E 01	14	9	0.12549E 01	15	10	0.69727E 00
16	13	0.77140E 00	1	17	0.96776E 00	1	19	0.14255E 01	1	21	0.31047E 01
18	0	0.95776E 00	20	0	0.14255E 01	22	0	0.31047E 01			

NODE VOLTAGES AT TIME = 0.20000E 00

1	0.31085E 02	2	0.31085E 02	3	0.12207E-03	4	0.12207E-03
5	0.31085E-05	6	0.37302E-05	7	0.31082E 02	8	0.31082E 02
9	0.38223E 00	10	0.50354E-02	11	0.13733E-03	12	0.27976E-04
13	0.17853E-02	14	0.31054E 02	15	0.31068E 02	16	0.31073E 02
17	0.31075E 02	18	0.30941E 02	19	0.31069E 02	20	0.30859E 02
21	0.31054E 02	22	0.31054E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.20000E 00

1	2	0.52169E-06	2	3	0.31085E-06	2	4	0.31085E-06	5	0	0.31085E-06
6	0	0.31085E-06	1	8	0.31082E 00	7	0	0.31082E 00	1	11	0.31085E-06
12	0	0.31085E-06	1	14	0.30511E-09	14	15	-0.13942E 01	15	16	-0.44072E 00
1	15	0.15594E 01	1	16	0.12175E 01	14	9	0.13942E 01	15	10	0.70598E 00
16	13	0.77677E 00	1	17	0.10314E 01	1	19	0.15429E 01	1	21	0.31054E 01
18	0	0.10314E 01	20	0	0.15429E 01	22	0	0.31054E 01			

NODE VOLTAGES AT TIME = 0.30000E 00

1	0.31085E 02	2	0.31085E 02	3	0.30518E-04	4	0.30518E-04
5	0.31085E-06	6	0.37302E-06	7	0.31082E 02	8	0.31082E 02
9	0.42435E-01	10	0.91553E-04	11	0.61035E-04	12	0.27977E-04
13	0.61035E-04	14	0.31054E 02	15	0.31068E 02	16	0.31073E 02
17	0.31075E 02	18	0.31066E 02	19	0.31070E 02	20	0.31052E 02
21	0.31054E 02	22	0.31054E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.30000E 00

1	2	0.52170E-06	2	3	0.31085E-06	2	4	0.31085E-06	5	0	0.31085E-06
6	0	0.31085E-06	1	8	0.31082E 00	7	0	0.31082E 00	1	11	0.31085E-06
12	0	0.31085E-06	1	14	0.30769E-09	14	15	-0.14096E 01	15	16	-0.44590E 00
1	15	0.15698E 01	1	16	0.12227E 01	14	9	0.14096E 01	15	10	0.70610E 00
16	13	0.77682E 00	1	17	0.10355E 01	1	19	0.15526E 01	1	21	0.31054E 01
18	0	0.10355E 01	20	0	0.15526E 01	22	0	0.31054E 01			

NODE VOLTAGES AT TIME = 0.40000E 00

1	0.31085E 02	2	0.31085E 02	3	0.30518E-04	4	0.30518E-04
5	0.31085E-06	6	0.37302E-06	7	0.31082E 02	8	0.31082E 02
9	0.47507E-02	10	0.0	11	0.45776E-04	12	0.27977E-04
13	0.45776E-04	14	0.31054E 02	15	0.31068E 02	16	0.31073E 02
17	0.31075E 02	18	0.31074E 02	19	0.31070E 02	20	0.31068E 02
21	0.31054E 02	22	0.31054E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.40000E 00

1	2	0.52170E-06	2	3	0.31085E-06	2	4	0.31085E-06	5	0	0.31085E-06
6	0	0.31085E-06	1	8	0.31082E 00	7	0	0.31082E 00	1	11	0.31085E-06
12	0	0.31085E-06	1	14	0.30798E-09	14	15	-0.14113E 01	15	16	-0.44647E 00
1	15	0.15710E 01	1	16	0.12233E 01	14	9	0.14113E 01	15	10	0.70610E 00
16	13	0.77682E 00	1	17	0.10358E 01	1	19	0.15534E 01	1	21	0.31054E 01
18	0	0.10358E 01	20	0	0.15534E 01	22	0	0.31054E 01			

NODE VOLTAGES AT TIME = 0.50000E 00

1	0.31085E 02	2	0.31085E 02	3	0.30518E-04	4	0.30518E-04
5	0.31085E-06	6	0.37302E-05	7	0.31082E 02	8	0.31082E 02
9	0.59509E-03	10	0.0	11	0.45776E-04	12	0.27977E-04
13	0.45776E-04	14	0.31054E 02	15	0.31068E 02	16	0.31073E 02
17	0.31075E 02	18	0.31075E 02	19	0.31070E 02	20	0.31069E 02
21	0.31054E 02	22	0.31054E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.50000E 00

1	2	0.62170E-06	2	3	0.31085E-06	2	4	0.31085E-06	5	0	0.31085E-06
6	0	0.31085E-06	1	8	0.31082E 00	7	0	0.31082E 00	1	11	0.31085E-06
12	0	0.31085E-06	1	14	0.30801E-09	14	15	-0.14115E 01	15	16	-0.44653E 00
1	15	0.15711E 01	1	16	0.12234E 01	14	9	0.14115E 01	15	10	0.70610E 00
16	13	0.77632E 00	1	17	0.10358E 01	1	19	0.15535E 01	1	21	0.31054E 01
18	0	0.10358E 01	20	0	0.15535E 01	22	0	0.31054E 01			

NODE VOLTAGES AT TIME = 0.60000E 00

1	0.31112E 02	2	0.31079E 02	3	0.31062E 02	4	0.31063E 02
5	0.17108E 02	6	0.19172E 02	7	0.31110E-04	8	0.24719E-02
9	0.16502E-01	10	0.81940E-02	11	0.24719E-02	12	0.27999E-04
13	0.72784E-02	14	0.31101E 02	15	0.31103E 02	16	0.31104E 02
17	0.31102E 02	18	0.31090E 02	19	0.31097E 02	20	0.31084E 02
21	0.31081E 02	22	0.31077E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.60000E 00

1	2	0.33085E 01	2	3	0.17108E 01	2	4	0.15977E 01	5	0	0.17108E 01
6	0	0.15977E 01	1	8	0.31110E-06	7	0	0.31110E-06	1	11	0.31110E-06
12	0	0.31110E-06	1	14	0.11569E 01	14	15	-0.25597E 00	15	16	-0.61793E-01
1	15	0.90088E 00	1	16	0.83921E 00	14	9	0.14129E 01	15	10	0.70570E 00
16	13	0.77741E 00	1	17	0.10363E 01	1	19	0.15542E 01	1	21	0.31077E 01
18	0	0.10363E 01	20	0	0.15542E 01	22	0	0.31077E 01			

NODE VOLTAGES AT TIME = 0.70000E 00

1	0.31126E 02	2	0.31077E 02	3	0.31051E 02	4	0.31054E 02
5	0.25924E 02	6	0.27477E 02	7	0.31125E-04	8	0.86975E-03
9	0.70343E-02	10	0.28839E-02	11	0.85449E-03	12	0.28013E-04
13	0.25482E-02	14	0.31115E 02	15	0.31117E 02	16	0.31118E 02
17	0.31116E 02	18	0.31111E 02	19	0.31111E 02	20	0.31105E 02
21	0.31095E 02	22	0.31094E 02				

```
 ALL THE RESISTIVE CURRENTS AT TIME T=     0.70000E 00
    1    2     0.48821E 01    2    3     0.25924E 01    2    4     0.22898E 01    5    0     0.25924E 01
    6    0     0.22898E 01    1    8     0.31125E-06    7    0     0.31125E-06    1   11     0.31125E-06
   12    0     0.31125E-06    1   14     0.11578E 01   14   15    -0.25620E 00   15   16    -0.61851E-01
    1   15     0.90148E 00    1   16     0.83974E 00   14    9     0.14140E 01   15   10     0.70714E 00
   16   13     0.77788E 00    1   17     0.10370E 01    1   19     0.15552E 01    1   21     0.31094E 01
   18    0     0.10370E 01   20    0     0.15552E 01   22    0     0.31094E 01

 NODE VOLTAGES AT TIME =     0.80000E 00
    1     0.31131E 02    2     0.31077E 02    3     0.31047E 02    4     0.31052E 02

    5     0.29163E 02    6     0.29975E 02    7     0.31130E-04    8     0.33569E-03
    9     0.25635E-02   10     0.99182E-03   11     0.33569E-03   12     0.28017E-04
   13     0.90027E-03   14     0.31119E 02   15     0.31122E 02   16     0.31122E 02
   17     0.31120E 02   18     0.31119E 02   19     0.31115E 02   20     0.31113E 02
   21     0.31100E 02   22     0.31099E 02

 ALL THE RESISTIVE CURRENTS AT TIME T=     0.80000E 00
    1    2     0.54143E 01    2    3     0.29163E 01    2    4     0.24979E 01    5    0     0.29163E 01
    6    0     0.24979E 01    1    8     0.31130E-06    7    0     0.31130E-06    1   11     0.31130E-06
   12    0     0.31130E-06    1   14     0.11581E 01   14   15    -0.25630E 00   15   16    -0.61891E-01
    1   15     0.90179E 00    1   16     0.83993E 00   14    9     0.14144E 01   15   10     0.70729E 00
   16   13     0.77804E 00    1   17     0.10373E 01    1   19     0.15557E 01    1   21     0.31099E 01
   18    0     0.10373E 01   20    0     0.15557E 01   22    0     0.31099E 01

 NODE VOLTAGES AT TIME =     0.90000E 00
    1     0.31132E 02    2     0.31076E 02    3     0.31046E 02    4     0.31051E 02
    5     0.30354E 02    6     0.30727E 02    7     0.31132E-04    8     0.13733E-03
    9     0.93079E-03   10     0.35095E-03   11     0.13733E-03   12     0.28019E-04
   13     0.35095E-03   14     0.31121E 02   15     0.31123E 02   16     0.31124E 02
   17     0.31122E 02   18     0.31121E 02   19     0.31117E 02   20     0.31116E 02
   21     0.31101E 02   22     0.31101E 02
```

ALL THE RESISTIVE CURRENTS AT TIME T= 0.90000E 00

1	2	0.55959E 01	2	3	0.30354E 01	2	4	0.25606E 01	5	0	0.30354E 01
6	0	0.25606E 01	1	8	0.31132E-06	7	0	0.31132E-06	1	11	0.31132E-06
12	0	0.31132E-06	1	14	0.11582E 01	14	15	-0.25633E 00	15	16	-0.61902E-01
1	15	0.90177E 00	1	16	0.83999E 00	14	9	0.14145E 01	15	10	0.70734E 00
16	13	0.77809E 00	1	17	0.10374E 01	1	19	0.15558E 01	1	21	0.31101E 01
18	0	0.10374E 01	20	0	0.15558E 01	22	0	0.31101E 01			

NODE VOLTAGES AT TIME = 0.10000E 01

1	0.31133E 02	2	0.31076E 02	3	0.31046E 02	4	0.31051E 02
5	0.30791E 02	6	0.30953E 02	7	0.31133E-04	8	0.91553E-04
9	0.33569E-03	10	0.12207E-03	11	0.91553E-04	12	0.28020E-04
13	0.15259E-03	14	0.31121E 02	15	0.31124E 02	16	0.31124E 02
17	0.31123E 02	18	0.31122E 02	19	0.31117E 02	20	0.31117E 02
21	0.31102E 02	22	0.31102E 02				

ALL THE RESISTIVE CURRENTS AT TIME T= 0.10000E 01

1	2	0.56585E 01	2	3	0.30791E 01	2	4	0.25794E 01	5	0	0.30791E 01
6	0	0.25794E 01	1	8	0.31133E-06	7	0	0.31133E-06	1	11	0.31133E-06
12	0	0.31133E-06	1	14	0.11582E 01	14	15	-0.25635E 00	15	16	-0.61906E-01
1	15	0.90180E 00	1	16	0.84002E 00	14	9	0.14146E 01	15	10	0.70736E 00
16	13	0.77811E 00	1	17	0.10374E 01	1	19	0.15559E 01	1	21	0.31102E 01
18	0	0.10374E 01	20	0	0.15559E 01	22	0	0.31102E 01			

REFERENCE

1. Stagg, G. W. and El-Abiad, A. H., "Computer Mehods in Power System Analysis

McGraw Hill Book Company., 1968

III.2.a Combinatorial Method

2.a.1 Preliminary

Consider the Electric Power System described by equation (5) of section III.1. Assume for simplicity a single-input system, and $f(I_1) = 0$. Then

$$\dot{q} = A q + B_1 u_1$$
$$y = C q$$

where

$$B u = [B_1 \mid B_2 \mid B_3] \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

and y is the measurable output vector $[y_1, \ldots, y_r]$.

Assume that faults occur due to openings of some relays connected to loads. The problem is to detect faults of the system and to find their locations from the output measurements.

One can detect faults simply by comparing the actual outputs with all the combinations of possible fault-outputs. However, this method requires 2^m comparisons for a single measurement output. In the case that one could measure r-outputs, the number of comparison can be substantially reduced to

$$\sum_{i=1}^{r} 2^{n_i}$$

where n_i is the number of loads in the i-th group associated with outputs z_i, $z_i = y_{i+1} - y_i$ (see fig. 2)

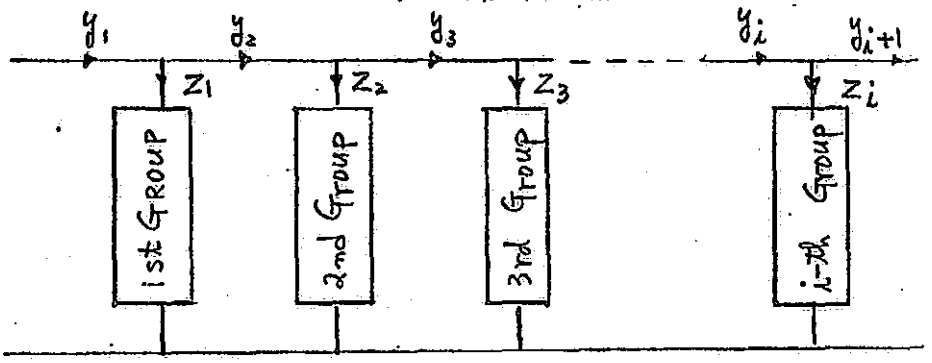


Fig. 2.1

a

2.a.2 Program Description For Combintorial Method

The program consists of main two steps.

(1) Partition the original system into NG Groups. For each Group find the ideal steady state load currents

(2) For each Group associated with the corresponding output, find all the possible faulted outputs in each Group if some faults are indicated in that Group and compare them with the corresponding actual output. (Refer to flow chart.)

As shown in the accompanying flow chart (see also the program listings), the FAULT Program consists of two major subroutines FAULII, COMBT

(i) Subroutine FAULII (N, A, NG, Yact, LL, AL)

Inputs:

N: the order of system (the number of loads)

NG: the number of partitions

LL(K): the number of loads in the K-th partitioned Group (K=1,2,...,NG)

AL(K): the k-th ideal load current. k=1,2,...,N.

Yact(I): the I-th output load current flowing in the I-th partition.

Outputs:

A(J)=AL(II+J), the J-th load in the II-th partitioned Group.

YIDEL(I): Ideal load current flowing in the I-th partitioned Group.

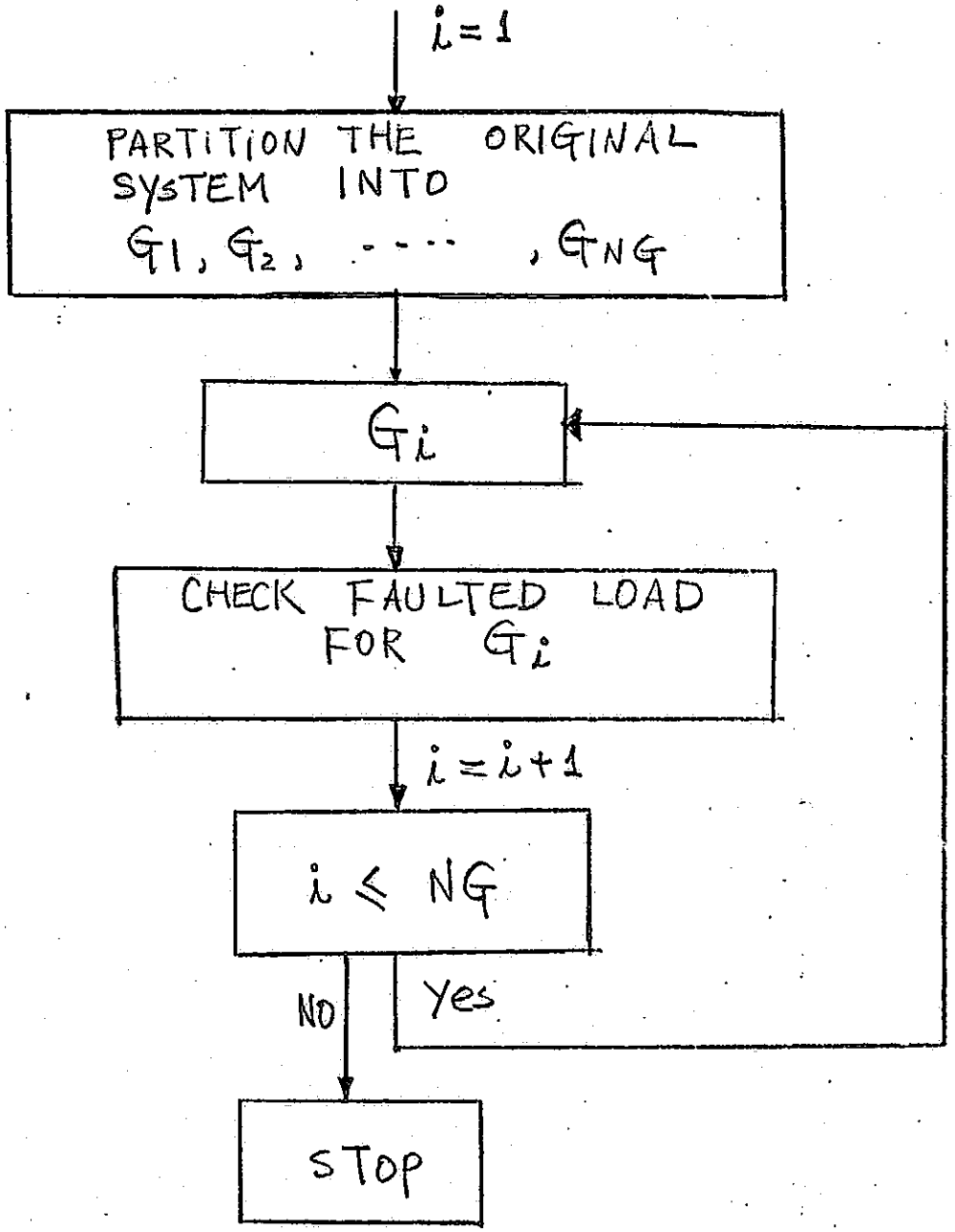
(ii) COMBT (N, A, Yact, INDEX)

Input:

N: the order of system

A(FAULII), Yact (FAULII)

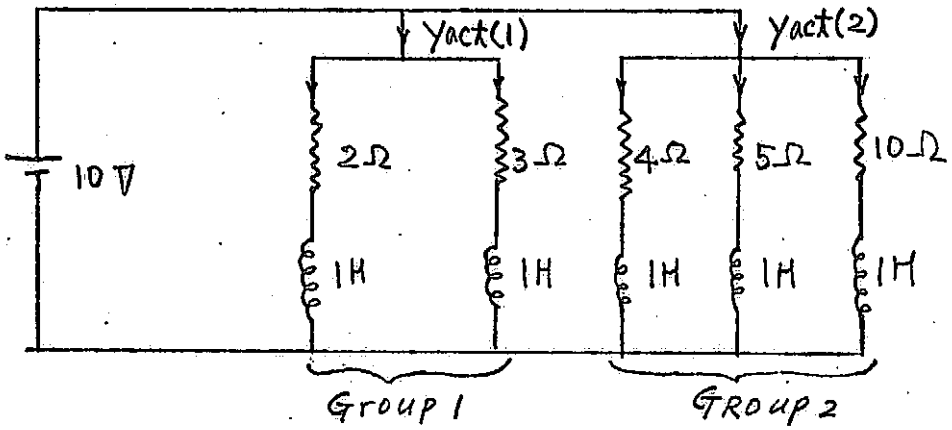
FLOW CHART


i = 1
PARTITION THE ORIGINAL SYSTEM INTO
G1, G2, ... , GNG
Gi
CHECK FAULTED LOAD FOR Gi
i = i + 1
i ≤ NG
NO
Yes
STOP

Output:

Index indicates the index-th faulted partition. This program can detect faults of order less than five.

Example:



Assume that the above system was operating in the steady states. Assume that suddenly the actual output measurements Yact(1), Yact(2) are different from the ideal ones and that they are

(i) Yact(1) = 5
Yact(2) = 0

(ii) Yact(1) = 5
Yact(2) = 3

find fault locations for each case.

Input Data

Block 1

(i) Read N 5 (in the format I5)

(ii) Read AL(I), I=1, ..., 5 (4E15.5)
(5., 3.33, 2.5, 2., 1.)

Block 2

(i) Read NG (I5), NG=2

(ii) Read LL(I), I=1,2, (2,3)

Printouts for each case

(i) Yact = 5, Yact(2) = 0, (ii) Yact(1) = 5
Yact (2) = 3

are as follows:

(i)

```
 A DIAGNOSIS OF  THE FAULT LOCATIONS
 ***********************************

A SINGLE FAULT MAY OCCUR AT THE    2-TH  LOAD IN THE     1-TH GROUP

THE TRIPPLE FAULTS MAY OCCUR AT    1,    2,    3IN THE   2-TH GROUP

     THE END OF DIAGNOSIS
     ********************
```

(ii)

```
 A DIAGNOSIS OF  THE FAULT LOCATIONS
 ***********************************

A SINGLE FAULT MAY OCCUR AT THE    2-TH  LOAD IN THE     1-TH GROUP

A SINGLE FAULT MAY OCCUR AT THE    1-TH  LOAD IN THE     2-TH GROUP

     THE END OF DIAGNOSIS
     ********************
```

LISTINGS FOR COMBINATORIAL METHOD

```
      $JOB
 1              DIMENSION A(20),YACT(10),LL(10),AL(20)                  181)
 2             READ(5,11) N
 3              READ(5,15)(AL(I),I=1,N)
 4              READ(5,11) NG
 5              READ(5,11)(LL(I),I=1,NG)
 6               YACT(1)=5.
 7                YACT(2)=0.
 8              CALL FAULII(N,A,NG,YACT,LL,AL)
 9     11       FORMAT(I5)
10     15       FORMAT(4E15.5)
11                STOP
12               END

13             SUBROUTINE FAULII(N,A,NG,YACT,LL,AL)
14             DIMENSION AL(20),LL(10),YACT(10),YIDEL(10),A(20)
       C        NO NUMBER OF LOADS
       C       NGO NUMBER OF GROUPS
       C       LL(I)O NUMBER OF LOADS FOR THE I-TH GROUP
15              M2=LL(1)
16                M=1
17               N1=NG-1
18              K=1
19     28         YIDEL(K)=0.0
20             WRITE(6,500)
21              WRITE(6,550)
22     550      FORMAT(' ',5X,'*****************************************',///)
23     500     FORMAT('1',5X,' A DIAGNOSIS OF  THE FALLT LOCATIONS')
24              DO 21 I=M,M2
25     21         YIDEL(K)=YIDEL(K)+AL(I)
26             IF((K+1).GT.NG) GO TO 24
27             M=M+LL(K)
28              M2=M2+LL(K+1)
29              K=K+1
30              GO TO 28
31     24        CONTINUE
32              DO 30 I=1,NG
33             IF(ABS(YACT(I)-YIDEL(I)).LE.0.01) GO TO 31
34             II=0
35               INDEX=I
36              N=LL(INDEX)
37             IF(INDEX.LE.1) GO TO 46
38     40      II=II+LL(INDEX-1)
39               GO TO 46
40     46        DO 41 J=1,N
41     41       A(J)=AL(II+J)
42              YACTV=ABS(YACT(I)-YIDEL(I) )
43             CALL COMBT(N,A,YACTV   ,INDEX)
44     31       CONTINUE
45     30       CONTINUE
46             WRITE(6,501)
47               WRITE(6,551)
48     551      FORMAT(' ',10X,'*************************',////)
49     501      FORMAT(/////,10X,' THE END OF DIAGNOSIS')
50                RETURN
51              END

52              SUBROUTINECOMBT(N,A,YACTL,INDEX)
53              DIMENSION A(20),B(20)
54             DO 10 I=1,N
```

```
      IF(ABS(YACTL-A(I)).GE.0.01) GO TO 11
      WRITE(6,300) I,INDEX
300   FORMAT(//,5X,' A SINGLE FAULT MAY OCCUR AT THE',I5,'-TH',
     12X,'LOAD IN THE ',I5,'-TH GROUP')
11      CONTINUE
10    CONTINUE
      IF(N.LT.2) GO TO 200
      N1=N-1
       DO 20 I=1,N1
        I1=I+1
       DO 20 J=I1,N
      B(J)=A(I)+A(J)
      IF(ABS( YACTL-B(J)).GT.0.01) GO TO21
      WRITE(6,301) I,J,INDEX
301   FORMAT(//,5X,' THE DOUBLE FAULTS MAY OCCUR AT',I5,',',I5,
     1'TH LOAD   IN THE',I5,'-TH GROUP')
21      CONTINUE
20    CONTINUE
      IF(N.LT.3) GO TO 3000
      N1=N-2
      DO 30 I=1,N1
        N2=N-1
      I1=I+1
       DO 30 J=I1,N2
       J1=J+1
       DO 30 K=J1,N
      B(K)=A(I)+A(J)+A(K)
      IF(ABS(YACTL-B(K)).GT.0.01) GO TO 31
       WRITE(6,302) I,J,K,INDEX
302   FORMAT(//,5X,'THE TRIPPLE FAULTS MAY OCCUR AT ',I5,',',I5,',',
     1 I5,' IN THE',I5,'-TH GROUP')
31    CONTINUE
30       CONTINUE
3000  IF(N.LT.4) GO TO 400
         N1=N-3
      DO 40 I=1,N1
       I1=I+1
        N2=N-2
       DO 40 J=I1,N2
       J1=J+1
       N3=N-1
        DO 40 K=J1,N3
          K1=K+1
       DO 40 L=K1,N
      B(K)=A(I)+A(J)+A(K)+A(L)
       IF(ABS(YACTL-B(L)).GE. 0.01) GO TO 41
      WRITE(6,303) I,J,K,L ,INDEX
303   FORMAT(//,5X,'THE QUADRAPLE FAULTS',
     1I5,',',I5,',',I5,',',I5,'   IN THE ',I5,' GRROUP')
41    CONTINUE
40     CONTINUE
200    CONTINUE
400     CONTINUE
      RETURN
       END
```

$DATA

III 2.b Fault Detection Using Dynamic Observer Networks

We assume that measurements are made at discrete sampling time instants and, hence, the dynamic model of the distribution system of section III.1 is converted to a set of difference equations:

$$\underline{x}\,(k+1) = A\,\underline{x}\,(k) + B\,\underline{f} \qquad (1)$$

where the available measurements are

$$\underline{y}\,(k) = C\,\underline{x}\,(k) \qquad (2)$$

The vector of inputs $\underline{f}$ are "pseudoinputs" that do not exist in the actual distribution system, but are introduced in the fault detection system design to simulate possible fault conditions. For example, $\underline{f} = \underline{0}$ might represent the no-fault condition and $\underline{f} = \underline{f}^1$, a vector of constant components, would represent a faulted condition characterized, in the case of an open circuit, by having a state component $x_i = 0$. A dynamic observer will be designed to use the available distribution system measurements to obtain estimates of the state of the power system and thereby determine whether a fault condition exists.

A discrete dynamic observer is a sampled-data system described by

$$\hat{\underline{x}}\,(k+1) = F\,\hat{\underline{x}}\,(k) + K\,\underline{y} \qquad (3)$$

$$y\,(k) = C\,\underline{x}\,(k) \qquad (4)$$

where

$$F = A - K\,C \qquad (5)$$

Note that the inputs to the observer are the measurements (2). The gain matrix K is selected so that the state of the observer $\hat{\underline{x}}$ (k) is an estimate of the state $\underline{x}$ (k). Using (1) - (4) it can be shown that the observation error

$$\underline{e}\ (k) = \underline{x}\ (k) - \hat{\underline{x}}\ (k) \qquad (6)$$

and the difference between actual measurements and observer outputs

$$\tilde{\underline{y}}\ (k) = y\ (k) - \hat{y}\ (k) \qquad (7)$$

satisfy

$$\underline{e}\ (k+1) = F\ \underline{e}\ (k) + B\ \underline{f} \qquad (8)$$

$$\tilde{\underline{y}}\ (k) = C\ \underline{e}\ (k) \qquad (9)$$

where

$$\underline{e}\ (o) = \underline{x}\ (o) - \hat{\underline{x}}\ (o) \qquad (10)$$

The automatic fault detection strategy is as follows: a sequence of measurements of the distribution system $\underline{y}$ (1), $\underline{y}$ (2), ..., $\underline{y}$ (n) are made. Then, with the observer initially set in the zero state, $\hat{\underline{x}}$ (0) = $\underline{0}$, the sequence $\tilde{\underline{y}}$ (1), $\tilde{\underline{y}}$ (2), ..., $\tilde{\underline{y}}$ (n) is obtained from (3) and (4). Then, the following matrix equation is obtained:

$$\begin{bmatrix} \tilde{y}(1) \\ \tilde{y}(2) \\ \vdots \\ \tilde{y}(n) \end{bmatrix} = \left[\begin{array}{c|c} CF & CB \\ CF^2 & C(FB+B) \\ \vdots & \vdots \\ CF^n & C(F^{n-1}+\ldots+F+I)B \end{array}\right] \left[\begin{array}{c} \underline{e}(o) \\ \hline \underline{f} \end{array}\right] \qquad (11)$$

for the unknown $\underline{e}$ (0) = $\underline{x}$ (0) and $\underline{f}$.

Assume that when the measurement process, starts the power system is in either a faulted or a no-fault steady-state condition, $\underline{x}(0) = \underline{x}_{ss}$.

Then, from (1)

$$\underline{e}\,(0) = (I - A)^{-1} B \underline{f} \tag{12}$$

One can rewrite (12) as

$$\begin{bmatrix} \underline{e}_1 \\ \hline \underline{e}_n \end{bmatrix} = \left[\begin{array}{c|c} M_{11} & M_{12} \\ \hline M_{21} & M_{22} \end{array}\right] \begin{bmatrix} \underline{f}_1' \\ \hline f_n \end{bmatrix}, \text{ where } \underline{e}(0) = \begin{bmatrix} \underline{e}_1 \\ \hline \underline{e}_n \end{bmatrix}, \underline{f} = \begin{bmatrix} \underline{f}_1' \\ \hline f_n \end{bmatrix}$$

Thus,

$$\underline{e}_1 = M_{11} \underline{f}_1' + M_{12} f_n \tag{13}$$

and

$$e_n = M_{21} \underline{f}_1' \tag{14}$$

Also one can rewrite (11) as

$$\begin{bmatrix} \tilde{\underline{y}}_1 \\ \hline \tilde{y}_n \end{bmatrix} = \overset{\begin{matrix} n-1 & 1 \end{matrix}}{\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]} \begin{matrix} \} n-1 \\ \} 1 \end{matrix} \begin{bmatrix} \underline{e}_1 \\ \hline e_n \end{bmatrix} + \overset{\begin{matrix} n-1 & 1 \end{matrix}}{\left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]} \begin{matrix} \} n-1 \\ \} 1 \end{matrix} \begin{bmatrix} \underline{f}_1' \\ \hline f_n \end{bmatrix} \tag{15}$$

Eliminating the $\underline{e}_1$ in (15) using (13), (14) gives

$$\begin{bmatrix} \tilde{\underline{y}}_1 \\ \hline \tilde{y}_n \end{bmatrix} = \left[\begin{array}{c|c} A_{11} M_{11} + B_{11} & A_{12} \\ \hline A_{21} M_{11} + B_{21} & A_{22} \end{array}\right] \begin{bmatrix} \underline{f}_1' \\ \hline e_n \end{bmatrix} + \begin{bmatrix} A_{11} M_{12} f_n + B_{12} f_n \\ \hline A_{21} M_{12} f_n + B_{22} f_n \end{bmatrix} \tag{16}$$

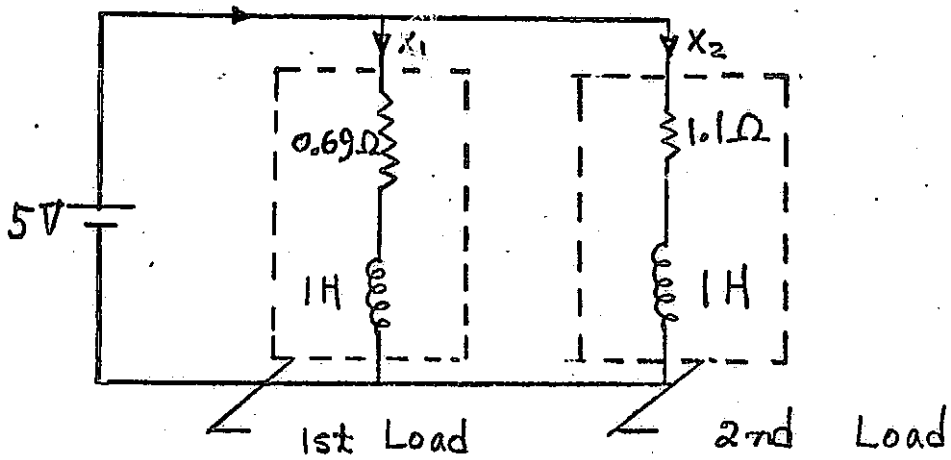
Where f_n = 0 (fault) or α_n (no-fault), and α_n is a given constant.

The problem of fault detection is reduced to the problem of solving the linear algebraic equations in (16) for $\underline{f}_1'$ and e_n .

One must be careful in designing the observer system to guarantee that (16) can be solved uniquely. As yet, general conditions that guarantee uniqueness have not been obtained. The example below illustrates application of the approach to a simple problem.

Example

Consider a system illustrated by



One can formulate a corresponding discrete system with zero-order hold as follows:

$$\begin{bmatrix} x_1[(k+1)T] \\ x_2[(k+1)T] \end{bmatrix} = \begin{bmatrix} e^{-0.69T} & 0 \\ 0 & e^{-1.1T} \end{bmatrix} \begin{bmatrix} x_1(kT) \\ x_2(kT) \end{bmatrix} + \begin{bmatrix} (1-e^{-0.69T})/0.69 & 0 \\ 0 & \dfrac{(1-e^{-1.1T})}{1.1} \end{bmatrix} \underline{f}'$$

$$\equiv A\,\underline{x}(kT) + B\,\underline{f}'$$

$$y(kT) = x_1(kT) + x_2(kT) \equiv \underline{c}'\,\underline{x}(kT)$$

where

$$\underline{f}' = \begin{bmatrix} 5 - f_1 \\ 5 - f_2 \end{bmatrix}, \quad \text{and} \quad y(\cdot) \text{ is}$$

the measuring output.

One can also obtain the corresponding observer with gain vector K so that (F, C) is observable, where

$$F = A - \underline{K}\, C$$

Choose $K = \begin{bmatrix} 18/25 \\ 14/75 \end{bmatrix}$, $T = 1$

Then the corresponding observer can be built as

$$\underline{x}\,(k+1)\,T = F\,\underline{x}\,(kT) + \underline{K}\, y^{*}$$

Where y^{*} is an actual output measurement.

Using the program listed below we have simulated the automatic fault location procedure for the following output measurement:

(i) $y^{*} = 7.2464$

(ii) $y^{*} = 4.5455$

(iii) $y^{*} = 11.792$

Input Data to the program:

N = 2

C(1) = 1, C (2) = 1

K(1) = 18/25, K (2) = 14/75

A(1, 1) = Exp. (-0.69), A (1, 2) = A(2, 1) = 0. 0

A(2, 2) = Exp. (-1. 1)

B(1, 1) = (1- Exp. (-0.69)) /0. 69

B(2, 2) = (1- Exp. (-1. 1)) / 1. 1

B(1, 2) = B(2, 1) = 0. 0

(i) THE OUPUT MEASUREMENT Y= 0.72464E 01

THE N X 2N MATRIX

****

-0.40509E 00 -0.57380E 00 0.72235E 00 0.60648E 00 0.19559E 00

0.20777E 00 0.42974E 00 0.25848E 00

F(1),F(2),.....,F(N-1),E(N), N= 2 ARE
****

0.50000E 01 0.48601E-06

↑ Fault in the 2nd load

(ii) THE CUPUT MEASUREMENT Y= 0.45455E 01

THE N X 2N MATRIX

****

-0.40509E 00 -0.57380E 00 0.72235E 00 0.60648E 00 0.19559E 00

0.20777E 00 0.42974E 00 0.25848E 00

F(1),F(2),.....,F(N-1),E(N), N= 2 ARE
****

0.41787E-06 0.45454E 01

↑ fault in the first load

(iii) THE OUPUT MEASUREMENT Y= 0.11792E 02

THE N X 2N MATRIX

****

-0.40509E 00 -0.57380E 00 0.72235E 00 0.60648E 00 0.19559E 00

0.20777E 00 0.42974E 00 0.25845E 00

F(1),F(2),....,F(N-1),E(N), N= 2 ARE

****

0.50000E 01 0.45454E 01 No fault

AUTOMATIC FAULT DETECTION PROGRAM (TWO LOADS)

```
      $JOB
  1         DIMENSION A(10,10),B(10,10),C(10),AK(10),AF(3),FF(2)          192)
  2           N=2
  3             AF(3)=5./1.1+5./0.69
  4            AF(1)=5./1.1
  5             AF(2)=5./0.69
  6                FF(1)=0.0
  7              FF(2)=5.
  8           DO 100 IJK=1,3
  9          WRITE(6,500) AF(IJK)
 10  500      FORMAT('1',///,'  THE OUPUT MEASUREMENT Y=',E15.5,//)
 11           NN=1
 12  300        CONTINUE
 13            C(1 )=1.0
 14             C(2)=1.0
 15            AK(1)=18./75.
 16            AK(2)=14./75.
 17               U=5.
 18           A(1,1)=EXP(-0.69)
 19            A(1,2)=0.
 20            A(2,1)=0.
 21            A(2,2)=EXP(-1.1)
 22            B(1,1)=(1.-EXP(-0.69))/0.69
 23           B(1,2)=0.
 24            B(2,1)=0.
 25            B(2,2)=(1.-EXP(-1.1))/1.1
 26           FO=FF(NN)
 27          CALL FAULT(AF(IJK),A,B,C,AK,U,N,FO,INDEX,IND)
 28          IF(INDEX.EQ.0.AND.IND.EQ.1) GO TO 200
 29           NN=NN+1
 30           IF(NN.GT.2) GO TO 200
 31          GO TO 300
 32  200      CONTINUE
 33  100     CONTINUE
 34           STOP
 35           END

 36          SUBROUTINE FAULT(Y0,A,B,C,AK,U,N,F0,INDEX,IND)
 37          DIMENSION AK(10),Y(10),FF(10),X(10,10),A(10,10)
 38           DIMENSION B(10,10),C(10),O(10,20),S(2),D(2,2),BB(2)
 39          DIMENSION DD(2,2),F(10,10)
 40              LMN=1
 41            NUMBR= 0
 42  171        CONTINUE
 43           IND=0
 44            INDEX=0
 45            N3=N-1
 46              NUMBR=1+NUMBR
 47           DO 5 I=1,N
 48           DO 6 J=1,N
 49  6        X(I,J)=0.
 50  5        CONTINUE
 51            DO 10 I=1,N
 52              DO 11 J=1,N
 53           O(I,J)=0.
 54  11         CONTINUE
 55  10       CONTINUE
 56              DO 20 I=1,N
 57            DO 21 J=1,N
 58           O(I,J)=AK(I)*C(J)
```

```
      21          CONTINUE
      20        CONTINUE
              DO 30 I=1,N
              DO 31 J=1,N
                F(I,J)=A(I,J)-D(I,J)
      31        CONTINUE
      30          CONTINUE
              N2=N*2
            CALL MATFM(N,C,F,B,D)
           DO 40 I=1,N
            DO 41 J=1,N
      41      A(I,J)=-A(I,J)
      40      CONTINUE
            DO 50 I=1,N
      50      A(I,I)=A(I,I)+1.C
           DO 60 J=1,N
            DO 61 I=1,N
      61    BB(I)=B(I,J)
            DO 65 IJK=1,N
             DO 62  IJJ=1,N
      62    D(IJK,IJJ)=A(IJK,IJJ)
      65      CONTINUE
           CALL GAUSS(N,D,BB,S,ILL)
            DO 63 K=1,N
      63          DD(K,J)=S(K)
      60        CONTINUE
           DO 70 I=1,N
            DO 71 J=1,N
           D(I,J)= DD(I,J)
      71      CONTINUE
      70      CONTINUE
            N1=N-1
            DO 80 I=1,N1
             DO 81 J=1,N1
              A(I,J)=0.0
            DO 82 K=1,N1
      82    A(I,J)=A(I,J)+D(I,K)*D(K,J)
      81    CONTINUE
      80    CONTINUE
             DO 90 I=1,N1
             DO 91 J=1,N1
      91    DD(I,J)=A(I,J)+D(I,J+N)
      90    CONTINUE
             DO 100 I=1,N
     100    DD(I,N)=D(I,N)
             DO 110 I=1,N1
            S(I)=0.
            DO 111 J=1,N1
      111     S(I)=S(I)+D(I,J)*D(J,N)
      110     CONTINUE
            DO 120 I=1,N1
      120     S(I)=S(I)+D(I,2*N)
            CALL OBSCY(X,Y,N,YC,C)
            DO 130 I=1,N1
      130    BB(I)=Y(I)-S(I)*F0
           DO 140 I=1,N1
           DD(N,I)=0.0
            DO 141 J=1,N1
      141     DD(N,I)=DD(N,I)+D(N,J)*D(J,I)
         DD(N,I)=DD(N,I)+D(N,N+I)
```

```
      140    CONTINUE
             CC(N,N)=D(N,N)
             R=0.0
             DO 160 I=1,N1
      160    R=R+D(N,I)*D(I,N)
            R=R+D(N,2*N)
             BB(N)=Y(N)-R*F0
             CALL GAUSS(N,CD,BB,S,ILL)
             DO 170 I=1,N3
            IF(ABS(S(I)-U).LE.0.001.OR.ABS(S(I)).LE.0.001) GO TO 170
             INDEX=1
      170    CONTINUE
           IF(ABS(S(N)-F0      *D(N,N)).LE.0.CC1) IND=1
             LMN=LMN+1
           IF(INDEX.EQ.0.AND.IND.EQ.1) GO TO 6000
             IF(NUMBR.GT.1) GO TO 5000
             GO TO 171
     6000   WRITE(6,2100)
            WRITE(6,5502)
             WRITE(6,2000)((D(I,J),J=1,N2),I=1,N)
            WRITE(6,2600)
     2600    FORMAT(//////,' ')
          WRITE(6,5501)N
            WRITE(6,5500)
     5500    FORMAT(' ','**********************************************',//)
             WRITE(6,2000)(S(I),I=1,N)
     2100   FORMAT(///,' THE  N X 2N MATRIX',/)
     5501  FORMAT(' ','F(1),F(2),....,F(N-1),E(N), N=',I3,2X,'ARE')
     5502    FORMAT(' ','**********************************************')
     2000     FORMAT(' ',8E15.5)
     5000      CONTINUE
           RETURN
           END

            SUBROUTINEMATFM(N,C,F,B,D)
            DIMENSION D(10,20),C(10),F(10,10),C2(10)    ,B(10,10)
           DIMENSION D(10,10),C3(10)
          DO 20 K=1,N
      20    C2(K)=C(K)
          DO 23 IJK=1,N
          DO 22 K=1,N
      22    C3(K)=0.0
          DO 24 I=1,N
          DO 25 J=1,N
      25    C3(I)=C3(I)+C2(J)*F(J,I)
      24    CONTINUE
            DO 26 L=1,N
            C2(L)=C3(L)
            D(IJK,L)=C3(L)
      26    CONTINUE
      23    CONTINUE
            DO 30 I=1,N
           C2(I)=0.
            DO 31 J=1,N
            C2(I)=C2(I)+C(J)*B(J,I)
      31    CONTINUE
      30    CONTINUE
           DO 50 I=1,N
          C(I,N+I)=0.0
           DO 51 J=1,N
```

```
      51     O(I,I+N)=O(I,I+N)+C(J)*B(J,I)                                    195)
   50         CONTINUE
              DO 40 IJK =2,N
                DO 42 J=1,N
              O(IJK,J+N)=0.0
                DO 43 K=1,N
                O(IJK,J+N)=O(IJK,J+N)+O(IJK-1,K)*B(K,J)
   43           CONTINUE
   42           CONTINUE
   41           CONTINUE
   40          CONTINUE
               DO 61I=1,N
               DO 60 J=1,N
   60          Q(I,J)=O(I,N+J)
   61          CONTINUE
               DO 70 I=2,N
               DO 71 J=1,N
   71           O(I,J+N)=O(I,J+N)+Q(I-1,J)
   70          CONTINUE
                 RETURN
               END

               SUBROUTINE OBSCY(X,Y,N,YO,C)
               DIMENSION X(10,10),Y(10),Z(10),C(10)
                Z(1)=0.0
               DO 10 K=1,N
          X(2,K+1)=(EXP(-1.1)-14./75.)*X(2,K)-14./75.*X(1,K)+14./75.*YO
            X(1,K+1)=(EXP(-0.69)-18./25.)*X(1,K)-18./25.*X(2,K)+18./25.*YO
                Z(K+1)=0.0
              DO 15 J=1,N
                Z(K+1)=Z(K+1)+X(J,K+1)*C(J)
   15            CONTINUE
   10          CONTINUE
               DO 20 K=1,N
               Y(K)=YO-Z(K+1)
   20           CONTINUE
               RETURN
                END

            SUBROUTINE GAUSS(N,A,B,X,ILL)
             DIMENSION A(N,N),B(N),X(N)
              ILL=0
            IF(N-1)4,1,4
    1        IF(A(1,1))2,3,2
    2        X(1)=B(1)/A(1,1)
              RETURN
    3         ILL=1
              RETURN
    4         NLESS1=N-1
              DO 13 I=1,NLESS1
              BIG=ABS(A(I,I))
               L=I
               IPLUS1=I+1
                DO 6 J=IPLUS1,N
              IF(ABS(A(J,I))-BIG) 6,6,5
    5         BIG=ABS(A(J,I))
               L=J
    6         CONTINUE
              IF(BIG)8,7,8
    7          ILL=1
```

```
236                RETURN
237     8          IF(L-I)9,11,9                                          196)
238     9          DO 10 J=1,M
239                TEMP=A(L,J)
240                A(L,J)=A(I,J)
241     10        A(I,J)=TEMP
242                TEMP=B(L)
243                              B(L)=B(I)
244               B(I)=TEMP
245     11         DO 13 J=IPLUS1,N
246                QUOT=A(J,I)/A(I,I)
247                DO 12K =IPLUS1,N
248     12           A(J,K)=A(J,K)-QUOT*A(I,K)
249     13           B(J)=B(J)-QUOT*B(I)
250                IF(A(N,N)) 15,14,15
251     14         ILL=1
252                RETURN
253     15           X(N)=B(N)/A(N,N)
254                I=N-1
255     16           SUM=0.
256                IPLUS1=I+1
257                DO 17 J=IPLUS1,N
258     17              SUM=SUM+A(I,J)*X(J)
259                X(I)=(B(I)-SUM)/A(I,I)
260                I=I-1
261                IF(I) 18,18,16
262       18       RETURN
263                END
```

```
$DATA
```