MICROCOMPUTER PROCESS-CONTROL AND PROCESS-SIMULATION: HARDWARE AND SOFTWARE CONSIDERATIONS

A Graduate Project Submitted

For Completion of CS-595

by

William Mitchell Bunting

February, 1982

TABLE OF CONTENTS

LIST OF FIGURES

I. Summary

Digital to analog, analog to digital, and digital to digital interfacing is important in many computer applications involving the petrochemical industry. This graduate project implemented these three types of interfaces using two Cromemco microcomputers, a process-simulator program, and a process-controller program. The primary purpose of the two programs was to demonstrate the interfaces.

The process-simulator was ultimately run in one microcomputer while the process-controller was run in a second microcomputer. The two computers were linked by a digital to digital interface through one parallel port in each computer.

All analog interfacing was done with a single computer. The appropriate digital signals from the process-simulator were converted to analog with a Cromemco D+7AI/O board and output to an analog recorder. The analog output was also input to the same D+7AI/O board to serve as input to the process-controller (Figure 1).

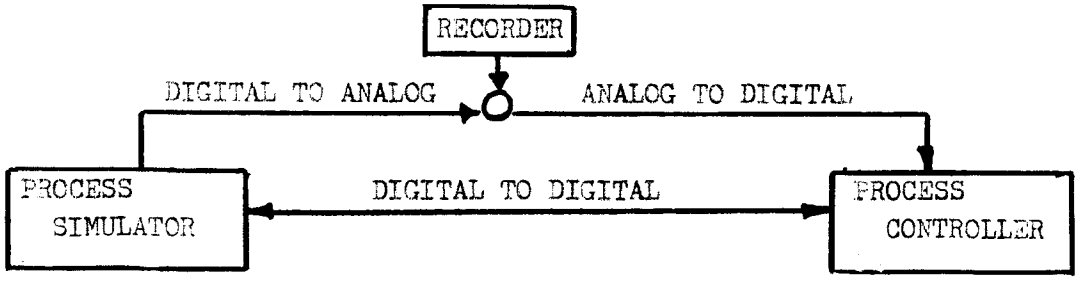


Figure 1: Types of Interfaces Implemented

This report discusses digital to analog, analog to digital, and digital to digital interfaces, both in theory and as used, as well as the process-simulator and process-controller programs.

II. The Process-simulator and Process-controller Programs

The process-simulator and process-control programs were written in Cromemco 16K BASIC. They relate to the petroleum process of naphtha reforming. Naphtha reforming is the primary process for upgrading the octane of gasoline, particularly unleaded gasoline. This process uses a platinum-containing or similar catalyst to convert low octane aliphatic hydrocarbons to high octane aromatics (primarily benzene, toluene, and xylenes) and higher octane, more highly branched aliphatic hydrocarbons. The primary undesirable side reactions occurring are coke deposition on catalyst reducing catalyst activity and light gas, primarily methane, production which is low in value compared to reformate, the liquid product from naphtha reforming.

The process-simulator and process-controller programs were written to generate and control only temperature data. Often times mathematical models of petrochemical processes are based upon a constant temperature across a reactor (isothermal), whereas in reality a considerable temperature gradient occurs. Naphtha reforming is such a process. In the front of the reactor (several reactors are used in practice) cyclic hydrocarbons are converted to aromatics in an endothermic reaction while in the back part of the reactor an undesirable exothermic side reaction occurs, namely hydrocracking. The resulting temperature profile is shown in Figure 2. This temperature profile is produced by the process-simulator upon receipt of a single set point temperature from the process-controller.

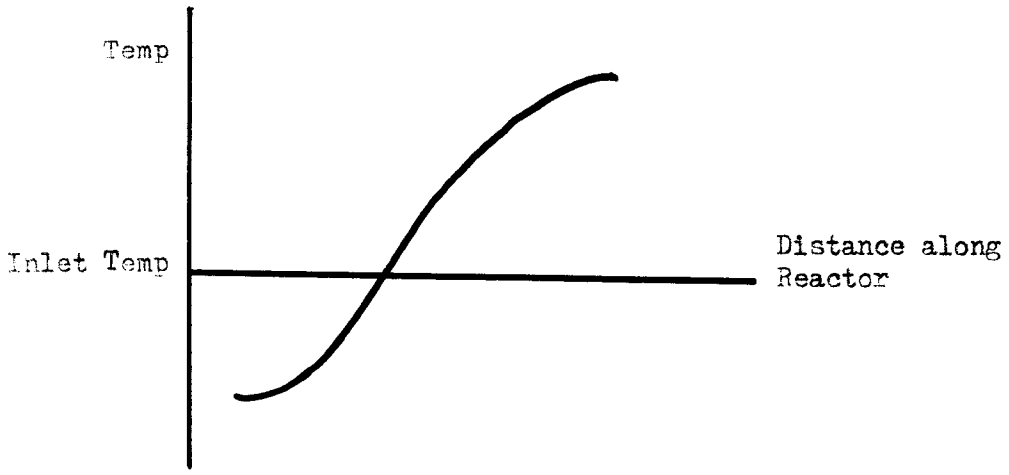


Figure 2: Typical temperature profile along a naphtha reforming reactor.

Values for two variables are input at the beginning of a computer run: a pseudo-random number generator seed for the BASIC random number generator function and the desired number of simulation-control cycles. Flowcharts for the process-controller and process-simulator are shown in Figures 3 and 4.

A run begins with the process-controller sending the process-simulator a setpoint temperature. The process-controller generates the setpoint temperature from the cubic equation:

$$Y = 75*(T1^3 + 1) + 850$$

where Y is the setpoint temperature and T1 is an independent variable which increases each cycle by an amount determined from the number of simulation-control cycles desired. When T1 is greater or equal to one, the process-controller program stops(and hence the entire simulation-control process stops). This cubic equation generates the temperature-time profile shown in Figure 5.

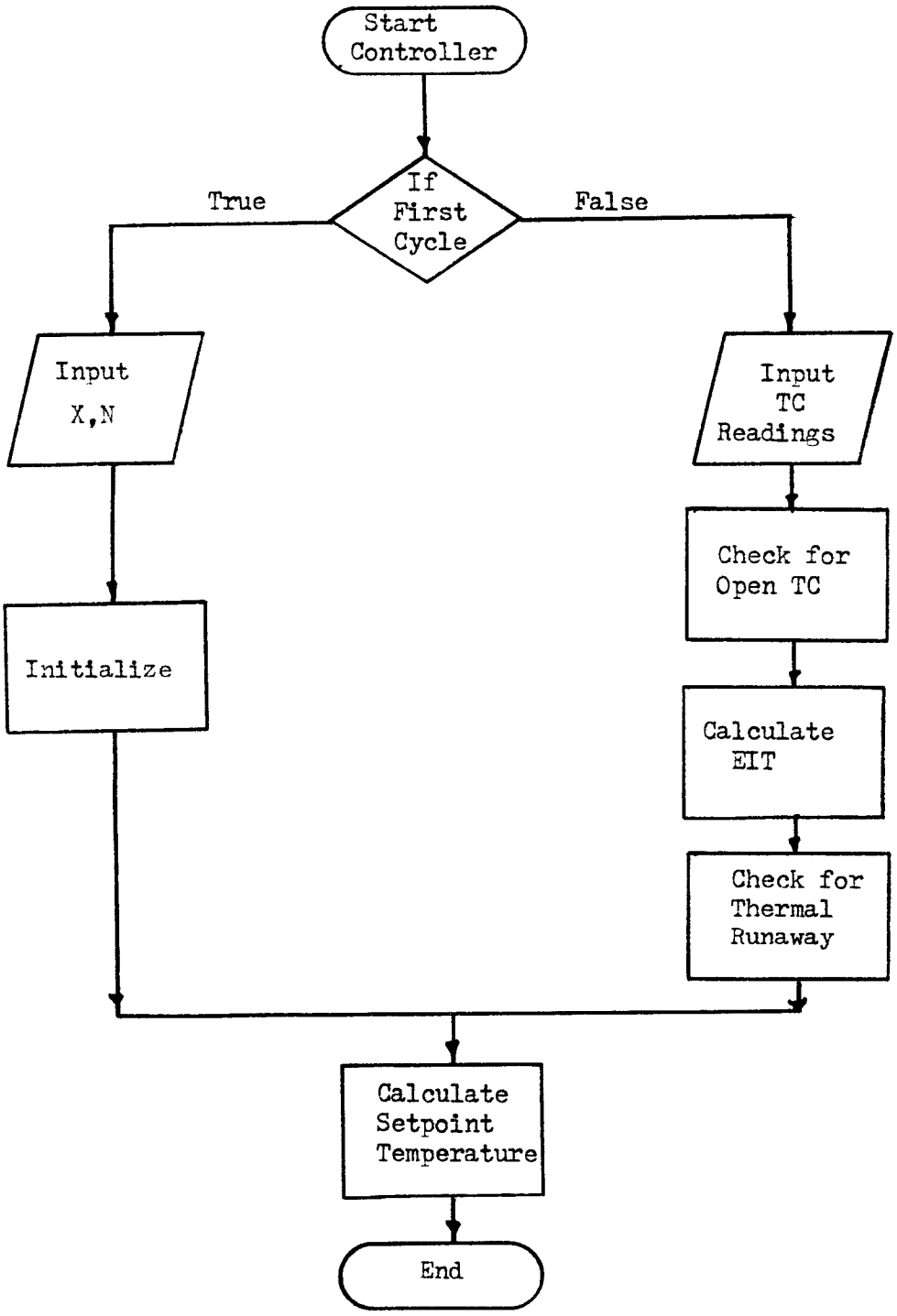


Figure 3: Flowchart for Process Controller
X = Random Number Seed, N = Number of Cycles.

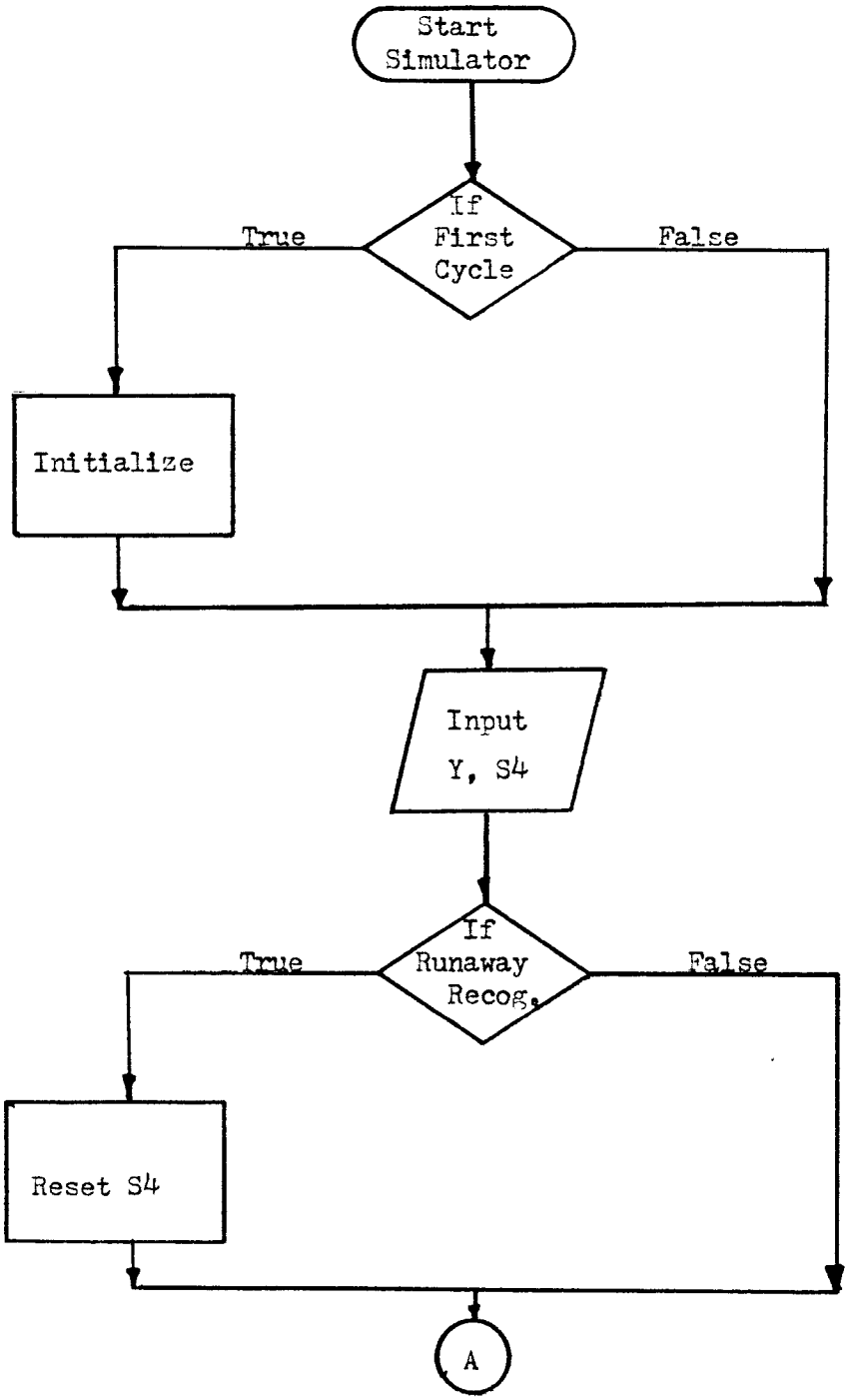


Figure 4: Flowchart for Process Simulator
Y = Setpoint Temperature, S4 = Thermal Runaway Switch (Continued on Next Page).

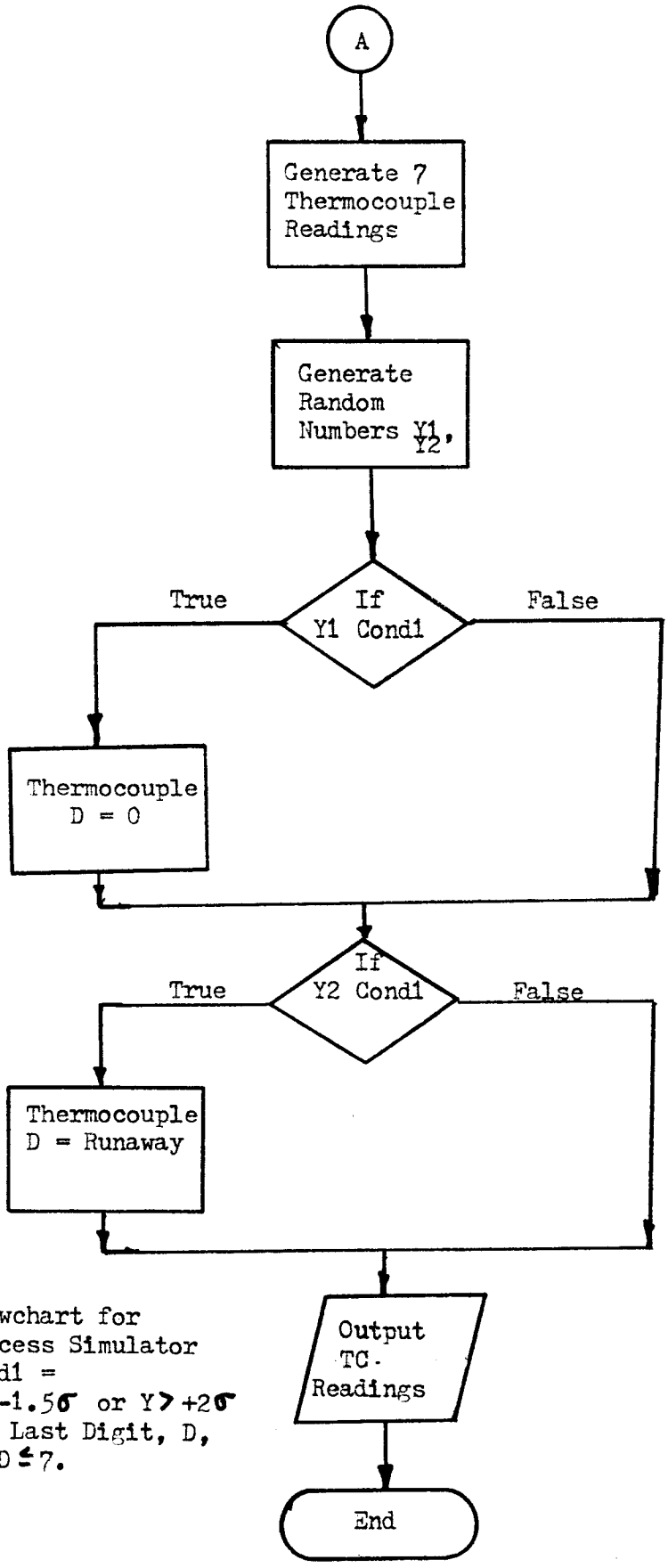


Figure 4: Flowchart for Process Simulator
Cond1 = $Y < -1.5\sigma$ or $Y > +2\sigma$ and Last Digit, D, $1 \leq D \leq 7$.

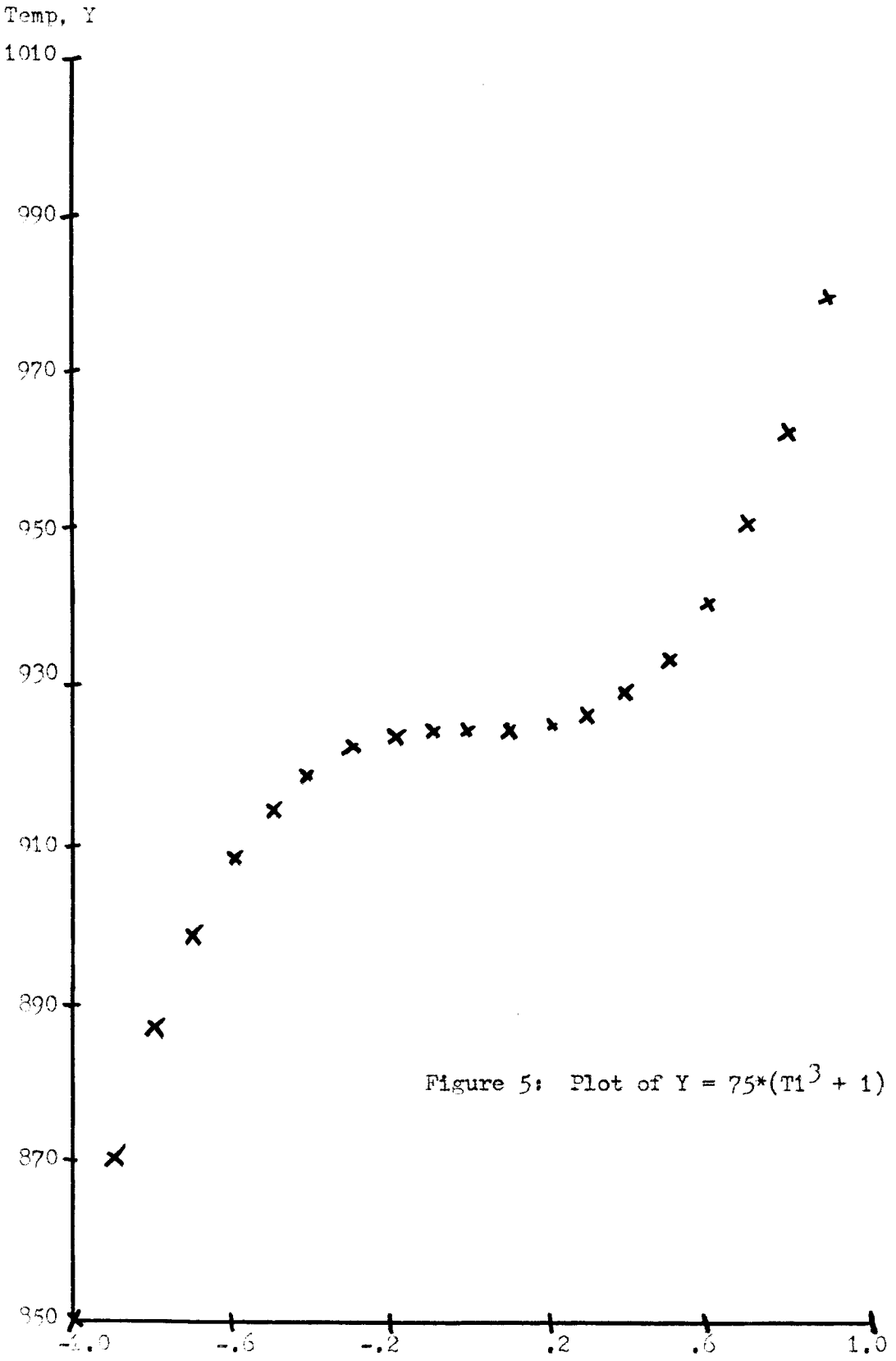


Figure 5: Plot of $Y = 75*(T1^3 + 1) + 850.$

The process-simulator converts the target temperature into seven separate thermocouple readings. This simulates the temperatures recorded in seven different places in a reactor. The reactor temperature profile shown in Figure 2 is generated by multiplying the target temperature (setpoint temperature) by seven different constants. In reality this temperature profile is a function of catalyst type, feedstock type, age of catalyst, history of catalyst, hydrogen/oil ratio, space velocity, temperature, total pressure, feedstock poison content (sulfur, nitrogen, and water primarily) and a host of other variables.

In order to generate run to run variations, a random element is introduced into the process-simulator output. The process-simulator generates pairs of pseudo-random numbers drawn from a normal distribution of mean = 50 and standard deviation = 1. If the first random number is outside -1.5 sigma to 2.0 sigma and the last digit is one through seven, that thermocouple fails (goes open, temperature = 0). If the second random number is outside -1.5 sigma to 2.0 sigma and the last digit is one through seven, a thermal runaway (local reactor temperature increases without limit at an increasing rate) is measured on that thermocouple.

Since an open thermocouple is easy to spot and in practice usually easy to fix, all thermocouples are assumed to be working at the beginning of each process-simulator cycle. In other words a failed thermocouple is not carried over from one cycle to the next. In contrast, a thermal runaway is often difficult to spot, at least

initially. Thermal runaways are carried over from cycle to cycle until recognized by the process-controller. A thermal runaway is recognized when the thermocouple reading is at least 20° F more than expected by the process-controller. When one thermal runaway is recognized by the process-controller, the process-controller sends a signal to the process-simulator which resets all thermocouples recording thermal runaways. In practice a reactor may have several hot spots (thermal runaways) at once. The techniques applied to quenching one hot spot generally quench all hot spots.

A thermocouple reporting a thermal runaway is carried from one process-simulator cycle to another by recording the thermocouple number, its current temperature, and number of cycles it has been thermally running away in a matrix. The next cycle its temperature is calculated from the equation $T = T + 5^{N}$ where T is the temperature reading recorded in the matrix and N is the number of cycles recorded in the matrix.

Seven thermocouple readings are transmitted to the process-controller. In addition, these simulator temperatures are output to the printer and, when appropriate, to an analog recorder. The process-controller notes the invalid (open) thermocouple readings, if any, and calculates an equivalent isothermal temperature (EIT) from the valid temperature data. The EIT is compared to the setpoint temperature. The setpoint temperature, EIT, and difference are output to the printer along with any malfunctioning thermocouples and thermal runaways. A new target temperature is then sent to the process-simulator. This process continues for the requested number of cycles.

The pairs of normally distributed random numbers are generated using a variation of a method suggested by Box and Muller.[1] First, two uniformly distributed pseudo-random numbers, r_1 and r_2, are generated from the built-in BASIC function RND(X) where X is the seed for generating the pseudo-random number:

$$V_1 = 2r_1 - 1$$

$$V_2 = 2r_2 - 1$$

$$S = V_1^2 + V_2^2$$

if $S \geq 1$, two new random numbers are generated and the process begins again.

if $S < 1$ then

$$Y_1 = V_1((-2\ln S)/S)^{\frac{1}{2}}$$

$$Y_2 = V_2((-2\ln S)/S)^{\frac{1}{2}}$$

Y_1 and Y_2 are the desired normally distributed random numbers. r_2 becomes the seed for RND(X) to generate the next pair of uniformly distributed pseudo-random numbers. The numbers are pseudo-random numbers since the same seed entered at the beginning of the program always generates the same sequence of numbers.

This method of generating random numbers drawn from a normal distribution was compared to the normal distribution function by generating two-hundred random numbers in a computer run and comparing their distribution with that for the normal distribution. The results showed that this method satisfactorily generates pseudo-random numbers drawn from a normal distribution (Figure 6). A listing of the program and example printer output for this computer run are given in Appendix G.

The process-simulator and process-controller were initially run in the same microcomputer with a main program calling them as

subroutines. This step assured that the programs ran satisfactorily in the absence of interfacing problems. A listing of the program as it ran at this point and sample printer output are shown in Appendix A.

Values within	Found	Expected
±one sigma	143	137
±two sigma	191	191
±three sigma	200	199

Figure 6: A comparison of the method used for generating pseudo-random numbers drawn from a normal distribution with the normal distribution for 200 values.

III. Digital to Analog and Analog to Digital Interfaces

Theory of Digital to Analog Conversion

In general, digital to analog conversion (or DAC) is easier and cheaper than analog to digital conversion (ADC). In addition, D/A converters form the basis for a large fraction of all A/D converters.

There are many techniques for effecting D/A conversion.[2] The basic concept of DAC is shown in Figure 7. A digital interface circuit converts the digital inputs to the control levels of a set of switches. These switches operate in conjunction with a precision resistor network to give binary weighted currents or voltages. The precision resistor network is referenced to a stable precision voltage source. The output of the precision resistor network is the sum of

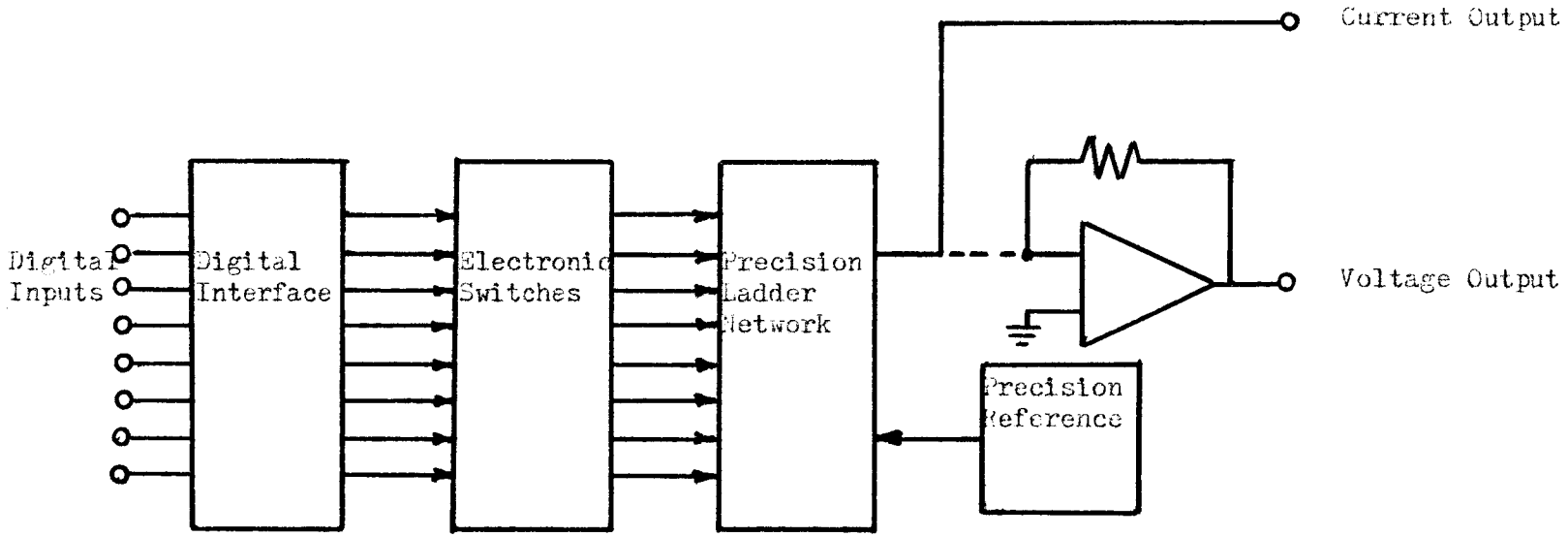


Figure 7: Conceptual Diagram of a Digital to Analog Converter*

*Datel Electronic Products Handbook

all the binary weights in the form of a voltage or a current.

The precision ladder network shown in Figure 7 can include transistors and diodes.[2] An example of a D/A converter using only resistors in the precision ladder network is shown in Figure 8. From point X in Figure 8 looking to the right a resistance of R is seen and looking to the left a resistance of 2R is seen; from point X' looking to the right a resistance of 2R is seen and looking to the left a resistance of R is seen. These properties hold for any of the junctions along the ladder. If a 2R resistor is switched to the voltage reference source, the source sees a resistance of 2R plus 2R in parallel with 2R, or 3R total, and a current of $V_{ref}/3R$ flows into the junction. At the junction this current divides equally, with half flowing to the left and half to the right. The right-hand current flows to the next junction where it is again divided in half, and so on to the right end of the ladder where it becomes part of the total output current.

D/A converters are commercially available as an integrated circuit. An example of such an integrated circuit is the MC 1408L8 used in the Cromemco D+7AI/O board. These integrated circuits almost certainly do not use precision resistor networks due to the difficulty of predicting solid state (silicon) resistor values with great precision.[3]

Theory of Analog to Digital Conversion

Analog to digital conversion is more expensive than digital to analog conversion. A digital to analog converter can be used in conjunction with a microprocessor executing an algorithm (eg. a binary

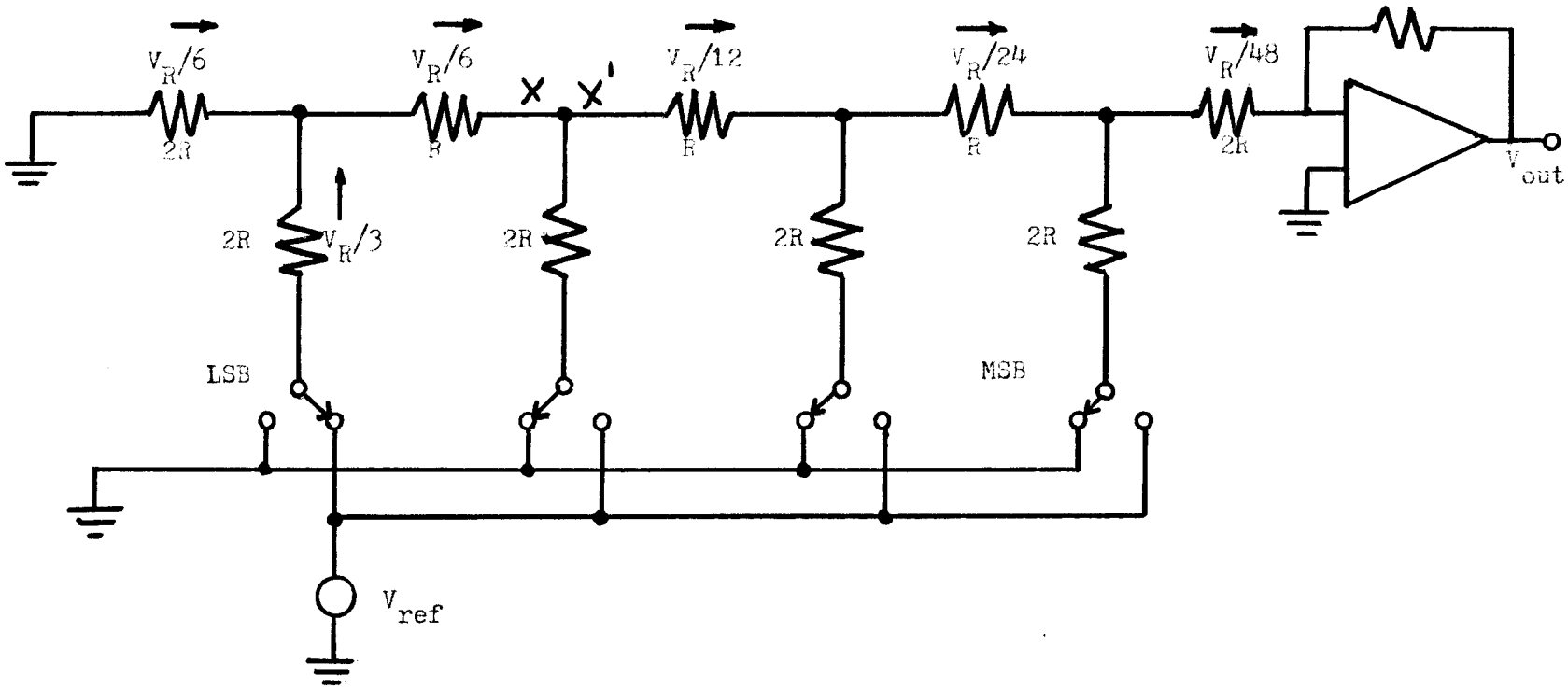


Figure 8: R/2R Network Digital to Analog Converter*

*From Datel Electronic Products Handbook

search) to search for the digital equivalent of the analog input voltage (Figure 9).[4]

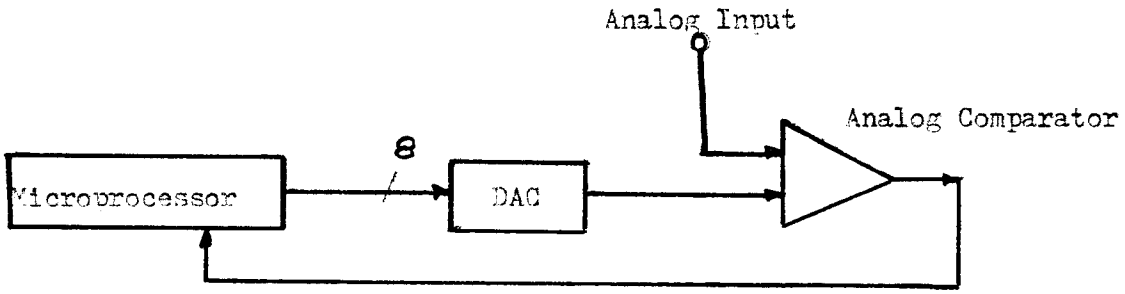


Figure 9: Use of a D/A Converter in Analog to Digital Conversion.

The first guess of the microprocessor would be binary 1000 0000 (to convert the analog input to eight binary digits). This digital guess is converted to analog by the D/A converter and compared to the analog input signal by the analog comparator which outputs a binary 1 if the signal is high or a binary 0 if the signal is low. Based on the output of the analog comparator the microprocessor decides whether its guess should remain a 1000 0000 or be changed to 0000 0000. The second guess would be D100 0000 where D stands for the preceding decision, and so on.

The Cromemco D+7AI/O board implements the binary search algorithm in hardware external to the microprocessor. The hardware is a 2502 successive approximation register.

Accomplishing Digital to Analog and Analog to Digital Conversion

All experiments involving analog signals were made using the Cromemco D+7AI/O circuit board. This board has eight I/O ports,

seven analog and one digital, and provides both analog to digital and digital to analog conversion. A detailed pin-out for this board is shown in Figure 10.[5]

CONNECTOR PIN ASSIGNMENTS

COMPONENT SIDE		SOLDER SIDE	
Analog Ground	A	Analog Ground	1
Analog Input 7	B	Analog Output 7	2
6	C	6	3
5	D	5	4
4	E	4	5
3	F	3	6
2	H	2	7
Analog Input 1	J	Analog Output 1	8
-12V Regulated	K	+12V Regulated	9
Analog Ground	L	Analog Ground	10
-17V Unregulated	M	+17V Unregulated	11
-5V Regulated	N	+5V Regulated	12
Input $\overline{\text{Stb}}$	P	Output $\overline{\text{Stb}}$	13
Parallel Input Bit 7	R	Parallel Output Bit 7	14
6	S	6	15
5	T	5	16
4	U	4	17
3	V	3	18
2	W	2	19
1	X	1	20
Parallel Input Bit 0	Y	Parallel Output Bit 0	21
Digital Ground	Z	Digital Ground	22

Figure 10: D+7AI/O Connector Pin Assignments

The D+7AI/O uses the successive approximation register technique to convert analog to digital.[5] During digital to analog conversion, the successive approximation register is used only as a timing device to generate a sufficient number of wait states (5.0 microsec. of wait states at 4MHz) to allow output to stabilize. Analog outputs are not latched and must be refreshed at a rate of 1Hz or faster by out-

put instructions. The D+7AI/O is hardwired to respond to port addresses 24 (the digital port) and 25-31 (the seven analog ports).

Sending Digital Output to an Analog Recorder

A Houston Instruments one channel analog recorder was used. A one volt input to the recorder provides full scale deflection. The recorder was connected to pins 1 and 2 of the D+7AI/O board (which had been previously plugged into the S-100 microcomputer bus). Since the recorder had only one channel, only one thermocouple reading was sent to the D+7AI/O board and on to the recorder. The BASIC code to communicate with the recorder is shown in Figure 11.

```
1780  IF T(0)<800 OR T(0)>1100 THEN T(0) = 800
1790  J9 = INT((T(0) - 800)/6)
1800     FOR I = 1 TO 20
1810     OUT 31,J9
1820        FOR J = 1 TO 8
1830        H = SIN(1)
1840        NEXT J
1850     NEXT I
```

Figure 11: Cromemco 16K BASIC code to communicate with recorder through port 31.

Line 1780 compresses temperature readings into the range of 800 to 1100°F in order to provide more sensitivity in differentiating slightly different temperatures in the recorder scan. The least significant bit corresponds to 20mV and full scale deflection on the recorder is one volt. Normal thermocouple readings (excluding thermal runaways and open couples) will be in the range of 850 to 1010°F. Open couples have T=0 and thermal runaways are detected by the process-controller when they exceed the expected temperature by

20°F. Hence the maximum temperature reading possible is 1030°F unless the computer program malfunctions. Line 1790 reduces the temperature reading to a one byte digit in the range of 0 to 50 (0 to 1.0 volts). Line 1810 outputs the temperature to an analog port. This output is refreshed at a rate slightly in excess of 1Hz. The loop in lines 1820 to 1840 is a delay loop to give the recorder, which is slow relative to the microprocessor, time to respond with a meaningful mark on the recorder chart paper.

Example recorder scans are shown in Figures 12 and 13. The program listings and printer output accompanying the scans shown in Figures 12 and 13 are given in Appendix B. It should be noted that the curve shape in Figures 12 and 13 is the same as the curve shape in Figure 5, which was the objective. The curve in Figure 5 is drawn from digital output, while the curves in Figures 12 and 13 are generated from the equivalent analog output.

Inputting Analog Data to the Microcomputer

The appropriate digital signals were converted to analog by the D+7AI/O board. The analog output from the D+7AI/O board was then input to the same D+7AI/O board and converted to digital data for input to the microprocessor. This was accomplished by connecting the pairs of pins on the D+7AI/O board shown in Figure 14 (a detailed pin-out was shown in Figure 10).

2-B	3-C
4-D	5-E
6-F	7-H
8-J	

Figure 14: Pairs of Connector Pins Wired Together for Analog Output and Subsequent Analog Input Using the Cromemco D+7AI/O Board.

Figure 12: Recorder Scan, Twenty Cycles,
Random Number Seed is One

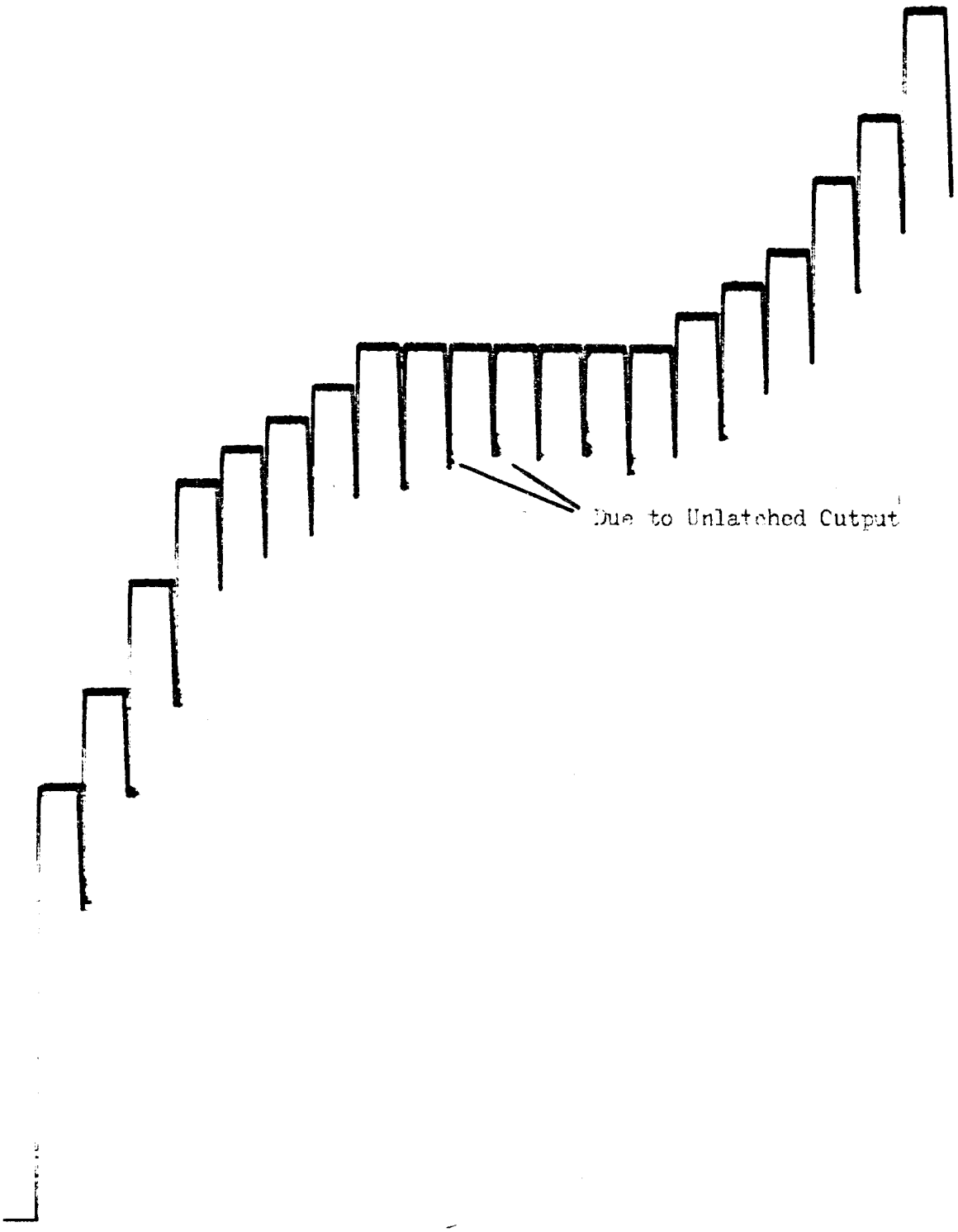
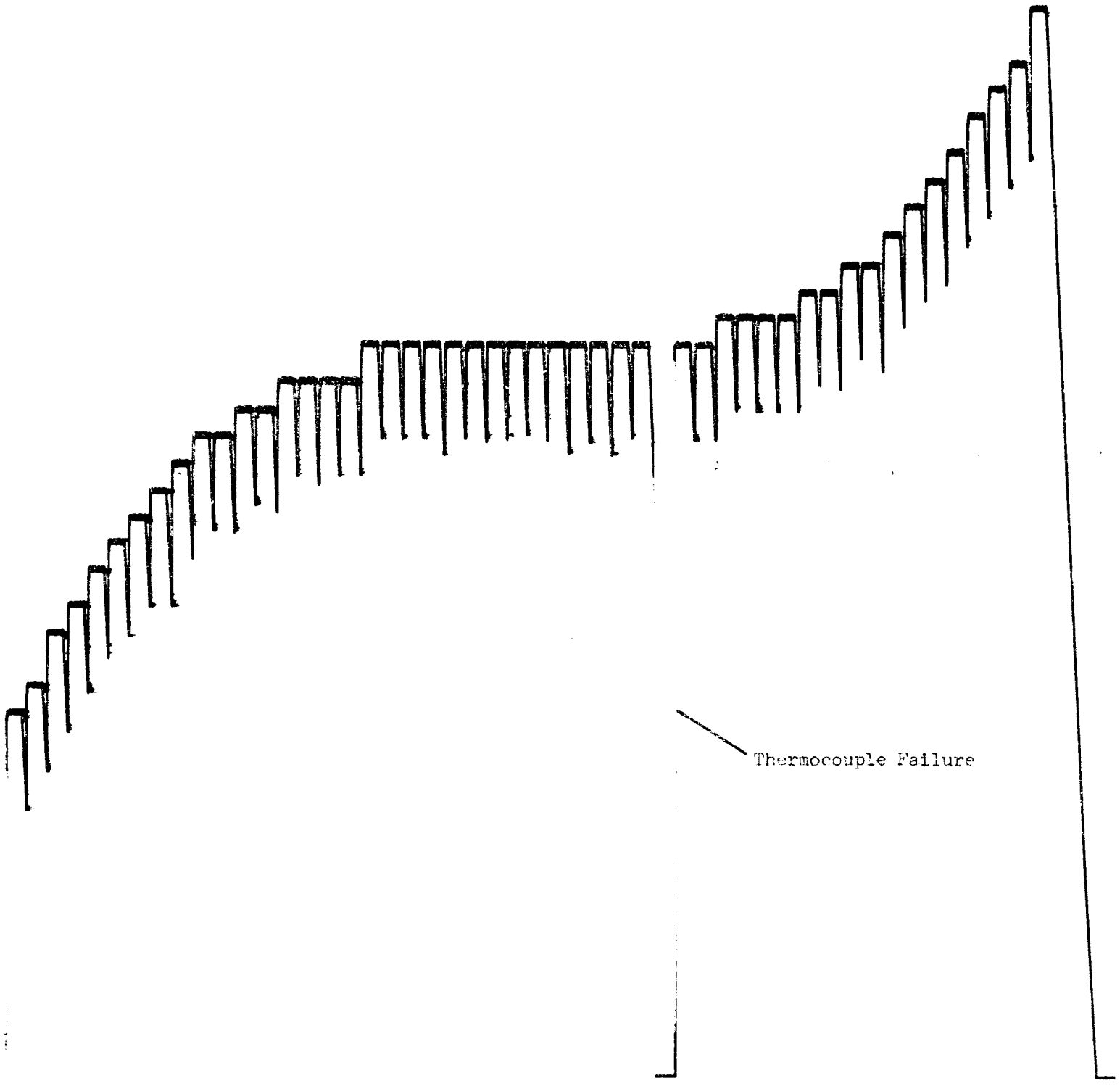


Figure 13: Recorder Scan, Fifty Cycles, Random Number Seed is One



The BASIC code to accomplish the digital to analog and analog to digital conversion is shown in Figure 15. This code is similar to the

```
1870  FOR J=0 TO 6
1880  IF T(J)<800 OR T(J)>1100 THEN T(J)=800
1890  NEXT J
1900  FOR J=0 TO 6
1910  B=INT((T(J)-*)))/6)
1920  P=25+J
1930  OUT P,B
1940  A(J)=INP(P)
1950  A(J)=6*A(J)+800
1960  NEXT J
```

Figure 15: Cromemco 16K BASIC Code to Accomplish Digital to Analog and Analog to Digital Conversion.

code already discussed for driving a recorder (shown in Figure 11) except that all seven thermocouple readings are being converted and no delay loop is necessary since no device is being interfaced to the microprocessor. The program listing and printer output ARE shown in Appendix C. It should be mentioned that two of the input analog temperatures in cycle 1 of Appendix C give values of about 2300°F instead of values in the expected range (850-1030°F) indicating that the analog input signal had not stabilized before being converted back to digital. Subsequent cycles give accurate conversions. Attempts to solve this problem with software (other than putting in a dummy cycle 1) were not successful indicating the problem may be in the printed circuit board.

IV. Digital to Digital Interfacing of Two Microcomputers

The hardware problems involved in connecting two microcomputers together by means of a digital interface are nominal provided one has

the required hardware. One merely wires together two parallel ports, one in each of the two microcomputer systems. The main problem involved in such a digital interface is one of timing. One computer must be waiting for a piece of information, say thermocouple one's reading, from computer number two when computer number two is sending thermocouple one's reading, not when computer number two is sending thermocouple seven's reading. The required timing is accomplished by software handshaking protocol: "I'm ready to send the data, are you ready to receive it? Yes, I'm ready, go ahead and send it. Did you get the data? yes, I got the data."

In this project two Cromemco microcomputers were wired together utilizing the parallel port on the D+7AI/O board discussed previously and an unused parallel port on an available Cromemco TU-ART board.

Higher Level Language Code

Initially, however, the parallel interface was accomplished using a single computer and the parallel port available on the D+7AI/O board for both output and input. The output pins of the parallel port were wired to the input pins of the same parallel port. The use of a single microcomputer avoids the timing problem entirely and allows one to concentrate initially on the less difficult problem of sending a multi-byte representation of a number through a one-byte port.

Two approaches were used. One approach was to convert the number to a character string and send the characters through the parallel port, one character at a time, and subsequently convert the character string back to a number. This approach can be implemented entirely from within Cromemco 16K BASIC using the VAL and STR$ functions. The

second approach is to send the number in its internal, binary representation through the interface one byte at a time. This approach must be implemented in Z80 assembly language and the assembler program linked to the BASIC program using the USR and POKE functions. This second approach is limited to sending two-byte integer numbers when implemented with Cromemco 16K BASIC due to the nature of the USR function. These approaches are discussed further, each in turn.

The BASIC implementation is discussed first. Representative code for sending a long-floating-point number through a one-byte port entirely from BASIC is shown in Figure 16. In line 1480,

```
1480  A$=STR$(Y1)
1490      FOR J=0 TO 14
1500      C=VAL(A$(J,J))
1510      IF A$(J,J)="." THEN OUT 24,%002E%
1520      IF A$(J,J) <>"." THEN OUT 24,C
1530      IF INP(24)=%002E% then B$(J,J)="."
1540      IF INP(24)<>%002E% then B$(J,J)=STR$(INP(24))
1550      NEXT J
1560  Y1=VAL(b$)
```

Figure 16: Cromemco 16K BASIC code for sending a multi-byte number through a one-byte port.

Y1 is a long-floating-point number (14 digits plus a decimal point, eight bytes) which is converted to a 15-character string. Line 1500 takes each character in turn. If it is a character representation of a digit, it is converted to a two-byte integer (the digit is stored entirely in the lower byte). If it is not a character representation of a digit, it is assigned the value zero. Hence there is a problem in handling the decimal point (ASCII $2E_X$), which is addressed in lines 1510 to 1540.

The use of the VAL function in line 1500 might be considered to be an unnecessary level of indirection. The 16K BASIC, however, does not allow the statement OUT 24,C$; it only allows OUT 24,C. In other words, one cannot output string variables with the OUT command. A program listing and representative printer output for this implementation is shown in Appendix D.

The BASIC/Z80 Assembler implementation involves first writing the appropriate Z80 assembler code to send and subsequently receive a two-byte integer, hand-translating the assembler code into decimal or hexadecimal, and then having the BASIC program incorporate suitable statements to read the translated assembler code and output it as machine language to a known memory location which is empty (it is not advisable to overlay either the operating system or the BASIC program). The instruction POKE is used by BASIC to place a one-byte number into a specified memory location. The assembler program and BASIC code to suitably locate the assembler code is shown in Figure 17. Calling the assembler program (translated) causes a two-byte integer to be output and subsequently input. Note that only the lower byte of a two-byte integer is poked into place.

Figure 18 shows the BASIC code to link to the Z80 assembler code shown in Figure 17 to transmit the seven thermocouple readings discussed earlier. The USR function specifies an address to branch to followed by a set of parameters which are converted to two-byte integers and placed on a stack. This is the same stack accessible from Z80 assembler and pointed to by the SP register. The contents of the DE register pair are assigned to A(J) in line 1880 by convention.

Z80 Assembler Subroutine and Hexadecimal Translation

```
POP   BC          C1        ;get integer from stack
LD    A,B         78        ;load first byte
OUT   (24),A      D318      ;output first byte
IN    A,(24)      DB18      ;input first byte
LD    D,A         57        ;return first byte to BASIC
LD    A,C         79        ;load second byte
OUT   (24),A      D318      ;output second byte
IN    A,(24)      DB18      ;input second byte
LD    E,A         5F        ;return second byte to BASIC
RET               C9        ;return to BASIC
```

BASIC code to POKE Assembler Routine into Suitable Memory Location

```
FOR J=%C000% to %C00D%
READ N
POKE J,N
NEXT J
DATA  %00C1%,%0078%,%00D3%,%0018%,%00DB%
DATA  %0018%,%007A%,%0079%,%00D3%,%0018%
DATA  %00DB%,%0018%,%005F%,%00C9%
```

Figure 17: A Z80 Assembler Routine and the Cromemco 16K BASIC Code to Suitably Locate the Code so that the BASIC Program Can Subsequently Link to the Assembler Code.

```
1370  FOR J=0 TO 6
1380  A(J)=USR(%C000%,T(J))
1390  NEXT J
```

Figure 18: Cromemco 16K BASIC code to link to Z80 Assembler Code to Transmit Seven Thermocouple Readings.

A program listing and representative printer output for this implementation is shown in Appendix E. The less common BASIC commands used in this report are summarized in Appendix H.

The Timing Problem

To address the timing problem involved in synchronizing two microcomputers, the process-simulator was run in one microcomputer and the process-controller was run in a second microcomputer. The two computers were linked through a parallel port in each computer as previously discussed at the beginning of this section (Section IV). Only integers were transmitted from one computer to the other. The subroutines to fetch an integer and to send an integer were the same for both computers (although the port addresses were different). Sending and fetching was implemented entirely from within the Cromemco 16K BASIC language in the format already discussed.

The timing problem can be resolved into two components. In the first component of the problem the BASIC code of the two programs must be written such that the order of sending and receiving data in one program is the same as the order of receiving and sending data in the other program. Note that the order of receiving and sending data is reversed in the two programs.

The second component of the problem is the handshaking protocols necessary in transmitting and receiving data: "are you ready to send? Yes, are you ready to receive? Yes, go ahead and send. Did you get the data? Yes, I got the data." When a computer is not ready to send or not ready to receive, it outputs hexadecimal FF from its parallel port. The port outputs are latched and do not need to be refreshed as in the case of analog outputs discussed earlier. When a computer is ready to receive data, it changes its port output to hexadecimal

00. Four digit integers were transmitted, a digit at a time. The receiving computer switches its output port alternately between hexadecimal FF and hexadecimal 00 to show receipt of the data. The BASIC code to send and fetch an integer is shown in Figure 19.

Subroutine to Send an Integer

```
1520  J$=STR$(J1)
1530  L=LEN(J$)
1540  S$="    "
1550     FOR M=4-L TO 3
1560     S$(M,M)=J$(M-4+L,M-4+L)
1570     NEXT M
1580  C=VAL(S$(0,0))
1585  IF INP(84)<>%0000% THEN GO TO 1585
1590  OUT 84,C
1600  C=VAL(S$(1,1))
1610  IF INP(84)<>%00FF% THEN GOTO 1610
1620  OUT 84,C
1630  C=VAL(S$(2,2))
1640  IF INP(84)<>%0000% THEN GOTO 1640
1650  OUT 84,C
1660  C=VAL(S$(3,3))
1670  IF INP(84)<>%00FF% THEN GOTO 1670
1680  OUT 84,C
1690  IF INP(84)<>%0000% THEN GOTO 1690
1700  OUT 84,%00FF%
1710  RETURN
```

Subroutine to Fetch an Integer

```
1020  OUT 84,%0000%
1030  IF INP(84)=%00FF% THEN GOTO 1030
1040  S$(0,0)=STR$(inp(84))
1050  S$(0,0)=STR$(INP(84))
1060  OUT 84,%00FF%
1070  S$(1,1)=STR$(INP(84))
1080  S$(1,1)=STR$(INP(84))
1090  OUT 84,%0000%
1100  S$(2,2)=STR$(INP(84))
1110  S$(2,2)=STR$(INP(84))
1120  OUT 84,%00FF%
1130  S$(3,3)=STR$(INP(84))
1140  S$(3,3)=STR$(INP(84))
1150  OUT 84,%0000%
1160  J1=VAL(S$)
1165  OUT 84,%00FF%
1170  RETURN
```

Figure 19: Cromemco 16K BASIC Code to Send and Fetch an Integer through a Parallel Port Connecting Two Computers.

When fetching an integer, it should be noted that two fetches are made for each byte of data, e.g. lines 1040 and 1050. The first fetch is in effect a delay loop to allow the output from the other microprocessor parallel port to stabilize and latch. Reliable input data were not received without duplicating these input instructions. If another parallel port were available, a more elegant approach would be to check the appropriate output strobe for valid data. The instructions in lines 1550 to 1570 of the sending routine are necessary to right justify the character representation of digits in the four-byte field. If right justification is not done, integers of less than four digits pick up trailing zeroes.

Program listings and representative printer output are shown in Appendix F for the digital-digital interfacing of the process-simulator and process-controller run in separate microcomputers.

V. EPILOGUE

All programs were written in Cromemco 16K BASIC which is an interpreted language. In actual process control applications, programs are generally written in a compiled language, or even in assembler, to improve execution speed.

REFERENCES

1. Maisel, Herbert and Gnugnoli, Giuliano, Simulation of Discrete Stochastic Systems, Science Research Associates, Inc., 1972, pp.153-154.

2. The discussion on D/A and A/D converters is taken primarily from Datel Engineering Product Handbook, 1977-1978, Datel Systems, Inc., 1020 Turnpike Street, Canton, Mass. 02021.

3. Meindl, James D., Scientific American, 237, 76 (1977).

4. Garland, Harry, Introduction to Microprocessor System Design, McGraw-Hill Book Co., 1979, pp. 145ff.

5. Cromemco D+7AI/O Technical Manual Revision E, Cromemco, Inc., 280 Bernardo Ave., Mountain View, CA 94040.