MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A
THESIS

AUTOMATED DESIGN OF MICROPROCESSOR-BASED DIGITAL FILTERS

by

Martin Ralph Heilstedt

June 1983

Thesis Advisor: H. H. Loomis

Approved for public release; distribution unlimited
Automated Design of Microprocessor-Based Digital Filters

Martin Ralph Heilstedt

Naval Postgraduate School
Monterey, California 93940

Naval Postgraduate School
Monterey, California 93940

Approved for public release; distribution unlimited

Computer Aided Design; Automated Design; Digital Filters; Microprocessor Based Systems

This thesis investigates the feasibility of automating the design of microprocessor-based digital filters. The ability of a prototype design system to successfully produce filter realizations is tested. General filter structures and programming algorithms are presented. Shortcomings in the current version of the design system are determined.
Modifications are made as required to support digital filter realizations. The feasibility of filter generation is demonstrated using realistic examples taken from the literature.
ABSTRACT

This thesis investigates the feasibility of automating the design of microprocessor-based digital filters. The ability of a prototype design system to successfully produce filter realizations is tested. General filter structures and programming algorithms are presented. Shortcomings in the current version of the design system are determined. Modifications are made as required to support digital filter realizations. The feasibility of filter generation is demonstrated using realistic examples taken from the literature.
TABLE OF CONTENTS

I. INTRODUCTION
------------------------------------- 10

II. SYSTEM OVERVIEW
---------------------------------- 17
   A. PROBLEM MODEL
-------------------------------- 17
      1. Contingency/Task Pairs 17
      2. High Level Problem Description 18
   B. INTERMEDIATE FORM OF THE PROBLEM
---------------------------------- 21
   C. FUNCTIONAL ELEMENTS
---------------------------------- 22
      1. Optimizer Module 22
      2. Functional Mapper 24
      3. Timing Analyzer 25
      4. Formatter Module 26
   D. REALIZATION LIBRARY
---------------------------------- 27

III. FILTERING METHODS
---------------------------------- 31
   A. TRANSFER FUNCTION
---------------------------------- 31
      1. First Direct Structure 32
      2. Second Direct Structure 35
      3. Third Direct Structure 37
      4. Fourth Direct Structure 39
   B. SECOND ORDER SECTIONS
---------------------------------- 39
   C. COMBINATIONAL STRUCTURES
---------------------------------- 42
      1. Cascade Structure 42
      2. Parallel Structure 46
   D. DESIGN CONSIDERATIONS
---------------------------------- 47
      1. Sampling Frequency 48
2. Structure Selection---------------------- 50

IV. MODIFICATIONS TO THE SYSTEM---------------------- 53
   A. PARALLEL PROCESSING---------------------- 53
      1. Single Contingency/Multiple Tasks-------- 53
      2. FORK Construct-------------------------- 54
      3. Timing Requirements---------------------- 55
   B. MULTIPLE REFERENCES TO I/O VARIABLES------ 66
      1. Hardware Binding------------------------- 66
      2. Symbol Table Listing of I/O Variables---- 66
   C. ANALOG TO DIGITAL INTERFACING------------- 67
      1. Processor Controlled Conversions-------- 67
      2. Library Entry Development---------------- 68

V. DESIGN EXAMPLES----------------------------- 72
   A. CASCADE REALIZATION---------------------- 73
      1. CSDL Description------------------------- 73
      2. Intermediate Representation------------- 80
   B. PARALLEL REALIZATION---------------------- 83
      1. CSDL Description------------------------- 83
      2. Intermediate Representation------------- 89

VI. CONCLUSIONS AND RECOMMENDATIONS------------- 90
   A. CONCLUSIONS----------------------------- 90
   B. RECOMMENDATIONS------------------------- 90
      1. Implementation Language---------------- 90
      2. Validation of Current Program---------- 91
      3. Realization Library--------------------- 92
      4. Interrupt Driven Monitors--------------- 93
5. Applications Problems--------------------- 94
6. Documentation----------------------------- 94
APPENDIX A - DATA, CASCADE REALIZATION--------- 95
APPENDIX B - DATA, PARALLEL REALIZATION-------- 136
APPENDIX C - ERROR CORRECTIONS---------------- 186
LIST OF REFERENCES----------------------------- 189
INITIAL DISTRIBUTION LIST---------------------- 191
LIST OF FIGURES

1. The Design Process---------------------------------- 12
2. Automated Hardware and Software Generation-------- 14
3. Example CSDL Listing-------------------------------- 20
4. Flowchart of Automated Design System---------------- 23
5. Sample Realization Library Entry--------------------- 30
6. Flow Diagram, Nth Order Transfer Function---------- 33
7. Flow Diagram, First Direct Structure---------------- 36
10. Flow Diagram, Fourth Direct Structure-------------- 41
11. Block Diagram of Filter Structures------------------ 44
12. Flowchart, Basic Digital Filter Operations--------- 49
13. Delay Time of Filter Structures---------------------- 52
14. CSDL Listing of FORK Construct---------------------- 56
15. Timing Diagram of FORK Construct, Single Processor------------------------------------------- 59
16. Data Loss in Multi-processor FORK Construct Implementation-------------------------------------- 61
17. Timing Diagram of FORK Construct, Multiple Processors------------------------------------------- 63
18. CSDL Listing, Modified FORK Construct------------- 64
19. Analog Input Primitive----------------------------- 70
20. Analog to Digital Converter Primitive------------- 71
21. Flowchart of Cascade Implementation---------------- 75
22. Flowchart of Parallel Implementation---------------- 84
ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Professor Herschel Loomis, and second reader, Lieutenant Colonel Alan Ross, for their assistance and support in this thesis. I appreciate the time and skills of Elaine Christian in typing the manuscript. I thank my parents for their guidance, and my wife, Nancy, for her encouragement, support, and faith in me.
I. INTRODUCTION

Advances in integrated circuit and microprocessor technology have allowed increasingly diverse applications of these devices. The decrease in their size and power requirements and the increased market for them has resulted in a decrease in costs while yielding increased computing power. If this trend continues, an even broader variety of applications can be expected. The effects of increased complexity and the decrease in component costs will lower total hardware costs. However, as shown by Shooman [Ref. 1], software costs can be expected to remain high for three reasons: new applications will require new programs to be written, the replacement of older, existing computers with newer versions will require either completely new software or, at the very least, modifications to existing software, and because programming is highly labor intensive, it will be strongly affected by inflation. In fiscal year 1980, fifty-seven billion dollars were spent on computer systems. Of this amount, thirty-two billion dollars, or fifty-six percent, was allocated to software [Ref. 2]. It can be concluded that the most costly expenditures associated with computer system development and/or modification are software related, and this situation can be expected to remain unchanged. It is therefore to our advantage to automate
software production. The effort, in terms of manpower, required to develop a given software implementation, as well as the monetary expense necessary to support it, can then be devoted to the research and development of new systems and applications.

The application of digital computers to the problems of real time control is an increasingly important aspect of computer technology, and is representative of the uses which can be expected with technological advances. The design of such systems has proven to be a complex and expensive undertaking. The concepts which will reduce this complexity and a methodology for the automated production of real time control systems have been developed. [Ref. 3] While no computer exists which can duplicate the creative process of design illustrated in Figure 1, there are elements of it which can be tasked to the computer for accomplishment. In order to determine what portions of the design process can be automated, the methodology used by the designer must be studied. The steps he follows usually involve combining existing elements or components in such a way that the desired system is realized. The components that are used are selected from a library of such items which is located either in the designer's memory or is contained in a physical listing. Because the components contained in the listing are regularly ordered, a computer can be used to maintain it, and, when given the specifications of a desired
Figure 1. The Design Process
system, it can select the necessary entries. This portion of the design process has been successfully automated [Ref. 4] and is illustrated in Figure 2. The proposed design tool merges the ideas of automated software generation and automated hardware production into one system. The description of this system is the subject of chapter two.

The original intent of the design tool was to automate the production of the software and hardware necessary for the realization of real time controllers, but if this tool is to be a truly useful design aid, it must be applicable to a wider variety of problems. The implementation of digital filters is suitable for a dedicated microprocessor system, and is typical of potential applications problems.

Digital filtering has been the subject of considerable research during the past fifteen years. Various implementations have been achieved, including hardwired logic, special-purpose computers, and general-purpose computers. With the development of the microprocessor, and the recent increases in wordsize and speed, a new alternative is available. The application of the proposed design system to the implementation of microprocessor-based digital filters is the subject of this thesis.

The difference equation form of the filtering function is well-suited to microprocessor realization. However, the algorithm used can be expressed in a variety of ways. Chapter three discusses four possible implementations. The
DESIGNER

PROBLEM STATEMENT

HARDWARE LIBRARY → DESIGN PROGRAM → REALIZATION

SOFTWARE LIBRARY

HARDWARE LISTING

SOFTWARE LISTING

Figure 2. Automated Hardware and Software Generation
mathematical descriptions discussed are chosen for their simplicity and ease of implementation, but should not be construed as the only available alternatives. The possibilities are virtually unlimited. The filtering function can be performed using one of two general methods, independent of the algorithm chosen. The first method defines the transfer function as the product of the first and second order sections. This is the familiar cascade form of the digital filter. The alternative method is the parallel form, which expresses the transfer function as the sum of first and second order sections. [Ref. 5] The methods for the general implementation of each of these applications within the bounds of the design system are also discussed in chapter three. The optimization of the solution is not of concern. The best realization, as determined by metrics such as memory use and chip count, is not the current objective of the design system. If the implementation produced satisfies the requirements of the problem in terms of timing constraints and function, the solution is acceptable.

The application of the design system to problems other than controller design is important for several reasons. By demonstrating the ability of the system to generate successful realizations for a wide variety of problems, its overall utility as a design tool will be proven. Varying the type of implementation problem presented to the system extends the applicability of the design technique and refines our
understanding of the system. By testing various applications, inadequacies in the current implementation of the system can be detected and corrected. The attempt to generate digital filters using the design system revealed a number of such problems and the search for these inconsistencies became a major portion of the thesis. The problems found are presented with corresponding solutions in chapter four.

Chapter five describes the application of the design system to realistic problems. The concepts discussed in the previous chapters are utilized to implement both the cascade and parallel forms of a fourth order digital filter.

The system described in this thesis represents a useable design tool. Unfortunately, the coding that the user is required to produce in order to achieve the desired hardware and software realization is both awkward to use and tedious to generate. The addition of new realization volumes, as well as the maintenance of existing ones, is a difficult process as well. The potential of the design system is therefore limited by our ability to simplify the user input specification and provide a means by which the software and hardware library may be modified to accommodate any potential design problem. Recommendations for these and other improvements are contained in the sixth and final chapter.
II. SYSTEM OVERVIEW

The current version of the design system has evolved from the model proposed by Matelan [Ref. 6] and implemented by Ross [Ref. 7]. Each of these efforts used the design of real time controllers as the problem model for development of the system. The concepts and terminology introduced by Matelan can be found in the current implementation. The results of this work are summarized here.

A. PROBLEM MODEL

The first aspect of the system to be considered is the problem model. In order to produce a successful realization for any problem, it must first be expressed in a form understandable by the design system.

1. Contingency/Task Pairs

Each problem can be expressed as a collection of contingency/task pairs, where a contingency is defined as a logical or relational function of some input variable or variables. The associated task is executed when the contingency is satisfied. Therefore, the first step in developing a realization is to express the problem in terms of contingency/task pairs.
2. **High Level Problem Description**

Implicit in the model of the problem are rigid constraints on the testing of each contingency and the time allowed for execution of the corresponding task. The designer must specify these real time requirements in the statement of the problem. Matelan proposed a new high level design language called a Control System Design Language, or CSDL, for this purpose. The language consists of four sections: identification section, environment section, contingency list, and procedures section. The identification section is a record of the user identification of the problem. The environment section specifies the interface between the device and the process which is to be operated upon. It defines all input and output variables and their characteristics, as well as those variables internal to the mathematical operation of the device. The contingency list is a declaration of those conditions that the device must respond to, the associated task that each must execute, and the real time constraints imposed upon each pair. The timing constraints are determined by the maximum time allowed to recognize a contingency and the maximum time available to execute the corresponding task. The routines which comprise each contingency/task are contained in the procedures section. By definition, contingencies are written as functions, while tasks are specified as procedures. Each contains the high level descriptions necessary for the
performance of its role in the system being produced. Figure 3 is an example CSDL listing of a simple filtering problem. The identification section is readily found and easily understood. The environment section specifies two variables, one for input and one for output. Each contains eight TTL-compatible bits. The contingency section indicates that the function READY is executed every millisecond, and when true, the task FILTER is performed. The procedures section contains the descriptions of the steps necessary to perform the test for the contingency READY and the execution of the process FILTER. In the case of the contingency READY, an external variable, DATA, is read. If its value is one, indicating the presence of data for processing, the contingency is satisfied and the value of READY is set to one, and the task FILTER is executed. If, on the other hand, DATA is equal to zero, the contingency will not be satisfied and READY will remain equal to zero to indicate a false condition. The task FILTER is not executed under these circumstances.

The function FILTER contains the description of the difference equation

\[ y(n) = x(n) + 0.5y(n-1) - 0.5y(n-2). \]

The value of the input is assumed to be in digital form and is read first. The value of the corresponding filtered output is then computed and the result is issued as an output.
IDENTIFICATION:
DESIGNER: M. R. Heilstedt
DATE: 19 April 1983
PROJECT: CSDL Example - Filter

ENVIRONMENT:
INPUT: X,8,TTL
OUTPUT: Y,8,TTL

CONTINGENCY LIST:
When READY:1MS Do FILTER

PROCEDURES:
Contingency READY
  Sense (DATA);
  If DATA=1 Then READY:=1 fi;
Exit READY

Task FILTER
  Sense (X);
  Y=X+(0.5*YN1)-(0.5*YN2);
  Issue (Y);
  YN2=YN1;
  YN1=Y;
Exit FILTER

Figure 3. Example CSDL Listing
The current version of the design system does not support the use of CSDL for problem specifications, requiring that the user generate the list of primitives manually. A user specification format based on the Ada programming language is currently under development. [Ref. 8] In the interim, CSDL serves as a useful tool in understanding the structure of the problem specification required by the design system and the transformation it undergoes in the course of generating a design.

B. INTERMEDIATE FORM OF THE PROBLEM

The input specification is translated into an intermediate representation consisting of a list of primitives. This transformation is analogous to the operation performed by a compiler on high level languages such as Fortran and Pascal. The list of primitives is a complete description of the problem, containing all of the information necessary to generate a hardware and software implementation of the design problem and to verify that all timing requirements have been satisfied with the selected realization volume.

A second file, called the IADEFL file, is generated at the same time as the primitive list, and contains the list of contingency/task pairs and their corresponding timing constraints. This data is extracted from the ID table, the environment table, the timing data from the contingency list, and the design criteria table. The design criteria
table is generated from the data listed in the design criteria section of the CSDL listing. This section was added by Ross to provide a method for the designer to specify the criteria upon which an acceptable realization could be chosen. The criteria are: first realization generated, least costly realization, and the realization that uses the least power. The current version of the system only supports the first realization generated criterion. The creation of the IADEFL file and the primitive list is the first action taken by the design system in its attempt to generate a hardware realization. The operation of the design system in relation to these files is depicted in Figure 4.

C. FUNCTIONAL ELEMENTS

The problem formulation is performed by the user for both the high level description and intermediate representation. Each of these tasks will eventually be automated as well, providing a user friendly interface to the design system. As stated previously, development of the input module is in progress, but until a satisfactory input specification to intermediate form translator can be developed, the input to the design system must be manually compiled.

1. Optimizer Module

After the input files have been produced, they are passed to the first component in the design system, the
Figure 4. Flowchart of Automated Design System
optimizer module. It serves two purposes, acting as the main program controlling the operation of the system, and functioning as the input module for the primitive list. Although not implemented in the current version of the system, this module will have the capability of comparing the multiple realizations generated by the system and selecting that one which best satisfies the chosen metric as specified by the user in the design criteria section of the input listing.

2. **Functional Mapper**

The Functional Mapper is the first module called by the Optimizer, and is used to ensure that each primitive contained in the intermediate problem specification can be realized with the current realization library. This is accomplished using two separate operations. The realization volume index is first searched for the primitive name as given by the current line of the intermediate list. If the primitive is found, the specifications associated with the primitive are compared to those contained in the realization library. The mapping is considered successful only if all of the primitives can be realized and their criteria satisfied. If a given primitive cannot be mapped to a realization, or if its selection criteria fail to fit those of the realization, a new realization library will be searched. Failure to satisfactorily realize a primitive in any library results in an unsuccessful mapping.
3. **Timing Analyzer**

The second module executed is the Timing Analyzer. It generates the monitor necessary for controlling the operation of the device under development, ordering the execution of the realization contingency tests and task executions such that the timing constraints will be satisfied. In order to make such an assurance, the timing analyzer assumes that all contingencies are true. This requires that all of the tasks must be executed by the monitor and therefore defines the worst case situation. The resulting premise is that if the worst case can satisfy the timing requirements, all cases satisfy them as well. The timing analyzer determines the length of time required to execute all of the code contained in each of the contingency/task pairs and compares it to the timing constraints listed in the Applications Timing Table as generated from the IADEFL file. If all of the timing constraints are met, the realization is successful. If not, the realization fails. The output of the Timing Analyzer is used to generate monitor primitives for successful realizations. These primitives determine the sequence of execution for the contingency/task pairs, and are added to the list of primitives derived from the high level description of the problem.

An important result of the research conducted by Ross was that the design system is capable of automatically producing multi-processor realizations. This is done in the
Timing Analyzer. If a single processor realization is impossible, the Timing Analyzer partitions the Applications Timing Table into two separate lists. Each of these contains the timing information corresponding to complete sets of contingency/task pairs. In other words, the partitioning of the problem results in groupings of these pairs. The regularly ordered nature of the contingency/task model makes this possible. In order to generate separate sets of hardware, the two timing tables are passed to the Formatter module individually. Theoretically, it is possible to produce a realization in which each contingency/task pair is located in its own processor. It is important to note that such a system may require shared resources. The current version of the design system will only test for a dual processor realization when a single one fails. The generation of such a realization cannot be forced.

4. **Formatter Module**

The Formatter receives the complete list of primitives necessary to produce the output listing for the design. The listing consists of the software necessary to perform the desired function and the hardware required to support it. The hardware and software listings are written to separate files which are in turn copied to an output device such as a terminal or line printer. The design program is then terminated.
D. REALIZATION LIBRARY

The realization library has thus far been mentioned only in describing the operation of the other components of the design system, but its importance in the determination of a successful design should not be overlooked. Each realization library contains volumes of hardware and software primitives based on individual processor families. The only volume currently implemented uses the Intel 8080 microprocessor and its various support devices to generate realizations.

The general format of each line in a volume is the same. Each line is assigned a unique identifying number, and contains a maximum of eighty characters. The first set of lines within the volume serve as an index to the primitives it contains. The lines of the index are copies of the title line of each entry. The current format of the realization volume allows a maximum of 9999 lines. The index is not considered when determining the number of lines contained in the volume.

Each line of the volume must conform to a specific format, of which there are ten: Primitive Title line, Comment line, Calc line, Attr line, Call line, Include line, If line, Begin Text line, End Text line, and Text line. The title line begins with an S or H, to denote hardware or software, and contains the name of the primitive. As used in the current version of the design system, it is the most important of the lines found in the volume, providing the
correct format for generating each of the entries contained in the intermediate problem description. The calling arguments, selection criteria, and attributes of the primitive are enclosed in parentheses following its name. The attributes, which vary depending on the nature of the primitive, define such parameters as power consumption, latency, and chips used. Any or all of the arguments, selection criteria, and attributes may be omitted, but the commas that separate them must appear. The comment line is denoted by the appearance of the letters C-O-M as the first characters on the line. The text that follows is ignored by the system. The Calc line allows the use of global variables within the system. An example of its application is the counter variable used to total the number of chips used in a particular design. In the current realization library, this variable is called ICN and is incremented by any of the hardware primitives that contain integrated circuits. The Attr line is similar to the Calc line, but is used to compute the value of an attribute of the primitive realization. Inc1 and Call lines are used to invoke other primitives from within a primitive. The difference between the two is in the placement of the output that each generates. The output from a Call is inserted immediately following any previously generated output. Output from an Inc1 statement will be added to that of the primitive lists after all other output from the including primitive has been produced. The If line
provides more flexibility in library construction by allowing branch instructions within the realization volume. The Begin and End Text lines are used to reproduce descriptive lines in the files generated for system output by the Formatter module. These lines are in the final category of text lines. The most common use of text lines is for assembly code associated with a software primitive. Figure 5 is an example of a short realization volume. The construction of the volume is further demonstrated and clarified in the text of the analog to digital converter realization developed in chapter four.
Figure 5. Sample Realization Library Entry
[Ref. 8]

V2222S.SAMPLE (P1,P2:0,8,0,5:10,10,-17,18,19,2222,2258)
V2223COM THIS IS A COMMENT DESCRIBING S.SAMPLE. P1 AND
V2224COM P2 ARE THE DUMMY ARGUMENTS OF S.SAMPLE. 0 AND 8
V2225COM ARE THE MINIMUM AND MAXIMUM VALUES OF THE ACTUAL
V2226COM ARGUMENT REPRESENTED BY P1, 0 AND 5 ARE THE
V2227COM CORRESPONDING MINIMUM AND MAXIMUM FOR THE ACTUAL
V2228COM ARGUMENT REPRESENTED BY P2. THE 10,10 INDICATES
V2229COM THAT THE STORAGE AND TIME ATTRIBUTES FOR THIS ENTRY
V2230COM ARE EACH OF VALUE 10, THE -17 INDICATES THAT THE
V2231COM VALUE OF THE EXTERNAL ATTRIBUTE IS CALCULATED ON
V2232COM LINE 2239 (2222-(-17)=2239). THE 18 INDICATES THAT
V2233COM LINE 2240 (2222+18=2240) CALLS FOR CALCULATION OF
V2234COM SOME GLOBAL VALUE. THE 19 INDICATES THAT LINE 2241
V2235COM CALLS FOR THE INCLUSION OF SOME OTHER REALIZATION.
V2236COM 2222 AND 2258 ARE THE FIRST AND LAST LINE NUMBERS
V2237COM OF THIS REALIZATION.
V2238COM
V2239ATTR EXTERNAL = DUMMY1 * 7
V2240CALC TOTEVT = TOTEVT+1
V2241INCL H.ALSOSAMPLE(0,0)
V2242CALL S.SAMPTHREE (TOTEVT)
V2243COM
V2244COM H.ALSOSAMPLE AND S.SAMPTHREE ARE TWO OTHER ENTRIES.
V2245COM ALSOSAMPLE HAS TWO DUMMY ARGUMENTS, BOTH ASSIGNED
V2246COM VALUE ZERO IN THIS CASE. S.SAMPTHREE HAS ONE DUMMY,
V2247COM SET EQUAL TO THE GLOBAL TOTEVT FOR THIS CASE.
V2248COM
V2249IF DUMMY1 .GE. 2 SKIP 2
V2250INCL H.SAMPFOUR ( )
V2251COM INCLUDE H.SAMPFOUR ONLY IF DUMMY1 IS LESS THAN TWO.
V2252COM BEGIN THE TEXT BLOCK AFTER THE NEXT LINE. .
V2253BEGIN STEXT
V2254EVERYTHING IN THIS BLOCK BETWEEN BEGIN AND END IS
V2255COPIED TO THE OUTPUT LISTING EXCEPT FOR DUMMY
V2256ARGUMENTS AND GLOBALS WHICH ARE REPLACED BY THEIR
V2257ACTUAL ARGUMENTS OR VALUES.
V2258ENDTTEXT

Figure 5. Sample Realization Library Entry
[Ref. 8]
III. FILTERING METHODS

With the advent of the microprocessor, a new and interesting device for the implementation of digital filters has been created. Increases in word length and computational speed will encourage more complex filtering applications using this device. It is therefore important to investigate the feasibility of producing digital filter realizations using the automated design system. The fundamental requirements that the application places on the system can be determined using the Intel 8080 microprocessor library currently available. Despite the filtering limitations presented by its eight bit format, the deficiencies present in the current system can be identified and corrected. The resulting improvements will provide a more useful design tool and make possible the generation of satisfactory solutions to more demanding problems when additional realization volumes constructed around advanced microprocessors are added to the library.

A. TRANSFER FUNCTION

The general form of the digital filter transfer function is

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}{1 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}$$
where \( a_1 \) and \( b_1 \) are real coefficients and \( n \) is the maximum of the orders of the numerator and denominator polynomials.

Figure 6 is a flow diagram representation of this transfer function. There are a number of similar structures that can be used to describe the basic filtering operation. Those in which the real coefficients \( a_1 \) and \( b_1 \) are multipliers are referred to as direct structures. Nagle and Nelson [Ref. 10] describe four direct structures and the equations associated with them.

1. **First Direct Structure**

   The first direct structure is derived from the equation

   \[
   H(z) = \frac{\sum_{i=0}^{n} a_i z^{-i}}{\sum_{i=0}^{n} b_i z^{-i}}.
   \]

   The coefficient \( b_0 \) is equal to one. Defining the input to the filter as \( X(z) \) and the output as \( Y(z) \), the previous equation can be rewritten as

   \[
   \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{n} a_i z^{-i}}{\sum_{i=0}^{n} b_i z^{-i}}.
   \]

   This can be redefined using an intermediate variable \( M(z) \), such that

   \[
   \frac{Y(z) \cdot M(z)}{M(z) \cdot X(z)} = \frac{\sum_{i=0}^{n} a_i z^{-i}}{\sum_{i=0}^{n} b_i z^{-i}}.
   \]
Figure 6. Flow Diagram, Nth Order Transfer Function
The following relationships can now be derived:

\[
\frac{Y(z)}{M(z)} = \left( \sum_{i=0}^{n} a_i z^{-i} \right)
\]

and

\[
\frac{M(z)}{X(z)} = \frac{1}{\sum_{i=0}^{n} b_i z^{-i}}
\]

or

\[
\frac{X(z)}{M(z)} = \left( \sum_{i=0}^{n} b_i z^{-i} \right).
\]

Rearranging these equations provides expressions for the input \(X(z)\) and the output \(Y(z)\) in terms of \(M(z)\):

\[
Y(z) = \left( \sum_{i=0}^{n} a_i z^{-i} \right) M(z)
\]

and

\[
X(z) = \left( \sum_{i=0}^{n} b_i z^{-i} \right) M(z)
\]
Because the input is a known quantity, it is desirable to derive an equation for \( M(z) \) in terms of \( X(z) \). Manipulation of the previous equation for \( X(z) \) gives

\[
M(z) = X(z) - \sum_{i=1}^{n} b_i z^{-i} M(z)
\]

Because the calculations necessary to produce the filtering function are to be performed in real time, the inverse \( z \)-transform of \( M(z) \) and \( Y(z) \) is taken to provide a time domain solution of the first direct structure. The results are:

\[
m(k) = x(k) - \sum_{i=1}^{n} b_i m(k-i)
\]

and

\[
y(k) = \sum_{i=0}^{n} a_i m(k-i).
\]

The flow diagram of the first direct structure, of order two, is shown in Figure 7.

2. **Second Direct Structure**

The transpose of a digital filter structure is formed by reversing the signal flow in all of the branches of the block diagram. Its transfer function is the same as that of the structure from which it was derived. The second
Figure 7. Flow Diagram, First Direct Structure
direct structure is generated by applying this principle to the first direct structure. The corresponding time domain solution also implements the same filtering function, but an additional difference equation is required. The equations for the second direct structure are:

\[ p_i(k) = p_{i+1}(k-1) + a_i x(k) - b_i y(k); \quad i = 1, \ldots, n-1 \]

\[ p_n(k) = a_n x(k) - b_n y(k) \]

\[ y(k) = a_0 x(k) + p_i(k-1). \]

The second direct structure, of order two, is illustrated in the flow diagram of Figure 8.

3. **Third Direct Structure**

To obtain the third direct structure, the expression

\[ H(z) = \frac{Y(z)}{X(z)} = \left( \sum_{i=0}^{n} a_i z^{-i} \right) \left( \sum_{i=0}^{n} b_i z^{-i} \right) \]

is rewritten as

\[ Y(z) \left( \sum_{i=0}^{n} b_i z^{-i} \right) = X(z) \left( \sum_{i=0}^{n} a_i z^{-i} \right). \]

The output expression is then

\[ Y(z) = \sum_{i=0}^{n} a_i z^{-i} X(z) - \sum_{i=1}^{n} b_i z^{-i} Y(z). \]
Figure 8. Flow Diagram, Second Direct Structure
Once again taking the inverse z-transform, the time domain solution is derived:

\[ y(n) = \sum_{i=0}^{n} a_i x(n-i) - \sum_{i=1}^{n} b_i y(n-i). \]

Figure 9 illustrates the flow diagram for the third direct structure of order two.

4. **Fourth Direct Structure**

The fourth direct structure is derived by taking the transpose of the third direct structure. The time domain representation is:

\[
\begin{align*}
r_0(k) &= x(k) + r_{n-1}(k-1) \\
q_1(k) &= a_n r_0(k) \\
r_n(k) &= -b_n r_0(k) \\
q_i(k) &= a_i r_0(k) + q_{i+1}(k-1), \ i=1, \ldots, n-1 \\
r_i(k) &= -b_i r_0(k) + r_{i+1}(k-1).
\end{align*}
\]

The flow diagram for the fourth direct structure, of order two, is shown in Figure 10.

**B. SECOND ORDER SECTIONS**

Each of the structures described applies to the general case in which the filter transfer function is of order \( n \). The filtering function can be successfully realized using
Figure 9. Flow Diagram, Third Direct Structure
any of these methods, provided that \( n \) does not exceed two. For transfer functions of higher order, coefficient sensitivity becomes a significant factor in accurately generating the filter output. This phenomenon is created by the effects of quantization errors in the representation of the transfer function coefficients. The filter structures and their related difference equations assume finite precision in the filter parameters. However, because the word length used by microprocessors is fixed, the precision with which the filters can be realized is limited. This means that the coefficient representations are not exact, resulting in a set of poles and zeros for the realization that differ from those desired. The frequency response of the actual filter will therefore be different from the design specification. In the most extreme case, the filter will become unstable if one or more of the poles are moved outside of the unit circle [Ref. 11]. As shown by Rader and Gold [Ref. 12], high order filters can be redefined as combinations of first and second order sections in order to minimize these effects.

C. COMBINATIONAL STRUCTURES

1. **Cascade Structure**

The first combinational structure to be considered is the cascade form of the digital filter. It is illustrated in Figure 11(a) and is obtained by factoring the transfer
function into a product of second order sections. This is represented by [Ref. 13]

\[ H(z) = \prod_{i=1}^{M} H_i(z) \]

where

\[ H_i(z) = \frac{a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} \]

For the fourth order transfer function

\[ H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4}} \]

an equivalent representation is

\[ H(z) = \frac{a_{01} + a_{11} z^{-1} + a_{12} z^{-2}}{1 + b_{11} z^{-1} + b_{21} z^{-2}} \cdot \frac{a_{02} + a_{12} z^{-1} + a_{21} z^{-2}}{1 + b_{12} z^{-1} + b_{22} z^{-2}} \]

The first element in the cascade receives its input from the signal to be filtered, while each of the remaining elements acts upon the output of its predecessor.

In z-transform notation, the cascade structure output is written

\[ Y(z) = H_M(z)Y_{M-1}(z) \]
Figure 11. Block Diagram of Filter Structures
where
\[ Y_1(z) = H_1(z)X(z) \]
\[ Y_i(z) = H_i(z)Y_{i-1}(z) \text{ for } i=2,3,\ldots,M-1. \]

The equations necessary for a real time implementation are formed by taking the inverse z-transform of these expressions. For a system of second order modules, the general equation for each section is

\[ y(n) = a_{0M}y_{M-1}(n) + a_{1M}y_{M-1}(n-1) + a_{2M}y_{M-1}(n-2) \]
\[-b_{1M}y(n-1) - b_{2M}y(n-2) \]

where
\[ y_1(n) = a_{01}x(n) + a_{11}x(n-1) + a_{21}x(n-2) \]
\[-b_{11}y_1(n-1) - b_{21}y_1(n-2) \]
\[ y_i(n) = a_{1i}y_{i-1}(n) + a_{1i}y_{i-1}(n-1) + a_{2i}y_{i-1}(n-2) \]
\[-b_{1i}y_i(n-1) - b_{2i}y_i(n-2). \]

The subscript \( i \) corresponds to the number of modules needed to realize the desired filtering function, and is equal to \( \lfloor n/2 \rfloor \), where \( n \) is the order of the denominator polynomial. For the case where \( n \) is odd, one of the modules will be of order one.
2. **Parallel Structure**

The parallel form of the digital filter is the second structure to be considered, and is illustrated in Figure 11(b). It is formed by factoring the transfer function such that [Ref. 14]

\[ H(z) = \sum_{i=1}^{M} H_i(z) \]

where

\[ H(z) = \frac{a_{0i} + a_{1i} z^{-1}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} \]

For example, the fourth order transfer function

\[ H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4}} \]

can be redefined as

\[ H(z) = \frac{a_{01} + a_{11} z^{-1} + a_{12} z^{-2}}{1 + b_{11} z^{-1} + b_{21} z^{-2}} + \frac{a_{02} + a_{12} z^{-1} + a_{22} z^{-2}}{1 + b_{12} z^{-1} + b_{22} z^{-2}} \]

Each second order section receives the same input. The output of the system is found by summing the outputs of each of the parallel elements in the realization, and is represented by
\[ Y(z) = \sum_{i=1}^{M} H_i(z)X(z). \]

The corresponding time domain equation is

\[ y(n) = y_1(n) + y_2(n) + \ldots + y_i(n) \]

where

\[ y_1(n) = a_{01}x(n) + a_{11}x(n-1) - b_{11}y(n-1) - b_{21}y(n-2) \]

\[ y_2(n) = a_{02}x(n) + a_{12}x(n-1) - b_{21}y(n-1) - b_{22}y(n-2) \]

\[ y_i(n) = a_{0i}x(n) + a_{1i}x(n-1) - b_{ii}y(n-1) - b_{i2}y(n-2). \]

The equations presented for the parallel and cascade structures are sufficient to develop the high level problem description necessary to realize microprocessor-based digital filters using the design system described in the previous chapter.

D. DESIGN CONSIDERATIONS

The current implementation of the design system generates the control code governing execution of the realization using a polled monitor strategy. Pollack [Ref. 15] argued that this approach was not adequate for general controller applications and proposed the introduction of an interrupt driven monitor as a user selectable
alternative. The existing realization volume has the ability to recognize an interrupt, but its immediate response is limited to the setting of a corresponding condition flag. The status of the flag is examined and the appropriate task executed as required on a time schedule basis which conforms to the structure of the other tasks. The theoretical development necessary for implementing an interrupt driven monitor has been completed [Ref. 16] and will make possible the realization of a wider variety of control devices.

The polled monitor is the preferred strategy for digital filter implementations. Figure 12 is a flowchart representation of three operations basic to the filtering function. Each of these tasks is performed once for each sample taken. The interval of time which elapses between successive executions of each contingency/task pair is determined by the sampling interval required for the signal being processed. Only one execution is allowed during that time. Therefore, the generation of the filter function is periodic, with one output calculated each period of the sampling frequency. Because the polled monitor results in the execution of the contingency/task pairs at regularly determined intervals, it is well suited for applications in which periodicity is required.

1. **Sampling Frequency**

The Nyquist criterion determines the rate at which the signal to be processed must be sampled in order to
Figure 12. Flowchart, Basic Digital Filter Operations
accurately produce the corresponding filter output. The amount of time available to perform the required computations is then equal to the elapsed time between successive samples. This, in turn, is equal to the period of the sampling frequency. This means that the total time required to test for all contingencies and execute their corresponding tasks must be less than or equal to the sampling period in order to produce a successful single processor realization. If this requirement cannot be met, the design system will partition the problem into two sets of contingency/task pairs and attempt to generate a solution using two processors. In this case, the contingency/task pairs assigned to a given processing element must be able to complete execution during one period of the sampling frequency. The maximum frequency that can be filtered by a given realization is therefore limited by the speed with which the filtering algorithm can be executed, which in turn is a function of the microprocessor family used to generate the realization.

2. **Structure Selection**

The parallel and cascade structures are both readily adapted to microprocessor implementation. Because both methods are combinations of first and second order sections, it is a straightforward procedure to implement each section as a contingency/task pair. The natural partitioning that exists in each of these structures is well-suited to implementation using the automated design system and is
highly compatible with the partitioning performed for multi-processor realizations.

The selection criterion to be considered when choosing a filtering structure concerns the delay time incurred in producing an output from a specific input. The delay time will be a constant value for the parallel realization, independent of the number of second order modules required to perform the filtering operation. This will correspond to an interval equal to twice the period of the sampling frequency, assuming a multi-processor realization. The delay time associated with the cascade implementation is dependent upon the number of second and first order modules contained in the realization. The delay time will then be equal to the number of modules in the cascade times the period of the sampling frequency. Therefore, as the order of the filter increases, the number of second order modules required to implement it will increase, causing a corresponding rise in the filter delay time. These relationships are illustrated in Figure 13.

The effects related to ordering of the second order sections within a cascade structure are not of concern for the application presented in this thesis, but in practice, consideration must be given to the changes that can occur in the limit cycle and quantization noise properties of the filter as the second order modules are rearranged in the cascade. [Ref. 17]
a) Cascade Structure: Delay = Mt_1

b) Parallel Structure: Delay = t_1 + t_2

Figure 13. Delay Times of Filter Structures
IV. MODIFICATIONS TO THE SYSTEM

Applications problems are important to the development of the design system because each one possesses specific requirements for implementation. The testing of widely varying realizations will indicate those areas in which the system is deficient and identify hardware and software primitives that are required but absent from the available library. The digital filtering problem revealed several shortcomings in the current implementation of the design system.

A. PARALLEL PROCESSING

1. Single Contingency/Multiple Tasks

The parallel filter structure presented in chapter three consists of a finite number of elements that act concurrently on the same data. The output of each processor is then passed to a summing element which generates the filter output. In terms of the original problem model, this corresponds to a single contingency (the availability of data for processing) and multiple related tasks (each of the parallel processing elements and the summing junction). The design system implementation, as well as the specification language CSDL, provide no means for expressing such a problem directly.
2. FORK Construct

Representation of the single contingency/multiple task structure subject to the limitations of the design language can be solved by creating a new construct. This statement has been added to the syntax of CSDL and is called FORK. FORK allows the concurrent execution of two tasks followed by a third, which in turn requires data generated by the first two. The structure has the general form

```
If MAIN then begin
  Do FORK(TSK1, TSK2);
  Do TSK3;
End
```

which is stated in the syntax of CSDL as

```
When MAIN: TIME Do FORK(TSK1, TSK2, TSK3).
```

TSK1 and TSK2 are the procedures to be executed in parallel and are referred to as the "forked" tasks. Because it combines the results of TSK1 and TSK2 in a predetermined fashion, TSK3 is termed a "joined" task. The CSDL expression of the FORK construct is not sufficient to permit the use of the single contingency/multiple task structure in design specifications because no translator exists to produce the intermediate form of the instruction. However, it is possible to manually generate the required high level description of the FORK construct and from it determine the intermediate representation. To do so requires that a set of "dummy" contingencies corresponding to each of the tasks be generated. These contingencies test unique flags that are
set by an additional contingency/task pair which controls the execution of the composite FORK structure. Figure 14 is a general CSDL representation of the FORK construct.

3. Timing Requirements

The FORK construct makes possible the implementation of design specifications incorporating single contingency/multiple task algorithms. The determination of the timing constraints associated with the "dummy" contingencies it generates presents a new problem. There are two potential solutions for consideration. Each of these can be related to the single and multiple processor realizations that the design system is capable of generating. The former is the more straightforward derivation and will be presented first. The contingency and task names of Figure 14 are used to provide clarification.

Implicit in both developments is the assumption that the user will only specify the time constraint associated with the test for the contingency MAIN. The timing requirements for the dummy contingencies C1, C2, and C3 must be determined from this specification. The scheduling of the test for MAIN determines how often the concurrent processes must be tested. Therefore, the scheduling interval of this contingency will also be used for C1 and C2. This value will be denoted as ρ. The scheduling requirement for contingency C3 is related to the availability of data from tasks TSK1 and TSK2. The worst case condition, that is, the minimum
Contingency List:
  when MAIN:t1 SE Do FORK
  When C1 :t2 SE Do TSK1
  When C2 :t2 SE Do TSK2
  When C3 :t3 SE Do TSK3

Procedures:
  Function MAIN
    Binary MAIN,1;
    If GO=1 Then MAIN:=1;
    Exit MAIN
  Function C1
    Binary C1,1;
    If VAR1=1 Then C1:=1;
    VAR1:=0;
    Exit C1
  Function C2
    Binary C2,1;
    If VAR2=0 then C2:=1;
    VAR2:=0;
    Exit C2
  Function C3
    Binary C3,1;
    If DONE1=1 .and. DONE2=1 then C3:=1;
    DONE1:=0;
    DONE2:=0;
    Exit C3

Figure 14. CSDL Listing of FORK Construct
Task FORK
VAR1:=1;
VAR2:=1;
Exit FORK

Task TSK1
 (generate output, OUT1)
DONE1:=1;
Exit TSK1

Task TSK2
 (generate output, OUT2)
DONE2:=2;
Exit TSK2

Task TSK3
 OUT3:=OUT1+OUT2;
Exit TSK3

Figure 14. (continued)
interval at which C3 must be tested, exists when C1 and C2 are always true. If these contingencies are scheduled for testing every \( p \) seconds, new data will be generated by TSK1 and TSK2 with the same frequency. This implies that TSK3 must be executed at intervals of \( p \) seconds as well in order to maintain proper sequencing of the device. Therefore, the timing constraint specified for contingency MAIN will also be applicable to C1, C2, and C3. An important fact that must be considered in reaching this conclusion is that contingencies C1 and C2 can only occur as often as contingency MAIN. Similarly, C3 can only be true when C1 and C2 have been true as well. The timing diagram associated with the scheduling of the FORK construct for the single processor realization is illustrated in Figure 15.

The scheduling for the multiprocessor realization is not as readily determined. As in the serial case, contingencies C1 and C2 must be tested at intervals of \( p \) seconds, the timing specification of the MAIN/FORK contingency/task pair. The scheduling of C3 is made difficult by the possibility that each of the contingency/task pairs may reside in different processors and no synchronization exists between these elements. If C3 is tested with the same frequency as C1 and C2, it is possible to "lose" a set of data from TSK1 and TSK2. This occurs when these tasks generate data for use by TSK3 near the beginning of their respective executions and TSK3 processes the data near the end of its computations.
Figure 15. Timing Diagram of FORK Construct, Single Processor
If TSK1 and TSK2 begin another processing cycle and produce new data before TSK3 has completed its calculations on the previous set of data, its output will be determined using the second data set generated and the first values will never be processed. The timing diagram for such an occurrence is shown in Figure 16.

There are two possible solutions to prevent the loss of a data set. The first is to decrease the interval of testing for contingency C3. Checking for the presence of data from TSK1 and TSK2 with the correct frequency will ensure that it is processed by TSK3 almost immediately after it is generated. This solution results in a new question to be answered: how must of a decrease in the scheduling interval is sufficient to prevent data loss? The answer is not general in nature because the timing specification required will be dependent upon the particular application being considered. The incorporation of such a construct in the design system would require a more detailed analysis by the design engineer. This is contrary to the original goals of the system and is therefore unacceptable. An alternative approach which preserves application independence is to mark the data as it is produced by TSK1 and TSK2 and save it for further processing by TSK3. This implies the creation of a queue in which the data is placed to await action by TSK3. The implementation of this proposal requires two queues, one each for TSK1 and TSK2. Each time a task generates new data,
Figure 16. Data Loss in Multiprocessor FORK Construct Implementation
a counter associated with its queue is incremented to record the entry of a new element in the service line. The contingency C3 then tests the value of each counter. If both are greater than zero, the data that has been in each queue the longest is removed for processing. The process is illustrated in Figure 17.

The creation and maintenance of these data queues solves the problem of scheduling C3. The contingency need only be tested at intervals corresponding to $\rho$. If the queue counters are greater than zero, data has been produced and is ready for processing. The criteria upon which the existence of a true condition for contingency C3 is based has therefore been changed from the generation of output by TSK1 and TSK2 to the presence of data for processing in their respective queues. This approach to scheduling of the FORK construct does not necessitate the maintenance of separate algorithms for the serial and parallel case. The use of data queues to insure accurate output from a device is suitable for single processor realizations as well. Because the execution of contingency/task pairs is synchronized for such designs, data is processed by TSK3 following its generation by TSK1 and TSK2. The output of each of these tasks is stored in its respective queue and is then removed for processing by TSK3. Therefore, the number of individual entries in either queue will not exceed two. The CSDL specification of the modified FORD construct is shown in Figure 18.

62
Figure 17. Timing Diagram of FORK Construct, Multiple Processors
Contingency List:
  When MAIN: t1 SE Do FORK
  When C1 :t2 SE Do TSK1
  When C2 :t2 SE Do TSK2
  When C3 :t3 SE Do TSK3

Procedures:
  Contingency MAIN
    Binary MAIN,1;
    If GO=1 Then MAIN:=1;
    Exit MAIN

  Contingency C1
    Binary C1,1;
    If VAR1=1 Then C1:=1;
    VAR1:=0;
    Exit C1

  Contingency C2
    Binary C2,1;
    If VAR2=0 then C2:=1;
    VAR2:=0;
    Exit C2

  Contingency C3
    Binary C3,1;
    If CTR1.qt.0 .and. CTR2.qt.0 then C3:=1;
    Exit C3

Figure 18. CSDL Listing, Modified FORK Construct
Task FORK
VAR1:=1;
VAR2:=1;
Exit FORK

Task TSK1
(generate output, OUT1)
Q(12):=OUT1;
CTR1:=CTR1+1;
Exit TSK1

Task TSK2
(generate output, OUT2)
Q(22):=OUT2;
CTR2:=CTR2+1;
Exit TSK2

Task TSK3
Q(11):=Q(12);
Q(21):=Q(22);
OUT3:=Q(11)+Q(21);
CTR1:=CTR1-1;
CTR2:=CTR2-1;
Exit TSK3

Figure 18. (continued)
B. MULTIPLE REFERENCES TO I/O VARIABLES

1. Hardware Binding

The organization of the Intel 8080 realization volume as implemented in the current design system is based on the concept of hardware binding developed by Matelan [Ref. 18]. It specifies that all of the software required for a particular implementation must be generated first, followed by the hardware necessary to support it. The present version of the design program modifies this concept, binding the needed hardware to each individual software realization as it is generated. This results in the duplication of interface hardware each time a given external variable is referenced. When the design program encounters an input or output primitive, it first generates the necessary software followed by the corresponding support hardware. The variable name is not compared to previously generated I/O interfaces. Therefore, multiple sensing of an input or issuing of an output will cause the generation of an input/output port for each reference.

2. Symbol Table Listing of I/O Variables

To prevent redundant generation of interface hardware, a record of each I/O variable must be kept which stores each variable as it is defined by the user during the initial design specification. This can be done very effectively by establishing a symbol table which will maintain the status of each input/output variable and the hardware generated for it.
Rather than producing the hardware required from the software primitives representing each I/O operation, it will be generated after the user defines the external variables in the environment section of the design specification. This feature will be incorporated in the input module being developed for the system.

C. ANALOG TO DIGITAL INTERFACING

. Processor Controlled Conversions

The realization library available in the current implementation of the design system allows communications between dedicated microprocessor systems and external analog signals using a Burr-Brown ADC82AG eight bit analog to digital converter. Synchronization among contingency/task pairs is a critical factor in accurately generating digital filter output. Therefore, the sampled data input must be available from the a to d converter on a periodic basis as requested by the main processor. The converter available in the Intel 8080 realization volume is controlled by an independent clock and is therefore not suitable for such an application. The addition of an analog to digital converter suitable for digital filtering applications is therefore warranted.

The device chosen for inclusion in the realization library is the Analog Devices AD570 and is an appropriate selection for several reasons. The most significant of these
is its ability to perform a conversion when directed by an outside signal. Connections are provided for both a data convert signal and an associated output to indicate the completion of the conversion operation. The circuits required for interfacing the device to the Intel 8080 data bus are provided in the manufacturer's specification sheets [Ref. 19]. The device is also capable of accepting unipolar or bipolar data input.

2. Library Entry Development

The generation of the library entry for this device is a simple task. Because the use of hardware in any realization is controlled through software primitives, a new software entry was generated as well. This new primitive is called S.ANAIN. User specified parameters for the primitive are determined by those required for inclusion of the necessary hardware realizations. These consist of the name of the output signal, the maximum and minimum values that it can assume, and the number of bits in the digital representation. These parameters represent the data required for correct generation of the a to d converter hardware. The 3dB bandwidth of the input signal is also required in order to produce a buffer amplifier to the input of the device. The high voltage specification is used to determine the gain provided by the buffer amplifier. The interface hardware listed in the specification sheets is not required for
implementation by the design system. The inclusion of the primitive S.SENSECOND provides a suitable interface for the device.

The a to d converter is listed in the hardware primitive H.ADC2 and contains the pin connections necessary to implement the device. The low voltage limit is evaluated to determine if the interface for bipolar input is needed. The title line for both of these primitives is copied to the index of available realizations that is at the beginning of the volume. The attributes of time, storage requirements, and inclusions are listed as well. A complete listing of each primitive is contained in Figure 19 and 20, respectively.
s.anain (sig,h,1,b:0,8,50,-25,1,100:,,,3816,3847)
com primitive to define processor controlled analog input
com list = input,hv volt limit,lv volt limit,3db rolloff:
com bits,voltage limits,bw limits:
com time,stor,ext,calc,incl,addr
com added by m.r. heilstedt, may 1983
name $na001-$na030
begin stext
;analog input channel for signal <sig>,
;range <h> to <l> volts
;3db rolloff at <b> khz
endtext
incl h.conn-al (<$na006>,<$na007>,gnd,<sig>::)
com select gain for buffer amp to match input range
if <l> .lt. 0 skip 4
if <h> .le. 10 skip 6
if <h> .le. 25 skip 8
if <h> .le. 50 skip 10
com set buffer amp if bipolar output
if <l> .ge. -5 skip 3
if <l> .ge. -12 skip 5
if <l> .eq. -25 skip 7
com gain 1.0 (written 10) for input range 0,+10;+-5 volts
incl h.bufframp (<$na006>,<$na007>,<$na005>,10,<b>::)
skip 5
com gain 2.5 (written 25) for input range 0,+25;+-12.5 volts
incl h.bufframp (<$na006>,<$na007>,<$na005>,25,<b>::)
skip 2
com gain 5.0 (written 50) for input range 0,+50;+-25 volts
incl h.bufframp (<$na006>,<$na007>,<$na005>,50,<b>::)
incl h.adc2 (<$na005>,<sig>,<h>,<l>:8:)
call s.sensecond (<sig>:8,12A)
com

Figure 19. Analog Input Primitive
h.adc2 (in,out,h1:0,8:8,3848,3899)
com primitive to define 8 bit processor controlled adc
com list = input,output,hi volt limit, lo volt limit:
com bits:lat,pwr,chips,calc,incl,addr
com added by m. r. heilstedt, may 1983
name $na031-$na060
incl h.trimpot (gnd,<$na041>,<in>,200:)
begin htext
    a/d converter, 8 bit
device is analog devices ad570, ic <icn>
connections:
    pin 1 = n.c.
    pin 2 = <out>(0)  (1sb)
    pin 3 = <out>(1)  (2lsb)
    pin 4 = <out>(2)  (3lsb)
    pin 5 = <out>(3)  (4lsb)
    pin 6 = <out>(4)  (5lsb)
    pin 7 = <out>(5)  (6lsb)
    pin 8 = <out>(6)  (7lsb)
    pin 9 = <out>(7)  (msb)
    pin 10 = +5 volts
    pin 11 = conv  (blank and .not. convert)
    pin 12 = -15 volts
    pin 13 = <$na041>  (analog input)
    pin 14 = gnd    (analog common)
endtext
if <1> .lt. 0 skip 6
com if unipolar input, make direct connections
begin htext
    pin 15 = gnd    (bipolar offset)
    pin 16 = gnd    (digital common)
endtext
skip 11
begin htext
    pin 15 = <$na049>  (bipolar offset)
    pin 16 = <$na048>  (digital common)
endtext
incl h.nand4 (0,1,+5v,<$na048>,<$na045:>)
incl h.invert (<$na045>,<$na046:>)
incl h.diode-sw (<$na046>,<$na047:>)
incl h.diode-sw (<$na047>,<$na048:>)
incl h.diode-sw (<$na048>,<$na049:>)
incl h.resmfqtrwt(-15v,<$na048>,30000:)
begin htext
    pin 17 = dr    (data ready)
    pin 18 = n.c.
endtext
calc icn=icn+1

Figure 20. Analog to Digital Converter Primitive
V. DESIGN EXAMPLES

The ability of the design system to produce an acceptable realization is demonstrated using a realistic problem taken from the article by Nagle and Nelson [Ref. 20]. The implementation considered is a fourth order digital rate filter with the transfer function

\[ H(z) = \frac{1.0+0.390244z^{-1}-1.24247z^{-2}+0.344333z^{-3}+1.77044z^{-4}}{1.0-3.02828z^{-1}+3.53682z^{-2}-1.88867z^{-3}+0.397506z^{-4}} \]

This expression is redefined as a combination of second order modules from which the difference equations necessary to generate the filtering function in real time are derived. The choice of implementation algorithm for these modules is a difficult one because none of the four methods presented in chapter three is clearly superior. Each of the direct forms consists of the same number of multiplication and addition/subtraction operations. However, the direct form one algorithm possesses the minimum storage requirements of the four candidates and will therefore be used in the development of the example designs.
A. CASCADE REALIZATION

The cascade implementation of the proposed filter consists of n/2 or two second order sections. The equivalent expression is

\[
H(z) = H_1(z) \cdot H_2(z)
\]

\[
= \frac{1 + 1.7906z^{-1} + 1.2872z^{-2}}{1 + 1.823z^{-1} + 0.8959z^{-2}} \cdot \frac{1 - 2.188z^{-1} + 1.3832z^{-2}}{1 + 1.2054z^{-1} + 0.4438z^{-2}}
\]

1. CSDL Description

Despite the fact that the high level design language CSDL will not be implemented in the input specification module, it remains a useful tool in the current limited implementation of the system, providing a means of translating the problem statement into its intermediate form. Therefore, the first step in the development of the desired filter implementation is to express the problem in the syntax of CSDL.

The identification section of the listing contains all of the administrative data associated with the problem, such as the designer's name, date of creation, and project title. It is as follows:

IDENTIFICATION;  
Designer: M. R. Heilstedt  
Date: 4-7-83  
Project: Sample Cascade Realization

73
The environment section is more easily determined after the completion of the other subsections in the CSDL listing. It will therefore be developed last.

The generation of the contingency/task pairs is accomplished with the aid of the flowchart shown in Figure 21. Each of the decision diamonds represents a contingency to be tested, while the process boxes associated with them are their corresponding tasks. The contingency/task pairs and their order of specification in the contingency list are therefore determined by the sequence of execution indicated in the flowchart.

The first contingency/task pair determines when a sample of the input signal must be taken. Because this must be accomplished in accordance with the sampling theorem, the "dummy" contingency EVERY is used to force the generation of a sample once each period of the sampling frequency. Arbitrarily choosing a sampling interval of sixty milliseconds, the first pair is

EVERY 60MS do SAMPLE.

The second contingency/task pair tests for the availability of data from the analog to digital converter. Because the Analog Devices AD570 takes approximately twenty-five microseconds to perform a conversion, a test for the presence of data at its outputs is required. This is accomplished by sensing the value of the single bit input.
data ready. The variable DR is chosen to represent this control line. The value of this input is normally zero, but will change to logical one while the conversion process takes place. A return to zero indicates that the operation has been completed. A sample of the input signal is required at intervals determined by the period of the sampling frequency. The second contingency/task pair is therefore

When READY:60MS do READ.

The third contingency/task pair tests to insure that the reading of data has taken place and when true, executes the first filtering function of the cascade. The time constraint associated with this pair is once again equal to the period of the sampling frequency because data for processing is produced with that interval. The third contingency/task pair is then

When NEWDATA:60MS do FIL1.

The final contingency/task pair in the list tests for the generation of data by the first element in the cascade. When true, the second filtering function is executed. Because the contingency can be satisfied only as often as data is produced by the first filtering function, its time constraint is also equal to the sampling interval. The final contingency/task pair is

When ONEOUT:60MS do FIL2.
The procedures section of the CSDL listing requires a specification of the high level code required to test the contingencies and execute the tasks. The former shall be developed first.

The contingency for EVERY is a "dummy" specification, used to force the execution of a specified task at user defined intervals. As such, it has no high level listing.

The contingency READY senses the value of the input line DR. When DR is equal to zero, READY is set to one. Because the analog to digital converter will require twenty-five microseconds to complete the conversion operation, a fixed wait corresponding to this period is inserted in the contingency to guarantee that sufficient time for the conversion to take place prior to testing for its completion. The contingency is

Contingency READY
Binary READY,1;
Wait 25US;
Sense(DR);
If DR=0 then READY:=1;
Exit READY

The third contingency, NEWDATA, tests for the value of a flag, called REDDAT, which is set at the conclusion of the task READ. A value of one indicates that data has been read and is awaiting processing. The contingency is

Contingency NEWDATA
Binary NEWDATA,1;
If REDDAT=1 then NEWDATA:=1;
REDDAT:=0;
Exit NEWDATA
The final contingency, ONEOUT, tests for the availability of data from the first filtering function. When the value of the flag DONE1 is set to one at the end of task FIL1, data is ready for processing by task FIL2. The final contingency is

Contingency ONEOUT
  Binary oneout,l;
  If DONE1=1 then ONEOUT:=1;
  DONE1:=0;
  Exit ONEOUT

In addition to generating the filter output, the tasks also perform internal operations, such as the setting of flags, which are invisible to the user. The first task to be implemented is SAMPLE, which is used to produce the signal required to initiate a conversion by the analog to digital converter. This is done by setting the control signal, CONV, to zero, issuing it as an output, resetting its value to logical one, and issuing it again. The listing for the task is

Task SAMPLE
  CONV:=0;
  Issue(CONV);
  CONV:=1;
  Issue(CONV);
  Exit SAMPLE

Task READ senses the value of the input variable as determined by the analog to digital converter. It must also
set the contingency READY to zero and the data available flag, REDDAT, to one. The high level description of READ is

Task READ
READY:=0;
Sense(X);
REDDAT:=1;
Exit READ

The tasks that generate each of the filtering operations are similar in design and will therefore be developed together. FIL1 resets contingency NEWDATA to the false condition. The filtering algorithm is executed, with the result being stored in a global variable. The flag DONE1 is set to one to indicate the completion of the computation, and the intermediate variables used are updated for use in the next computation. FIL2 first sets contingency ONEOUT to zero. It performs the same general calculations using different coefficients, and issues the result of its computations as the system output variable Y. The task listings for FIL1 and FIL2 are

Task FIL1
NEWDATA:=0;
M11:=X-(1.823*ML2)-(0.8959*ML3);
Y1:=M11+(1.7906*ML2)+(1.2872*ML3);
DONE1:=1;
M13:=M12;
M12:=M11;
Exit FIL1
Task FIL2
ONEOUT:=0;
M21:=Y1-(1.2054*M22)-(0.4438*M23);
Y:=M21-(2.188*M22)+(1.3832*M23);
Issue(Y);
M23:=M22;
M22:=M21;
Exit FIL2

With the completion of the contingency and task specifications, the environment section can be defined. Each of the input and output variables must be listed first. They are represented by X and DR, and Y and CONV, respectively. The arithmetic variables are the global parameters used within the CSDL listing by multiple contingencies and tasks. Listed with each variable is a description of the type of signal it is and the number of bits required to represent it. The completed environment section is

ENCIRONMENT;
Input: X,8,TTL;DR,1,TTL;
Output: Y,8,TTL;CONV,1,TTL;
Arithmetic: Y1,24;Y2,24;M11,24;M13,24;
M21,24:M22,24;M31,24:M32,24;
M33,24;READY,1;DONE1,1;
NEWDATA,1;ONEOUT,1;
REDDAT,1;

2. Intermediate Representation

The generation of the list of primitives and the IADEFL file for the example specification is a line by line translation of the CSDL listing. The IADEFL file is derived from the identification section and the contingency list. It is produced by extracting the names of the contingency/task
pairs and their timing constraints from the former and listing them individually. In addition to the period of the contingency, the user may also specify the maximum time allowed for the execution of the contingency, the maximum allowed duration of the task, the global order of the contingency/task pair, its priority, and the maximum allowed time duration of any timed block within the contingency and its maximum allowed duration. The metric upon which the design is to be generated is not part of the CSDL specification but is included in the IADEFL file. The available criteria for design selection are: first successful realization produced, realization with least power requirements, and least costly realization. Because it is the only one implemented, the first metric was chosen. The design identification data follows the listing of the design criteria.

The procedures section of the CSDL listing contains the information needed to generate the primitive list specification of the design problem. Each line of the high level expression is translated into one or more lines of intermediate code. In order to manually generate the primitive problem representation, the user must have available a listing of the index to the realization volume used. Each of the entries in the intermediate specification is derived from the title lines of the available primitives. The process of translating the CSDL listing into its
intermediate form is tedious and repetitious and will not be detailed here. However, a short example is instructive. The line to be translated is a mathematical expression taken from the first filtering task, FIL1. The example statement is

\[ M11 := X - (1.823 \times M12) - (0.8959 \times M13) \]

The corresponding primitive code is produced by working from right to left in this statement. Each of the expressions in parentheses is derived first. The results of these operations are then combined as indicated by the remaining operations to attain the desired result. The constants must also be specified as variables within the listing. Each variable name used requires an additional statement for the generation of a storage location. The primitive list for this equation is

- \( \text{s.fmul} \) (mla,all,ml2:24,24,24)
- \( \text{s.fmul} \) (mlb,al2,ml3:24,24,24)
- \( \text{s.fsub} \) (ma,mla,mlb:24,24,24)
- \( \text{s.float} \) (sigin,insig:8)
- \( \text{s.fsub} \) (mll,sigin,ma:24,24,24)
- \( \text{s.var} \) (mla:24)
- \( \text{s.var} \) (ml2:24)
- \( \text{s.var} \) (mlb:24)
- \( \text{s.var} \) (ml3:24)
- \( \text{s.var} \) (ma:24)
- \( \text{s.var} \) (insig:24)
- \( \text{s.fcons} \) (all,0,0,0,0:24)
- \( \text{s.fcons} \) (al2,0,0,0,0:24)

The IADEFL file, primitive list, CSDL problem statement, and design system output are includes in Appendix A.
B. PARALLEL REALIZATION

The parallel realization of the fourth order digital rate filter also consists of two second order modules. The filtering function to be generated is found by taking the partial fraction expansion of the original transfer function. The resulting expression is

\[ H(z) = 1.0 - 8.0 \frac{1.2640z^{-1} - 1.7475z^{-1}}{1 - 1.8230z^{-1} + 0.8959z^{-2}} \]

\[ - 4.0 \frac{1.6734z^{-1} - 1.5692z^{-2}}{1 - 1.2053z^{-1} + 0.4437z^{-2}} \]

It is illustrated by the flowchart of Figure 22.

1. CSDL Description

The CSDL description of the parallel realization is similar to its cascade counterpart. Therefore, only those areas in which they differ are presented in detail. The identification section for the parallel realization is

IDENTIFICATION;
Designer: M. R. Heilstedt
Project: Sample Parallel Filter Problem
Date: 7 April 1983

The dummy contingency EVERY is once again used to force the sampling of the input signal according to the interval determined by the sampling frequency. The second contingency tests for the completion of the conversion and
executes the task which reads the digital data input. Each of these contingency/task pairs is identical to those of the same name in the cascade problem specification.

The remaining entries in the contingency list are unique to the parallel problem and are used to implement the FORK construct. The first of these tests for the availability of data for processing. A true condition executes the task FORK, which sets two variable flags, VAR1 and VAR2, to one. Contingencies C1 and C2 test the values of these flags. When true, C1 permits the execution of FIL1, the first of the parallel filtering functions. Similarly, the second filtering algorithm, FIL2, is executed when the contingency C2 is satisfied. FIL1 and FIL2 each place their data in individual queues and increment the value of their respective queue counters. Contingency C2 tests for the existence of each queue and if both are present, causes the execution of task PLUS. Once again choosing an arbitrary sampling interval of sixty milliseconds, the contingency list is

CONTINGENCY LIST;
EVERY 60 MS do SAMPLE
When DATA:60MS do READ
When LOOP:60MS do FORK
When C1 :60MS do FIL1
When C2 :60MS do FIL2
When C3 :60MS do PLUS

Using the terminology introduced in chapter four, FIL1 and FIL2 represent the forked tasks and PLUS is the joined task.
The algorithm for contingency DATA is identical to that used for its counterpart in the cascade implementation. Contingency LOOP tests the value of variable GO which is set to one by the execution of task READ. When the contingency is satisfied, it permits task FORK to be executed. The contingency is

Contingency LOOP
   Binary LOOP,l;
   If GO=1 then LOOP:=l;
   GO=0;
Exit LOOP

Contingencies C1 and C2 perform similar functions, using the variables VAR1 and VAR2, respectively. When true, C1 permits execution of FIL1 and C2 permits execution of FIL2. Both of the test variables, VAR1 and VAR2, are set by execution of task FORK. The contingencies are

Contingency C1
   Binary C1,l;
   If VAR1=1 then C1=1;
   VAR1=0;
Exit C1

Contingency C2
   Binary C2,l;
   If VAR2=1 then C2=1;
   VAR2=0;
Exit C2

Contingency C3 tests the value of the queue counters to determine if data generated by tasks FIL1 and FIL2 is
waiting to be processed. If both queues contain data entries, task PLUS is executed. The CSDL listing for C3 is

Contingency C3
  Binary C3,1;
  If CTR1.gt.0 .and. CTR2.gt.0 then C3:=1;
Exit C3

The tasks SAMPLE and READ are identical to the tasks already described for the parallel filter implementation problem.

Task FORK sets the flags tested by contingencies C1 and C2 to one to represent the true condition. It also resets the value of its associated contingency, LOOP, to zero. The task listing is

Task FORK
  LOOP:=0;
  VAR1:=1;
  VAR2:=1;
Exit FORK

It is important to note that the contingency/task pairs that follow cannot be satisfied if the LOOP/FORK pair is not true as well. Because the tasks that perform the filtering operations are included in this group, FORK effectively controls the execution of the filtering function.

The CSDL descriptions of tasks FIL1 and FIL2 are similar to those used in the cascade realization. The direct form one algorithm is again used to implement the filtering function. The results produced by each of the tasks are
placed in individual data queues and their associated
counters are incremented. The high level descriptions of
the tasks are

Task **FILL**
\[
\begin{align*}
C1 &= 0; \\
M11 &= X + (1.823 \times M12) - (0.896 \times M13) ; \\
Q1B &= (1.264 \times M11) - (1.748 \times M12) ; \\
M13 &= M12 ; \\
M12 &= M11 ; \\
CTR1 &= CTR1 + 1 ; \\
\text{Exit FIL1}
\end{align*}
\]

Task **FIL2**
\[
\begin{align*}
C2 &= 0; \\
M21 &= X + (1.205 \times M22) - (0.444 \times M23) ; \\
Q2B &= (1.673 \times M21) - (1.569 \times M22) ; \\
M23 &= M22 ; \\
M22 &= M21 ; \\
CTR2 &= CTR2 + 1 ; \\
\text{Exit FIL2}
\end{align*}
\]

Task **PLUS** adds the data generated by FIL1 and FIL2 to
produce the output of the device. The queue counters are
decremented following the output of the filtered signal. The
CSDL description of PLUS is

Task **PLUS**
\[
\begin{align*}
C3 &= 0; \\
Q1A &= Q1B ; \\
Q2A &= Q2B ; \\
Y &= 1.0 - (8.0 \times Q1A) - (4.0 \times Q2A) ; \\
\text{Issue}(Y) ; \\
CTR1 &= CTR1 - 1 ; \\
CTR2 &= CTR2 - 1 ; \\
\text{Exit PLUS}
\end{align*}
\]
Now that the contingencies and tasks for the parallel problem have been defined, the environment section entries can be determined.

ENVIRONMENT;
Input: X,8,TTL;DR,1,TTL;
Output: Y,8,TTL;CONV,1,TTL;
Arithmetic: Q1A,24;Q1B,24;Q2A,24;Q2B,24;
M11,24;M12,24;M13,24;M21,24;
M22,24;M23,24;GO,1;VAR1,8;
VAR2,8;LOOP,8;DATA,8;

2. Intermediate Representation

The IADEFL file and the primitive list are generated from the CSDL listing as detailed in the development of the input specification for the cascade problem. Appendix B contains these listings, as well as the complete CSDL specification of the problem and the output data generated by the design program.
VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The feasibility of automating the production of microprocessor-based digital filters has been demonstrated. The use of applications problems has been shown to be a logical and productive step in the development of the design system. The modifications and additions made as a result of this research have increased the versatility of the current version of the design program and identified areas for future study.

The value of the design system described and tested in this thesis is readily apparent. The time now required to manually produce the hardware and software necessary to support dedicated microprocessor systems can be more productively spent if the computer can be programmed to accomplish this for us. Further development of the design system is therefore warranted.

B. RECOMMENDATIONS

1. Implementation Language

The current version of the design program is written in FORTRAN. This language places rigid requirements on the format of the input data files used by the system and restricts the organization of the program code. Modularization
of the program algorithm into specific levels of operation is not possible. The use of a language such as Pascal, PL/1, or Ada would permit this type of program organization, allowing additions and modifications to be made more easily. Validation, verification, and testing of the design program would also be enhanced.

2. Validation of Current Program

The design program installed on the Digital Equipment VAX 11/780 was received on magnetic tape from Lawrence Livermore Laboratory. Because of a compatibility problem between the machine that produced the tape and that which read it, sporadic errors occurred in the copying of the tape onto the VAX. These errors consisted primarily of incorrect branch statements, but erroneous variable references were also found. These problems were of sufficient severity to prevent realizations from being generated when in fact they were possible. Therefore, a considerable amount of time was spent debugging the design program and in fact became a major portion of the effort to produce this thesis. A thorough validation of the program remains to be accomplished. That which has been done thus far has been performed to identify the source of observed errors in program execution. Due to the nature of the inconsistencies found it is reasonable to assume that those subroutines not checked contain errors as well. A line by line manual comparison of
the listing of the design program installed on the VAX 11/780 with a copy of the code that is known to be correct is the best method for determining the correctness of the current implementation.

3. Realization Library

The choice of implementation language for the design program will also determine the format of the entries in the realization volume. Its current organization makes additions and modifications a difficult task. Independent of the implementation language and library format is the need for an algorithm to allow library updates by the user. This type of program would save considerable time in the development of the hardware/software database.

The organization of the library merits further study. In keeping with the concept of hardware binding, the user specifies his problem in terms of software primitives. The support hardware necessary is automatically generated. Because the design algorithm is intended to search only one volume at a time for the needed primitives, duplication of hardware between volumes can occur. To prevent this redundancy, the creation of a global realization volume of commonly used hardware primitives would be helpful. After an unsuccessful search of the current microprocessor volume, the program would scan the global listing in an attempt to locate the needed primitive. Failure to produce a realization would occur only after an unsuccessful search of both the current
and global volumes. The content of the global volume would not be limited to modular hardware. Individual components, such as resistors and capacitors, could be included as well. The principle of hardware binding is not violated as long as user access to these entries is only allowed through software primitives.

The ultimate goal of the design system is to produce a physical hardware realization of the problem specification. In such a system, the current program would be the first module in a group of three or more that would generate the layout for the device program read only memory, with the monitor and design program, and assemble the hardware. The program code required to implement such a system would necessitate the listing of the software primitives in terms of microprocessor operational codes rather than assembly language. Therefore, the organization and incorporation of op code based volumes in the realization library must be investigated.

4. **Interrupt Driven Monitors**

The theory for the implementation of an interrupt-driven monitor has been developed but inclusion in the design system remains to be accomplished. The availability of such a strategy as a user selectable alternative will greatly increase the potential of the design system.
5. **Applications Problems**

More applications problems are required to determine the remaining shortcomings in the design system. Signal processing implementations involving modulation/demodulation techniques, as well as high speed special purpose hardware realizations, remain to be tested. The availability of the interrupt-driven monitor strategy can be expected to create additional test applications.

6. **Documentation**

The documentation currently available is inadequate to permit rapid familiarity with the program. Therefore, reference data must be maintained which details the development and implementation of the system. This will aid subsequent research and provide the basis for a user's manual when a commercially use of the design system is produced.
APPENDIX A
DATA, CASCADE REALIZATION

This appendix contains the input and output files for the cascade realization developed in chapter five.

A. INPUT DATA

1. CSDL Listing

"This is the CSDL description of a" "fourth order digital 'rate' filter. It is" "taken from the article by Nagle and" "Nelson which appeared in the February 1981 issue" "of COMPUTER magazine. It is implemented as" "the cascade of two second order sections, which" "are expressed using the direct form one algorithm."

IDENTIFICATION;
   Designer: M. R. Heilstedt
   Date: 4-7-83
   Project: Sample Cascade Filter Problem

ENVIRONMENT;
   Input: X, 8, TTL; DR, 1, TTL;
   Output: Y, 8, TTL; CONV, 1, TTL;
   Arithmetic: Y1, 8; Y2, 8; M11, 8; M13, 8; M21, 8; M22, 8;
            M31, 8; M32, 8; M33, 8; READY, 1;
            DONE1, 1;

CONTINGENCY LIST;
   "Sample signal every 60 milliseconds"
   EVERY 60ms do SAMPLE
   "Check for conversion completed"
   When READY:60ms do READ
   "Do first second order filtering"
   When NEWDATA:60ms do FIL1
   "Then do second filtering"
   When ONEOUT:60ms do FIL2
AUTOMATED DESIGN OF MICROPROCESSOR-BASED DIGITAL FILTERS (U)
NAVAL POSTGRADUATE SCHOOL
M R HEILSTEDT JUN 83

UNCLASSIFIED
PROCEDURES;
Contingency READY
  Binary READY,l;
  Wait 25US;
  Sense(DR);
  If DR=0 then READY:=1 fi;
Exit READY
Contingency NEWDATA
  Binary NEWDATA,l;
  If REDDAT=1 then NEWDATA:=1
  REDDAT:=0;
Exit NEWDATA
Contingency ONEOUT
  Binary ONEOUT,l;
  If DONE1=1 then ONEOUT:=1;
  DONE1:=0;
Exit ONEOUT
Task SAMPLE
  CONV:=0;
  Issue(CONV);
  CONV:=1;
  Issue(CONV);
Exit SAMPLE
Task READ
  READY:=0;
  Sense(X);
  REDDAT:=1;
Exit READ
Task FIL1
  NEWDATA:=0;
  M11:=X-(1.823*M12)-(0.8959*M13);
  Y1:=M11+(1.7906*M12)+(1.2872*M13);
  DONE1:=1;
  M13:=M12;
  M12:=M11;
Exit FIL1
Task FIL2
  ONEOUT:=0;
  M21:=Y1-(1.2054*M22)-(0.4438*M23);
  Y:=M21-(2.188*M22)+(1.3832*M23);
  Issue(Y);
  M23:=M22;
  M22:=M21;
Exit FIL2
2. **IADEFPL File**
3. Primitive Listing

- generated for:system

  - main (: :)
  - start (: :)

- generated for:each

  - every (each::)
  - var (each:8)

- generated for:data

  - proc (data::)
  - fixedwait (25::)
  - sensecond (dr:1,1)
  - seq (@t2,dr,ac01:8,1,1)
  - jmpf (@t2,a1001:8)
  - assigncons(data,1:1,1)
  - loc (@1001::)
  - exitproc (data,0::)
  - cons (@c001,0:1,1)
  - var (dr:1)
  - var (data:1)
  - var (@t2:8)

- generated for:newdata

  - proc (newdata::)
  - seq (@t5,reddat,ac002:8,1,1)
  - jmpf (@t3,a1002:8)
  - assigncons(reddat,0:1,1)
  - assigncons(newdata,1:1,1)
  - loc (@1002:8)
  - exitproc (newdata,0::)
  - cons (@c002,0:1,1)
  - var (newdata:1)
  - var (reddat:1)
  - var (@t3:8)

- generated for:oneout

  - proc (oneout::)
  - seq (@t4,done1,ac003:8,1,1)
  - jmpf (@t4,a1003:8)
  - assigncons(oneout,1:1,1)
  - assigncons(done1,0:1,1)
  - loc (@1003::)
  - exitproc (oneout,0::)
  - cons (@c003,1:1,1)
  - var (oneout:1)
  - var (done1:1)
  - var (@t4:8)

- generated for:sample

  - proc (sample::)
  - assigncons(conv,0:1,1)
s.issuecond (conv:1,2)
s.assigncons (conv:1,1,1)
s.issuecond (conv:1,2)
s.exitproc (sample,each::)
s.var (conv:1)

--- GENERATED FOR: READ ----------------------------------

s.proc (read::)
s.assigncons (data,0:1,1)
s.anain (x,10,10,5:8::)
s.assigncons (reddata,1:1,1)
s.exitproc (read::)
s.var (x:8)

--- GENERATED FOR: FILL ----------------------------------

s.proc (fill::)
s.assigncons (newdata,0:1,1)
s.fmul (m1b,b2,m13:24,24,24)
s.fmul (m1a,b1,m12:24,24,24)
s.fsub (m1a,m1b,m1b:24,24,24)
s.float (ix,x:8)
s.fsub (m11,ix,ma:24,24,24)
s.fmul (y1b,a2,m13:24,24,24)
s.fmul (y1a,a1,m12:24,24,24)
s.fadd (ya,y1a,ylb:24,24,24)
s.fadd (yl,y11,ya:24,24,24)

--- GENERATED FOR: FILL ----------------------------------

s.assigncons (done1,1:1,1)
s.fassign (m13,m12:24,24)
s.fassign (m12,m11:24,24)
s.exitproc (fill,newdata::)
s.var (m1b:24)
s.var (m13:24)
s.var (m1a:24)
s.var (m12:24)
s.var (ma:24)
s.var (ix:24)
s.var (ylb:24)
s.var (ya:24)
s.var (ya:24)
s.fcons (b2,0,168,114,64:24)
s.fcons (b1,0,170,114,64:24)
s.fcons (a2,0,192,124,64:24)
s.fcons (a1,0,22,228,64:24)

--- GENERATED FOR: FILL ----------------------------------

s.proc (fill2::)
s.assigncons (oneout,0:1,1)
s.fmul (m2b,m23,b22:24,24,24)
s.fmul (m2a,m22,b21:24,24,24)
s.fsub (m2a,m2a,m2b:24,24,24)
s.fsub (m2a,y1,y2a:24,24,24)
s.fmul (yba2,23,24,24,24)
s.fmul (yhr, a21, m22: 24, 24, 24)
s.fadd (yhr, yhr, yhr: 24, 24, 24)
s.fsub (y, m21, yhr: 24, 24, 24)
s.fix (iy, iy: 8)
s.anaout (iy, -10, 10: 8)
s.fassign (m23, m22: 24, 24)
s.fassign (m22, m21: 24, 24)
s.exitproc (fil2, oneout::)
s.var (m2b: 24)
s.var (m23: 24)
s.var (m2a: 24)
s.var (m22: 24)
s.var (ybr: 24)
s.var (ybr: 24)
s.var (iy: 8)
s.fcons (b22, 0, 168, 56, 64: 24)
s.fcons (b21, 0, 36, 205, 64: 24)
s.fcons (a22, 0, 8, 49, 64: 24)
s.fcons (a21, 0, 4, 198, 192: 24)
s.end (::)
SAMPLE CASCADE FILTER PROBLEM

org 65503  ;establish stack in first ram
@stak ds 32  ;define stack area
lxi sp, @stak+32 ;initialize stack pointer
org 64  ;begin code after reserved interrupt area
jmp @spvsl  ;force monitor execution
; dummy procedure for every-period type contingency
@each: nop  ;dummy function entry point
mvi a,1  ;force function value
sta each  ;to true value (1)
ret  ;return to monitor

org 65502  ;8 bit variable each in ram

@each: db 0
org 74

; procedure data
@data: nop  ;entry point for data
mvi a,0  ;wait 25 us
mvi a,2  ;load loop cnt
dcr c  ;delay
dcr a  ;decrement loop count
in 0  ;sense environmental data
sta dr  ;and store in dr
; if dr .eq. @c001 set @t2 true
lxi h, @t2  ;preset @t2
mvi m,1  ;to true (1)
lx dr  ;fetch first argument
lxi h, @c001  ;compare
cmp     m                 ; to second argument
jz      $+6               ; set @t2
lxi     h, @t2            ; to false (0)
mvi     m, 0               ; if args are not equal
lda     @t2               ; fetch @t2 and
cpi     0                 ; test for false (0)
jz      @1001              ; branch to @1001 if false
mvi     a, 1               ; assign constant 1
sta     data               ; to data
@1001:   nop               ; define location @1001
          ret               ; return to monitor, exit data
@c001:   db      0              ; 8 bit variable dr in ram
orq     65501             ; 8 bit variable data in ram
          dr: db 0
orq     130
orq     65500             ; 8 bit variable data in ram
          data: db 0
orq     130
orq     65499             ; 8 bit variable @t2 in ram
          @t2: db 0
orq     130
; procedure newdat
@newdat:   nop                ; entry point for newdat
            ; if reddat ,eq, @c002 set @t3 true
lxi     h, @t3            ; preset @t3
mvi     m, 1               ; to true (1)
lda     reddat            ; fetch first argument
lxi     h, @c002           ; compare
cmp     m                 ; to second argument
jz      $+6               ; set @t3
lxi     h, @t3            ; to false (0)
mvi     m, 0               ; if args are not equal
lda     @t3               ; fetch @t3 and
cpi     0                 ; test for false (0)
jz      @1002              ; branch to @1002 if false
mvi a, 0 ; assign constant 0
sta reddat ; to reddat
mvi a, 1 ; assign constant 1
sta newdat ; to newdat
@1002: nop ; define location @1002
   ret ; return to monitor, exit newdat
@c002: db 0
org 65498 ; 8 bit variable newdat in ram
newdat: db 0
org 177
org 65497 ; 8 bit variable reddat in ram
reddat: db 0
org 177
org 65496 ; 8 bit variable @t3 in ram
@t3: db 0
org 177
;
; procedure oneout
@oneout: nop ; entry point for oneout
   ; if done1, eq. @c003 set @t4 true
lxi h, @t4 ; preset @t4
mvi m,1 ; to true (1)
lda done1 ; fetch first argument
lxi h, @c003 ; compare
cmp m ; to second argument
jz $+6 ; set @t4
lxi h, @t4 ; to false (0)
mvi m,0 ; if args are not equal
lda @t4 ; fetch @t4 and
cli 0 ; test for false (0)
jz @1003 ; branch to @1003 if false
mvi a, 1 ; assign constant 1
sta oneout ; to oneout
mvi a, 0 ; assign constant 0
sta done1 ; to done1
@1003: nop ; define location @1003
; ret
; return to monitor, exit oneout
ac003: db 1
org 65495 ; 8 bit variable oneout in ram
oneout: db 0
org 224
org 65494 ; 8 bit variable done1 in ram
done1: db 0
org 224
org 65493 ; 8 bit variable dt4 in ram
dt4: db 0
org 224
;
; procedure sample
; entry point for sample
@sample: nop
mvi a, 0 ; assign constant 0
sta conv ; to conv
lda conv ; issue control
out 0 ; data through signal conv
mvi a, 1 ; assign constant 1
sta conv ; to conv
lda conv ; issue control
out 1 ; data through signal conv
ret ; return to monitor, exit sample
org 65492 ; 8 bit variable conv in ram
conv: db 0
org 251
;
; procedure read
; entry point for read
@read: nop
mvi a, 0 ; assign constant 0
sta data ; to data
; analog input channel for signal x, range -10 to 10 volts,
; 3db rolloff at 5 khz
in 1 ; sense environmental data
sta x ; and store in x
mvi a, 1 ; assign constant 1
sta reddat ; to reddat
ret ; return to monitor, exit read
org 65491 ; 8 bit variable x in ram
   x: db 0
org 273

; procedure fill
afill: nop ; entry point for fill
   mvi a, 0 ; assign constant 0
   sta newdat ; to newdat
; multiply f.p. b2 by m13 to get mlb
   lda b2 ; load arguments
   out 2
   lda b2+1
   out 2
   lda b2+2
   out 2
   lda b2+3
   out 2
   lda m13
   out 2
   lda m13+1
   out 2
   lda m13+2
   out 2
   lda m13+3
   out 2
   mvi a, 22b ; output command
   out 3
   in 2 ; unload result
   sta mlb+3
   in 2
   sta mlb+2
   in 2
   sta mlb+1
   in 2
sta mlb
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,7b ;identify error source
call error

;multiply f.p. b1 by m12 to get m1a
lda b1 ;load arguments
out 2
lda b1+1
out 2
lda b1+2
out 2
lda b1+3
out 2
lda m12
out 2
lda m12+1
out 2
lda m12+2
out 2
lda m12+3
out 2
mvi a,22b ;output command
out 3
in 2 ;unload result
sta m1a+3
in 2
sta m1a+2
in 2
sta m1a+1
in 2
sta m1a
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,7b ; identify error source
call error

; subtract f.p. mib from mla to get ma
lda mla ; load arguments
out 2
lda mla+1 out 2
lda mla+2 out 2
lda mla+3 out 2
lda m1b out 2
lda m1b+1 out 2
lda m1b+2 out 2
lda m1b+3 out 2
mvi a,21b ; output command
out 3
in 2 ; unload result
sta ma+3
in 2
sta ma+2
in 2
sta ma+1
in 2
sta ma
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,6b ; identify error source
call error
; convert integer x to floating point ix
lda x ; load argument lsbyte
out 2 ;
cpi 0 ; set flags
jp %8
mvi a,377b ; negative, extend sign
jmp %8
xra a ; positive, clear the msbyte
out 2 ; load the argument dummy msbyte
mvi a,35b ; output command
out 3
in 2
sta ix+3 ; unload result
in 2
sta ix+2
in 2
sta ix+1
in 2
sta ix
in 3 ; check error code
ani 36b ; mask the error code field
jz %8 ; no error, continue
push psw ; error, call error routine
mvi a,11b ; identify source of error

call error
; subtract f.p. ma from ix to get m11
lda ix ; load arguments
out 2
lda ix+1
out 2
lda ix+2
out 2
lda ix+3
out 2
lda ma
out 2
lda ma+1
out 2
lda ma+2
out 2
lda ma+3
out 2
mvi a,21b ;output command
out 3
in 2 ;unload result
sta m11+3
in 2
sta m11+2
in 2
sta m11+1
in 2
sta m11
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,6b ;identify error source
call error
;multiply f.p. a2 by m13 to get ylb
lda a2 ;load arguments
out 2
lda a2+1
out 2
lda a2+2
out 2
lda a2+3
out 2
lda m13
out 2
lda m13+1
out 2
1de m13+2  
out 2 m13+3  
mvi a,22b  
in 2 y1b+3  
out 2 y1b+2  
in 2 y1b+1  
out 2 y1b  
sta y1b  
sta y1b+2  
sta y1b+1  
sta y1b+3  
out 2 yla+1  
dec yla  
out 2 yla+2  
dec yla+3  
out 2 yla+4  
ida m12  
out 2 m12+1  
ida m12+2  
out 2 m12+3  
ida m12+4  
out 2 m12+5  
ida m12+6  
out 2 m12+7  
ida m12+8  
out 2 m12+9  
ida m12+10  
out 2 m12+11  
ida m12+12  
out 2 m12+13  
ida m12+14  
out 2 m12+15  
ida m12+16  
out 2 m12+17  
ida m12+18  
out 2 m12+19  
ida m12+20  
out 2 m12+21  
ida m12+22  
out 2 m12+23  
ida m12+24  
out 2 m12+25  
ida m12+26  
out 2 m12+27  
ida m12+28  
out 2 m12+29  
ida m12+30  
out 2 m12+31  
ida m12+32  
out 2 m12+33  
ida m12+34  
out 2
out 2
mvi a,22h ;output command
out 3
in 2 ;unload result
sta yla+3
in 2
sta yla+2
in 2
sta yla+1
in 2
sta yla
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,7b ;identify error source
call error
;add f.p. yla to ylb to get ya
lda yla ;load arguments
out 2
lda yla+1
out 2
lda yla+2
out 2
lda yla+3
out 2
lda ylb
out 2
lda ylb+1
out 2
lda ylb+2
out 2
lda ylb+3
out 2
mvi a,20h ;output command
out 3
sta ya+3
sta ya+2
sta ya+1
sta ya
sta ya 3
check for error status
and $+8$ 3bb; mask error code
jz $+8$ err
error, continue
push psw
err
error, call error routine
mvi a, 5b
identify error source
mvi a, 5b
err
error, call error routine
add f.p.
out 2
mvi a, 20b
out 3
sta y1+4
unload result
sta y1+3
unload arguments
sta y1+2
sta y1+1
sta y1
sta in
push psw
org 985
org 65467 ; floating point variable m1 in ram
   m1: ds 4
org 985
org 65463 ; floating point variable ix in ram
   ix: ds 4
org 985
org 65459 ; floating point variable ylb in ram
   ylb: ds 4
org 985
org 65455 ; floating point variable yla in ram
   yla: ds 4
org 985
org 65451 ; floating point variable ya in ram
   ya: ds 4
org 985
b2: db 0
   db 168
   db 114
   db 64
b1: db 0
   db 170
   db 114
   db 64
a2: db 0
   db 192
   db 164
   db 64
a1: db 0
   db 22
   db 228
   db 64
;
; procedure fil2
@fil2: nop ; entry point for fil2
   mvi a, 0 ; assign constant 0
sta oneout ; to oneout

; multiply f.p. m23 by b22 to get m2b
lda m23 ; load arguments
out 2
lda m23+1
out 2
lda m23+2
out 2
lda m23+3
out 2
lda b22
out 2
lda b22+1
out 2
lda b22+2
out 2
lda b22+3
out 2
mvi a,22b ; output command
out 3
in 2 ; unload result
sta m2b+3
in 2
sta m2b+2
in 2
sta m2b+1
in 2
sta m2b
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,7b ; identify error source
call error

; multiply f.p. m22 by b21 to get m2a
lda m22 ; load arguments
out 2
lda m22+1
out 2
lda m22+2
out 2
lda m22+3
out 2
lda b21
out 2
lda b21+1
out 2
lda b21+2
out 2
lda b21+3
out 2
mvi a,22b  ;output command
out 3
in 2  ;unload result
sta m2a+3
in 2
sta m2a+2
in 2
sta m2a+1
in 2
sta m2a
in 3  ;check for error status
ani 36h  ;mask error code
jz $+8  ;no error, continue
push psw  ;error, call error routine
mvi a,7b  ;identify error source
call error
;subtract f.p. m2b from m2a to get m2a
lda m2a  ;load arguments
out 2
lda m2a+1
out 2
lda m2a+2
out 2
lda m2a+3
out 2
lda m2b
out 2
lda m2b+1
out 2
lda m2b+2
out 2
lda m2b+3
out 2
mvi a,21b  ;output command
out 3
in 2  ;unload result
sta m2a+3
in 2
sta m2a+2
in 2
sta m2a+1
in 2
sta m2a
in 3  ;check for error status
ani 36b  ;mask error code
jz $+8  ;no error, continue
push psw  ;error, call error routine
mvi a,6b  ;identify error source
call error
;subtract f.p. m2a from y1 to get m21
lda y1  ;load arguments
out 2
lda y1+1
out 2
lda y1+2
out 2
lda y1+3
out 2
lda m2a
out 2
lda m2a+1
out 2
lda m2a+2
out 2
lda m2a+3
out 2
mvi a, 21b ; output command
out 3
in 2 ; unload result
sta m21+3
in 2
sta m21+2
in 2
sta m21+1
in 2
sta m21
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a, 6b ; identify error source
call error
; multiply f.p. a22 by m23 to get yb
lda a22 ; load arguments
out 2
lda a22+1
out 2
lda a22+2
out 2
lda a22+3
out 2
lda m23
out 2
lda m23+1
out 2
lda m23+2
out 2
lda m23+3
out 2
mvi a, 22h ;output command
out 3
in 2 ;unload result
sta yb+3
in 2
sta yb+2
in 2
sta yb+1
in 2
sta yb
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a, 7b ;identify error source
call error
;multiply f,p, a21 by m22 to get yh
lda a21 ;load arguments
out 2
lda a21+1
out 2
lda a21+2
out 2
lda a21+3
out 2
lda m22
out 2
lda m22+1
out 2
lda m22+2
out 2
lda m22+3
out 2
mvi a,22b ;output command
out 3
in 2 ;unload result
sta yh+3
in 2
sta yh+2
in 2
sta yh+1
in 2
sta yh
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,7b ;identify error source
call error

;add f.p. yh to yb to get yh
lda yh ;load arguments
out 2
lda yh+1
out 2
lda yh+2
out 2
lda yh+3
out 2
lda yb
out 2
lda yb+1
out 2
lda yb+2
out 2
lda yb+3
out 2
mvi a,20b  ; output command
out 3
in 2   ; unload result
sta yh+3
in 2
sta yh+2
in 2
sta yh+1
in 2
sta yh
in 3   ; check for error status
ani 36b  ; mask error code
jz $+8  ; no error, continue
push psw  ; error, call error routine
mvi a,5b  ; identify error source
call error
; subtract f.p. yh from m21 to get y
lda m21  ; load arguments
out 2
lda m21+1
out 2
lda m21+2
out 2
lda m21+3
out 2
lda yh
out 2
lda yh+1
out 2
lda yh+2
out 2
lda yh+3
out 2
mvi a,21b  ; output command
out 3
in 2   ; unload result
sta y+3
in 2
sta y+2
in 2
sta y+1
in 2
sta y
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,6b ; identify error source
call error
; convert floating bit y to integer iy
lda y ; load arguments
out 2
lda y+1
out 2
lda y+2
out 2
lda y+3
out 2
mvi a,37b ; load command
out 3
in 2 ; unload result
mov b,a
in 2
sta iy
call dstst
jz $+8 ; no error detected
push psw ; error, call the error routine
mvi a,32b
push psw ; send a code to indicate source of error
call error
in 3 ; check chip for error status
ani 36b
4. 

Call error 

analog output channel for signal iv, range -10 to 10 

out x70 

x70 

issue control 

assign value of fp.m21 to m22 

assign value of fp.m22 to m23 

return to monitor, exit file 

floating point variable m23 in ram 

floating point variable m22 in ram 

floating point variable yh in ram 

floating point variable yh in ram 

floating point variable yh in ram 

floating point variable yh in ram 

org 6547 

org 65439 

org 65435 

org 1717 

org 1717 

org 1717 

org 1717 

org 1717 

org 1717 

org 65423
dstst: mov c,a
mov a,b
cri 0 ; set flags
jp dispo ; test for pos
cpi -1b ; test for -1
jnz dsufl ; underflow detected
mov a,b ; ok so far, test sign of lsbyte
cpi 0 ;
ji dsufl ; underflow by sign error
ret ; ok, return

; dispo: jnz dsol ; + overflow
mov a,c ; zero msbyte, test sign of lsbyte
cpi 0 ;
ji dsol ; sign error overflow
ret ; ok, return
dsul: mvi a,4b ; underflow, return error code
ret
dsol: mvi a,2b ; underflow, return error code
ret

this realization consumes 9.110 watts of power
and contains 18 chips.
central processing unit
device: intel 8080 8-bit microprocessor, ic 1
connections:
pins 25, 26, 27, 29, 30, 31, 32, 33,
        34, 35, 1, 40, 37, 38, 39, 36 = a(0:15)
pins 10, 9, 8, 7, 3, 4, 5, 6, = d(0:7)
pin 21 = hlda
pin 12 = reset
pin 13 = hold
pin 14 = int
pin 16 = intc
pin 17 = dbin
pin 18 = wr-bar
pin 19 = sync
pin 22 = phi1
pin 15 = phi2
pin 23 = ready
pin 24 = wait
pin 2 = gnd
pin 20 = +5v
pin 11 = -5v
pin 28 = +12v
clock generator (0.5 us)
device: intel 8224 clock generator and driver for 8080 cpu, ic 2
connections:
  pin 1 (reset) = reset
  pin 4 (ready) = ready
  pin 5 (sync) = sync
  pin 10 (phi2) = phi2
  pin 11 (phi1) = phi1
  pin 14, 15 (xtal1(1:2)) = device: 18 mhz crystal
  pin 16 (vcc) = +5v
  pin 9 (vdd) = +12v
  pin 8 (gnd) = gnd
d(0:7) = db(0:7)
cpu status latch
y: ds 4
org 1717
org 65422 ; 8 bit variable iy in ram
iy: db 0
org 1717
b22: db 0
    db 168
    db 56
    db 64
b21: db 0
    db 36
    db 205
    db 64
a22: db 0
    db 8
    db 49
    db 64
a21: db 0
    db 4
    db 198
    db 192
end
; software complete

; =monitor section=
; @spvrs: lhld @table ; initialize table pointer
    shl @pntr ; to beginning
@mlop : lhld @pntr ; main loop: get pntr
    inx h ; increment
    inx h ; the pointer by 3
    inx h ; (bytes to bypass jmp)
    shl @pntr ; store pointer address for table
    pchl ; begin execution at current entry

; data section
    org 65421
@pntr:   dw 0 ; table entry address pointer
org 1748

@table: dw @ptrntr ; table header (define top)
jmp @teach ; test for contingency each
jmp @data ; test for contingency data
jmp @newdat ; test for contingency newdat
jmp @oneout ; test for contingency oneout
jmp @oneout ; test for contingency oneout
        ; go to start of table
;
@teach:call @teach ; execute contingency code each
lda each ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @sample ; execute task sample if true
jmp @mlop ; return to monitor
;
@data:call @data ; execute contingency code data
lda data ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @read ; execute task read if true
jmp @mlop ; return to monitor
;
@newdat:call @newdat ; execute contingency code newdat
lda newdat ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @fill ; execute task fill if true
jmp @mlop ; return to monitor
;
@oneout:call @oneout ; execute contingency code oneout
lda oneout ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @fill2 ; execute task fill2 if true
jmp @mlop ; return to monitor
;
; routine to test if double length conversion to single length
; format has caused overflow or underflow
; enter with msbyte in b, lsbyte in a
device: int 8212 8-bit i/o port, ic 3
connections:
  pins 3,5,7,9,16,18,20,22 (di (0:7)) = d(0:7)
  pin 4 (do(1)) = inta
  pin 6 (do(2)) = wo-bar
  pin 8 (do(3)) = stacã
  pin 10 (do(4)) = hltã
  pin 15 (do(5)) = out
  pin 17 (do(6)) = m1
  pin 19 (do(7)) = inp
  pin 21 (do(8)) = memr

condition-mode input interface hardware to sense signal dr
device: int 8212 8-bit i/o port, ic 4
connections:
  pins 3,5,7,9,16,18,20,22 (di (1:8)) = dr(1:8), remainder to gnd
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = db(1:8)
  pin 2 (md) = gnd
  pin 11 (stb) = qnd
  pin 1 (ds1-bar) = .not. (decode a(0:7) value 0)
  pin 13 (ds2) = inp .and. dbin
  pin 24 (vcc) = +5v
  pin 12 (gnd) = qnd

condition-mode output interface hardware to issue signal: conv
device: int 8212 8-bit i/o port, ic 5
connections:
  pins 3,5,7,9,16,18,20,22 (di (1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = conv(1:8); if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = qnd
  pin 1 (ds1-bar) = wr-bar
  pin 13 (ds2) = out .and. (decode a(0:7) value 0)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = qnd

condition-mode output interface hardware to issue signal: conv
device: int 8212 8-bit i/o port, ic 6
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = conv(1:8) if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 1)
pin 24 (vcc) = +5v
pin 12 (qnd) = qnd
connector j 1, for analog signal x,
16 pin dip socket
connections:
pin 1 = x6
pin 2 = x7
pin 3 = gnd grounded at signal source only
resistor r 1, 10000 ohms, 1/4 watt 1% metal film
pin 1 = x6
pin 2 = x1
resistor r 2, 10000 ohms, 1/4 watt 1% metal film
pin 1 = x7
pin 2 = x2
resistor r 3, 10000 ohms, 1/4 watt 1% metal film
pin 1 = x1
pin 2 = x5
resistor r 4, 10000 ohms, 1/4 watt 1% metal film
pin 1 = x2
pin 2 = gnd
capacitor, c 1, ceramic, 3200 pf
connections:
pin 1 = x1
pin 2 = x5
trimpot, r 5, 10000 ohms, 22t 1/2w cermet
pin 1 = x3
pin 2 = -15v (wiper)
pin 3 = x4 (cw end)
op amp, ic 7
device is analog devices ad741k
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = x(1:8), remainder to gnd
pins 4,6,8,10,15,17,19,21 (do(1:8) = db(1:8)

pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = .not. (decode a(0:7) value 1)
pin 13 (ds2) = inp .and. dbin
pin 24 (vcc) = +5v
pin 12 (qnd) = qnd

Math processor chip, ic10
Type is Intel c8231

connections:
  pin 1 = qnd (vss
  pin 2 = +5v (vcc
  pin 3 = n.c. (/eack)
  pin 4 = n.c. (/svack)
  pin 5 = n.c. (svreq)
  pin 6 = n.c. (reserved)
  pin 7 = n.c. (reserved)
  pin 8 = db0
  pin 9 = db1
  pin 10 = db2
  pin 11 = db3
  pin 12 = db4
  pin 13 = db5
  pin 14 = db6
  pin 15 = db7
  pin 16 = +12v (vdd
  pin 17 = ready
  pin 18 = .not. (decode(a1-a7);value 2) (cs)
  pin 19 = .not. iow (wbar)
  pin 20 = .not ior (rbar)
  pin 21 = a0
  pin 22 = reset
  pin 23 = ph2 (clock)
  pin 24 = n.c. (/end)
connections:

pin 1 = x3  (zero trimpot)
2 = x1  (negative input)
3 = x2  (positive input)
4 = -15v
5 = x4  (zero trimpot)
6 = x5  (output)
7 = +15v
8 = n.c.  tab at pin 8

trimpot, r 6, 200 ohms, 22t 1/2w cermet
pin 1 = gnd
pin 2 = x5  (wiper)
pin 3 = x41  (cw end)

a/d converter, 8 bit

device is analog devices ad570, ic 8
connections:

pin 1 = n.c.
pin 2 = x(0)  (lsb)
pin 3 = x(1)  (2lsb)
pin 4 = x(2)  (3lsb)
pin 5 = x(3)  (4lsb)
pin 6 = x(4)  (5lsb)
pin 7 = x(5)  (6lsb)
pin 8 = x(6)  (7lsb)
pin 9 = x(7)  (msb)
pin 10 = +5 volts
pin 11 = conv  (blank and not, convert)
pin 12 = -15 volts
pin 13 = x41  (analog input)
pin 14 = gnd  (analog common)
pin 15 = gnd  (bipolar offset)
pin 16 = gnd  (digital common)
pin 17 = dr  (data ready)
pin 18 = n.c.

condition-mode input interface hardware to sense signal x
device: intel 8212 8-bit i/o port, ic 9
connector j 2, for analog signal iy,
16 pin dip socket
connections:
  pin 1 = x70
  pin 2 = gnd
  pin 3 = open grounded at signal source only
voltage follower, ic11
device is lm310
connections:
  pin 1 = n.c. (balance)
  2 = n.c.
  3 = x61 ;(input)
  4 = -15v (v-)
  5 = n.c. (booster)
  6 = iy ;(output)
  7 = +15v (v+)
  8 = n.c. (balance, tab at pin 8)
8 bit dac, ic 12
device is burr-brown dac82, laser trimmed, no adj riqd.
connections:
  pin 3 = +15v
  pin 4 = x62(7)
  pin 5 = x62(6)
  pin 6 = x62(5)
  pin 7 = x62(4)
  pin 8 = x62(3)
  pin 9 = x62(2)
  pin 10 = x62(1)
  pin 11 = x62(0)
  pin 12 jumper to pin 15 to use internal current ref
  pin 13 = +15v
  pin 14 = gnd
range dependent connections for 5000 to -5000 mv range
  pin 1 = gnd
  pin 2 = x61 ;(output)
  pin 16 jumper to pin 18
pin 17 jumper to pin 2

**TTL inverter, element 1 of IC 13, 7404**

- pin 1 = iy(1) ; (input)
- pin 2 = x62 ; (output)
- pin 7 = gnd
- pin 14 = +5V

**TTL inverter, element 2 of IC 13, 7404**

- pin 3 = x71 ; (input)
- pin 4 = x63 ; (output)

**TTL inverter, element 3 of IC 13, 7404**

- pin 5 = x72 ; (input)
- pin 6 = x64 ; (output)

**TTL inverter, element 4 of IC 13, 7404**

- pin 9 = x73 ; (input)
- pin 8 = x65 ; (output)

**TTL inverter, element 5 of IC 13, 7404**

- pin 11 = x74 ; (input)
- pin 10 = x66 ; (output)

**TTL inverter, element 6 of IC 13, 7404**

- pin 13 = x75 ; (input)
- pin 12 = x67 ; (output)

**TTL inverter, element 1 of IC 14, 7404**

- pin 1 = x76 ; (input)
- pin 2 = x68 ; (output)
- pin 7 = gnd
- pin 14 = +5V

**TTL inverter, element 2 of IC 14, 7404**

- pin 3 = x77 ; (input)
- pin 4 = x69 ; (output)

**Condition-mode output interface hardware to issue signal: x70**

**Device:** Intel 8212 8-bit I/O port, IC 15

**Connections:**

- pins 3, 5, 7, 9, 16, 18, 20, 22 (DI(1:8)) = DB(1:8)
- pins 4, 6, 8, 10, 15, 17, 19, 21 (DO(1:8)) = x70(1:8) if 8 are req
- pin 2 (MD) = +5V
- pin 11 (STB) = GND
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 5)
pin 24 (vcc) = +5v
pin 12 (qnd) = qnd

system memory
time delay to match 8111-2 latency to 8080
device: texas instruments sn74175 hex-quad d-type flip-flop with clear
ic 16
connection:
  pin 9 (clock) = .not. phi2
  pin 1 (clear) = .not. sync
  pin 16 (vcc) = +5v
  pin 8 (qnd) = qnd
  pin 2 (1q) = q
  pin 4 (1d) = +5v
  ready = q .and. .not. memr

16k eprom, ic 17, 8 pages of 256 words, starting at page 0
device is 2716 (do not use ti -- different pinout)

c134
connections:
  pin 1 = a(8)
  pin 2 = a(7)
  pin 3 = a(6)
  pin 4 = a(5)
  pin 5 = a(4)
  pin 6 = a(3)
  pin 7 = a(2)
  pin 8 = a(1)
  pin 9 = db(1)
  pin 10 = db(2)
  pin 11 = db(3)
  pin 12 = qnd
  pin 13 = db(4)
  pin 14 = db(5)
  pin 15 = db(6)
  pin 16 = db(7)
  pin 17 = db(8)
pin 18 = grd  (pd/pgm)
pin 19 = a(11)
pin 20 = .not. (decode(a(12:16) value 0) .and. .not. rdbar)
pin 21 = +5v  (vpp)
pin 22 = a(10)
pin 23 = a(9)
pin 24 = +5v

random access memory (lower half of page 255)
device: intel 8111-2 1024 bit (256x4) static mos ram, ic 18
connections:
pins4,3,2,1,17,5,6,7 (a(0:7)) =a(0:7)
pin 9 (od) = dbin
pin 16 (rw) = rw
pin 15 (ce1-bar) = ce
pin 10 (ce2-bar) = gnd
pins 11,12,13,14 (io(1:4)) = db(0:3)
ce = (.not. (out .and. wr-bar) .or. memr)
      .and. ( .not. (decode a(8:15) value 255))
pin 18 (vcc) = +5v
pin 8 (gnd) = qnd

random access memory (upper half of page 255)
device: intel 8111-2 1024 bit (256x4) static mos ram, ic 19
connections:
pins 4,3,2,1,17,5,6,7 (a(0:7)) = a(0:7)
pin 9 (od) = dbin
pin 16 (rw) = rw
pin 15 (ce1-bar) = ce
pin 10 (ce2-bar) = gnd
pins 11,12,13,14 (io(1:4)) = db (4:7)
pin 18 (vcc) = +5v
pin 8 (gnd) = qnd
APPENDIX B
DATA, PARALLEL REALIZATION

This is the input and output data for the parallel filter problem developed in chapter five. The timing constraints were increased to ninety milliseconds to permit successful generation of a realization. The design program attempted to generate a dual processor realization for the original specification, but a system error prevented completion of the output.

A. INPUT DATA

1. CSDL Listing

"This is a parallel realization"
"of the fourth order rate filter"
"as presented in the article"
"by Nagle and Nelson in the"
"February 1981 issued of IEEE COMPUTER."

IDENTIFICATION:
Designer: M. R. Heilstedt
Date: 7 April 1983
Project: Sample Parallel Filter Problem

ENVIRONMENT;
Input:X,8,TTL;DR,1,TTL;
Output:Y,8,TTL;CONV,1,TTL;
Arithmetic:Yl,8;Mll,8;Ml2,8;Ml3,8;
QlA,24;Q2A,24;QlB,24;Q2B,24;M21,8;M22,8;M23,8;
GO,1;VARl,1;VAR2,1;CTRl,1;CTR2,1;

CONTINGENCY LIST:
"Initiate sequence every 60 milliseconds:
EVERY 60ms do SETFLG
"Sample Signal once each period"
EVERY 60ms do SAMPLE
"Check for data available after sample converted"
When DATA:60ms do READ
"Set flags for parallel operations when ready"
When LOOP:60ms do FORK
"First parallel filter function"
When C1:60ms do FIL1
"Second parallel filter function"
When C2:60ms do FIL2
"Add results of each function and issue sum"
When C3:60ms do PLUS

PROCEDURES;
Contingency DATA
    Binary DATA,l;
    Sense (DR);
    If DR=0 then DATA:=1;
Exit DATA

Contingency LOOP
    Binary LOOP,l;
    If GO=1 then LOOP:=1;
    GO=0;
Exit LOOP

Contingency C1
    Binary C1,l;
    If VAR1=1 then C1:=1;
    VAR1:=0;
Exit C1

Contingency C2
    Binary C2,l;
    If VAR2=1 then C2:=1;
    VAR2:=0;
Exit C2

Function C3
    Binary C3,l;
    If CTR1.gt.0 .and. CTR2.gt.0 then C3:=1;
Exit C3

Task SAMPLE
    CONV:=0;
    Issue(CONV);
    CONV:=1;
    Issue(CONV);
Exit SAMPLE
Task READ
    DATA:=0;
    Sense(X);
    GO:=1;
Exit FORK

Task FILL
    Cl:=0;
    M11:=X+(1.823*M12)-(0.896*M13);
    Q1B:=(1.264*M11)-(1.748*M12);
    M13:=M12;
    M12:=M11;
    CTR1:=CTR1+1;
Exit FILL

Task FIL2
    C2L:=0;
    M21:=X+(1.205*M22)-(1.444*M23);
    Q2B:=(1.673*M21)-(1.569*M22);
    M23:=M22;
    M22:=M21;
    CTR2:=CTR2+1;
Exit FIL2

Task PLUS
    C3:=0;
    Q1A:=Q1B;
    Y:=1.0-(8.0*Q1A)-(4.0*Q2A);
    Issue(Y);
    CTR1:=CTR1-1;
    CTR2:=CTR2-1;
Exit PLUS
2. IADEFL File

```
A 001 :system :ms: 90, 90, 90, 1, 1, 0, 0, 0
A 002 :each :sample :ms: 90, 90, 90, 1, 1, 0, 0, 0
A 003 :data :read :ms: 90, 90, 90, 1, 1, 0, 0, 0
A 004 :loop :fork :ms: 90, 90, 90, 1, 1, 0, 0, 0
A 005 :c1 :fill1 :ms: 90, 90, 90, 1, 1, 0, 0, 0
A 006 :c2 :fill2 :ms: 90, 90, 90, 1, 1, 0, 0, 0
A 007 :c3 :plus :ms: 90, 90, 90, 1, 1, 0, 0, 0
D:frst:1,2,3:1,2,3:

i
i SAMPLE PARALLEL FILTER PROBLEM
i
```
3. Primitive Listing

- t. generated for:system
- s.main
- s.start
- t. generated for:each
- s.every
- s.var
- t. generated for:data
- s.proc
- s.fixedwait
- s.sensecond
- s.eq
- s.jmpf
- s.assigncons
- s.loc
- s.exitproc
- s.cons
- s.var
- s.var
- s.var
- t. generated for:loop
- s.proc
- s.eq
- s.jmpf
- s.assigncons
- s.loc
- s.exitproc
- s.cons
- s.var
- s.var
- s.var
- t. generated for:c1
- s.proc
- s.eq
- s.jmpf
- s.assigncons
- s.loc
- s.exitproc
- s.cons
- s.var
- s.var
- t. generated for:c2
- s.proc
- s.eq
- s.jmpf
- s.assigncons
s.assigncons(var2,0:1,1)
s.loc (@1004:)
s.exitproc (c2,0:)
s.cons (@c004,1:1,1)
s.var (@t5:8)
s.var (var2:1)
s.var (c2:1)

s.exe
t. generated for: c3

s.proc (c3:)
s.quot (@t6,ctr1,@c005:8,8,8)
s.jmpf (@t6,@1005:8)
s.quot (@t6,ctr2,@c005:8,8,8)
s.jmpf (@t6,@1005:8)
s.assigncons(c3,1:1,1)
s.sub (ctr1,ctr1,dec:8,8,8)
s.sub (ctr2,ctr2,dec:8,8,8)
s.loc (@1005:)
s.exitproc (c3,0:)
s.cons (@c005,0:8)
s.var (@t6:8)
s.var (c3:1)
s.var (ctr1:8)
s.var (ctr2:8)
s.cons (dec,1:8:)
t. generated for: sample

s.proc (sample:)
s.assigncons(conv,0:1,1)
s.issuecond (conv:1,2)
s.assigncons(conv,1:1,1)
s.issuecond (conv:1,2)
s.exitproc (sample,each:)
s.var (conv:1)
t. generated for: read

s.proc (read:)
s.assigncons(data,0:1,1)
s.anain (insig,10,-10,1:8:)
s.assigncons(90,1:1,1)
s.exitproc (read,data:)
s.var (insig:8)
t. generated for: fork

s.proc (fork:)
s.assigncons(loop,0:1,1)
s.assigncons(var1,1:1,1)
s.assigncons(var2,1:1,1)
s.exitproc (fork,loop:)
t. generated for: fill

s.proc (fill:)
s.assigncons(c1,0:1,1)
s.fmul (m1a,a11,m12:24,24,24)
s.fmul (m1b,a12,m13:24,24,24)

141
```plaintext
0 s.fsub (ma,mla,m1b:24,24,24)
0 s.float (sign,insig:8)
0 s.fadd (m11,sigin,ma:24,24,24)
0 s.fmul (m1a,m12,b11:24,24,24)
0 s.fmul (m1b,m11,b12:24,24,24)
0 s.fsub (q1b,m1b,m1a:24,24,24)
0 s.fassign (m13,m12:24,24)
0 s.fassign (m12,m11:24,24)
0 s.add (ctrl1,ctrl1,inc:8,8,8)
0 s.exitproc (fill1,c1::)
0 s.cons (inc,1:8)
0 s.var (q1b:24)
0 s.var (mla:24)
0 s.var (m12:24)
0 s.var (m1b:24)
0 s.var (m13:24)
0 s.var (ma:24)
0 s.var (sigin:8)
0 s.fcons (a11,0,246,208,64:24)
0 s.fcons (a12,0,212,239,192:24)
0 s.fcons (b11,0,172,244,192:24)
0 s.fcons (b12,0,172,114,64:24)
0 t. generated for:fil12 ****************************
0 s.proc (fill1::)
0 s.assigncons(c2,0:1,1)
0 s.fmul (m2a,m22,a21:24,24,24)
0 s.fmul (m2b,m23,a22:24,24,24)
0 s.fsub (mb,m2a,m2b:24,24,24)
0 s.float (intx,insig:8)
0 s.fadd (m21,intx,mb:24,24,24)
0 s.fmul (m2a,m21,b21:24,24,24)
0 s.fmul (m2b,m22,b22:24,24,24)
0 s.fsub (q2b,m2a,m2b:24,24,24)
0 s.fassign (m23,m22:24,24)
0 s.fassign (m22,m21:24,24)
0 s.add (ctr2,ctr2,inc:8,8,8)
0 s.exitproc (fill2,c2::)
0 s.var (q2b:24)
0 s.var (m2a:24)
0 s.var (m22:24)
0 s.var (m2b:24)
0 s.var (m23:24)
0 s.var (mb:24)
0 s.var (intx:8)
0 s.var (m21:24)
0 s.fcons (a21,0,24,235,64:24)
0 s.fcons (a22,0,104,228,192:24)
0 s.fcons (b21,0,34,205,192:24)
0 s.fcons (b22,0,202,56,64:24)
0 t. generated for:plus ****************************
```

s.proc  (plus::)
s.assigncons(c3,0;1,1)
s.fassign  (q1a,q1b::)
s.fassign  (q2a,q2b::)
s.fmul   (y11,q1a,pa:24,24,24)
s.fmul   (y22,q2a,ob:24,24,24)
s.fsub   (yy,y11,y22:24,24,24)
s.fsub   (y,pc,yy:24,24,24)
s.fix    (iy,y:8)
s.anaout (iy,25,-25:8;)
s.exitproc (plus,c3::)
s.var    (q1a:24)
s.var    (q2a:24)
s.var    (iy:8)
s.fcons (oa,0,0,64,194:24)
s.fcons (ob,0,0,192,193:24)
s.fcons (oc,0,0,192,128:24)
s.end    (::)
SAMPLE PARALLEL FILTER PROBLEM

; intel 8080 based system

org 65503 ; establish stack in first ram

@stack ds 32 ; define stack area
lx1 sp, @stack+32 ; initialize stack pointer
org 64 ; begin code after reserved interrupt area
jmp @spvsr ; force monitor execution

; dummy procedure for every period type contingency
	@each:
		nop ; dummy function entry point
		mvi a, 1 ; force function value
		sta each ; to true value (1)
		ret ; return to monitor

org 65502 ; 8 bit variable each in ram

@each: db 0
org 74

; procedure data
@data: nop ; entry point for data
	mvi a, 0 ; wait 25 us
	mvi a, 2 ; load loop cnt
	dcr c ; delay
	dcr a ; decrement loop count
	jnz $-2 ; loop until time is up
	in 0 ; sense environmental data
	sta dr ; and store in dr

; if dr .eq. @c001 set @t2 true
lx1 h, @t2 ; preset @t2
mvi m, 1 ; to true (1)
lda dr ; fetch first argument
lx1 h, @c001 ; compare
; procedure loop
@loop: nop ; entry point for loop
@loop: ; if qo, eq, a002 set @t3 true
lxi h, @t3 ; preset @t3
mov m,1 ; to true (1)
ida @t2 ; fetch @t2 and
cpi 0 ; test for false (0)
jz @1001 ; branch to @1001 if false
mov a, 1 ; assign constant 1
data: sta data ; to data

@1001: ; define location @1001
nop ; return to monitor, exit data

ac001: ; 8 bit variable @t2 in ram
db 0
org 65501

dt2: ; 8 bit variable data in ram
db 0
org 130
org 65500

data: ; 8 bit variable dr in ram
db 0
org 130
org 65499

dr: ;
db 0
org 130

;
mvi  a, 1 ; assign constant 1
sta  loop ; to loop
mvi  a, 0 ; assign constant 0
sta  go ; to go
@1002:  nop ; define location @1002
ret  ; return to monitor, exit loop
@c002:  db  1
org  65498 ; 8 bit variable @t3 in ram
@t3:  db  0
org  177
org  65497 ; 8 bit variable loop in ram
loop:  db  0
org  177
org  65496 ; 8 bit variable go in ram
go:  db  0
org  177
; procedure c1 ; entry point for c1
@c1:  nop ; if var1 .eq. @c003 set @t4 true
  lxi  h, @t4 ; preset @t4
  mvi  m, 1 ; to true (1)
  lda  var1 ; fetch first argument
  lxi  h, @c003 ; compare
  cmp  m ; to second argument
  jz  $+6 ; set @t4
  lxi  h, @t4 ; to false (0)
  mvi  m, 0 ; if args are not equal
  lda  @t4 ; fetch @t4 and
  cpi  0 ; test for false (0)
  jz  @1003 ; branch to @1003 if false
  mvi  a, 1 ; assign constant 1
  sta  c1 ; to c1
  mvi  a, 0 ; assign constant 0
  sta  var1 ; to var1
@1003:  nop ; define location @1003
ret  ; return to monitor, exit c1
acc003: db 1
org 65495 ; 8 bit variable at4 in ram
at4: db 0
org 224
org 65494 ; 8 bit variable c1 in ram
c1: db 0
org 224
org 65493 ; 8 bit variable var1 in ram
var1: db 0
org 224

; procedure c2
acc2: nop  ; entry point for c2
    ; if var2 .eq. acc004 set at5 true
    lxh h, at5 ; preset at5
    mvi m, 1 ; to true (1)
    lda var2 ; fetch first argument
    lxh h, acc004 ; compare
    cmp m ; to second argument
    jz $+6 ; set at5
    lxh h, at5 ; to false (0)
    mvi m, 0 ; if args are not equal
    lda at5 ; fetch at5 and
    cpi 0 ; test for false (0)
    jz @1004 ; branch to @1004 if false
    mvi a, 1 ; assign constant 1
    sta c2 ; to c2
    mvi a, 0 ; assign constant 0
    sta var2 ; to var2
@1004: nop  ; define location @1004
    ret  ; return to monitor, exit c2
acc004: db 1
org 65492 ; 8 bit variable at5 in ram
at5: db 0
org 271
org 65491
var2: db 0
org 271
org 65490
C2: db 0
org 271

; procedure c3

@c3: nop
lii h, @t6
mvi m, I
lda ctri
lii h, @c005
cmp m
jnc $+6
lii h, @t6
mvi m, 0
lda @t6
cpi 0
jz @1005
lii h, @t6
mvi m, I
lda ctri
lii h, @c005
cmp m
jnc $+6
lii h, @t6
mvi m, 0
lda @t6
cpi 0
jz @1005
mvi a, 1
sta c3
lda ctri
lii h, dec
sub m

;8 bit variable var2 in ram
;8 bit variable c2 in ram

; entry point for c3
; preset @t6
; to true (1)
; fetch first argument
; compare
to second argument
; set @t6
; to false (0)
; if arg1 not greater than arg2
; fetch @t6 and
test for false (0)
; branch to @1005 if false
; preset @t6
to true (1)
; fetch first argument
; compare
to second argument
; set @t6
; to false (0)
; if arg1 not greater than arg2
; fetch @t6 and
test for false (0)
; branch to @1005 if false
; assign constant 1
to c3
; fetch argument1
; fetch argument2
; subtract
sta ctrl1 ; store answer in ctrl1
lda ctrl2 ; fetch argument1
lxi h, dec ; fetch argument2
sub m ; subtract
sta ctrl2 ; store answer in ctrl2
@1005: nop ; define location @1005
ret ; return to monitor, exit c3
@c005: db 0
org 65489 ; 8 bit variable @c00 in ram
@t6: db 0
org 361
org 65488 ; 8 bit variable c3 in ram
c3: db 0
org 361
org 65487 ; 8 bit variable ctrl1 in ram
ctrl1: db 0
org 361
org 65486 ; 8 bit variable ctrl2 in ram
ctrl2: db 0
org 361
dec: db 1

; procedure sample
@sample: nop ; entry point for sample
mvi a, 0 ; assign constant 0
sta conv ; to conv
lda conv ; issue control
out 0 ; data through signal conv
mvi a, 1 ; assign constant 1
sta conv ; to conv
lda conv ; issue control
out 1 ; data through signal conv
ret ; return to monitor, exit sample
org 65485 ; 8 bit variable conv in ram
conv: db 0
org 389
; procedure read
read:  nop
  mvi a, 0
  sta data
; analog input channel for signal insig, range 10 to -10 volts,
; 3db rolloff at 1 khz
  in 1
  sta insig
  mvi a, 1
  sta qo
  ret
; sense environmental data
; and store in insig
; return to monitor, exit read
org 65484
; 8 bit variable insig in ram
insig: db 0
org 411

; procedure fork
fork:  nop
  mvi a, 0
  sta loop
  mvi a, 1
  sta var1
  mvi a, 1
  sta var2
  ret
; entry point for fork
; assign constant 0
; to loop
; assign constant 1
; to var1
; assign constant 1
; to var2
; return to monitor, exit fork

; procedure fill
fill:  nop
  mvi a, 0
  sta c1
; entry point for fill
; assign constant 0
; to c1
; multiply f.p. all by m12 to get mla
lda all
load arguments
out 2
lda all+1
out 2
lda all+2
out 2
lda a11+3
out 2
lda m12
out 2
lda m12+1
out 2
lda m12+2
out 2
lda m12+3
out 2
mvi a, 22b ; output command
out 3
in 2 ; unload result
sta mla+3
in 2
sta mla+2
in 2
sta mla+1
in 2
sta m1a
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a, 7b ; identify error source
call error

; multiply f.p. a12 by m13 to get m1b
lda a12 ; load arguments
out 2
lda a12+1
out 2
lda a12+2
out 2
lda a12+3
out 2
lda m13
out 2
lda m13+1
out 2
lda m13+2
out 2
lda m13+3
out 2
mvi a,22b ; output command
out 3
in 2 ; unload result
sta m1b+3
in 2
sta m1b+2
in 2
sta m1b+1
in 2
sta m1b
in 3 ; check for error status
ani 36b ; mask error code
jjz $$+8 ; no error, continue
push psw ; error, call error routine
mvi a,7b ; identify error source
call error
; subtract f.p. m1b from m1a to get ma
lda m1a ; load arguments
out 2
lda m1a+1
out 2
lda m1a+2
out 2
lda m1a+3
out 2
lda m1b
out 2
lda m1b+1
out 2
lda mlb+2
out 2
lda mlb+3
out 2
mvi a,21b ; output command
out 3
in 2 ; unload result
sta ma+3
in 2
sta ma+2
in 2
sta ma+1
in 2
sta ma
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,6b ; identify error source
call error

; convert integer insig to floating point sigin
lda insig ; load argument lsbyte
out 2
cpi 0 ; set flags
jp $+8
mvi a,377b ; negative, extend sign
jmp $+2
xra a ; positive, clear the msbyte
out 2 ; load the argument dummy msbyte
mvi a,35b ; output command
out 3
in 2
sta sigin+3 ; unload result
in 2
sta sigin+2
in 2
sta sigin+1
in 2
sta sigin
in 3 ;check error code
ani 36b ;mask the error code field
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,11b ;identify source of error
call error
;add f.p. sigin to ma to get m11
lda sigin ;load arguments
out 2
lda sigin+1
out 2
lda sigin+2
out 2
lda sigin+3
out 2
lda ma
out 2
lda ma+1
out 2
lda ma+2
out 2
lda ma+3
out 2
mvi a,20b ;output command
out 3
in 2 ;unload result
sta m11+3
in 2
sta m11+2
in 2
sta m11+1
in 2
sta m11
in  3
ani 36b
jz $+8
push psw
mvi a,5b
call error

;muliply f.p. m12 by b11 to get m1a
lda m12
out 2
lda m12+1
out 2
lda m12+2
out 2
lda m12+3
out 2
lda b11
out 2
lda b11+1
out 2
lda b11+2
out 2
lda b11+3
out 2
mvi a,22b
out 3
in  2
sta m1a+3
in  2
sta m1a+2
in  2
sta m1a+1
in  2
sta m1a
in  3
ani 36b
jz $+8  ; no error, continue
push psw  ; error, call error routine
mvi a,7b  ; identify error source
call error

; multiply f.p. m11 by b12 to get m1b
lda m11  ; load arguments
out 2
lda m11+1
out 2
lda m11+2
out 2
lda m11+3
out 2
lda b12
out 2
lda b12+1
out 2
lda b12+2
out 2
lda b12+3
out 2
mvi a,22b  ; output command
out 3
in 2  ; unload result
sta m1b+3
in 2
sta m1b+2
in 2
sta m1b+1
in 2
sta m1b
in 3  ; check for error status
ani 3bb  ; mask error code
jz $+8  ; no error, continue
push psw  ; error, call error routine
mvi a,7b  ; identify error source
call error

; subtract f.p. mla from mib to get q1b
lda mib ; load arguments
out 2
lda mib+1
out 2
lda mib+2
out 2
lda mib+3
out 2
lda mla
out 2
lda mla+1
out 2
lda mla+2
out 2
lda mla+3
out 2
mvi a,21b ; output command
out 3
in 2 ; unload result
sta q1b+3
in 2
sta q1b+2
in 2
sta q1b+1
in 2
sta q1b
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,6b ; identify error source
call error
lshld m12 ; assign value of f.p. m12 to m13
shld m13
lho d m12+2
shld m13+2
lho d m11 ; assign value of f.p m11 to m12
shld m12
lho d m11+2
shld m12+2
lda ctrl ; fetch first argument
lxi h, inc ; fetch second argument
add m ; add
sta ctrl ; store answer in ctrl
ret ; return to monitor, exit fill

inc: db 1
org 65480 ; floating point variable qlb in ram
qlb: ds 4
org 1074
org 65476 ; floating point variable m1a in ram
m1a: ds 4
org 1074
org 65472 ; floating point variable m12 in ram
m12: ds 4
org 1074
org 65468 ; floating point variable m1b in ram
m1b: ds 4
org 1074
org 65464 ; floating point variable m13 in ram
m13: ds 4
org 1074
org 65460 ; floating point variable ma in ram
ma: ds 4
org 1074
org 65459 ; 8 bit variable sigin in ram
sigin: db 0
org 1074
all: db 0
db 246
db 208
db 64
a12: db 0
  db 212
  db 239
  db 192
b11: db 0
  db 172
  db 244
  db 192
b12: db 0
  db 172
  db 114
  db 64

; procedure fil2
@fil2: nop ; entry point for fil2
  mvi  a, 0 ; assign constant 0
  sta  c2 ; to c2
; multiply f.p. m22 by a21 to get m2a
  lda m22 ; load arguments
  out  2
  lda m22+1
  out  2
  lda m22+2
  out  2
  lda m22+3
  out  2
  lda a21
  out  2
  lda a21+1
  out  2
  lda a21+2
  out  2
  lda a21+3
  out  2
  mvi  a,22b ; output command
in 2
sta m2b+2
in 2
sta m2b+1
in 2
sta m2b
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,7b ;identify error source
call error
;subtract f.p. m2b from m2a to get mb
lda m2a ;load arguments
out 2
lda m2a+1
out 2
lda m2a+2
out 2
lda m2a+3
out 2
lda m2b
out 2
lda m2b+1
out 2
lda m2b+2
out 2
lda m2b+3
out 2
mvi a,21b ;output command
out 3
in 2 ;unload result
sta mb+3
in 2
sta mb+2
in 2
sta mb+1
in 2
sta mb
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a, 6b ; identify error source
call error

; convert integer insig to floating point intx
lda insig ; load argument 1sbyte
out 2 ;
cpi 0 ; set flags
jp $+8
mvi a, 377b ; negative, extend sign
jmp $+2
xra a ; positive, clear the msbyte
out 2 ; load the argument dummy msbyte
mvi a, 35b ; output command
out 3
in 2
sta intx+3 ; unload result
in 2
sta intx+2
in 2
sta intx+1
in 2
sta intx
in 3 ; check error code
ani 36b ; mask the error code field
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a, 11b ; identify source of error
call error

; add f.p. intx to mb to get m21
lda intx ; load arguments
lda m21+2
out 2
lda m21+3
out 2
lda b21
out 2
lda b21+1
out 2
lda b21+2
out 2
lda b21+3
out 2
mvi a,22b ;output command
out 3
in 2 ;unload result
sta m2a+3
in 2
sta m2a+2
in 2
sta m2a+1
in 2
sta m2a
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,7b ;identify error source
call error
;multiply f.p. m22 by b22 to get m2b
lda m22 ;load arguments
out 2
lda m22+1
out 2
lda m22+2
out 2
lda m22+3
out 2
lda b22
out 2
lda b22+1
out 2
lda b22+2
out 2
lda b22+3
out 2
mov a, 22b ; output command
out 3
in 2 ; unload result
sta m2b+3
in 2
sta m2b+2
in 2
sta m2b+1
in 2
sta m2b
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mov a, 7b ; identify error source
call error
; subtract f.p. m2b from m2a to get q2b
lda m2a ; load arguments
out 2
lda m2a+1
out 2
lda m2a+2
out 2
lda m2a+3
out 2
lda m2b
out 2
lda m2b+1
out 2
lda m2b+2
out 2
lda m2b+3
out 2
mvi a,21b ;output command
out 3
in 2 ;unload result
sta q2b+3
in 2
sta q2b+2
in 2
sta q2b+1
in 2
sta q2b
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,6b ;identify error source
call error
lhld m22 ;assign value of f.p m22 to m23
shld m23
lhld m22+2
shld m23+2
lhld m21 ;assign value of f.p m21 to m22
shld m22
lhld m21+2
shld m22+2
lda ctr2 ;fetch first argument
lxh h, inc ;fetch second argument
add m ;add
sta ctr2 ;store answer in ctr2
ret ;return to monitor, exit file
org 65455 ;floating point variable q2b in ram
db 202
db 56
db 64

; procedure plus
@plus: nop
    mvi a, 0 ; assign constant 0
    sta c3   ; to c3
    lhl d qlb ; assign value of f.p qlb to qla
    shl d qla
    lhl d qla+2
    shl d qla+2
    lhl d q2b ; assign value of f.p q2b to q2a
    shl d q2a
    lhl d q2a+2
    shl d q2a+2

; multiply f.p. qla by pa to get y11
    lda qla  ; load arguments
    out 2
    lda qla+1
    out 2
    lda qla+2
    out 2
    lda qla+3
    out 2
    lda pa
    out 2
    lda pa+1
    out 2
    lda pa+2
    out 2
    lda pa+3
    out 2
    mvi a,22b ; output command
    out 3
    in 2 ; unload result
sta y11+3
in 2
sta y11+2
in 2
sta y11+1
in 2
sta y11
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,7b ; identify error source
call error

; multiply f.p. q2a by pb to get y22
lda q2a ; load arguments
out 2
lda q2a+1
out 2
lda q2a+2
out 2
lda q2a+3
out 2
lda pb
out 2
lda pb+1
out 2
lda pb+2
out 2
lda pb+3
out 2
mvi a,22b ; output command
out 3
in 2 ; unload result
sta y22+3
in 2
sta y22+2
in 2
sta y22+1
in 2
sta y22
in 3 ;check for error status
ani 36b ;mask error code
jz $+8 ;no error, continue
push psw ;error, call error routine
mvi a,7b ;identify error source
call error
;subtract f.p. y22 from y11 to get yy
lda y11 ;load arguments
out 2
lda y11+1
out 2
lda y11+2
out 2
lda y11+3
out 2
lda y22
out 2
lda y22+1
out 2
lda y22+2
out 2
lda y22+3
out 2
mvi a,21b ;output command
out 3
in 2 ;unload result
sta yy+3
in 2
sta yy+2
in 2
sta yy+1
in 2
sta yy
in 3 ; check for error status
ani 36b ; mask error code
jz $+8 ; no error, continue
push psw ; error, call error routine
mvi a,6b ; identify error source
call error
; subtract f.p. yy from pc to get y
lda pc ; load arguments
out 2
lda pc+1
out 2
lda pc+2
out 2
lda pc+3
out 2
lda yy
out 2
lda yy+1
out 2
lda yy+2
out 2
lda yy+3
out 2
mvi a,21b ; output command
out 3
in 2 ; unload result
sta y+3
in 2
sta y+2
in 2
sta y+1
in 2
sta y
in 3 ; check for error status
ani 36b ; mask error code
jz  $+8 ; no error, continue
push psw ; error, call error routine
mvi a,6b ; identify error source
call error
; convert floating bit y to integer iy
lda y ; load arguments
out 2
lda y+1
out 2
lda y+2
out 2
lda y+3
out 2
mvi a,37b ; load command
out 3
in 2 ; unload result
mov b,a
in 2
sta iy
call dstat
jz  $+8 ; no error detected
push psw ; error, call the error routine
mvi a,32b
push psw ; send a code to indicate source of error
call error
in 3 ; check chip for error status
ani 36b
jz  $+8
push psw
mvi a,32b
push psw
call error

analog output channel for signal iy, range 25 to -25
lda x70 ; issue control
out 5 ; data through signal x70
ret ; return to monitor, exit plus
org 65426 ; floating point variable q1a in ram
q1a: ds 4
org 2154
org 65422 ; floating point variable q2a in ram
q2a: ds 4
org 2154
org 65421 ; 8 bit variable iy in ram
    iy: db 0
org 2154
pa: db 0
db 0
db 64
db 194
pb: db 0
db 0
db 192
db 193
pc: db 0
db 0
db 192
db 128
end ; software complete
; =monitor section=
; @spvsr: lld @table ; initialize table pointer
    shld @pntr ; to beginning
@mlop : lld @pntr ; main loop: get pntr
    inx h ; increment
    inx h ; the pointer by 3
    inx h ; (bytes to bypass jmp)
    shld @pntr ; store pointer address for table
    pchl ; begin execution at current entry
; data section
    org 65420
@pntr: dw 0 ; table entry address pointer
org 2181

@table: dw @ptr ; table header (define top)
jmp @each ; test for contingency each
jmp @data ; test for contingency data
jmp @loop ; test for contingency loop
jmp @c1 ; test for contingency c1
jmp @c2 ; test for contingency c2
jmp @c3 ; test for contingency c3
jmp @spvswr ; go to start of table
;
@each:call @each ; execute contingency code each
lda each ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @sample ; execute task sample if true
jmp @mlop ; return to monitor
;
@data:call @data ; execute contingency code data
lda data ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @read ; execute task read if true
jmp @mlop ; return to monitor
;
@loop:call @loop ; execute contingency code loop
lda loop ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @fork ; execute task fork if true
jmp @mlop ; return to monitor
;
@c1:call @c1 ; execute contingency code c1
lda c1 ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @fill ; execute task fill if true
jmp @mlop ; return to monitor
;
@c2:call @c2 ; execute contingency code c2
lda c2 ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @fl2 ; execute task fl2 if true
jmp @mloop ; return to monitor
;
@tc3:call @c3 ; execute contingency code c3
lda c3 ; fetch contingency result
cpi 1 ; and compare to true flag (1)
cz @plus ; execute task plus if true
jmp @mloop ; return to monitor

; routine to test if double length conversion to single length
; format has caused overflow or underflow
; enter with msbyte in b, lsbyte in a
dstst: mov c,a
      mov a,b
      cpi 0 ; set flags
      jp dspos ; test for pos
      cpi -1b ; test for -1
      jnz dsuf1 ; underflow detected
      mov a,b ; ok so far, test sign of lsbyte
      cpi 0 ;
      jn dsuf1 ; underflow by sign error
      ret ; ok, return

dspos: jnz dsol1 ; + overflow
      mov a,c ; zero msbyte, test sign of lsbyte
      cpi 0
      jn dsol1 ; sign error overflow
      ret ; ok, return

dsufl: mvi a,4b ; underflow, return error code
      ret

dsol1: mvi a,2b ; underflow, return error code
      ret

this realization consumes 9,635 watts of power
central processing unit
device: intel 8080 8-bit microprocessor, ic 1
connections:
pins 25, 26, 27, 29, 30, 31, 32, 33,
    34, 35, 1, 40, 37, 38, 39, 36 = a(0:15)
pins 10, 9, 8, 7, 3, 4, 5, 6, = d(0:7)
pin 21 = hlda
pin 12 = reset
pin 13 = hold
pin 14 = int
pin 16 = inte
pin 17 = dbin
pin 18 = wr-bar
pin 19 = sync
pin 22 = phi1
pin 15 = phi2
pin 23 = ready
pin 24 = wait
pin 2 = gnd
pin 20 = +5v
pin 11 = -5v
pin 28 = +12v
clock generator (0.5 us)
device: intel 8224 clock generator and driver for 8080 cpu, ic 2
connections:
pin 1 (reset) = reset
pin 4 (ready) = ready
pin 5 (sync) = sync
pin 10 (phi2) = phi2
pin 11 (phi1) = phi1
pin 14,15 (xtal(1:2)) = device: 18 mhz crystal
pin 16 (vcc) = +5v
pin 9 (vdd) = +12v
pin 8 (gnd) = gnd
d(0:7) = db(0:7)
cpu status latch
device: intel 8212 8-bit i/o port, ic 3
connections:
pins 3,5,7,9,16,18,20,22 (di (0:7)) = d(0:7)
pin 4 (do(1)) = inta
pin 6 (do(2)) = wo-bar
pin 8 (do(3)) = stack
pin 10 (do(4)) =HLTA
pin 15 (do(5)) = out
pin 17 (do(6)) = ml
pin 19 (do(7)) = inp
pin 21 (do(8)) = memr
condition-mode input interface hardware to sense signal dr
device: intel 8212 8-bit i/o port, ic 4
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = dr(1:8), remainder to qnd
pins 4,6,8,10,15,17,19,21 (do(1:8)) = db(1:8)
pin 2 (md) = qnd
pin 11 (stb) = qnd
pin 1 (ds1-bar) = .not. (decode a(0:7) value 0)
pin 13 (ds2) = ino .and. dbin
pin 24 (vcc) = +5v
pin 12 (gnd) = qnd
condition-mode output interface hardware to issue signal: conv
device: intel 8212 8-bit i/o port, ic 5
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = conv(1:8); if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = qnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 0)
pin 24 (vcc) = +5v
pin 12 (gnd) = qnd
condition-mode output interface hardware to issue signal: conv
device: intel 8212 8-bit i/o port, ic 6
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = conv(1:8) if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = qnd
pin 1 (dso-bar) = wr-bar
pin 13 (ds2) = out . and. (decode a(0:7) value 1)
pin 24 (vcc) = +5v
pin 12 (gnd) = qnd
connector j 1, for analog signal insig,
16 pin dio socket
connections:
  pin 1 = x6
  pin 2 = x7
  pin 3 = qnd grounded at signal source only
resistor r 1, 10000 ohms, 1/4 watt 1% metal film
  pin 1 = x6
  pin 2 = x1
resistor r 2, 10000 ohms, 1/4 watt 1% metal film
  pin 1 = x7
  pin 2 = x2
resistor r 3, 10000 ohms, 1/4 watt 1% metal film
  pin 1 = x1
  pin 2 = x5
resistor r 4, 10000 ohms, 1/4 watt 1% metal film
  pin 1 = x2
  pin 2 = qnd
capacitor, c 1, ceramic, 16000 pf
  connections:
    pin 1 = x1
    pin 2 = x5
trimpot, r 5, 10000 ohms, 22t 1/2w cermet
  pin 1 = x3
  pin 2 = -15v (wiper)
  pin 3 = x4 (cw end)
oo amp, ic ~
  device is analog devices ad741k
connections:
pin 1 = x3  (zero trimpot)
2 = x1  (negative input)
3 = x2  (positive input)
4 = -15v
5 = x4  (zero trimpot)
6 = x5  (output)
7 = +15v
8 = n.c.  tab at pin 8
trimpot, r 6,  200 ohms, 22t 1/2w cermet
pin 1 = gnd
pin 2 = x5  (wiper)
pin 3 = x41  (cw end)
a/d converter, 8 bit
device is analog devices ad570, ic 8
connections:
  pin 1 = n.c.
  pin 2 = insig(0)  (lsb)
  pin 3 = insig(1)  (2lsb)
  pin 4 = insig(2)  (3lsb)
  pin 5 = insig(3)  (4lsb)
  pin 6 = insig(4)  (5lsb)
  pin 7 = insig(5)  (6lsb)
  pin 8 = insig(6)  (7lsb)
  pin 9 = insig(7)  (msb)
  pin 10 = +5 volts
  pin 11 = conv  (blank and not convert)
  pin 12 = -15 volts
  pin 13 = x41  (analog input)
  pin 14 = gnd  (analog common)
  pin 15 = gnd  (bipolar offset)
  pin 16 = gnd  (digital common)
  pin 17 = dr  (data ready)
  pin 18 = n.c.
condition-mode input interface hardware to sense signal insig
device: intel 8212  8-bit i/o port, ic 9
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = insig(1:8), remainder to gnd
  pins 4,6,8,10,15,17,19,21 (do(1:8) = db(1:8)
  pin 2 (md) = gnd
  pin 11 (stb) = gnd
  pin 1 (ds1-bar) = .not. (decode a(0:7) value 1)
  pin 13 (ds2) = inp .and. dbin
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd

math processor chip, ic10
  type is intel c8231
connections:
  pin 1 = qnd (vss
  pin 2 = +5v (vcc
  pin 3 = n.c. (/eack)
  pin 4 = n.c. (/svack)
  pin 5 = n.c. (svreq)
  pin 6 = n.c. (reserved)
  pin 7 = n.c. (reserved)
  pin 8 = db0
  pin 9 = db1
  pin 10 = db2
  pin 11 = db3
  pin 12 = db4
  pin 13 = db5
  pin 14 = db6
  pin 15 = db7
  pin 16 = +12v (vdd
  pin 17 = ready
  pin 18 = .not. (decode(a1-a7);value 2) (cs)
  pin 19 = .not. iow (wbar)
  pin 20 = .not ior (rbar)
  pin 21 = a0
  pin 22 = reset
  pin 23 = ph2 (clock)
  pin 24 = n.c. (/end)
connector j 2, for analog signal iy,
16 pin dip socket
connections:
    pin 1 = x70
    pin 2 = gnd
    pin 3 = open grounded at signal source only
voltage follower, ic11
device is lm310
connections:
    pin 1 = n.c.  (balance)
    2 = n.c.    
    3 = x61    ;(input)
    4 = -15v   (v-)
    5 = n.c.   (booster)
    6 = iy    ;(output)
    7 = +15v   (v+)
    8 = n.c.   (balance, tab at pin 8)

8 bit dac, ic 12
device is burr-brown dac82, laser trimmed, no adj rqd,
connections:
    pin 3 = +15v
    pin 4 = x62(7)
    pin 5 = x62(6)
    pin 6 = x62(5)
    pin 7 = x62(4)
    pin 8 = x62(3)
    pin 9 = x62(2)
    pin 10 = x62(1)
    pin 11 = x62(0)
    pin 12 jumper to pin 15 to use internal current ref
    pin 13 = +15v
    pin 14 = gnd
range dependent connections for 5000 to -5000 mv range
    pin 1 = gnd
    pin 2 = x61    ;(output)
    pin 16 jumper to pin 18
pin 17 jumper to pin 2
ttl inverter, element 1 of ic 13, 7404
  pin 1 = i1y(1) ;(input)
  pin 2 = x62 ;(output)
  pin 7 = gnd
  pin 14 = +5v
ttl inverter, element 2 of ic 13, 7404
  pin 3 = x71 ;(input)
  pin 4 = x63 ;(output)
ttl inverter, element 3 of ic 13, 7404
  pin 5 = x72 ;(input)
  pin 6 = x64 ;(output)
ttl inverter, element 4 of ic 13, 7404
  pin 9 = x73 ;(input)
  pin 8 = x65 ;(output)
ttl inverter, element 5 of ic 13, 7404
  pin 11 = x74 ;(input)
  pin 10 = x66 ;(output)
ttl inverter, element 6 of ic 13, 7404
  pin 13 = x75 ;(input)
  pin 12 = x67 ;(output)
 ttl inverter, element 1 of ic 14, 7404
  pin 1 = x76 ;(input)
  pin 2 = x68 ;(output)
  pin 7 = gnd
  pin 14 = +5v
ttl inverter, element 2 of ic 14, 7404
  pin 3 = x77 ;(input)
  pin 4 = x69 ;(output)

condition-mode output interface hardware to issue signal: x70
device: intel 8212 8-bit i/o port, ic 15
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x70(1:8) ; if 8 are req
  pin 2 (md) = +5v
  pin 11 (sth) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 5)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

system memory
time delay to match 8111-2 latency to 8080
device: texas instruments sn74175 hex-quad d-type flip-flop with clear
ic 16
connection:
  pin 9 (clock) = .not. phi2
  pin 1 (clear) = .not. sync
  pin 16 (vcc) = +5v
  pin 8 (gnd) = gnd
  pin 2 (1q) = q
  pin 4 (1d) = +5v
  ready = q .and. .not. memr

16k eprom, ic 17, 8 pages of 256 words, starting at page 0
device is 2716 (do not use ti -- different pinout)
connections:
  pin 1 = a(8)
  pin 2 = a(7)
  pin 3 = a(6)
  pin 4 = a(5)
  pin 5 = a(4)
  pin 6 = a(3)
  pin 7 = a(2)
  pin 8 = a(1)
  pin 9 = db(1)
  pin 10 = db(2)
  pin 11 = db(3)
  pin 12 = grd
  pin 13 = db(4)
  pin 14 = db(5)
  pin 15 = db(6)
  pin 16 = db(7)
  pin 17 = db(8)
pin 18 = grd (pd/pqm)
pin 19 = a(11)
pin 20 = .not.(decode(a(12:16) value 0) .and. .not. rdbar)
pin 21 = +5v (vpp)
pin 22 = a(10)
pin 23 = a(9)
pin 24 = +5v

16k eprom, ic 18, 8 pages of 256 words, starting at page 1
device is 2716 (do not use ti -- different pinout)
connections:
  pin 1 = a(8)
  pin 2 = a(7)
  pin 3 = a(6)
  pin 4 = a(5)
  pin 5 = a(4)
  pin 6 = a(3)
  pin 7 = a(2)
  pin 8 = a(1)
  pin 9 = db(1)
  pin 10 = db(2)
  pin 11 = db(3)
  pin 12 = grd
  pin 13 = db(4)
  pin 14 = db(5)
  pin 15 = db(6)
  pin 16 = db(7)
  pin 17 = db(8)
  pin 18 = grd (pd/pqm)
  pin 19 = a(11)
  pin 20 = .not.(decode(a(12:16) value 1) .and. .not. rdbar)
  pin 21 = +5v (vpp)
  pin 22 = a(10)
  pin 23 = a(9)
  pin 24 = +5v

random access memory (lower half of page 255)
device: intel 8111-2 1024 hit (256*4) static mos ram, ic 19
connections:

pins 4, 3, 2, 1, 17, 5, 6, 7 (a(0:7)) = a(0:7)

pin 9 (od) = dbin
pin 16 (rw) = rw
pin 15 (ce1-bar) = ce
pin 10 (ce2-bar) = gnd
pins 11, 12, 13, 14 (io(1:4)) = db(0:3)

ce = (not. (out .and. wr-bar) .or. memr)
    .and. (not. (decode a(8:15) value 255))

pin 18 (vcc) = +5v
pin 8 (gnd) = gnd

random access memory (upper half of page 255)
device: intel 8111-2 1024 bit (256x4) static mos ram, ic 20

connection:

pins 4, 3, 2, 1, 17, 5, 6, 7 (a(0:7)) = a(0:7)

pin 9 (od) = dbin
pin 16 (rw) = rw
pin 15 (ce1-bar) = ce
pin 10 (ce2-bar) = gnd
pins 11, 12, 13, 14 (io(1:4)) = db(4:7)

pin 18 (vcc) = +5v
pin 8 (gnd) = gnd
APPENDIX C
ERROR CORRECTIONS

The following errors in the design system were detected and corrected during the course of completing this thesis.

A. DESIGN PROGRAM

The call to subroutine GETNM for subroutine SYMVAL was corrected. A variable passed in the call, JJ, was incorrectly passed with a value of zero. This was changed to one prior to the call to GETNM.

Various errors in branch statements existed throughout the design program listing and appeared to be the result of either an incorrect transfer of the original program to magnetic tape or errors in reading the magnetic tape by the VAX 11/780. Errors of this nature were detected in subroutines ARGIF, SEDUL, and TMANAL. These have been corrected, but validation of the remainder of the program must be accomplished.

Subroutine OUTCOM writes the power consumption of the realization and its chip count to the terminal screen. This was observed to occur twice at the end of each execution. The subroutine was corrected so that only one such statement is now displayed for the user. The destination of the duplicate statement was changed to the end of the software listing generated for each successful realization.
Subroutine SEDUL calls a second subroutine, INSERT, from two possible locations. The first of these used an integer variable as a parameter for a variable defined within INSERT as real. This was observed to prevent successful generation of realizations for test input. The variable RZZ was added as a parameter in the calling routine. It is equated to the value of the integer array element ORDER(istop-1,stop) prior to the call to INSERT.

INSERT sets the values of the array ORDER as part of its execution. The type of ORDER is declared to be integer. However, the values assigned to it by INSERT are obtained from real variables. This caused an integer overflow error in the program which resulted in termination of the design attempt. The problem was corrected by fixing the values of the real variables when equated to the elements of array ORDER.

B. REALIZATION LIBRARY

Entry h.dac was missing the qualifiers on all skip statements. These were determined and added.

Qualifiers for skip statements in EPROM hardware primitive were set to zero. This caused infinite generation of real only memory. The qualifiers were corrected to their proper values.
Numerous errors attributable to improper transfer of the program from magnetic tape to the VAX were detected and corrected. Validation of the complete listing must be performed.

The values of the qualifiers for the SKIP statements in primitives h.adc and h.bufframp were incorrect. These were corrected to their proper values.

C. UNCORRECTED ERRORS

Two errors are known to exist in the main program but their source could not be determined. Both of these prevent proper program execution.

The first error occurs when a dual processor realization is generated. A system error is produced when the program attempts to access the file of monitor primitives.

The second error consists of a failure to properly evaluate the value of parameters enclosed in the symbols '< >'. This prevents the proper utilization of library primitives which use parameters as qualifiers to logical statements.
LIST OF REFERENCES


2. Ibid., p. 10.


14. Ibid., p. 77-78.


<table>
<thead>
<tr>
<th>No.</th>
<th>Distribution List</th>
<th>Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Defense Technical Information Center, Cameron Station, Alexandria, Virginia</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td>Library, Code 0142, Naval Postgraduate School, Monterey, California</td>
<td>2</td>
</tr>
<tr>
<td>3.</td>
<td>Department Chairman, Code 62, Naval Postgraduate School, Monterey, California</td>
<td>2</td>
</tr>
<tr>
<td>4.</td>
<td>Professor H. H. Loomis, Jr., Code 62Lm, Naval Postgraduate School, Monterey, California</td>
<td>4</td>
</tr>
<tr>
<td>5.</td>
<td>Lieutenant Colonel A. A. Ross, USAF, Code 52Rs, Naval Postgraduate School, Monterey, California</td>
<td>2</td>
</tr>
<tr>
<td>6.</td>
<td>Dr. M. N. Matelan, Dallas, Texas</td>
<td>1</td>
</tr>
<tr>
<td>7.</td>
<td>Lieutenant M. R. Heilstedt, USN, Naval Sea Systems Command, St. Paul, Minnesota</td>
<td>2</td>
</tr>
</tbody>
</table>