Automatic Design of Logical Networks*

T. C. BARTEET†

The frequent use of transcendental functions in digital computer programs has presented a basic problem in the design of real-time control systems. Since there are usually no single instructions to perform such functions as sine $x$, arc sine $x$, etc., two other techniques have been used.

The first technique involves storing a table of values for the function in a rapid access memory. The advantage of this technique lies in the speed obtainable; however, in order to provide reasonable accuracy, the table must be of considerable length. Computations which involve physical measurements such as those made by one of the newer radars would require a table which would occupy a significant portion of most storage devices. An alternate technique involves storing a smaller table, and then interpolating between the values stored in this table. The principal advantage of table storage is then lessened, for the interpolation routine requires time and the function can no longer be generated as quickly.

The second technique involves the use of a programmed approximation routine. Most of the routines used are based on polynomial approximations which require several multiplications, and as a consequence, are time-consuming.

The utilization of digital computers in real-time control systems which must process large amounts of data at high speeds has made the problem more acute. Most programs for such systems require values for sines, cosines, etc., quite often, and the necessary computations must be made quickly and with considerable accuracy. The need therefore exists for a very fast and accurate method of generating trigonometric functions.

Lincoln Laboratory has recently prepared a set of flexible computer programs which automatically perform the design of logical networks which will perform such functions in a single step. These networks may then be connected into the arithmetic element of an internal-binary-operation digital computer, providing the machine with instructions which will yield the sine, arc sine, etc., of an angle stored in one of the registers of the arithmetic element. The design technique yields a logical network with a number of input and output lines, the input lines representing the independent variable and the output lines the dependent variable. The networks are completely digital in operation, utilizing high-speed transistor and diode logical circuitry.

Because of the magnitude of the design problem for large logical networks, conventional manual design tech-

* The work reported here was performed at Lincoln Lab., Mass. Inst. Tech., Lexington, Mass., with the joint support of the Army, Navy, and Air Force.
† Lincoln Lab., M.I.T., Lexington, Mass.
work is that for each combination of input values there will be a unique output value, represented by signals on the output lines. Since the output from a logical network of this sort is determined by the inputs to the logical network, it is possible to construct a logical network that will perform any function having the characteristic that for each state which the inputs assume, there will be a unique output. It may be seen that this does not limit the function performed to transcendental or straightforward mathematical relationships, but allows any input-to-output relationship.

When a logical network contains several input and output lines, it is possible to simplify the design problem by considering only one output line at a time. The signal on this particular line is determined by the values of the inputs to the logical network at that time, and may be specified by means of a Boolean function of the input variables. This expression will be equal to 1 when the output from this particular line of the network is 1 and to 0 when the output is 0. An expression that represents the operation of a single line of a logical network is called the transmission function for that line of the network. A fully developed transmission function, when written in the sum-of-products form, consists of a number of terms, each of which is a product of all of the input variables, certain of which may be complemented. This type of expression is referred to as a canonical expansion for the circuit transmission.

The first step in the design procedure for a network with several input and output lines consists of deriving the canonical expansion for each output line of the network. Table I shows the derivation of the canonical expansion for the transmission function for one output line bit of a small network which will yield the sine of \( x \) within the limits 0° to 90°. The angles in the first column of the table are expressed as binary numbers, starting with 0000 and increasing by steps of 0001, which increases the angle by 6° in each case. The values of sine \( x \), expressed in binary form, are listed in the second column of the table. The computer program generates the sine values for each input angle and then, examining one bit of the

![Diagram 1: The 'or' circuit.](image1)

![Diagram 2: Diode 'and' circuit.](image2)

![Diagram 3: Two-level 'and-or' circuit.](image3)

![Diagram 4: Transistor inverter.](image4)
Table I illustrates the derivation of the product terms which generated the 1 output in the output table. The second-output-bit column is underlined, as is the product sine values at a time, develops and stores the canonical expansion for each output line of the logical network. In effect, the computer proceeds down a column of output values until it finds a 1. It then stores the input values which generated the 1 output in the output table. Table I illustrates the derivation of the product terms for the second bit of the four output bits. Each 1 in the second-output-bit column is underlined, as is the product term for this particular output. The complete canonical expansion for the output line representing the second least significant output bit of this particular table is illustrated at the bottom of the figure. An expression is therefore generated for each output line of the logical network.

The minimization technique which has been programmed is based on work done by Quine of Harvard, and by McClusky during the preparation of his doctoral dissertation at M.I.T. The input to this particular program is the canonical expansion developed by the preceding program, and the output is a minimized equivalent sum-of-products expression.

From the collection of the Computer History Museum (www.computerhistory.org)
variables in order. Each term is therefore represented by an equivalent to the original expression. An important characteristic of these terms is that none of them can be eliminated in this process, it is convenient to utilize another register, one containing the "values" of the variables as variables are eliminated, while maintaining the register of computer storage as a mask, and to alter the shortened terms of the original expression are stored in binary form, using positional notation to maintain the identity of the variables. A 1 is used to indicate an unprimed variable and a 0 to indicate a primed variable, so that \( ab'c'd \) is expressed as 1011. Since variables will be eliminated in this process, it is convenient to utilize another register of computer storage as a mask, and to alter the mask as variables are eliminated, while maintaining the variables in order. Each term is therefore represented by two registers, one containing the "values" of the variables and another the mask for the term. The mask is altered to indicate the eliminated variables. For instance, if two terms 1011 and 1010 and their masks are matched, the resulting term is 1011- or \( ab'c' \). The first step in the matching process in the computer, therefore, is to compare the masks to see if they are identical. If the masks are identical, the terms are then matched and if they differ in only one variable, a new term is formed along with a new mask which indicates the missing variable or variables.

In order to shorten the matching problem, McCluskey has shown that the terms may first be sorted according to the number of 1's in each term. Table III illustrates the same set of terms after they have been sorted and arranged in tabular form. Each section of a column of the table contains terms with one more 1 than the terms in the preceding section of the table. It is necessary to match only the terms from one section of the table with the terms in the preceding and following sections of the table, for two terms which differ by more than one 1 cannot match: 1011 cannot match with any term containing only one 1, for instance 1000, for the two terms must, of necessity, vary in more than one variable. Further, if the terms from one section of the table are matched with those of the next section, the resulting shortened terms can be matched only with shortened terms formed by matches in the preceding and following sections of the table. This technique significantly reduces the number of matches which must be made and also materially lessens the amount of fast-access memory that is required at a given time. The number of terms formed by the matching process tends to increase considerably for large problems before finally decreasing. By storing only the sections of the table which are being matched in high-speed memory, and storing the rest of the terms on tape or drums, large problems may be handled more easily.

The set of terms derived in this manner, if collected in sum-of-products form, will form an expression equivalent to the original expression. An important characteristic of these terms is that none of them can be shortened by omitting a variable. Quine has shown that the shortest sum-of-products expression must consist of a subset of these terms. It may be shown that certain terms that did not match during the process comprise the prime implicants.

The right half of Table II illustrates how the above matching process is performed in the computer. The terms of the original expression are stored in binary form, using positional notation to maintain the identity of the variables. A 1 is used to indicate an unprimed variable and a 0 to indicate a primed variable, so that \( ab'c'd \) is expressed as 1011. Since variables will be eliminated in this process, it is convenient to utilize another register of computer storage as a mask, and to alter the mask as variables are eliminated, while maintaining the variables in order. Each term is therefore represented by two registers, one containing the "values" of the variables and another the mask for the term. The mask is altered to indicate the eliminated variables. For instance, if two terms 1011 and 1010 and their masks are matched, the resulting term is 1011- or \( ab'c' \). The first step in the matching process in the computer, therefore, is to compare the masks to see if they are identical. If the masks are identical, the terms are then matched and if they differ in only one variable, a new term is formed along with a new mask which indicates the missing variable or variables.

In order to shorten the matching problem, McCluskey has shown that the terms may first be sorted according to the number of 1's in each term. Table III illustrates the same set of terms after they have been sorted and arranged in tabular form. Each section of a column of the table contains terms with one more 1 than the terms in the preceding section of the table. It is necessary to match only the terms from one section of the table with the terms in the preceding and following sections of the table, for two terms which differ by more than one 1 cannot match: 1011 cannot match with any term containing only one 1, for instance 1000, for the two terms must, of necessity, vary in more than one variable. Further, if the terms from one section of the table are matched with those of the next section, the resulting shortened terms can be matched only with shortened terms formed by matches in the preceding and following sections of the table. This technique significantly reduces the number of matches which must be made and also materially lessens the amount of fast-access memory that is required at a given time. The number of terms formed by the matching process tends to increase considerably for large problems before finally decreasing. By storing only the sections of the table which are being matched in high-speed memory, and storing the rest of the terms on tape or drums, large problems may be handled more easily.

The set of terms derived in this manner, if collected in sum-of-products form, will form an expression equivalent to the original expression. An important characteristic of these terms is that none of them can be shortened by omitting a variable. Quine has shown that the shortest sum-of-products expression must consist of a subset of these terms. It may be shown that certain terms that did not match during the process comprise the prime implicants.

The right half of Table II illustrates how the above matching process is performed in the computer. The terms of the original expression are stored in binary form, using positional notation to maintain the identity of the variables. A 1 is used to indicate an unprimed variable and a 0 to indicate a primed variable, so that \( ab'c'd \) is expressed as 1011. Since variables will be eliminated in this process, it is convenient to utilize another register of computer storage as a mask, and to alter the mask as variables are eliminated, while maintaining the variables in order. Each term is therefore represented by two registers, one containing the "values" of the variables and another the mask for the term. The mask is altered to indicate the eliminated variables. For instance, if two terms 1011 and 1010 and their masks are matched, the resulting term is 1011- or \( ab'c' \). The first step in the matching process in the computer, therefore, is to compare the masks to see if they are identical. If the masks are identical, the terms are then matched and if they differ in only one variable, a new term is formed along with a new mask which indicates the missing variable or variables.

In order to shorten the matching problem, McCluskey has shown that the terms may first be sorted according to the number of 1's in each term. Table III illustrates the same set of terms after they have been sorted and arranged in tabular form. Each section of a column of the table contains terms with one more 1 than the terms in the preceding section of the table. It is necessary to match only the terms from one section of the table with the terms in the preceding and following sections of the table, for two terms which differ by more than one 1 cannot match: 1011 cannot match with any term containing only one 1, for instance 1000, for the two terms must, of necessity, vary in more than one variable. Further, if the terms from one section of the table are matched with those of the next section, the resulting shortened terms can be matched only with shortened terms formed by matches in the preceding and following sections of the table. This technique significantly reduces the number of matches which must be made and also materially lessens the amount of fast-access memory that is required at a given time. The number of terms formed by the matching process tends to increase considerably for large problems before finally decreasing. By storing only the sections of the table which are being matched in high-speed memory, and storing the rest of the terms on tape or drums, large problems may be handled more easily.

The set of terms derived in this manner, if collected in sum-of-products form, will form an expression equivalent to the original expression. An important characteristic of these terms is that none of them can be shortened by omitting a variable. Quine has shown that the shortest sum-of-products expression must consist of a subset of these terms. It may be shown that certain
of programs are slightly over 3000 orders in length. The program requires about 40 seconds to generate the canonical expansion for each output line of a 12 input-line by 14 output-line problem and from eight to ten minutes to minimize and print the final expression. About 25,000 registers are required to store partial results during the processing. As a result, drum storage is used during the minimization procedure. The procedure is now being programmed for an IBM 709 located at the laboratory, making possible the solution of larger problems and somewhat shortening the programs' running time due to the large core storage of the 709.

The design procedure described here appears very flexible. It can be used to perform automatically the logical design of circuitry which will perform any function which has a unique value of the dependent variable for each value of the independent variable.

To date, networks which yield sine, arc sine, and the square root of the input value have been constructed. The concept of programmed logic as an aid to computer design appears quite attractive for the design of future machines.

ACKNOWLEDGMENT

The computer programs described in this report were written by H. C. Peterson who has also contributed many helpful suggestions. The original problem of designing computer instructions which would yield transcendental operations arose during Lincoln Laboratories' ballistic missile early warning system studies and was suggested to the author by W. I. Wells. The author also wishes to thank V. A. Nedzel, R. W. Sittler, and J. F. Nolan for their helpful comments during the writing of this report.

The Role of Digital Computers in the Dynamic Optimization of Chemical Reactions

R. E. KALMAN† AND R. W. KOEPCKE‡

I. INTRODUCTION

Along with the increasing availability of high-speed, large-storage digital computers, there has been growing interest in their utilization for real-time control purposes. A typical problem in this connection and one of long-standing interest is the optimal static and dynamic operation of chemical reactors. To our knowledge, no digital computer is being used for this purpose, chiefly because of the many difficulties encountered in utilizing real-time machine computation in reactor control. These difficulties range from the unavailability or inadequacy of hardware (i.e., transducers, measuring instruments, low-level analog-to-digital converters, etc.) to the lack of a well-established body of fundamental theoretical principles. Although a great deal is known about the basic concepts governing control systems, present methods cannot be readily applied to designing a program for a real-time digital control computer. This is because the existing design methods are applicable primarily to fairly small-scale systems, whereas the use of a digital computer (in fact the very attractiveness of computer control) arises primarily in connection with large-scale problems.

The role of the digital computer in real-time control consists essentially of "digesting" large amounts of information obtained from the primary measuring instruments and then calculating, as rapidly as possible, the control action to be taken on the basis of these measurements.

One purpose of this report is to provide a broad outline of a new approach to designing control systems for chemical processes which are to be built around a fast, general-purpose digital computer operating in real time. The specific engineering details of the computer will not be of any interest here; rather, we have concentrated on studying the types of computations the computer is to perform. To lend concreteness to the discussion, the chemical process under consideration will be a continuous-flow, stirred reactor. After the fundamental concepts have been established, the detailed analytic equations (in the linear case) leading to the dynamically optimal (and thus also statically optimal) design of the reaction control system are given in Section III. The equations of Section III represent a special case of the new design theory of linear control systems formulated...