1967

A programmed synthesis procedure for asynchronous sequential circuits

Robert Judson Smith

Follow this and additional works at: http://scholarsmine.mst.edu/masters_theses
Department: Electrical and Computer Engineering

Recommended Citation
A PROGRAMMED SYNTHESIS PROCEDURE
FOR
ASYNCHRONOUS SEQUENTIAL CIRCUITS

BY
ROBERT J. SMITH II - 1944

A
THESIS
submitted to the faculty of
THE UNIVERSITY OF MISSOURI AT ROLLA
in partial fulfillment of the requirements for the
Degree of
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
Rolla, Missouri
1967

Approved by

James W. Cady (advisor)
Billy E. Stillett

Frank J. Kern
ABSTRACT

The synthesis procedure for large asynchronous normal fundamental-mode sequential circuits is time-consuming and tedious when performed manually. This paper describes a series of computer programs which, given a machine specification in flow table form, automatically produces simplified normal form implementation design equations.

The user may specify that either minimum-variable state assignments, or more rapidly generated near-minimum-variable codes be found. Next-state equations are formed using the generated state assignments and converted to simplified sums of products. Sample machines having flow tables of up to eighteen rows by four columns are synthesized.
ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Dr. James H. Tracey for his guidance and advice in the preparation of this thesis.

Appreciation is also extended to Wayne L. Schoeffel for his helpful suggestions on the utilization of PL/1 and for his cooperation in the modification of his state assignment generation programs.

Special thanks are extended to the author's fiancee, Miss Jean-Marie Masson, for her typing efforts.
# TABLE OF CONTENTS

Abstract .................................................. 11
Acknowledgements ........................................... iii
List of Figures .............................................. v
I. Introduction .............................................. 1
II. Internal State Code Generation ......................... 5
   A. Partition Lists ........................................ 5
   B. Programming Considerations ......................... 12
   C. Matrix Reduction ..................................... 14
   D. An Example ........................................... 16
III. Generation of Design Equations ......................... 21
   A. Next-State Equations ................................. 21
   B. Simplification of Next-State Expressions .......... 31
   C. An Example ........................................... 40
IV. Conclusion ............................................... 45
Appendix I: Program Listings ............................... 47
Appendix II: Worked Examples ............................. 78
Bibliography ................................................ 84
Vita .......................................................... 86
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Flow table for Sequential Machine A</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>The design equation procedure</td>
<td>4</td>
</tr>
<tr>
<td>3a.</td>
<td>Partition and k-set subprogram flowchart</td>
<td>9</td>
</tr>
<tr>
<td>3b.</td>
<td>Subprogram &quot;pink&quot; flowchart</td>
<td>10</td>
</tr>
<tr>
<td>3c.</td>
<td>Partition and k-set subprogram flowchart</td>
<td>11</td>
</tr>
<tr>
<td>4.</td>
<td>Machine A flow table</td>
<td>16</td>
</tr>
<tr>
<td>5.</td>
<td>State assignment and partial flow table</td>
<td>22</td>
</tr>
<tr>
<td>6.</td>
<td>P-subcube and 1-set generation flowchart</td>
<td>25</td>
</tr>
<tr>
<td>7a.</td>
<td>Unspecified next-state generation flowchart</td>
<td>29</td>
</tr>
<tr>
<td>7b.</td>
<td>Unspecified next-state generation flowchart</td>
<td>30</td>
</tr>
<tr>
<td>8.</td>
<td>Iterative consensus subprogram flowchart</td>
<td>34</td>
</tr>
<tr>
<td>9a.</td>
<td>The cover program</td>
<td>38</td>
</tr>
<tr>
<td>9b.</td>
<td>The cover program (branching section)</td>
<td>39</td>
</tr>
<tr>
<td>10.</td>
<td>Machine A with an assignment</td>
<td>40</td>
</tr>
<tr>
<td>11.</td>
<td>Machine A design equations</td>
<td>44</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

Sequential switching circuits are commonly classified as being either synchronous or asynchronous. Clock pulses synchronize the operation of the synchronous circuit; the operation of an asynchronous circuit is usually assumed to be independent of such clocks. The operating speed of an asynchronous circuit is thus limited only by basic device speed.

The operation of a sequential switching circuit is often described by means of a flow table. The columns of a flow table represent input states, while the rows represent internal states assumed by the circuit. Each entry in the flow table specifies the next internal state and output which result from the given input and internal states. An example of a flow table appears in Figure 1. When the next state entry equals the present state, as in column I₁, state 3 of

<table>
<thead>
<tr>
<th>Column</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>f</td>
<td>6</td>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 1. Flow table for sequential Machine A.

Machine A, the state is said to be stable and is customarily
circled.

Next state entries which cause transitions among internal states are termed unstable. A flow table in which each unstable next state leads directly to a stable state is called a normal flow table.

A sequential circuit for which the input state is unchanged unless the circuit is in a stable state is said to be operating in the fundamental mode.

An important step in the synthesis of sequential circuits is the assignment of a binary code to the internal states. This paper is concerned only with state assignments for normal fundamental-mode asynchronous sequential circuits. Furthermore, codes constructed as a result of the procedures described here have the property that a single internal state is assigned to each row of the flow table.

The absence of clock pulses in asynchronous circuits introduces the problem of variations in transmission delays of signals within the circuit, which may cause an unpredictable sequence of internal states in the transition between stable states. A satisfactory state assignment requires that each internal transition always lead to the proper stable state, regardless of the relative transmission delays. Such a state assignment is said to be critical-race-free (a critical race exists when, because of unequal transmission delays, there is a possibility that the stable state reached is not the intended one).

Liu\(^2\) has described procedures for making critical-race-
free state assignments for normal fundamental-mode asynchronous circuits. A sufficient number of state variables are assigned to each flow table column so that no critical races exist in the transitions within that column. The state assignment for each flow table row is then formed by putting together all individual column assignments for that row. Any assignment formed in this manner is shown to be critical-race-free.

Liu's state assignments are not generally minimal, i.e., they do not contain the minimum number of state variables required to successfully code the internal states. Tracey\textsuperscript{3} has developed algorithms which produce minimal and near-minimal state assignments. The internal states are partitioned in a manner dependent on flow table specifications and the resulting partition list is converted into a Boolean matrix. The partition matrix is then reduced by any of three reduction procedures, one of which yields minimum-variable codes. The Tracey algorithms, however, become long and tedious when exercised manually for large flow tables.

Section II of this paper describes a series of computer programs which implement two of Tracey's algorithms. Section III describes a group of programs which, given these codes, find simplified design equations for the desired sequential circuit. The procedure which these programs follow is illustrated in the flow chart of Figure 2, page 4.

From the input flow table, a Boolean matrix is generated following the procedure described by Tracey.\textsuperscript{3} Internal state
codes are then obtained by simplifying this matrix, using either of two algorithms due to Tracey. Next-state expressions are then found for each internal state variable; each includes all of the unspecified next-states. Prime implicants for the resulting Boolean expressions are found. Finally, covers are determined for the next-state expressions and these Boolean expressions are printed as design equations.

It should be noted that several sections of the above program will, with slight modifications, be found independently useful in the solution of other logic design problems.
II. INTERNAL STATE CODE GENERATION

A. Partition Lists

The theory of partitions plays a central role in the Tracey internal state coding procedures.

Definition 1: A partition \( P \) of a set \( S \) is a collection of subsets such that their pairwise intersection is the null set. The disjoint subsets are called the blocks of \( P \).\(^3\)

Each variable \( y_j \) of a state assignment may be thought of as inducing a two-block partition \( \gamma_j \) of flow table rows, since all elements in the same block of \( \gamma_j \) are coded with the same value of \( y_j \). Consider, for example, the state assignment and induced partitions below:

<table>
<thead>
<tr>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( y_3 )</th>
<th>( y_4 )</th>
<th>( \gamma_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( \gamma_1 = (\overline{c,d}; a,b) )</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \gamma_2 = (\overline{a,d}; b,c) )</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>( \gamma_3 = (a,b,\overline{d}; \overline{c}) )</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>( \gamma_4 = (\overline{d}; a,c,\overline{d}) )</td>
</tr>
</tbody>
</table>

For such two-block partitions, it obviously does not matter which block is coded with a zero and which with a one; all assignments created by complementing or permuting one or more state variables are symmetrical codes. Clearly, if binary notation is used, a list of two-block partitions may conveniently be thought of as a Boolean matrix. Indeed, it has been found convenient to work with the two-block partitions if the list is expressed in Boolean matrix form.\(^3\)
The generation of a two-block partition list is a direct result of applying Tracey's theorem 1, which is here reproduced without proof:

**Theorem 1**: A row assignment allotting one y-state per row can be used for direct transition realization of normal flow tables without critical races if, and only if for every transition \((S_i, S_j)\):

a) if \((S_m, S_n)\) is another transition in the same column, then at least one y-variable partitions the pair \((S_i, S_j)\) and \((S_m, S_n)\) into separate blocks and

b) if \(S_k\) is a stable state in the same column, then at least one y-variable partitions the pair \((S_i, S_j)\) and the state \(S_k\) into separate blocks and

c) for \(i \neq j\), \(S_i\) and \(S_j\) are in separate blocks of at least one y-variable partition.

The first section of the computer program generates a partition list satisfying the above theorem. Input data are the dimensions of the flow table and the flow table itself. The program generates partitions satisfying parts a) and b) of Theorem 1 on a per-column basis, then checks the combined list from all flow table columns for compliance with Theorem 1, part c), generating additional partitions as required.

Partitions generated by the computer program in satisfying theorem parts a) and b) may be visualized as of three types:
The distinction between the above three partition types is made only to clarify the technique used to generate them.

Since the only flow tables considered are normal, every transition in a flow table column must involve a stable state. Thus, one state of each block of the partition types listed above must be stable. Furthermore, if there is another member of the block, it must be unstable and be in the same state as the stable member in order to effect the implied transition. Clearly, no more than one unstable state is specified per block.

The flowchart of Figure 3, page 9, illustrates the partitioning procedure. The computer program causes each flow table column to be searched for the topmost stable state in that column, say \( S_i \). State \( S_i \) is compared to a list of stable states which, in that column, have already been completely partitioned from all other stable states in the column. If \( S_i \) is found in this list, the search continues down the column. If \( S_i \) has not been partitioned from every other stable state in the column, \( S_i \) is entered in the "completely partitioned" list and the column is searched for all unstable identical states \( S_{j1}, S_{j2}, \text{ etc.} \) (if they exist). The
flow table is then searched for other stable states, with partitions being generated whenever the conditions specified above are met.

A provision has been included here to remove duplicate partitions. Experience has shown that for large partition lists, redundant data significantly slow the execution of following program sections.

In a later section, it becomes useful to know which rows in a given column of the flow table lead to each stable state.

Definition 2: A k-set of a flow table consists of all k-1 unstable entries leading to the same stable state, together with that stable state. The k-sets are conveniently generated at the same time the partitions are made, since the procedures for producing both partition lists and k-set lists may be the same.

A listing of the partition and k-set generation subprogram may be found in Appendix 1.
Figure 3a. Partition and k-set subprogram flowchart.
Figure 3b. Subprogram "pink" flowchart.
Figure 3c. Partition and k-set subprogram flowchart.
B. Programming Considerations

All of the programs described in this paper were written in the PL/1 language\textsuperscript{5} for an IBM S/360 model 40 computer having 131K bytes of core storage. PL/1 has many features not available in any other single high-level language, of which some of the most useful are listed below:

1) Flexible data specification formats.
2) Controllable storage allocation.
3) Type bitstring data with (optional) varying length.
4) Boolean operators.
5) Numerous debug assistance options.

PL/1 is, on the other hand, a relatively new language and still has some significant weaknesses. Compilers available to date are extremely slow. Implementation of PL/1 is incomplete and language documentation often does not reflect this fact. Compiler error analysis is still being improved and documentation of error messages is weak. (These problems should be solved in the near future.)

Input format for flow table data is in decimal form for user convenience. Each number is separated from the one following by a comma or a blank (PL/1 list directed input). The first and second numbers are two-digit decimal integers specifying the number of columns and rows, respectively, of the flow table. These are followed by a pair of integers for each flow table entry, the first being two digits representing a state number, and the second a single digit indicating the "stability" of the flow table entry according to the following.
code:

0 --- Unstable state  
1 --- Stable state  
2 --- Unspecified (don't care) state  

The first data card used to input the flow table of Figure 1 might read: 3, 6 1, 1 6, 0 4, 0 3, 0. Note that data for flow table entries are listed by row, i.e., data for row 1, column 1 is followed by that for row 1, column 2 etc. No distinction need be made between data for different rows, but rows are assumed to be listed in descending order (note that reordering the flow table rows does not change circuit specifications).

The partition list is stored in the form of a Boolean matrix which may be thought of as a list of pairs of bit-strings, each string having a length equal to the number of rows in the flow table, each representing one partition. Elements of the first block of the partition are coded with zeros and those of the second block with ones. Two bitstrings are required for each partition, since "don't care" entries must be distinguished from specified entries. The code selected for this "mask" representation is as follows: superposition of the two strings for a given partition yields some bit positions where the two halves disagree. These are the unspecified entries. Where the two halves agree, the entry is a specified entry in that position, and a one if both equal one there; similarly, agreeing zeros indicate a significant zero in that position. This data representation was selected
because if the first string list is initialized to all zeros, and the "mask" list to all ones, no manipulation of the strings need occur until a significant digit is recorded; only one of the two halves of the partition string need be modified in order to record the significant digit.

Internal representation of k-set data is similar to that of the partition list. Each stable state in a flow table column is assigned a pair of bitstrings, both having a length equal to the number of flow table rows. The first half is "one" in each position corresponding to the appearance of the state in the appropriate row of the given column, while the "mask" records with "one" that row of the column where the stable state appears. Both halves of the k-set string are zero elsewhere.

C. Matrix Reduction

The partition list generated as a result of applying Tracey's Theorem 1 to a flow table is related to the internal state codes by the following:

Definition 3: The two-block partitions $\gamma_1, \gamma_2, \ldots, \gamma_n$ induced by the internal state variables $y_1, y_2, \ldots, y_n$, respectively, are called the set of $\gamma$-partitions of that assignment.

One of the Tracey Boolean matrix reduction schemes used to obtain $\gamma$-partitions from the partition matrix requires that the complements of binary coded partitions also be considered, since the order of coding of the blocks of each par-
tion is arbitrary. It has been shown that for problems of this type, only complements of certain terms need be considered.

A column of the partition matrix having the least number of unspecified entries is designated the "base" column (if two or more columns have fewest unspecified entries, the first such column is arbitrarily chosen). Each partition having an unspecified entry in the base column must, in effect, have its complement added to the partition list. A bitstring of length equal to the number of partitions (program variable "unger") is set to one in bit position $n$ if the $n$th partition complement is to be considered, and zero elsewhere. In order to simplify bookkeeping, all partitions are arbitrarily complemented (if appropriate) to make all specified entries in the base column of the partition matrix zero.

Computer programs implementing two of the matrix reduction procedures described by Tracey have been written by Schoeffel and are described in detail in reference 7. The matrix reduction programs used in this paper were written by the author and Mr. Schoeffel and are based on those presented in (7). Algorithm #1 produces minimum-variable codes and algorithm #2 (which is less complex and therefore faster) produces near-minimum-variable assignments.

For many flow tables, more than one minimum-variable assignment may be found; the number of components required to construct the circuit will usually depend upon which of these codes is used. Maki has described a method for pre-
dicting which of several codes would yield simpler next-state expressions, hence the least complex circuit. The program described in this paper generates design equation sets for either the first, or all internal state codes (up to a maximum of ten codes). Data is printed which may be used in the Maki code evaluation procedure.

Sequential circuit synthesis starting with a flow table description of circuit performance requires selection of input-state codes. These input-state codes are needed to form the next-state expressions and will be discussed further in Section III. It should be mentioned here, however, that binary codes for each input-state are specified by the user and no attempt is made to optimize them.

D. An Example

The algorithm described above for generating internal state assignments may be further illustrated by considering Machine A of Figure 1, whose flow table is reproduced below. The output states are not shown.

<table>
<thead>
<tr>
<th></th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>f</td>
<td>6</td>
<td>6</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 4. Machine A flow table.
The partition list generated by applying Theorem I, parts a) and b), to the flow table is shown below:

<table>
<thead>
<tr>
<th>Partition List</th>
<th>Partition Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a, d ; b, c))</td>
<td>(0 \ 1 \ 1 \ 0 - -)</td>
</tr>
<tr>
<td>((a, \overline{d} ; \overline{f}))</td>
<td>(0 - - 0 - 1)</td>
</tr>
<tr>
<td>((\overline{b}, c ; \overline{f}))</td>
<td>(- 0 0 - - 1)</td>
</tr>
<tr>
<td>((\overline{c} ; \overline{b}, \overline{d}))</td>
<td>(- 1 0 1 - -)</td>
</tr>
<tr>
<td>((\overline{c} ; \overline{a}, \overline{f}))</td>
<td>(1 - 0 - - 1)</td>
</tr>
<tr>
<td>((\overline{b}, \overline{d} ; \overline{e}))</td>
<td>(- 0 - 0 1 -)</td>
</tr>
<tr>
<td>((\overline{b}, \overline{d} ; \overline{a}, \overline{f}))</td>
<td>(1 0 - 0 - 1)</td>
</tr>
<tr>
<td>((\overline{e} ; \overline{a}, \overline{f}))</td>
<td>(1 - - - 0 1)</td>
</tr>
<tr>
<td>((\overline{b} ; \overline{a}, d))</td>
<td>(1 0 - 1 - -)</td>
</tr>
<tr>
<td>((\overline{b} ; \overline{d}, \overline{e}))</td>
<td>(- 0 - 1 1 -)</td>
</tr>
</tbody>
</table>

Applying part c) of Theorem I, it is found that flow table row "c" is not partitioned from row "e", requiring that the following partition be added to the list:

\[ (\overline{c} ; \overline{e}) \quad -- 0 - 1 - \]

At the same time that the above partition matrix is generated, the k-sets are constructed. In column \(I_1\) of the flow table, the first stable state encountered is in row "a". The only unstable state in column \(I_1\) leading to this stable state is found in row "d"; the k-set for stable state 1, column \(I_1\) is thus: \( (\overline{a}, d) \). Internally, this data is represented as two bit strings. The first lists all elements of the k-set as ones and the second tags the stable state of
the \( k \)-set:

\[
\begin{array}{ll}
100100 & 100000 \\
\end{array}
\]

The complete list of \( k \)-sets generated for Machine A is:

\[
\begin{array}{ll}
\text{abcdef} & \text{abcdef} \\
100100 & 100000 \\
011000 & 001000 \\
000001 & 000001 \\
001000 & 001000 \\
010100 & 000100 \\
000010 & 000010 \\
100001 & 000001 \\
010000 & 010000 \\
100110 & 000100 \\
\end{array}
\]

Column two of the partition matrix has the least number of unspecified entries (four) and thus will be designated the "base" column. The bitstring "unger" will have a one in each position where column two of the partition matrix is unspecified:

\[
\text{unger} = 01001001001.
\]

Partitions having specified entries in column two are then complemented if that specified entry is one. The partition matrix for Machine A after this operation is shown below, along with the internal data representation used in the computer program:
<table>
<thead>
<tr>
<th>Partition Matrix</th>
<th>Internal Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcdef</td>
<td>abcdef abcdef</td>
</tr>
<tr>
<td>1001--</td>
<td>100100 100111</td>
</tr>
<tr>
<td>0--0-1</td>
<td>000001 011011</td>
</tr>
<tr>
<td>-00--1</td>
<td>000001 100111</td>
</tr>
<tr>
<td>-010--</td>
<td>001000 101011</td>
</tr>
<tr>
<td>1-0--1</td>
<td>100001 110111</td>
</tr>
<tr>
<td>-0-01-</td>
<td>000010 101011</td>
</tr>
<tr>
<td>10-0-1</td>
<td>100001 101011</td>
</tr>
<tr>
<td>1--0-01</td>
<td>100001 111101</td>
</tr>
<tr>
<td>10-1--</td>
<td>100100 101111</td>
</tr>
<tr>
<td>-0-11-</td>
<td>000110 101111</td>
</tr>
<tr>
<td>--0-1-</td>
<td>000010 110111</td>
</tr>
</tbody>
</table>

Using the partition matrix shown above, Schoeffel's implementation of Tracey's Algorithm #1 obtains the following irredudant internal state assignments:

<table>
<thead>
<tr>
<th>I</th>
<th>IV</th>
<th>VII</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>100111</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$y_2$</td>
<td>000011</td>
<td>$y_2$</td>
</tr>
<tr>
<td>$y_3$</td>
<td>101001</td>
<td>$y_3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>II</th>
<th>V</th>
<th>VIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>100111</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$y_2$</td>
<td>001011</td>
<td>$y_2$</td>
</tr>
<tr>
<td>$y_3$</td>
<td>101001</td>
<td>$y_3$</td>
</tr>
</tbody>
</table>

(Continued)
Schoeffel's reduction program #2 uses the partition matrix form (mask coded) shown to the right of the partition list on page 17. For Machine A, the following codes are found using matrix reduction Algorithm #2:

<table>
<thead>
<tr>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>100111</td>
<td>( y_1 )</td>
</tr>
<tr>
<td>( y_2 )</td>
<td>101001</td>
<td>( y_2 )</td>
</tr>
<tr>
<td>( y_3 )</td>
<td>00-011</td>
<td>( y_3 )</td>
</tr>
<tr>
<td>( y_4 )</td>
<td>0--0-1</td>
<td>( y_4 )</td>
</tr>
</tbody>
</table>

Section II has described programs which find satisfactory state assignments; these codes may then be used to generate design equations.
III. GENERATION OF DESIGN EQUATIONS

In order to most efficiently construct a sequential circuit using the assignments generated by the programs described in Section II, simplified design equations must be found. The programs described in this section find non-minimal next-state expressions; prime implicants for these expressions are then generated. Finally, simplified (but not necessarily minimal) covers for the next-state 1-cells yield the desired design equations.

A. Next-State Equations

Given an internal state assignment for a sequential circuit, the next step in the synthesis procedure is to find next-state equations for each state variable. Next-state equations have, in the past, been found by constructing transition tables.\(^1\) For fundamental mode asynchronous circuits, however, the transition table is often significantly larger than the flow table.

Maki\(^4\) has described a procedure for finding next-state equations without constructing a transition table. Maki's algorithm and a computer program based on it will be discussed below.

If there are \(n\) variables in a given state assignment, then let an \(n\)-cube represent all possible internal states under that assignment. Now let a \(p\)-subcube of that \(n\)-cube define all internal states possibly entered in a transition (within a single column) between any unstable state and its
corresponding stable state. For example, given the partial flow table and state assignment shown in Figure 5,

![Figure 5. State assignment and partial flow table.](image)

the following "transition pairs" of states are defined for column I₂:

(a,d) ; (b,c) ; (f).

Corresponding to each of these transition pairs, a p-subcube is defined, representing all possible internal states entered during the transition:

\[
\begin{align*}
10- & \quad \{a,d\} \\
0-- & \quad \{b,c\} \\
111 & \quad \{f\}
\end{align*}
\]

Note that wherever digits of the flow table row assignments for elements of the transition pair agree, the associated p-subcube is specified and equal to this unchanging digit. If a state variable changes during a transition, it is unspecified in the p-subcube identified with that transition.

The group of p-subcubes due to a single k-set represents a unique set of specified states in a particular flow table.
No state may be a member of more than one p-subcube group, since this would imply two different next-state entries for that internal state. Within a flow table column, then, the total number of specified states equals the sum of all internal states represented in the p-subcubes of that column. If there are $r$ internal state variables and $S_1$ internal states specified by the p-subcubes, then the number of internal states unspecified in a flow table column is: $2^r - S_1$.

The sum of all p-subcubes of a flow table column represents all internal states having specified next-state entries. Furthermore, the logical complement of this expression represents the unspecified states for the given column; in fact, the unspecified next-states will be found in precisely this manner.

It is also necessary to obtain the 1-cells for each next-state variable in each flow table column. All internal states represented by a single p-subcube will have the same next-state entry, namely that of the stable state of the associated transition pair. If this next-state entry for the stable state equals one in digit $j$, then the Boolean product represented by the p-subcube is a term in the sum-of-product expression for next-state variable $Y_j$, in column $I_m$. All such terms appearing in the list of products for a given next-state variable will hence be referred to as 1-sets. (Note that a 1-set may be composed of one or more 1-cells on a Karnaugh map.)

The computer program implementation of the procedures briefly described above is designed to find all 1-sets and
unspecified terms for all next-state variables for a single flow table column. These terms, when combined with the appropriate user-specified input-state codes, yield the desired next-state equations.

The flow chart of Figure 6, page 25, will be useful in the following discussion of subcube generation. Data representing all transition pairs of a flow table column are input to the next-state expression subprogram in the form of k-sets (recall that these k-sets were found during the flow table partitioning procedure). One of the state assignments generated by the Boolean matrix reduction subprograms is also input to this section.

For each k-set, the stable state is found and the state assignment for the corresponding flow table row is constructed. The program then iterates through all unstable states within the k-set, thus defining all transition pairs of that k-set. The row code for each unstable state is constructed and a simple Boolean operation between the row codes of the stable and unstable states yields the p-subcube for the transition pair under consideration. This p-subcube is added to the list of p-subcubes for the flow table column. The p-subcube is concatenated with the input state code for the flow table column. This term is added to the l-set lists of those state variables equaling one in the row code of the transition pair stable state.

When all transition pairs of the k-set have been thus considered, the program steps to the next k-set, until all
Figure 6. P-subcube and 1-set generation flowchart.
p-subcubes and l-sets have been found for the specified flow table column and state assignment.

As was mentioned previously, unspecified next-state entries for a flow table may be determined by complementing the Boolean sum of that column's p-subcubes. Although methods have been devised to perform complementation of Boolean sums of products, investigation showed that none of these were well suited for inclusion in the series of computer programs described in this paper.

A brief review of the Boolean algebra concepts used in the complementation procedure is given below. The scheme is characterized by simplification during the multiplication rather than the conventional complete multiplication followed by simplification.

A general Boolean sum-of-products expression may be represented by

\[ f = P_1 + P_2 + \ldots + P_n \]  

(1)

where each \( P_i = Y_{i1}Y_{i2} \ldots Y_{ij} \ldots Y_{im} \) is a product of at most \( m \) variables. The complement of this expression may be written

\[ f' = q = (P'_1)(P'_2) \ldots (P'_n) \]  

(2)

where \( P'_i = Y'_{i1} + \ldots + Y'_{ij} + \ldots + Y'_{im} \). Alternatively, (2) may be written

\[ q = (H)(Y'_{n1} + \ldots + Y_{nj} + \ldots Y'_{nm}) \]  

(3)

where

\[ H = (P'_1)(P'_2) \ldots (P'_{n-1}) \].  

(4)
Some terms of the complemented product of sums may be eliminated before multiplying (3), if this equation is rewritten in the form
\[ q = H(Y'_{n1}) + \ldots + H(Y'_{nj}) + \ldots + H(Y'_{nm}) \]  
and the Boolean algebra theorem illustrated by
\[ A(A + B + C) = A \]  is applied. For example, if some sum of (4) contains a literal equal to \( Y_{nj} \), then that sum may be deleted from the list of those sums to be multiplied by \( Y_{nj} \).

The multiplication implied by a product of sums Boolean expression may be further simplified by application of the Boolean algebra theorem illustrated by
\[ A(A' + B + C) = A(B + C). \]

Each sum to be multiplied by some literal \( Y_{nj} \) which contains its complement \( Y'_{nj} \) may have that literal removed from the sum term without changing the result.

The computer program used to complement the sum of p-subcubes is illustrated by the flowchart of Figure 7, pages 29 and 30. It first complements the sum expression and expands it into the form of equation (5) using the significant literals of the last p-subcube as coefficients. The list of remaining sum terms is then examined for application of the theorems illustrated by equations (6) and (7); modifications or deletions are incorporated in a modified product of sums.
to be multiplied by the appropriate coefficient $V_{nj}$. A sum of products form of the expression for unspecified next-states is then found by conventional Boolean multiplication.

The list of p-subcubes is stored in data array "psub". Array "ones" stores products of the form found in equation (6); each column is headed by a product term containing a complemented literal from the last p-subcube. Following elements of each column of "ones" represent sum terms of the product to be multiplied by the column's topmost entry. These sum terms are examined prior to entry in the column for modification or deletion according to equations (6) and (7).

If there are "n" p-subcubes for a flow table column, the nth of which contains "m" significant literals, then "m" columns of array "ones" will be used and each column may contain at most "n" terms. After loading of a column of array "ones", multiplication of the product and sum terms therein is begun. The product term at the top of the "ones" column is loaded into array "scratch". The program then iterates through all remaining terms in the column, multiplying the present contents of "scratch" by all significant literals in each sum term and placing each resulting product in "scratch". An inclusion and duplication check insures that only non-redundant products are listed in "scratch".

After multiplication of "scratch" by the last sum term in a column, the resulting product terms are entered in the list "dcare", which contains all unspecified product terms generated to date. When all columns of array "ones" have been multiplied,
Figure 7a. Unspecified next-state generation flowchart.
Figure 7b. Unspecified next-state generation flowchart.
duplications and inclusions within the list "dcare" are eliminated. The list of unspecified products is then combined with the appropriate input state code and these terms are added to the list "dontcar", which accumulates unspecified next-state entries for the entire flow table.

The program listing corresponding to the above description is in Appendix 1. It should be noted that this portion of the program solves a problem often encountered in logic design and Boolean algebra and may, with slight modification, be used in other applications.

The procedure described above is repeated for each column of the input flow table.

B. Simplification of Next-state Expressions

The computer program implementation of Maki's procedure yields a sum-of-products Boolean expression for the unspecified next-states due to the application of a state assignment to the input flow table. Sum-of-products expressions are also found for each next-state variable. Minimum form design equations may then be found by combining the unspecified next-state expression with each of the next-state variable l-set expressions, then finding minimal forms of the resulting sums.

Several procedures for obtaining minimal sum forms of Boolean functions have been described. One of the most well known is the map method due to Karnaugh. This procedure works well only for problems having less than six variables and is not directly programmable.
Harris\textsuperscript{9} relates an n-variable Boolean expression to an n-cube, describing subcubes of this n-cube using ternary notation. Products of the given Boolean sum expression are converted to this ternary notation and the list of ternary numbers thus produced is operated on to produce minimal sums. A computer program based on Harris' procedure has been described by Butler and Warfield,\textsuperscript{10} but the program was not machine- or language-compatible with the other programs described in this paper.

Another simplification procedure involves finding prime implicants of the given expression, then finding minimal covers for the specified 1-cells using only prime implicant terms. This method was selected as the basis of the program to be used in the present case since it requires no internal data conversion and is easily programmable.

The prime implicant form of the Boolean expression representing a next-state variable is generated using iterative consensus. The iterative consensus algorithm is briefly reviewed below; a more complete description is given on pages 165-174 of (11).

**Definition 3.** Let $P = x y_1 y_2 \ldots y_n$ and $Q = x' z_1 z_2 \ldots z_n$, where it is possible that $y_i = z_j$ for some $i$ and $j$. The consensus of $P$ and $Q$, written $P \# Q$, is defined to be $y_1 y_2 \ldots y_n z_1 z_2 \ldots z_m$ (with any repeated literals removed) unless $y_i = z'_j$, in which case the consensus is said not to exist.
Note that in order for $P \not\subset Q$ to exist, one and only one literal in $P$ must appear complemented in $Q$. It has been shown that the successive addition of consensus terms to a sum of products expression and the removal of terms included in other terms will result in a complete sum.\textsuperscript{10}

The flow chart of Figure 8, page 34, will illustrate the iterative consensus program explained below. The unspecified and 1-set products of a next-state variable are loaded into array "prime". The number of elements in "prime" at the beginning of each iteration is stored in variable "oterm". Each of the first $oterm-1$ products in "prime" are compared to all following terms to determine if redundancy exists, or consensus is defined between the product pair according to definition (3). Where $P \not\subset Q$ is defined, it is calculated. An inclusion check is immediately performed; if any existing term includes or duplicates $P \not\subset Q$, the new consensus term is dropped and the search for new consensus terms continues. If $P \not\subset Q$ is not included in any existing product, the new term is recorded. As the inclusion check is performed, all existing terms included in the new consensus term are marked and in effect are removed from further consideration. (At the end of each pass, all such included terms are removed from the product list "prime".)

A pass is completed when all possible combinations of terms existing at the beginning of the pass have been considered. Array "prime" contains the desired complete sum when a pass is completed during which no irredundant new terms are
Figure 8. Iterative consensus subprogram flowchart.
generated. (It can be shown that at most "m" passes will be required to generate the complete sum if "m" equals the number of variables appearing in the problem.)

After the complete sum of the next-state Boolean expression has been found, the classical covering problem remains. That is, a minimal sum may be constructed only from terms in the complete sum, which includes all of the 1-cells of the original function (often more than one such minimal sum exists).

The prime implicant table method of solving the covering problem is due to McCluskey.12 A table having "n" rows and "m" columns is constructed for a problem having "n" prime implicants and "m" 1-cells. A mark is placed in block \((q,r)\) of the table if prime implicant "q" includes 1-cell "r". Rows which uniquely cover any column are called essential rows and must appear in all minimal sums. Row and column dominances are used to reduce the table remaining when essential rows and all "included" columns are removed. A prime implicant table which cannot be completely reduced in this manner is often solved by a branching technique.

Another method for solving the covering problem is due to Petrick.13 A prime implicant function (or "p-function") is formed by assigning variable names to each of the prime implicants and constructing Boolean sums of those variables "covering" each 1-cell; these sums are then converted to minimum sum-of-product form. Each product term of the p-function represents a cover. Minimal covers are those products containing the least number of variables, hence the fewest prime
implicant terms.

It may be shown that the covering problem described here closely resembles that of the first state assignment scheme described by Tracey and implemented by Schoeffel. Experience with the latter covering programs has shown that for covering problems comparable to twenty prime implicants or larger, exactly minimal solutions may not be practical.

It has been found that, given the prime implicants of a Boolean next-state sum, a form much simpler than the original expression may usually be found by covering the 1-sets rather than the 1-cells. A survey of the worked examples of Appendix II shows that this method reduces average literal count by more than 50% for generated next-state equations.

Also, it may be shown that this method eliminates static hazards associated with transitions within a flow table column. At present, "horizontal transition" static hazards must be manually removed from the programs' output. This task has proven to be trivial for all examples run to date.

The covering algorithm used in this paper is a combination of the McCluskey and Petrick methods described above. A Boolean matrix resembling the "prime implicant table" records the inclusion of 1-sets in prime implicant terms. Essential rows are then removed along with all 1-sets "covered" by these terms.

A branching method similar to Petrick's method is then used to find minimal covers for the remaining 1-sets. Instead of finding all covers, however, only covers of length less
than or equal to the length of the shortest previously discovered cover are considered. As soon as this maximum number of prime implicant terms is exceeded in the branching search for covers, or a cover is found, the program goes back to the last branch point and attempts to form a cover by selecting a previously untried path. When all branches at a given level have been exhausted, a "backup" causes alternate branching from the next higher level.

The branching process continues until all irredundant shortest covers have been constructed and all other covers discarded. The Boolean expressions represented by these covers are generated, converted from bitstring to literal notation, and printed as the design equations for a single internal state variable. A flow chart of the covering problem program is shown in Figure 9, pages 38 and 39.
Input Primes, 1-sets → A: Loop Through 1-sets → Oflag = 0, Noepi = 0 → B: Loop Through Primes

String (A,B) = 0 → Prime Covers 1-set?

Yes → Increment Noepi, Oflag = 1, Subscr = B → String (A,B) = 1

No → Print Error Message: Halt

Is Oflag 0?

Yes → End B?

No → Is Noepi 1?

Yes → Pflag = 1, Oflag = 1 → End A?

No → Noepi = 0

C: Loop Primes

Is Pflag(C) 1?

Yes → Relocate Prime Element

No → Increment Noepi, Load into Eprime

End C?

Yes

D: Loop 1-sets

Is Oflag(D) 1?

Yes → Relocate 1-set String (D,*) → End D?

No

Figure 9a. The cover program.
Figure 9b. The cover program (branching section).
C. An Example

The following table of Machine A is reproduced below, together with one of the state assignments found by the programs described in Section II.

<table>
<thead>
<tr>
<th>Row</th>
<th>Y1 Y2 Y3</th>
<th>I1 - 00</th>
<th>I2 - 01</th>
<th>I3 - 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1 0 1</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>0 0 0</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>0 0 1</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>d</td>
<td>1 0 0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>1 1 0</td>
<td>-</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>f</td>
<td>1 1 1</td>
<td>6</td>
<td>6</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 10. Machine A with an assignment.

The list of k-sets for Machine A may be found on page 18 of Section II. Specified next-state p-subcubes are found in the manner described above. For instance, the first k-set in column 1 is

\[
100100 \quad 100000
\]

defining the transition pair

\( (\bar{a}, d) \).

The p-subcube associated with this transition pair is easily seen to be

\[
\begin{align*}
101 & \quad \text{row "a" code} \\
100 & \quad \text{row "d" code} \\
10- & \quad (\bar{a}, d) \text{ p-subcube}
\end{align*}
\]

In a like manner, the complete list of specified next-state
(p-subcubes) for Machine A is found to be:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>00-</td>
<td>-00</td>
<td>10-</td>
</tr>
<tr>
<td>111</td>
<td>110</td>
<td>1-0</td>
</tr>
<tr>
<td></td>
<td>1-1</td>
<td></td>
</tr>
</tbody>
</table>

As each of these p-subcubes is generated, it is combined with the appropriate input state for a flow table column. This term is added to the lists of 1-sets for those state variables coded one in the row of the transition pair stable state. The list of 1-sets after completion of this task is shown below:

<table>
<thead>
<tr>
<th>$Y_1$</th>
<th>$Y_2$</th>
<th>$Y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-00</td>
<td>11100</td>
<td>10-00</td>
</tr>
<tr>
<td>11100</td>
<td>11001</td>
<td>00-00</td>
</tr>
<tr>
<td>-0001</td>
<td>1-101</td>
<td>11100</td>
</tr>
<tr>
<td>11001</td>
<td>1-101</td>
<td>00101</td>
</tr>
<tr>
<td>1-101</td>
<td>1-101</td>
<td>1-101</td>
</tr>
<tr>
<td>10-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Boolean sum of products representing p-subcubes is complemented and converted to the form of equation (5). After application of the theorems illustrated by equations (6) and (7), the Boolean product to be multiplied is stored in array "ones". For example, the Boolean product for flow table column 2 would be:

Ones (1, -):     Ones (2, -):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0--</td>
<td>--0</td>
</tr>
<tr>
<td>-10</td>
<td>-1-</td>
</tr>
<tr>
<td>-11</td>
<td>00-</td>
</tr>
</tbody>
</table>
After completion of the multiplication algorithm, the following unspecified sum term will be found for column 2:

\[ 010 = A'BC' \]

The complete list of unspecified terms found by summing all column terms, is:

\[
\begin{align*}
-11 & \\
01-00 & \\
-1000 & \\
01-01 & \\
01-10 & \\
0-110 & \\
-1110 & \\
\end{align*}
\]

Expressions for the next-state variables are found first by combining the above list of don't-care terms with the 1-sets for the next-state variable. For instance, the list of these terms for next-state variable \( Y_2 \) is:

\[
\begin{align*}
1) & \quad -11 & 6) & \quad 0-110 \\
2) & \quad 01-00 & 7) & \quad -1110 \\
3) & \quad -1000 & 8) & \quad 11100 \\
4) & \quad 01-01 & 9) & \quad 11001 \\
5) & \quad 01-10 & 10) & \quad 1-101 \\
\end{align*}
\]

Given this next-state expression, the complete sum form must be formed using iterative consensus. Note that \( 1 \not\sim 2, \)

\[
-11 \\
\not\sim 01-00
\]

does not exist, since these two terms disagree in more than one position. Likewise, \( 1 \not\sim 3 \) does not exist: however, \( 1 \not\sim 4 \)
exists and is not included in any other term present. It is thus added to the list (note that 1 φ 4 includes term 4 and term 4 will thus be deleted from the list of terms). The complete sum form of next-state equation Y is:

\[
\begin{align*}
\phi &= \frac{---ll}{0l-0l} \\
= &= 0l--l
\end{align*}
\]

The prime implicant table subprogram finds that one of these terms is essential; namely

\[
1-l-l-l
\]

This term must be included in each minimal expression for next-state variable \( Y_2 \).

1-sets remaining to be covered are: 11100 and 11001. Both are included in prime implicant term \(-1-0-\); no other equally short cover is possible. The minimal design equation for next-state \( Y_2 \) is thus: \(-1-0- + 1-l-l-l\) or, in literal form,

\[
Y_2 = (y_2v' + y_1y_3w)
\]

where the input-state variables are "v" and "w". Design equations for example Machine A (using code 1 above) are listed in Figure 11.
\[ Y_1 = (y_3'w + y_1) \]
\[ Y_2 = (y_2v' + y_1y_3w) \]
\[ Y_3 = (y_3w + v'w') \]

Figure 11. Machine A design equations.

Section III has described a set of computer programs which, given a flow table and a proper state assignment for it, find simplified design equations for each internal state variable. The algorithms used do not involve construction of transition tables or Karnaugh maps.

Due to both hardware and software limitations, the set of programs described in this paper have not been combined into a single program. Instead, a supervisory program is executed which calls for execution of the proper sequence of subprograms. The total synthesis procedure is thus accomplished by a supervisory and seven overlayed subprograms. Since this group of programs (totaling about 1800 statements) requires more than 30 minutes to compile, it is practical to run it only in machine language form from magnetic tape or disc files. Execution times are, of course, quite dependent on problem complexity; representative examples of various sizes, their results, and execution times are given in Appendix II.
IV. CONCLUSION

The synthesis procedure for complex sequential circuits is often time-consuming and costly when performed manually. This paper has described a series of computer programs which finds simplified design equations for asynchronous sequential circuits operating in the normal fundamental-mode.

Input data include a machine description in the form of a flow table and input state codes. Internal state assignments are generated from the machine specifications using procedures described by Tracey.\textsuperscript{3} The program user may choose to use either a relatively slow minimal code generator, or a faster near-minimal code program. Non-minimal equations are then found for the circuit's next-state variables using techniques developed by Maki.\textsuperscript{4} Transition tables are not constructed. Iterative consensus is used to find prime implicants of the next-state equations. A branching technique is used to find an approximate solution to the classic "prime implicant table" problem, yielding simplified but usually non-minimal design equations for the desired circuit.

As shown in Appendix II, the use of this program set in the synthesis of complex sequential circuits has exposed a need for more efficient prime implicant generation algorithms. Further investigation also needs to be done in improvements to covering procedures. A procedure for eliminating "horizontal transition" static hazards would be a practical addition. More efficient Boolean sum simplification algorithms would
also be useful. Finally, procedures for generating output-state design equations should be investigated.

The computer programs described in this paper will perform the synthesis operations for normal fundamental-mode asynchronous sequential circuits, resulting in considerable time and cost savings. They are also capable of generating near-minimal designs for problems of this type that are too large to solve manually, and may thus result in significant hardware-associated savings.
APPENDIX I: PROGRAM LISTINGS

A listing of the complete set of computer programs follows. (Note that job control cards establishing the overlay scheme are not included.) The system of programs contains approximately 1800 PL/1 statements and requires about 31 minutes to compile on a S/360 model 40 using the Version 2 release 11 compiler. In object deck form, the programs occupy about 3200 cards. Execution times for various designs may be found in Appendix II, Worked Examples.

WLSRJS: PROC /* ASYNCHRONOUS SEQUENTIAL CIRCUIT PROGRAM CALLS SUBPROGRAMS. VERSION.A */ OPTIONS(MAIN); ********************************************** /* WLSRJS: A PROGRAM WHICH SYNTHESIZES MINIMAL */ /* ASYNCHRONOUS SEQUENTIAL CIRCUITS. INPUT IS */ /* FLOW TABLE SPECIFICATION OF CIRCUIT PERFORMANCE, */ /* AND OUTPUT IS MINIMAL AND-OR IMPLEMENTATION */ /* DESIGN EQUATION SETS. SEVERAL MODES OF OPERATION* */ /* ARE OPTIONAL. */ /* THE PROGRAM IS WRITTEN IN PL/1, FOR A S/360.440 */ /* AND CONSISTS OF A MAIN PROGRAM AND SEVERAL SUB- */ /* PROGRAMS; SUBPROGRAMS MUST BE OVERLAYERED TO FIT */ /* CORE SIZE 131K. DATA STORAGE REQUIREMENT TOTALS */ /* ABOUT 25K BYTES MAXIMUM. CERTAIN PORTIONS OF */ /* THIS PROGRAM MAY, WITH SLIGHT MODIFICATION, BE */ /* USED TO SOLVE OTHER LOGIC DESIGN PROBLEMS. */ ********************************************** PUT EDIT('TIME ON ENTRY TO MAIN SYNTHESIS PROGRAM =', TIME)(SKIP(9),A,A); PUT EDIT('INTERNAL STATE VARIABLE DESIGN EQUATION ', 'DETERMINATION WILL BE MADE USING FIRST 10 ', 'ALFA LETTERS FOR INTERNAL STATE VARIABLES ', 'AND LAST 10 FOR INPUT STATE VARIABLES') SKIP(4),A,A,SKIP(1),A,A; DECLARE BITCODE BIT(5) EXTERNAL, (ROWMUX(AK,ROWMAX),MASK(AK,ROWMAX)) BIT(1) CONTROLLED PACKED EXTERNAL, (SIZEL,MAXNOK,IKJL,I,J,ISV) BIN FIXED(15) EXTERNAL, (ROWMAX,COlMAX) DEC FIXED(3) EXTERNAL, KSET(*,*,*) BIT(ROWMAX) CONTROLLED EXTERNAL PACKED, UNGER BIT(PINO) CONTROLLED VARYING EXTERNAL, ROW(PINO) BIT(2*ROWMAX) CONTROLLED EXTERNAL
PACKED,
CLASS(5*PINO*AV,2) BIT(PINO) CONTROLLED
PACKED EXTERNAL,
NUMMAX BIN FIXED(15) EXTERNAL,
MINCOV(AP) BIN FIXED(7) CONTROLLED EXTERNAL,
NUMCOV BIN FIXED(7) EXTERNAL,
(PTERMS,TERMS) BIN FIXED(15) EXTERNAL,
{VAR,NOL} BIN FIXED(15) EXTERNAL,
CODE(*,*,*) BIT(ROWMAX) CONTROLLED EXTERNAL
PACKED,
INCOD(*,*) BIT(INDIG) CONTROLLED EXTERNAL,
DNO BIN FIXED(15) INITIAL(0) EXTERNAL,
DONTCAR(*,*) BIT(NOL) CONTROLLED EXTERNAL
PACKED,
ONESCAT(*) BIN FIXED(7) CONTROLLED EXTERNAL,
ONESUR(*,*,*) BIT(NOL) CONTROLLED EXTERNAL
PACKED,
PRTME(*,*) BIT(1) CONTROLLED EXTERNAL PACKED,
HEAD CHAR(RO) CONTROLLED,
OK BIT(1) EXTERNAL,
{AR,AS,AT,AU, AV,AW,AX,AY,AZ} DEC FIXED(4,1)
EXTERNAL,
PIND DEC FIXED(3) INITIAL(0) EXTERNAL;
OK = '1'B;
/*****************************/
/* BITCODE KEY */
/* 1ST DIGIT: =1, 1ST ROWLMX RED: =0, 2ND RED */
/* 2ND DIGIT: =1, ONLY REST CODE: =0, ALL CODES */
/* 3RD DIGIT: =1, EXIT AFTER CODE GEN: =0, CONTINUE */
/* 4TH DIGIT: =1, EXIT AFTER NSE GEN: =0, SIMPLIFY */
/* 5TH DIGIT: =1, ONLY 1 NSE PTR ISV: =0, MAX/AT NSE */
/*****************************/
/* COEFFICIENT KEY */
/* AR = 10 = MAX NO OF ASSIGNMENTS PRESERVED */
/* AS = 7.5 = COEFF/NO COVER TERMS:AS*TERMS.1 */
/* AT = MAX NO ISV'S ALLOWED IN CODE GEN'S 10 */
/* AU = EVALUATION COEFF IN RJSOPT = 2. */
/* AV = 5 = COEFF NO CLASSES IN PJSMAX: 5*PINO*AV */
/* AN = 3.0 = COEFF MAX SIZE DCARE = JVAR*AW*KSMAX */
/* AX = 75 = MAX NO PARTITIONS */
/* AY = 7. = SIZE COEFF FOR PI ARRAY: AY**INPUT */
/* AZ = COEFF FOR F.T. UNSPEC TERMS = AZ*(NOL**2) */
/*****************************/
/* 1ST CARD = HEADING, OR INSERT BLANK IF UNWANTED */
ALLOCATE HEAD; GET EDIT(HEAD)(A(80));
PUT EDIT(***... ',HEAD, ' ...***)(PAGE,A,A,A);
FREE HEAD;
GET LIST(BITCODE,AR,AS,AT,AU, AV,AW,AX,AY,AZ);
PUT DATA(BITCODE,AR,AS,AT,AU, AV,AW,AX,AY,AZ);
GET LIST (COLMAX,ROWMAX); IJKL = 0;
CALL RJSOPT;
IF OK = 'O'B THEN GO TO END_SS; ELSE;
IF SUBSTR(BITCODE,1,1) = 'O'B THEN SS00: CALL WLSCG2;
ELSE SS16: DO;
CALL RJSMAX;
IF OK = '0' THEN GO TO END_SS;
CALL WLSCOV;
FREE ROW,UNGER;
END SS16;
IF OK = '0' THEN GO TO END_SS;
IF SUBSTR(BITCODE,2,1) = '1' THEN CALL RJSORT;
/* IF 2ND DIGIT OF BITCODE = 1 ONLY PREDICTED BEST */
/* ASSIGNMENT YIELDS DESIGN EQUATIONS */
IF SUBSTR(BITCODE,3,1) = '1' THEN GO TO END_SS;
IF SUBSTR(BITCODE,5,1) = '1' THEN AT = 1;
GET LIST(INDIG,INDCNO);
ALLOCATE INCOD(COLMAX+INDCNO,2);
GET LIST((INCOD(I,1B),INCOD(I,10B) DO I=1B TO COLMAX+INDCNO));
PUT EDIT('INPUT STATES INPUT ARE:')(SKIP(4),A)(
('COLUMN ' ,I', ',INCOD(I,1B),INCOD(I,10B) DO
I = 1B TO COLMAX))(SKIP(2),A,F(2),A,
COLUMN(15),B,SKIP(0),COLUMN(15),B);
DO I = 1B TO NUMCOV; /* LOOP THRU CODES */
IF IJKL = OB THEN IF I > IJKL THEN GO TO END_SS;
ELSE I = IJKL;
PUT EDIT('**** DESIGN EQUATION GENERATION FOR CODE',
', 'I', ' ' FOLLOWS: '****')(LINE(20),A,A,F(2),A)
JVAR = MINCOV(I); NOL = JVAR + INDIG;
ALLOCATE ONESCNT(JVAR);
ONECNT = OB;
ALLOCATE DONTCAP(A*(NOL**2),10B):
DON = 0B;
DO IX = 1B TO INDCNO; DON = DON +1B;
DONTCAR(DON,1B) = '0'8;
DONTCAR(DON,10B) = -DONTCAR(DON,1B);
SUBSTR(DONTCAR(DON,1B),JVAR+1B) = INCOD(COLMAX+IX,1B);
SUBSTR(DONTCAR(DON,10B),JVAR+1B) = INCOD(COLMAX+IX,10B);
END;
ALLOCATE ONESUB(I,JVAR,COLMAX*MAXNOK,2);
DO J = 1B TO COLMAX;
CALL RJSMTK;
IF OK = '0' THEN GO TO END_SS;
END; /* END OF COL LOOP */
IF SUBSTR(BITCODE,4,1) = '1' THEN GO TO END_CODE;
DO ISV = 1B TO JVAR; /* LOOP THRU STATE VARIABLES */
CALL RJSCHS;
IF OK = '0' THEN GO TO END_SS;
III = I;
OTERMS = ONESCNT(ISV);
CALL RJSPIAT;
NOL = JVAR + INDIG;
I = III;
IF OK = '0' THEN GO TO END_SS;
END; /* END ISV LOOP */
END_CODE: END;
FREE KSET;
RJSOPT: PROC;  /* ROUTINE FOR EVALUATING AND SELECTING BEST CODE */
/* GOES HERE. TEMPORARILY 'SELECT' FIRST CODE... */
IJKL = 18;
END RJSOPT;
END_SS: END WLSRJS;

RJSOPT: PROC;  /* VERSION 6 */
PUT EDIT('TIME ON ENTRY TO RJSOPT V.6 BOOLEAN MATRIX ',
' GENERATOR =',TIME)(SKIP(2),A,A,A);

DECLARE /* DATA INPUT AND OUTPUT THIS ROUTINE */
(SIZE1,MAXNOK ) BIN FIXED(15) EXTERNAL,
PINO DEC FIXED(3) INITIAL(0) EXTERNAL,
ROW(PINO) BIT(2*ROWMAX) CONTROLED EXTERNAL PACKED,
UNGERT(PINO) CONTROLED VARYING EXTERNAL,
KSET(*,*,*) BIT(ROWMAX) CONTROLED EXTERNAL PACKED,
(ROWMAX,COLMAX) DEC FIXED(3) EXTERNAL,
(ROOLMX(AX,ROWMAX),MASK(AX,ROWMAX)) BIT(1)
CONTROLED PACKED EXTERNAL,
BITCODE BIT(5) EXTERNAL,
OK BIT(1) EXTERNAL INITIAL('1'B),
AX DEC FIXED(4,1) EXTERNAL;

DECLARE (I,K,ST1,ST2,P1) DEC FIXED(2),
STATE(ROWMAX, COLMAX) DEC FIXED(2) CONTROLED,
STAB(ROWMAX,COLMAX) DEC FIXED(1) CONTROLED,
SAVE(20) DEC FIXED(2) CONTROLED,
BSTRING (?,AX) BIT( AX) VARYING CONTROLED,
(X,NOUNSTAR) BINARY FIXED(1);

ALLOCATE STATE, STAR, ROOLMX, MASK, SAVE ;
SAVE=0;
AX = AX + 1;
PUT EDIT('THE INPUT FLOW TABLE:' )(SKIP(4),A);
DO I=1 TO ROWMX ; DO J=1 TO COLMAX ;
GET LIST(STATE(I,J),STAR(I,J));
IF STAR(I,J) = 1 THEN SAVE(J) =SAVE(J) + 1 ; END;
PUT EDIT(' ')(SKIP(1), X(4), A(1)) ((STATE(I ,M),
STAR(I,M)) DO M = 1B
TO COLMAX)(X(5),F(2),X(2),F(1)); END;
DO I= 1 TO COLMAX ; PINO = MAX(PINO, SAVE(I)); END;

END_SS: END WLSRJS;
/* COLUMN(I) CONTAINS ANY ELEMENT OF KSFT(J) */
BOOLMX = '0'B;    MASK = '1'B;    KSFT = '0'B;
COL_LOOP: DO I = 1 TO COLMAX; /* GEN PER COL PARTNS */
    IK = 1;    SAVE = 0;
    ROW_LOOP: DO J = 1 TO ROWMAX; /* FIND STAR STATES */
        IF STA8(J,IK) = 1 THEN RED: DO;
        ST1 = STATE(J,IK);
        DO N = 1 TO IK-1;
            IF ST1 = SAVE(N) THEN DO;
                SUBSTR(KSET(1,N,1),J,1) = '1'B;
                SUBSTR(KSET(1,N,2),J,1) = '1'B;
            GO TO END_ROW_LOOP; END; END;
        SAVE(IK) = ST1;    IK = IK + 1;
        SUBSTR(KSET(1,IK-1,1),J,1) = '1'B;
        SUBSTR(KSET(1,IK-1,2),J,1) = '1'B;
        /* FOUND TOPMOST STAB STATE LEFT IN COLUMN */
        NOUNSTAB = 1B;
        WHITE: DO K = 1 TO ROWMAX;
            IF((STA8(K,I) = 0) & (STATE(K,I) = ST1)) THEN DO;
                NOUNSTAB = OB;
                SUBSTR(KSET(1,IK-1,1),K,1) = '1'B;
                ELSE GO TO END_WHITE; CALL PINK;
            END;
            SUBSTR(KSET(1,IK-1,2),K,1) = '1'B;
            IF ST2 = STATE(K2,I) THEN BLACK: DO;
            Y:DO NN = 1 TO IK-1;
                IF STATE(K2,INN) = SAVE(NN) THEN
                    GO TO END_GREEN; END;
            ST2 = STATE(K2,I);
            PI = PINO;
            IF STAB(K3,I) = 0 & ST2 = STATE(K3,I) THEN ORANGE: DO;
                Y:DO NN = 1 TO IK-1;
                    IF STATE(K3,INN) = SAVE(NN) THEN
                        GO TO END_GREEN; END;
                ST2 = STATE(K2,I);
            END;
                IF PINO>AX THEN GO TO ERR;
                GO TO END_YELLOW; END COLOR; ELSE DO;
            END;
                IF PINO>AX THEN GO TO ERR;
                GO TO END_YELLOW; END ORANGE; ELSE;
            END;
                IF PI = PINO THEN /* NO 4 */
                    IF NOUNSTAB = OB THEN GRAY: DO;
                        BOOLMX (PINO,K3) = '1'B;
                        MASK (PINO,J) = '0'B;
                        BOOLMX = '0'B; PINO = PINO + 1;
                    IF PINO>AX THEN GO TO ERR;
                    GO TO END_YELLOW; END COLOR; ELSE DO;
                END;
                    IF PINO>AX THEN GO TO ERR;
                    GO TO END_YELLOW; END ORANGE; ELSE;
                END;
                    END_YELLOW: END YELLOW;
            END;
                IF PINO>AX THEN GO TO ERR;
                GO TO END_GREEN; END;
        END_GREEN: END GREEN;
        END PINK;
        END_WHITE: END WHITE;
        IF NOUNSTAB = 1B THEN CALL PINK; END;
        END_ROW_LOOP: END ROW_LOOP;
OUT: ENn FOL_loop;
PUT EDIT ('KSETs LISTING FOLLOWS')(PAGE, A);
DO I = 1 TO COLMAX;
PUT EDIT('COLUMN NO', I)(SKIP(2), A, X(I), F(2));
DO J = 1 TO HBOUND(KSET, 10); IF KSET(I, J, 1) = '0'B THEN GO TO FIRE;
PUT EDIT(KSET(I, J, 1), KSET(I, J, 2))(SKIP, B, X(I), B);
END; FIRE: END; FREE SAVE, STATE, STA;
MAKESTR: PROCEDURE( MATRIX, STRING indicator, NN); /* RETURNS BIT STRINGS IN RSTRING, EACH ELEMENT * /
/* = ROW OF MATRIX IF STRING INDICATOR = 1B, OR EACH */
/* ELEMENT = COLUMN OF MATRIX IF STRING INDICATOR = 0B */
DECLARE MATRIX(*,*) BIT(1) PACKED,
STRING INDICATOR 8INARY FIXED(1);
IF STRING INDICATOR = 1B THEN DO I = 1 TO PINO;
BSTRING(NN, I) = '0'B;
DO J = 1 TO ROWMAX;
BSTRING(NN, I) = RSTRING(NN, I) (MATRIX(I, J));
END; END; ELSE DO I = 1 TO ROWMAX;
BSTRING(NN, I) = '0'B;
DO J = 1 TO PINO - 1;
BSTRING(NN, I) = RSTRING(NN, I) (MATRIX(J, I));
END; END; RETURN; END MAKESTR;
DECLARE (W, W2, F, G, H) BIT(PINO - 1) CONTROLLED;
ALLOCATE W, W2, F, G, H, RSTRING;
CALL MAKESTR( BOOLMX, OB, 1B);
CALL MAKESTR( MASK, OB, 10B);
CHECK: DO I = 1 TO ROWMAX;
W = BOOL ( BSTRING(1, I), BSTRING(2, I), '0110'8) ;
CLEAR: DO K = I+1 TO ROWMAX;
W2 = BOOL( BSTRING(1, K), BSTRING(2, K), '0110'8) ;
F = BOOL ( BSTRING(1, I), BSTRING(1, K), '0110'8) ;
G = BOOL( F, W, '0010'8 );
H = BOOL( G, W2, '0010'8 ) ;
IF H = '0'B THEN PARTITION: DO;
MASK( PINO, I) = '0'B;
BOOLMX( PINO, K) = '1'B;
PINO = PINO + 1 ;
IF PINO > AX THEN GO TO ERR;
END;
END CLEAR; END CHECK; PINO = PINO - 1;
DO I = PINO TO 1B BY -1B;
RE2: DO J = I -1B TO 1B BY -1B;
DO K = 1B TO ROWMAX;
IF ((BOOLMX( I, K ) => BOOLMX( J, K ))) (MASK(I, K) => MASK(J, K)) THEN GO TO END_RE2; END;
/* REACHED ONLY IF PART(I) = PART(J) */
DO K = 1B TO ROWMAX;
MASK(I, K) = MASK(PINO, K); END;
PINO = PINO - 1B; /* REDUNDANT PART(I) DELETED */
END_RE2: END RE2; END;
END;
IF SUBSTR(BITCODE, 1, 1) = '0'B THEN DO;
PUT EDIT('MATRIX TO BE PASSED TO REDUCTION ALGORITHM',
DO I=1 TO PINO;
PUT EDIT('BOOLMX(I,J),MASK(I,J) DO J=1 TO ROWMAX)(
X(2),B(1),B(1)) SKIP(2);  
END;
FREE STATE,SAVE,STAB,RSTRING,W,W2,F,G,H;
RETURN;  
END;
DECLARE (COUNT, MINCOUNT, MINCOL,SUM1,SUM2 )
DEC FIXED(2);
MINCOUNT = PINO;
DO I = 1 TO ROWMAX;
SUM1 = 0;  SUM2 = 0;
DO J = 1 TO PINO;
SUM2= SUM2+BIN(MASK(J,I));
SUM1= SUM1+BIN(BOOLMX(J,I));
END;  COUNT = SUM2 - SUM1;
IF COUNT< MINCOUNT THEN DO;
MINCOL =I;
MINCOUNT = COUNT ; END; END;
/* COLUMN MINCOL CONTAINS MINCOUNT DONT CARES */
SIZE1 = PINO + MINCOUNT;
PUT EDIT('NO OF PARTITIONS+ COMPLEMENTS GENERATED = ',
SIZE1)(SKIP(3),A,F(2));
ALLOCATE UNGER,ROW;
/* UNGER = 1 WHERE COMPL OF PARTN NEED BE CONSIDERED*/
PUT EDIT ('BITSTRINGS TO BE TRANSMITTED TO REDUCTION'
, 'ALGORITHM #1:') (PAGE,A,A);
UNGER = 'R';
CALL MAKESTR( BOOLMX, 1B, 1B);
CALL MAKESTR( MASK, 1B, 10B);
DO J = 1 TO PINO;
ROW(J)=(RSTRING(I,J))RSTRING(2,J));
IF BOOLMX(J,MINC0L)= MASK(J,MINC0L) THEN
UNGER = ((UNGER)|('1'B)); ELSE
UNGER = ((UNGER)|('0'B));
IF BOOLMX(J,MINC0L)= '1'B THEN ROW(J)=(~RSTRING(2,J))
(~RSTRING(1,J));
PUT EDIT( ROW(J))(SKIP(1),COLUMN(5),B);
END;
FREE BOOLMX,MASK,RSTRING,W,W2,F,G,H;
PUT EDIT ('VECTOR TO BE USED IN FOLLOWING PROCEDURE')
(SKIP(2),A);
PUT LIST(UNGER);  PUT DATA(PINO,ROWMAX);
GO TO END_RJSPRT;
ERR: PUT EDIT('NO PARTITIONS EXCEEDS AVAILABLE '
, 'STORAGE. INCREASE AX AND RERUN.') (PAGE,A,A);
OK = 'O'B;
END_RJSPRT: END RJSPRT;

(SUBRG):RJSMAX:PROC; /*HOPFULLY FASTER MAX INTERSECT*/
PUT EDIT('TIME ON ENTRY TO SUBPROGRAM RJSMAX = ',TIME
)(SKIP(3),A,A);
**************************************************************************************************
I*
GIVEN A SET OF PARTITIONS IN BOOLEAN MATRIX FORM, */
I* THIS PROGRAM RAPIDLY FINDS ALL MAXIMAL INTERSEC- */
I* TLES OF THE GIVEN MATRIX. THESE ARE STORED IN */
I* BITSTRING FORM IN ARRAY 'CLASS'. */
I*****************************************************
DECLARE NUMMAX BIN FIXED(15) EXTERNAL,
ROW(PINO) BIT(2*ROWMAX) CONTROLLED EXTERNAL
PACKED,
CLASS(*,*1) BIT(PINO) CONTROLLED EXTERNAL
PACKED,
UNGEX BIT(PINO) CONTROLLED EXTERNAL,
PINO DEC FIXED(3) EXTERNAL,
(ROWMAX, COLMAX) DEC FIXED(3) EXTERNAL,
SIZE1 BIN FIXED(15) EXTERNAL,
AV DEC FIXED(4,1) EXTERNAL;
I************************************************************
DECLARE DATA(ROWMAX) BIT(ROWMAX) CONTROLLED PACKED,
SIZE1 BIT(SIZE1) INITIAL(0),
(BOUM,DDUM) BIT(PINO) CONTROLLED,
DUM BIT(SIZE1) INITIAL(0),
TABLE(SIZE1-PINO) BIN FIXED(15) CONTROLLED,
(IN,INX,COUNT,I,J) BIN FIXED(15),
MAYBE(SIZE1*ROWMAX+.7) BIT(SIZE1) CONTROLLED
PACKED,
(DUMMY,POINT) BIT(SIZE1) CONTROLLED,
(FIRSTI,LASTI,FIRSTJ,LASTJ) BIT(ROWMAX)
CONTROLLED,
ALLOCATE DATA,BDUM,LASTJ,DDUM,CLASS(5*PINO+AV,2),
DUM,MAYBE,DUMMY,POINT,FIRSTI,LASTI,FIRSTJ,
TABLE;
I* ROW = INPUT PARTITIONS */
I* CLASS = OUTPUT MAXIMAL INTERSECTABLES */
I* NUMMAX = NUMBER OF M.I. IN CLASS */
I* UNGER = LIST OF PARTITION COMPLEMENTS */
I* ARRAY = LIST OF PAIRWISE INTERSECTABLES */
I* MAYBE = STEPWISE LIST OF M.I. CANDIDATES */
I* POINT = BACKWARD LIST OF PARTITIONS IN LAST MAYBE*/
PUT DATA(AY,ROWMAX,CMAX,PINO,SIZE1) SKIP(1);
PUT DATA(UNGEX) SKIP(1);
PUT EDIT('INPUT PARTITION LIST FOLLOWS:')(SKIP(4),
\(\text{A}I\)(ROW(I) DO I = 1B TO PINO)) (SKIP(1),B);
NUMMAX = 0B;
I* FORM PAIRWISE INTERSECTABLE LIST IN ARRAY */
DO I = 1B TO PINO;
FIRSTI = SUBSTR(ROW(I),1,ROWMAX);
LASTI = SUBSTR(ROW(I),ROWMAX+1);
DUM = '1'8;
DO J = I+1B TO PINO;
FIRSTJ = SUBSTR(ROW(J),1,ROWMAX);
LASTJ = SUBSTR(ROW(J),ROWMAX+1);
IF (BOOL(FIRSTI,LASTJ,'0010'B)|BOOL(FIRSTJ,LASTI,
 '0010'B)) = '0'B THEN CDUM = ((CDUM))('1'B);
ELSE CDUM = ((CDUM))('0'B);
END;
BOUM = UNGER;
DO WHILE (IN > OB);
FIRSTJ = (¬SUBSTR ROW(IN), ROWMAX+1));
LASTJ = (¬SUBSTR ROW(IN), 1, ROWMAX));
IF (BOOL (FIRSTJ, LASTJ, '0010'B) | BOOL (FIRSTJ, LASTJ, '0010'B)) = '0'B THEN CDUM = ((CDUM) || ('1'B));
ELSIF CDUM = ((CDUM) || ('0'B));
SUBSTR(BDUM, IN, 1) = '0'B; IN = INDEX(BDUM, '1'B);
END;
ARRAY(1) = '0'B; SUBSTR ARRAY(1), 1) = CDUM;
END;
BOUM = UNGER;
COUNT = OB;
IN = INDEX(BDUM, '1'B); /* LOOP THRU I'S IN UNGER */
DO WHILE (IN > OB);
CDUM = 'B'; COUNT = COUNT + 1B;
TABLE(COUNT) = IN;
FIRSTI = (¬SUBSTR ROW(IN), ROWMAX+1));
LASTI = (¬SUBSTR ROW(IN), 1, ROWMAX));
DDUM = UNGER;
INX = INDEX(DDUM, '1'B);
DO WHILE (INX > OB);
FIRSTJ = (¬SUBSTR ROW(INX), ROWMAX+1));
LASTJ = (¬SUBSTR ROW(INX), 1, ROWMAX));
IF (BOOL (FIRSTJ, LASTJ, '0010'B) | BOOL (FIRSTJ, LASTJ, '0010'B)) = '0'B THEN CDUM = ((CDUM) || ('1'B));
ELSIF CDUM = ((CDUM) || ('0'B));
SUBSTR(DOUM, INX, 1) = '0'B; INX = INDEX(DOUM, '1'B);
END;
ARRAY(COUNT+PINO) = '0'B;
SUBSTR ARRAY(COUNT+PINO), PINO+1) = CDUM;
SUBSTR(DOUM, IN, 1) = '0'B; IN = INDEX(DOUM, '1'B);
END;
/* ARRAY HAS BEEN LOADED WITH BITSTRINGS INDICATING */
/* PAIRWISE INTERSECTIONS */
PUT EDIT('ARRAY OF PAIRWISE INTERSECTABLES follows:')(PAGE, A)((ARRAY(I) DO I = 1B TO SIZE1)) (SKIP(1), B);
PUT EDIT('TIME AT START OF M.I. GENERATION = ', TIME)(SKIP(2), A, A);
PUT EDIT('MAXIMAL INTERSECTABLES GENERATED BY RJSMSAX', 'FOLLOW')(PAGE, A, A);
A: DO I = 1B TO SIZE1 -1B; /* LOOP PARTITIONS */
PUT EDIT('TIME IN ENTRY TO ', I, ' LOOP = ', TIME)(SKIP(3), A, F(3), A, A);
/* AT THIS POINT ALL M.I. CONTAINING PARTITIONS */
/* NUMBERED LESS THAN I HAVE BEEN GENERATED */
MAYBE(1B) = ARRAY(I);
J = 1B;
POINT = '0'B;
/* POINT = I FOR DECISION POINTS----BACKWARD */
IN = I;
IN = IN + INDEX(SUBSTR(MAYBE(J), IN+1B), '1'B);
B: DO WHILE (IN > OB);
/* LOOP THRU PARTITIONS INTERSECTABLE WITH MAYBE(J) */
SUBSTR(POINT, SIZE1-IN+1, 1) = '1'B;
DUMMY = ARRAY(IN); SUBSTR(DUMMY, 1, IN) = SUBSTR(MAYBE(
J, 1, IN);  
J = J + 1R;  
MAYBE(J) = BOOL(MAYBE(J-1R), DUMMY, '0001' B);  
IF IN = SIZE1 THEN GO TO MAXI; /* LAST PARTITION */  
INX = IN + INDEX(SUBSTR(MAYBE(J), IN+1R), '1'B);  
IF IN = INX THEN GO TO MAXI;  
/* NO PARTITIONS INTERSECTABLE WITH MAYBE(J) REMAIN */  
IN = INX;  
END B;  
MAXI: /* THIS POINT REACHED ONLY IF A M.I. IS FOUND */  
/* WHICH CONTAINS PARTITION(IN). THIS M.I. STORED IN */  
/* MAYBE(J). */  
/* NOW GENERATE SECOND HALF OF CLASS */  
DDUM = '0'B; CDUM = SUBSTR(MAYBE(J), PINO+1);  
IN = INDEX(CDUM, '1'B); /* IN = COMPLEMENT IN M.I. */  
IF IN = OR THEN GO TO RECORD;  
C : DO WHILE (IN > 0R);  
SUBSTR(CDUM, TABLE(IN), 1) = '1'B;  
SUBSTR(CDUM, IN), 1 = '0'B; IN = INDEX(CDUM, '1'B); END;  
RECORD:  
BDUM = SUBSTR(MAYBE(J), 1, PINO);  
IF SUBSTR(DDUM, SIZE1, 1) = '1'B THEN L = 1R;  
IF IN = SIZE1 THEN IF L = 0B THEN DU; BDUM, DDUM = '0'B;  
IF SIZE1 = PINO THEN SUBSTR(BDUM, PINO, 1) = '1'B;  
ELSE SUBSTR(DDUM, PINO, 1) = '1'B; END; ELSE ELSE;  
DO K = 1R TO NUMMAX;  
IF (((BOOL(CLASS(K, 1R), BDUM, '0100' B))) || (BOOL(CLASS(K, 1R), DDUM, '0100' B))) = '0'B  
(((BOOL(CLASS(K, 0B), BDUM, '0100' B))) || (BOOL(CLASS(K, 1B), DDUM, '0100' B))) = '0'B) THEN  
GO TO O;  
END;  
NUMMAX = NUMMAX + 1B;  
IF NUMMAX > HROUND(CLASS, 1) THEN DO;  
PUT EDIT('NUMBER OF MAX. INTERSECTABLES TOO GREAT',  
'FOR ALLOCATED STORAGE', ' AV = ', ' AV = ',  
' INCREASE AV AND RERUN.') (PAGE, A, A, SKIP(1),  
A, F(4, 1, 1));  
GO TO END_RJS_MAX;  
END;  
CLASS(NUMMAX, 1B) = BDUM;  
CLASS(NUMMAX, 0R) = DDUM;  
PUT EDIT(NUMMAX, BDUM, DDUM) (SKIP(1), F(3), COLUMN(10), B,  
SKIP(1), COLUMN(10), B);  
O: IN = SIZE1 - INDEX(POINT, '1'B) + 1R;  
/* PARTITION(IN) WAS THE LAST ONE ADDED TO MAYBE(J) */  
/* AND MAY CONFLICT WITH PARTITIONS LOWER ON THE */  
/* LIST, RESULTING IN THEIR DELETION FROM THE M.I. */  
/* JUST FOUND */  
IF J = 1R THEN GO TO END_A;  
IF (MAYBE(J) = MAYBE(J-1R)) THEN DO;  
J = J-1B; SUBSTR(POINT, SIZE1-IN+1B, 1) = '0'B;  
GO TO O; END;  
ELSE DO;  
/* THIS POINT REACHED ONLY IF LOWER PARTITION(INX) */
/* CONFLICTS WITH MAXIMAL INTERSECTABLE MAYBE(J) */
SUBSTR(MAYBE(J-1R),IN,1) = '0'8;
SUBSTR(POINT,SIZE1-IN+1R,1) = '0'8;
IN = IN + INDEX(SUBSTR(MAYBE(J-1),IN+1),'0'8);
J = J -1R;
GO TO B;
END:
END_A: END A:
FREE UNGER,ARRAY,BDUM,CDUM,MAYBE,DUMMY,POINT,FIRSTI,
LASTJ,FIRSTJ,LASTJ,DDUM,TABLE:
PUT EDIT('TIME ON EXIT FROM SUBPROGRAM RJSMAX = ',
TIME)(SKIP(4),A,A);
END_RJSMAX: END RJSMAX:

WLSCOV:PROC /* BRANCHING ROUTINE FINDS MIN CODES */
PUT EDIT('TIME ON ENTRY TO SUBPROGRAM WLSCOV = ',TIME)
(SKIP(4),A,A);
DECLARE /* DATA INPUT/OUTPUT THIS SUBPROGRAM */
NUMMAX BIN FIXED(15) EXTERNAL,
ROW(CINO) BIT(2*ROWMAX) CONTROLLED EXTERNAL
PACKED,
CLASS(*,*) BIT(CINO) CONTROLLED EXTERNAL
PACKED,
PINO DEC FIXED(3) EXTERNAL,
ROWMAX DEC FIXED(3) EXTERNAL,
CODE(*,*,*) BIT(ROWMAX) CONTROLLED EXTERNAL
PACKED,
NUMCOV BIN FIXED(7) EXTERNAL,
MINCOV(AP) BIN FIXED(7) CONTROLLED EXTERNAL,
OK BIT(1) EXTERNAL,
(AR,AT) DEC FIXED(4,1) EXTERNAL;
DECLARE (TESTR(NUMMAX),CODESTR,COVSTR(0:AT))
BIT(CINO) CONTROLLED,
BADCHECK(AR,NUMMAX) BIT(CINO) PACKED,
BADFLAG(AR,NUMMAX) BIT(1) PACKED,
MAKFC00F(2) LABEL,
(INDXX(CINO),PATH(AR)) BIT(NUMMAX)
CONTROLLED,
PATH# BINARY FIXED INITIAL(18),
COVER(AR,AT) BINARY FIXED CONTROLLED,
(CHEK(AR),DUMMY,ALFA(CINO)) BIT(NUMMAX)
PACKED CONTROLLED;
/*****************************/
/* TESTR(I) INDICATES THE COLUMNS COVERED BY INT. I */
/* ALFA(J) IS '1'8 WHERE MAX INT COVERS PARTITION J */
/* INDXX(J) IS CURRENT INT COVERING COLUMN J */
/* COVER(I,*) INDICATES THE MAXIMAL INTERSECTABLES */
/* IN THE ITH COVERING SET */
/* PATH(K) IS KTH INTERSECTABLE IN CURRENT PATH */
/* NUMCOV IS # OF COVERING SFTS OF INTERSECTABLES */
/* MINCOV IS MINIMUM # OF INTERSECTABLES READ */
/* CHEK(K) SHOWS INTERSECTABLES IN KTH COVER */
/* COVSTR(I) IS MASK OF COVER OF FIRST I INT. OF PATH*/
**********************************************************************
ALLOCATE ALFA, INDEX, PATH, TESTR, CODESTR, COVSTR;
ALLOCATE MINCOV(AR), CHECK, DUMMY, COVER;
ALFA = '0' B; DUMMY = '0'B;
PUT EDIT (' THE INTERSECTABLE COVERING PROBLEM')
(PAGE, A);
DO II = 1 TO NUMMAX;
TESTR(II) = CLASS(II, 1) | CLASS(II, 2);
CODESTR = TESTR(II);
IN = INDEX(CODESTR, '1'B);
DO WHILE (IN > 0);
SUBSTR(ALFA(IN), II, 1) = '1'B;
SUBSTR(CODESTR, IN, 1) = 'O'B;
IN = INDX(CODESTR, '1'B);
END;
PUT EDIT (TESTR(II)) (SKIP, XI41, 81);
END;
NUMCOV = 0B;
MINCOV = AT;
COVSTR(0) = 'O'B;
CODESTR = COVSTR(0); /* TEST STRING OF ALL 1'S */
RADFLAG = 'O'B; BADCHECK = CODESTR;
I = OB;
FORWARD: I = I + 1B;
IF I > MINCOV(I) THEN GO TO BACKUP;
J = INDX(COVSTR(I-1B), 'O'B);
IF J = OB THEN GO TO MUST_COVER;
IF RADFLAG(I, J) = '1'B THEN
IF BOOL(RADCHECK(I, J), COVSTR(I-1B), '0010'B)
/* RADCHECK(*,*) K-POSITION =1 IF NO COVER FOR KTH */
/* PARTITION EXISTS AT THIS LEVEL; =0 IF COVER */
/* DOES NOT EXIST. */
/* INITIALIZE BADCHECK TO '1'B */
= '0'B THEN
THEN /* NO COVER CAN RESULT FROM BRANCHING PAST */
GO TO BACKUP; /* THIS POINT */
ELSE ELSE;
INDXX(J) = INDX(ALFA(J), '1'B);
END;
ELSE;
BRANCH: PATH(I) = INDXX(J);
IF (BADFLAG(I, J) = 'O'B) & (I = MINCOV(I)) THEN
RADCHECK(I, J) = BOOL(RADCHECK(I, J), TESTR(PATH(I)),
'0010'B);
SUBSTR(DUMMY, PATH(I), 1) = '1'B;
COVSTR(I) = COVSTR(I-1B) | TESTR(PATH(I));
MUST_COVER: IF COVSTR(I) = CODESTR
THEN DO;
PUT EDIT (' COVER') (SKIP, XI10, A);
IF (I < MINCOV(I1)) ((NUMCOV = AR) & (I = MINCOV(I)))
THEN DO: NUMCOV = LB;
PUT EDIT (I, ' VARIABLE COVERS: ') (SKIP(2), X(5), F(2), A);
END;
ELSE DO;
DO K = 1 TO NUMCOV;
IF DUMMY = CHECK(K) THEN GO TO DUP;
END;
NUMCOV = NUMCOV + 18;
END;
CHECK(NUMCOV) = DUMMY;
DO II = 1 TO I;
COVER(NUMCOV, II) = PATH(II);
END;
DUP: MINCOV = 1;
PUT EDIT (NUMCOV, 'AFTER ', 'PATH', 'PATHS')
(F(3), A, F(6), A);
I = I + 18;
IF NUMCOV = AR THEN DO;
MINCOV = MINCOV - 18;
I = I - 18;
SUBSTR(DUMMY,PATH(I),1) = '0'B;
END;
END;
ELSE GO TO FORWARD;
BACKUP: I = I - 18;
IF I = OR THEN GO TO OUT;
SUBSTR(DUMMY,PATH(I),1) = '0'B;
J = INDEX(COVSTR(I-18),'0'B);
JNDXX = INDEX(SUBSTR(ALFA(J),INDXX(J)+18),'1'B);
IF JNDXX = OB THEN DO;
IF I = MINCOV(1)
THEN BADFLAG(I, J) = '1'B;
GO TO BACKUP;
ELSE
END;
PATH# = PATH# + 18;
GO TO BRANCH;
OUT: IF NUMCOV = AR THEN MINCOV = MINCOV + 18;
PUT EDIT ('MINIMUM COVER IS ', MINCOV(1),
' INTERSECTABLES')(SKIP(2),A,F(7),A);
FINI: IF NUMCOV = AR
THEN PUT EDIT ('ONLY', AR, ' UNIQUE ASSIGNMENTS ',
'PRESERVED.') (SKIP(4), A, F(2), A, A);
ELSE PUT EDIT (NUMCOV, ' UNIQUE ASSIGNMENT(S)'
(SKIP(4), X(2), F(2), A);
AR = MIN(NUMCOV, AR);
ALLOCATE CODE(AR, MINCOV(1), 2);
M = ROWMAX;
DO II = 1 TO AR;
PUT EDIT ('MINIMAL COVER ', II) (SKIP(2), X(6), A, F(2));
DO JJ = 1 TO MINCOV(1);
CODE(II, JJ, 1) = '0'B; CODE(II, JJ, 2) = -CODE(II, JJ, 1);
PUT EDIT (COVER(II, JJ)) (SKIP, X(5), F(5));
DO K3 = 1B, 10B;
CODESTR = CLASS(COVER(II, JJ), K3);
IN = INDEX(CODESTR, '1'B);
DO WHILE (IN > 0);
GO TO MAKECODE(K3);
MAKECODE(1): CODE(II, JJ, 1) =
CODE(IJ, JJ, 1) = SUBSTR(ROW(IN), 1, M);
CODE(IJ, JJ, 2) =
CODE(IJ, JJ, 2) = SUBSTR(ROW(IN), M+1, M);
GO TO MORE;
MAKECODE(1): CODE(IJ, JJ, 1) =
BOOL(CODE(IJ, JJ, 1), SUBSTR(ROW(IN), M+1, M), '1011' B);
CODE(IJ, JJ, 2) =
BOOL(CODE(IJ, JJ, 2), SUBSTR(ROW(IN), 1, M), '0010' B);
MORE: SUBSTR(CODESTR, IN, 1) = 'O'B;
IN = INDEX(CODESTR, '1'B);  
END;
END;
PUT EDIT (CODE(IJ, JJ, 1), CODE(IJ, JJ, 2))
(X(5), B, SKIP(0), X(15), B);
END END
ENDB\:
FREE ALFA, CHK, DUMMY, CODESTR, COVSTR, COVER,
INDEX, PATH, TESTR, ROW, CLASS;
IF NUMCOV = 08 THEN DO;
PUT EDIT('TABLE CANNOT BE SUCCESSFULLY CODED WITH ','
AT,' VARIABLES. INCREASE AT AND RERUN.') 
PAGE, A, F(2), A);
OK = 'O'B; END;
END WLSCOV;

WLSCG2: PROC;
PUT EDIT('TIME ON ENTRY TO SUBPROGRAM WLSCG2 = ', TIME)(
SKIP(4), A, A);
DECLARE /* DATA INPUT AND OUTPUT THIS ROUTINE */
(BOOLMX MAX, ROWMAX), MASK(A, ROWMAX)) BIT(1) 
CONTROLLED PACKED EXTERNAL,
CODE(*, *, *) BIT(ROWMAX) CONTROLLED EXTERNAL
PACKED,
NUMCOV BIN FIXED(7) EXTERNAL,
MINCOV(AR) BIN FIXED(7) CONTROLLED EXTERNAL,
ROWMAX DEC FIXED(3) EXTERNAL,
PINO DEC FIXED(3) EXTERNAL,
OK BIT(1) EXTERNAL,
(AR, AT) DEC FIXED(4, 1) EXTERNAL;
DECLARE (TEST(4), RULE(4)) LABEL,
A(2*ROWMAX) BINARY FIXED (1) CONTROLLED,
(B, C) (PINO) BIT (1) CONTROLLED,
D (PINO) BINARY FIXED (7) CONTROLLED,
BRANCH(0B:10000B) BIT(AT) PACKED CONTROLLED,
RCOUNT(10000B) BIN FIXED CONTROLLED,
ASET(ROWMAX) BIT (1) CONTROLLED,
HOLD(ROWMAX) BIT(1) CONTROLLED,
(VARI, VARI2, VARJ1, VARJ2) BIT(ROWMAX) 
CONTROLLED,
(ISAVE(5), ITSAVE(5)) BINARY FIXED,
(PASS#, #ONES, #ZEROS, COLTEST)
**Binary Fixed,**

ITERATION, PICOV, #CHOICES)

**Binary Fixed Initial (0B);**

/*****************************/
/\ AL(*) IS THE STRING BEING GENERATED */
/\ B(1) IS '1'B IF ROW 1 IS STILL DON'T CARE */
/\ C(1) IS '1'B IF ROW 1 DOESN'T FIT IN */
/\ D(1) CONTAINS TABLE WHERE ROW 1 IS COVERED */
/\ BRANCH(*) INDICATES READ CHOICES FOR THIS PATH */
/\ COUNT(*) INDICATES # OF CHOICES IN CURRENT PATH */
/\ COUNT(#) IS # OF CURRENT PASS (BRANCH EXPLORED) */
/*****************************/

AT = MAX(AT, PIMJ/1B);
ALLOCATE A, B, C, D, ASET, HELD, MINCOV(A1R),
CODE, A1R, AT, 10B), BRANCH, BCOUNT;
K = 1B; L = 1B;*** TEMPORARY ****/
NUMCOV = 1B;
NDUP = 0B; K16 = 0B;
PASS# = 0B; BRANCH(0B) = '0'B;
RESTART: /* GENERATES A SET OF COVER PARTITIONS */
ITERATION = 1B; KOUNT = 1B;
B = '0'B; C = '0'B; D = 1111B; II = 1B;
NEW_TABLE: /* GENERATES A COVER PARTITION */
ITERATION = ITERATION + 1B;
IF ITERATION > AT THEN DO;
OK = '0'B;
PUT EDIT('UNABLE TO COVER MATRIX WITH AT VARIABLES. '
'OPTIONALLY INCREASE AT AND RUN.') (PAGE, A);
PUT DATA(AT); GO TO END_2; END;
COLTEST = 0B;*** TEMPORARY ****/
PICOV = 0B;
DO J = 1 TO POWMAX; *** FIND 1ST COL TO SET ***
CALL COUNT01;
GO TO TEST(K);
TEST(1): IF (#ONES + #ZEROS) > COLTEST
THEN DO; COLTEST = #ONES + #ZEROS; IND = J;
IF #ONES > #ZEROS
THEN CALL SET(1B); ELSE CALL SET(0B);
END;
GO TO END_LOOK;
TEST(2): GO TO END_LOOK;
TEST(3): GO TO END_LOOK;
TEST(4): GO TO END_LOOK;
END_LOOK: END;
COUNT01: PROC; /* COUNTS 0'S, 1'S IN COL J */
#ONES = 0B; #ZEROS = 0B;
DO I = 1 TO PIMJ;
IF ((B(1) = '1'B) & (C(1) = '1'B) & (D(1) < ITERATION))
THEN GO TO END_COL;
IF PIMJX(I, J) = '1'B THEN #ONES = #ONES + 1B;
ELSE IF MASK(I, J) = '0'B THEN #ZEROS = #ZEROS + 1B;
END_COL: END;
NUMDIFF = #ONES - #ZEROS;
END COUNTO;

SET: PROCEDURE(X); DECLARE X BINARY FIXED (1);
A(IND)=X; A(IND+ROWMAX)=X; END SET;
DO I=1 TO PINO; /* SET 1ST COL AND !(DON'T CARES)*/
IF D(I)<ITERATION THEN GO TO END_SET;
IF BOOLMX(I,IND)=MASK(I,IND)
THEN IF A(IND)=-BOOLMX(I,IND)
THEN CALL COMPROW; ELSE;
ELSE R(I)='1'B;
END_SET: END;
COMPROW: PPDC; /* COMPLEMENTS ROW I */
DO M=1 TO ROWMAX;
IF BOOLMX(I,M)='0'B THEN HOLD(M)='1'B;
ELSE HOLD(M)='0'B;
IF MASK(I,M)='0'B THEN BOOLMX(I,M)='1'B;
ELSE BOOLMX(I,M)='0'B;
MASK(I,M)=HOLD(M); END: END COMPROW;
ASET='0'B; ASET(IND)='1'B;
DO N=2 TO ROWMAX; /* SET REMAINING COLUMNS */
COLTEST=OR; /* TEMPORARY*/
MINOUT=PINO;
DO J=1 TO ROWMAX; /* EXAMINE REMAINING COLS */
IF ASET(J)='1'B THEN GO TO END_SCAN;
CALL COUNTOL;
GO TO RULE(L);
RULE(1): IF ABS(NUMDIFF) > COLTEST
THEN DO: COLTEST = ABS(NUMDIFF);
IND = J; IF NUMDIFF > OR
THEN CALL SET(1B);
ELSE CALL SET(0B);
END;
ELSE IF COLTEST = OR
THEN IF #ONES < MINOUT
THEN DO; MINOUT = #ONES;
IND = J;
IF #ONES = OR
THEN DO:
A(IND)='0'B;
A(IND+ROWMAX)='1'B;
END;
GO TO END_SCAN;
RULE(2): GO TO END_SCAN;
RULE(3): GO TO END_SCAN;
RULE(4): GO TO END_SCAN;
END_SCAN: END;
IF (COLTEST=OR) & (MINOUT>08)
THEN DO; KOUNT = KOUNT + 1R;
IF PASS# = OR THEN GO TO ARBDEC;
IF KOUNT < BCOUNT(PASS#) THEN DO;
CALL SET(BIN(SUBSTR(BRANCH(PASS#),
KOUNT,1R)));
GO TO SET_IND; END;
IF KOUNT = BCOUNT(PASS#) THEN DO;
CALL SET(1B);
SUBSTR(BRANCH(PASS#),KOUNT,1B) = '1'B;
GO TO SET_IND;
END;

ARBDEC: DO; /* REGISTER ARBITRARY CHOICE */
#CHOICES = #CHOICES +1B;
IF #CHOICES > 10000B
THEN IF K16 = 18
THEN GO TO NO_MORE;
ELSE DO; K16 = 18;
PUT EDIT (' 16 CHOICES ENCOUNTERED'
(SKIP(4),A);
GO TO NO_MORE; END;
BRANCH(#CHOICES) = BRANCH(PASS#);
SUBSTR(BRANCH(#CHOICES),KOUNT,1B) = '0'B;
RCOUNT(#CHOICES) = KOUNT;
NO_MORE: CALL SET(0B);
END /* ARBDEC*/;

SET_IND: ASEITIND)='1'B;
DO I=1 TO PIIO; /* SET COL AND B(DOUNT CARES) */
IF (D(I)<ITERATION)|C(I)='1'B
THEN GO TO END_LOOP;
IF BOOLMX(I,IND) = MASK(I,IND) THEN GO TO END_LOOP;
ELSE DO; IF A(IND) = BOOLMX(I,IND) THEN B(I)='0'B;
ELSE IF B(I)='0'B THEN C(I)='1'B;
ELSE DO; B(I)='0'B;
CALL COPYROW; END;
END;
END_LOOP: END; END; /* COMPLETES ALL COLS */
PUT EDIT (' CODE ',NUMCOV,' VARIABLE ',ITERTIATION,
' COVERS PARTITIONS : ') (SKIP(2),A,F(2),A,F(2),A);
NROWS = 0B;
DO I=1 TO PIIO; /* NOTE ROWS COVERED, REMAINING */
IF D(I)<ITERATION
THEN DO; PICOV=PICOV+1B; GO TO GO_ON; END;
D(I) = 11111B;
IF C(I)='1'B THEN DO; B(I)='0'B; C(I)='0'B; END;
ELSE DO; D(I) = ITERATION;
PUT EDIT (I,' ',(F(3),A);
PICOV = PICOV+1B;
NROWS=NROWS+1B; [ISAVE(I)]=I;
END;
GO_ON: END;
CODE(NUMCOV,ITERATION,1) = '0'B;
CODE(NUMCOV,ITERATION,2) = CODE(NUMCOV,ITERATION,1);
IF NROWS = 1B THEN DO;
I = ISAVE(I); ITSAVE(I) = ITERATION;
DO J =1 TO ROWMAX;
IF BOOLMX(I,J) = '1'B
THEN SUBSTR(CODE(NUMCOV,ITERATION,1),J,1) = '1'B;
ELSE IF MASK(I,J) = '0'B
THEN SUBSTR(CODE(NUMCOV,ITERATION,2),J,1) = '0'B;
END;
II = II +1B; GO TO NEXT; END;
DO J=1 TO ROWMAX; /* FILL OUTPUT ARRAY */
IF A(J) =1B
THEN SUBSTR(CODE(NUMCOV, ITERATION, I), J, 1) = '1'B;
ELSE IF A(J+ROWMAX) =OB
THEN SUBSTR(CODE(NUMCOV, ITERATION, I), J, 1) = '0'B;
END;
NEXT: IF PICOV<PINO THEN GO TO NEW_TABLE;
IF II > 10B THEN DO;
ALLOCATE VARI1, VARI2, VARJ1, VARJ2;
DO II = 1B TO II-10B;
IF ISAVE(II) = OR THEN GO TO NEW_1;
I = ITSSAVE(II);
VARI1 = CODE(NUMCOV, I, 1);
VARI2 = CODE(NUMCOV, I, 2);
DO I2 = II+1B TO II-1B;
IF ISAVE(I2) = OB THEN GO TO NEW_2;
J = ITSSAVE(I2);
VARJ1 = CODE(NUMCOV, J, 1);
VARJ2 = CODE(NUMCOV, J, 2);
IF (BOOL(VARI2, VARJ2, '0010'B)
    BOOL(VARI2, VARJ2, '0100'B)) = '0'B THEN DO;
    CODE(NUMCOV, I, I) = VARI1|VARJ1;
    CODE(NUMCOV, I, 2) = VARI2&VARJ2;
    PUT EDIT (" PARTITIONS ",ISAVE(II), ' , ', ISAVE(I2),
    ' COMBINED TO YIELD VARIABLE ', ITSAVE(II))
    (SKIP(2), X(5), A,F(3), A,F(3), A,F(2));
    ISAVE(I2) = OB;
    GO TO NEW_1; END;
IF (BOOL(VARII, VARJ1, '0001'B))
    BOOL(VARI2, VARJ2, '1000'B)) = '0'B THEN DO;
    CODE(NUMCOV, I, I) = BOOL(VARI1, VARJ2, '1011'B);
    CODE(NUMCOV, I, 2) = BOOL(VARI2, VARJ1, '0010'B);
    PUT EDIT (" PARTITIONS ",ISAVE(II), ' , ', ISAVE(I2),
    ' COMBINED TO YIELD VARIABLE ', ITSAVE(II))
    (SKIP(2), X(5), A,F(3), A,F(3), A,F(2));
    ISAVE(I2) = OB;
    GO TO NEW_1; END;
NEW_2: END;
NEW_1: END;
FREE VARI1, VARI2, VARJ1, VARJ2;
J = OB;
DO I = 10B TO II-10B;
IF ISAVE(I) = OB THEN DO;
    ITERATION = ITERATION -1B;
    DO IT = ITSSAVE(I)-J TO ITERATION;
    CODE(NUMCOV, IT, *) = CODE(NUMCOV, IT+1B, *);
    J = J +1B;
END;
END;
END;
IF ISAVE(II-1B) = OB THEN
    ITERATION = ITERATION -1B;
END; /* LOOP IF II > 10B */
MINCOV(NUMCOV) = ITERATION;

ONE:  DO KONE = 1 TO NUMCOV-1B; /* COMPARE ALL PREVIOUS */
    IF MINCOV(NUMCOV) = MINCOV(KONE) THEN GO TO END_ONE;

TWO:  DO KTHWO = 1 TO MINCOV(NUMCOV); /* ALL NEW PART */
    DO KTHREE = 1 TO MINCOV(KONE); /* AGAINST COVER K1*/
    IF (CODE(NUMCOV, KTHWO, 1) = CODE(KONE, KTHREE, 1))
        (CODE(NUMCOV, KTHWO, 2) = CODE(KONE, KTHREE, 2))
        THEN GO TO END_TWO;
    END;
    GO TO END_ONE;

END_TWO: END TWO;
    GO TO DUP;

END_ONE: END ONE;
    GO TO NONRED;

DUP:  PUT SKIP(2);
    PUT EDIT (' A DUPLICATE ASSIGNMENT GENERATED,!
        ' DROPPED ')(A, A);
    NUMCOV = NUMCOV - 1B;
    NDUP = NDUP + 1B;  IF NDUP > 100B THEN GO TO END_2;
    GO TO MORE;

NONRED:  PUT EDIT (' ASSIGNMENT ' NUMCOV ' HAS 
        ITERATION,' VARIABLES : ')(SKIP(3), A, F(2), A, F(2), A);
    NDUP = OR;
    DO N = 1 TO ITERATION;
        PUT EDIT (CODE(NUMCOV, N, 1), CODE(NUMCOV, N, 2))
        (SKIP(2), COL(15), B, SKIP(0), COL(15), B);
    END;

MORE:  IF PASS# < #CHOICES THEN DO;
        PASS# = PASS# + 1B;
        NUMCOV = NUMCOV + 1B;
    IF NUMCOV > AR THEN DO;
        PUT EDIT ('MORE THAN AR UNIQUE ASSIGNMENTS',
            ' GENERATED. ONLY AR ARE PRESERVED.') (SKIP(4),
            A, A);
        PUT DATA(AR);
        NUMCOV = NUMCOV - 1B;
        GO TO END_2;  END;
    GO TO RESTART;
    END;

END_2: FREE A, B, C, D, ASET, HOLD, BOOLMX, MASK, BRANCH, BCOUNT;
END WLSCG2;

RJSMKI:PROC; /* GENERATES NEXT-STATE EXPRESSIONS */
    /* GIVEN KSET(1:K,*,2) FROM JTH COLUMN OF A K COLUMN */
    /* FLOATABLE AND CODE(1,JVAR,2), A STATE ASSIGNMENT */
    /* I HAVING JVAR VARIABLES. EACH ELEMENT OF CODE */
    /* IS A BITSTRING OF LENGTH ROWMAX; MASK CODINGUSED*/
    PUT EDIT ('TIME ON ENTRY TO SUBPROGRAM RJSMKI V.6 = ',
            TIME)(SKIP(1), A, A);
DECLARE /* DATA INPUT AND OUTPUT THIS ROUTINE */
    (SIZE1, MAXNOK, IJKL, J, I) BIN FIXED(15)
    EXTERNAL,
    (JVAR, NOL) BIN FIXED(15) EXTERNAL,
    ONESUB(*, *, *) BIT(NOL) CONTROLLED EXTERNAL
    PACKED,
ONESCNT(*) BIN FIXED(7) CONTROLLED EXTERNAL,
DONTCAR(*,*) BIT(NOL) CONTROLLED EXTERNAL
PACKED,
ONQ BIN FIXED(15) EXTERNAL,
INCOD(*,*) BIT(INDIG) CONTROLLED EXTERNAL,
CODE(*,*,*) BIT(ROWMAX) CONTROLLED EXTERNAL
PACKED,
MINCOVAR) BIN FIXED(7) CONTROLLED EXTERNAL,
KSET(*,*,*) BIT(ROWMAX) CONTROLLED EXTERNAL
PACKED,
(ROWMAX, COLMAX) DEC FIXED(3) EXTERNAL,
OK BIT(1) EXTERNAL,
(AY, AW) DEC FIXED(4, 1) EXTERNAL;
DECLARE DCARE(*,*) BIT(JVAR) CONTROLLED PACKED,
PSUB(MAXNOK*(ROWMAX+1-MAXNOK),2) BIT(JVAR)
CONTROLLED PACKED,
ONES(JVAR, MAXNOK*(ROWMAX+1-MAXNOK),2) BIT(JVAR)
CONTROLLED PACKED,
S6R(*,*) BIT(JVAR) PACKED CONTROLLED,
(DTI, DT2, DT3, DT4) BIT(JVAR) CONTROLLED,
(BITS, DITS, F1TS, G1TS) BIT(32) VARYING,
(IN, INX, KSLGH) BIN FIXED(7),
DCNT BIN FIXED(15) INITIAL(0);
PUT EDIT('FLOW TABLE COLUMN', J)(SKIP(2), A, X(2), F(2));
PUT EDIT('INPUT CODE cOllOWS') (SKIP(2), A)((( CODE(I,M ,N) DO N=18 TO 108) DO M=18 TO JVAR))(SKIP(1),
R, SKIP(0), R);
PUT EDIT('COLUMN', J, 'KSETS FOLLOW') (SKIP(2), A, X(1),
F(7), X(1), A)((KSET(J, M, N) DO N=18 TO 108) DO M=18 TO MAXNOK))(SKIP(1),
R, SKIP(0), R);
ALLOCATE ONES, PSUP, DT1, DT2, DT3, DT4;
/* P-- AND 1-- SURCUB GENERATION FOR ROW J */
PUT EDIT('SUBCUBES GENERATED FOR COLUMN', J, ' FOLLOW'
)(PAGE, A, X(1), F(?), A);
PSLGH = OB; KSLGH = OB;
PS: DO K = 18 TO MAXNOK; /* LOOP THRU KSETS */
IF KSET(J, K, 1B) = 'O'B THEN GO TO B;
KSLGH = KSLGH + 1B;
BITS = KSET(J, K, 1B);
INX = INDEX(KSET(J, K, 1B), '1'B);
DO L = 18 TO JVAR; /* LOOP ISY'S FORM STAB ROWCODE */
SUBSTR(DT2, L, 1) = SUBSTR(CODE(I, L, 1), INX, 1);
SUBSTR(DT4, L, 1) = SUBSTR(CODE(I, L, 2), INX, 1); END;
IF KSET(J, K, 1B) = KSFT(J, K, 108) THEN DO;
PSLGH = PSLGH + 1B;
PSUP(PSLGH, 1A) = DT2; PSUP(PSLGH, 10B) = DT4;
DT3 = DT2; IN = INDEX(DT3, '1'B);
DO WHILE (IN<0B);
ONESCNT(IN) = ONESCNT(IN) + 1B;
ONESUB(IN, ONESCNT(IN), 1B) = DT211 INCOD(J, 1B);
ONESUB(IN, ONESCNT(IN), 10B) = DT411 INCOD(J, 10B);
SUBSTR(DT3, IN, 1) = 'O'B; IN = INDEX(DT3, '1'B);
END; GO TO END_PS;
END;
IN = INDEX(BITS, '1'B);
DO WHILE (TN>OB); /* LOOP SIGN ROWS OF KSET */
IF INX = INX THEN GO TO S;
PSLGH = PSLGH +1B;
DO L = 1B TO JVAR; /* LOOP ISV'S FORM UMSTAB ROWCODE */
SUBSTR(DT1,L,1) = SUBSTR(CODE(I,L,1),I',1); END;
SUBSTR(DT3,L,1) = SUBSTR(CODE(I,L,2),IN,1);
DTI = BOOL(DT1,DT2,'0001'B);
DT3 = BOOL(DT3,DT4,'0111'B);
DO L = 1B TO JVAR; /* LOOP ISV'S FORM L-SUBCUBES */
IF SUBSTR(DT2,L,1) = '1'B THEN DO;
ONESCNT(L) = ONESCNT(L) +1B;
ONESUBL,ONESCNT(L),1B) = (((DT1)||(INCOND(I,1R)));
ONESUBL,ONESCNT(L),10B) = (((DT3)||(INCOND(I,10B)));
END; END;
PSUBL(PSLGH,1B) = DTI;
PSUBL(PSLGH,10B) = DT3;
PUT EDIT(DT1,DT3)(SKIP(1),A,SKIP(0),B);
S = SUBSTR(BITS,IN,1) = '0'B;
IN = INDEX(BITS,'1'B);
END;
END_PSN: END PS;
/* PSUB CONTAINS PSLGH PSUBCUBES DUE TO CODE I */
/* BEING APPLIED TO JTH FLOW TABLE COLUMN */
/* NOW COMPL PSUB TO OBTAIN DONTCARE TERMS */
/* IF PSUB = P1 + P2 + P3 + ... PN THEN LET */
/* F = (PSUB) WHERE F = (P1')(P2')(P3') ...(PN') */
/* NOW IF PN' = Y1 + Y2 + ... +YJ THEN */
/* IF(Y1(P1')(P2')...Y2(P1')(P2')...+YJ(P1')(P2')... */
/* SOME TERMS OR FACTORS OF WHICH MAY BE ELIMINATED */
/* BY USING THE BOOLEAN ALGEBRA THEOREMS: X(X + Y) = X */
/* AND XY(X1 + W + Z) = XY(W + Z) */
R: PUT DATA(J,PSLGH,KSLGH,JVAR);
BITS = BOOL(PSUBL(PSLGH,1B),PSUBL(PSLGH,10B),'0110'B);
IN = INDEX(BITS,'0'B); M = OB;
AA = AW * PSLGH * JVAR;
ALLOCATE SCR(AA,10R), NCARE(AA,10R);
PUT EDIT('SCRATCH SIZE = ',AA)(SKIP(2),A,F(3));
G: DO WHILE (IN>OB);
M = M +1B; ONFS(M,1B,1R) = '0'B;
ONES(M,1B,10R) = (~ONES(M,1B,1B));
DITS, SUBSTR(ONES(M,1,1),IN,1) = (
  ~SUBSTR(PSUBL(PSLGH,1),IN,1));
SUBSTR(ONES(M,1,2),IN,1) = DITS;
SUBSTR( BITS, IN,1) = '1'B;
INX = 1B;
C: DO I = 1 TO PSLGH -1;
/* IF PSUB(*) is EXCLUDED FROM THIS COLUMN BY A(A+B) = A */
/* FITS = SUBSTR (PSUBL, L,1), IN,1); */
GITS = ~ SUBSTR(PSUBL(L,10R),IN,1);
IF((FITS = GITS) & (FITS = DITS)) THEN GO TO END_C;
INX = INX +1B; ONFS(M,INX,1B) = (~PSUBL(L,10B));
ONES(M,INX,10B) = (~PSUBL(L,1B));
/* IF PSUB(*) is MODIFIED FOR THIS COLUMN BY A(A') = AB */
/* IF (FITS = GITS) THEN IF (FITS = DITS) THEN */
IF (FITS='0'OR) THEN
SUBSTR(ONES(M, INX, 1B), IN, 1) = '1'OR; ELSE
SUBSTR(ONES(M, INX, 1B), IN, 1) = '0'OR; ELSE;
IF ONES(M, INX, 1B) = (ONES(M, INX, 1B)) THEN DO;
  IN = INDEX(BITS, '0'OR); GO TO END_C;
END_C: IF INX > 18 THEN DO;
SCR(1B, 1B) = ONES(M, 1B, 1B);
SCR(1B, 10B) = ONES(M, 1B, 10B);
L=1B; ENDO; LNEW = 1B;
/* SCR CONTAINS LITERAL PRODUCT TERMS OF THE FORM */
/* (AB) WHERE A = SOME LITERAL PRODUCT FROM LAST */
/* ITERATION AND B = A LITERAL FROM TOPMOST */
/* UNMULTIPLIED SUM TERM IN COLUMN M: ITERATE TO */
/* MULTIPLY SCR BY REMAINING SUM TERMS IN COLUMN */
F: DO INX=10B TO INX; /* LOOP THRU REMAINING SUMS */
  DT3 = BOOL(ONES(M, IN, 1B), ONES(M, IN, 10B), '1001'B);
  I2 = INDEX(DT3, '1'B);
H: DO WHILE (I2>0B); /* LOOP THRU SIG DIGIT/SUM TERM */
  DITS = SUBSTR(ONES(M, IN, 1B), I2, 1B);
  DT1 = SCR(INX, 1B); /* LOOP THRU OLD SCR */
  DT2 = SCR(INX, 10B);
  FITS = SUBSTR(DT1, I2, 1B);
  GITS = SUBSTR(DT2, I2, 1B);
  IF (GITS = FITS) THEN IF (GITS = DITS) THEN
    /* AC(A+8)=AC */ GO TO CHECK;
  ELSE GO TO END_D; /* ABC(C+D+E) = ABC(D+E) */
  SUBSTR(DT1, I2, 1B) = DITS;
  SUBSTR(DT2, I2, 1B) = DITS;
/* INCLUSION AND DUPLICATION CHECK: NUTERM */
  CHECK: DO LA = L+1B BY 1B WHILE (LA<=LNEW);
  DT4 = (BOOL(DT1, SCR(LA, 1B), '0110'B)) (BOOL(DT2, SCR(LA, 10B), '0110'B));
  IF DT4 = '0'B THEN GO TO END_D;
  IF (BOOL(DT4, BOOL(SR(LA, 1B), SCR(LA, 10B), '0110'B),
    '0010'B)='0'B) THEN GO TO END_D;
/* NUTERM PRODUCT INCLUDED IN SOME OLDER PRODUCT */
END_D: LNEW = LNEW + 1B; IF LNEW > HBOUND(SCR, 1B) THEN GO TO ERR;
SCR(LNEW, 1B) = DT1; SCR(LNEW, 10B) = DT2;
END_D: END D;
SUBSTR(DT3, I2, 1B) = '0'B; I2 = INDEX(DT3, '1'B);
END_H: END H;
/* PARTIAL PRODUCT SCR */ SOME SUM FROM ONES(M, 1B, 1B) */
/* NOW STORED IN SCR(L+1B:LNEW, 1B). THIS AREA OF SCR */
/* CONTAINS NO DUPLICATE TERMS AND NO HIGH TERM */
/* INCLUDES ANY LOWER TERMS. THUS FINAL INCLUSION */
/* CHECK CONSIDERS ONLY LOWER TERMS */
NOT = 0B;
C: DO LA = L + 1B TO LNFW-1B;
DO NIX = LA + 1B TO LNFW;
DT4 = (BOOL(SUP(LA,1B),SCP(NIX,1B),'0110'B)) (BOOL(
   SCR(LA,1OR),SCP(NIX,1OR),'0110'B));
IF DT4 = '0'B THEN GO TO END_F;
IF (BOOL(BOOL(SUP(NIX,1B),SCR(NIX,1OR),'0110'B), DT4,
   '0100'B)) = '0'B THEN GO TO END_F;
/
/* LOWER TERM INCLUDES UPPER TERM */
END;
IF IN = INX THEN DO; NOT = NOT + 1B;
SCR(NOT,1B) = SCR(LA,1B);
SCR(NOT,1OR) = SCR(LA,1OR);
END; ELSE DO; DCNT = DCNT + 1B;
   IF DCNT > HROUND(DCARE, 1) THEN GO TO ERR;
DCARE(DCNT,1B) = SCR(LA,1B);
DCARE(DCNT,1OR) = SCR(LA,1OR);   END;
END_E: END E;
IF IN = INX THEN DO;
DCNT = DCNT + 1B;
IF DCNT > HROUND(DCARE, 1) THEN GO TO ERR;
DCARE(DCNT,1B) = SCR(LNEW,1B);
DCARE(DCNT,1OR) = SCR(LNEW,1OR);   END;
ELSE DO;
NOT = NOT + 1B;
SCR(NOT,1B) = SCR(LNEW,1B);
SCR(NOT,1OR) = SCR(LNEW,1OR);   END;
END_F: END F;
IN = INDEX(RITS,'0'B);
IF INX = 1B THEN DO; DCNT = DCNT + 1B;
DCARE(DCNT,1B) = ONES(M,1B,1B);
DCARE(DCNT,1OR) = CMES(M,1B,1OR);   END;
END_G: END G; FREE SCR;
Q: DO M = DCNT TO 1B BY -1B; DT1=DCARE(M,1B);
DT2 = DCARE(M, 1B8);
R: DO L=DCNT TO 1B BY -1B; IF M=L THEN GO TO END_R;
DT3 = DCARE(L,1B);
DT4 = DCARE(L,1OR);
IF BOOL(BOOL(DT1,DT3,'0110'B)) (BOOL(DT2,DT4,'0110'B)
   ) ,BOOL(DT3,DT4,'0110'B), '0010'B) = '0'B THEN DO;
DCARE(M,1B) = DCARE(DCNT,1B);
DCARE(M,1OR) = DCARE(DCNT,1OR);
DCNT=DCNT-1B; GO TO END_Q;   END;
IF BOOL(BOOL(DT1,DT3,'0110'B)) (BOOL(DT2,DT4,'0110'B)
   ) ,BOOL(DT3,DT4,'0110'B), '0010'B) = '0'B THEN DO;
DCARE(L,1B) = DCARE(DCNT,1B);
DCARE(L,1OR) = DCARE(DCNT,1OR);
DCNT=DCNT-1B; GO TO END_Q;   END;
END_R: END R;   END_Q: END Q;
PUT EDIT('DCARE TERMS ON EXIT')((SKIP(4), A);
DO M=1B TO DCNT; DNO = DNO + 1B;
DONTCAR (DNO,1B) = (DCARE(M,1B)) (INCOD(J,1B));
DONTCAR (DNO,1OR) = (DCARE(M,1OR)) (INCOD(J,1OR));
PUT EDIT(DONTCAR(DNO,1B),DONTCAR(DNO,1OR)) (SKIP(2), B,
   SKIP(0), B);   END;
PUT EDIT('ONES SUBCUBES LISTING FOLLOWS')(PAGE,A);
DO IN =1 TO JVAR;
PUT EDIT('INTERNAL STATE VARIABLE ', IN (SKIP(1), A, F(2)))
((ONESUB(IN, INX, 1R), ONESUB(IN, INX, 1O8) DO
INX = 1R TO ONESCNT(IN))) (SKIP(1), 3, SKIP(0), B);
END:
FREE DT1, DT2, DT3, DT4, SCR, ONES, PSUB, DCARE;
GO TO END_RJS;
ERR: PUT EDIT('SUBPROGRAM RJSMKI OVERFLOWED INTERMEDIATE',
DATA AREA CONTAINING DONT CARE PROD OF',
SUM TERMS OR INTERMEDIATE DATA GENERATING',
THEM SIZE OF THIS AREA=', AA, ' = AW*JVAR*',
'PSLGH THEREFORE INCREASE AW AND RERUN.')
(PAGE, A, A, SKIP(1), A, A, F(3), A);
OK = '0' B;
END_RJS: END RJSMKI;

RJSCBS: PROC;
*******************************************************************************/
/* THIS PROGRAM FINDS ALL PRIME IMPLICANTS OF AN */
/* ARBITRARY SUM OF PRODUCTS BOOLEAN EXPRESSION */
/* HAVING 'PTERMS' TERMS, EACH CONTAINING 'NOL' */
/* LITERALS. MASK CODING IS USED */
*******************************************************************************/
PUT EDIT('AT START OF RJSCPS V.7 EXECUTION TIME=',
TIME)(SKIP(2), A);
DECLARE /* DATA INPUT AND OUTPUT THIS ROUTINE */
PRIME(*,*) BIT(1) CONTROLLED EXTERNAL PACKED,
ONESUB(*,*,*) BIT(NOL) CONTROLLED EXTERNAL PACKED,
ONESCNT(*), BIN FIXED(7) CONTROLLED EXTERNAL,
DONTCAR(*,*) BIT(NOL) CONTROLLED EXTERNAL PACKED,
(PTERMS, OTERMS) BIN FIXED(15) EXTERNAL,
NOL EXTERNAL,
DNO BIN FIXED(15) EXTERNAL,
MINCOV(A) BIN FIXED(7) CONTROLLED EXTERNAL,
(ROWMAX, COLMAX) DEC FIXED(3) EXTERNAL,
ISV BIN FIXED(15) EXTERNAL,
OK BIT(1) EXTERNAL,
AY DEC FIXED(4,1) EXTERNAL;
PTERMS = DNO+ ONESCNT (ISV); OTERMS = PTERMS;
NOT5 = MIN(300, PTERMS*AY);
DECLARE (BITS, DD1, DD2, DT1, DT2, DML, DM2, DITS, FITS)
BIT(16) VARYING,
FLAG(NOT5) BIT(1) PACKED CONTROLLED,
PASS DEC FIXED(3) INITIAL(0),
(WON, NUTERMS) BIN FIXED(15);
ALLOCATE PRIME(10B, NOT5) BIT(NOL), FLAG;
PUT EDIT('PTERMS INPUT TO RJSCBS:')(SKIP(2), A);
DO II = 1B TO DNO;
PRIME(1B, II) = DONTCAP(II, 1B);
PRIME(1OB, II) = DONTCAP(II, 1OB);
PUT EDIT(PRIME(1B, II), PRIME(1OB, II))(SKIP(1), B, SKIP(0), B);
END;
DO II = DNO + 1B TO PTERMS;
PRIME(1B, II) = ONESUB(ISV, II-DNO, 1B);
PRIME(1OB, II) = ONESUB(ISV, II-DNO, 1OB);
PUT EDIT(PRIME(1B, II), PRIME(1OB, II))(SKIP(1), B, SKIP(0), B);
END;
START: NUTERMS = OB; FLAG = '0'B;
PUT DATA(PTERMS) SKIP(2);
/* CREATE CONSENSUS TERMS FOR PASS */
ONE: DO II = 1 TO OTERMS - 1B;
IF FLAG(II) = '1'B THEN /* TERM(II+) > TERM(II) */
GO TO END_ONE;
DT1 = PRIME(1B, II); DM1 = PRIME(1OB, II);
TWO: DO II = II + 1B TO OTERMS;
IF FLAG(II) = '1'B THEN GO TO END_TWO;
DT2 = PRIME(1B, II); DM2 = PRIME(1OB, II);
DD1 = (BOOL(DT1, DT2, '0110'B)) || (BOOL(DM1, DM2, '0110'B));
/* DD1 = 1 WHERE EITHER FIRST OR 2ND PARTS DISAGREE */
IF (BOOL(DD1), BOOL(DT2, DM2, '0110'B), '0010'B) = '0'B
THEN DO; /* TERM2 INCLUDES TERM1 */
FLAG(II) = '1'B; GO TO END_ONE; END;
BITS = BOOL(DD1, BOOL(DT1, DM1, '0110'B), '0010'B);
/* BITS = 1 WHERE POSSIBLE DISAGREES NOT COVERED BY */
/* *'S IN #1 */
IF BITS = '0'B
THEN DO; /* TERM1 INCLUDES TERM2 */
FLAG(II) = '1'B; GO TO END_TWO; END;
BITS = BOOL(BITS, BOOL(DT2, DM2, '0110'B), '0010'B);
/* BITS = 1 WHERE DISAGREES STILL NOT COVERED BY */
/* *'S IN #2 */
WON = INDEX(BITS, '1'B); IF WON > OB THEN DO;
BITS = SUBSTR(BITS, WON + 1B);
IF (INDEX(BITS, '1'B) > OB) THEN
/* CONSENSUS NOT DEFINED */
GO TO END_TWO; ELSE DO;
/* CONSENSUS DEFINED FOR THIS PAIR */
DD1 = BOOL(DT1, DT2, '0111'B);
SUBSTR(DD1, WON, 1B) = '0'B;
DD2 = BOOL(DM1, DM2, '0001'B);
SUBSTR(DD2, WON, 1B) = '1'B;
/* INCLUSION & DUPLICATION CHECK */
CK: DO J = 1B TO PTERMS;
IF FLAG(J) = '1'B THEN GO TO END_CK;
DT2 = PRIME(1B, J); DM2 = PRIME(1OB, J);
BITS = (BOOL(DT2, DD1, '0110'B)) || (BOOL(DM2, DD2, '0110'B));
IF BITS = '0'B THEN GO TO END_TWO;
/* TERM1 = THE NEW CON TERM */
IF INDEX(BOOL(BITS, BOOL(DT2, DM2, '0110'B), '0010'B),
IF INDEX(BITS,BOOL(DD1,DD2,'0111'B), '0010'B), '1'B) = OB THEN IFLAG(1) = '1'B;

NEW CON TERM INCLUDES TERM1 */ END CHK: END CHK;

IF PTERMS > NOTS THEN NO;
PUT EDIT("NUMBER OF CON TERMS EXCEEDS SIZE OF STORAGE!
"ALLOWED IN RJSCBS. PROGRAM MUST BE MODIFIED"
"BY INCREASING AY.")(SKIP(3),A,SKIP(1),A);

OK = 'O'B;
GO TO HALT ; END;

PRIME(1B,PTERMS) = DD1; PRIME(10B,PTERMS) = DD2;
IF IFLAG(II) = '1'B THEN GO TO END_ONE;

PTERMS = NTERMS + 1B;
IF PTERMS > NOTS THEN NO;
PUT EDIT("NUMBER OF CON TERMS EXCEEDS SIZE OF STORAGE!
"ALLOWED IN RJSCBS. PROGRAM MUST BE MODIFIED"
"BY INCREASING AY.")(SKIP(3),A,SKIP(1),A);

OK = 'O'B;
GO TO HALT ; END;

PRIME(1B,PTERMS) = DD1; PRIME(10B,PTERMS) = DD2;
IF IFLAG(II) = '1'B THEN GO TO END_FIE; OTERMS=OTERMS+1B;
PRIME(1B,OTERMS) = PRIME(1B,II);
PRIME(10B,OTERMS) = PRIME(10B,II);
END_FIE: END FIE; PASS=PASS+1; PTERMS=OTERMS;
NTERMS =OB THEN GO TO OUT ;
IF PASS>2*NO THEN GO TO EXIT; ELSE GO TO START;
EXIT: PUT EDIT ('ABNORMAL EXIT: INADEQUATE STORAGE ',
"FOR PASSES > 2 *NO LITERALS") (SKIP(2),A,A);

OK = 'O'B;
GO TO HALT ; END;

FREE FLAG;
HALT: END RJSCBS;

RJSPIT: PROC;
PUT EDIT('TIME ON ENTRY TO SUBPROGRAM RJSPIT = ',
TIME)(SKIP(4),A,A);

*******************************************************************************/
/* INPUT TO THIS SECTION IS A BOOLEAN S OF P
/* EXPRESSION IN COMPLETE SUM FORM, PLUS A LIST OF
/* SUBCURVES WHICH MUST BE COVERED BY TERMS OF THE
/* SUM. OUTPUT IS A SIMPLIFIED SUM OF PRODUCTS TERM*/
/* COVERING THE SUBCURVES. MASK CODING IS USED FOR
/* BOTH I/O AND INTERNAL DATA REPRESENTATIONS. */
*******************************************************************************/
DECLARE /* DATA INPUT AND OUTPUT THIS ROUTINE */
PRIME(*,*) BIT(1) CONTROLLED EXTERNAL PACKED,
ONESUB(*,*,*) BIT(NOL) CONTROLLED EXTERNAL
PACKED,
ISV BIN FIXED(15) EXTERNAL,
(JVAR,NOL) BIN FIXED(15) EXTERNAL,
ONESCNT(*) BIN FIXED(7) CONTROLLED EXTERNAL,
(PTERMS,OTERMS) BIN FIXED(15) EXTERNAL,
MINOVAR BIN FIXED(7) CONTROLLED EXTERNAL,
(PDMAX,CDLMAX) DEC FIXED(3) EXTERNAL,
OK BIT(1) EXTERNAL,
BITCODE BIT(5) EXTERNAL,
(AS,AT) DEC FIXED(4,1) EXTERNAL;
DTERMS = ONESCNT(ISV);
/* PTERMS = NO OF PRIME IMPLICANT TERMS */
/* OTERMS = NO OF 1-SUBCUBE TERMS */
/* NOL = NO OF LITERALS PER TERM */
DECLARE (SUBSCR,NOEPI) BIN FIXED(15),
(PTERMS,OTERMS,STRING(PTERMS,
OTERMS)) BIT(1) PACKED CONTROLLED,
EPRIME(10B,OTERMS) BIT(NOL) PACKED CONTROLLED,
CHECK(AT+1) BIT(PTERMS) CONTROLLED,
MNCOV(SUBSCR,NOEPI+NOL,10B) BIT(K) CONTROLLED
PACKED,
(BITS,DT1,DM1,DT2,DM2) BIT(NOL) CONTROLLED,
DITS(OTERMS) BIT(OTERMS) CONTROLLED,
VECTOR(AT+1,OTERMS) BIN FIXED(15) CONTROLLED;
ALLOCATE PFLAG,OFLAG,EPIMF,STRING,BITS,DM1,DM2,DT1,
DT2;
PUT EDIT ('PRIME IMPLICANT TERMS FOLLOW:')(SKIP(4),A);
DO I = 1 TO PTERMS ;
PFLAG(I) = '0'8;
PUT EDIT(PRIME(1B, I),PPIMF)(STRING(I),B,
SKIOR),B);
END;
PUT EDIT('ONES SUBCUBES LISTING FOLLOWS:')(SKIP(4),A);
/* BEGIN ESSENTIAL PRIME IMPLICANT SEARCH */
RED: DO I = 1 TO OTERMS ;
OFLAG(I) = '0'8;
DT1 = ONESUB(ISV, I,18); DM1 = ONESUB(ISV, I,10B);
NOEPI = OB;
PUT EDIT(DT1,DM1)(SKIP(1),B,SKIP(0),B);
BLK: DO J = 1 TO PTERMS ;
STRING(J, I) = '0'8; DT2 = PRIME(1J, J);
DM2 = PRIME(10B, J))
BITS = /* DIFFERS BTN TERM1 & TERM2 */((BOOL(DT1,
DT2,'0110'8))||(BOOL(DM1,DM2,'0110'8)))
IF ((BOOL(BITS,BOOL(DT2,DM2,'0110'8), '0010'8))='0'8)
/* TERM2 INCLUDES OR EQUALS TERM1 */ THEN DO;
STRING(J, I) = 1B; NOEPI = NOEPI+1;
/* NOEPI HERE COUNTS NO OF TERMS COVERING TERM1 */
SUBSCR = J; /* STORES LAST TERM2 COVERING TERM1 */
END; END_BLK: END BLK;
IF NOEPI = OB THEN DO;
PUT EDIT('ERROR--NO PRIME IMPLICANT TERM COVERS ONES',
'TERM', I,ONESUB(ISV, I,18),ONESUB(ISV, I,10B))
PAGE,A,A,F(2),X(4),B,X(1),B);
OK = '0'B;  GO TO END;  END;
IF NOEPI = 1B THEN DO; /* TERM? AN EPI DUE TO TERM? */
PFLAG(SUBSCR) = '1'B;  OFLAG(I) = '1'B;  END;
END_REFD:  END_REFD;
/* EPI TERMS HAVE PFLAGS SET = 1; ALL SUBCUBES COV'D */
/* BY EACH PI ARE RECORDED IN ARRAY STRING. NOW */
/* ELIMINATE ALL EPI'S AND INCLUDED 1-SUBCUBES */
NOEPI = 0R; /* HERE COUNTS NO. OF EPI'S */
PUT EDIT('ESSENTIAL PRIME IMPLICANT TERMS :')
  (SKIP(4),A);
BLU:  DO I = 1 TO PTERMS;
  IF PFLAG(I) = '1'B THEN DO;
  NOEPI = NOEPI +1;
  EPRIME(1,N0EPI)=PRIME(1,I);
  EPRIME(2,N0EPI)=PRIME(2,I);
  PUT EDIT(PRIME(18,I),PRIME(10B,I))(SKIP(I),B,
    SKIP(0),B);
  END;  ELSE DO;
  NOL = I - NOEPI;
  PRIME(1B,NOL) = PRIME(1B,I);
  PRIME(7,NOL) = PRIME(7,I);
  PFLAG(NOL) = '0'B;  END;
WHT:  DO J = 1 TO OTERMS;
  IF OFLAG(J) = '0'B THEN DO;
  /* LOOP TO ELIM 1-SUBCUBES < EPI PRIME(I) */
  IF STRING(I,J)='1'B THEN OFLAG(J) = '1'B;  ELSE;  ELSE
  /* PRIME(J) = EPI SO RELOCATE STRING(I,J) */
  STRING(NOL,J) = STPING(I,J);
  END_WHT:  END_WHT;
  END_BLU:  END BLU;
PTERMS = PTERMS - NOEPI;
PUT DATA(PTERMS);
/* ALL EPI'S HAVE BEEN REMOVED TO EPRIME, ALL */
/* SUBCUBES COVERED BY THEM HAVE OFLAG SET =1, & NON*/
/* ESSENTIAL PI'S OCCUPY TOPMOST (PTERMS-N0EPI) */
/* LOCATIONS OF PRIME. (DITTO CORRESPONDING */
/* COVER DATA IN STRING(*,*) */
I = 0B;
PUT EDIT('NONCRITICAL ONES SUBCUBES :')
  (SKIP(4),A);
GRY:  DO J = 1 TO OTERMS;
  IF OFLAG(J) = '0'B THEN DO;
  PUT EDIT(ONESUB(ISV,J,1B),ONESUB(ISV,J,1OB))(SKIP(I),
    B,SKIP(0),B);
  I=I+1B;  ONESUB(ISV,I,1B)=ONESUB(ISV,J,1B);
  ONESUB(ISV,I,1OB)=ONESUB(ISV,J,1OB);
  DO K=1 TO PTERMS;  STRING(K,I)=STRING(K,J);  END;  END;
END_GRY: END_GRY;
/* ONE'S CONTAINS ONLY SUBCUBES NOT COVERED BY EPI'S.*/
/* STRING MODIFIED. USE BRANCHING TECHNIQUE TO FIND*/
/* MINIMUM COVERS FOR REMAINING 1-SUBCUBES */
IF I = OR THEN DO;  PUT EDIT('')
  'ALL SUBCUBES COVERED BY EPI TERMS'  (PAGE,A):
  SUBSCR = 1B;  NOL = .OR;  GO TO OUTPUT;  END;
OTERMS=I;
PUT EDIT('NONCRITICAL PRIME IMPLICANTS AND COVER ',
  'MATRIX :')
  (SKIP(4),A,A);
DO I = 1 TO PTERMS;
PUT EDIT(PRIME(1,I), PRIME(2,I), (STRING(I,J) DO J=1
  TO OTERMS))(SKIP(I),B,SKIP(0),B,X(4),(OTERMS)(
    X(1),B(1)));
END;
/* VECTOR(SUBSCR,*) CONTAINS MINIMUM COVER CANDIDATE */
/* HIGHER VECTOR ROWS CONTAIN PREV. FOUND COVERS */
/* LGH = COVER LGH, DITS(LGH) RECORDS SUBCURES NOT */
/* COVERED BY CURRENT BRANCH FOR EACH STEP */

NOL = OTEMRNS; SUBSCR = 1B;
ALLOCATE VECTOR, CHECK, DITS;
HI: DO I = 1 TO PTERMS; L = I;
IF STRING(L,1) = '0'B THEN GO TO END_HI;
K = 1B; LGH = OR;
DITS = SUBSTR((60) '1'B, I, OTERMS);
TOP: LGH = LGH +1B;
IF LGH>NOL THEN DO; LGH= LGH -1B;
L=VECTOR(SUBSCR, LGH ); K=INDEX(DITS(LGH -1), '1'B);
GO TO DOWN; END;
IF LGH>1B THEN DITS(LGH) = DITS(LGH-1B);
VECTCR(SUBSCR, LGH) = L;
DO J = K TO OTERMS ; IF STRING(L,J) = '1'B THEN
SUBSTR(DITS(LGH), J, 1) = '0'B; END;
IF DITS(LGH) = '0'B THEN GO TO LO;
K = INDEX(DITS(LGH), '1'B); L = OR;
MID: L = L +1B; IF L> PTERMS THEN DO;
L=VECTOR(SUBSCR, LGH ); K=INDEX(DITS(LGH -1), '1'B);
GO TO DOWN; END;
IF STRING(L,K) = '1'B THEN GO TO TOP;
ELSE GO TO MID;
DOWN: /* STEPS TO NEXT LOWER PR IN SAME COLUMN */
LGH = LGH -1B; IF LGH <1B THEN GO TO END_HI;
GO TO MID;
LO: /* ENTERED ONLY IF COVER OF LGH < NOL FOUND */
IF LGH < NOL THEN DO;
DO M = 1B TO LGH;
VECTOR(1, M)=VECTOR(SUBSCR, M); END;
NOL=LGH; SUBSCR=1OB; END;
ELSE SUBSCR = SUBSCR +1B;
CHECK(SUBSCR-1B) = '0'B;
DO M = 1B TO LGH;
SUBSTR(CHECK(SUBSCR-1B),VECTOR(SUBSCR-1B ,M),1)= '1'B;
VECTOR(SUBSCR, M) = VECTOR(SUBSCR-1, M); END;
DO M = 1B TO SUBSCR - 10B;
IF CHECK(M)=CHECK(SUBSCR-1B) THEN DO;SUBSCR=SUBSCR-1B;
GO TO DOWN; END; END;
/* THIS POINT REACHED IFF COVER(SUBSTR-1) IS UNIQUE */
IF SUBSCR>AT THEN FULL: SUBSCR = SUBSCR-1B;
GO TO DOWN;
END_HI: END_HI ;
SUBSCR = SUBSCR-1B;
/* AT THIS POINT THERE ARE 'SUBSCR' MINIMUM COVERS */
/* EACH CONTAINING 'NOL' TERMS STORED IN VECTOR. */
/* EACH OF THESE, WHEN ADDED TO THE EPRIIMES, IS A */
/* MINIMAL SOLUTION TO THE INPUT PROBLEM. */
OUTPUT: FREE PFLAG,OFLAG,STRING,DM1,DM2,DT1,DT2,CHECK;
K = LENGTH(PRIME(1,1));
PUT DATA(SUBSCR,NOFPJ,NOL);
ALLOCATE MNCOV;
PUT EDIT('GENERATED MINIMAL DESIGN EQUATIONS FOR ';
  'INTERNAL STATE ',ISV,' FOLLOW:')(PAGE,A,A
  ,F(2),A);
NOCOV = NOEPI + NOL;
DO I = 1 TO SUBSCR ;
PUT EDIT('COVER NO', I, ' LISTING FOLLOWS')(SKIP(2),
  \$F(2),X(2),A);
DO J = 1 TO NOL;
MNCOV(I, J, 1) = PRIME(1, VECTOR(I , J));
MNCOV(I, J, 2) = PRIME(2, VECTOR(I , J));
PUT EDIT(MNCOV(I,J,1),MNCOV(I,J,109))(SKIP(1), B,
  SKIP(0), B); END;
DO J = NOL + 1 TO NOCOV;
MNCOV(I, J, 1) = EPRIME ( 1, J-NOL);
MNCOV(I, J, 2) = EPRIME ( 2, J-NOL);
PUT EDIT(MNCOV(I,J,1),MNCOV(I,J,109))(SKIP(1), B,
  SKIP(0), B); END; END;
FREE PRIME, EPRIME, VECTOR, DITS;
/* MNCOV NOW CONTAINS 'SUBSCR' COVERS OF NOEPI+NOL */
/* TERMS EACH. NOW CONVERT THESE COVERS TO LITERAL */
/* NOTATION AND PRINT THEM IN LITERAL EQUATION FORM */
RJSOUT: BEGIN;
DECLARE LINE CHAR(100) VARYING,
  ALFA(10,2) CHAP(1);
/* INITIALIZED EXPLICITLY ONLY BECAUSE OF V.2 REL 11 */
/* COMPILER LIMITATIONS */
ALFA(1,1)='A'; ALFA(2,1)='R'; ALFA(3,1)='C';
ALFA(5,1)='E'; ALFA(6,1)='T'; ALFA(7,1)='G';
ALFA(9,1)='J'; ALFA(10,1)='K'; ALFA(1,2)='V';
ALFA(3,2)='X'; ALFA(4,2)='Y'; ALFA(5,2)='Z';
ALFA(7,2)='R'; ALFA(8,2)='S'; ALFA(9,2)='T';
ALFA(6,2)='Q'; ALFA(2,2)='W'; ALFA(9,1)='H';
ALFA(4,1)='D'; ALFA(10,2)='U';
DO I = 1 TO SUBSCR ; /* LOOP THRU COVERS */
PUT SKIP(3);
LINE = (ALFA(ISV,1R))|(( = ('.');
IF ONECNT ( ISV ) = OR THEN DO;
LINE = ((LINE))||('0'));
PUT EDIT(LINE)(SKIP(1), A );
  GO TO END_RJSOUT ; END;
IF (SUBSCR = 1) & (MNCOV(I,1,1)= (~MNCOV(I,1,2)))
  THEN DO; LINE = ((LINE))||('1'));
  PUT EDIT(LINE)(SKIP(1), A );
  GO TO END_RJSOUT ; END;
DO J = 1 TO NOCOV ; /* LOOP THRU TERMS OF COVER */
IF LENGTH(LINE) >85 THEN DO;
PUT EDIT(LINE)(SKIP(1), A);
LINE = (' ');
END;
BITS = BOOL(MNCOV(I,J,1), MNCOV(I,J,109),1001'B);
K = INDEX(BITS,'1'B);
IF J = 1 THEN LINE =((LINE))||(' ')) ELSE
LINE =((LINE))||(' + '));
DO WHILE (K>0); /* LOOP THRU SIG LITS IN TERM */
  IF K > JVAR THEN DO;
K2 = 10B;  K = K - JVAP:  END;
ELSE K2 = 1B;
IF SUBSTR( MNC0V{I,J,1},K+JVAR*(K2-1B),1) = 'L'9 THEN
    LINE=((LINE)||(ALFA(K,K2)));
ELSE
    LINE=((LINE)||(ALFA(K,K2))||""');
END:
SUBSTR(BITS,K+JVAR*(K2 -1B),1) = '0'8;
K = INDEX(BITS,'L'9);
END:  END;
LINE = (((LINE)||("'"')));
PUT EDIT(LINE)(SKIP(1), A);
END_RJSOUT: END RJSOUT;
FREE MNC0V;
END: END RJSPIIT;
APPENDIX II: WORKED EXAMPLES

The examples listed below are representative of the size and complexity of circuits designed using the programs described in this paper. Intermediate data and execution time information is given to further illustrate the programs' operation.

Most of the execution times listed are for a 131K byte S/360/40. It has, however, been necessary to run some of the examples on a S/360/50. Model 50 execution times shown below will be marked "*". Since the efficiency of both hardware and software are constantly being improved, the execution times listed here should be considered only as upper bounds for solution of the problems described.

Example 1
Summary Characteristics:

1. Flow table: 3 columns by 6 rows (Machine A of Figure 1).
3. State Assignment Generation: Algorithm #2 found 3 codes (2 had 4 variables, 1 had 3 variables) in total time of 26 seconds.
4. Next-state Equations for Code #1 (4 state variables):
   - Total number of unspecified terms 6
   - Total number of 1-sets 16
   - Generation time 11 seconds
5. Prime Implicant Generation (Code #1, 7 variable terms):
6. Covering Problem:

<table>
<thead>
<tr>
<th>Internal State Variable</th>
<th>Prime Implicants Input</th>
<th>1-sets Input</th>
<th>Terms (Literal)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>7</td>
<td>2(3)</td>
<td>3.80 sec.</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1(1)</td>
<td>2.80</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>1</td>
<td>1(3)</td>
<td>3.20</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>2</td>
<td>2(6)</td>
<td>3.20</td>
</tr>
</tbody>
</table>

7. Simplified next-state equations:

\[
Y_1 = y_1'x + y_1
\]
\[
Y_2 = w'
\]
\[
Y_3 = y_1y_3x
\]
\[
Y_4 = y_4 + w'x' + y_1y_3x
\]

(where primary input variables are \( v, w, \) and \( x \)).
Example 2
Summary Characteristics:

1. Flow table: 4 columns by 6 rows as shown below:

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>


3. State assignment generation: Algorithm #1 found 10 codes each having 5 variables in total time of 101 seconds. Algorithm #2 found 10 codes, 8 having 6 variables and 2 having 5 variables, in total time of 52 seconds.

4. Next-state equations:

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Total # 1-sets</th>
<th>Total # Unspecified</th>
<th>Generation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31</td>
<td>17</td>
<td>23.0 sec.</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>12</td>
<td>21.5</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>12</td>
<td>21.0</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>14</td>
<td>22.0</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>18</td>
<td>23.2</td>
</tr>
<tr>
<td>6</td>
<td>29</td>
<td>16</td>
<td>25.0</td>
</tr>
<tr>
<td>7</td>
<td>33</td>
<td>26</td>
<td>33.3</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>16</td>
<td>**</td>
</tr>
<tr>
<td>9</td>
<td>31</td>
<td>24</td>
<td>28.4</td>
</tr>
<tr>
<td>10</td>
<td>29</td>
<td>24</td>
<td>30.0</td>
</tr>
</tbody>
</table>

**Time not available.
5. Prime Implicant Generation (Code #1, 8 variable terms):

<table>
<thead>
<tr>
<th>Internal State Variable</th>
<th>Terms Input</th>
<th>Maximum Intermediate Steps</th>
<th>Prime Implicants Generated</th>
<th>Time Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>32</td>
<td>18</td>
<td>4:05 min.</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>51</td>
<td>23</td>
<td>10:24</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>53</td>
<td>19</td>
<td>9:32</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>48</td>
<td>17</td>
<td>7:15</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>60</td>
<td>15</td>
<td>11:19</td>
</tr>
</tbody>
</table>

6. Covering Problem:

<table>
<thead>
<tr>
<th>Internal State Variable</th>
<th>Prime Implicants Input</th>
<th>l-sets Input</th>
<th>Terms(Literals) in Simplified Equation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>4</td>
<td>2(5)</td>
<td>5.50 sec.</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>6</td>
<td>6(16)</td>
<td>8.00</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>7</td>
<td>5(14)</td>
<td>6.50</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>7</td>
<td>4(10)</td>
<td>7.10</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>7</td>
<td>5(12)</td>
<td>6.50</td>
</tr>
</tbody>
</table>

7. Simplified Next-state equations (based on code #1):

\[
Y_1 = y_4 w x' + y_1 w'
\]
\[
Y_2 = y_4' w x' + y_2' w' x' + y_5 w x' + y_1 w' + y_2 y_5 + y_3' x w'
\]
\[
Y_3 = y_1' y_2' w' x' + y_4' w x' + y_1 w x + y x + y x
\]
\[
Y_4 = y_4 x' + w' x' + y_2' y_3 w x + y_1 w'
\]
\[
Y_5 = y_4' w x' + y_2' w' x' + y_1 w' + y_3' x + y_5' w
\]

(where primary input variables are v, w, and x).
Example III
Summary Characteristics:

1. Flowtable: 4 columns by 12 rows (Liu's Figure 7, page 205 of reference 2). 41 cells are specified.

2. Partitioning: 61 partitions and 31 complements generated in 1:45 minutes.

3. State assignment generation: assignment algorithm #2 found 3 codes of 5 variables each in total time of 5:33 minutes. (Algorithm #1 had not completed generation of maximal intersectables after 71* minutes.)

4. Next-state equations for code #2:

   | Total number of unspecified terms | 15 |
   | Total number of 1-sets            | 59 |
   | Generation time                   | 31 sec. |

5. Prime implicant generation (Code #2, 9 variables per term):

<table>
<thead>
<tr>
<th>Internal State Variable</th>
<th>Terms Input</th>
<th>Maximum Intermediate Steps</th>
<th>Prime Implicants Generated</th>
<th>Time Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>33</td>
<td>20</td>
<td>4:46 min.</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>28</td>
<td>12</td>
<td>2:43</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>43+</td>
<td>**</td>
<td>6:01</td>
</tr>
</tbody>
</table>

*Execution terminated due to excessive job time.

6. Covering problem (state variables 1 and 2 only):

<table>
<thead>
<tr>
<th>Internal State Variable</th>
<th>Prime Implicants Input</th>
<th>1-sets Input</th>
<th>Terms (18) in Simplified Equation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>13</td>
<td>6(18)</td>
<td>10.0 sec.</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>13</td>
<td>5(14)</td>
<td>7.0</td>
</tr>
</tbody>
</table>
Example IV
Summary Characteristics:

1. Flow table: 4 columns by 18 rows; 72 cells are specified.
3. State assignment generation: algorithm #2 found 2 codes (7 variables each) in total time of 16:15* minutes.
BIBLIOGRAPHY


VITA

The author was born on May 12, 1944, in San Francisco, California. He received his primary and secondary education in Wichita, Kansas. He received a Bachelor of Science Degree in Electrical Engineering (cum laude) from Wichita State University in June, 1966.

He has been enrolled in the Graduate School of the University of Missouri at Rolla since September, 1966, and has held a National Defense Education Act Title IV Fellowship since that date. The author is a member of Eta Kappa Nu and Tau Beta Pi.