Missouri University of Science and Technology
Scholars' Mine

Electrical and Computer Engineering Faculty Research & Creative Works

Electrical and Computer Engineering

30 Apr 1968

# Automation In The Design Of Asynchronous Sequential Circuits

Richard T. Smith
*Missouri University of Science and Technology*

James H. Tracey
*Missouri University of Science and Technology*

W. L. Schoeffel

G. K. Maki

Follow this and additional works at: https://scholarsmine.mst.edu/ele_comeng_facwork

 Part of the Electrical and Computer Engineering Commons

## Recommended Citation

R. T. Smith et al., "Automation In The Design Of Asynchronous Sequential Circuits," *AFIPS 1968 Conference Proceedings - Spring Joint Computer Conference*, pp. 55 - 60, Associatoin for Computing Machinery, Apr 1968.
The definitive version is available at https://doi.org/10.1145/1468075.1468084

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

# Automation in the design of asynchronous sequential circuits*

by R. J. SMITH, II, J. H. TRACEY, W. L. SCHOEFFEL and G. K. MAKI
*University of Missouri at Rolla*
Rolla, Missouri

## INTRODUCTION

Sequential switching circuits are commonly classified as being either synchronous or asynchronous. Clock pulses synchronize the operations 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. One disadvantage of asynchronous circuit design has been the complexity of the synthesis procedures for large circuits.

This paper describes a computer program[1,2] which automatically generates the complete set of design equations for asynchronous sequential circuits. Many of the algorithms employed are new and have been shown to be much more practical than classical techniques for the synthesis of large circuits.

Minimum or near-minimum variable internal state assignments are generated using two of the Tracey algorithms.[3] An evaluation procedure predicts which of several codes generated will most likely yield the least complex design equations. Next-state equations, including don't-cares, are then produced without constructing transition tables. Output-state equations are also generated. Finally, simplified normal form design equations containing no static hazards are produced. The program is capable of designing circuits much too large to design manually.

The operation of a sequential circuit is often described by means of a flow table.[4] An example is shown in Figure 1. The columns of a flow table represent input states, while the rows represent internal states assumed by the circuit. Each flow table entry specifies the next internal state and output which result from the given input and internal states. When the next-state equals the present state, that state is said to be stable and is customarily circled. Unstable states correspond to transitions within a flow table column.

| | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|
| 1 | (1)/10 | - | 4 |
| 2 | 3 | 4 | (2)/01 |
| 3 | (3)/01 | (3)/00 | - |
| 4 | 1 | (4)/11 | (4)/00 |
| 5 | 1 | (5)/10 | 2 |

Figure 1 – Flow table

A sequential circuit is operating in fundamental mode if the inputs are never changed unless the circuit is stable internally. If, in addition, each unstable state leads directly to a stable state, the circuit is said to be operating in normal fundamental mode. The computer program described in this paper automatically generates design equations for asynchronous sequential circuits operating in the normal fundamental mode. Circuit specifications are conveniently input in flow table form. Input state binary codes are specified by the program user. A summary flow chart of the procedure followed by the synthesis algorithm is shown in Figure 2.

*The research reported in this paper was supported in part by the National Science Foundation through Grant GK-820.

The program used to implement the design procedure suggested above is written in PL/1. This language was chosen because of its bit-string data format, Boolean operations, and the controllable storage feature. The program consists of about 2000 PL/1 statements divided into 8 subroutines. The system was designed to run on an IBM 360/40 but could be run on a somewhat smaller machine.

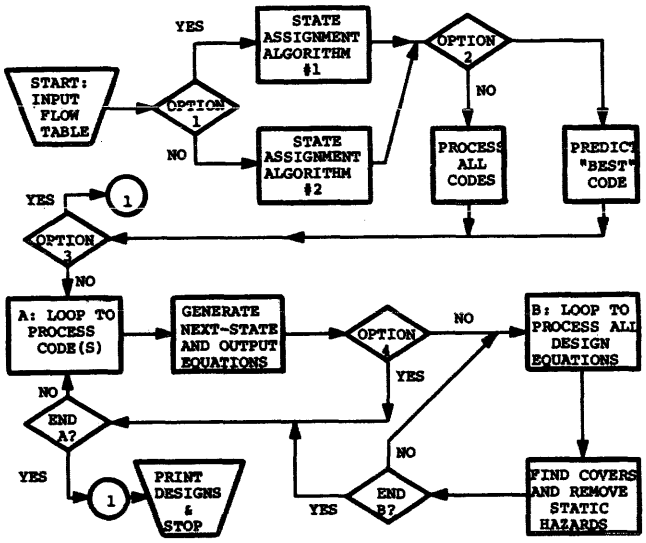


Figure 2 – The programmed synthesis algorithm

*State assignment algorithm*

The internal state assignment procedures employed are a modification of those described by Tracey.[3] Either completely or partially simplified flow tables may be input to the program. Flow table simplification is not presently included in the synthesis procedure, but will be included in a later version of the program.

The Tracey state assignment algorithms are based on the following theorem which is reproduced without proof. "A row assignment to a flow table which allots one internal state per row is satisfactory for the realization of normal fundamental mode 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 internal state variable partitions the pair $\{S_i, S_j\}$ and the pair $\{S_m, S_n\}$ into separate blocks; b) if $S_k$ is a stable state not involved in any transitions in the column, then at least one internal state 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 internal state variable partition."

Constraints generated as a result of applying the above theorem may be listed in Boolean matrix form, with each row corresponding to a partially specified state variable. Consider, for example, the constraint list generated by the flow table of Figure 1. The constraints shown below are generated on a per-column basis in satisfying theorem parts a) and b):

| Constraint List | Boolean Matrix 12345 |
|---|---|
| (14; 23) | 0110- |
| (15; 23) | 011-0 |
| ( 3; 24) | -101- |
| (24; 5) | -0-01 |
| (25; 14) | 10-10 |
| ( 1; 4) | 0--1- |

The program forms all partitions associated with the topmost stable state of column 1, then all irredundant partitions due to the second state, and continues until all stable states in the column have been examined. The process is then repeated for the remaining columns.

Note that for the above example, none of the column-generated constraints partitioned flow table rows 1 and 4; the constraint (1;4) was thus included in the list to satisfy theorem part c). The program checks all pairs of flow table rows, and generates additional partitions as required by c).

The state assignment problem now becomes one of finding a minimum number of internal state variables satisfying all of the constraints just generated. This problem has been shown to be analogous to the generation of maximal compatibles by the Paull-Unger algorithms for flow table simplification. A set of completely specified state variables, at least one of which covers each constraint, corresponds to the maximal compatibles. These completely specified state variables will be referred to here, therefore, as maximal constraints. The selection of a minimum number of maximal constraints, and hence minimum number of internal state variables is similar to the covering problem in the Quine-McCluskey method for simplifying Boolean equations. Details of these algorithms are available in the literature and will not be given here.[3,5,6]

As in the Paull-Unger Method A,[5] maximal constraint generation may begin with the assumption that no constraints exist. Then each constraint is examined for contradictions to this assumption and all implied partitions are generated. After each constraint has been so examined, one is left with the set of maximal constraints. In another approach, Paull-Unger Method B, one begins with the list of constraints and seeks

to enlarge each through the complete specification of the corresponding state variable until all enlargements are found that cover one or more of the original constraints. Both procedures were programmed and the second was found to be nearly a factor of two faster than the first in this application.

As stated previously, the selecting of a minimum number of maximal constraints is similar to the covering problem in Boolean equation simplification. A branching method has been used which is capable of producing all irredundant minimum-variable assignments. Operating speed of this algorithm is increased by reorganization of the maximal constraint list, based on the idea that those maximal constraints including the largest number of the original constraints would most likely be members of a minimal cover. Internal representation of each maximal constraint is restructured in such a manner that the covering problem cannot be further simplified using column dominance techniques.

It is obvious that either of the two assignments below satisfy all of the constraints shown above:

| Assignment #1 | | | | | | Assignment #2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | | 1 | 2 | 3 | 4 | 5 |
| $y_1$ | 0 | 1 | 1 | 0 | 0 | $y_1$ | 0 | 1 | 1 | 0 | 1 |
| $y_2$ | 0 | 1 | 1 | 0 | 1 | $y_2$ | 0 | 1 | 1 | 1 | 0 |
| $y_3$ | 0 | 1 | 0 | 1 | 0 | $y_3$ | 0 | 1 | 0 | 1 | 0 |

Furthermore, these are the only two significantly different minimum assignments which successfully code the Figure 1 flow table.

It has been found that even with certain look-ahead provisions in the branching routine, generation of minimum variable assignments becomes a time-consuming problem for typical flow tables of 12 rows or more. A second and much faster algorithm has been programmed. It is an approximate method, and generates near-minimum variable codes.

The fast algorithm reduces the Boolean matrix corresponding to the maximal constraints through the use of an approximate reduction technique. A constraint is constructed which seems to include a large number of matrix rows. The included matrix rows are then removed. This process is then continued until all rows of the original matrix are included in at least one of the generated constraints. This reduced matrix corresponds to a near-minimum variable state assignment.

The fast Boolean matrix reduction program usually produces satisfactory assignments having less than 1⅓ times the minimum number of variables. Assignment generation times for large flow tables may be reduced by two orders of magnitude using this approximate procedure. Near-minimal assignments have been efficiently generated for flow tables having up to 75 specified next-state entries and 150 constraints with approximately 15 minutes computer time on an IBM 360/50. Many satisfactory assignments are often generated. One of these may be selected by a test routine or chosen by the designer. The test routine, to be discussed below, chooses a "good" assignment for reduced hardware realization.

*Design equations*

As the example above illustrates, the code generating algorithms frequently produce several satisfactory assignments. Generated codes may be evaluated by a procedure due to Maki,[7] which selects that assignment most likely to have simple next-state equations. Consider, for example, the assignments and next-state equations shown in Figure 3. Note that the next-state equations for column $I_1$ Assignment #1 are much less complex than those for Assignment #2.

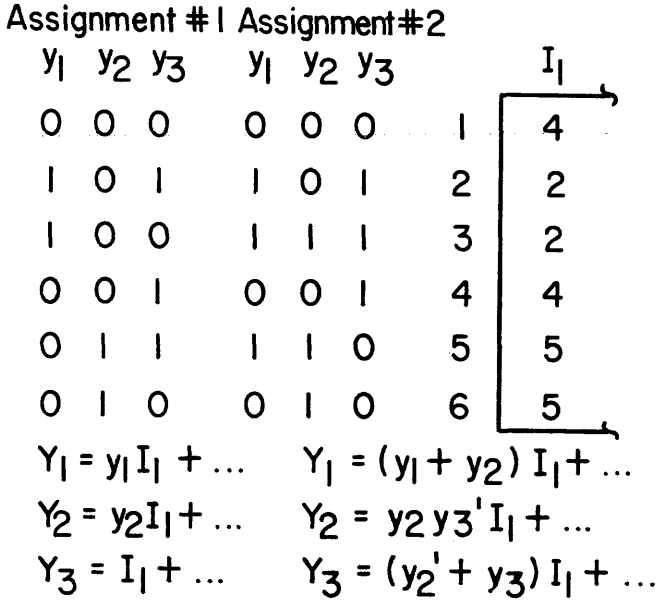


Figure 3 – Partial flow table and assignments

The test procedure searches for that assignment with a maximum amount of reduced dependency in the next-state equations. Two types of reduced dependency are easily detected from the assignment. First, observe that $Y_3$ is dependent only on the input in the given example. This can be predicted by noting that $y_3$ has the same value, 1, for all stable states in the column. A second observation is that $Y_1$ is dependent only on the input and the present state of $y_1$. Similarly, $Y_2$ is a function only of $y_2$ and the input. This can also be predicted by simply noting that $y_1$ and $y_2$ are never excited to change state for any transition under input $I_1$. In other words, one need simply observe that $y_1$

and $y_2$ have the same value for state pair (1, 4), state pair (2, 3) and again for state pair (5, 6). Observe the increased complexity of the next-state variables in Assignment #2 of Figure 3 as a result of its failure to insure reduced dependency. The programmed routine based on this method will evaluate each generated state assignment for reduced dependency in just a few seconds.

Maki has also described a procedure for obtaining next-state equations without construction of the traditional excitation matrix.[7] An algorithm derived from his method is presented here.

Each internal state transition may be associated with a p-subcube of the n-cube defined by the input and internal state variables. Furthermore, all of the next-state entries of p-subcubes associated with a single stable state will be identical, and equal to the row code of the stable state. Consider, for example, the application of Assignment 1 to Column $I_1$ of Figure 3, as shown in Figure 4.

In the transition between rows 2 and 3, all states in the p-subcube $y_1y_2$ ($y_1 = 1$, $y_2 = 1$) must have the same next-state entries, namely that of stable state 3, 110. A tabular form of p-subcube generation may be illustrated as follows:

| $y_1$ | $y_2$ | $y_3$ | |
|---|---|---|---|
| 1 | 1 | 0 | ③ Stable Row Code |
| 1 | 1 | 1 | 3 Unstable Row Code |
| 1 | 1 | - | P-Subcube Resulting from Transition |

The transitions from rows 4 and 5 to stable state 1 define the remaining two p-subcubes listed in $P_{I_1}$ of Figure 3. Note that the Boolean sum $P_{i_1}$ of these terms represent all next states requiring specified entries under input $I_1$.

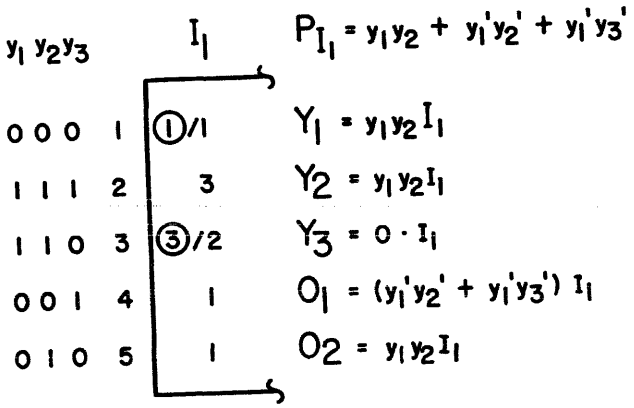


Figure 4 – Partial flow table, specified p-subcubes and 1-sets

Notice that if $y_i$ is 1 in the stable state row code, then next-state variable $Y_i = 1$ for *all* states in p-subcubes associated with that stable state. For example, since in row 3 $y_1 = 1$, all states in the p-subcube $y_1y_2$ will have next-state variable $Y_1 = 1$. In other words, all p-subcubes associated with transitions to a stable state will appear in the Boolean sum-of-products next-state equation $Y_i$ *if digit i of the stable row code is one.*

As the p-subcubes are generated by the computer program, they are added to the appropriate next-state 1-set lists only if the corresponding next-state variable is 1 in the subcube (see Figure 3). The final results are (partially simplified) Boolean equations representing the 1-cells of the next-state variables.

The synthesis program also generates the output equations of the sequential circuits. The output corresponding to a given stable state is also associated with all unstable states leading to the stable state. All p-subcubes generated previously are grouped according to stable state. If an output variable is 1 for a particular stable state, the associated p-subcubes become a partial list of 1-sets under the corresponding input. The output 1-sets for Column $I_1$ in Figure 1 are shown as sums in Figure 3.

To permit further simplification of the design equations generated above, it is desirable to compute the unspecified entries for all equations. Fortunately, unspecified p-subcubes are common to all the design equations. A Boolean equation for don't-care entries is generated by simply taking the complement of the available equation for specified entries (see Equation $P_{I_1}$ in Figure 3).

Complementation of a Boolean sum-of-products expression may be performed by complementing the expression, multiplying out the result, then simplifying the resultant sum-of-products expression to obtain the solution. The procedure used here is a modification of that method. Simplification illustrated by

$$A\cdot(A + B + C) = A \qquad \text{and}$$
$$A\cdot(A' + D + E) = A(D + E)$$

is performed both before and during the multiplication of the product-of-sums expression. Redundant terms are also deleted. A brief example will perhaps illustrate the method employed. Figure 5 shows complementation of the sum of p-subcubes shown as $P_{I_1}$ in Figure 3.

A normal-form Boolean equation for each next-state variable may be obtained by combining the don't-care terms found above with the appropriate next-state 1-sets. Since the output associated with don't-care internal states may be assumed to be unspecified, the output-state equations also include the same don't-care terms.

$$P_{I_1} = y_1y_2 + y_1'y_2' + y_1'y_3'$$
$$P'_{I_1} = (y_1' + y_2') \cdot (y_1 + y_2) \cdot (y_1 + y_3)$$
$$= y_1(y_2') + y_2 + y_3 \cdot (y_1' + y_2') \cdot (y_1 + y_2)$$
$$= y_1y_2' + y_1 + (y_1'y_3 + y_2'y_3) \cdot (y_1 + y_2)$$
$$= y_1 + y_1'y_2y_3$$

Figure 5 – P-subcube complementation and simplification

The program then finds prime implicants of each design equation produced above. A conventional consensus algorithm is used and will not be presented here.

A covering algorithm is used to find simplified, but not necessarily minimal design equations. Instead of covering the 1-cells of a design equation the program covers the 1-*sets* originally generated from flow table columns. (Recall that a 1-set is a subcube containing one or more vertices or 1-cells for which the expression is 1.) The problem of generating and covering a large number of 1-cells is thus avoided. More importantly, it can easily be shown that by covering the 1-*sets*, all static hazards associated with vertical flow table transitions are eliminated from combinational circuit outputs. A static hazard exists when there is a transition between a pair of adjacent states having the same output, during which it is possible for a momentary improper output level to occur. Using two-level AND-OR synthesis, if each product (prime implicant) covers only 1-sets, all transitions within that 1-set are static-hazard-free; static hazards may only be caused by input-state changes which correspond to horizontal transitions on the flow table.

A procedure for eliminating the remaining "horizontal" hazards has been included. It is based on the restriction that only one input-state variable at a time changes. All pairwise combinations of a design equation's products (prime implicants) are examined for horizontally adjacent 1-sets. If such an adjacency is found, a static hazard exists. Since a horizontal transition may only originate at a stable state, the static hazard cannot possibly cause a malfunction unless one of the 1-sets includes a stable state.

Consider, for example, the illustration shown below, which is the simplified design equation for $Y_2$ of Figure 1, using Assignment 1:

$$Y_2 = y_3'v'w + y_2v + y_1w'$$

(where the input variables are v w)

Note that the horizontally adjacent 1-sets $y_3'v'w$ and $y_1w'$ appear as the first and third terms. If any stable state has a code in the subcube $y_1y_3'v'$ then a static hazard exists which may cause a malfunction. Note that stable state 3 in columns $I_1$ and $I_2$ both satisfy

$$Y_2 = y_3'v'w + y_2v + y_1w' + y_1y_3'v'.$$

*Program performance*

Execution times obtained using the program described here depend on hardware and software efficiencies, as well as the complexity of the input flow table. The solution times stated here were obtained using an IBM 360/50 computer and the IBM Release 13 PL/1 Compiler.

Simplified design equations for flow tables of 6 rows by 4 columns (24 cells) have been produced in 45 seconds to 4 minutes, depending on problem complexity. Eight row by 4 column tables usually are solved in 1.5 to 8 minutes. Three assignments for a 12 × 4 (48 cell) flow table have been produced in about 8 minutes, with next-state equations (unsimplified) generated in 3 minutes per assignment. Two satisfactory codes for a 18 × 4 (72 cell) flow table were found in 15 minutes. Computation times for large problems have been found to be extremely problem-dependent.

## SUMMARY

A description of a programmed algorithm for the synthesis of normal fundamental mode sequential circuits has been presented. The program permits the logic designer to input his asynchronous sequential circuit specifications in the form of a flow table and obtain all next-state equations and output equations in the form of simplified sum-of-products. Two internal state assignment algorithms are available to the designer. One will generate a minimum-variable assignment but may be lengthy to execute while the other will execute much faster but guarantees only a near-minimum variable solution. A testing routine is then available to aid the designer in deciding which of several satisfactory state assignments will tend to reduce the complexity of the design equations. A "good" assignment will be selected and simplified next-state and output equations will be generated based on the selected assignment. The complete program has been written in PL/1 and is running on an IBM 360/50 computer at the University of Missouri at Rolla. Flow tables with up to 75 specified next-state entries have already been run and much larger flow tables will soon be generated for experimentation purposes.

## REFERENCES

1 R J SMITH II
*A programmed synthesis procedure for asynchronous sequential circuits*

Masters Thesis University of Missouri at Rolla 1967

2 W L SCHOEFFEL
*Programmed state assignment algorithms for asnychronous sequential machines*
Masters Thesis University of Missouri at Rolla 1967

3 J H TRACEY
*Internal state assignments for asynchronous sequential machines*
IEEE Transactions on Electronic Computers Volume EC-15 pp 551-560 August 1966

4 D A HUFFMAN
*The synthesis of sequential switching circuits*
Journal of the Franklin Institute vol 257 pp 151-190 and 275-303 March and April 1954

5 M C PAULL S H UNGER
*Minimizing the number of states in incompletely specified sequential switching functions*
IRE Transactions on Electronic Computers vol EC-8 pp 356-367 September 1959

6 E J MC CLUSKEY
*Minimization of Boolean functions*
Bell System Technical Journal pp 1417-1443 November 1956

7 G K MAKI
*Minimization and generation of next-state expressions for asynchronous sequential circuits*
Masters Thesis University of Missouri at Rolla 1967