COMPUTER AIDED DESIGN OF A DIGITAL FREQUENCY SYNTHESIZER

By

BIENVENIDO C. PERALTA

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1953

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1974

COMPUTER AIDED DESIGN OF A DIGITAL FREQUENCY SYNTHESIZER

Thesis Approved:

Paul A. McCollum
Thesis Adviser

Edward L. Shreve

Richard L. Cummins

D N Dutam
Dean of the Graduate College

ACKNOWLEDGMENTS

The author wishes to thank the members of his committee, Professors Paul A. McCollum, Richard L. Cummins, and Bennett L. Basore for their guidance and patience.

A note of thanks also to Max Buckles, a colleague at Magnavox, for many helpful programming ideas.

To my wife, Lydia, special thanks for her unquestioning support and encouragement.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Computer-aided design, known as CAD in the literature, means many things to many people. The term currently covers all engineering design tasks and certain manufacturing support functions which are assisted by computers.

One of CAD's most successful applications to date is in automating much of the drafting and manual preparation of the "artwork" needed to make printed circuit boards. Programs have been developed which optimize the layout (placement of components) of a circuit board in regard to conductor length and other constraints.

To digital systems engineering, a more significant development occurred when device technologists learned to apply CAD programs similar to those used in circuit board design to their own integrated circuit (IC) artwork generation. Results have been astounding although this started only within the past five years. Improvements in device processing and packaging per se (e.g., passivated silicon junction, epitaxy) had reached a plateau by 1967 or so. However, CAD was able to further reduce the cost of manufacturing IC's while permitting increases in device complexity and performance. Consequently there has been a

growing trend to find new applications of digital circuitry in traditional electronic equipment as well as in automotive, medical, vending machines, and other fields.

CAD of digital components has therefore impacted on the logic design engineer in terms of what work there is to do; it is interesting to note that it is also affecting how to do it. On the matter of "how to do it," device technology today offers components which allow the designer to work at essentially two levels: at the gate/FF level, and at the register transfer or system level. In general, both design levels will be encountered since the former permits synthesis of functions not manufactured as standard IC modules, while the latter uses pre-designed building blocks or standard modules.

Although as noted above CAD has been used profitably in digital device design and production, CAD programs for logic and digital system design and development are not as yet generally available. This may come as a surprise to the reader who knows that analog filter design programs may be purchased, or rented (along with computer time) via remote terminals.

The present situation in regard to digital CAD programs (and particularly sets or systems of inter-related programs) is somewhat similar to the era when a new tool such as the oscilloscope, or perhaps the minicomputer, had just arrived on the scene. Initially they are expensive, and many potential users prefer to wait for a proven, mass produced

model. Others find it more cost-effective to build their own. The author in doing this thesis project has joined with those in the builder category.

Objectives and Results

The goal of this study was to implement a set of programs applicable for the computer-aided design of small digital systems or subsystems.

A survey of expected sources of CAD programs in the areas of logic and digital system design indicated that several synthesis and simulation programs were being used in industry. With the limitations of this study in mind these programs were categorized as available and not available. The former included low-cost library items such as graphical plotting routines, as well as logic simulators and filter design programs which may be accessed via remote terminals. In the not available class were two types of programs: proprietary (for owner company's internal use) and secondly, the few existing CAD software-hardware packages. From this project's viewpoint, the software-hardware systems were unavailable not only because of the five-figure dollar amounts required for their lease but also due to their being primarily data base and documentation systems.

Accordingly, the following objectives were considered:

(1) Develop a computer program for designing synchronous sequential circuits.

(2) To complement item (1) develop a program suitable for system level design.

(3) Search for other programs that are available, in the sense defined previously, for CAD adaptation.

(4) Devise a procedure for applying these programs in CAD of digital systems.

Addressing each of the cited objectives the results of this study are as follows:

(1) A synchronous sequential logic synthesis program was written and "debugged." It accepts as input a simple tabular representation of the state flow diagram of a specification. It outputs all equations needed to construct the circuit synthesized from the state diagram. The schematic may be drawn from the equations using JK flip-flops and IC gates as components.

(2) A register transfer simulation program described in the literature was modified and improved. Although simulation is not synthesis, this simulator can aid the designer by allowing convenient experimentation and evaluation of tentative system configurations. The system structure is described to the program in building block format, while system behavior is simulated in terms of sequences of register transfers and related operations.

(3) Computations for non-digital aspects of the design problem, e.g. spectral analysis and filter design, may be handled with the aid of the fast Fourier transform or FFT

(a library subroutine) and through commercial remote terminals, respectively.

A gate/FF level simulator program was rented after familiarizing with a time-shared version. This program was used for verifying sequential logic designs.

(4) A methodology for computer-assisted design employing the set of programs is reported in this thesis. A case study involving the detailed design of a digital frequency synthesizer is summarized therein.

Overview

The five chapters following the Introduction are organized as follows:

Chapter II describes the approaches and algorithms required to implement the programs. A methodology for using these programs for computer-assisted design is also outlined. Topics introduced in Chapter II which may seem tangential include design language, system level and gate level simulation, and non-digital design aspects. Gate level simulation is explained with an example. Programming topics regarding major subroutines of original programs are relegated to Chapter V.

Chapter III considers the design problem at the system level. A new type of frequency synthesizer is presented as a case study in digital system design, hence its functions and specifications are described. Next, the size of the memory word for the synthesizer is determined by analyzing

with the FFT the deterioration of the output waveform as word size is decreased. Feasibility of a proposed system configuration is then studied by simulation. To this end, conversion of block diagrams into register transfer notation is illustrated.

Chapter IV describes the synthesis of synchronous (clocked) sequential circuits. The first example consists of a control function. Subsequent examples deal with shift registers and counters. The remaining examples are concerned with minimization of output gating, and effects of state assignment. All were used in designing the frequency synthesizer.

Programming details, flow charts, listings of the logic synthesis programs, and samples of input coding and resulting printouts (and these comprise "documentation" as used by programmers) are contained in Chapter V and the appendices. For the convenience of the reader, a program which generates asynchronous logic and which served as the prototype for the program employed in Chapter IV is included in Appendix D.

CHAPTER II

APPROACH

System Level CAD

Given today's pre-packaged gate arrays, flip-flops, registers, adders, and other building blocks the task of design starts naturally with the consideration of system level structure and behavior. One may assume that a system can be constructed by (1) selecting a set of building blocks and interconnecting them, and (2) designing non-standard functional blocks, if any are needed, after the system structure has been developed. For this purpose it is convenient to employ a simulator and its programming language.

Design Language

To simulate a digital system one needs to provide data to a simulator program which describes the system's organization. Additionally the data must convey the details of the system's operations, timing, and control. For ease of preparation and readability, a digital system should be described via a design language.

Duley, et al. [1] and Baray, et al. [2] have proposed languages in which programs containing design specifications may be written, and which serve as inputs for simulation and

synthesis. Chu [3] incorporated the register transfer concept in an Algol-like language that has seen actual use in computer design.

The language adapted in this study is a subset of Chu's Computer Design Language (CDL). As will be shown in Chapter III it can define any register, decoder, memory, and other building blocks. It is easier to learn than Fortran since it is a higher order language. Compared to an equivalent Fortran program, a CDL program would have considerably fewer statements.

Register Transfer Simulation

A CDL description, in conjunction with test data, permits the simulator program to compute a system's behavior or response. The response is characterized primarily as a sequence of values of contents of registers belonging to the system being simulated.

The simulator program used in this study contains two sections: the translator section and simulator proper. The former translates the CDL model of a system (in punched card form) into an internal compiler code and sets up various tables. The latter consists of four routines: Loader, Output, Switch, and Simulate. The Loader accepts the initialization and test data segments of the input card deck and stores them in simulated registers and memories. Results of the simulation are formatted for printing by the Output routine. Printout typically consists of the contents of

certain registers and memory words evaluated at each clock time. Items to be printed are selected by the user. The Switch routine simulates manual switches. The Simulate routine executes the internal compiler code interpretively (i.e. the simulation is performed as though the input program were in machine language).

Gate Level CAD: Synchronous Sequential Circuits

Manual versus CAD Procedures

A synthesis procedure for synchronous sequential circuits, modelled as Moore machines, consists of five steps:

(1) Make a flow table from the design specification.

(2) Reduce the number of rows of the flow table.

(3) Assign a binary code to each state.

(4) Determine the flip-flop input equations.

(5) Design the combinational logic for the output.

We are justified in treating synchronous logic exclusively since its preponderance over asynchronous sequential logic is well known. To a large degree this is due to the fact that the critical race problem does not exist in synchronous sequential circuits and so they are easier to design. When the Moore machine viewpoint is taken, designing the output gating is more straightforward compared with the Mealey model since the output function depends only on the internal state. Further, Friedmann and Menon [4] have

recently shown that this approach lends itself to more systematic production of test patterns (useful in manufacturing and maintenance).

In practice, state assignment is usually done by trial and error. Steps (2) and (4) also contain many tedious operations when more than a few input signals and internal states are required. When done manually step (5) may prove difficult if a large multi-output minimal cost network is desired. Hence the design procedure can benefit from CAD programs.

The CAD programs that were developed assist the designer in performing steps (4) and (5). The name Synchronous Logic Synthesis Program is given to the set since logic equations are generated from which a schematic diagram may be drawn. For reasons to be explained later, methods intended for flow table reduction and state assignment were not programmed.

The suggested CAD procedure follows the manual procedure with two modifications:

(a) The flow table prepared for step (3) is converted into a state diagram.

(b) Only JK flip-flops will be used.

The state diagram of item (a) serves as the input to the synthesis program. Punched cards are easily prepared which convey the node and transition signal data in the form of a from - to table or "wire list." Regarding (b), the JK flip-flop is widely used so that specializing the present

version to a single flip-flop type in order to simplify the program seems justified.

CAD Synthesis Algorithms

The following algorithm is commonly used for determining the flip-flop input equations of sequential circuits using JKFF's. (Notations of the form JKFF and FFA denote JK flip-flop and flip-flop A respectively.)

(1) Make a state transition table for each JKFF to be used.

(2) Draw Karnaugh maps for each state transition table.

(3) Derive the minimized JKFF input equations from the maps.

The Synchronous Logic Synthesis Program mechanizes the above procedure. Corresponding to the first step, a "list" data structure is constructed and stored in memory when the data cards are read. Instead of the Karnaugh map, a subroutine processes the list using a version of the Quine-McCluskey minimization algorithm. The minimal expressions (J_A and K_A for FFA, etc.) are then printed out.

To illustrate the equation generation process, a control logic function will be synthesized by going through steps (1) to (3) manually. The results are then compared with the CAD program's output.

The control logic function (Synchronization Indicator) is specified by the timing diagram shown in Figure 1. The diagram defines the behavior of the function's output for the

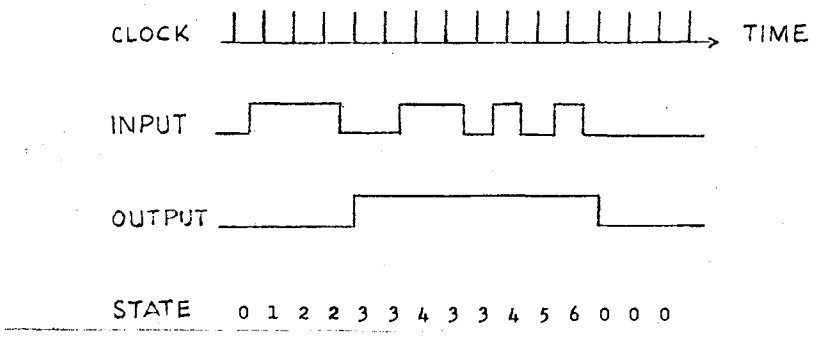
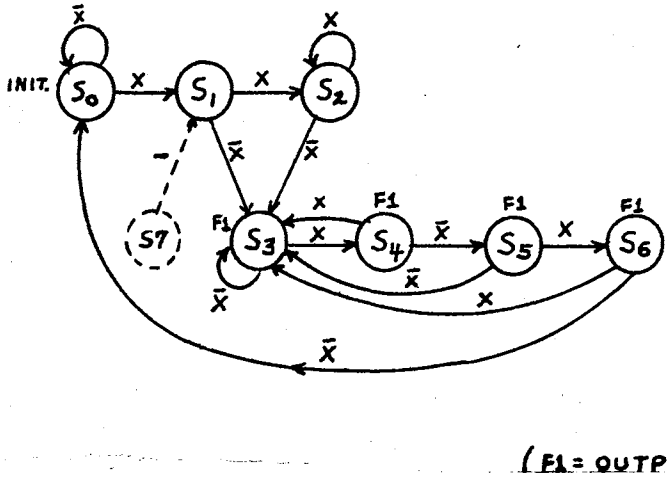


Figure 1. Timing Diagram for Synchronization Indicator Logic. Input = X Output = F1



(F1 = OUTPUT)

Figure 2. State Diagram of Synchronization Indicator

input signal given. Either a state diagram is drawn from a flow table derived from Figure 1, or made directly without benefit of formal state minimization techniques.

Figure 2 shows a state diagram for the Synchronization Indicator having seven states and an optional state, S7. The label F1 on nodes S3 through S6 indicates that the output is true when the present state is in any of these nodes.

The state transition table for FFA, Table I, was constructed from the state diagram after making the state assignment:

S0	S1	S2	S3	S4	S5	S6	S7
000	110	101	100	011	010	001	111

In Table I, the entries NC(0) and NC(1) in the Action Desired column mean "no change, state 0" and "no change, state 1" respectively. The table gives the values required for inputs J,K to cause the actions set, reset, NC(0), and NC(1) to occur during the next clock period. For example in the first row of the table, the next state of FFA is given as 1 when X = 1. Hence FFA must be set and this requires J = 1 and K = d, where d denotes don't care.

In Figure 3 the Karnaugh maps for J_A and K_A are depicted. These maps were constructed by treating Table I as a table of combinations for the present state and input X. For example, cell 0100 of the maps contain the entries 1, d respectively. These were taken from row 3 of Table I with X = 0 columns indicating set A is the desired action.

TABLE I

STATE TRANSITION TABLE FOR FFA OF SYNCHRONIZATION INDICATOR

Present State			Next State (FFA)		Action Desired*	
FFA	FFB	FFC	X = 0	X = 1	X = 0	X = 1
	000		0	1	NC(0)	Set
	001		0	1	NC(0)	Set
	010		1	0	Set	NC(0)
	011		0	1	NC(0)	Set
	100		1	0	NC(1)	Reset
	101		1	1	NC(1)	NC(1)
	110		1	1	NC(1)	NC(1)
	111		1	1	NC(1)	NC(1)

* Set ⇒ J = 1, K = d NC(0) ⇒ J = 0, K = d

Reset ⇒ J = d, K = 1 NC(1) ⇒ J = d, K = 0

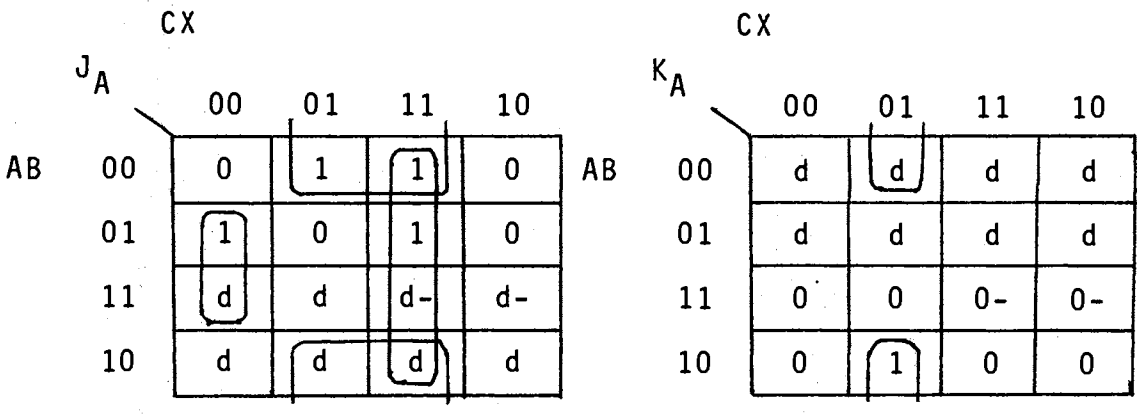


Figure 3. Karnaugh Maps for FFA of Synchronization Indicator

The minimal expressions derived from the loops of the J_A, K_A maps are:

$$J_A = B\bar{C}\bar{X} + \bar{B}X + CX$$

$$K_A = \bar{B}\bar{C}X$$

The CAD program when given the same state diagram and state assignment generates exactly the same equations for J_A and K_A. These equations as well as those for the two other FF's required are shown in the computer printouts of Appendix A. Note that the minterms comprising a loop are also displayed.

The equations for FFB and FFC are:

$$J_B = \bar{C}X$$

$$K_B = \bar{A}X + \bar{C}$$

and

$$J_C = AX + BX$$

$$K_C = \bar{X} + \bar{A} + B$$

The foregoing equations are depicted in logic schematic form in Figure 4.

The output gating in Figure 4 was obtained by making a truth table of the output function (F1) from the timing diagram and minimizing it with another CAD program. This completed the design of the Synchronizer Indicator.

Referring again to Figure 3, cells 1111 and 1110 of both maps contain dashes, d's, and 0's. The dashes are <u>don't care</u> entries which apply when the unspecified state S7 is ignored.

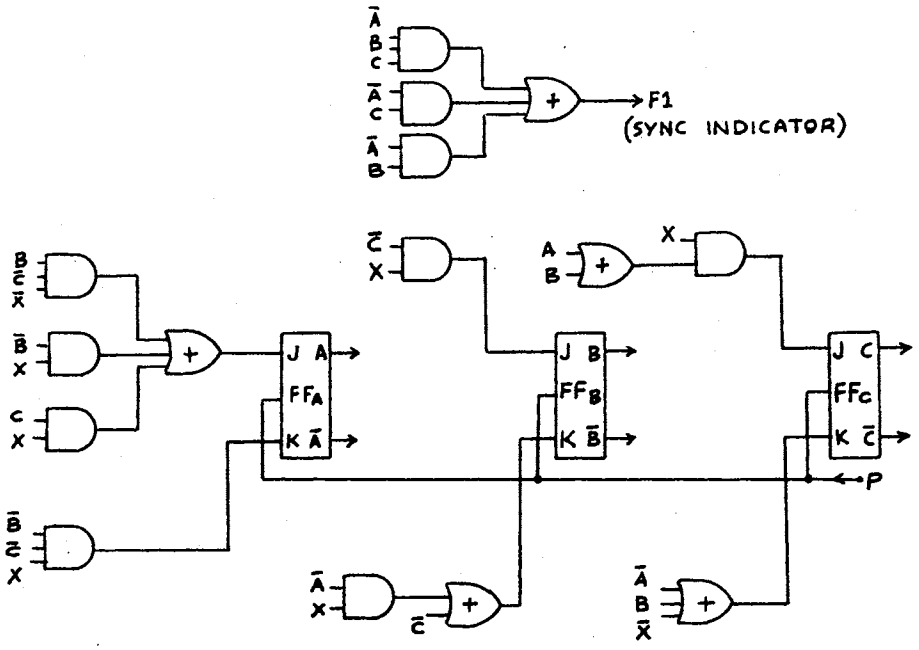


Figure 4. Synchronization Indicator Logic Drawn from Equations Generated by CAD Programs

The d's and 0's apply when S7 is assigned binary code 111. Although the expressions generated are the same in either case, they may be different from each other for other state assignments. State assignment and output gating will be discussed further in later sections.

Flow Table Reduction and State Assignment

Taking into account that the present set of CAD programs do not perform flow table reduction and state assignment, the following guidelines are suggested.

Flow Table Reduction. Manually process a primitive flow table using a method intended for the type of table. Procedures for reducing the number of rows have been detailed by Givone [5] for completely specified, incompletely specified, and input restricted types.

State Assignment. An algorithm is described in a recent paper by Story, Harrison, and Reinhard [6]. This and other methods known to the designer may be used to produce a number of state assignments for each problem. Since it is a simple matter to input these data to the CAD program, all these trials may be submitted in each run. The best solution (fewest number of input terms and/or number of gates) is then selected from the printouts.

Many papers have appeared in the literature proposing schemes for flow table reduction and optimal state assignment. These methods seem to have one or more problems such as incompatibility with don't care conditions, tedious to

apply, not appropriate for synchronous logic, and use of a heuristic approach rather than algorithmic.

From the viewpoint of CAD program implementation, it is important that a procedure be definable as an algorithm. Experience has shown that a heuristic method sometimes defies conversion into a program having reasonable memory and running time requirements.

Verification by Simulation

As an approach to the problem of verification of a logic design, simulation at the gate/FF level is gaining acceptance. Like most CAD tools it is appreciated most when the design to be verified exceeds a few dozen gates and FF's. This is due to the fact that program setup time for the simulation becomes significantly less than the cost of a comparable breadboarding effort. Verifying designs with a good simulator (one which can include effects of gate delays and detect violations of loading rules) permits prototyping with confidence. Hence the need for prototype "kluges" is minimized.

The simulator program employed in this study goes by the trade name of LOGSIM. Literature on its capabilities and an application manual are obtainable from Tymeshare, Inc. [7].

The maximum allowable number of gate-equivalents per simulation is 300, each JKFF being equivalent to 7-10 gates (depending on the particular commercial type). For example,

for a network containing 20 JKFF's, 140-200 gates would be allocated for modelling the JKFF's leaving 100-160 gates to be used as gates or inverters. Other simulators have comparable characteristics and may be rated as to availability of gate delay and other modelling features, editing and file saving facilities, and run time and memory requirements.

Table II indicates the coding format and language used for LOGSIM. The circuit modelled by the statements is an 18-gate version of the Synchronization Indicator which resulted from one of several state assignments tried.

The first three lines of Table II describe the flip-flops used in the circuit. For example, the significance of the symbols in line 1,

1 JK (2, QA, NA, JA, KA, P, 0, 0)

taken from left to right is: This statement is from card number 1 of the network's LOGSIM model. A JKFF is specified. There are two outputs, QA and NA. The synchronous inputs are JA, KA, and P. The initial condition for the asynchronous inputs (direct set and direct reset) are 0, 0, respectively.

Lines 4 through 21 specify AND, OR, and inverter gates and the way they are connected. Line 5 is read "KA is the output of an OR gate whose inputs are A5 and A2." AND gate and inverter declarations are read similarly.

The clock pulse source, P, is declared in line 23. This clock is "connected" to any component where the name P is declared as an input. That is, the node associated with

TABLE II

LOGIC SIMULATOR INPUT CODING FOR SYNCHRONIZATION INDICATOR

```
  1 JK(2,QA,NA, JA,KA,P,0,0)
  2 JK(2,QB,NB, JB,KB,P,0,0)
  3 JK(2,QC,NC, JC,KC,P,0,0)
  4 JA=OR(AN1,A2,A3,A4)
  5 KA= OR(A5,A2)
  6 AN1 =AND(QB,NC,NX)
  7 A2=AND(NB,X)
  8 A3=AND(NB,QC)
  9 A4=AND(QC,X)
 10 A5=AND(NC,X)
 11 JB=OR(A6,QA)
 12 KB=OR(A7,A8)
 13 A6=AND(QC,X)
 14 A7=AND(QA,NC,X)
 15 A8=AND(NA,QC,NX)
 16 JC=AND(QA,QB,X)
 17 KC=OR(A10,A11,A12)
 18 A10=AND(QA,NB,NX)
 19 A11=AND(NA,X)
 20 A12=AND (NA,QB)
 21 NX=INVERT(X)
 23 P=A(PULSE)
 25 F1=OR(A13,A14,A15)
 26 A13=AND(NA,QC)
 27 A14=AND(QA,QB,NC)
 28 A15=AND(NB,QC)
 29 JK(2,W, NW, JW,KW, P,0,0)
 30 JK(2,A1,NA1,W, NW, P,0,0)
 31 JK(2,QZ,NZ, Y, NY, P,0,0)
 32 JK(2, Y,NY, A1,NA1,P,0,0)
 33 JK(2, X,NQX,QZ,NZ, P,0,0)
 34 KW=INVERT(JW)
 35 JW=OR(A20,A21,A22,A23,A24,A25)
 36 A20=AND(NW,NA1,Y,QZ,X)
 37 A21=AND(NW,A1,NY,QZ,X)
 38 A22=AND(NW,NA1,NY,NZ,X)
 39 A23=AND(NW,A1,Y,NZ,NQX)
 40 A24=AND(W,NA1,NY,QZ,X)
 41 A25=AND(W,NQX,NY,NZ)
  OUTPUTS AST,      AST,P,AST,  AST,QC,QB,QA,AST,JA,KA,AST
  OUTPUTS JB,KB,AST,JC,KC,AST,F1,AST
  OUTPUTS AST,W,A1,Y,QZ,X,AST,AST,A20,A21,A22,A23,A24,A25
  END

--SIMULATION  COMMAND  DATA-- PAGE 1


T1 CLOCKED 'SYNCHRONIZER' USING J-K FFS  (CAD SOL'N)
 PULSE 150,0
  END
```

a literal is connected to any component declaring it as an input signal, assuming of course that the format rules exemplified by the JKFF example are observed.

Lines 29-41 model a shift register which simulates the X-input sequence needed to test the design. It is not part of the logic function being verified.

"Housekeeping" statements which specify outputs to be printed out, the title heading, and number of clock pulses to be generated comprise the rest of the simulation input program.

When the program of Table II is executed, the printout prepared by the LOGSIM simulator is shown in Table III. The left-most heading, TEST, denotes row or line number. The heading P identifies the column used for the clock pulses. Similarly QC, QB, QA, ..., A25 are for FF's and gates specified by the user. The output values are printed alternately for P = 0 and P = 1 in Table III. The JKFF's are shown to change state after a 1-0 transition.

In the present example, Table III shows that the output F1 followed the timing diagram of Figure 1. Note also that the shift register (see columns W through X) produced the input signal sequence specified in the timing diagram. Thus the simulated circuit (or model) driven by a test pattern generated a printout from which the designer can infer that the logic specification was met.

TABLE III

PRINTOUT FROM LOGIC SIMULATOR PROGRAM

CLOCKED 'SYNCHRONIZER' USING J-K FFS

TEST	INPUT ID	P QQQ JK JK JK F WAYQX AAAAAA CBA AA BB CC 1 1 Z 222222 012345
1		**0**000*11*00*01*0**11111**000000
2		**1**000*11*00*01*0**11111**000000
3		**0**001*11*11*00*0**01111**000000
4		**1**001*11*11*00*0**01111**000000
5		**0**010*01*00*01*0**00111**100000
6		**1**010*01*00*01*0**00111**100000
7		**0**010*01*00*01*0**10011**000010
8		**1**010*01*00*01*0**10011**000010
9		**0**010*01*00*01*0**11001**000000
10		**1**010*01*00*01*0**11001**000000
11		**0**010*10*00*01*0**01100**000100
12		**1**010*10*00*01*0**01100**000100
13		**0**011*10*10*00*1**10110**000000
14		**1**011*10*10*00*1**10110**000000
15		**0**011*01*11*10*1**01011**010000
16		**1**011*01*11*10*1**01011**010000
17		**0**100*11*10*01*1**10101**000000
18		**1**100*11*10*01*1**10101**000000
19		**0**011*10*10*00*1**01010**000000
20		**1**011*10*10*00*1**01010**000000
21		**0**011*01*11*10*1**00101**000000
22		**1**011*01*11*10*1**00101**000000
23		**0**100*10*01*00*1**00010**000000
24		**1**100*10*01*00*1**00010**000000
25		**0**101*11*10*00*1**00001**001000
26		**1**101*11*10*00*1**00001**001000
27		**0**110*00*01*01*1**10000**000001
28		**1**110*00*01*01*1**10000**000001
29		**0**000*00*00*00*0**11000**000001
30		**1**000*00*00*00*0**11000**000001
31		**0**000*00*00*00*0**11100**000000
32		**1**000*00*00*00*0**11100**000000
33		**0**000*00*00*00*0**01110**000000
34		**1**000*00*00*00*0**01110**000000
35		**0**000*11*00*01*0**00111**100000
36		**1**000*11*00*01*0**00111**100000
37		**0**001*11*11*00*0**10011**000010
38		**1**001*11*11*00*0**10011**000010
39		**0**010*01*00*01*0**11001**000000
40		**1**010*01*00*01*0**11001**000000
41		**0**010*10*00*01*0**01100**000100
42		**1**010*10*00*01*0**01100**000100
43		**0**011*10*10*00*1**10110**000000
44		**1**011*10*10*00*1**10110**000000
45		**0**011*01*11*10*1**01011**010000

Gate Level CAD: Minimization of Output Gating

To obtain the minimized expression to be used for an output gating circuit, punched cards representing its truth table are input to LOGICMIN. This program is external to the Synchronous Logic Synthesis Program. These two programs were separated since some modification of the minimization algorithm and different printout formats were desired in each case. LOGICMIN may be used for any combinational switching function provided its truth table has no more than 1024 rows and 32 literals.

Single-Output Functions

The algorithm used here is the well known Quine-McCluskey method which initially determines the prime implicants of a given function. The method then finds a set of irredundant expressions from which minimal expressions are formed. A hazard-free minimal sum is also computed for possible use with the asynchronous (direct set/reset) inputs of JKFF's.

Multiple-Output Functions

For this type of gating an extension of the Quine-McCluskey method described by Givone [8] was used in LOGICMIN. The extension amounts to employing the original algorithm to process tagged product terms and using a mask (AND type) operation on the tags.

LOGICMIN computes several candidate solutions for each truth table. One set of solutions tends to optimize cost with respect to the number of input lines and gates. A second set has a different criterion namely that of using smaller (2-input or 3-input) gates rather than larger gates.

To aid the user in drawing the logic schematic from the minimized gating expressions, cross-reference tables, print-plots, and binary code labels are included in the printout of results.

Non-Digital Aspects

Spectral Analysis with FFT

The FFT is an efficient method of computing the discrete Fourier transform. Basore [9] has prepared a monograph explaining this computational short-cut.

For CAD programming if the FFT is included in a computer library of subroutines, then it is simply called by the user's program. The version used in this study was written in Fortran and invoked by a statement of the form

```
CALL COOL (N, ARRAY, -1)
```

where N specifies the number of samples, ARRAY is the name of an array dimensioned as two rows and N columns, and -1 signifies the direct transform (+1 would specify inverse). ARRAY stores the real and imaginary components computed by the subroutine.

The user's program typically provides for the reading in of the sampled data, subsequent conversion of the spectral components into db, and tabulation of results.

Filter Design

An analog filter synthesis program available through remote terminals was used in designing the low-pass filter specified in Chapter III. The program, designated MATCH by the Applicon Company [10], employs the conjugate gradient approach for optimization.

Methodology

The CAD procedure suggested for systems consists of seven steps one or more of which may be optional. The choice of which step to by-pass and how many iterations to perform depends on the user's judgement.

(1) Convert the block diagram and timing chart of a system into a register transfer description for input to the simulator program. Reconfigure and simulate again as necessary to refine and simplify the design.

(2) Determine the standard and non-standard building blocks of the system developed in the first step.

For each non-standard block do steps (3) to (7).

(3) Input the state diagram data of a non-standard function to the Synchronous Logic Synthesis Program.

(4) Input to LOGICMIN the truth table of the output gating function.

(5) Draw the schematic for the equations generated in step (3). Label the JKFF terminals to correspond with the state assignment used. Select one of the equations obtained in step (4) and draw its schematic.

(6) Interconnect the output gating from step (5) to the JKFF's.

(7) Verify the complete schematic of the non-standard function by gate level simulation.

Following completion of the above, prototype hardware may be assembled from off-the-shelf items and from the non-standard functions designed with the procedure.

It is assumed that previous to step (1), non-digital aspects involving filters and related interface circuitry had been dealt with. As noted, CAD programs are commercially available for this portion of the design task.

CHAPTER III

SYSTEM LEVEL DESIGN

A new approach which competes favorably with analog techniques in the area of very stable frequency generation is described and used as a design example. The method computes a sequence of sinusoid samples with a simple table look-up scheme followed by interpolation by means of a low pass filter. Table look-up is practical since the number of samples is small, and low-cost read only memory (ROM) used to store the samples is now available. The technique is simpler than digital recursion and produces less noise [11].

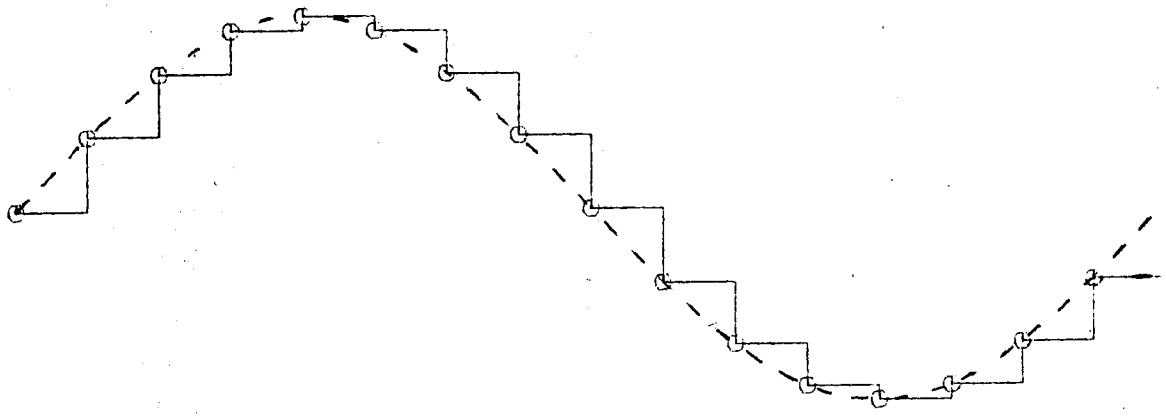
The process generates a time series of sine wave samples represented by the expression

$$\{\sin 2\pi fnT\} \qquad n = 0, 1, 2, \ldots$$

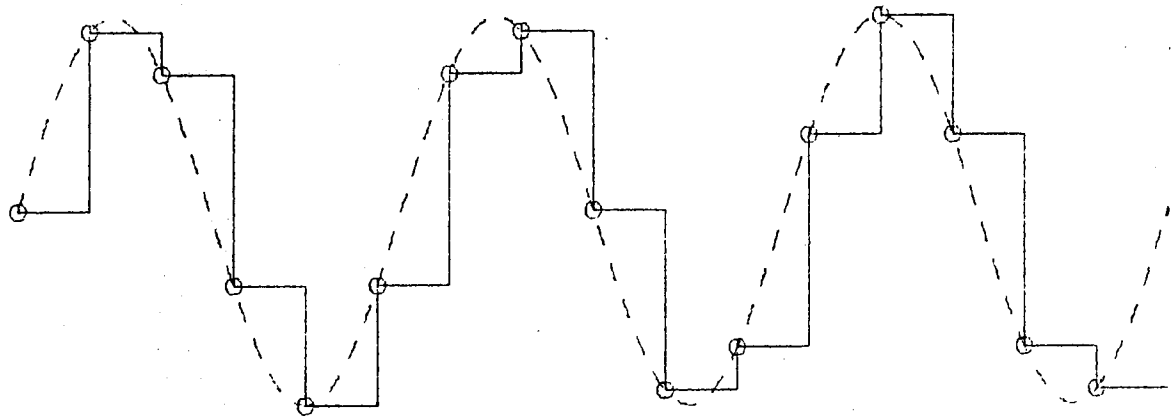
where f = frequency to be generated, n = time index, and T = sampling interval.

The lowest frequency, f_{lo} is synthesized when the total number of samples stored in the table, N, are used in each period. This is depicted in Figure 5a with N = 16. For a given T,

$$f_{lo} = 1/NT.$$



a.) Time Series for f_{lo}



b.) Time Series for $3f_{lo}$

Figure 5. Plot of Time Series of Samples Obtained from Read Only Memory

To select a frequency we provide a frequency index k such that

$$f = kf_{lo}$$

and the time series expression may be written

$$\{\sin(2\pi nk/N)\} \qquad k \leq N/4;\ n = 0,1,2,....$$

The above indicates that the generated frequency can be set by index k. Due to the Nyquist condition, and for ease of filtering, the highest frequency is constrained to $f = (N/4)f_{lo}$.

Figure 5b shows the sequence of samples corresponding to k = 3. It also shows that after the third cycle the sequence repeats. This implies that as n increases, the product nk is treated modulo N. The generation of time series therefore involves accumulating multiples of k. In terms of the table look-up scheme, accumulated values of k are used as memory addresses and no other computations are needed.

Digital Frequency Synthesizer Specification

Being a case study in system level design pertinent characteristics of the synthesizer are specified as analog and digital. The analog specifications are: (1) number of frequencies = 32; (2) lowest frequency = 1 Hz; (3) maximum in-band noise referred to a generated frequency = - 40 db; (4) low pass filter transition band is from 32 to 64 Hz with out-of-band attenuation of 80 db. (See Figure 6). The

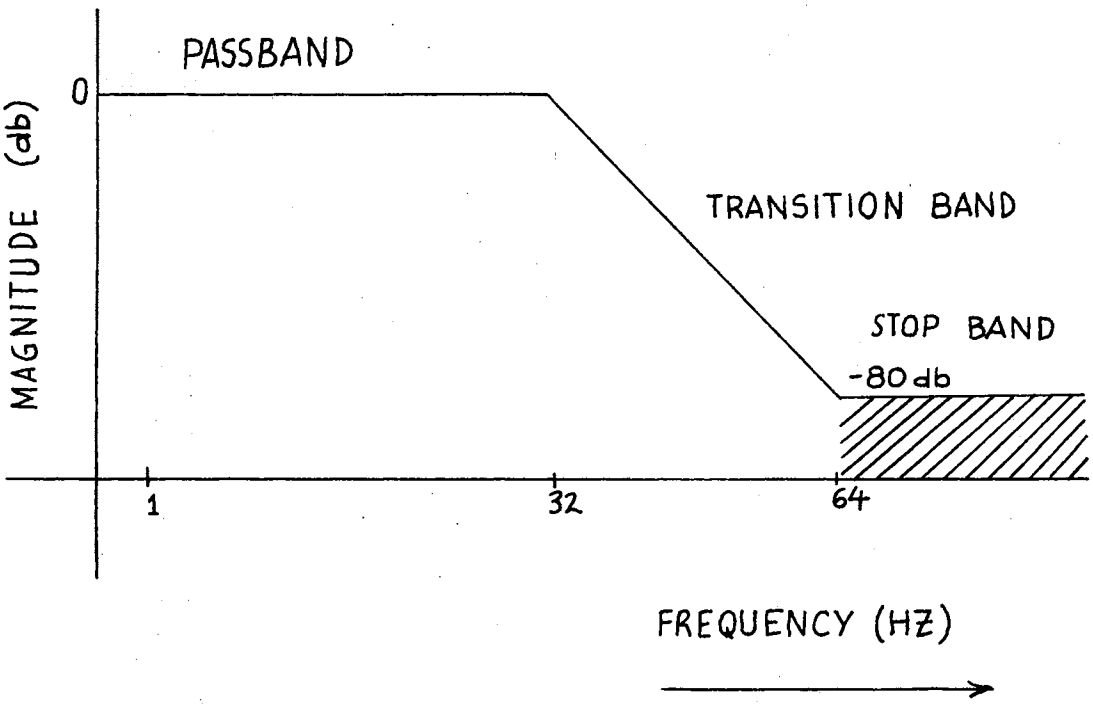


Figure 6. Characteristics of Low Pass Filter

digital portion of the design specification follows. Binary arithmetic is to be used in implementing the block diagram of Figure 7. There are two modes of frequency selection: fixed and sweep. Provide a synchronizer indicator function. The timing charts of Figure 8 and Figure 9 are part of the specification.

In Figure 7 the accumulation process for index k described earlier is done by an accumulator which consists of an address register that feeds back to an adder. The input register holds the value of the control word corresponding to index k for generating a constant frequency. Similarly, the frequency sweep up-counter serves as an

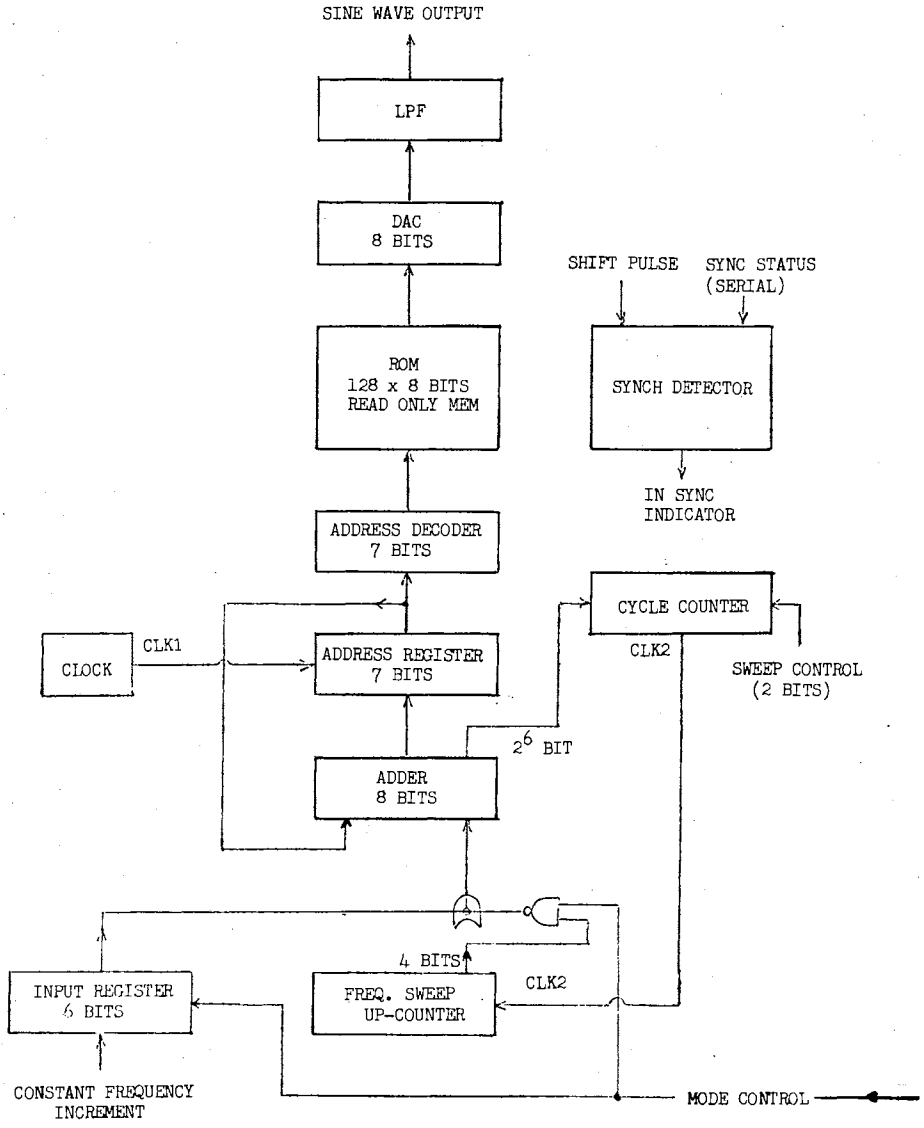


Figure 7. Block Diagram of Frequency Synthesizer

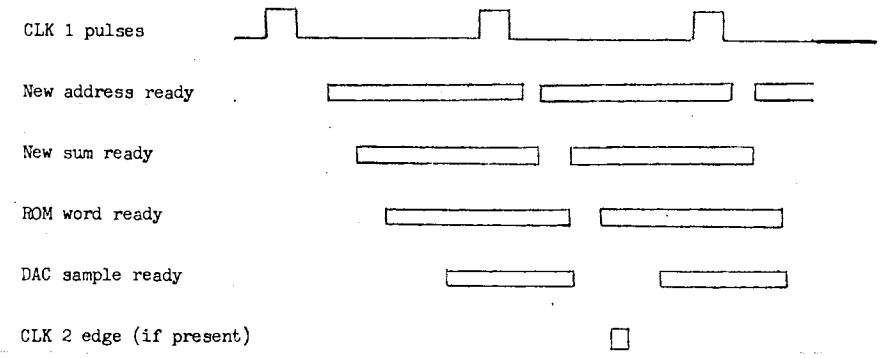


Figure 8. Frequency Synthesizer Timing Chart

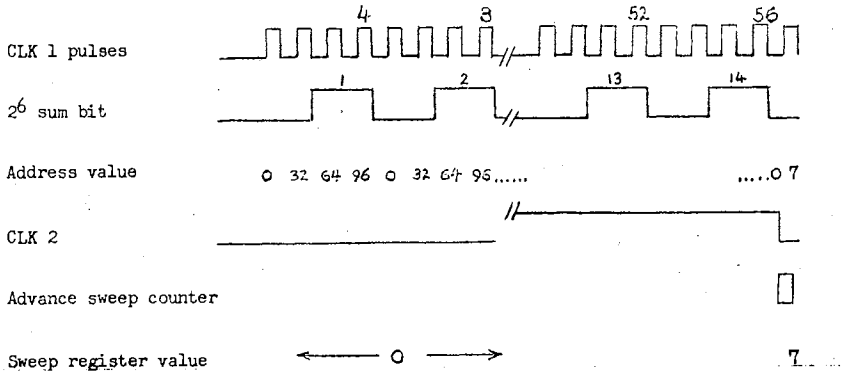


Figure 9. Timing Diagram for Sweep Mode, Initial f = $32f_{10}$ and Sweep Control set for 14 Cycles

input register except it increments the frequency during the sweep mode.

The table look-up operation is performed by the ROM and accumulator. The digital to analog converter DAC changes the binary coded samples into analog voltages which are then interpolated by the filter LPF. The sampling interval is determined by the clock CLK1 since the address decoder gets the next address after each CLK1 pulse. The box labelled "synch detector" has a logic function which determines if an external pulse train is in synchronism with CLK2.

The cycle counter in Figure 7 controls how long a frequency stays fixed during the sweep mode. It contains a ÷ 7 counter in series with a variable-modulo counter (÷ 2, 4, 6, 8 depending on sweep control word). Hence CLK2 pulses are derived from transitions of the 2^6 bit of the adder divided by 14, 28, 42, or 56. In turn, the frequency sweep up-counter supplies k index values 7, 8, 9, 10, 11, 7, ... Since CLK2 drives this counter, the output frequency will be $7f_{lo}$, $8f_{lo}$, ... and will remain fixed for 14, 28, 42, or 56 cycles before changing (sweep mode only).

Figure 8 gives the timing relationship between the operations which occur in the Frequency Synthesizer. After each CLK1 pulse enables the address register a new address is loaded into the address decoder. The next sum is ready after a slight delay needed for adding the k index and the

present sum. The time allowed for reading out the ROM and for analog conversion must be short enough so that the analog sample is ready before the next CLK1 pulse.

Some details of fixed and sweep mode operation are depicted in Figure 9. Initially the fixed mode applies, and the selected frequency is assumed to be $32f_{lo}$. The diagram shows that the 2^6 sum bit indicates overflow after every four samples. The address values are shown to be 0, 32, 64, 96, 0, ... (modulo 128). Because the sweep control is set to permit 14 cycles to be generated, CLK2 is shown to fall after 56 CLK1 pulses. The figure also assumes that the sweep mode was selected at this time, and that the sweep register (or sweep up-counter) was enabled to the adder when it contained the value 7.

The ROM used for storing the sine wave samples is described in Figure 7 as having 128 × 8 bit words. This means that the number of words, N, is double that required by the Nyquist criterion for a maximum frequency of 32 Hz. The 8-bit word size was obtained by taking the worst case error as being equal to the least significant bit. This gives for an 8-bit word (7 bits magnitude plus sign) $20 \log 2^{-7}$ or -42 db. This is less than the -40 db specification for in-band noise. The 8-bit word length will be validated later by means of the FFT.

Only 7 bits of the adder in Figure 7 are connected to the address register. That is, the sum bits 2^0 to 2^6 are used thereby converting the accumulated k index values modulo 128.

Effects of Truncation and Sampling Errors on Memory Word Length

Determining the correct word length is important since too small a word could result in failing to meet the noise specification due to truncation (round off) effects. Too large a word causes extra power dissipated due to unnecessarily large registers and adders. Further, ROM modules are relatively expensive due to the additional process of "programming" the values to be stored into the module. Hence a change of word size means that a completely new ROM must be programmed.

The effect of word size on spectral purity was studied empirically by using the FFT to compute spectra of samples having different word lengths. For simplicity it was assumed that the synthesizer had a total of 16 samples and that word lengths can be adjusted to 4, 7, and 31 bits.

The results from a 32-point transform are shown in Table IV for 7-bit and 31-bit samples. The 31-bit case was called non-truncated since this is the full word size for single-precision arithmetic. Only the positive 16 samples are shown. The table is for the case when the lowest frequency, f_{lo} was selected.

Table V presents the data of Table IV in db referred to f_{lo}. Additional data for the 4-bit case were included. Similarly, Tables VI and VII give the results for $5f_{lo}$ and $6f_{lo}$ respectively.

TABLE IV

PRINTOUT FROM FFT PROGRAM AFTER PROCESSING THIRTY TWO SAMPLES OF NON-TRUNCATED DATA (TOP) AND SEVEN BIT DATA (BOTTOM)

LINE NO.	MAGNITUDE	SAMPLED DATA
0	0.4192E-06	0.0
1	0.4976E 00	0.0
2	0.7764E-06	0.3826830
3	0.4079E-06	0.3826830
4	0.3128E-06	0.7071062
5	0.2016E-06	0.7071062
6	0.2020E-06	0.9238790
7	0.4142E-06	0.9238790
8	0.1489E-06	1.0000000
9	0.2300E-06	1.0000000
10	0.1350E-06	0.9238803
11	0.1661E-06	0.9238803
12	0.1296E-06	0.7071087
13	0.9045E-07	0.7071087
14	0.1544E-06	0.3826867
15	0.4901E-01	0.3826867
0	0.0	0.0
1	0.4968E 00	0.0
2	0.0	0.3828125
3	0.1795E-02	0.3828125
4	0.0	0.7109375
5	0.1263E-02	0.7109375
6	0.0	0.9218750
7	0.1169E-03	0.9218750
8	0.0	0.9921875
9	0.9601E-04	0.9921875
10	0.0	0.9218750
11	0.6750E-03	0.9218750
12	0.0	0.7109375
13	0.5444E-03	0.7109375
14	0.0	0.3828125
15	0.4894E-01	0.3828125

TABLE V

SPECTRAL ANALYSIS USING FFT FOR TRUNCATED AND NON-TRUNCATED SAMPLES
CASE I SYNTHESIZER SET TO LOWEST FREQUENCY

Harmonic*	No Truncation		7-bit Samples		4-bit Samples	
	Magnitude	H_1/H_n db	Magnitude	H_1/H_n db	Magnitude	H_1/H_n db
H_1	0.4976	0.0	0.4968	0.0	0.475	0.0
H_3	0.4079E-6	121.7	0.1795E-2	48.8	0.696E-2	36.7
H_5	0.2016E-6	127.8	0.1263E-2	51.9	0.130E-2	51.3
H_7	0.4142E-6	121.6	0.1169E-3	72.6	0.694E-3	56.7
H_9	0.2300E-6	126.7	0.9601E-4	74.3	0.569E-3	58.4
H_{11}	0.1661E-6	129.5	0.6750E-3	57.3	0.693E-3	56.7
H_{13}	0.9045E-7	134.8	0.5444E-3	59.2	0.211E-2	47.0
H_{15}	0.4901E-1	20.1	0.4894E-1	20.1	0.467E-1	20.1

* $H_1 = f_{lo}$. The zero and even order harmonics vanish for truncated data and less than 10^{-5} for non-truncated data.

TABLE VI

SPECTRAL ANALYSIS USING FFT FOR TRUNCATED AND NON-TRUNCATED SAMPLES
CASE II SYNTHESIZER SET TO FIVE TIMES LOWEST FREQUENCY

Harmonic*	No Truncation		7-bit Samples		4-bit Samples	
	Magnitude	H_5/H_n db	Magnitude	H_5/H_n db	Magnitude	H_5/H_n db
H_1	0.240E-6	125.3	0.187E-2	47.4	0.724E-2	35.3
H_3	0.532E-6	118.4	0.145E-3	69.6	0.859E-3	53.8
H_5	0.441	0.0	0.440	0.0	0.421	0.0
H_7	0.272E-6	124.2	0.111E-2	52.0	0.114E-2	51.3
H_9	0.182E-6	127.7	0.909E-3	53.7	0.933E-3	53.1
H_{11}	0.236	5.4	0.235	5.4	0.225	5.4
H_{13}	0.183E-7	147.6	0.439E-4	80.0	0.260E-3	64.2
H_{15}	0.417E-7	140.5	0.184E-3	67.6	0.713E-3	55.4

* $H = f_{lo}$. The zero and even order harmonics vanish for truncated data and less than 10^{-5} for non-truncated data.

TABLE VII

SPECTRAL ANALYSIS USING FFT FOR TRUNCATED AND NON-TRUNCATED SAMPLES
CASE III SYNTHESIZER SET TO SIX TIMES LOWEST FREQUENCY

Harmonic*	No Truncation		7-bit Samples		4-bit Samples	
	Magnitude	H_3/H_n db	Magnitude	H_3/H_n db	Magnitude	H_3/H_n db
H_1	0.545E-6	117.3	0.433E-3	59.2	0.566E-2	36.9
H_3	0.398	0.0	0.397	0.0	0.397	0.0
H_5	0.245	4.2	0.245	4.2	0.233	4.6
H_7	0.422E-7	139.5	0.668E-4	75.5	0.875E-3	53.1

* $H_1 = 2f_{lo}$

TABLE VIII

STRENGTH IN DB OF LARGEST UNDESIRED FREQUENCY WITHIN PASS BAND VERSUS SAMPLE SIZE

Size of Samples in Bits	Output Frequency Setting		
	f_{lo}	$5f_{lo}$	$6f_{lo}$
4	- 36.7 ($3f_{lo}$)	- 35.3 (f_{lo})	- 36.9 ($2f_{lo}$)
7	- 51.9 ($5f_{lo}$)	- 47.4 (f_{lo})	- 59.2 ($2f_{lo}$)
31	-122 ($3f_{lo}$)	-118 ($3f_{lo}$)	-117 ($2f_{lo}$)

In Table VIII in-band noise frequencies were collected from Tables V, VI, and VII. In-band is defined here as coincident with the pass band of a low pass filter having a corner frequency of $5f_{lo}$. The entry in the 4-bit row and under column f_{lo} is read "the strength of the strongest noise in the pass band is -36.7 db (referred to f_{lo}) and its frequency is $3f_{lo}$."

Analyzing Table VIII further, for any output frequency setting the noise decreases as the sample size is increased. The 4-bit word case would fail a -40 db noise specification, while the 7-bit word would pass. The number of samples (and this goes inversely with the output frequency) seems to have no effect on in-band noise. This implies that sampling effects appear to have no bearing on word size.

Simulation at the Register Transfer Level

Programming Considerations

The following comments are concerned with the format and language conventions for modelling the Frequency Synthesizer.

A source program deck is composed of statement, control, and data cards. Statement cards provide a description of a digital process or sequence. Control cards specify the user's commands and options to the simulator program. Data cards supply values for initialization and testing.

There are two types of statements: configuration statements and sequence description statements. The former declare the types of components to be used in the model. The latter are executable statements which define operations.

Logic operators .AND., .OR., .NOT. are used with multi-bit operands such as register contents. Incrementing by 1 is done with the .COUNT. operator.

Counters, data buses, and address lines are declared as registers. Constants and subscripts must be integers.

Models and Source Programs

The block diagram of Figure 7 may be considered as a tentative system configuration. We can model it in register transfer language in order to simulate operations needed for selecting a frequency and for table look-up. These test the correctness of the size of registers and adders and also indicate if a sinusoidal sequence of samples is being read out.

A flow chart (Figure 10) was drawn after studying Figure 7 and the timing charts of Figure 8, 9. The flow chart is a guide for writing the register transfer description. It is mainly used for sequence statements and need not contain configuration data.

Based on the flow chart of Figure 10 a source program was written (see Table IX for listing). The configuration statements come before the sequence description statements. Following these are the options and commands from the

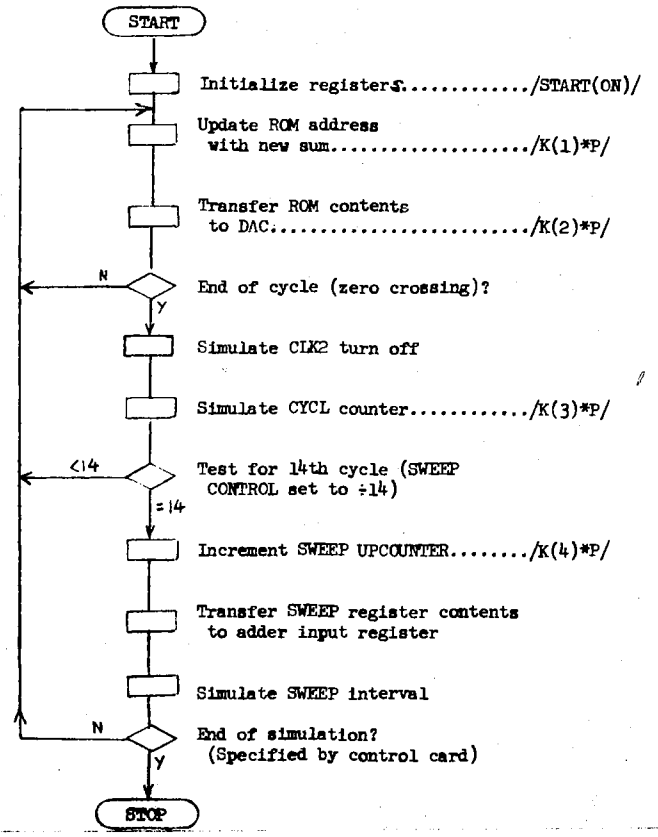


Figure 10. Flow Chart Used to Construct Register Transfer Description

TABLE IX

REGISTER TRANSFER DESCRIPTION USED AS THE SYSTEM SIMULATION PROGRAM

```
C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C          "FREQUENCY SYNTHESIZER" (SIMPLIFIED MODEL)
C
C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C          < CONFIGURATION STATEMENTS >
C
 REGISTER,      ADRS(0-6), DAC(0-6), INC(0-4),
1               STEP(0-3), CYCL(0-5), SWEEP(0-3)
C
C    DECLARE INPUT AND ADDRESS LINES AS REGISTERS:
 REGISTER,     IN1(0-4), ABUS(0-5)
C
 DECODER,       K(1-4) = STEP
C
 TERMINAL,      SUM = ADRS .ADD. INC
C
 MEMORY,        ROM(ABUS) = ROM(0-63,0-6)
C
 SWITCH,        START(ON)
C
 CLOCK,         P
C
C          < SEQUENCE  DESCRIPTIONS >
C
 /START(ON)/    STEP=1, ADRS=0,
                CYCL=0, SWEEP=7, DAC=1,
                INC=IN1
C
C  (ADRS REGISTER STROBE PULSE OCCURS AT K(1)*P TIME)
C
 /K(1)*P/       ADRS=SUM,
                ABUS=ADRS(1-6), STEP=2
C
 /K(2)*P/       DAC=ROM(ABUS),
C
C  ADRS(0) = MSB
C
                IF(ADRS(0).EQ.1) THEN(STEP=3) ELSE ( STEP=1 )
C
 /K(3)*P/       CYCL=CYCL .COUNT., ADRS(0)=0,
                IF( CYCL .EQ. 14) THEN( STEP=4,CYCL=0 ) ELSE( STEP=1 )
C
 /K(4)*P/       SWEEP=SWEEP .COUNT., STEP=1, INC=SWEEP,
                IF( SWEEP .EQ. 11) THEN ( SWEEP =1 )
C
CONTROL SEQUENCE STOPS WHEN NO. OF CLOCK PERIODS SPECIFIED IN  CONTROL
CARD IS EXCEEDED.
C
```

```
*SIMULATE
IN SIMULATE PHASE

                    SIMULATION CONTROL CARDS:


*OUTPUT         CLOCK(1)=ADRS,DAC,STEP,CYCLE,SWEEP,ABUS,INC
*SWITCH     1, START=ON
*LOAD

                    IN1 = 16

       ROM(0-)=1,2,3,4,5,6,7,8,9,16,

       ROM(10-)=17,18,19,20,21,22,23,24,25,32,

       ROM(20-)=33,34,35,36,37,38,39,40,41,48,

       ROM(30-)=49,50,51,52,53,54,55,56,57,64,

       ROM(40-)=65,66,67,68,69,70,71,72,73,80,

       ROM(50-)=81,82,83,84,85,86,87,88,89,96,

       ROM(60-)=97,98,99,100
*SIM      500,3
```

control cards and finally the values specified by data cards.

Components declared in the configuration statements of Table IX are identified as follows:

ADRS (0 - 6) signifies that ADRS is the name given to the address register in Figure 7, that it is 7 bits long, and the most significant bit is at the left end of the register (bit position 0). The input register and its input bus are named INC and IN1 respectively. (NOTE: These were declared 5 bits instead of 6 bits. The ROM was decreased to 64 words. The adder was shortened to 6 bits for proper overflow (modulo 64). These valid changes were made to simplify the model.)

ABUS is the address bus. CYCL and SWEEP are registers for simulating the cycle counter and sweep up-counter respectively.

The DECODER declaration provides sequencing and branching capabilities. The statement

K (1-4) = STEP

provides for four control steps. These are shown as /K (1)*P/ through /K(4)*P/ in the flow chart. The variable STEP assigned to decoder K takes values 1-4 from the sequence description statements and thus selects K(1) through K(4).

The TERMINAL declaration defines a variable SUM and assigns to it the sum of the contents of the registers

represented by ADRS and INC. Note that operator .ADD. is used in place of "+".

The MEMORY statement declares a ROM whose address values are taken from ABUS and contains 64 7-bit words.

The SWITCH declaration permits a start label to be used for initialization as required in Figure 10.

"CLOCK, P" defines a 1-phase clock, P. Figure 9 indicates that two clock sources are needed. CLK2 is derived from CLK1 and hence clock P is sufficient.

Sequence description cards. This part of Table IX corresponds 1:1 with Figure 10, the register transfer statements replacing the legends of the flow chart.

/START(ON)/ initialized registers at the start of a run to the values listed. This may be checked against the first line of the simulation printout, Table X.

The labels /K(1)*P/ through /K(4)*P/ have the same purpose as statement numbers in Fortran. That is, STEP=2 may cause a branch to the statement with label /K(2)*P/ at clock time. Note that the equal sign is used in STEP=2, while the operator .EQ. is used for specifying logical tests as in IF(CYCL .EQ. 14) THEN... Parallel operation is assumed whenever several statements are given under one label.

Near the end of Table IX we have *SIMULATE which calls the simulator program. The next two lines are messages and do not represent control cards.

TABLE X

SAMPLE OUTPUT FROM SYSTEM SIMULATION

REGISTER	TRANSFER	SIMULATION	PRINTOUT

```
TITLE:        "FREQUENCY SYNTHESIZER MODEL" (FOR USER'S MANUAL)  /BP/ FEB'73


SWITCH INTERRUPT
     STAR =   ON
     ADRS =   ..00      DAC =   ..01      STEP =   ...1      CYCL =   ..00      SWEE =   ...7      ABUS =   ..00      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    1              ACTIVE LABELS                CLOCK TIME =    1
                                /K(1  )*P/
     ADRS =   ..10      DAC =   ..01      STEP =   ...2      CYCL =   ..00      SWEE =   ...7      ABUS =   ..00      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    2              ACTIVE LABELS                CLOCK TIME =    2
                                /K(2  )*P/
     ADRS =   ..10      DAC =   ..01      STEP =   ...1      CYCL =   ..00      SWEE =   ...7      ABUS =   ..00      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    3              ACTIVE LABELS                CLOCK TIME =    3
                                /K(1  )*P/
     ADRS =   ..20      DAC =   ..01      STEP =   ...2      CYCL =   ..00      SWEE =   ...7      ABUS =   ..10      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    4              ACTIVE LABELS                CLOCK TIME =    4
                                /K(2  )*P/
     ADRS =   ..20      DAC =   ..17      STEP =   ...1      CYCL =   ..00      SWEE =   ...7      ABUS =   ..10      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    5              ACTIVE LABELS                CLOCK TIME =    5
                                /K(1  )*P/
     ADRS =   ..30      DAC =   ..17      STEP =   ...2      CYCL =   ..00      SWEE =   ...7      ABUS =   ..20      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    6              ACTIVE LABELS                CLOCK TIME =    6
                                /K(2  )*P/
     ADRS =   ..30      DAC =   ..33      STEP =   ...1      CYCL =   ..00      SWEE =   ...7      ABUS =   ..20      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    7              ACTIVE LABELS                CLOCK TIME =    7
                                /K(1  )*P/
     ADRS =   ..40      DAC =   ..33      STEP =   ...2      CYCL =   ..00      SWEE =   ...7      ABUS =   ..30      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    8              ACTIVE LABELS                CLOCK TIME =    8
                                /K(2  )*P/
     ADRS =   ..40      DAC =   ..49      STEP =   ...3      CYCL =   ..00      SWEE =   ...7      ABUS =   ..30      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
LABEL CYCLE    9              ACTIVE LABELS                CLOCK TIME =    9
                                /K(3  )*P/
     ADRS =   ..00      DAC =   ..49      STEP =   ...1      CYCL =   ..01      SWEE =   ...7      ABUS =   ..30      INC =   ..10
- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - - - - - - -|- - - - -
```

TABLE X (CONTINUED)

LABEL CYCLE 133	ACTIVE LABELS /K(1)*P/	CLOCK TIME = 133					
ADRS = ..40	DAC = ..33	STEP = ...2	CYCL = ..0E	SWEE = ...7	ABUS = ..30	INC = ..10	
LABEL CYCLE 134	ACTIVE LABELS /K(2)*P/	CLOCK TIME = 134					
ADRS = ..40	DAC = ..49	STEP = ...3	CYCL = ..0E	SWEE = ...7	ABUS = ..30	INC = ..10	
LABEL CYCLE 135	ACTIVE LABELS /K(3)*P/	CLOCK TIME = 135					
ADRS = ..00	DAC = ..49	STEP = ...4	CYCL = ..00	SWEE = ...7	ABUS = ..30	INC = ..10	
LABEL CYCLE 136	ACTIVE LABELS /K(4)*P/	CLOCK TIME = 136					
ADRS = ..00	DAC = ..49	STEP = ...1	CYCL = ..00	SWEE = ...8	ABUS = ..30	INC = ..07	
LABEL CYCLE 137	ACTIVE LABELS /K(1)*P/	CLOCK TIME = 137					
ADRS = ..07	DAC = ..49	STEP = ...2	CYCL = ..00	SWEE = ...8	ABUS = ..00	INC = ..07	
LABEL CYCLE 138	ACTIVE LABELS /K(2)*P/	CLOCK TIME = 138					
ADRS = ..07	DAC = ..01	STEP = ...1	CYCL = ..00	SWEE = ...8	ABUS = ..00	INC = ..07	
LABEL CYCLE 139	ACTIVE LABELS /K(1)*P/	CLOCK TIME = 139					
ADRS = ..0E	DAC = ..01	STEP = ...2	CYCL = ..00	SWEE = ...8	ABUS = ..07	INC = ..07	
LABEL CYCLE 140	ACTIVE LABELS /K(2)*P/	CLOCK TIME = 140					
ADRS = ..0E	DAC = ..08	STEP = ...1	CYCL = ..00	SWEE = ...8	ABUS = ..07	INC = ..07	
LABEL CYCLE 141	ACTIVE LABELS /K(1)*P/	CLOCK TIME = 141					
ADRS = ..15	DAC = ..08	STEP = ...2	CYCL = ..00	SWEE = ...8	ABUS = ..0E	INC = ..07	
LABEL CYCLE 142	ACTIVE LABELS /K(2)*P/	CLOCK TIME = 142					
ADRS = ..15	DAC = ..15	STEP = ...1	CYCL = ..00	SWEE = ...8	ABUS = ..0E	INC = ..07	
LABEL CYCLE 143	ACTIVE LABELS /K(1)*P/	CLOCK TIME = 143					
ADRS = ..1C	DAC = ..15	STEP = ...2	CYCL = ..00	SWEE = ...8	ABUS = ..15	INC = ..07	
LABEL CYCLE 144	ACTIVE LABELS /K(2)*P/	CLOCK TIME = 144					
ADRS = ..1C	DAC = ..22	STEP = ...1	CYCL = ..00	SWEE = ...8	ABUS = ..15	INC = ..07	
LABEL CYCLE 145	ACTIVE LABELS /K(1)*P/	CLOCK TIME = 145					
ADRS = ..23	DAC = ..22	STEP = ...2	CYCL = ..00	SWEE = ...8	ABUS = ..1C	INC = ..07	

TABLE X (CONTINUED)

LABEL CYCLE 418		ACTIVE LABELS /K(2)*P/		CLOCK TIME = 418									
ADRS = ..2A	DAC = ..36	STEP = ...1	CYCL = ..0E	SWEE = ...8	ABUS = ..23	INC = ..07							
LABEL CYCLE 419		ACTIVE LABELS /K(1)*P/		CLOCK TIME = 419									
ADRS = ..31	DAC = ..36	STEP = ...2	CYCL = ..0E	SWEE = ...8	ABUS = ..2A	INC = ..07							
LABEL CYCLE 420		ACTIVE LABELS /K(2)*P/		CLOCK TIME = 420									
ADRS = ..31	DAC = ..43	STEP = ...1	CYCL = ..0E	SWEE = ...8	ABUS = ..2A	INC = ..07							
LABEL CYCLE 421		ACTIVE LABELS /K(1)*P/		CLOCK TIME = 421									
ADRS = ..38	DAC = ..43	STEP = ...2	CYCL = ..0E	SWEE = ...8	ABUS = ..31	INC = ..07							
LABEL CYCLE 422		ACTIVE LABELS /K(2)*P/		CLOCK TIME = 422									
ADRS = ..38	DAC = ..50	STEP = ...1	CYCL = ..0E	SWEE = ...8	ABUS = ..31	INC = ..07							
LABEL CYCLE 423		ACTIVE LABELS /K(1)*P/		CLOCK TIME = 423									
ADRS = ..3F	DAC = ..50	STEP = ...2	CYCL = ..0E	SWEE = ...8	ABUS = ..38	INC = ..07							
LABEL CYCLE 424		ACTIVE LABELS /K(2)*P/		CLOCK TIME = 424									
ADRS = ..3F	DAC = ..57	STEP = ...1	CYCL = ..0E	SWEE = ...8	ABUS = ..38	INC = ..07							
LABEL CYCLE 425		ACTIVE LABELS /K(1)*P/		CLOCK TIME = 425									
ADRS = ..46	DAC = ..57	STEP = ...2	CYCL = ..0E	SWEE = ...8	ABUS = ..3F	INC = ..07							
LABEL CYCLE 426		ACTIVE LABELS /K(2)*P/		CLOCK TIME = 426									
ADRS = ..46	DAC = ..64	STEP = ...3	CYCL = ..0E	SWEE = ...8	ABUS = ..3F	INC = ..07							
LABEL CYCLE 427		ACTIVE LABELS /K(3)*P/		CLOCK TIME = 427									
ADRS = ..06	DAC = ..64	STEP = ...4	CYCL = ..00	SWEE = ...8	ABUS = ..3F	INC = ..07							
LABEL CYCLE 428		ACTIVE LABELS /K(4)*P/		CLOCK TIME = 428									
ADRS = ..06	DAC = ..64	STEP = ...1	CYCL = ..00	SWEE = ...9	ABUS = ..3F	INC = ..08							
LABEL CYCLE 429		ACTIVE LABELS /K(1)*P/		CLOCK TIME = 429									
ADRS = ..0E	DAC = ..64	STEP = ...2	CYCL = ..00	SWEE = ...9	ABUS = ..06	INC = ..08							
LABEL CYCLE 430		ACTIVE LABELS /K(2)*P/		CLOCK TIME = 430									
ADRS = ..0E	DAC = ..07	STEP = ...1	CYCL = ..00	SWEE = ...9	ABUS = ..06	INC = ..08							

The *OUTPUT card contains a list of variables to be printed out as in Table X. *SWITCH initiates the operation of a manual switch. "1, START(ON)" specifies the Label Cycle before which the switch operates. In the example, "1" is supplied and any larger number N would constitute simulating a delayed operation (turn on would wait for the N-1 th Label Cycle).

*LOAD tells the simulator to enter the constants listed. In the example, register IN1 will be initialized to 16, while the 64 ROM words are to be assigned the values listed. Thus "ROM(0-)=1,2,3...." means that address 0 will contain a 1, etc. The values in the example are not sine wave samples for simplicity but were chosen to force decimal digits to be printed for DAC.

"*SIM 500,3" is the last card and specifies that the simulation shall run for 500 Label Cycles. The 3 specifies that simulation would stop if in any three consecutive Label Cycles, the same group of labels were activated. This feature prevents "infinite loops" from wasting computer time and paper.

Simulation Results

Table X shows three of the five pages of results printed out after executing the source program. To verify the model, the contents of registers and other components at certain clock times are obtained from the table. (For compactness register values are printed in hexadecimal.

Thus in the row marked Label Cycle 1 of Table X, INC contains the initial value 10 which is 16 in decimal.)

By examining a few values in each column of the printout one may be able to determine if the flow chart of Figure 10 was followed. If it was, then the printout is assumed valid and further analysis may be done. Proceeding with the address register ADRS, a new value should come every /K(1)*P/ time. This is seen in every row where /K(1)*P/ is active, e.g. Label Cycles 1,3,5 and so on. It is clear that the contents of INC are being transferred and accumulated in ADRS. Label Cycles 136 and 428 depict the condition when INC changes value. The effect on ADRS may be seen in the next clock period.

The time series of samples are represented by the DAC sequence. Updating occurs at /K(2)*P/ as may be seen in Label Cycle 2. Label Cycle 128 shows that the DAC sequence is cyclic. This is due to the address sequence being cyclic also (modulo 64). (DAC numbers are read as decimal due to the way the ROM words were initialized.)

CYCL is incremented at Label Cycles 9 and 135 for the pages included in Table X. In each case the DAC values show that the end of a cycle occurred. In Label Cycle 427, after counting to 14, CYCL is reset to zero as required.

Simulation of the sweep mode is shown partially at Labels 136 and 428. Here the active label is /K(4)*P/. INC replaces its previously constant value with 07 in Label

Cycle 136, while it is incremented by 1 in Label Cycle 428. The latter cycle indicates the start of the "frequency sweep" action required in this mode.

CHAPTER IV

SYNCHRONOUS LOGIC DESIGN

In this chapter we use CAD Programs to design the non-standard functions of the digital frequency synthesizer discussed in Chapter III. These functions are the synchronization indicator, frequency sweep up-counter, and cycle counter.

All the examples to be discussed are used to implement the non-standard functions. In practice they would be assembled from gate level components known in the industry as small scale integrated (SSI) circuits. The other functional blocks in the system block diagram may be conveniently built from pre-designed medium scale integrated (MSI) circuits.

Input Formats

The set of punched cards needed to input a state diagram's data to the Sequential Logic Synthesis Program consists of a title card, node/transition cards (one per state), *STATES card, state assignment cards (one per state), and an "end" card containing a single *. They must be submitted in that order.

The title card contains in columns 6-45 any text for identifying the printout.

A node/transition card has three fields which are used as follows: column 1-16 for the name of a node, column 21-36 for successor node name, column 41-80 for the transition signal or a Boolean expression of several signals.

The *STATES card marks the beginning of the state assignment cards.

A state assignment card has two fields for defining the binary code assigned to a node. The octal equivalent of the code is punched in column 1-2, while column 21-36 is for the name of the node involved.

In the case of program LOGICMIN, one card is punched for each row of a truth table. The first card of a set specifies the number of input variables (columns 2-5) and also serves as a title card (columns 6-45 for text). The number of outputs is not declared.

The format of the truth table data card is as follows:

INPUT STATE/OUTPUT STATE

For a 4-variable problem for example, 12/-01 denotes a row of the truth table with the inputs (W,X,Y,Z)=1100 and outputs (F1,F2,F3)=-01. A second format option allows coding the input n-tuple in octal for rows with minterms, and binary-dash notation for rows with don't cares. The output is always in binary-dash.

Control Logic Design Example

The synchronization indicator of the Frequency Synthesizer detects synchronism between an external signal and CLK2 (see Figure 7). Its output may be used to stop the frequency sweep up-counter. Introduced in Chapter II, its state diagram (Figure 2) and schematic (Figure 4) are found therein.

As shown in Appendix A the printout from the Synchronous Logic Synthesis Program includes a record of the state diagram data submitted. In this regard the clock signal is implicit and is not AND-ed with the input variable X. Note that because of the don't care condition for the S7 to S1 transition, Line 12 of the present example does not have an input signal.

The second page of the printout contains the JKFF input equations from which the logic schematic of Figure 4 was drawn. A small table labelled "Input Summary" is provided for documentation.

The output gating expression was minimized by LOGICMIN. The four data cards which encoded the truth table were punched as 4/1, 3/2, 2/1, 1/1. The output expression returned by the program was

$$F1 = \bar{A}BC + \bar{A}C + \bar{A}B.$$

Shift Registers

The shift register in this example is a pattern generator. It shifts out a pattern used to test the synchronization indicator. As such it may be included in the actual hardware as a built in test signal source.

The circuit shown in Figure 11 was designed for RSFF type. The front-end JKFF was therefore provided with an inverter. This circuit resulted from coding the state transition table as a 5-variable single-output gating function and then submitted to LOGICMIN.

Counters

The CYCL counter divides its clock pulses by 14,28,42, or 56 depending on a 2-bit control word. Instead of being designed as a single counter with six JKFF's, it was partitioned into a variable-modulo (÷ 2, 4, 6, 8) counter and a ÷ 7 counter. Because the former is basically a binary counter the pair may have fewer gates than a single non-binary counter. Further the small counters may be used separately as building blocks.

The frequency sweep up-counter supplies the values 7, 8, 9, 10, 11, 7, ... in binary to an accumulator. A modulo-5 counter and output gating may be used.

Variable-Modulo

With input variables V, W selection of a modulo is depicted in Figure 12. The printout of results is included

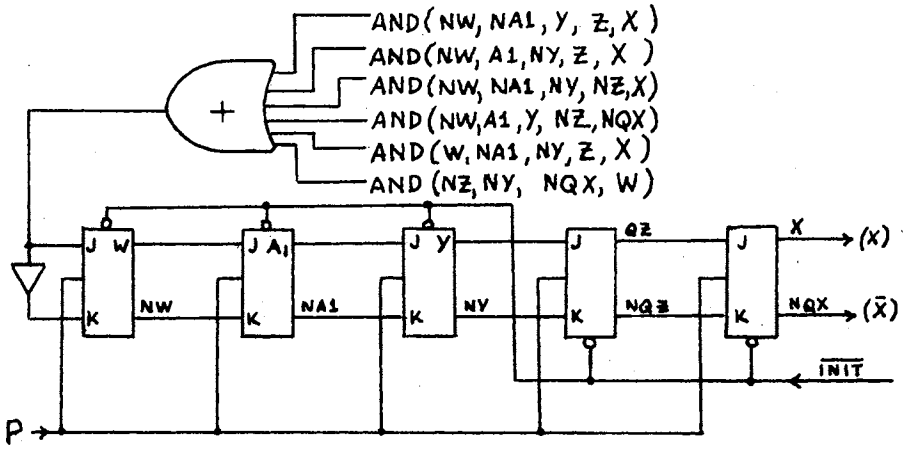


Figure 11. Shift Register for Testing Synchronization Indicator

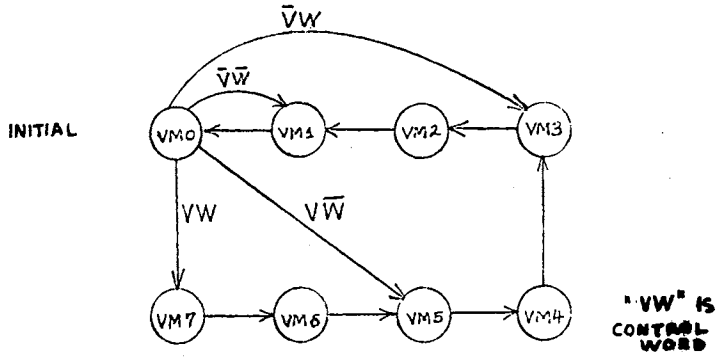


Figure 12. State Diagram Specifying Variable Modulo Counter

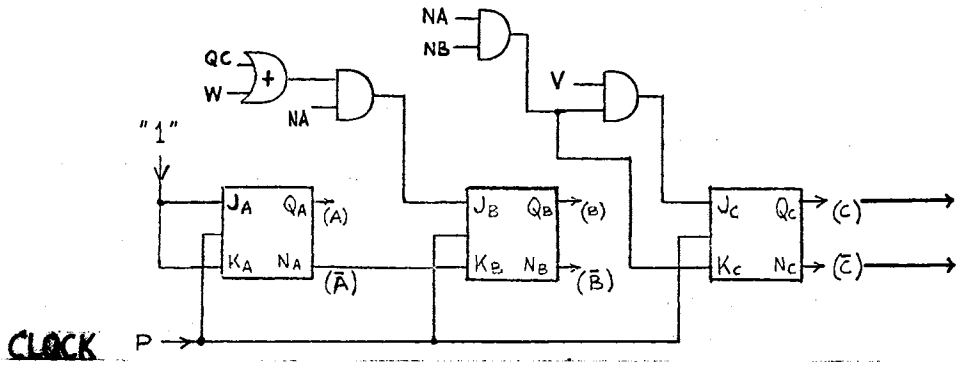


Figure 13. Synchronous Variable Modulo Counter Logic Diagram Drawn from Equations Generated by CAD Program

in Appendix A starting with the third page. In this page the state diagram data from Figure 12 is shown along with state assignments. The fourth page gives the design equations for the counter of Figure 13.

The output gating is just one 3-input NAND gate whose output is used as the clock for the ÷ 7 counter (not shown in Figure 13). The inputs of the NAND gate go 1:1 with the complimented outputs of the JKFF.

Divide-by-7

Figure 14 is a state diagram with nodes labelled according to the state assignment. This assignment produced the equations

$$J_A = \bar{B}\bar{C} = K_A$$

$$J_B = \bar{C} = K_B$$

$$J_C = A + B$$

$$K_C = 1.$$

The logic diagram for the above equations is shown in Figure 15. The output is taken directly from the flip-flop FFA.

Modulo 5

Given the state diagram of Figure 16 the program generated the equations

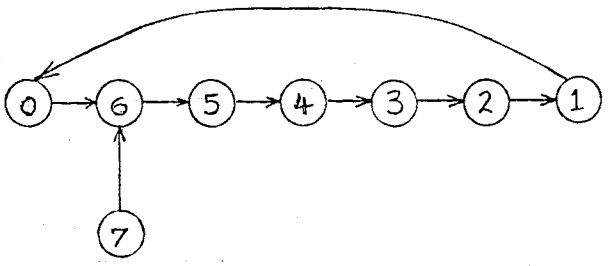


Figure 14. State Diagram Specifying Divide-by-7 Counter

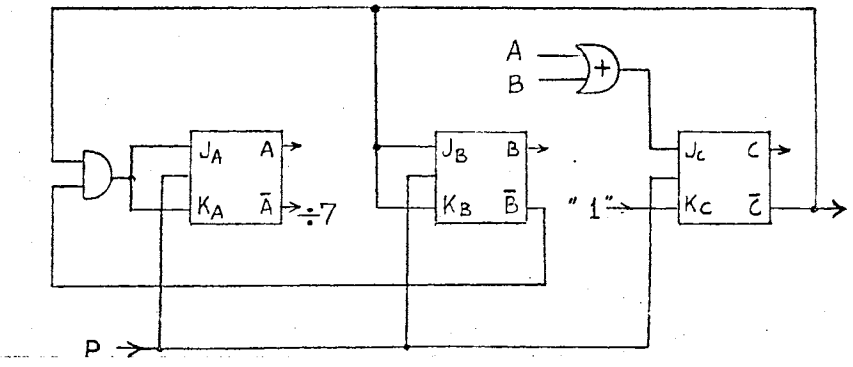


Figure 15. Divide-by-7 Counter Obtained from CAD Equations

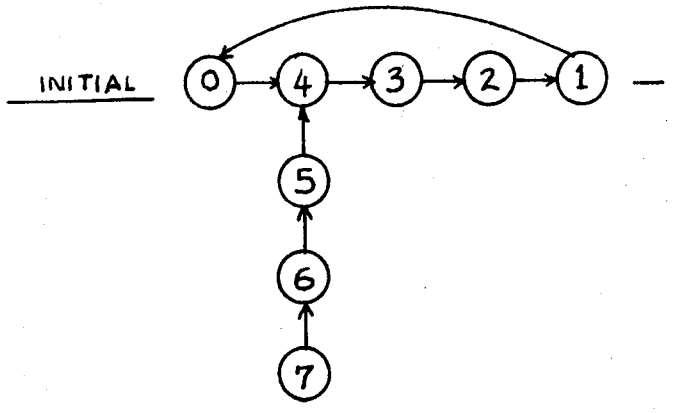


Figure 16. State Assignment for Modulo-5 Counter Used in Frequency Sweep Logic

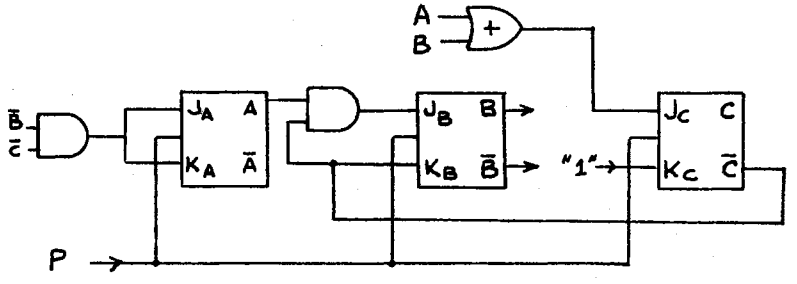


Figure 17. Modulo-5 Counter with No Latch Up States

$$J_A = K_A = \bar{B}\bar{C}$$

$$J_B = A\bar{C}$$

$$K_B = \bar{C}$$

$$J_C = A + B$$

$$K_C = 1.$$

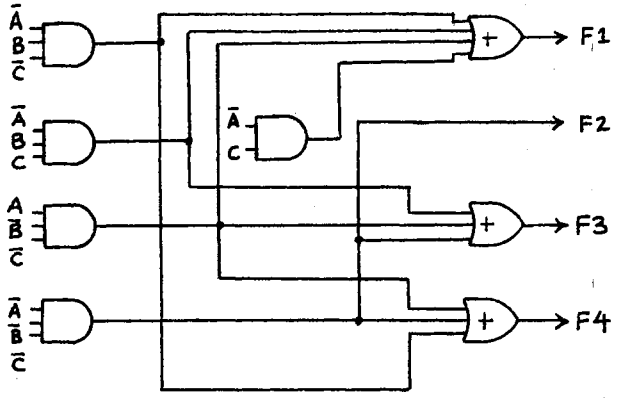
Figure 17 is the logic diagram of a counter drawn from the above equations. This counter has no latch up states because there is always a next state for any present state. This is in contrast with designs where an unused state could be entered due to noise.

Output Gating

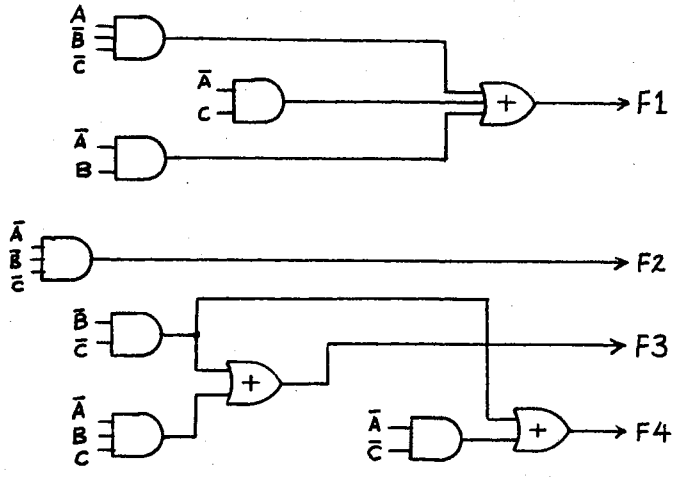
The output gating of the modulo-5 counter must encode the 3-bit internal state into 4-bit words. This makes it into a frequency sweep up-counter.

The truth table of the output gating is given in the second page of Appendix B. Two solutions from a set of six are included in the third page. The literals XYZ correspond to ABC in Figure 18 where the two solutions have been drawn as logic diagrams.

Part (a) of Figure 18 differs from part (b) in number of input lines, gate count, and size of gates. The first solution has fewer input lines and gates than the second. However the second uses mostly 2-input gates compared to mostly 3-input and one 4-input gate for the first. In other words the solutions are based on two different cost



a.) Minimized with Respect to Type 1 Criterion



b.) Minimized under Type 2 Criterion

Figure 18. Frequency Sweep Output Gating Obtained from Multiple Output Prime Implicants

criteria, and the user selects the one which applies to the problem at hand.

If the gating were a single-output function the print-out from LOGICMIN would be in the form shown in the last page of Appendix B.

State Assignment

Because the state assignment can be easily changed and a number of these trials submitted at the same time, the Synchronous Logic Synthesis Program may be used to search for improved assignments. Since evaluation is by inspection (compare equations of each solution) the number of trials will be finite. This number (and the speed involved) is still large in comparison with what one gets by paper and pencil methods.

An example of how a cluster of possible trials may develop in state assignment may be made by considering the rule:

> Two or more states having the same state for a given input state should be made adjacent.

In practice, for the medium size design range of 4 - 7 FF's with 2 or 3 input signals one may see several nodes of a state diagram where the rule may be applied, and dozens of ways to make the affected pairs (or sets) of states adjacent.

CHAPTER V

COMPUTER PROGRAM DOCUMENTATION

Three programs were implemented in this study: Synchronous Logic Synthesis, LOGICMIN, and Register Transfer Simulator. Because most of the programming effort was invested in it and due to space requirements, only the synthesis program and related routines have been listed.

Synchronous Logic Synthesis Program

The flow chart of this program is depicted in Figure 19. Appendix C contains the listing. This version is in Fortran IV and ran on the IBM 360/65. The flow chart and listing refer to certain techniques and data structures used in another program (see below).

Asynchronous Logic Synthesis Program

This program was based on the so-called Cube Logic Technique of Senders and Lucchesi [12]. The flow chart of Figure 20 is quite detailed and displays statement numbers used in the program (Appendix D). These items are supplementary to Appendix C.

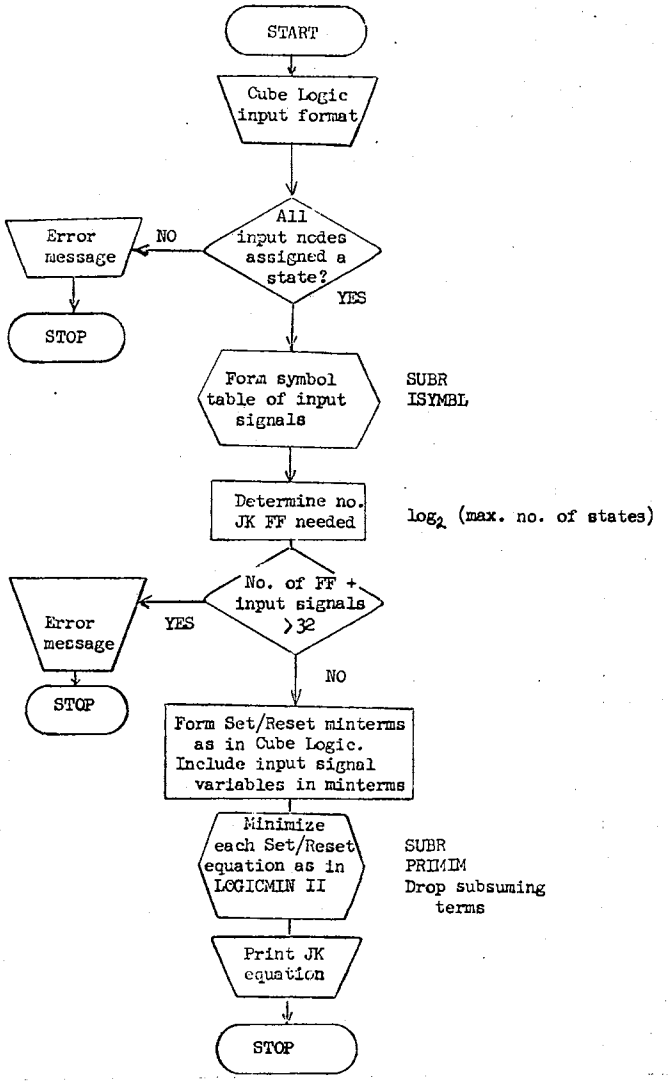


Figure 19. Flow Chart of Synchronous Logic Synthesis Program

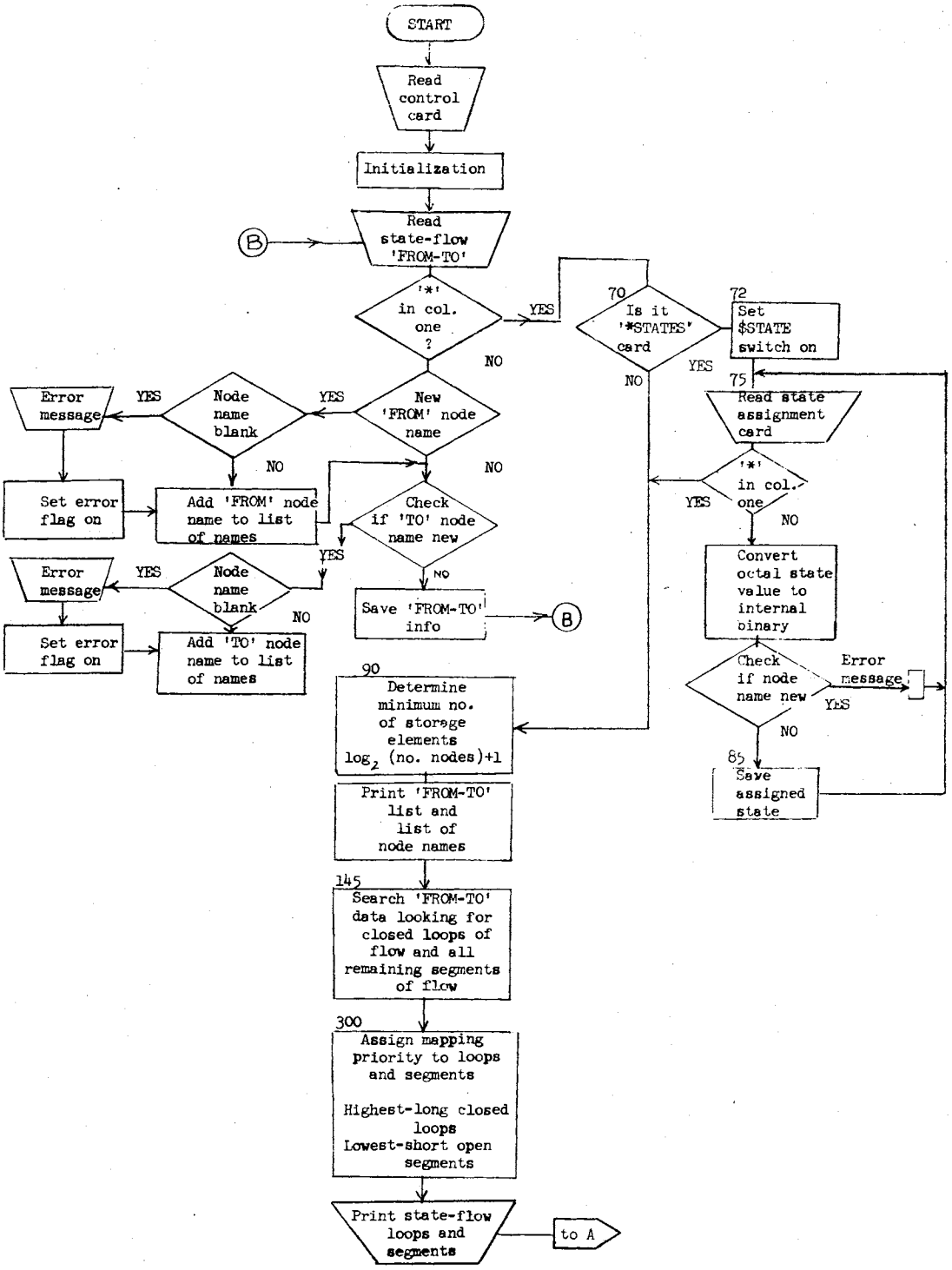


Figure 20. Flow Chart of Asynchronous Logic Synthesis Program

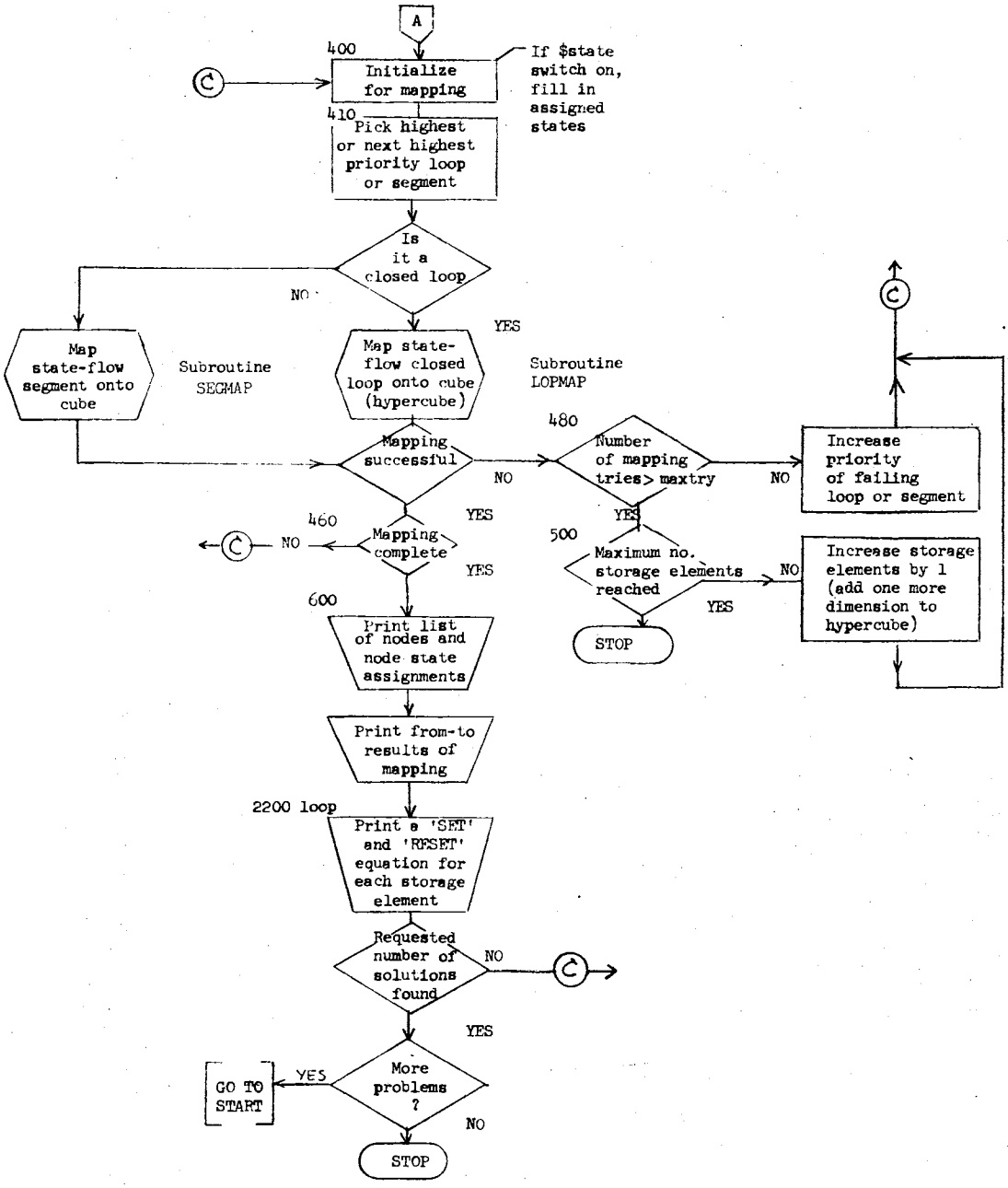


Figure 20. (Continued)

CHAPTER VI

SUMMARY AND CONCLUSIONS

Summary

This study was concerned with implementing a set of programs applicable for the computer-aided design (CAD) of small digital systems. Three areas of design for which CAD tools were desired are system level, gate level, and non-digital aspects. The thesis summarizes (1) approaches used to implement programs that were not available (2) a methodology for using these and existing programs (3) application of the method and programs in designing a digital frequency synthesizer.

Programs were written for the synthesis of a synchronous sequential circuit specified by its state diagram. A register transfer simulator program was also implemented. Gate level simulators and programs for servicing non-digital aspects are available commercially. With the digital frequency synthesizer as a case study it was found that the programs can assist the designer in synthesis of non-standard functions, verifying designs by simulation, and in dealing with analog oriented problems.

Recommendations for Further Study

Synthesis techniques at the system level seem to be non-existent. As an extension of the register transfer concept and using a design language, it should be possible to devise a synthesis procedure which operates on register transfer modules.

There appears to be an absence of efficient algorithms for flow table reduction and state assignment. Consequently the logic synthesis program can be improved if automatic state minimization and optimal assignment capabilities were added.

A SELECTED BIBLIOGRAPHY

(1) Duley, J. and Dietmeyer. "Translation of DDL Digital System Specification to Boolean Equations." IEEE Trans. Comput., Vol. C-18 (1969), 305-313.

(2) Baray, M. B., S. Y. Su, and R. Carberry. "The Structure and Operation of a Design Language Simulator." Proceedings of the 8th Annual Design Automation Workshop, Atlantic City, N. J.; 1971.

(3) Chu, Y. Introduction to Computer Organization. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1970.

(4) Friedman, A. and P. Menon. "Restricted Checking Sequences for Sequential Machines." IEEE Trans. Comput., Vol. C-22 (1973), 397-399.

(5) Givone, D. Introduction to Switching Circuit Theory. New York: McGraw-Hill; 1970, pp. 402-436.

(6) Story, J., H. Harrison, and E. Reinhard. "Optimum State Assignment for Synchronous Sequential Circuits." IEEE Trans. Comput., Vol. C-21 (1972), 1365-1373.

(7) LOGSIM User's Manual. 525 Univ. Ave. Palo Alto, Cal.: Tyme-share Inc., 1971.

(8) Givone, D. Introduction to Switching Circuit Theory. New York: McGraw-Hill; 1970, pp. 181-194.

(9) Basore, B. L. The Fast Fourier Transform: Speed from Cyclic Repetition. ARO Monograph Project, College of Engineering, Oklahoma State University, (1970).

(10) MATCH: Applicon's Optimizing Filter Synthesis Program. 83 Second Ave. Burlington, Mass.: Applicon Inc., 1972.

(11) Tierney, J., et al. "A Digital Frequency Synthesizer." in Digital Signal Processing (Rabiner, L., ed.). New York: IEEE Press; 1972.

(12) Senders, S. and J. Lucchesi. "Design of Sequential Switching Circuits with the Cube Logic Technique." Computer Design, Vol. 10 (1971), 59-64.

APPENDIX A

SAMPLE INPUT CODING AND PROGRAM PRINTOUT: SYNCHRONIZATION INDICATOR AND VARIABLE MODULO COUNTER

```
"SYNCHRONISM INDICATOR" LOGIC (J-K FF'S   TIME= 21.23.45   DATE=   73.135

 INPUT DATA

   1  FROM= S0                TO= S1                INPUT SIGNALS= X
   2  FROM= S1                TO= S2                INPUT SIGNALS= X
   3  FROM= S1                TO= S3                INPUT SIGNALS= ¬X
   4  FROM= S2                TO= S3                INPUT SIGNALS= ¬X
   5  FROM= S3                TO= S4                INPUT SIGNALS= X
   6  FROM= S4                TO= S3                INPUT SIGNALS= X
   7  FROM= S4                TO= S5                INPUT SIGNALS= ¬X
   8  FROM= S5                TO= S3                INPUT SIGNALS= ¬X
   9  FROM= S5                TO= S6                INPUT SIGNALS= X
  10  FROM= S6                TO= S3                INPUT SIGNALS= X
  11  FROM= S6                TO= S0                INPUT SIGNALS= ¬X
  12  FROM= S7                TO= S1                INPUT SIGNALS=

*STATE

ASSIGNED STATE(OCTAL)= 0              NODE NAME = S0
ASSIGNED STATE(OCTAL)= 6              NODE NAME = S1
ASSIGNED STATE(OCTAL)= 5              NODE NAME = S2
ASSIGNED STATE(OCTAL)= 4              NODE NAME = S3
ASSIGNED STATE(OCTAL)= 3              NODE NAME = S4
ASSIGNED STATE(OCTAL)= 2              NODE NAME = S5
ASSIGNED STATE(OCTAL)= 1              NODE NAME = S6
ASSIGNED STATE(OCTAL)= 7              NODE NAME = S7
```

```
INPUT SUMMARY

    3 J/K FLIP-FLOPS REQUIRED

   12 INPUT FROM-TO CARDS

    8 STATE-FLOW NODES

    1 INPUT SIGNAL VARIABLES



      NODE NAME          ASSIGNED STATE

   1  S0                 000 (¬A¬B¬C)
   2  S1                 110 (AB¬C)
   3  S2                 101 (A¬BC)
   4  S3                 100 (A¬B¬C)
   5  S4                 011 (¬ABC)
   6  S5                 010 (¬AB¬C)
   7  S6                 001 (¬A¬BC)
   8  S7                 111 (ABC)


INPUT VARIABLES

X

 J/K FLIP FLOP    1   EQUATION  1
    -001
    -111
    -100
    -011

  JA = B¬C¬X + ¬BX + CX
       -100    -0-1  --11

 J/K FLIP FLOP    1   EQUATION  2
    -001

  KA = ¬B¬CX
       -001

 J/K FLIP FLOP    2   EQUATION  1
    0-01
    1-01

  JB = ¬CX
       --01

 J/K FLIP FLOP    2   EQUATION  2
    1-01
    1-00
    0-11
    0-00
    0-01

  KB = ¬AX + ¬C
       0--1  --0-

 J/K FLIP FLOP    3   EQUATION  1
    11-1
    10-1
    01-1

  JC = AX  + BX
       1--1  -1-1

 J/K FLIP FLOP    3   EQUATION  2
    10-0
    01-1
    01-0
    00-1
    00-0
    11--

  KC = ¬X  + ¬A  + B
       ---0  0---  -1--
```

```
"VARIABLE MODULO COUNTER"                    TIME= 23.45.53   DATE=   73.135

 INPUT DATA

   1  FROM= VM0          TO= VM1          INPUT SIGNALS= ¬V.¬W
   2  FROM= VM0          TO= VM3          INPUT SIGNALS= ¬V.W
   3  FROM= VM0          TO= VM5          INPUT SIGNALS= V.¬W
   4  FROM= VM0          TO= VM7          INPUT SIGNALS= V.W
   5  FROM= VM7          TO= VM6          INPUT SIGNALS=
   6  FROM= VM6          TO= VM5          INPUT SIGNALS=
   7  FROM= VM5          TO= VM4          INPUT SIGNALS=
   8  FROM= VM4          TO= VM3          INPUT SIGNALS=
   9  FROM= VM4          TO= VM2          INPUT SIGNALS=
  10  FROM= VM2          TO= VM1          INPUT SIGNALS=
  11  FROM= VM1          TO= VM0          INPUT SIGNALS=

*STATES

ASSIGNED STATE(OCTAL)= 0          NODE NAME = VM0
ASSIGNED STATE(OCTAL)= 4          NODE NAME = VM1
ASSIGNED STATE(OCTAL)= 2          NODE NAME = VM2
ASSIGNED STATE(OCTAL)= 6          NODE NAME = VM3
ASSIGNED STATE(OCTAL)= 1          NODE NAME = VM4
ASSIGNED STATE(OCTAL)= 5          NODE NAME = VM5
ASSIGNED STATE(OCTAL)= 3          NODE NAME = VM6
ASSIGNED STATE(OCTAL)= 7          NODE NAME = VM7
```

```
INPUT SUMMARY

    3 J/K FLIP-FLOPS REQUIRED

   11 INPUT FROM-TO CARDS

    8 STATE-FLOW NODES

    2 INPUT SIGNAL VARIABLES



      NODE NAME          ASSIGNED STATE

   1  VM0                000 (¬A¬B¬C)
   2  VM1                100 (A¬B¬C)
   3  VM3                110 (AB¬C)
   4  VM5                101 (A¬BC)
   5  VM7                111 (ABC)
   6  VM6                011 (¬ABC)
   7  VM4                001 (¬A¬BC)
   8  VM2                010 (¬AB¬C)


INPUT VARIABLES

V
W

 J/K FLIP FLOP   1   EQUATION  1
    -0000
    -0001
    -0010
    -0011
    -11--
    -01--
    -10--

  JA = 1
       -----

 J/K FLIP FLOP   1   EQUATION  2
    -11--
    -01--
    -10--
    -00--

  KA = 1
       -----

 J/K FLIP FLOP   2   EQUATION  1
    0-001
    0-011
    0-1--

  JB = ¬AW  + ¬AC
       0---1  0-1--

 J/K FLIP FLOP   2   EQUATION  2
    0-1--
    0-0--

  KB = ¬A
       0----

 J/K FLIP FLOP   3   EQUATION  1
    00-10
    00-11

  JC = ¬A¬BV
       00-1-

 J/K FLIP FLOP   3   EQUATION  2
    00---

  KC = ¬A¬B
       00---
```

APPENDIX B

SPECIMEN PROGRAM OUTPUTS FROM LOGICMIN

```
PROGRAM LOGICMIN

THIS PROGRAM FINDS THE SINGLE OR MULTIPLE-OUTPUT MINIMAL SUMS
GIVEN THE INPUT MINTERMS OF A SUM OF BOOLEAN PRODUCTS AND THE
CORRESPONDING OUTPUT STATES. DON'T CARE STATES ARE ALLOWED
AND ARE REPRESENTED BY A DASH '-'.

INPUT TO THE PROGRAM HAS THE FOLLOWING FORMAT FOR THE PUNCHED
CARDS.

    DECIMAL EQUIVALENT OF MINTERM / FUNCTION OUTPUT STATES

EXAMPLE.

  INPUT
  VARIABLES    OUTPUT STATES              CARDS PUNCHED
   W X Y Z  /  F1 F2 F3                      4 (PROB. IDENT.)
   0 0 1 0  /  -  0  1                    2/-01
   1 1 0 0  /  0  0  1                    12/001
                                          *

  WHERE 0 REPRESENTS A COMPLEMENTED VARIABLE
        1 REPRESENTS AN UNCOMPLEMENTED VARIABLE
        - REPRESENTS DON'T CARE

 A MAXIMUM OF 32 INPUT VARIABLES AND 32 OUTPUTS ALLOWED

 THE FIRST CARD FOR EACH PROBLEM MUST CONTAIN THE FOLLOWING
   CC 2-5  THE NUMBER OF INPUT VARIABLES
   CC 6-45  A PROBLEM IDENTIFICATION

OUTPUT FROM THE PROGRAM CONSISTS OF THE FOLLOWING

 1. FOR SINGLE OUTPUT PROBLEMS
   A. A LIST OF THE PRIME IMPLICANTS
   B. PRIME IMPLICANT TABLE (NOT NECESSARILY FREE FROM HAZARDS)
   C. PRIME IMPLICANT TABLE FOR HAZARD FREE DESIGNS
   D. THE POSSIBLE MINIMAL SUMS BASED ON TABLE B.

 2. FOR MULTIPLE-OUTPUT PROBLEMS
   A. A LIST OF THE PRIME IMPLICANTS AND THE OUTPUT FUNCTIONS IMPLIED
   B. THE POSSIBLE MINIMAL SUMS (NOT NECESSARILY HAZARD-FREE)
      I. MINIMAL SUMS FROM MULTIPLE-OUTPUT PRIME IMPLICANTS
      II. MINIMAL SUMS AFTER ADDITIONAL MINIMIZATION OF EACH OUTPUT FUNCTION
```

RAMP GEN. GATING #3

CARD INPUT

0/0111
1/1000
2/1001
3/1010
4/1011

RAMP GEN. GATING #3

INPUT MINTERMS / MULTIPLE-FUNCTION

#	W	X	Y	/	F 1	F 2	F 3	F 4
1	0	0	0	/	0	1	1	1
2	0	0	1	/	1	0	0	0
3	0	1	0	/	1	0	0	1
4	0	1	1	/	1	0	1	0
5	1	0	0	/	1	0	1	1

RAMP GEN. GATING #3

PRIME IMPLICANTS / OUTPUT FUNCTIONS IMPLIED

	W	X	Y	/	F 1	F 2	F 3	F 4
1	0	0	0	/	0	1	1	1
2	0	1	0	/	1	0	0	1
3	0	1	1	/	1	0	1	0
4	1	0	0	/	1	0	1	1
5	0	-	0	/	-	-	-	1
6	-	0	0	/	-	-	1	1
7	0	-	1	/	1	-	-	-
8	0	1	-	/	1	-	-	-

```
SOLUTION NUMBER   1        RAMP GEN. GATING #3

I.  MINIMAL SUMS FROM MULTIPLE-OUTPUT PRIME IMPLICANTS
     WITH RESPECT TO SEVERAL CRITERIA

P-FUNCTION TERM EXPANDED =   1  2  3  4  7

F 1 = ¬WX¬Y + ¬WXY + W¬X¬Y + ¬WY
      010     011    100     0-1

F 2 = ¬W¬X¬Y
      000

F 3 = ¬W¬X¬Y + ¬WXY + W¬X¬Y
      000     011    100

F 4 = ¬W¬X¬Y + ¬WX¬Y + W¬X¬Y
      000     010    100


CROSS REFERENCE MINIMAL SUM MINTERMS WITH FUNCTION USAGE

MINTERMS       FUNCTION CROSS REFERENCE

010    1  4
011    1  3
100    1  3  4
0-1    1
000    2  3  4
-------------------------------------------------------------------------

II.  MINIMAL SUMS AFTER ADDITIONAL MINIMIZATION OF EACH OUTPUT FUNCTION


F 1 = W¬X¬Y + ¬WY + ¬WX
      100     0-1   01-

F 2 = ¬W¬X¬Y
      000

F 3 = ¬WXY + ¬X¬Y
      011    -00

F 4 = ¬W¬Y + ¬X¬Y
      0-0    -00


CROSS REFERENCE MINIMAL SUM MINTERMS WITH FUNCTION USAGE

MINTERMS       FUNCTION CROSS REFERENCE

100    1                    0-0    4
0-1    1
01-    1
000    2
011    3
-00    3  4
```

EX-OR TEST 3

PRIME IMPLICANTS / OUTPUT FUNCTIONS IMPLIED

	W X Y Z / F 1
1	0 0 0 1 / 1
2	0 0 1 0 / 1
3	0 1 0 0 / 1
4	1 0 0 0 / 1

PRIME IMPLICANT TABLE FOR EX-OR TEST 3

MINTERMS (LESS D.C.)	PRIME IMPLICANTS 1	2	3	4
0001	X			
0010		X		
0100			X	
1000				X

SOLUTION NUMBER 1 EX-OR TEST 3

I. MINIMAL SUMS FROM SINGLE-OUTPUT PRIME IMPLICANTS
WITH RESPECT TO SEVERAL CRITERIA

P-FUNCTION TERM EXPANDED = 1 2 3 4

F 1 = ¬W¬X¬YZ + ¬W¬XY¬Z + ¬WX¬Y¬Z + W¬X¬Y¬Z
0001 0010 0100 1000

PRIME IMPLICANT TABLE FOR EX-OR TEST 3

THIS TABLE FOR HAZARD FREE GATING

MINTERMS (LESS D.C.)	PRIME IMPLICANTS 1	2	3	4
0001	X			
0010		X		
0100			X	
1000				X

APPENDIX C

COMPUTER LISTING OF SYNCHRONOUS LOGIC DESIGN PROGRAM

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C        THIS PROGRAM IS FOR SYNTHESIS OF SYNCHRONOUS DIGITAL CIRCUITS
C     USING J/K FKIP FLOPS.   THE GENERATED CIRCUIT EQUATIONS ARE
C     MINIMIZED IN A COMBINATORIAL SENSE TO ARRIVE AT MINIMUM GATING.
C
C        INPUT TO THE PROGRAM IS BY PUNCHED CARDS AND HAS THE FOLLOWING
C     FORMAT.
C
C     CARD CARD      CONTENTS
C          COL.
C
C      1    6-45      ANY DESIRED PROBLEM TITLE
C      2    1-16      FROM NODE NAME OF STATE-FLOW FROM-TO
C           21-36     TO NODE NAME OF STATE-FLOW FROM-TO
C           41-80     A BOOLEAN EXPRESSION FOR THE INPUT SIGNALS THAT
C                      CAUSES THE FROM-TO TRANSITION (THIS FIELD MAY BE
C                      BLANK.)
C      3    REPEAT TYPE 2 CARDS UNTIL STATE-FLOW FROM-TO'S ALL DESCRIBED
C     ...
C      N    1-7       *STATES
C     N+1   1-20      THE INTERNAL FLIP FLOP STATE FOR THIS NODE. THIS
C                      NUMBER IS IN OCTAL.  (EX.  A¬BCD IN LITERAL FORM =
C                      1011 IN BINARY FORM = 13 IN OCTAL FORM.)
C           21-36     NODE NAME FOR THIS STATE-FLOW NODE.
C     N+2   REPEAT AS REQUIRED.
C     ...
C      M    1         *         (MARKS END OF A PROBLEM. ADDITIONAL PROBLEMS
C                                MAY BE STACKED FOLLOWING  *)
C
C        A MAXIMUM OF 32  FLIP FLOPS + INPUT VARIABLES ARE ALLOWED.
C
C        OUTPUT CONSISTS OF AN INPUT SUMMARY AND THE MINIMIZED  J,K
C     CIRCUIT EQUATIONS.
C
C
C              STATE DIAGRAM OF EXAMPLE
C   +-------------------->---------(V.W)------>---------+
C   |                                                   |
C   |  ---------->---------(V.¬W)-->-------+            |
C   |  |                                   |            |
C   |  |  +--->-----(¬V.W)--->-+           |            |
C   |  |  |  |(¬V¬W)|          |           |            |
C   |  |  |  V                 V           V            V
C   ****  ****  ****  ****  ****  ****  ****  ****
C   *N0*<---*N1*<---*N2*<---*N3*<---*N4*<---*N5*<---*N6*<---*N7*
C   ****  ****  ****  ****  ****  ****  ****  ****
```

```
FORTRAN IV G LEVEL  21                    MAIN                DATE = 73135        21/16/38

                C
                C      000     100     010     110     001     101     011     111   INTERNAL
                C                                                                    STATE
                C      0       4       2       6       1       5       3       7    (OCTAL)
                C
                C
                C
                C          INPUT CARDS FOR EXAMPLE  (CARD COLUMNS FOLLOW)
                C                1     1     2     2     3     3     4
                C     1    6     1     6     1     6     1     6     1
                C
                C          SAMPLE PROBLEM TITLE
                C     N0                      N1                      ¬V.¬W
                C     N0                      N3                      ¬V.W
                C     N0                      N5                      V.¬W
                C     N0                      N7                      V.W
                C     N7                      N6
                C     N6                      N5
                C     N5                      N4
                C     N4                      N3
                C     N3                      N2
                C     N2                      N1
                C     N1                      N0
                C     *STATES
                C     0                       N0
                C     4                       N1
                C     2                       N2
                C     6                       N3
                C     1                       N4
                C     5                       N5
                C     3                       N6
                C     7                       N7
                C     *
                C
                C           B. PERALTA  DPT 522      MARCH 73       49137-2
                C
                C
                C       PRIOR PROGRAMS    11384 - CUBE LOGIC
                C                         11539 - LOGICMIN
                C
0001                     LOGICAL LCOM
0002                     INTEGER  SIGLEN, KRDFRM(16), KRDTO(16), KRDSIG(40)
0003                     DIMENSION KARD(80)
0004                     EQUIVALENCE  (KARD(1),KRDFRM(1)), (KARD(21),KRDTO(1)),
                        A    (KARD(41),KRDSIG(1))
0005                     INTEGER OSTRNG(80), INSYMB(4,32), LENEQ(128)
0006                     INTEGER WORK(132)
0007                     DIMENSION NODNAM (4,128), NXFROM(128), NXTO(128), NXSIG(40,128)
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

 0008                 DIMENSION NODE(128),  JTERMO(128),JTERMD(128)
 0009                 DIMENSION  JFUNO(128),  JFUND(128),  IEQP(256),IEQOID(256)
 0010                 DIMENSION IUNASN(128)
 0011                 DIMENSION KTERMO(128),KTERMD(128),KFUNO(128),KFUND(128)
 0012                 DIMENSION TITLE(10)
 0013                 REAL*8  DATE,TIME
 0014                 DATA IBLNK/'    '/
 0015                 DATA ILEFT,IRIGHT /'(',')'/
 0016                 DATA IOR,INOT /'+','¬'/
 0017                 INTEGER IVAR(32)/'A','B','C','D','E','F','G','H','I','J','K','L',
                     1 'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','0','1',
                     2 '2','3','4','5'/
 0018                 CALL TIMEX (TIME,DATE)
 0019                 IIN=5
 0020                 IOUT=6
 0021                 MAXFTX=128
 0022                 MAXNOD=128
 0023                 MXTERM=128
 0024                 NAMLEN=16
 0025                 SIGLEN=40
 0026                 IEND=0
 0027               4 NXFTO=0
 0028                 NSYMB=0
 0029                 IERR=0
 0030                 DO 1 I=1,MAXNOD
 0031                 NODE(I)=-1
 0032               1 CONTINUE
                C
                C      START DATA INPUT. READ STATE-FLOW FROM TO'S WITH INPUT SIGNALS.
                C       BUILD NODE SYMBOL TABLE AS INPUT ACCEPTED SAVING INPUT SIGNALS.
                C
 0033                 READ (IIN,1150,END=990) TITLE
 0034              1150 FORMAT (5X,10A4)
 0035                 WRITE (IOUT,1151) TITLE, TIME, DATE
 0036              1151 FORMAT (1H1,10A4,'   TIME= ',A8,'   DATE= ',A8/)
 0037                 WRITE (IOUT,1190)
 0038              1190 FORMAT ('   INPUT DATA'/)
 0039                 5 NXFTO=NXFTO+1
 0040                 IF (NXFTO .LE. MAXFTX)  GO TO 7
 0041                 WRITE (IOUT,1195) MAXFTX
 0042              1195 FORMAT ('0*** STATE-FLOW FROM-TO''S EXCEED',I5)
 0043                 GO TO 999
                C
 0044                 7 READ (IIN,1200,END=999) KARD
 0045              1200 FORMAT (80A1)
                C              CHECK FOR '*' IN COLUMN 1
 0046                 IF (LCOM(KARD(1),'*',1) ) GO TO 70
 0047                 WRITE (IOUT,1205) NXFTO,KARD
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

0048          1205 FORMAT (1X,I5,'  FROM= ',20A1,'TO= ',20A1,'INPUT SIGNALS= ',40A1)
                   C
                   C              BUILD NODE NAME TABLE  (FROM STATE-FLOW NODE)
0049                    CALL ISYMBL (NAMLEN, KRDFRM ,NOS,OSTRNG,NSYMB,NODNAM,MAXNOD,IER)
0050                    IF (IER .EQ. 0 .AND. NOS.EQ.4) GO TO 20
0051                    WRITE (IOUT,1220)
0052          1220 FORMAT (' *** ERROR IN ''FROM'' STATE-FLOW NODE NAME')
0053                    IERR=1
0054                    NXFROM(NXFTO)=0
0055                    GO TO 21
                   C              PICK OFF NAME ADDRESS
0056             20 NXFROM(NXFTO) =OSTRNG(3)
                   C              PICK UP 'TO' NAME
0057             21 CALL ISYMBL (NAMLEN, KRDTO, NOS,OSTRNG,NSYMB,NODNAM,MAXNOD,IER)
0058                    IF (IER .EQ.0 .AND. NOS.EQ.4) GO TO 30
0059                    WRITE (IOUT,1230)
0060          1230 FORMAT (' *** ERROR IN ''TO'' STATE-FLOW NODE NAME')
0061                    IERR=1
0062                    NXTO(NXFTO)=0
0063                    GO TO 31
                   C              PICK OFF NAME ADDRESS
0064             30 NXTO(NXFTO) =OSTRNG(3)
0065             31 CALL MOVE ( NXSIG(1,NXFTO),KRDSIG,4*SIGLEN)
0066                    GO TO 5
                   C              CHECK FOR '*STATES' CARD
                   C
0067             70 NMAX=0
0068                    IF (.NOT.LCOM(KARD(1),'*   S   T   A   T   E   ',24)) GO TO 90
0069                    WRITE (IOUT,1772) KARD
0070          1772 FORMAT ('0',80A1/)
                   C              YES IT IS. READ STATE ASSIGNMENT CARDS
0071             75 READ (IIN,1200,END=89)  KARD
0072                    IF ( LCOM(KARD(1),'*',1) ) GO TO 90
0073                    WRITE (IOUT,1775) KARD
0074          1775 FORMAT (' ASSIGNED STATE(OCTAL)= ',20A1,' NODE NAME = ',60A1)
                   C              CONVERT STATE ASSIGNMENT IN OCTAL TO INTERNAL BINARY
0075                    NSTATE=0
0076                    DO 77 I=1,20
0077                    IE=21-I
0078                    IF (KARD(IE) .NE. IBLNK) GO TO 78
0079             77 CONTINUE
                   C              ERROR.  STATE ASSIGNMENT BLANK
0080                    WRITE (IOUT,1777)
0081          1777 FORMAT (' *** ERROR. STATE ASSIGNMENT BLANK')
0082                    IERR=1
0083                    GO TO 80
0084             78 DO 79 I=1,IE
0085             79 CALL MPUT (  NSTATE       , 3*(IE-I)+1, KARD(IE+1-I), 5, 3)
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

                C               CHECK NODE NAME
0086               80 NS=NSYMB
0087                  CALL ISYMBL (NAMLEN, KARD(21), NOS,OSTRNG,NSYMB,NODNAM,MAXNOD,IER)
0088                  IF (NS.EQ.NSYMB .AND. IER.EQ.0 .AND. NOS.EQ.4) GO TO 82
                C               ERROR IN NODE NAME
0089                  WRITE (IOUT,1780)
0090               1780 FORMAT (' *** ERROR IN NODE NAME')
0091                  IERR=1
0092                  NSYMB=NS
0093                  GO TO 75
                C
                C               SAVE STATE DATA
0094               82 N=OSTRNG(3)
0095                  NODE(N) = NSTATE
0096                  NMAX = MAXO(NMAX,NSTATE)
0097                  GO TO 75
                C
                C               CHECK IF ANY NODE NAMES HAVE UNASSIGNED STATES
0098               89 IEND=1
0099               90 NXFTO=NXFTO-1
0100                  DO 95 I=1,NSYMB
0101                  IF ( NODE(I) .NE. -1 ) GO TO 95
                C               ERROR
0102                  WRITE (IOUT,1191)   (NODNAM(J,I),J=1,4)
0103               1191 FORMAT (' *** ERROR. NO ASSIGNED STATE FOR NODE NAME = ',4A4)
0104                  IERR=1
0105               95 CONTINUE
                C
                C               DETERMINE NUMBER OF FLIP-FLOPS REQUIRED
0106                  NFF =  1.442695*ALOG(FLOAT(NMAX))  + 1.
                C
                C               ADD FLIP FLOP VARIABLE NAMES TO SYMBOL TABLE
                C
0107                  DO 97 I=1,NFF
0108                  INSYMB(1,I)=IVAR(I)
0109                  DO 97 J=2,4
0110               97 INSYMB(J,I)=IBLNK
                C
                C               FORM SYMBOL TABLE OF INPUT SIGNAL VARIABLES
                C
0111                  INP=0
0112                  DO 100 I=1,NXFTO
0113                  CALL ISYMBL ( SIGLEN,NXSIG(1,I),NOS,OSTRNG,INP,
                     A      INSYMB(1,NFF+1), 32-NFF, IER)
0114                  IF (IER.EQ.2)  IERR=1
0115                  LENEQ(I)=NOS
0116                  IF(NOS.EQ.0) GO TO 100
                C               SAVE PARTIALLY PROCESSED INPUT SIGNAL EQUATION
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

0117                  IF (NOS .GE. SIGLEN) WRITE (IOUT,1210) I, SIGLEN
0118                1210 FORMAT (' *** ERROR. INPUT SIGNAL EQUATION ON LINE',I3,' EXCEEDS',
                     A  I3,' AFTER ISYMBL PROCESSING')
0119                  NOS = MINO (NOS,SIGLEN)
0120                  CALL MOVE ( NXSIG(1,I), OSTRNG, 4*NOS)
0121                100 CONTINUE
                    C
0122                  WRITE (IOUT,1221) NFF,NXFTO,NSYMB, INP
0123                1221 FORMAT ('1INPUT SUMMARY'/
                     A'0',I5,' J/K FLIP-FLOPS REQUIRED'/
                     B'0',I5,' INPUT FROM-TO CARDS'/
                     C'0',I5,' STATE-FLOW NODES'/
                     D'0',I5,' INPUT SIGNAL VARIABLES'/
                     E )
                    C
                    C             PLACE (  ) AROUND VARIABLE NAMES GREATER THAN 1 CHAR
                    C
0124                  DO 120 I=1,INP
0125                  CALL CVA4A1 (INSYMB(1,I+NFF),1,4,WORK(2),1,NAMLEN,IER)
0126                  DO 125 J=1,NAMLEN
0127                  JJ=NAMLEN+2-J
0128                  IF (WORK(JJ) .NE. IBLNK)  GO TO 126
0129                125 CONTINUE
0130                126 IF (JJ .LE. 2)  GO TO 120
                    C             NEED PARANS
0131                  WORK(1) = ILEFT
0132                  WORK(JJ+1) = IRIGHT
0133                  CALL CVA1A4 (WORK,1,NAMLEN,INSYMB(1,I+NFF),1,4,IER)
0134                120 CONTINUE
                    C
                    C             PRINT STATE-FLOW NODES
                    C
0135                  WRITE (IOUT,1319)
0136                1319 FORMAT (///'0      NODE NAME          ASSIGNED STATE (LITERAL FORM)
                     A' /)
0137                  DO 130 I=1,NSYMB
                    C               CONVERT NODE STATE TO 0,1,- FORM
0138                  CALL TERMOT (NFF     , NODE(I),0,WORK)
                    C               CONVERT NODE STATE TO LITERAL FORM
0139                  WORK(NFF+1) = IBLNK
0140                  WORK(NFF+2) = ILEFT
0141                  NOS1=NFF+2
0142                  DO 135 J=1,NFF
0143                  CALL PBTST (4,NODE(I), 31-NFF     +J,IVAL)
0144                  IF (IVAL .EQ. 1)  GO TO 136
                    C             COMPLEMENTED VARIABLE
0145                  NOS1=NOS1+1
0146                  WORK(NOS1)=INOT
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

                C                UNCOMPLEMENTED VARIABLE
0147              136 NOS1=NOS1+1
0148                  WORK(NOS1) = IVAR(J)
0149              135 CONTINUE
0150                  NOS1 = NOS1+1
0151                  WORK(NOS1)=IRIGHT
0152                  WRITE (IOUT,1320) I,(NODNAM(J,I),J=1,4), (WORK(J),J=1,NOS1)
0153              130 CONTINUE
0154             1320 FORMAT (1X,I4,2X,4A4,2X,100A1)
                C
                C               PRINT INPUT VARIABLES
                C
0155                  IF (INP .GT. 0)
                     *WRITE (IOUT,1340) (((INSYMB(J,I+NFF),J=1,4),I=1,INP)
0156             1340 FORMAT (///'0INPUT VARIABLES'// (1X,4A4))
                C
0157                  IF (IERR .GT. 0)  GO TO 999
                C
                C                FIND UNASSIGNED STATES
                C
0158                  NS=2**NFF
0159                  NUN=0
0160                  DO 150 I=1,NS
0161                  NA=I-1
0162                  DO 152 J=1,NSYMB
0163                  IF ( NODE(J) .EQ. NA)  GO TO 150
0164              152 CONTINUE
                C                FOUND ONE
0165                  IF (NUN .LT. MXTERM) GO TO 153
0166                  WRITE (IOUT,1152)
0167             1152 FORMAT ('0***ERROR.  UNASSIGNED STATES EXCESSIVE. PROCESSING BYPAS
                    ASED')
0168                  NUN=0
0169                  GO TO 155
                C
0170              153 NUN=NUN+1
0171                  IUNASN(NUN)=NA
0172              150 CONTINUE
                C
0173              155 CONTINUE
                C
                C
                C               SHIFT FF STATE OVER TO ACCOMMODATE INPUT VARIABLES
                C
0174                  IMULT=2**INP
0175                  DO 160 I=1,NSYMB
0176                  NODE(I) = NODE(I)*IMULT
0177              160 CONTINUE
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

0178              IF (NUN .EQ. 0) GO TO 162
0179              DO 161 I=1,NUN
0180            161 IUNASN(I) = IUNASN(I)*IMULT
                C
                C                 PROCESS INPUT SIGNAL EQUATIONS TO SUM OF PRODUCTS FORM
0181            162 DO 1900 I=1,NXFTO
0182              CALL BOOLEQ (  LENEQ(I),  NXSIG(1,I), WORK )
0183           1900 CONTINUE
                C
                C
                C                 FORM  SET  /  RESET EQUATIONS FROM  STATE-FLOW FROM-TO'S
0184              IBITS = NFF + INP
0185              IDSET=0
0186              IF (INP .LE. 0)  GO TO 2001
0187              DO 2000 I=1,INP
0188           2000 CALL PBTST (1, IDSET,  32-I)
                C
                C
0189           2001 DO 2200 N=1,NFF
0190              NN=31-IBITS+N
                C
                C               THE FOLLOWING LOOP INSERTED FOR ADDITIONAL ASSIGNMENT OF
                C                 DON'T CARE STATES TO ALLOW FURTHER MINIMIZATION
                C  "DO 2200" INHIBITED TEMPORARILY WITH  COL. 1 "C".
                C     DO 2200 NES=1,2
                C                 FORM FIRST SET  THEN  RESET EQUATION
0191              DO 2199 NEQ=1,2
                C
                C                 DO 2100 LOOP SEARCHS FOR TRANSITIONS OF FF IN QUESTION
                C
0192              NTRM=0
0193              DO 2100 KK=1,NXFTO
                C
                C                 CHECK IF FF IS CHANGING
0194           2110 CALL PBTST ( 4, NODE(NXFROM(KK)),  NN, IVAL)
0195              CALL PBTST ( 4, NODE(NXTO(KK)), NN, JVAL)
                C               CHECK FOR REDUNDANT SET / RESET TERMS (SAME STATE MAINTAIND)
0196              IF (IVAL .EQ. JVAL) GO TO 2100
                C     GO TO 2100
                C      IF (IVAL .EQ. MOD(NEQ,2))  GO TO 2120
                C                 CHECK FOR SET / RESET TERM
0197           2111 IF  ( 2*IVAL + JVAL .NE. NEQ ) GO TO 2100
                C
                C                 ADD TERM TO  SET / RESET EQUATION
                C
0198           2120 J=0
0199           2121 NTRM=NTRM+1
0200              JTERMO(NTRM) = NODE(NXFROM(KK))
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135        21/16/38

0201                JTERMD(NTRM) = IDSET
                C             ADD DASH INTO THIS FF'S BIT POSITION TO ELIMINATE
                C               FLIP FLOP REDUNDENCIES
0202                CALL PBTST (3,JTERMO(NTRM),NN,IVAL)
0203                CALL PBTST (1,JTERMD(NTRM),NN,IVAL)
0204                JFUNO(NTRM)=1
0205                JFUND(NTRM)=0
                C               INCLUDE INPUT VARIABLES IN MINTERM  (INTERNAL 0,1,-)
0206                2122 J=J+1
0207                IF (J .GT. LENEQ(KK))  GO TO 2100
                C           IF  OR ENCOUNTERED IN INPUT SIGNAL EQUATION.  ADD ANOTHER TERM
0208                IF ( LCOM(NXSIG(J,KK), '+', 1) ) GO TO 2121
                C               CHECK FOR NOT '¬'
0209                IF ( .NOT. LCOM(NXSIG(J,KK), '¬', 1) )  GO TO 2126
                C               COMPLEMENTED VARIABLE
0210                J=J+1
0211                IF (  NXSIG(J,KK) .GT. 10000) GO TO 2130
0212                NB=31-INP+NXSIG(J,KK)
0213                CALL PBTST ( 3, JTERMO(NTRM), NB, IVAL)
0214                CALL PBTST ( 3, JTERMD(NTRM), NB, IVAL)
0215                GO TO 2130
                C             UNCOMPLEMENTED VARIABLE
0216                2126 IF ( NXSIG(J,KK) .GT. 10000) GO TO 2130
0217                NB=31-INP+NXSIG(J,KK)
0218                CALL PBTST ( 1, JTERMO(NTRM), NB, IVAL)
0219                CALL PBTST ( 3, JTERMD(NTRM), NB, IVAL)
                C
0220                2130 GO TO 2122
0221                2100 CONTINUE
                C
                C               PRINT TERMS GENERATED
                C
0222                WRITE (6,8010)  N, NEQ
0223                8010 FORMAT ('0 J/K FLIP FLOP ',I3,'   EQUATION ',I2 )
0224                NB = NFF+INP
0225                DO 8000 KQ = 1,NTRM
0226                CALL  TERMOT(  NB, JTERMO(KQ), JTERMD(KQ), KARD)
0227                WRITE (6,8005)  (KARD(KN),KN=1,NB)
0228                8005 FORMAT ( 5X, 80A1)
0229                8000 CONTINUE
                C
                C             SUBROUTINE EXMNDC EXPANDS MINTERM DON'T CARES INTO
                C               APPROPRIATE 0,1 MINTERMS
                C
0230                CALL EXMNDC (IBITS,NTRM,MXTERM,JTERMO,JTERMD,JFUNO,JFUND,IER)
0231                IF (IER .GE.2)  GO TO 999
                C
                C               INCLUDE ADDITIONAL UNASSIGNED STATES IF NES=2
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

            C
 0232             IF (NES .NE. 2)  GO TO 540
 0233             NT=NTRM+1
 0234             IDS=IDSET
 0235             CALL PBTST (1,IDS,NN,IVAL)
 0236             CALL ADTRME (IBITS,NFF,NTRM,MXTERM,JTERMO,JTERMD,NUN,IUNASN,
                 A     IDS   ,IER)
 0237             IF (IER .NE. 0)  GO TO 999
            C                FILL IN FUNCTIONAL ARRAYS FOR MINTERMS JUST ADDED
 0238             IF (NTRM .LT. NT)  GO TO 2200
 0239             DO 530 KQ=NT,NTRM
 0240             JFUNO(KQ)=1
 0241           530 JFUND(KQ)=0
            C                EXPAND MINTERM DON'T CARES
 0242             CALL EXMNDC (IBITS,NTRM,MXTERM,JTERMO,JTERMD,JFUNO,JFUND,IER)
 0243             IF (IER .GE. 2)  GO TO 999
            C
            C
            C
            C     FIND PRIME IMPLICANTS FOR EACH FUNCTION  (IE, MINIMIZE TERMS FOR
            C      FOR SINGLE OUTPUT FUNCTION )
            C
 0244           540 CALL PRIMIM (IBITS,NTRM,JTERMO,JTERMD,JFUNO,JFUND
                A ,NPRIM, KTERMO,KTERMD,KFUNO,KFUND,IRET)
            C
            C                PRINT MINIMIZED FUNCTION
 0245             DO 542 KQ=1,256
 0246             IEQP(KQ) = IBLNK
 0247           542 IEQOID(KQ)=IBLNK
 0248             NOS1 = 0
 0249             DO 549 KS = 1,NPRIM
 0250             KSW=0
            C        CHECK IF THIS SINGLE OUTPUT PRIME IMPLICANT SUBSUMES ANY OF THE
            C        FOLLOWING PRIME IMPLICANTS.  IF IT DOES, DO NOT INCLUDE IT IN
            C        THE EQUATION STRING.
 0251             KT=KS+1
 0252             IF (KT .GT. NPRIM )  GO TO 5461
 0253             DO 546 KR=KT,NPRIM
 0254             CALL SUBSUM (IBITS,KTERMO(KS),KTERMD(KS),KTERMO(KR),KTERMD(KR),
                 A  IRET)
 0255             IF (IRET .EQ. 1)  GO TO 549
 0256           546  CONTINUE
 0257          5461 CONTINUE
            C
 0258             NS=NOS1+1
 0259             DO 548 L=1,IBITS
 0260             LL=31-IBITS+L
 0261             CALL PBTST (4,KTERMD(KS),LL,IVAL)
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 73135          21/16/38

0262                IF (IVAL .EQ. 1)  GO TO 548
0263                CALL PBTST (4,KTERMO(KS),LL,IVAL)
0264                IF (IVAL .EQ. 1)  GO TO 547
                C               COMPLEMENTED VARIABLE
0265                NOS1 = NOS1+1
0266                IEQP(NOS1) = INOT
                C               UNCOMPLEMENTED VARIABLE
0267              547 NOS1=NOS1+1
0268                CALL CVA4A1 ( INSYMB(1,L),1,4, IEQP(NOS1), 1,NAMLEN , IER)
                C               SUPPRESS TRAILING BLANKS
0269                KSW=1
0270                NOS1=NOS1+NAMLEN-1
0271                DO 5471  KQ=1,NAMLEN
0272                IF ( IEQP(NOS1) .NE. IBLNK)  GO TO 548
0273             5471 NOS1=NOS1-1
0274              548 CONTINUE
                C               IF TERM WAS ALL '-', FILL IN WITH A '1'
0275                IF (KSW.EQ.0)  CALL MOVE (IEQP(NOS1+1),'1',1)
                C          FILL IN 0,1,- LINE OF EQUIVALENTS
0276                CALL TERMOT (IBITS,KTERMO(KS),KTERMO(KS),IEQOID(NS))
                C               PUT IN ' + '
0277                NOS1=MAXO(NOS1+2,NS+IBITS)
0278                IEQP(NOS1   ) = IOR
0279                NOS1=NOS1+1
0280              549 CONTINUE
                C
                C               FORM EQUATION IDENTIFIER
0281                ID = IBLNK
0282                CALL MPUT (ID,24,IVAR(N),0,8)
0283                IF (NEQ .EQ. 1)  CALL MPUT (ID,16,'J',0,8)
0284                IF (NEQ .EQ. 2)  CALL MPUT (ID,16,'K',0,8)
                C               PRINT EQUATION
0285                CALL JKPRNT (ID,NOS1-2,IEQP,IEQOID)
0286             2199 CONTINUE
0287             2200 CONTINUE
0288                IF (IEND) 4,4,990
                C
                C               RUN ABORTED.  PRINT TERMINATION MESSAGE.
                C
0289              999 WRITE (IOUT,1999)
0290             1999 FORMAT ('0 EXECUTION TERMINATED BECAUSE OF ERROR')
0291              990 STOP
0292                END
```

```
FORTRAN IV G LEVEL  21                ADTRME              DATE = 73135          21/16/38

0001                  SUBROUTINE ADTRME(IBITS,NFF,NTERM,MXTERM,JTERMO,JTERMD, NUN
                     A ,IUNASN,  IDSET,IRET)
                C
                C        THIS SUBROUTINE ADDS UNSPECIFIED MINTERMS TO THE GIVEN
                C     INPUT LIST.  AN EXHAUSTIVE APPROACH IS USED,  THE CHECK MADE
                C     IS THAT AN ADDED TERM MUST BE ADJACENT TO AN EXISTING TERM.
                C     PRESENTLY ONLY INTERNAL STATES ARE CONSIDERED.
                C
0002                  DIMENSION TEMP(10)
0003                  DIMENSION  JTERMO(1),JTERMD(1), IUNASN(1)
0004                  IRET=0
0005                  K1=1
0006                5 IANY=0
0007                  K2=NTERM+1
                C
                C               START LOOP THROUGH UNASSIGNED STATES
0008                  DO  50 I=1,NUN
                C               CHECK IF ADJACENT TO ANY EXISTING TERM
0009                  NT=NTERM
0010                  DO 40 J=K1,NT
0011                  M=0
0012                  DO 30 K=1,NFF
0013                  NN=31-IBITS+K
0014                  CALL PBTST (4,JTERMD(J),NN, IVAL)
0015                  IF ( IVAL .EQ. 1) GO TO 30
0016                  CALL PBTST (4,IDSET,NN,IVAL)
0017                  IF (IVAL .EQ. 1)  GO TO 30
0018                  CALL PBTST (4,JTERMO(J),NN, IVAL)
0019                  CALL PBTST (4,IUNASN(I),NN, JVAL)
0020                  IF ( IVAL .EQ. JVAL) GO TO 30
0021                  M=M+1
0022                  IF ( M .GT. 1)  GO TO 40
0023               30 CONTINUE
0024                  IF (M .LT. 1)  GO TO 40
                C               SEARCH LIST FOR LIKE TERMS.  DO NOT ADD IF ONE FOUND.
0025                  JO=IUNASN(I)
0026                  CALL PBTST(11,JO,IDSET,IVAL)
0027                  DO 32 KQ=1,NTERM
0028                  IF (JTERMO(KQ).EQ.JO .AND. JTERMD(KQ).EQ.IDSET)  GO TO 40
0029               32 CONTINUE
                C               ADD TERM
0030                  IF ( NTERM .LT. MXTERM) GO TO 35
0031                  WRITE (6,100)  MXTERM
0032              100 FORMAT ('0***ERROR.  MINTERM LIST EXCEEDS',I5)
0033                  IRET=1
0034                  RETURN
0035               35 IANY=1
0036                  NTERM=NTERM+1
-------------------------------------------------------------------------------
0037                  JTERMO(NTERM) = JO
0038                  JTERMD(NTERM) = IDSET
0039                  CALL TERMOT (IBITS,JTERMO(NTERM),JTERMD(NTERM),TEMP)
0040                  WRITE (6,1000) (TEMP(KQ),KQ=1,IBITS)
0041             1000 FORMAT (' UNASSIGNED TERM ADDED = ',10A1)
0042                  GO TO 50
                C
0043               40 CONTINUE
                C
0044               50 CONTINUE
0045                  K1=K2
0046                  IF (IANY .EQ. 1) GO TO 5
0047                  RETURN
0048                  END
```

APPENDIX D

COMPUTER LISTING OF CUBE LOGIC OR ASYNCHRONOUS DESIGN PROGRAM

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C     SQLGSYN  -  SEQUENTIAL LOGIC SYNTHESIS PROGRAM
C
C     REFERENCES
C     1. 'DESIGN OF SEQUENTIAL SWITCHING CIRCUITS WITH THE CUBE LOGIC
C        TECHNIQUE', S.L. SENDERS AND J.R. LUCCHESI (IBM), COMPUTER
C        DESIGN, APRIL 1971, PP.59-64
C
C     THE FOLLOWING ARRAYS CONTAIN DATA ABOUT NODES ON THE STATE FLOW
C      DIAGRAM AND THE INTERNAL CUBE NODES.
C
C     NODE     - CONTAINS CUBE VERTEX NUMBER FOR NODE IN NODNAM ARRAY
C     NODNAM   - CONTAINS ALL UNIQUE NODE NAMES
C     NODADR   - EACH ELEMENT REPRESENTS ONE VERTEX OF THE CUBE AND
C                CONTAINS A POINTER TO THE NODE, NODNAM, AND NODUSE
C                ARRAYS.
C     JODE     - A DUPLICATE OF ARRAY 'NODE'.  USED AS A WORK AREA.
C     JODADR   - A DUPLICATE OF ARRAY 'NODADR'. USED AS A WORK AREA.
C     NODUSE   - COUNT OF THE NUMBER OF TIMES NODNAM APPEARS IN THE STATE
C                -FLOW FROM-TO LIST.
C     NODPRI   - PRIORITY OF NODE WITH RESPECT TO MAPPING (HIGHER NUMBERS
C                GET MAPPED FIRST)
C
C     THE FOLLOWING ARRAYS CONTAIN DATA ON FROM-TO RELATIONSHIPS OF
C      THE STATE-FLOW DIAGRAM
C
C     NXFROM   - FROM NODE OF STATE-FLOW FROM-TO'S
C     NXTO     - TO NODE OF STATE-FLOW FROM-TO'S
C     NXSIG    - SIGNALS PRESENT FOR TRANSITION OF FROM-TO'S
C                (MAY BE BOOLEAN EXPRESSION)
C     NXSTAT   - STATUS OF CORRESPONDING STATE-FLOW FROM-TO
C                =0 NOT YET MAPPED ONTO CUBE
C                =1 MAPPED ONTO CUBE
C                =2 USED IN PRINTING EQUATIONS
C
C     THE FOLLOWING ARRAYS CONTAIN DATA ON THE INTERNAL CUBE MAPPING
C
C     NDFROM   - FROM CUBE VERTEX NUMBER FOR THIS FROM-TO
C     NDTO     - TO CUBE VERTEX NUMBER FOR THIS FROM-TO
C                THE CONTENTS OF NDFROM AND NDTO CONTAIN POINTERS TO THE
C                NODADR ARRAY WHICH IN TURN POINTS TO STATE-FLOW NODES OR
C                DUMMY NODES.
C
C     THE FOLLOWING ARRAYS CONTAIN DATA ABOUT INPUT ASSIGNED STATES
C
C     NSTATE   - CONTAINS THE STATE ASSIGNMENTS
C     NSTADR   - CONTAINS ADDRESS OF THE NODE NAME FOR EACH ASSIGNMENT
C     NSTAT    - NOT AN ARRAY BUT CONTAINS COUNT OF NUMBER OF ASSIGNED
C                STATES
C
C
C     NFF      - NUMBER OF STORAGE ELEMENTS REQUIRED
C     MAXNOD   - MAXIMUM NUMBER OF STATE-FLOW + DUMMY NODES ALLOWED
```

```
C      MAXFTX  - MAXIMUM NUMBER OF STATE-FLOW FROM-TO'S
C      MAXFTO  - MAXIMUM NUMBER OF INTERNAL FROM-TO'S
C      NCOUNT  - NUMBER OF STATE-FLOW + DUMMY NODES
C      NXFTO   - NUMBER OF STATE-FLOW FROM-TO'S
C      NOFTO   - NUMBER OF INTERNAL FROM-TO'S
C      NODDUM  - FIRST DUMMY NODE IN NODE ARRAYS
C
       COMMON /COMMAP/   NFF,      NXFTO,    NOFTO,    NCOUNT,   MAXFTX,
      1                  MAXNOD,   MAXFTO,   NODDUM,   ITRACE,
      2   JODE(128),     JODADR(128),
      4   NODE(128),     NODNAM(16,128),NODADR(128),   NODPRI(128),
      5   NODUSE(128),
      6   NXFROM(64),    NXTO(64),      NXSIG(40,64),  NXSTAT(64),
      7   NOFROM(128),   NOTO(128)
       DIMENSION ITAGA(128)
       DIMENSION NSTATE(128), NSTADR(128)
       LOGICAL $STATE
       INTEGER IVAR(32)/'A','B','C','D','E','F','G','H','I','J','K','L',
      1 'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','0','1',
      2 '2','3','4','5'/
       INTEGER IDUM(6)/'D','U','M','M','Y',' '/
       INTEGER SIGLEN,KRDTO(16),KRDSIG(40)
C      LOOPLN  - LENGTH OF LOOPS OR SEGMENTS
C      NXLOOP  - HOLDS CLOSED NON-SELF-INTERSECTING LOOPS OF STATE-FLOW
C                DIAGRAM
C      LOOPRI  - HOLDS MAPPING PRIORITY OF LOOP OR SEGMENT
       DIMENSION NXLOOP(24,32),LOOPRI(32),LOOPLN(32),LOPTAG(32)
       DIMENSION KARD(80)
       DIMENSION ISET(240)
       EQUIVALENCE (KARD(21),KRDTO(1)), (KARD(41),KRDSIG(1))
       DATA IAST /'*'/
       DATA IBLNK,IOR,INOT,ILEFT,IRIGHT /' ','+','¬','(',')'/
       KSETR(KSET) = MOD(KSET-1,240) + 1
C
C      IO UNIT NUMBERS
C
       IIN = 5
       IOUT = 6
       IFND=0
C
C      MAXNFF  - MAXIMUM NUMBER OF STORAGE ELEMENTS TO BE CONSIDERED
C      MAXTRY  - MAXIMUM NUMBER OF TRIALS AT MAPPING AT PRESENT NUMBER
C                  OF STORAGE ELEMENTS
C      MSUCES  - NUMBER OF SUCCESSFUL SOLUTIONS DESIRED
C      ITRACE  - TRACE OF MAPPING
C                =0 FINAL PRINTOUTS ONLY
C                =1 PRINTOUTS AFTER EACH SUCCESSFUL & UNSUCCESSFUL
C                     MAPPING IN THE MAIN PROGRAM
C                =2 SAME AS =1 BUT INCLUDES UNSUCCESSFUL ATTEMPTS
C                     AT LOWER LEVELS
C
    1 READ (IIN,1180,END=999)  MAXNFF,MAXTRY,MSUCES,ITRACE
 1180 FORMAT (4I5)
      WRITE (IOUT,1170)  MAXNFF,MAXTRY,MSUCES,ITRACE
 1170 FORMAT ('1  SQLGSYN - SEQUENTIAL LOGIC CIRCUIT SYNTHESES'//
     1 1X,I5,' - MAXIMUM NUMBER OF STORAGE ELEMENTS TO BE USED FOR SOLUT
```

```
      IF (NCOUNT .EQ. 0)  GO TO 20
      DO 15 I = 1,NCOUNT
      IF (ICOMPR (NODNAM(1,I),KARD(1),4*NAMLEN)) 15,25,15
   15 CONTINUE
C               NEW NODE NAME.  CHECK IF ROOM FOR IT
   20 IF (NCOUNT .LT. MAXNOD)  GO TO 22
      WRITE (IOUT,1210) MAXNOD
 1210 FORMAT ('0*** NUMBER OF STATE-FLOW NODES EXCEED ',I5)
      GO TO 999
C
C               CHECK FOR ALL BLANKS FOR FROM NODE NAME
   22 DO 23 K = 1,NAMLEN
      IF (KARD(K) .NE. IBLNK)  GO TO 24
   23 CONTINUE
      WRITE (IOUT,1220)
 1220 FORMAT ('0*** FROM NODE NAME ALL BLANKS. TRY AGAIN.')
      IERR = 1
C               SAVE NODE NAME
   24 NCOUNT = NCOUNT + 1
      I = NCOUNT
      DO 21 K= 1,NAMLEN
   21 NODNAM(K,I) = KARD(K)
C
   25 NODUSE(I) = NODUSE(I) + 1
      NXFROM(NXFTO) = I
C
C     CHECK IF STATE-FLOW TO NODE NAME IS NEW
C
      DO 30 I=1,NCOUNT
      IF (ICOMPR(NODNAM(1,I),KRDTO,4*NAMLEN)) 30,45,30
   30 CONTINUE
C               NEW NODE NAME.  CHECK IF ROOM FOR IT.
      IF (NCOUNT .LT. MAXNOD)  GO TO 35
      WRITE (IOUT,1210) MAXNOD
      GO TO 999
C               CHECK FOR ALL BLANKS IN TO NODE NAME
   35 DO 40 K=1,NAMLEN
      IF (KRDTO(K) .NE. IBLNK)  GO TO 42
   40 CONTINUE
      WRITE (IOUT,1230)
 1230 FORMAT ('0*** TO NODE NAME ALL BLANKS.  TRY AGAIN.')
      IERR = 1
   42 NCOUNT = NCOUNT + 1
      I = NCOUNT
      DO 44 K = 1,NAMLEN
   44 NODNAM(K,I) = KRDTO(K)
C               BUMP USAGE COUNT AND SET POINTER
   45 NODUSE(I) = NODUSE(I) + 1
      NXTO(NXFTO) = I
C
C     SAVE EXTERNAL SIGNAL(S) OR BOOLEAN EXPRESSION.
C
      DO 50 K = 1,SIGLEN
   50 NXSIG(K,NXFTO) = KRDSIG(K)
      GO TO 5
C
```

```
     2ION'//
     3 1X,I5,' - MAXIMUM VALUE OF TRIALS FOR EACH NUMBER OF STORAGE ELEM
     4ENTS'//
     5 1X,I5,' - NUMBER OF SOLUTIONS DESIRED'//
     6 1X,I5,' - TRACE PARAMETER')
C
      NAMLEN = 16
      SIGLEN = 40
      IERR = 0
      MAXNOD = 128
      MAXFTX = 64
      MAXFTO = 128
      NCOUNT = 0
      NXFTO = 0
      NOFTO = 0
      $STATE=.FALSE.
C
C     INITIALIZE ARRAYS
C
      DO 4 I = 1,MAXNOD
      NODE(I) = -1
      NODADR(I) = 0
      NODUSE(I) = 0
      NODPRI(I) = 0
      DO 4 J = 1,NAMLEN
    4 NODNAM(J,I) = IBLNK
      DO 2 I =1,MAXFTX
      NXFROM(I) = 0
      NXTO (I) = 0
      NXSTAT(I) = 0
      DO 2 J=1,SIGLEN
    2 NXSIG(J,I) = IBLNK
      DO 3 I=1,MAXFTO
      NOFROM(I) = 0
    3 NOTO(I) = 0
C
C     READ STATE-FLOW FROM-TO LIST
C      PICK OUT ALL UNIQUE STATE-FLOW NODE NAMES AND COUNT HOW MANY TIME
C      S EACH APPEARS.
C
      WRITE (IOUT,1190)
 1190 FORMAT ('1   INPUT DATA'/)
    5 NXFTO = NXFTO + 1
      IF (NXFTO .LE. MAXFTX)  GO TO 7
      WRITE (IOUT,1195) MAXFTX
 1195 FORMAT ('0*** STATE-FLOW FROM-TO''S EXCEED',I5)
      GO TO 999
    7 READ (IIN,1200,END=89) KARD
 1200 FORMAT (80A1)
C             CHECK FOR '*'
      IF (KARD(1) .EQ. IAST)  GO TO 70
      WRITE (IOUT,1205) NXFTO,KARD
 1205 FORMAT (1X,I5,'  FROM= ',20A1,'TO= ',20A1,'SIGNAL(S)= ',40A1)
C
C     CHECK IF FROM STATE-FLOW NODE IS NEW
C
```

```
C     CHECK FOR '*STATES' CARD
C
   70 IF (ICOMPR(KARD(1), '*   S   T   A   T   E   ',24) ) 90,72,90
C             YES IT IS.  READ STATE ASSIGNMENT CARDS
   72 $STATE = .TRUE.
      NSTAT=0
      WRITE (IOUT,1772) KARD
   75 READ (IIN,1200,END=89)  KARD
 1772 FORMAT ('0',80A1/)
      IF (KARD(1) .EQ. IAST)  GO TO 90
      WRITE (IOUT,1775)  KARD
 1775 FORMAT (' ASSIGNED STATE(OCTAL) = ',20A1,'  NODE NAME = ',60A1)
C             CONVERT STATE ASSIGNMENT (IN OCTAL) TO INTERNAL BINARY
      NSTAT=NSTAT+1
      NSTATE(NSTAT)=0
      DO 77 I=1,20
      IE = 21-I
      IF (KARD(IE) .NE. IBLNK)  GO TO 78
   77 CONTINUE
C             ERROR.  STATE ASSIGNMENT BLANK.
      WRITE (IOUT,1777)
 1777 FORMAT ('0*** ERROR.  STATE ASSIGNMENT BLANK')
      IERR=1
      GO TO 80
C
   78 DO 79 I=1,IE
   79 CALL MPUT ( NSTATE(NSTAT), 32-3*I, KARD(IE+1-I), 5, 3)
C             CHECK IF NODE NAME NEW
   80 DO 82 I=1,NCOUNT
      IF (ICOMPR (NODNAM(1,I),KARD(21), 4*NAMLEN )) 82,85,82
   82 CONTINUE
C             ERROR.  NEW NAME
      WRITE (IOUT,1782)
 1782 FORMAT ('0*** ERROR. NODE NAME ON ABOVE CARD NOT ON ANY FROM-TO CA
     ARD'/)
      IEPR=1
      GO TO 75
C
C             SAVE ASSIGNED STATE DATA
   85 NSTADR(NSTAT) = I
      GO TO 75
C
C     DETERMINE MINIMUM NUMBER OF STORAGE ELEMENTS
C     NEEDED FOR NUMBER OF STATE-FLOW NODES.
C
   89 IEND=1
   90 NFF = 32
      IF (NCOUNT .EQ. 0 .OR. NXFTO .EQ. 1)  GO TO 900
      DO 91 K=1,32
      CALL PBTST (4,NCOUNT-1,K-1,IVAL)
      IF (IVAL .EQ. 1)  GO TO 92
   91 NFF = NFF - 1
   92 NXFTO = NXFTO - 1
      NODDUM = NCOUNT+1
C
C             FILL IN DUMMY NODE NAMES
```

```
      IF (NODDUM .GT. MAXNOD)  GO TO 101
      N = 1
      DO 100 I=NODDUM,MAXNOD
      DO 105 J = 1,6
  105 NODNAM(J,I) = IDUM(J)
      CALL CNVIA1 (NODNAM(1,I),6,8,N,IER)
  100 N = N+ 1
  101 MAXNOD = 2**NFF
      CALL LOGIC (NFF)
      WRITE (IOUT,1240) NXFTO,NCOUNT,NFF
 1240 FORMAT ('1   SQLGSYN  -  SEQUENTIAL LOGIC CIRCUIT SYNTHESIS'/
     1  '0 STATE-FLOW DIAGRAM FROM-TO RECORDS= ',I5/
     2 '0 STATE-FLOW DIAGRAM NODES = ',I5/
     3 '0 MINIMUM NUMBER OF STORAGE ELEMENTS= ',I5)
      WRITE (IOUT,1250) (I,(NODNAM(K,NXFROM(I)),K=1,NAMLEN),NXFROM(I),
     1      (NODNAM(K,NXTO(I)),K=1,NAMLEN),NXTO(I),
     2   (NXSIG(J,I),J=1,SIGLEN),I=1,NXFTO)
 1250 FORMAT ('1 STORED STATE-FLOW FROM-TO LIST'/
     1    'ONO.   FROM',22X,'TO',24X,'SIGNAL'//
     2    (1X,I4,2X,16A1,' (',I3,')',4X,16A1,' (',I3,')',4X,40A1))
C
C     FOR FIRST PASS ASSIGN PRIORITY BASED ON NODUSE ARRAY
C
      CALL TAGSRT (NCOUNT, NODUSE, ITAGA)
      DO 110 J = 1,NCOUNT
  110 NODPRI(ITAGA(J)) = J
C
C     PRINT LIST OF STATE-FLOW NODES
C
      WRITE (IOUT,1260)
 1260 FORMAT ('1  STATE-FLOW NODES'//'  NO.  NAME',12X,'  CNT  PRI'/)
      WRITE (IOUT,1270) (I,(NODNAM(K,I),K=1,NAMLEN),NODUSE(I),NODPRI(I),
     1    I=1,NCOUNT)
 1270 FORMAT (1X,I4,2X,16A1,2I5)
      IF (IERR .EQ. 1)  GO TO 900
C
C     DETERMINE LOOPS AND SEGMENTS IN STATE FLOW DIAGRAM
C
C     NLOOP   - COUNT OF NUMBER OF LOOPS IN NXLOOP
      NLOOP = 0
C             USE HIGHEST PRIORITY NODE AS A CONVENIENT STARTING POINT
      KODE = ITAGA(NCOUNT)
C
C             SEARCH STATE FLOW FROM'S FOR THIS NODE
  145 DO 150 I = 1,NXFTO
      IF (NXFROM(I) .EQ. KODE .AND. NXSTAT(I) .EQ. 0)  GO TO 160
  150 CONTINUE
C             ELSE START AT FIRST AVAILABLE FROM-TO
      DO 155 I = 1,NXFTO
      IF (NXSTAT(I) .EQ. 0)  GO TO 160
  155 CONTINUE
      GO TO 300
C             START LOOP
  160 NLOOP = NLOOP + 1
      NXLOOP(1,NLOOP) = NXFROM(I)
      NXLOOP(2,NLOOP) = NXTO(I)
```

```
C        2.  OPEN SEGMENTS - LENGTH
C
  300 CALL TAGSRT (NLOOP,LOOPLN,LOPTAG)
      N = 0
      DO 310 I = 1,NLOOP
      K = LOPTAG(I)
      IF (NXLOOP(1,K) .EQ. NXLOOP(LOOPLN(K),K))  GO TO 310
      N = N + 1
      LOOPRI(K) = N
  310 CONTINUE
      DO 320 I = 1,NLOOP
      K = LOPTAG(I)
      IF (NXLOOP(1,K) .NE. NXLOOP(LOOPLN(K),K))  GO TO 320
      N = N + 1
      LOOPRI (K) = N
  320 CONTINUE
C
C     PRINT LOOPS AND SEGMENTS FOUND.
C
      WRITE (IOUT,1300)
 1300 FORMAT ('1  LOOPS AND SEGMENTS IN STATE-FLOW FROM-TO LIST'/)
      DO 350 I = 1,NLOOP
      WRITE (IOUT,1310) I,LOOPRI(I)
 1310 FORMAT ('0LOOP NUMBER =',I4,6X,'INITIAL PRIORITY=',I4//
     1    20X,'NODE'/)
      WRITE (IOUT,1320)  (NODNAM(J,NXLOOP(1,I)),J=1,NAMLEN),NXLOOP(1,I)
 1320 FORMAT (10X,' FROM',5X,16A1,1X,'(',I3,')')
      NN = LOOPLN(I)
      DO 330 K = 2,NN
      WRITE (IOUT,1330) (NODNAM(J,NXLOOP(K,I)),J=1,NAMLEN),NXLOOP(K,I)
 1330 FORMAT (10X,'   TO',5X,16A1,1X,'(',I3,')')
  330 CONTINUE
  350 CONTINUE
C     ************** **************************************************
C     MAP LOOPS AND SEGMENTS ONTO CUBE
C
      NC = NCOUNT
      NSUCES = 0
      NTRY = 0
  400 DO 401 I=1,MAXNOD
      NODE(I) = -1
  401 NODADR(I) = 0
      NCOUNT = NC
      NDFTO = 0
      NTRY = NTRY + 1
C
C             CHECK FOR INPUT ASSIGNED STATES
C
      IF (.NOT. $STATE)  GO TO 409
C             THERE ARE ASSIGNED INPUT STATES.  FILL IN NODE AND NODADR
C              ARRAYS TO REFLECT INPUT ASSIGNED STATES.
      DO 405 I=1,NSTAT
      NODE(NSTADR(I)) = NSTATE(I)
  405 NODADR(NSTATE(I)+1) = NSTADR(I)
C
C
```

```
      LOOPLN(NLOOP) = 2
      NXSTAT(I) = 1
  165 KFROM = NXTO(I)
  166 K1 = 1
      NREM = 0
C               SEARCH FOR A CONTINUATION OF THIS LOOP
  167 IF (K1 .GT. NXFTO)  GO TO 175
      DO 170 I = K1,NXFTO
      IF (NXSTAT(I) .NE. 0)  GO TO 170
      NREM = NREM + 1
      IF (NXFROM(I) .EQ. KFROM)  GO TO 180
  170 CONTINUE
C               END OF LOOP PROCESSING
  175 IF (NREM .EQ. 0)  GO TO 300
      GO TO 145
C               LOOP CONTINUATION FOUND.  CHECK IF THIS NODE ALREADY IN
C                 LOOP.
  180 NN = LOOPLN(NLOOP)
      DO 190 K=1,NN
      IF (NXLOOP(K,NLOOP) .EQ. NXTO(I))  GO TO 200
  190 CONTINUE
C               OK.  SAVE IT
      LOOPLN(NLOOP) = LOOPLN(NLOOP) + 1
      NXLOOP(LOOPLN(NLOOP),NLOOP) = NXTO(I)
      NXSTAT(I) = 1
      GO TO 165
C
C               IF LOOPS TO IMMEDIATELY PRECEDING NODE, THEN OK.
  200 IF (K .NE. NN-1)  GO TO 210
C               THE SKIPPED STATE-FLOW FROM-TO WILL BE PICKED UP LATER
C               K1 IS SET HERE SO THAT THE FROM-TO NEXT EXAMINED WILL BE
C                 THE FIRST ONE FOLLOWING THE LAST ONE PROCESSED.
      K1 = I+1
      GO TO 167
C               IF LOOP TO FIRST NODE IN LOOP, SAVE IT AND GO TO NEXT LOOP
  210 IF (K .NE. 1)  GO TO 220
      LOOPLN(NLOOP) = LOOPLN(NLOOP) + 1
      NXLOOP(LOOPLN(NLOOP),NLOOP) = NXTO(I)
      NXSTAT(I) = 1
      GO TO 145
C               APPARENTLY A SMALLER LOOP FOUND.  SAVE BEGINNING SECTION
  220 DO 230 J = 1,K
  230 NXLOOP(J,NLOOP+1) = NXLOOP(J,NLOOP)
      LOOPLN(NLOOP+1) = K
      KFROM = NXLOOP(K,NLOOP+1)
C               SHIFT NLOOP STRING OVER
      DO 240 J = K,NN
  240 NXLOOP(J-K+1,NLOOP) = NXLOOP(J,NLOOP)
      LOOPLN(NLOOP) = LOOPLN(NLOOP) - K + 2
      NXLOOP(LOOPLN(NLOOP),NLOOP) = NXTO(I)
      NXSTAT (I) = 1
      NLOOP = NLOOP + 1
      GO TO 166
C
C      ASSIGN INITIAL LOOP PRIORITY
C       1.  CLOSED LOOPS - LENGTH
```

```
C
C
C
C              SUCCESSFUL SOLUTION.  PRINT SOME PARAMETERS
C
  600 NSUCES = NSUCES + 1
      WRITE (IOUT,1610) NSUCES,NFF,NCOUNT,MAXNOD,NTRY
 1610 FORMAT ('1 RESULTS OF SOLUTION NUMBER',I4//
     1 1X,I5,' STORAGE ELEMENTS USED'//
     2 1X,I5,' STATES USED OUT OF',I5,' TOTAL STATES AVAILABLE'//
     3 1X,I5,' TRIALS THIS NUMBER OF STORAGE ELEMENTS'///)
C
C              PRINT NODE MAP.
C
      WRITE (IOUT,1620)
 1620 FORMAT ('0',10X,'NODE MAP'//7X,'NODE NAME',9X,'STATE (LITERAL FORM
     A)'/)
      DO 650 I = 1,NCOUNT
      CALL TERMOT (NFF,NODE(I),0,KARD)
C              CONVERT INTERNAL NODE TO A,B,C'S
       KARD(NFF+1)=IBLNK
      KARD(NFF+2)=ILEFT
      K=NFF+2
      DO 630 NN = 1,NFF
      CALL PRTST (4,NODE(I),31-NFF+NN, IB)
      IF (IB .EQ. 1)  GO TO 625
      KARD(K+1) = INOT
      KARD(K+2) = IVAR(NN)
      K = K+2
      GO TO 630
  625 KARD(K+1) = IVAR(NN)
      K = K+1
  630 CONTINUE
      K=K+1
      KARD(K)=IRIGHT
      WRITE (IOUT,1630) I,(NODNAM(J,I),J=1,NAMLEN),(KARD(J),J=1,K)
 1630 FORMAT (1X,I4,2X ,16A1,2X,80A1)
  650 CONTINUE
C
C              PRINT INTERNAL FROM-TO MAPPING
C
      WRITE (IOUT,1660)
 1660 FORMAT ('1',10X,'FROM-TO TABLE'/' FROM NODE',19X,'TO NODE'/)
      DO 670 I = 1,NDFTO
      WRITE (IOUT,1670) (NODNAM(J,NODADR(NDFROM(I)+1)),J=1,NAMLEN),
     1   NODADR(NDFROM(I)+1),
     1       (NODNAM(J,NODADR(NDTO(I)+1)),J=1,NAMLEN)
     2  ,NODADR(NDTO(I)+1)
 1670 FORMAT (1X,16A1,1X,'(',I3,')',6X,16A1,1X,'(',I3,')')
  670 CONTINUE
C     ****************************************************************
C     FORM SET AND RESET EQUATIONS FOR STORAGE ELEMENTS
C
      WRITE (IOUT,1090)
 1090 FORMAT (1H1)
C
```

```
C               SORT INTO PRIORITY
  409 CALL TAGSRT (-NLOOP,LOOPRI,ITAGA)
      IF (ITRACE .GE. 1)  WRITE (IOUT,1400)  (NTRY,NFF,I=1,188)
 1400 FORMAT (1H1/4I'      BEGIN TRIAL=',I3,' FOR NFF=',I3))
      NLL = 1
  410 NL = ITAGA(NLL)
C               CHECK FOR CLOSED LOOPS
      IF (NXLOOP(1,NL) .NE. NXLOOP(LOOPLN(NL),NL)) GO TO 420
C               YES CLOSED.  GO MAP IT
      CALL LOPMAP (LOOPLN(NL)-1, NXLOOP(1,NL),IRET)
      IF (IRET .GT. 0)  GO TO 480
      GO TO 460
C
C               MAP SEGMENT
  420 CALL SEGMAP (LOOPLN(NL),NXLOOP(1,NL),IRET)
      IF (IRET .GT. 0)  GO TO 480
C
C               SET UP FOR NEXT LOOP OR SEGMENT
  460 NLL = NLL + 1
      IF (ITRACE .GE. 1)  WRITE (IOUT,1460) NL
 1460 FORMAT ('1 THE FOLLOWING NODE MAP IS FOR LOOP',I4,'  WHICH WAS SUC
     1CESSFULLY MAPPED'/)
      IF (ITRACE .GE. 1)  CALL ERRPRT
      IF (NLL .GT. NLOOP)  GO TO 600
      GO TO 410
C
C               MAPPING UNSUCCESSFUL.  UP PRIORITY OF LAST LOOP OR
C               SEGMENT TO GET IT MAPPED SOONER.
C
  480 IF (ITRACE .GE. 1)  WRITE (IOUT,1480) NL,NTRY
 1480 FORMAT (' *** MAPPING OF LOOP OR SEGMENT',I3,' UNSUCCESSFUL. NUMBE
     1R OF TRIES=',I4)
      IF (NTRY .GE. MAXTRY)  GO TO 500
C
C               BUMP PRIORITY OF LOOP THAT FAILED
      ITEMP = LOOPRI(NL)
      LOOPRI(NL) = LOOPRI(NL) + 1
      IF (ITRACE .GE. 1)  WRITE (IOUT,1482) NL,LOOPRI(NL),ITEMP
 1482 FORMAT ('0*** LOOP',I4,' PRIORITY NOW=',I4,' WAS=',I4)
  485 IF (ITRACE .LT. 1)  GO TO 400
      CALL ERRPRT
      WRITE (IOUT,1481)
 1481 FORMAT ('0 STATUS OF MAPPING AT FAILURE PRECEDES THIS MESSAGE'/
     1    1X,120('*')/)
      GO TO 400
C
C               TRY BUMPING NFF BY ONE
C
  500 NFF = NFF + 1
      IF (NFF .GT. MAXNFF)  GO TO 900
      WRITE (IOUT,1500) NFF
 1500 FORMAT ('0*** STORAGE ELEMENTS INCREASED TO ',I4/)
      MAXNOD = 2**NFF
      CALL LOGIC (NFF)
      NTRY = 0
      GO TO 485
```

```
      IF (NODADR(KTO+1) .LT. NODDUM)  GO TO 2125
C             CONTINUE DUMMY PATH
      NP = 0
      KS = KTO
      DO 2121 NN=1,NDFTO
      IF (NDFROM(NN) .NE. KTO)  GO TO 2121
      NP = NP + 1
      JTO = NDTO(NN)
 2121 CONTINUE
      IF (NP-1)  2122,2119,2123
C             NO APPARENT PATH (HAZARD)
 2122 WRITE (IOUT,12122) KS
12122 FORMAT ('0',120('*')/' PROBABLE HAZARD AT CUBE NODE= ',Z8/
     1   1X,120('*'))
      GO TO 2125
C             MULTIPLE PATHS FROM A DUMMY NODE (PROBABLE RACE).
 2123 WRITE (IOUT,12123) KS
12123 FORMAT ('0',120('*')/' PROBABLE RACE AT CUBE NODE= ',Z8/
     1    1X,120('*'))
      GO TO 2119
 2125 CONTINUE
C
C
      DO 2130 NN=1,NXFTO
      IF (NODE(NXFROM(NN)) .NE. NDFROM(KK) .OR. NODE(NXTO(NN)) .NE.
     1     KTO) GO TO 2130
C             FOUND SIGNAL.  SCAN FIELD BACKWARDS TO ELIMINATE BLANKS
      DO 2131 I=1,SIGLEN
      IK = SIGLEN+1 - I
      IF (NXSIG(IK,NN) .NE. IBLNK)  GO TO 2132
 2131 CONTINUE
C
 2132 ISET(KSETR(KSET+1)) = ILEFT
      KSET = KSET + 1
      DO 2133 I=1,IK
      KSET = KSET + 1
 2133 ISET(KSETR(KSET)) = NXSIG(I,NN)
      KSET = KSET + 1
      ISET(KSETR(KSET)) = IRIGHT
 2130 CONTINUE
C             CHECK FOR PRINT LINE OVERFLOW
C             ARRAY ISET IS TREATED AS CIRCULAR
      IF (KSET - IQBEG .LT. 120)  GO TO 2100
C             PRINT FROM IQBEG TO KORSET
      IF (KPR .GT. 0)  GO TO 2105
      KPR = 1
      IF (NEQ .EQ. 1) WRITE(IOUT,1100) IVAR(N),(ISET(KSETR(I)),I=IQBEG,
     1     KORSET)
      IF (NEQ .EQ. 2) WRITE(IOUT,1110) IVAR(N),(ISET(KSETR(I)),I=IQBEG,
     1     KORSET)
      GO TO 2109
 2105 WRITE (IOUT,1105) (ISET(KSETR(I)),I=IQBEG,KORSET)
 2109 IQBEG = KORSET+1
C
 2100 CONTINUE
C             PRINT REMAINING PORTION OF THE EQUATION
```

```
C      START SCAN OF VARIABLES  (NODE BIT 31 IS FIRST)
C
       DO 2200 N=1,NFF
C            LOOP TO 2199 TO PICK FIRST SET THEN RESET EQUATIONS
       DO 2199 NEQ = 1,2
       KSET = 0
       DO 2101 I=1,240
 2101  ISET(I) = IBLNK
       KORSET = 0
       IQBEG = 1
       KPR = 0
C
C     DO 2100 LOOP MATCHES ALL INTERNAL FROM.TO'S WITH EACH OTHER LOOKING
C      FOR ONES WHICH DIFFER ONLY IN THE VARIABLE IN QUESTION
C
       DO 2100 KK=1,NDFTO
       CALL NODECM (NDFROM(KK),NDTO(KK),32-N,IVAL)
       IF (IVAL .NE. NEQ)  GO TO 2100
C
C      ADD TERMS TO EQUATIONS.   NEQ=1 FOR SET.  NEQ=2 FOR RESET
C
       IF (KSET .EQ. 0) GO TO 2111
C        OR TERMS AFTER FIRST
       IF (KSET-IQBEG-IQBEG .LT. 120)  KORSET = KSET
       ISET(KSETR(KSET+2)) = IOR
       KSET = KSET + 3
 2111  DO 2120 NN=1,NFF
C              IF THE SET OR RESET EQUATION BEING WRITTEN IS FOR THE SAME
C              INTERNAL VARIABLE (IE, N=NN), THEN THE VARIABLE IS
C              PROBABLY REDUNDANT.  A CHECK IS PERFORMED TO DETECT IF
C              THIS FROM-TO MAPPING IS PART OF AN EDGE LOOP.  IF IT IS,
C              RETAIN THE FULL PRESENT STATE MINTERM.
       IF (N .NE. NN)  GO TO 2114
       DO 2112 KQ=1,NDFTO
       IF (NDFROM(KK) .EQ. NDTO(KQ) .AND. NDTO(KK) .EQ. NDFROM(KQ))
      A       GO TO 2114
 2112  CONTINUE
       GO TO 2120
 2114  CALL PBTST (4,NDFROM(KK),31-NFF+NN,IB)
       IF (IB .EQ. 1)  GO TO 2115
       ISET(KSETR(KSET+1)) = INOT
       ISET(KSETR(KSET+2)) = IVAR(NN)
       KSET = KSET + 2
       GO TO 2120
C
 2115  ISET(KSETR(KSET+1)) = IVAR(NN)
       KSET = KSET + 1
 2120  CONTINUE
C              LOOK FOR EXTERNAL SIGNALS TO BE AND'ED WITH THIS TERM
C
C        IF NDTO(KK) IS A DUMMY NODE, THEN FOLLOW DUMMY PATH UNTIL AN
C      EXTERNAL NODE IS REACHED.  IF THERE ARE TWO POSSIBLE PATHS TO
C      FOLLOW, PRINT ERROR MESSAGE TO THAT EFFECT (PROBABLE RACE).
C
       JTO = NDTO(KK)
 2119  KTO = JTO
```

```
C               CHECK IF KFROM AND KTO ARE NEIGHBORS
      CALL NABOR (NFF,KFROM,KTO,M)
C               IF ONLY ONE BIT DIFFERENT, NODE MAPPING SUFFICIENT
      IF (M .NE. 1) GO TO 120
C               CHECK IF APPROPRIATE FROM-TO IN TABLES
      IF (NDFTO .LE. 0)  GO TO 111
      DO 110 JF=1,NDFTO
      IF (KFROM .EQ. NDFROM(JF) .AND. KTO .EQ. NDTO(JF))  GO TO 900
  110 CONTINUE
C               NOT FOUND.  ADD IT.
  111 NDFTO = NDFTO + 1
      NDFROM(NDFTO) = KFROM
      NDTO(NDFTO) = KTO
      GO TO 900
C
C               MORE THAN ONE CUBE NODE AWAY.  CHECK IF ANY EXISTING
C               DUMMY PATHS LEAD TO NODE KTO.
  120 IF (M .EQ. 0)  GO TO 900
      CALL DUMXST (KTO,N,NODSAV)
      IF (N .NE. 0)  GO TO 250
C
C               NO DUMMY PATHS EXIST.  ROUTE ONE
      JTO = KTO
      ISW = 1
  130 DO 150 MAX=1,NFF
      IFROM = KFROM
      ITO = JTO
  140 IPREF = 0
      IF (2**MAX .LE. NFF)  IPREF = LXOR(IFROM,ITO)
      CALL NSELCT (IFROM,IPREF,NFF,JODADR,NEIGH,IERR)
      IF (IERR .GT. 0) GO TO 145
C               ADD DUMMY NODE
C               CHECK FOR NODE ARRAY OVERFLOW
      IF (NCOUNT .GE. MAXNOD)  GO TO 145
      NCOUNT = NCOUNT + 1
      JODE(NCOUNT) = NEIGH
      JODADR(NEIGH+1)  = NCOUNT
      NDFTO = NDFTO + 1
      NDFROM(NDFTO) = IFROM
      NDTO(NDFTO) = NEIGH
C               CHECK IF ROUTING COMPLETE
      CALL NABOR (NFF,NEIGH,JTO,M)
      IF (M .EQ. 1) GO TO 160
      IFROM = NEIGH
      GO TO 140
C               SET UP TO TRY ANOTHER PATH
  145 NN = NCA + 1
      IF (NN .GT. NCOUNT)  GO TO 149
C               RESTORE ARRAYS
      DO 148 J=NN,NCOUNT
      JODADR(JODE(J)+1) = 0
  148 JODE(J) = -1
  149 NDFTO = NDA
      NCOUNT = NCA
  150 CONTINUE

      IF (KPR .GT. 0)  GO TO 2205
      IF (NEQ.EQ.1) WRITE(IOUT,1100) IVAR(N),(ISET(KSETR(I)),I=IQBEG,
     1     KSET)
 1100 FORMAT ('0  SET ',A1,' = ',120A1)
      IF (NEQ.EQ.2) WRITE(IOUT,1110) IVAR(N),(ISET(KSETR(I)),I=IQBEG,
     1     KSET)
 1110 FORMAT('ORESET ',A1,' = ',120A1)
      GO TO 2199
 2205 WRITE (IOUT,1105) (ISET(KSETR(I)),I=IQBEG,KSET)
 1105 FORMAT (11X,120A1)
 2199 CONTINUE
C
C
 2200 CONTINUE
      WRITE (IOUT,1090)
      IF (NSUCES .LT. MSUCES)  GO TO 400
      IF (IEND) 1,1,999
C
C               RUN UNSUCCESSFUL.  PRINT MESSAGE.
  900 WRITE (IOUT,1900)
 1900 FORMAT ('1  RUN UNSUCCESSFUL')
      GO TO 999
  999 STOP
      END
```

VITA

Bienvenido C. Peralta

Candidate for the Degree of

Master of Science

Thesis: COMPUTER AIDED DESIGN OF A DIGITAL FREQUENCY SYNTHESIZER

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Manila, Philippines, December 7, 1927, the son of B. L. Peralta and Dr. Maria A. Cid.

Education: Graduated from Ilocos Norte High School, Ilocos Norte, Philippines in 1946; received the Bachelor of Science degree in Electrical Engineering from Oklahoma State University in 1953; completed requirements for the Master of Science degree from Oklahoma State University in May, 1974.

Professional Experience: Educational grant and on-the-job training at FAA Aeronautical Center, Oklahoma City, 1947-49; instructor, University of Missouri at Rolla, 1953-55; research assistant, Emerson Electric, St. Louis, Mo., 1956-60; design engineer, Honeywell Corporation, Clearwater, Fla., 1960-65; project engineer, Battelle Laboratories, 1965-68; staff engineer in computer programming, Magnavox Company, 1968.

Professional Organizations: Member IEEE and National Management Association.