AD-A040 269 BREUER AND ASSOCIATES ENCINO CALIF F/G 9/5
FUNCTIONAL LEVEL MODELING OF COMPLEX ELEMENTS IN TEST/80,(U)
SEP 76 M A BREUER N00014-75-C-1053
UNCLASSIFIED TEST/80-1-76 NL

1 OF 1
AD
A040269

END
DATE
FILMED
6-77

AD A040269

12

DDC
JUN 7 1977
C

BREUER & ASSOCIATES

Specialists in Design Automation

AD No.
DDC FILE COPY

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

copy

BREUER & ASSOCIATES

SOFTWARE SYSTEMS AND
CONSULTATION IN CAD/CAM

16857 BOSQUE DRIVE
ENCINO, CALIF. 91436

(213) 981-7583

(12)

(14) TEST/80
Report No. 1-76

(6) Functional Level Modeling of Complex Elements in TEST/80*

DDC
RECEIVED
JUN 7 1977
C

Submitted to:

Department of the Navy
Office of Naval Research
Arlington, Virginia 22217

Submitted by:

Breuer and Associates

(12) 81 p.

(10)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Project Director:
Melvin A. Breuer
Melvin A. Breuer
September 1, 1976

(11) 1 Sep 76

*This work was carried out under Contract No. N00014-75-C-1053 (5) NR 048-631.

SC 392 045

TABLE OF CONTENTS

Page

1. Introduction 1

1.1 Generic Functional Language 2

1.2 System Overview 3

1.3 Example: Treatment of Edge-Triggered Devices 6

2. JK Master-Slave Flip-Flop 9

2.1 Implication for the JK Master-Slave Flip-Flop 14

2.1.1 Forward Implication 14
2.1.2 Backward Implication 15
2.1.3 Equation Approach 16

2.2 D-Drive for the JK Master-Slave Flip-Flop 19

2.3 Line Justification for JK Flip-Flop 22

3. Bidirectional Shift Register 26

3.1 Implication 28

3.1.1 Tabular Approach 28
3.1.2 Equation Approach 37

3.2 D-Drive for the Shift Register 39

3.2.1 Single Time Frame D-Drive 39
3.2.2 Multiple Time Frame D-Drive 43

3.3 Line Justification for the Shift Register 49

3.3.1 Single Time Frame Line Justification 50
3.3.2 Multiple Time Frame Line Justification 51

4. Up-Down Counter 54

4.1 Implication for UP/DOWN Counter 56

4.2 D-Drive for the Counter 66

4.2.1 Single Time Frame D-Drive 66
4.2.2 Multiple Time Frame D-Drive 68

4.3 Line Justification for UP/DOWN Counter 71

4.3.1 Single Time Frame Line Justification 71
4.3.2 Multiple Time Frame Line Justification 73

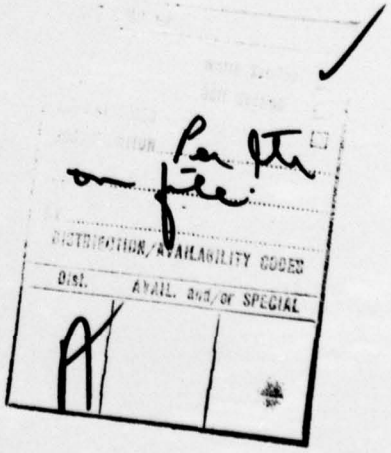
5. Summary 77

ABSTRACT

This report documents the result of just one task of our contract. This task was to investigate the feasibility of handling complex elements functionally, rather than at the logic level, in an ATG system. We have assumed a test generation system based upon the concepts of path sensitization a la the D-algorithm.

In this report we demonstrate this feasibility. More precisely, we have developed function models for three devices (a flip-flop, a counter, and a shift register). These models include mechanisms for processing implication, D-drive and line justification.

We introduce concepts for dealing with both edge-triggered and level devices, as well as a generic language for specifying the functional operation of a device.



FUNCTIONAL TEST GENERATION: IMPLICATION, D-DRIVE AND LINE JUSTIFICATION

1. Introduction

This report deals with the subject of functional modeling of devices as it will be used in the Breuer and Associates automatic test generation system called TEST/80. One main concept inherent in the design of our system is that most complex digital systems consist of functional elements such as decoders, multiplexers, ALU's, shift registers, counters, etc.

In order to reduce the complexity of test generation, these units are primitives in TEST/80. This approach is of crucial importance when one considers such new devices as LSI's, where gate level modeling is either impossible or else impractical.

To prove the feasibility of our approach we chose to analyze the development of functional primitives for three sequential devices, namely a JK master-slave flip-flop, an UD/DOWN counter, and a bilateral shift register.

In this report we document the results of this analysis. We consider three operations on each element, namely implication (simulation), D-drive (error propagation), and line justification (backward simulation). Line justification could be extended to deal with critical cubes rather than primitive cubes, in which case the LASAR test generation algorithm could be executed at the functional level.

In our system tables describing the operation of a device are seldom used. Instead, each device is associated with a set of algorithms.

These algorithms generate solutions to given problems. Often there are more than one solution to a problem. In this case, by backtracking the other solutions are obtained. Also, solutions may require one or more vectors (time frames). Our algorithms work in one of two modes, i.e., they can either give a single-time frame solution, in which case they must repeatedly be re-entered to get the next time frame solution, or else they can directly generate a multiple-time frame solution.

1.1 Generic Functional Language

All of the devices we have considered are synchronous and control driven. That is, we can divide the inputs into three groups, namely clock, data and control lines. The outputs are considered data lines.

All operations on a device are governed by the clock and control lines. Also, many devices operate in a similar fashion, e.g. there may be numerous shift register chips which differ in minor logical aspects but functionally operate in the same way. To make it easier to develop functional models for new devices we have devised a functional level language which will be sketched below. Some of the basic symbols in this language are listed below, along with their meaning.

H: hold

P: parallel load

R: right shift

L: left shift

U: increment (up)

D: decrement (down)

etc.

We abbreviate the concept $\underbrace{\times \times \cdots \times}_{n \text{ times}}$ by $\times^n$, and the notation $\times^*$ represents zero or more repetitions of the symbol $\times$. Information in parenthesis represents data on the data lines. Hence, the sequency L(00)H*UH*UH*U shows one way to get an (11) into a counter, namely, parallel load (00) and count up three times. Note that each increment can be followed by 0 or more applications of holds.

In order to convert these symbolic solutions to 0's and 1's as understood by the test generation algorithm, a translator is used (see Figure 1). Each device has such a translator. For example, for a JK flip-flop the concept of hold can be implemented in many ways such as:

1. $J = K = 0$ and CLOCK = $\times$
2. $J = Q$, $K = \bar{Q}$ and CLOCK = $\times$

etc.

In short, the translator converts a symbolic concept to the set of all logic values corresponding to that concept for a specific device. The translator works vector-by-vector, and again, backtracking is necessary because there may be more than one logic vector associated with a given symbolic vector.

Note that devices can be clocked in several ways. Some are level devices and others are edge-triggered (either positive, negative or both). Again, our generic language compensates for these differences. For example, for a positive edge-triggered device we may define C = 1 as the condition for clocking, i.e., a 0/1 transition, while C = 0 represents the other three conditions, namely 1/0, 1/1, and 0/0. Again, the translator converts the generic clock to logic values.

1.2 System Overview

In Figure 1 we illustrate the general structure for our functional modeling software system. In this figure we show the problem selector portion of the ATGP sending a problem to the modeling software. A typical problem might be "...justify a 1 on line i of defice j." Also, the current value of all lines associated with this device are also specified in the problem. The system then selects the correct functional model associated with device j (several devices can share the same functional model), and a generic solution is produced. This solution is then processed through the translator to get a logic solution which is then sent back to the ATGP.

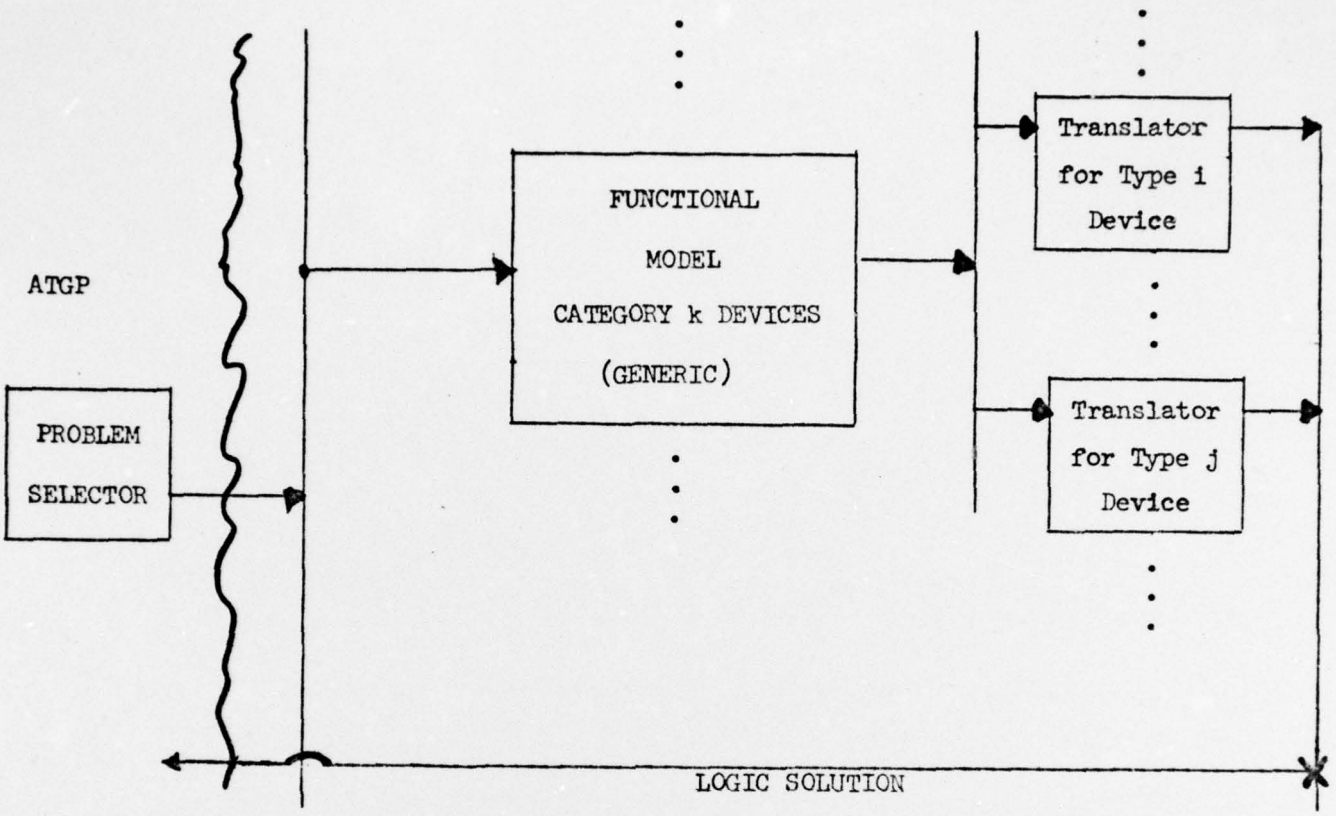


FIGURE 1: MODELING SOFTWARE

The uniqueness of the work presented here is that we have developed general techniques for handling complex sequential primitives over three-valued logic $(0,1,\times)$.

1.3 Example: Treatment of Edge-Triggered Devices

In our work, line values are restricted to be either 0, 1 or × (unspecified). However, for edge-triggered devices, over a period of time from t_1 to t_2, we can have the line values 0/1, 1/0, 0/0 or 1/1, where 0/1 is a positive edge, 1/0 a negative edge, 0/0 a static zero, and 1/1 a static 1. To illustrate how our three logic values handle the four cases, consider a positive edge-triggered D-flip-flop whose primitive cubes are shown in Figure 2(a). Here, the actual clock $C\ell$ is represented by two signals $C\ell^-$ and $C\ell^+$. Figure 2(b) shows the primitive cubes of a level D flip-flop. It is possible to use the primitive cubes of Figure 2(b) for both level and edge-triggered devices with different interpretations. The signal C is interpreted as a condition signal. For a positive edge-triggered device, $C = 1$ implies $C\ell^+ = 1$ which in turn implies $C\ell^- = 0$. Thus, the cube 3 of Figure 2(b) is interpreted (in this case) in the same way as cube 4 of Figure 2(a). Similarly, cube 8 of Figure 2(a) and cube 6 of Figure 2(b) are identically interpreted. For a positive edge-triggered device the condition $C = 0$ implies $C\ell^+ = 0$ or (since the identity $C\ell^+ \cdot C\ell^- = 0$

is valid) $C\ell^- = 1$. Thus, cube 2(5) of Figure 2(b) corresponds to cubes 2(6) and 3(7) of Figure 2(a). In a similar manner the cubes for a

	D	$C\ell^-$	$C\ell^+$	y	Y
1	0	-	-	0	0
2	-	1	-	0	0
3	-	-	0	0	0
4	0	0	1	-	0
5	1	-	-	1	1
6	-	1	-	1	1
7	-	-	0	1	1
8	1	0	1	-	1

(a)

	D	C	y	Y
1	0	-	0	0
2	-	0	0	0
3	0	1	1	0
4	1	-	1	1
5	-	0	1	1
6	1	1	-	1

(b)

FIGURE 2*

negative edge-triggered D flip-flop (Figure 2(c)) can be derived from the cubes of Figure 2(b). Thus it is only necessary to consider one set of cubes to represent a whole family of devices. This comment also applies

*A " - " entry implies that this value can be replaced by a 0, 1 or ×.

to D-cubes and to more complex functional devices. Consequently, in the remainder of this report we will ignore the edge-triggered aspects of devices and concentrate on cubes which, if properly interpreted, represent the behavior of a family of functional devices. The interpretation is done by the translator.

D	$C\ell^+$	$C\ell^-$	y	Y	Corresponding to Cube in Figure 2(b)
0	-	-	0	0	1
-	1	-	0	0	2
-	-	0	0	0	2
0	0	1	-	0	3
1	-	-	1	1	4
-	1	-	1	1	5
-	-	0	1	1	5
1	0	1	-	1	6

(c)

FIGURE 2: NEGATIVE EDGE-TRIGGERED F/F

For more complex devices in which changes can be triggered by both positive and negative edges, some modifications are necessary. For instance, consider a master-slave flip-flop in which a positive edge enables the master and a negative edge enables the slave. Positive edge cubes should be represented with $C = 1$. Negative edge cubes should be represented with $\hat{C} = 1$. If neither a positive or negative edge is present, then $C = \hat{C} = 0$.

2. JK Master-Slave Flip Flop

We shall first consider the JK Master-Slave Flip-Flop shown in Figure 3. We shall derive procedures for determining implication, D-drive and line justification for this functional device. There are three control inputs (J,C,K) for this device. For each value of the vector (J,C,K) the device behaves in a simple functional manner which is unique if J,C,K = 0 or 1. If J,C, or K is unspecified (×), then the device may behave in any of several of the aforementioned simple functional manners. This behavior corresponds to the union of the corresponding simple functional behaviors* (see footnote below). For instance if (J,C,K) = (1,1,0) the master flip-flop is set unless the slave was reset and the slave was stable. If (J,C,K) = (1,0,0) the master is stable and the slave assumes the same value as the master. If (J,C,K) = (1,×,0) the functional behavior of the device corresponds to the union of the behavior for (J,C,K) = (1,1,0) and (J,C,K) = (1,0,0).

The cubes in the table of Figure 4 map each of the $27 = 3^3$ possible values of the vector (J,C,K) into a specific type of functional

---

*The union operation for a 3-valued algebra is defined in the following table:

U	0	1	×
0	0	×	×
1	×	1	×
×	×	×	×

.

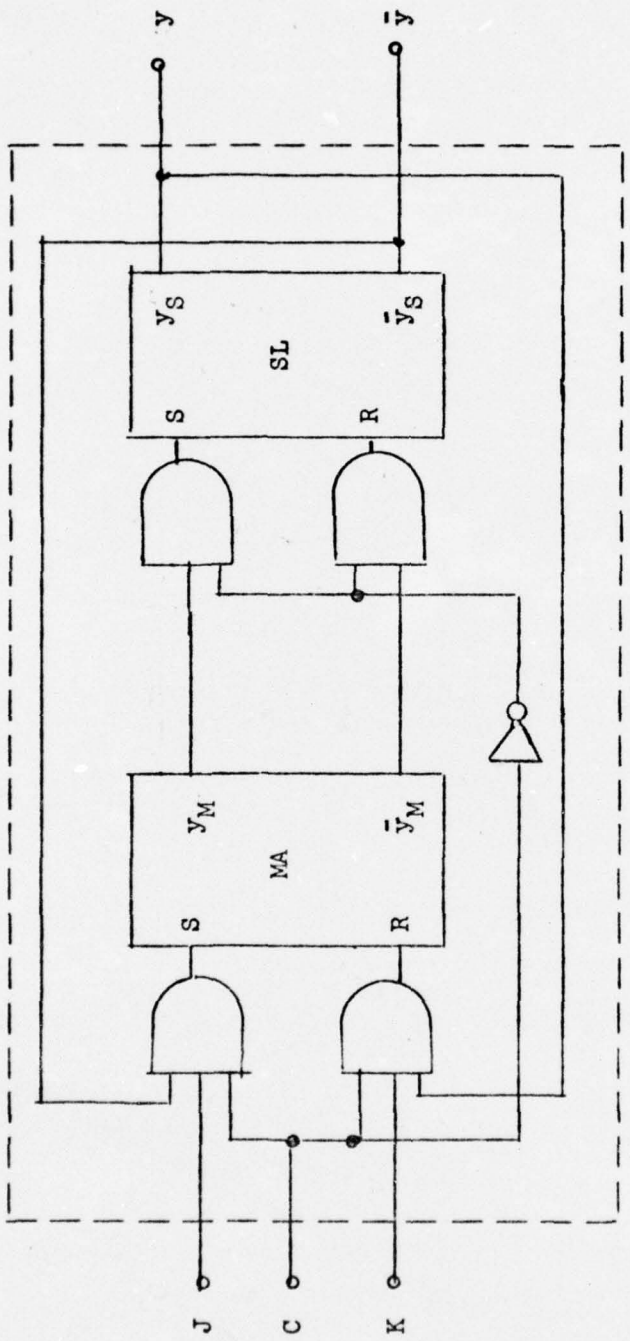


FIGURE 3. JK MASTER SLAVE FLIP-FLOP

behavior (denoted as A_i). The state behavior of this device for each of these A_i is specified in the table of Figure 5.

C	J	K	Functional Algorithm
0	×	×	A_0
1	0	0	A_1
1	0	1	A_2
1	1	0	A_3
1	1	1	A_4
1	0	×	A_5
1	×	0	A_6
1	1	×	A_7
1	×	1	A_8
1	×	×	A_9
×	0	0	A_{10}
×	0	1	A_{11}
×	1	0	A_{12}
×	1	1	A_{13}
×	0	×	A_{14}
×	×	0	A_{15}
×	1	×	A_{16}
×	×	1	A_{16}
×	×	×	A_{16}

FIGURE 4

y_M y_S	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}
0 0	0 0	0 0	0 0	1 0	1 0	0 0	× 0	1 0	× 0	× 0	0 0	0 0	× ×	× ×	0 0	× ×	× ×
0 1	0 0	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 ×	0 ×	× ×	× ×	0 ×	× ×	× ×
1 0	1 1	1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 ×	× ×	1 ×	× ×	× ×	1 ×	× ×
1 1	1 1	1 1	0 1	1 1	0 1	× 1	1 1	× 1	0 1	× 1	1 1	× ×	1 1	× ×	× ×	1 1	× ×
0 ×	0 0	0 ×	0 ×	× ×	× ×	0 ×	× ×	× ×	× ×	× ×	0 ×	0 ×	× ×	× ×	0 ×	× ×	× ×
× 0	× ×	1 0	× 0	1 0	1 0	× 0	1 0	1 0	× 0	× 0	× ×	× ×	× ×	× ×	× ×	× ×	× ×
1 ×	1 1	1 ×	× ×	1 ×	× ×	× ×	1 ×	× ×	× ×	× ×	1 ×	× ×	1 ×	× ×	× ×	1 ×	× ×
× 1	× ×	× 1	0 1	× 1	0 1	× 1	× 1	× 1	0 1	× 1	× ×	× ×	× ×	× ×	× ×	× ×	× ×
× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×	× ×

FIGURE 5: Y_M Y_S

Together the two tables of Figures 4 and 5 specify the normal behavior of this functional device.*

In some applications it may be possible to use tables which are simpler than those of Figures 4 and 5. Specifically, suppose that only the state of the slave latch of the master-slave flip-flop is of interest. Since the slave can only change state for an input corresponding to A_0, time can be considered on a less refined or "macroscopic" level, by considering only input sequences (read from left to right) of the form $A_k A_0$, $1 \leq k \leq 9$. Furthermore, for the input A_0

$$Y_S = y_M$$

and

$$Y_M = y_M \ .$$

Therefore

$$Y_M = Y_S \ .$$

Consequently in this analysis, the 9 row table of Figure 5 can be reduced to the 3 row table of Figure 6.

---

*For this simple device a single table can be used, instead of these two tables. However, in the succeeding sections we will be considering general n-bit functional devices where n is arbitrary. In these cases, two tables are necessary. For uniformity we also employ two tables to describe the flip-flop.

y	A_1 A_0	A_2 A_0	A_3 A_0	A_4 A_0	A_5 A_0	A_6 A_0	A_7 A_0	A_8 A_0	A_9 A_0
0	0	0	1	1	0	×	1	×	×
1	1	0	1	0	×	1	×	0	×
×	×	×	×	×	×	×	×	×	×

2.1 Implication for the JK Master-Slave Flip-Flop

2.1.1 Forward Implication

Forward implication determines what values are implied on the signals (Y_M, Y_S) by the input signal values (C,J,K,y_M,y_S). The tables of Figures 4 and 5 can be used to determine forward implication as illustrated in the following example.

EXAMPLE 1:

Suppose the values of (C,J,K) are $(\times,0,\times)$ and the values of (y_M,y_S) are $(0,\times)$. From the input mapping table of Figure 4 we find that this corresponds to A_{14} and from the table of Figure 5, A_{14} maps $(0,\times)$ into $(0,\times)$. Therefore, $Y_M = 0$ is implied. □

A formal procedure for forward implication in the JK Master-Slave Flip-Flop is as follows:

Procedure 1 (Forward Implication)

If a signal C,J,K,y_M,y_S changes value:

(1) Take the cubical intersection of the vector (C,J,K) with the cubes of the table of Figure 4 to determine the functional behavior A_i.

(2) Using the values of (y_M,y_S) and the A_i determined in (1), determine the values of (Y_M,Y_S) from the table of Figure 5.

2.1.2 Backwards Implication

Backward implication determines what signal values are implied on the signals (C,J,K,y_M,y_S) by the partially specified values on the signals (C,J,K,y_M,y_S,Y_M,Y_S). Backwards implication can also be determined from the input mapping table (Figure 4) and the algorithm table (Figure 5) using the cubical intersection operation.

EXAMPLE 2:

Suppose $(C,J,K,y_M,y_S,Y_M,Y_S) = (\times,\times,\times,0,0,1,0)$. From the table of Figure 5 we see that row 0 0 is mapped into the next state entry 1 0 by 3 algorithms A_3, A_4 and A_7. The intersection of the corresponding 3 cubes of Figure 4 reveals that the signal values $C = J = 1$ are implied. □

A formal procedure for backwards implication is as follows:

Procedure 2 (Backward Implication)

(1) Select the row R_i of the table of Figure 5 corresponding to the values of (y_M, y_S). Let C_i be the set of columns in Figure 5 which have entries in row R_i corresponding to the values of (Y_M, Y_S).

(2) From the set C_i, and the table of Figure 4, determine a subset $C_i' \subseteq C_i$ which is consistent with the previously specified values of (C,J,K).

(3) Form the cubical intersection of all cubes in C_i to determine implied values of (C,J,K).

In general, during each iteration forward and backward implication should be alternately repeated until stabilization occurs. Even then it is possible that some implied signals may not be determined. This will be illustrated in Section 4.1 for an UP/DOWN Counter. It is, of course, possible to rectify this deficiency by developing more complex implication procedures. However, since implication is only used to speed up the execution of the D-algorithm, the use of overly complex implication procedures probably cannot be justified.

2.1.3 Equation Approach

The behavior of a JK flip-flop can also be described using Boolean equations,

$$
\begin{aligned}
Y_M &= J\,C\,\bar{y}_S + (\bar{C} + \bar{K} + \bar{y}_S)y_M \quad ; \\
Y_S &= \bar{C}\,y_M + (C + y_M)y_S \quad .
\end{aligned}
\tag{1}
$$

These equations can then be used to determine implication.

EXAMPLE 3:

We shall consider the same cases as were examined in Examples 1 and 2.

(a) If the values of (C,J,K) are $(\times,0,\times)$ and the value of (y_M,y_S) is $(0,\times)$, then substituting in (1) we obtain:

$$
\begin{aligned}
Y_M &= J\,C\,\bar{y}_S + (\bar{C} + \bar{K} + \bar{y}_S)y_M \\
&= 0 + 0 = 0 \quad ; \\
Y_S &= \bar{C}\,y_M + (C + y_M)y_S \\
&= 0 + (\times + 0)\times = \times \quad .
\end{aligned}
$$

(b) If $(Y_M,Y_S,y_M,y_S) = (1,0,0,0)$, then from (1) we obtain:

$$
\begin{aligned}
Y_M &= J\,C\,\bar{y}_S + (\bar{C} + \bar{K} + \bar{y}_S)y_M \\
1 &= J\,C\cdot 1 + (C + K + 1)\cdot 0 \\
&= J\,C \quad .
\end{aligned}
$$

This implies $J = C = 1$.

$$Y_S = \bar{C}\, y_M + (C + y_M) y_S$$

$$0 = \bar{C} \cdot 0 + (C + 0) = 0 \quad .$$

Thus, nothing is implied. □

The equation approach, when used for implication requires the solution of a set of simultaneous Boolean equations. It may also fail to recognize implied values (as we shall illustrate in Section 4.1) due to the manner in which the $\times$ signal is handled. A comparison of the relative efficiency of the equation and tabular methods would require extensive analysis and will not be included in this report.

2.2 D-Drive for the JK Master-Slave Flip-Flop

D-drive consists of specifying values on the signals (y_M, y_S, J, C, K) to propagate a D (or $\bar{D}$) on one (or more) of these signals to Y_M or Y_S. The classical technique for carrying out D-drive on combinational elements utilizes propagation D-cubes and cubical intersection. Propagation D-cubes can also be derived for the JK Master-Slave Flip-Flop using familiar techniques which have been presented in the literature. The propagation D-cubes having a single D on (y_M, y_S, J, C, K) are shown in Figure 7. Propagation D-cubes with several D elements can be derived using the cubical intersection techniques of Roth's D-algorithm.

A formal procedure for D-drive is as follows:

Procedure 3:

(1) Generate a propagation D-cube corresponding to the D (or D's) on the signal vector (y_M, y_S, J, C, K).

(2) Compute the cubical intersection of (y_M, y_S, J, C, K) with the propagation D-cube generated in (1), if possible. If the intersection does not exist, generate a different propagation D-cube and repeat. □

The flip-flop characteristic equations can also be used for D-drive. This requires solution of a set of Boolean equations in a 5-valued $(0, 1, \times, D, \bar{D})$ algebra and is illustrated in the following example.

	J	K	C	y_M	y_S	Y_M	Y_S
1	D	×	1	0	0	D	0
2	×	D	1	1	1	$\bar{D}$	1
3	1	×	D	0	0	D	0
4	×	1	D	1	1	$\bar{D}$	1
5	×	×	0	D	×	D	D
6	×	0	1	D	1	D	1
7	0	×	1	D	0	D	0
8	×	×	1	×	D	×	D
9	1	×	1	0	D	$\bar{D}$	D
10	×	1	1	1	D	$\bar{D}$	D
11	1	1	1	×	D	$\bar{D}$	D

FIGURE 7. "SINGLE D" PROPAGATION D-CUBES OF JK FLIP-FLOP

EXAMPLE 4:

Consider the propagation of a D signal from the present state y_S to the next state (Y_S, Y_M). From the flip-flop characteristic equations we derive:

$$Y_M = J\,C\,\bar{y}_S + (\bar{C} + \bar{K} + \bar{y}_S)y_M$$
$$= J\,C\,\bar{D} + (\bar{C} + \bar{K} + \bar{D})y_M \quad ,$$

$$Y_S = \bar{C}\,y_M + (C + y_M)y_S$$
$$= \bar{C}\,y_M + (C + y_M)\,D \quad .$$

To propagate the $\bar{D}$ to Y_M we must solve the equation:

$$\bar{D} = J\,C\,\bar{D} + (\bar{C} + \bar{K} + \bar{D})y_M \quad .$$

This has the following three solutions:

$$J = C = 1, \quad y_M = 0$$
$$J = C = 1, \quad K = 1$$
$$C = K = y_M = 1$$

which correspond to the cubes 9, 10, 11 of Figure 7. To propagate the D to Y_S we must solve the equation:

$$D = \bar{C}\,y_M + (C + y_M)D \quad .$$

This has the unique solution: C = 1 which corresponds to cube 8 of Figure 7. □

2.3 Line Justification for JK Flip-Flop

Line justification is used to determine values of the signals (J,C,K,y_M,y_S) which are consistent with the values of the signals (Y_M,Y_S). In the D-algorithm this is done using cubical intersection and the primitive cubes of the element. The primitive cubes of the JK Master-Slave Flip-Flop are shown in Figure 8.*

A formal procedure for line justification is as follows:

Procedure 4.

(1) Select a cube C_i from the table of Figure 8 which is consistent with the specified values of (Y_M,Y_S).

(2) Form the cubical intersection of C_i with the vector of signal values (J,C,K,y_M,y_S) if possible. If the intersection does not exist, return to (1) and select a new cube.

□

---

*This table represents the same information as appeared in the tables of Figures 4 and 5 in a form more convenient for use in determining line justification.

	J	C	K	y_M	y_S	Y_M	Y_S
1	×	0	×	0	×	0	0
2	×	1	×	×	0	×	0
3	×	1	×	0	1	0	1
4	×	1	1	×	1	0	1
5	0	×	×	0	×	0	×
6	×	0	×	1	×	1	1
7	×	1	×	×	1	×	1
8	×	×	0	1	×	1	×
9	×	1	×	1	0	1	0
10	1	1	×	×	0	1	0

FIGURE 8

Line justification can also be done in a manner similar to backwards implication using the tables of Figures 4 and 5, or the flip-flop characteristic equations as illustrated in the following example.

EXAMPLE 5.

Suppose $(Y_M, Y_S) = (1,1)$ and $y_S = 1$. We wish to determine values of C, J, K, y_M which justify these signal values.

$$Y_M = J\,C\,\bar{y}_S + (\bar{C} + \bar{K} + \bar{y}_S)\,y_M \ .$$

Substituting (1,1) for (Y_M, Y_S) yields:

$$1 = J\,C \cdot 0 + (\bar{C} + \bar{K} + 0)y_M \ .$$

This implies $y_M = 1$, and $C = 0$ or $K = 0$.

Similarly for

$$Y_S = \bar{C}\, y_M + (C + y_M) y_S$$

we obtain

$$1 = \bar{C} \cdot 1 + (C + 1) \cdot 1 = 1 \ .$$

Thus, the line values can be justified by

$$(C,J,K,y_M) = (0,\times,\times,1) \text{ or } (\times,\times,0,1) \ ,$$

which correspond to primitive cubes 6 and 8 of Figure 8.

These justified values can also be determined from the Tables of Figures 4 and 5. From the table of Figure 5 we see there are 7 entries with $(Y_M,Y_S) = (1,1)$ of which only 5, all in row $(y_M,y_S) = (1,1)$, have $y_S = 1$. Hence, $y_M = 1$ is implied. These five entries correspond to algorithms A_0, A_1, A_3, A_6, A_{10}, A_{12} and A_{15}. A_0 corresponds to the value $(C,J,K,y_M) = (0,\times,\times,1)$. The other six algorithms correspond to $(C,J,K,y_M) = (\times,\times,0,1)$.

Notice that our representation assumes a level device. The same device could be represented as an edge-triggered device, in which case the characteristic equations would become:

$$Y_M = J\,C_+\,\bar{y}_S + (\bar{C}_+ + \bar{K} + \bar{y}_S)y_M \quad ,$$

$$Y_S = C_-\,y_M + \bar{C}_- + y_S \quad ,$$

which has the additional solution $C_+ = C_- = 0$ which would correspond to the additional primitive cube

J	C_+	C_-	K	y_M	y_S	Y_M	Y_S
×	0	0	×	1	1	1	1

which represents a hold.

3. Bidirectional Shift Register

In this section we will consider a general n-bit shift register of the type shown in Figure 9.

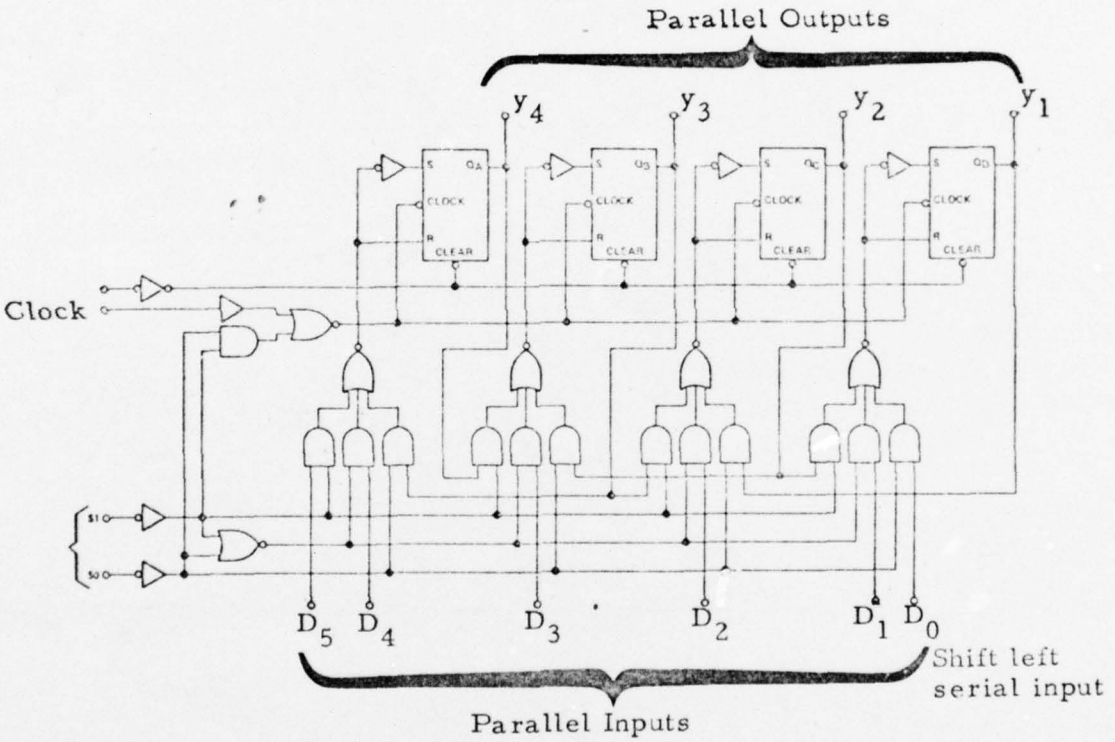


FIGURE 9. 4-BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER

Under normal operation this device can perform five functional operations:

(H) Hold - The contents of the register are unchanged:

(C) Clear - each bit of the register is reset;

(R) Shift Right - the contents of the register is shifted one bit to the right, the leftmost bit being set to the value of D_{n+1};

(L) Shift Left - the contents of the register is shifted one bit to the left, the rightmost bit being set to the value of D_0; and

(P) Parallel Load - the contents of the register are set to the values of the signals $D_1,\ldots,D_n$.

The state of bit i of the register will be denoted by y_i and the next value of this bit is denoted by Y_i.

The functional behavior of this element is determined by four control signals Z, C, S_1, S_0 as specified by the following table.

Z	C	S_1	S_0	Functional Behavior
1	×	×	×	Clear
0	0	×	×	Hold
0	1	1	1	Hold
0	1	0	0	Parallel Load
0	1	0	1	Left Shift
0	1	1	0	Right Shift

FIGURE 10

3.1 Implication

Implication can again be performed using a tabular approach or an equation concept.

3.1.1 Tabular Approach

The tabular approach requires an input mapping table which specifies the functional behavior of the device for any values of the control inputs. Unknown values on some control inputs may lead to ambiguity in the functional behavior. The resultant behavior may correspond to the UNION of several of the possible basic functional behaviors which we shall denote by concatenating the corresponding signals (H,P,L,R,C). The control inputs can assume any of $3^4 = 81$ possible values. The resultant functional behavior of the device for any of these values can be determined from the input mapping table of Figure 11.

A second table is required to specify the next state ($\underline{Y}$) of the register as a function of its present state ($\underline{y}$) and the specified functional behavior (as determined from the input mapping table of Figure 11). Unlike the case of the flip-flop it is not possible to simply list the next state information in finite tabular form, since we wish to represent the state behavior of any n-bit shift register and hence cannot restrict the value of n. However, it is possible to write very simple algorithms (in the form of small programs) to determine $\underline{Y}$ from $\underline{y}$ for each possible functional behavior. For the shift register, these algorithms are listed in the table of Figure 12.

Z	C	S_1	S_0	Algorithm	No. of (new) Input Conditions Covered
1	-	-	-*	C	27
0	0	-	-	H	9
0	-	1	1	H	2
0	1	0	0	P	1
0	1	0	1	L	1
0	1	1	0	R	1
0	1	×	0	RP	1
0	1	×	1	LH	1
0	1	0	×	LP	1
0	1	1	×	RH	1
0	1	×	×	RLPH	1
0	×	0	0	PH	1
0	×	0	1	LH	1
0	×	1	0	RH	1
0	×	×	0	RPH	1
0	×	×	1	LH	1
0	×	0	×	LPH	1
0	×	1	×	RH	1
0	×	×	×	RLPH	1
×	0	-	-	HC	9
×	-	1	1	HC	2
×	1	0	0	PC	1
×	1	0	1	LC	1
×	1	1	0	RC	1
×	1	×	0	RHC	1
×	1	×	1	LHC	1
×	1	0	×	LPC	1
×	1	1	×	RHC	1
×	1	×	×	RLPHC	1
×	×	0	0	PHC	1
×	×	0	1	LHC	1
×	×	1	0	RHC	1
×	×	×	0	RPHC	1
×	×	×	1	LHC	1
×	×	0	×	LPHC	1
×	×	1	×	RHC	1
×	×	×	×	RLPHC	1

FIGURE 11. INPUT-ALGORITHM MAPPING TABLE FOR BIDIRECTIONAL SHIFT REGISTER

*A - is used to indicate 0 or 1 or ×.

Algorithm Number	Algorithm	Algorithm Equations
1	C(Clear)	$Y_i = 0$ for all i
2	R(Shift Right)	$Y_i = y_{i+1}$ for $1 \leq i < n$ $Y_n = D_{n+1}$
3	L(Shift Left)	$Y_i = y_{i+1}$ for $1 < i \leq n$ $Y_1 = D_0$
4	H(Hold)	$Y_i = y_i$ for $1 \leq i \leq n$
5	P(Parallel Load)	$Y_i = D_i$ for $1 \leq i \leq n$
6	RH(Right or Hold)	$Y_i = \times$ unless $y_i = y_{i+1}$ for $1 \leq i < n$ $Y_i = y_i$ if $y_i = y_{i+1}$ for $1 \leq i < n$ $Y_n = y_n$ if $y_n = D_{n+1}$ $Y_n = \times$ if $y_n \neq D_{n+1}$
7	LH(Left of Hold)	$Y_i = y_i$ if $y_i = y_{i-1}$ for $1 < i \leq n$ $Y_i = \times$ if $y_i \neq y_{i-1}$ for $1 < i \leq n$ $Y_1 = y_1$ if $y_1 = D_0$ $Y_1 = \times$ if $y_1 \neq D_0$
8	PH(Load or Lold)	$Y_i = D_i$ if $D_i = y_i$ for $1 \leq i \leq n$ $Y_i = \times$ if $D_i \neq y_i$ for $1 \leq i \leq n$
9	LP(Left or Load)	$Y_i = D_i$ if $D_i = y_{i-1}$ for $1 \leq i \leq n$ $Y_i = \times$ if $D_i \neq y_{i-1}$ for $1 < i \leq n$ $Y_1 = D_0$ if $D_0 = D_1$ $Y_1 = \times$ if $D_0 \neq D_1$

FIGURE 12

Algorithm Number	Algorithm	Algorithm Equations
10	RP(Right or Load)	$Y_i = D_i$ if $D_i = y_{i+1}$ for $1 \leq i < n$ $Y_i = \times$ if $D_i \neq y_{i+1}$ for $1 \leq i < n$ $Y_n = D_{n+1}$ if $D_{n+1} = D_n$ $Y_n = \times$ if $D_{n+1} \neq D_n$
11	RLP(Right or Left or Load)	$Y_i = D_i$ if $D_i = y_{i+1} = y_{i-1}$ for $1 < i < n$ $Y_i = \times$ otherwise for $1 < i < n$ $Y_1 = D_0$ if $D_0 = D_1 = D_2$ $Y_1 = \times$ otherwise $Y_n = D_{n+1}$ if $D_{n+1} = D_n = D_{n-1}$ $Y_n = \times$ otherwise
12	LPH(Left or Load or Hold)	$Y_i = D_i$ if $D_i = y_i = y_{i-1}$ for $1 < i \leq n$ $Y_i = \times$ otherwise for $1 < i \leq n$ $Y_1 = D_0$ if $D_0 = D_1 = y_1$ $Y_1 = \times$ otherwise
13	RPH(Right or Load or Hold)	$Y_i = D_i$ if $D_i = y_i = y_{i+1}$ for $1 \leq i < n$ $Y_i = \times$ otherwise for $1 \leq i < n$ $Y_n = D_{n+1}$ if $D_{n+1} = y_n = D_n$ $Y_n = \times$ otherwise

FIGURE 12. (cont'd)

Algorithm Number	Algorithm	Algorithm Equations
14	RLPH (Right or Left or Hold)	$Y_i = D_i$ if $D_i = y_i = y_{i+1} = y_{i-1}$ for $1 < i < n$ $Y_i = \times$ otherwise for $1 < i < n$ $Y_1 = D_0$ if $D_0 = D_1 = y_1 = y_2$ $Y_1 = \times$ otherwise $Y_n = D_{n+1}$ if $D_{n+1} = D_n = y_n = y_{n-1}$ $Y_n = \times$ otherwise
15	HC*	$Y_i = 0$ if $y_i = 0$ else $Y_i = \times$
16	PC	$Y_i = 0$ if $D_i = 0$ else $Y_i = \times$
17	LC	$Y_i = 0$ if $y_{i-1} = 0$, $1 < i \leq n$, else $Y_i = \times$ $Y_1 = 0$ if $D_0 = 0$, else $Y_1 = \times$
18	RC	$Y_i = 0$ if $y_{i+1} = 0$, $1 \leq i < n$, else $Y_i = \times$ $Y_n = 0$ if $D_{n+1} = 0$ else $Y_n = \times$
19	RHC	$Y_i = 0$ if $y_{i+1} = y_i = 0$, $1 \leq i < n$, else $Y_i = \times$ $Y_n = 0$ if $D_{n+1} = y_n = 0$, else $Y_n = \times$
20	LHC	$Y_i = 0$ if $y_{i-1} = y_i = 0$, $1 < i \leq n$, else $Y_i = \times$ $Y_1 = 0$ if $D_0 = y_1 = 0$, else $Y_1 = \times$

FIGURE 12. (cont'd)

*Note that the procedure QC, where Q is any algorithm not containing C, can be computed by first computing Q and then changing all 1's in the result to ×'s.

Algorithm Number	Algorithm	Algorithm Equations
21	LPC	$\begin{cases} Y_i = 0 \text{ if } y_{i-1} = D_i = 0,\ 1 < i \leq n, \text{ else } Y_i = \times \\ Y_1 = 0 \text{ if } D_0 = D_1 = 0, \text{ else } Y_1 = \times \end{cases}$
22	RLPHC	$\begin{cases} Y_i = 0 \text{ if } y_{i-1} = y_i = y_{i+1} = D_i = 0,\ 1 < i < n, \text{ else } Y_i = \times \\ Y_1 = 0 \text{ if } y_1 = y_2 = D_0 = 0 \text{ else } Y_1 = \times \\ Y_n = 0 \text{ if } y_n = y_{n-1} = D_{n+1} = 0 \text{ else } Y_n = \times \end{cases}$
23	PHC	$\{ Y_i = 0 \text{ if } D_i = y_i = y_{i+1} = 0,\ 1 \leq i \leq n, \text{ else } Y_i = \times$
24	RPHC	$\begin{cases} Y_i = 0 \text{ if } D_i = y_i = y_{i+1} = 0,\ 1 \leq i < n, \text{ else } Y_i = \times \\ Y_n = 0 \text{ if } D_n = D_{n+1} = y_n = 0, \text{ else } Y_n = \times \end{cases}$
25		$\begin{cases} Y_i = 0 \text{ if } D_i = y_i = y_{i-1} = 0,\ 1 < i \leq n, \text{ else } Y_i = \times \\ Y_1 = 0 \text{ if } D_1 = D_0 = y_1 = 0 \text{ else } Y_1 = \times \end{cases}$

FIGURE 12. (end)

The tables of Figures 11 and 12 can be used to perform implication. Implication may be considered as consisting of forward implication (in which the values of the control inputs $\underset{\sim}{x}$ and $\underset{\sim}{D}$ are used to determine $\underset{\sim}{Y}$) or backwards implication (in which the value of $\underset{\sim}{Y}$ and $\underset{\sim}{y}$ and $\underset{\sim}{D}$ are used to determine the value of the control inputs or the value of $\underset{\sim}{Y}$ and the control inputs are used to determine $\underset{\sim}{D}$ and $\underset{\sim}{y}$). Implication should be performed as alternating forward implication and backward implication until both stabilize.

Let $B = \{H,C,P,L,R\}$ be the set of five primitive functional algorithms of the shift register. From the values of the control inputs (Z,C,S_1,S_0) and the table of Figure 11, a set of possible algoritems $B_1 \subseteq B$ is determined. Similarly, from the values of $\underset{\sim}{Y}$, $\underset{\sim}{y}$ and $\underset{\sim}{D}$ another set of possible algorithms $B_2 \subseteq B$ is determined. Specifically,

$$H \in B_2 \quad \text{iff} \quad \sum_{i=1}^{n} (Y_i \oplus y_i) \neq 1 \quad ;$$

$$P \in B_2 \quad \text{iff} \quad \sum_{i=1}^{n} (Y_i \oplus D_i) \neq 1 \quad ;$$

$$L \in B_2 \quad \text{iff} \quad \sum_{i=2}^{n} (Y_i \oplus y_{i-1}) + (Y_1 \oplus D_0) \neq 1 \quad ;$$

$$R \in B_2 \quad \text{iff} \quad \sum_{i=1}^{n-1} (Y_i \oplus y_{i+1}) + (Y_n \oplus D_{n+1}) \neq 1 \quad ;$$

$$C \in B_2 \quad \text{iff} \quad \sum_{i=1}^{n-1} Y_i \neq 1 \quad .$$

The actual algorithm must be in the set $B_1 \cap B_2$. From this set and the table of Figure 11, using cubical intersection, additional values of the control inputs may be implied. From the algorithm table of Figure 12, additional values of the outputs $\underset{\sim}{Y}$ may then be implied. The algorithm table can also be used to determine implied values of $\underset{\sim}{y}$ and $\underset{\sim}{D}$ using the concept of inverse algorithms. That is, if

$$\underset{\sim}{Y} = g(\underset{\sim}{y})$$

then

$$\underset{\sim}{y} = g^{-1}(\underset{\sim}{Y}) \quad .$$

Fortunately, the inverse relationships for the shift register are simple.

$$P^{-1} = P, \text{ (i.e. } \underset{\sim}{Y} = P(\underset{\sim}{D}) \text{ , } \underset{\sim}{D} = P^{-1}(\underset{\sim}{Y}) = P(\underset{\sim}{Y})$$

$$H^{-1} = H,$$

$$L^{-1} = R,$$

$$R^{-1} = L.$$

However, C^{-1} is undefined (thus nothing is implied on $\underset{\sim}{y}$ by $\underset{\sim}{Y}$ for the clear operation). The following example illustrates the determination of implication, using the tables of Figures 11 and 12.

EXAMPLE 6:

Let

$$(Z,C,S_1,S_0) = (0,1,1,x)$$

$$\underset{\sim}{Y} = (Y_1,Y_2,Y_3,Y_4) = (x,0,x,x)$$

$$\underset{\sim}{y} = (y_1,y_2,y_3,y_4) = (0,x,1,x)$$

and

$$\underset{\sim}{D} = (D_0,D_1,D_2,D_3,D_4,D_5) = (x,x,x,x,x,x) .$$

Then from Figure 11 and the cube $0\ 1\ 1 \times RH$ we conclude that

$$B_1 = \{R,H\} ,$$

and from Figure 12,

$$B_2 = \{L,H,P,C\} .$$

Thus

$$B_1 \cap B_2 = H .$$

From the table of Figure 11, $S_0 = 1$ is implied. For both H and H^{-1}, $Y_i = y_i$. Thus,

$$Y_1 = y_1 = 0,\ Y_3 = y_3 = 1 \quad \text{and} \quad y_2 = Y_2 = 0$$

are implied.

A formal procedure for performing implication, using the tables of Figures 11 and 12 is as follows:

Procedure 5.

(1) From the previously specified values of (Z,C,S_1,S_0) using cubical intersection on the table of Figure 11, determine a

UNION of possible algorithms. Let B_1 be the set of simple algorithms contained in the UNION of algorithms so determined.

(2) From $\underset{\sim}{Y}$, $\underset{\sim}{y}$ and $\underset{\sim}{D}$ and the algorithms of Figure 12, determine a set B_2 of possible algorithms.

(3) Define $B = B_1 \cap B_2$. Using the table of Figure 12 and the concept of inverse algorithms, define additional values of $\underset{\sim}{y}$, $\underset{\sim}{Y}$ and $\underset{\sim}{D}$;

$$\underset{\sim}{Y} = B(\underset{\sim}{y},\underset{\sim}{D}) \ ,$$

$$\underset{\sim}{y} = B^{-1}(\underset{\sim}{Y}) \ ,$$

$$D = B^{-1}(\underset{\sim}{Y}) \ .$$

(4) Use cubical intersection on the table of Figure 11 to define additional values of (Z,C,S_1,S_0) from B. □

3.1.2 Equation Approach

The behavior of the bidirectional shift register can also be defined by the following single canonical equation:

$$Y_i = \bar{Z}\,\bar{C}\,y_i + \bar{Z}\,C[S_1\,S_0\,y_i + \bar{S}_1\,\bar{S}_0\,D_i + \bar{S}_1\,S_0\,y_{i-1} + S_1\,\bar{S}_0\,y_{i+1}]$$

for $1 \leq i \leq n$; where

$$y_0 = D_0$$

and

$$y_{n+1} = D_{n+1} \quad .$$

This equation actually represents a set of n simultaneous equations. The solution of this set of equations can be used to determine implication.

EXAMPLE 7:

We will consider the same conditions as in Example 6. Substituting in the canonical equation for all signals which have values of 0 or 1, we obtain the following set of simultaneous equations:

$$Y_1 = 0 \quad ;$$

$$0 = Y_2 = S_0\, y_2 + \bar{S}_0 = y_2 + \bar{S}_0 \quad ;$$

$$Y_3 = S_0 + \bar{S}_0\, y_4 \quad ;$$

$$Y_4 = S_0\, y_4 + \bar{S}_0\, D_5 \quad .$$

From the second equation, $y_2 = 0$, $S_0 = 1$, is implied. From the first equation $Y_1 = 0$ is implied. From equation 3 with $S_0 = 1$, $Y_3 = 1$ is implied. These results are identical with those obtained previously in Example 6.

3.2 D-Drive for the Shift Register

This problem consists of propagating D's from the control signals, the inputs $\underset{\sim}{Q}$ and/or the state variables $\underset{\sim}{y}$ to the outputs $\underset{\sim}{Y}$ by selecting appropriate input line values (0,1).

We shall consider the following two cases:

(A) Single-Time Frame D-Drive

Determine all input sequences of length 1 which can propagate a D or $\bar{D}$ to any output.

(B) Multiple-Time Frame D-Drive

Determine all input sequences (of any length) which can propagate a D or $\bar{D}$ to a specific output Y_i.

3.2.1 Single-Time Frame D-Drive

When signals can assume the values D and $\bar{D}$ the implied signal values can be determined from the tables of Figures 11 and 12 by a process of composition.* That is, the normal machine executes algorithm B_N resulting in outputs $B_N(\underset{\sim}{y}_N)$. The faulty machine executes algorithm B_F on word $\underset{\sim}{y}_F$, resulting in $B_F(\underset{\sim}{y}_F)$. Both

---

*The concept of composition can also be applied to determine implication of D's and $\bar{D}$'s.

$B_N(\underset{\sim}{y}_N)$ and $B_F(\underset{\sim}{y}_F)$ can be computed from the tables of Figures 11 and 12 and hence the composite value $B_N(\underset{\sim}{y}_N)/B_F(\underset{\sim}{y}_F)$ can be computed. Alternatively, we could define a set of composite generalized algorithms which could be applied to 5-valued signals. However, the complexity of these algorithms would be greatly increased. For the special case $B_N = B_F$ the basic algorithms for the shift register can be easily extended to 5-valued signals.

The determination of inputs which propagate a D to an output of the shift register can be computed from the canonical set of simultaneous equations, or can be specified by a canonical set of propagation D-cubes such as those in Figure 13 which specify the propagation of a single D signal to an output of the shift register. These D-cubes can be derived in a manner similar to the D-algorithm using the tables of Figures 11 and 12. In using the D-cubes of Figure 13, it should be recalled that $y_{n+1} = D_{n+1}$ and $y_0 = D_0$.

EXAMPLE 8:

We wish to specify input values to propagate a D from S_1 to $\underset{\sim}{Y}$ with initial state

$$\underset{\sim}{y} = (y_1, y_2, y_3, y_4) = (0,0,0,1)$$

and

$$\underset{\sim}{D} = (D_0, D_1, D_2, D_3, D_4, D_5)$$
$$= (0,0,0,0,1,0) \ .$$

Composite Algorithm	Z	C	S_1	S_0	y_{i-1}	y_i	y_{i+1}	D_i	Y_i	Cube Number
C/H	D	0	-	-	-	1	-	-	$\bar{D}$	1
C/H	D	-	1	1	-	1	-	-	$\bar{D}$	2
C/P	D	1	0	0	-	-	-	1	$\bar{D}$	3
C/L	D	1	0	1	1	-	-	-	$\bar{D}$	4
C/R	D	1	1	0	-	-	1	-	$\bar{D}$	5
P/H	0	D	0	0	-	0	-	1	D	6
P/H	0	D	0	0	-	1	-	0	$\bar{D}$	7
L/H	0	D	0	1	1	0	-	-	D	8
L/H	0	D	0	1	0	1	-	-	$\bar{D}$	9
R/H	0	D	1	0	-	0	1	-	D	10
R/H	0	D	1	0	-	1	0	-	$\bar{D}$	11
H/L	0	1	D	1	1	0	-	-	$\bar{D}$	12
H/L	0	1	D	1	0	1	-	-	D	13
R/P	0	1	D	0	-	-	1	0	D	14
R/P	0	1	D	0	-	-	0	1	$\bar{D}$	15
L/P	0	1	0	D	1	-	-	0	D	16
L/P	0	1	0	D	0	-	-	1	$\bar{D}$	17
H/R	0	1	1	D	-	0	1	-	$\bar{D}$	18
H/R	0	1	1	D	-	1	0	-	D	19
L/L	0	1	0	1	D	-	-	-	D	20
H/H	0	0	-	-	-	D	-	-	D	21
H/H	0	-	1	1	-	D	-	-	D	22
R/R	0	1	1	0	-	-	D	-	D	23
P/P	0	1	0	0	-	-	-	D	D	24

FIGURE 13. "SINGLE-D" PROPAGATION D-CUBES FOR BIDIRECTIONAL SHIFT REGISTER

From the cube 13 of Figure 13 we determine that by setting $Z = 0$, $C = 1$, $S_0 = 1$, then $Y_4 = D$. Similarly, from cubes 14 and 15, if $Z = 0$, $C = 1$, $S_0 = 0$, then $Y_4 = D$ and $Y_3 = D$. The same results could be obtained from the set of simultaneous equations derived from the canonical characteristic equation

$$Y_i = \bar{Z}\,\bar{C}\,y_i + \bar{Z}\,C[S_1\,S_0\,y_i + \bar{S}_1\,\bar{S}_0\,D_i + \bar{S}_1\,S_0\,y_{i-1} + S_1\,\bar{S}_0\,y_{i+1}] \cdot$$

To propagate a D from S_1 we obtain:

$$D(\bar{D}) = Y_i = \bar{Z}\,\bar{C}\,y_i + \bar{Z}\,C[D\,S_0\,y_i + \bar{D}\,\bar{S}_0\,D_i + \bar{D}\,S_0\,y_{i-1} + D\,\bar{S}_0\,y_{i+1}] \cdot$$

The solution of this equation requires $Z = 0$, $C = 1$ which simplifies the equation to

$$D(\bar{D}) = Y_i = D\,S_0\,y_i + \bar{D}\,\bar{S}_0\,D_i + \bar{D}\,S_0\,y_{i-1} + D\,\bar{S}_0\,y_{i+1} \cdot$$

If $S_0 = 1$,

$$Y_i = D\,y_i + \bar{D}\,y_{i+1} \cdot$$

For the initial state

$$\underset{\sim}{y} = (0,0,0,1) \quad \text{and} \quad \underset{\sim}{D} = (0,0,0,0,1,0)$$

a D is propagated to Y_4. If $S_0 = 0$,

$$Y_i = \bar{D}\, D_i + D\, y_{i+1}$$

and

$$Y_4 = \bar{D}, \quad Y_3 = D \ .$$

A formal procedure for single-time-frame D-drive is as follows:

Procedure 6.

(1) Generate a propagation D-cube corresponding to the D (or $\bar{D}$) elements of the signal vector $(Z,C,S_1,S_0,\underset{\sim}{D},\underset{\sim}{y})$.

(2) Compute the cubical intersection of the partially specified vector $(Z,C,S_1,S_0,\underset{\sim}{D},\underset{\sim}{y})$ with the propagation D-cube generated in (1), if possible. If the intersection does not exist, generate a different propagation D-cube and repeat. □

3.2.2 Multiple Time Frame D-Drive for Shift Register

It may be desired to drive a D to a specific output Y_j. It is possible to take advantage of the functional behavior of the

device to define input sequences by which this can be achieved. We assume that error-free input sequences can be defined and it is desired to derive such sequences one at a time, shortest sequences first.*

We will generate sequences of generic signals, R, L and H. These must be translated into a sequence of signals on the appropriate control lines. Notice that the generic signal H can be translated into two different sets of signals on the control lines. For the shift register we assume we wish to drive a D from y_i to Y_j, where (without loss of generality) $|i - j| = k$. Consider the following generation rules for a unidirectional shift register. (Assume the shift register can shift in one direction, denoted by S, and that $i < j$).

Given solution number K, solution number K + 1 can be obtained by using one of the following three rules (only one will apply), using the following notation: $X^{\alpha} = X \cdots X$ (α times) and Σ = a sequence of S's or H's, or both.

---

*If this assumption is not valid the complexity of the multiple time frame approach is drastically increased and its use is probably unwarranted.

Rule 1:: Push the leftmost H to the left, one position:

$$\begin{aligned} K &: S^{\alpha} H \Sigma \\ K+1 &: S^{\alpha-1} H \Sigma \end{aligned} \quad ; \ \alpha \geq 1 \ .$$

Rule 2:: Start new sequence:

$$\begin{aligned} K &: H^{\alpha} S^{\beta} H \Sigma \\ K+1 &: S^{\beta-1} H^{\alpha+1} S \Sigma \end{aligned} \quad ; \ \alpha \geq 1, \beta \geq 1 \ .$$

Rule 3:: Creation of longer test sequence:

$$\begin{aligned} K &: H^{\gamma} S^{\beta} \\ K+1 &: S^{\beta-1} H^{\gamma+1} S^{1} \end{aligned} \quad ; \ \gamma \geq 0, \beta \geq 1 \ .$$

For bidirectional shift registers these rules must be modified by changing S to R for a right-shift, or by changing S to L for a left-shift. It is also necessary to add the following rule.

Rule 4:: Given a previously generated sequence:

$$K: \Sigma \ H \ H \ \Sigma$$

$$K+1: \Sigma \ L \ R \ \Sigma$$

for a net right-shift (or

$$K+1: \Sigma \ R \ L \ \Sigma$$

for a net left shift).

Note that now two rules may be applicable to a given sequence K and hence K + 1 is not uniquely defined. All applicable rules are applied at each step to guarantee generation of all sequences. Some sequences may be generated more than once.

EXAMPLE 9:

We will apply the previously presented rules to generate multiple time frame sequences which shift left three positions.

No.	Sequence	Derived By
(1)	LLL	
(2)	LLHL	Rule 3 from (1)
(3)	LHLL	Rule 1 from (2)
(4)	HLLL	Rule 1 from (3)
(5)	LLHHL	Rule 3 from (4)
(6)	LLRLL	Rule 4 from (5)
(7)	LHLHL	Rule 1 from (5)
(8)	HLLHL	Rule 1 from (7)
(9)	LHHLL	Rule 2 from (8)
(10)	LRLLL	Rule 4 from (9)
(11)	HLHLL	Rule 1 from (9)
(12)	HHLLL	Rule 1 from (11)
(13)	RLLLL	Rule 4 from (12)
(14)	LLRLHL	Rule 3 from (6)

These rules will succeed in generating all sequences, shortest sequence first, but may generate some invalid sequences (i.e. sequences which do not drive a D from y_i to y_j since the D may be shifted out of the register before reaching y_j).

It may be possible to derive better rules which eliminate this problem. However, in general, this may be quite difficult and it may be getter to generate the invalid sequences and eliminate them by performing a quick test on the prefix of the sequence.

A more interesting problem is the relative complexity of this multi-time frame approach to D-drive with the classical single-time frame approach* originally presented by Roth.

Consider the problem of driving a D 2-bits to the right in a bilateral shift register and the following (partial) tree of solutions. (Note: Circled nodes are terminal in Figure 14 below.) A single-time frame approach would first justify an R, then if successful a second R, which if unseccessful would then try an H, etc., and, in general, walk down the tree (using local backup) in a right-to-left manner.

The disadvantage of this approach is that it may find a solution which is very poor (i.e. requires more than the minimal number of time frames. For example it may find a solution RHHH $\cdots$ HR in the above example when the optimal solution is HRR.

*A multiple time frame input sequence is generated as a sequence of single time frame inputs.

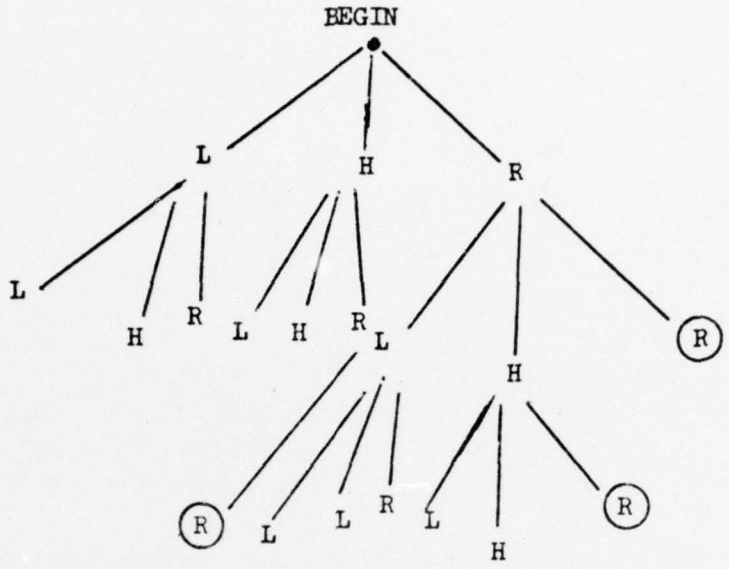


FIGURE 14

The advantage of this approach is that a failure at a high level non-terminal node will prune a large part of the tree. For example, if the first R fails, the right-most third of the tree is eliminated and many terminal nodes (such as R H R, R L R R, R H H R, etc.) need not be evaluated.

To improve the computational aspects of this technique, a dynamic bound on depth of tree search can be imposed. This would cause the high-level terminal nodes on the other parts of the tree to be considered before low-level terminal nodes on the right-most part of the tree. If m is the length of the shortest possible D-drive sequence, the bound might be $m + \alpha$ for some constant, positive integer $\alpha \geq 0$.

The multi-frame approach uses a set of rules to generate terminal nodes of the tree with all such nodes at level i being generated before such nodes at level j, $j > i$. For the purpose of this analysis, it is assumed that the control unit generates a potential solution and gives it to the main program for evaluation. If it is rejected another solution is generated.

To be computationally efficient the information acquired in evaluation must be utilized in determining whether or not to evaluate future potential solutions. Each rejected potential solution has associated with it a prohibited prefix (i.e. no actual solution will have this prefix). If these prohibited prefixes are stored in a table the potential solution can be matched against this table before evaluation. (This can be done either by the control unit or the main program.) A more elegant solution would be to have solution-generation rules which prevented the generation of potential solutions with prohibited prefixes. However, this would greatly increase the complexity of these rules and would not result in a significant savings since it appears that the complexity of evaluation of a potential solution is much greater than that of generation and table comparison of potential solutions.

3.3 Line Justification of Shift Register

Line justification may also be considered as a single-time frame or multiple-time frame problem.

3.3.1 Single-Time Frame Line Justification

For single-time frame line justification the same general procedure that was used for shift register implication (Section 3.1.1) can be applied. Let B_1 be the set of possible algorithms defined by the (partially unspecified) control inputs. Let B_2 be the set of possible algorithms defined by $\underset{\sim}{y}$, $\underset{\sim}{D}$ and $\underset{\sim}{Y}$. Compute $B_1 \cap B_2$ and define additional control inputs and values of $\underset{\sim}{y}$ and $\underset{\sim}{D}$ using the tables of Figures 11 and 12. Alternatively, the set of simultaneous equations defined by the canonical characteristic equation can be solved.

EXAMPLE 10:

Assume we wish to justify the outputs:

$$\underset{\sim}{Y} = (0,1,1,0) \text{ with } \underset{\sim}{D} = (x,x,x,x,x,x) \ ;$$
$$\underset{\sim}{y} = (x,0,x,x) \text{ and } (Z,C,S_1,S_0) = (0,1,x,x) \ .$$

Then

$$B_1 = \{P,L,R\} \text{ and } B_2 = \{P,L\} \ ,$$

$$B_1 \cap B_2 = \{P,L\} \ .$$

Justification Solution No. 1 - Algorithm P, $S_1 = S_0 = 0$, $\underset{\sim}{D} = (x,0,1,1,0,x)$.

Justification Solution No. 2 - Algorithm L, $S_0 = 1$, $S_1 = 0$.

$\underset{\sim}{y} = (\times,0,1,1)$, $\underset{\sim}{D} = (\times,\times,\times,\times,\times,0)$.

3.3.2 Multiple-Time Frame Line Justification

The same comments which were made for multiple-time frame D-drive are applicable here. The problem of interest is to develop rules which will generate all sequences which justify a specific register value. We will use the following notation:

R_a - shift right, shift in value a, $a = 0,1,\times$,

L_a - shift left, shift in value a, $a = 0,1,\times$,

$P_{abc,\ldots}$ - parallel load values $abc,\ldots;a,b,c,\varepsilon\ 0,1,\times$.

We will demonstrate the generation of the tree of all solutions for a specific register value for a 3-bit bidirectional shift register.

(Nondeterministic)* Line Justification Generation Rules

(1) Apply sequence P_{abc};

(2) Replace P_{abc} by $P_{bc\times} R_a$;

(3) Replace P_{abc} by $P_{\times ab} L_c$;

---

*Nondeterministic implies that if more than one rule can be applied, all such rules are applied (in some sequence).

(4) (a) Replace P_{abc} by $H\ P_{abc}$;

(b) Replace $P_{abc}\ \Sigma$ by $P_{abc}\ H\ \Sigma$ where Σ is not null;

(c) Replace R_a by $H\ R_a$;

(d) Replace $R_a\ \Sigma$ by $R_a\ H\ \Sigma$ where Σ is not null;

(e) Replace L_a by $H\ L_a$;

(f) Replace $L_a\ \Sigma$ by $L_a\ H\ \Sigma$ where Σ is not null.

□

These rules will generate a tree of all solutions. However, the same solution sequence may be generated in different ways.

EXAMPLE 11:

To justify the value $(y_3,y_2,y_1) = (0,1,1)$ in a 3-bit shift register, the following solution tree (to a depth of 3) is generated. The parenthetical numbers indicate the specific generation rule utilized..

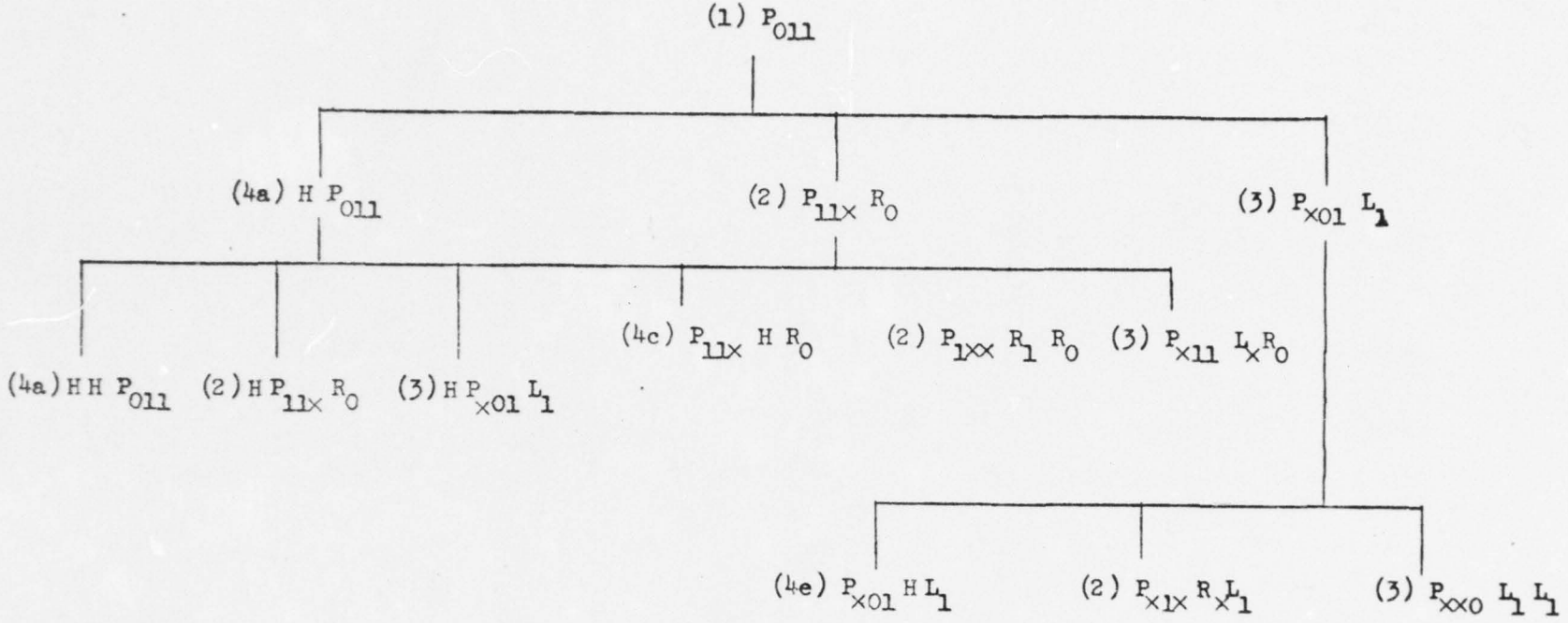

(1) P_{011}
(4a) H P_{011}
(2) $P_{11\times}$ R_0
(3) $P_{\times 01}$ L_1
(4a) H H P_{011}
(2) H $P_{11\times}$ R_0
(3) H $P_{\times 01}$ L_1
(4c) $P_{11\times}$ H R_0
(2) $P_{1\times\times}$ R_1 R_0
(3) $P_{\times 11}$ $L_\times$ R_0
(4e) $P_{\times 01}$ H L_1
(2) $P_{\times 1\times}$ $R_\times$ L_1
(3) $P_{\times\times 0}$ L_1 L_1

FIGURE 15

4. Up-Down Counter

In this section we will consider a general n-bit counter of the type shown in Figure 17. Under normal operation this device can perform four functional operations.

(U) Count up - the contents of the register are incremented by one;

(D) Count down - the contents of the register are decremented by one;

(L) Load - the contents of the register are set to the values of the data inputs $D_1,\ldots,D_n$; and

(H) Hold - the contents of the register are unchanged.

The functional behavior of this device is determined by the values of four control signals (C,U/D,G,L) as specified by the following table.

L	C	U/D	G	Algorithm
0	-	-	-	(L) Load
1	-	-	1	(H) Hold
1	1	-	0	(H) Hold
1	0	1	0	(U) Up
1	0	0	0	(D) Down

FIGURE 16

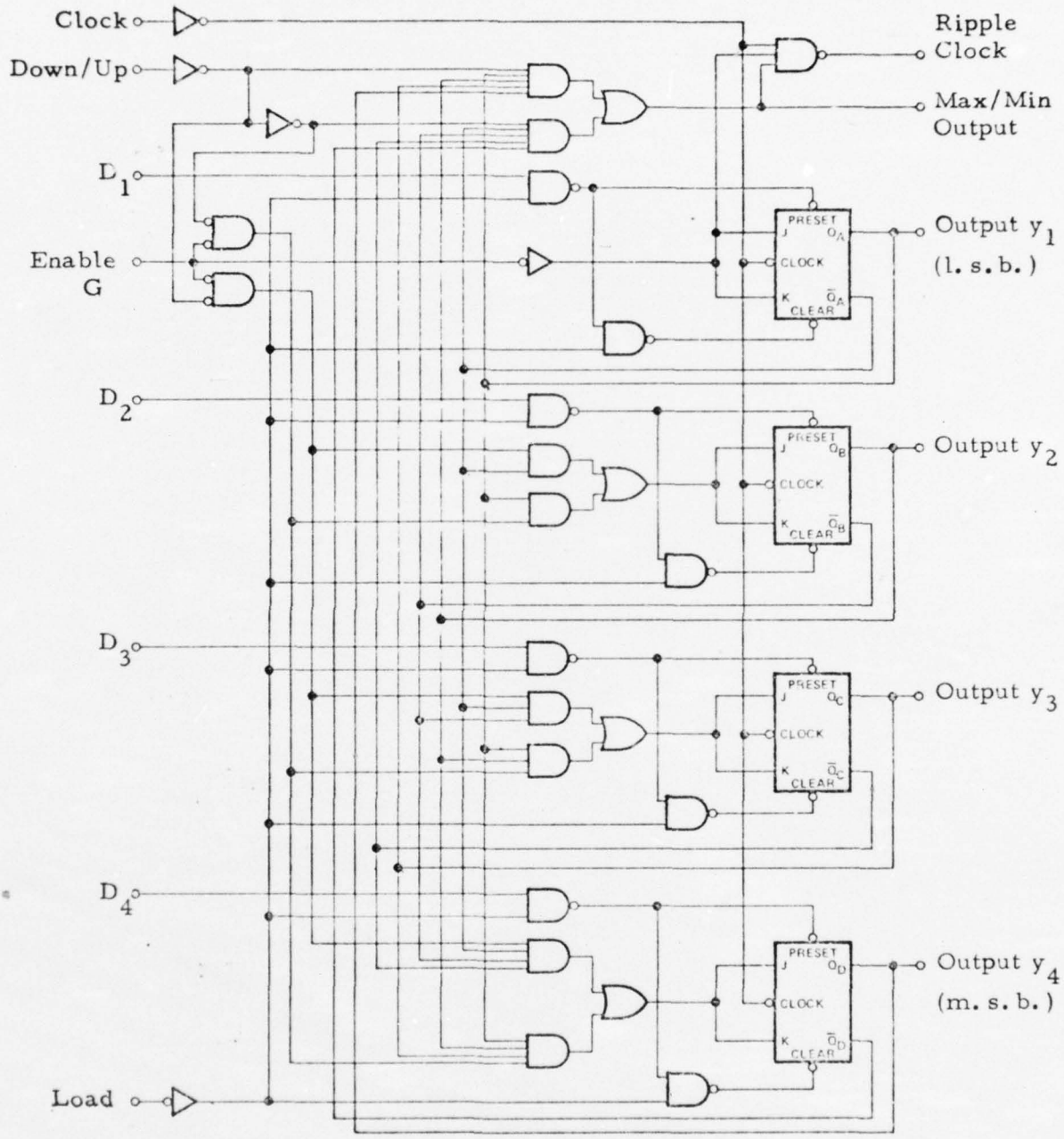

Clock
Down/Up
D1
Enable G
D2
D3
D4
Load
Ripple Clock
Max/Min Output
Output y1 (l. s. b.)
Output y2
Output y3
Output y4 (m. s. b.)
PRESET
CLOCK
CLEAR

Figure 17

4.1 Implication for UP/DOWN Counter

Implication can be performed using a tabular approach or an equation approach.

The tabular approach utilizes an input mapping table which specifies the functional behavior of the device for any values of the control inputs. Allowing inputs on the control signals to have the values $0,1,\times$, there are $(3)^4 = 81$ possible input conditions. The table maps each of these 81 input conditions into one of 15 possible algorithms which correspond to the UNION of the four basic algorithms previously listed. The input mapping table for the UP/DOWN Counter is shown in Figure 18.

The algorithms for the counter are more complex than for the bidirectional shift register and are defined as follows:

15 Counter Algorithms

(1) U (COUNT UP) - i) all bits to the left of the least significant 0 are unchanged;

ii) all bits (if any) to the right of and including the least significant 0 which are also to the left of and including the least significant $\times$ become $\times$;

iii) least significant 0 becomes 1 if it is to the right of least significant $\times$; and

C	U/D	G	L	Algorithms	No. of Input Conditions Uniquely Correct	
-	-	-	0	Load (L)	27	
×	×	×	1	Up - down or hold (UDH)	1	
-	-	1	1	Hold (H)	9	
1	-	-	1	Hold (H)	6	
0	×	0	1	Up - down (UD)	1	
0	1	0	1	Up (U)	1	
0	0	0	1	Down (D)	1	
×	1	0	1	Up or Hold (UH)	1	
×	0	0	1	Down or Hold (DH)	1	27
×	×	0	1	Up, down or Hold (UDH)	1	
0	0	×	1	UH	1	
0	1	×	1	DH	1	
0	×	×	1	UDH	1	
×	0	×	1	DH	1	
×	1	×	1	DH	1	
1	-	-	×	Hold or Load (HL)	9	
-	-	1	×	HL	6	
0	×	0	×	Up or down or Load (UDL)	1	
0	1	0	×	Up or Load (UL)	1	
0	0	0	×	Down or Load (DL)	1	
0	1	×	×	Up, Hold or Load (UHL)	1	
0	0	×	×	Down, Hold, Load (DHL)	1	27
0	×	×	×	Up, down, hold or Load (UDHL)	1	
×	0	0	×	DHL	1	
×	1	0	×	UHL	1	
×	×	0	×	UDHL	1	
×	0	×	×	DHL	1	
×	1	×	×	UHL	1	
×	×	×	×	UDHL	1	

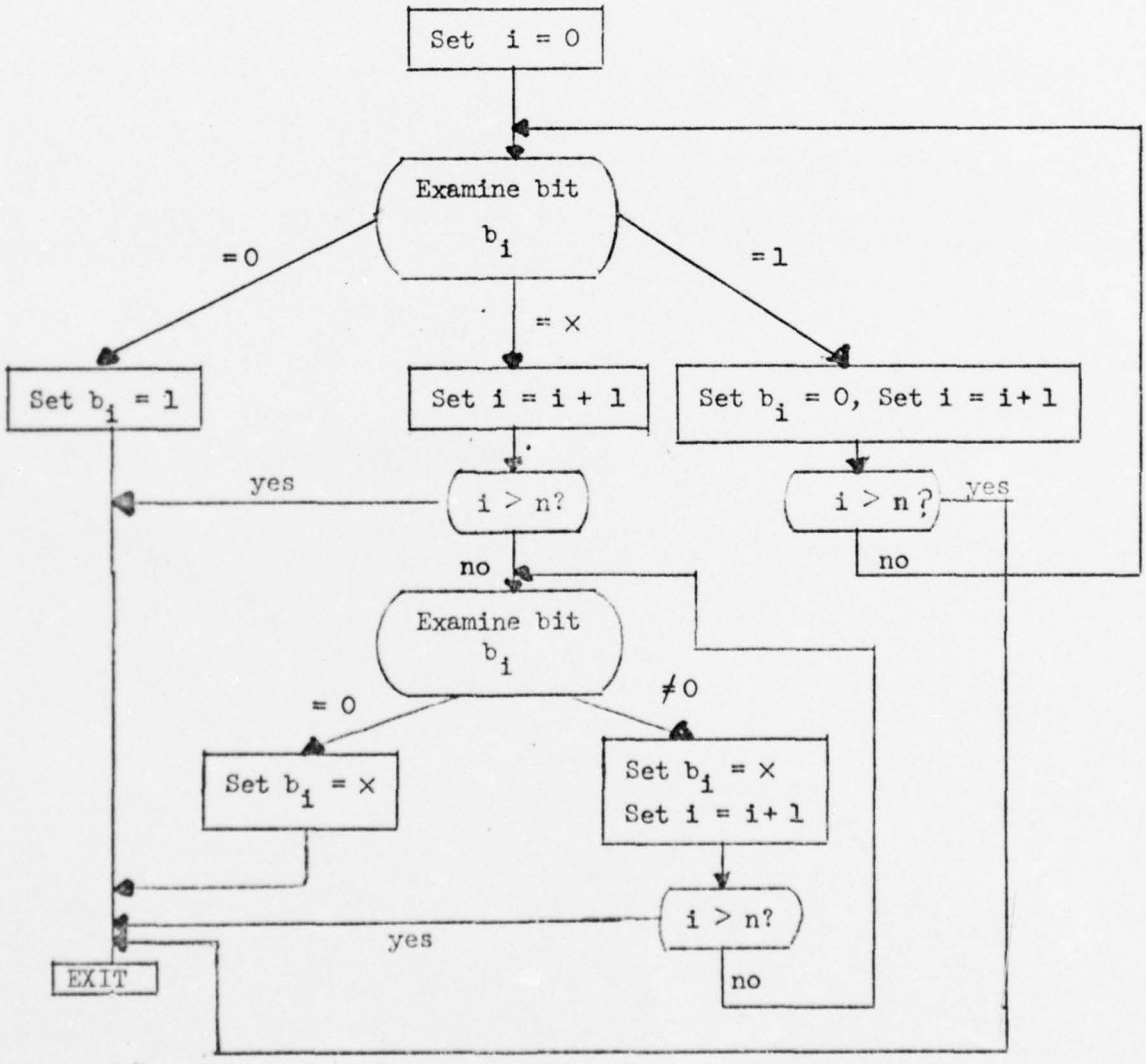
FIGURE 18

iv) a rightmost string of 1's (if any) become 0's.

EXAMPLE: $\underbrace{1\,0\,1\times 0\,1\,1}_{1\,0\,1\times 0\,1\,1}\quad \underbrace{0\times\times 1\times}_{\times\times\times\times\times}\quad \underbrace{1\,1\,1}_{0\,0\,0}$

by i) by ii) by iv)

A flow chart for this algorithm is shown in Figure 19, where n is the number of bits in the register.



** **

(2) D (Count down) - i) all bits to the left of the least significant 1 are unchanged;

ii) all bits (if any) to the right of and including the least significant 1's which are also to the left of and including the least significant ×'s become ×;

iii) least significant 1 becomes 0 if it is to the right of least significant ×; and

iv) a rightmost string of 0's (if any) become 1's.

Note duality of U and D algorithms

** **

(3) H (Hold) $Y_i = y_i; \quad 0 \leq i \leq n$

** **

(4) L (Load) $Y_i = D_i; \quad 0 \leq i \leq n$

** **

(5) HL (Hold or Load) - Compare Algorithm

Compare y_i, D_i if $D_i = y_i \Rightarrow Y_i = y_i$

if $D_i \neq y_i \Rightarrow Y_i = \times$

** **

(6) UH (Up or Hold) - All bits to the left of the least significant 0 are unchanged; all bits to the right of and including the least significant 0 become × .

** **

** ** ** *** **

(7) DH (Down or Hold) - All bits to the left of the least significant 1 are unchanged. All bits to the right of and including the least significant 1 become × .

** **

(8) UD (Up or Down) - i) All bits which are to the left of both the least significant 0 and the least significant 1 are unchanged;

ii) All bits to the right of and including the least significant 1 or the least significant 0 become × except bit b_0 which is complemented (if it was 0 or 1).

EXAMPLE:

0 × × × 1 × ⟶ × × × × × × (Rule ii))
0 × × × 1 1 1 ⟶ × × × × × × 0 (Rule ii))
0 0 1 × × ⟶ 0 × × × × (Rule i, Rule ii)

** **

(9) UL (Up or Load) 1) Perform U Algorithm;

2) then compare y_i with D_i. If $(D_i = y_i)$ then leave y_i unchanged. If $(D_i \neq y_i)$ then $y_i \longleftrightarrow \times$.

** **

** **

(10) DL (Down or Load)

1) Perform Down (D) Algorithm;

2) Then compare y_i with D_i. If $(D_i = y_i)$ then leave y_i unchanged. If $(D_i \neq y_i)$ then $y_i \longleftarrow \times$.

** **

(11) UDH (Up or Down or Hold)

1) All bits to the left of both the least significant 0 and least significant 1 are unchanged.

2) All other bits become $\times$.

Note: This is the same as UD algorithm except for bit b_0.

** **

(12) UDL (Up or Down or Load)

1) Do UD algorithm.

2) Then compare y_i with D_i, if $(y_i \neq D_i)$ set $y_i \leftarrow \times$.

** **

(13) DHL (Down or Hold or Load)

1) Do DH Algorithm;

2) Then compare y_i with D_i. If $(y_i \neq D_i)$ set $y_i \leftarrow \times$.

** **

(14) UHL (Up or Hold or Load)

1) Do UH algorithm.

2) Then compare y_i with D_i. If $(y_i \neq D_i)$ set $y_i \leftarrow \times$.

** **

** **

(15) UDHL (Up or Down or Hold or Load)

1) Do UDH algorithm.

2) Then compare y_i with D_i. If $(y_i \neq D_i)$ set $y_i \leftarrow \times$.

** **

These algorithms are independent of the specific implementation of the UP/DOWN COUNTER. The table mapping control inputs into algorithms will, in general, be implementation dependent.

Equation Approach:

The two tables just considered completely specify the behavior of the counter and can be used for implication, D-drive and line justification. Alternatively, the counter can be described by the following canonical characteristic equation:

$$Y_i = \underbrace{\bar{C}\,\bar{L}\,D_i}_{\text{Load}} + \underbrace{(L\,G + C)y_i}_{\text{Hold}} + \underbrace{\bar{C}\,\bar{L}\,\bar{G}\,U \prod_{\substack{\text{all} \\ j<i}} y_j}_{\text{Count Up}} + \underbrace{\bar{C}\,\bar{L}\,\bar{G}\,U \prod_{\substack{\text{all} \\ j<i}} \bar{y}_j}_{\text{Count Down}}$$

The table of Figure 18 and the counter algorithms can be used to perform implication in a manner similar to that used for the shift register.

Let $B = \{H, L, U, D\}$ be the set of four primitive functional algorithms of the counter. From the values of the control inputs (C, U/D, G, L) and the table of Figure 18, a set of possible algorithms $B_1 \subseteq B$ is determined. Similarly, from the values of $\underset{\sim}{Y}$, $\underset{\sim}{y}$ and $\underset{\sim}{D}$ another set of possible algorithms $B_2 \subseteq B$ is determined. Specifically,

$$H \in B_2 \quad \text{iff} \quad \sum_{\text{all } i} (Y_i \oplus y_i) \neq 1 \quad ;$$

$$L \in B_2 \quad \text{iff} \quad \sum_{\text{all } i} (Y_i \oplus D_i) \neq 1 \quad ;$$

$$U \in B_2 \quad \text{iff} \quad \sum_{\text{all } i} (U(\underset{\sim}{y}) \oplus Y_i) \neq 1 \quad ;$$

$$D \in B_2 \quad \text{iff} \quad \sum_{\text{all } i} (D(\underset{\sim}{y}) \oplus Y_i) \neq 1 \quad .$$

The actual algorithm must be in the set $B_1 \cap B_2$. From this set and the table of Figure 18, using cubical intersection, additional values of the control inputs may be implied. From the counter algorithms, additional values of the outputs $\underset{\sim}{Y}$ may then be implied. The counter algorithms can also be used to determine implied values of $\underset{\sim}{y}$ and $\underset{\sim}{D}$ using the concept of inverse algorithms. For the counter, the inverse relationships are as follows:

$$U^{-1} = D \ ;$$

$$D^{-1} = U \ ;$$

$$L^{-1} = L \ ;$$

$$H^{-1} = H \ .$$

The following example demonstrates both the tabular and equation approach to implication for the UP/DOWN Counter and illustrates the deficiencies of each.

EXAMPLE:

(a) Consider an 8-bit counter and suppose $(C, U/D, G, L) = (0, \times, 0, \times)$. From the table of Figure 18, the implied algorithm is UDL. If

$$\underset{\sim}{y} = (0\,1\,0\,1\,1\,0\,\times\,0)$$

and

$$\underset{\sim}{Q} = (0\,0\,1\,1\,0\,0\,0\,1)$$

then the implied state vector, determined from the counter algorithms is

$$(0\,\times\,\times\,1\,\times\,\times\,\times\,1) \ .$$

Using the equation approach we encounter a problem. We have:

$$Y_0 = \bar{C}\,\bar{L}\,D_0 + (L\,G + C)y_0 + \bar{C}\,L\,\bar{G}\,U \prod_{j<0} y_j + \bar{C}\,L\,\bar{G}\,\bar{U} \prod_{j<0} \bar{y}_j$$

$$= 1 \cdot \times \cdot 1 \ + \ \times \cdot 0 \cdot 0 \ + \ 1 \cdot \times \cdot 1 \cdot \times \cdot 1 \ + \ 1 \cdot \times \cdot 1 \cdot \times \cdot 1 \ = \ \times \ .$$

The correct answer (which was determined using the tabular approach) is $Y_0 = 1$. This problem is caused by the fact that $\bar{x} = x$.

(b) Consider a 6-bit counter and let

$$(C, U/D, G, L) = (0, x, x, x)$$

and

$$\underset{\sim}{Q} = (0,1,1,1,0,1) , \underset{\sim}{Y} = (x,1,0,x,x,x)$$

and

$$\underset{\sim}{y} = (0,0,1,x,x,x) .$$

From the table of Figure 18 $B_1 = \{U,D,H,L\}$ and from $\underset{\sim}{Y}$, $\underset{\sim}{y}$ and $\underset{\sim}{D}$, $B_2 = \{U\}$. Consequently,

$$B_1 \cap B_2 = \{U\}$$

and from Figure 18 the signal values $L = 1$, $G = 0$, and $U = 1$ are implied.

$$\underset{\sim}{y} = U^{-1}(\underset{\sim}{Y}) = \dot{D}(\underset{\sim}{Y}) = (x,x,x,x,x,x) ,$$

(i.e. - nothing new implied.)

$$\underset{\sim}{Y} = U(\underset{\sim}{y}) = (0,x,x,x,x,x)$$

implies $Y_5 = 0$. After completion of implication:

$$Y = (0,1,0,x,x,x) , y = (0,0,1,x,x,x) ;$$
$$(C, U/D, G, L) = (0, 1, 0, 1) .$$

Note that the algorithms do not enable us to correctly infer that $y_2 = y_1 = y_0 = 1$ and $Y_2 = Y_1 = Y_0 = 0$.

We will now attempt to solve the same problem using the equation approach. As we shall see, although this requires the solution of a set of simultaneous equations, it may lead to a less pessimistic answer than the algorithm approach just considered.

$$\begin{aligned} Y_3 = 0 = \bar{C}\,\bar{L}\,D_3 + (L\,G + C)y_3 + \bar{C}\,L\,\bar{G}\,U\,y_0\,y_1\,y_2\,\bar{y}_3 \\ + \bar{C}\,L\,\bar{G}\,\bar{U}\,\bar{y}_0\,\bar{y}_1\,\bar{y}_2\,\bar{y}_3 \\ = \bar{L} + L\,G + 0 + 0 \\ = \bar{L} + G \Rightarrow L = 1\,,\ G = 0\ ; \end{aligned}$$

$$\begin{aligned} Y_4 = 1 = \bar{C}\,\bar{L}\,D_4 + (L\,G + C)y_4 + \bar{C}\,L\,\bar{G}\,U\,y_0\,y_1\,y_2\,y_4\,y_4 \\ + \bar{C}\,L\,\bar{G}\,\bar{U}\,\bar{y}_0\,\bar{y}_1\,\bar{y}_2\,\bar{y}_3\,\bar{y}_4 \\ = 0 + 0 + U\,y_0\,y_1\,y_2 + 0 \\ \Rightarrow U = 1\,,\ y_0 = y_1 = y_2 = 1\ . \end{aligned}$$

From the equations for Y_0, Y_1 and Y_2 we conclude that:

$$Y_0 = Y_1 = Y_2 = 0\ .$$ □

In part (b) of the previous example the difficulty encountered in the Algorithm-Table approach reflected the fact that even though the algorithm U was implied, the effects of this implication on $\underset{\sim}{Y}$ was determined only by examining $\underset{\sim}{y}$ and vice-versa. The implications of U and the combined values of $\underset{\sim}{Y}$, $\underset{\sim}{y}$ are not considered.

4.2 D-Drive for the Counter

This problem consists of propagating error signals, D or $\bar{D}$, from the control signals, the data inputs $\underset{\sim}{x}$ and the state variables $\underset{\sim}{y}$ to the outputs $\underset{\sim}{Y}$. We shall again consider both single time frame and multiple time frame D-drive.

4.2.1 Single Time Frame D-Drive

When signals can assume the values D and $\bar{D}$ the implied signal values can be determined from the table of Figure 18 and the counter algorithms by the composition process.

The determination of inputs which propagate a D to an output of the counter can be computed from the canonical set of simultaneous equations, or can be specified by a "canonical" set of propagation D-cubes such as those in Figure 20 which specify the propagation of a single D signal to an output of the counter.

Composite Algorithm	L	C	U/D	G	D_i	y_i	$\prod_{j<i} y_j$	$\prod_{j<i} \bar{y}_j$	Y_i
H/L	D	-	-	1	1	0			D
H/L	D	-	-	1	0	1			D
H/L	D	1	-	-	1	0			$\bar{D}$
H/L	D	1	-	-	0	1			D
U/L	D	0	1	0	0	0	1		D
U/L	D	0	1	0	0	1	0		D
U/L	D	0	1	0	1	1	1		$\bar{D}$
U/L	D	0	1	0	1	0	0		$\bar{D}$
H/U	1	D	1	0	-	0	1		$\bar{D}$
H/U	1	D	1	0	-	1	1		D
H/D	1	D	0	0	-	0	-	1	$\bar{D}$
H/D	1	D	0	0	-	1	-	1	D
U/D	1	0	D	0	-	0	1	-	D
U/D	1	0	D	0	-	0	-	1	$\bar{D}$
U/D	1	0	D	0	-	1	1	-	$\bar{D}$
U/D	1	0	D	0	-	1	-	1	D
H/D	1	0	0	D	-	0	-	1	$\bar{D}$
H/D	1	0	0	D	-	1	-	1	D
H/U	1	0	1	D	-	0	1		$\bar{D}$
H/U	1	0	1	D	-	1	1		D
L/L	0	-	-	-	D				D
H/H	1	-	-	1	-	D	-	-	D
H/H	1	1	-	-	-	D	-	-	D
U/U	1	0	1	0	-	D	0		D
U/U	1	0	1	0	-	D	1		$\bar{D}$
D/D	1	0	0	0	-	D		0	D
D/D	1	0	0	0	-	D		1	$\bar{D}$

FIGURE 20

4.2.2 Multiple Frame D-Drive

In this section we will consider the problem of driving a D from a bit y_i of the counter at time t to a bit y_j of the counter. We wish to generate one at a time, all error-free input sequences, which can accomplish this.

In the case of the counter the input sequences will consist of U (UP), D (DOWN) and H (HOLD) symbols. (Loads cannot occur since they will wipe out the D in the register.)

Let S be a sequence and $|U|_S$ be the number of U's in S; and $|D|_S$ be the number of D's in S, for all k which satisfy the following constraint:

> Let the state of the good machine represent the binary number N_1.
> Let the state of the faulty machine represent the binary number N_2.
> Then, k must be such that the binary numbers $N_1 + k$ and $N_2 + k$ differ in bit j (i.e.
>
> $$b_j(N_2 + k) \neq b_j(N_1 + k)$$
>
> where $b_i(x)$ is the value of the i-th bit in the binary representation of the number x).

Of course, the interesting problem is how to generate these sequences in a computationally efficient manner. This problem is almost identical to but actually somewhat easier than the same problem for shift registers.

(1) For a shift register, a sequence S consists of R's (Right Shifts), L's (Left Shifts) and Hold's (H). We wish to generate all sequences such that

$$|R|_S - |H|_S = k \ .$$

For D-drive in this case, there is a unique value of k which is quite easily determined in contrast to the counter which may have many values of k.

(2) For the shift register case some sequences generated may not be valid because they may contain a prefix which causes the D to be lost (e.g. a D in y_1 is lost if S has the prefix of R). However, for the counter this never happens since for any k:

$$(N_1 + k) \bmod 2^n = (N_2 + k) \bmod 2^n$$
$$\text{iff } (N_1) \bmod 2^n = (N_2) \bmod 2^n.$$

Of course, the interesting problem is how to generate these sequences in a computationally efficient manner. This problem is almost identical to but actually somewhat easier than the same problem for shift registers.

(1) For a shift register, a sequence S consists of R's (Right Shifts), L's (Left Shifts) and Hold's (H). We wish to generate all sequences such that

$$|R|_S - |H|_S = k \ .$$

For D-drive in this case, there is a unique value of k which is quite easily determined in contrast to the counter which may have many values of k.

(2) For the shift register case some sequences generated may not be valid because they may contain a prefix which causes the D to be lost (e.g. a D in y_1 is lost if S has the prefix of R). However, for the counter this never happens since for any k:

$$(N_1 + k) \bmod 2^n = (N_2 + k) \bmod 2^n$$
$$\text{iff } (N_1) \bmod 2^n = (N_2) \bmod 2^n.$$

(3) The problems as to relative efficiency of walking down a tree of solutions as opposed to hopping between terminal branches seems relatively the same.

Thus, the same rules as for the shift register case can be used to generate the multiple time frame propagation sequences for the counter if the value of each bit of the counter is 0, 1, D, $\bar{D}$. The presence of an $\times$ in bit $k < i$ of the counter may make it impossible to propagate a D from y_i to y_j.

4.3 Line Justification for UP/DOWN Counter

4.3.1 Single Time Frame Line Justification

The same general procedure that was used for counter implication can be applied. Let B_1 be the set of possible algorithms defined by the control input values. Let B_2 be the set of possible algorithms defined by the values of $\underset{\sim}{y}$, $\underset{\sim}{D}$ and $\underset{\sim}{Y}$. Compute $B_1 \cap B_2$ and define additional control inputs and values of $\underset{\sim}{y}$ and $\underset{\sim}{D}$ using the table of Figure 19, and the counter algorithms as specified in the following general procedure.

Procedure:

(Single Time Frame Line Justification for Counter)

(1) Generate $B_1 \cap B_2$.

(2) Select a primitive algorithm α in $B_1 \cap B_2$ if one exists. (If none, justification is impossible.)

(3) Specify inputs $(C, U/D, G, L)$ to appropriate values for α using cubical intersection on table of Figure 19.

(4) Specify $\underset{\sim}{y} = \alpha^{-1}(\underset{\sim}{Y})$ unless $\alpha = L$, in which case $\underset{\sim}{D} = \alpha^{-1}(\underset{\sim}{Y})$. (Note: $H^{-1} = H$, $L^{-1} = L$, $U^{-1} = D$ and $D^{-1} = U$.)

(5) Normal backtrack can be used to generate other possible jutifications. □

Alternatively, the set of simultaneous equations defined by the canonical characteristic equation for the counter can be solved.

EXAMPLE: Consider a 4-bit counter and assume

$$\underset{\sim}{Y} = (0,1,0,1), \quad \underset{\sim}{y} = (x,x,x,0), \quad \underset{\sim}{D} = (x,x,0,1)$$

and

$$(C, U/D, G, L) = (x,x,x,x) \ .$$

Then

$$B_1 = \{U,D,H,L\} \text{ and } B_2 = \{U,D,L\} = B_1 \cap B_2 \ .$$

Justification No. 1:

Select $\alpha = U$. From Figure 19, $C = 0$, $U/D = 1$, $G = 0$ and $L = 1$. Furthermore, $\underset{\sim}{x} = U^{-1}(\underset{\sim}{Y}) = D(0\ 1\ 0\ 1)$, which from the Counter Algorithm for D implies $\underset{\sim}{y} = (0\ 1\ 0\ 0)$.

Justification No. 2:

Select $\alpha = D$. From Figure 19, $C = 0$, $U/D = 0$, $G = 0$ and $L = 1$. Furthermore, $\underset{\sim}{y} = D^{-1}(\underset{\sim}{Y}) = U(0\ 1\ 0\ 1)$, which from the Counter Algorithm for UP implies $\underset{\sim}{y} = (0\ 1\ 1\ 0)$.

Justification No. 3:

Select $\alpha = L$. From Figure 19 this implies $L = 0$. Furthermore,

$$D = L^{-1}(\underset{\sim}{Y}) = L(0\ 1\ 0\ 1) = (0\ 1\ 0\ 1) \ .$$

4.3.2 Multiple Time Frame Line Justification

In this case we may wish to specify all input sequences which can be used to justify a counter state $\underset{\sim}{Y}$ from some past counter state $\underset{\sim}{y}$. The set of all such input sequences can be expressed in the form of a tree. Note that if $\underset{\sim}{y}$ is completely unspecified (i.e. $y = (x,x,\ldots,x)$) then the justifying input sequence for the counter must

contain at least one LOAD (L). The number of possible justifying seuqences with L's is practically limitless. If $\underset{\sim}{y}$ is at least partially specified, then the set of input sequences which can justify $\underset{\sim}{Y}$ from $\underset{\sim}{y}$ (excluding those sequences containing L's) may be able to be expressed in terms of $U_S - D_S$ (the net count increment).

Without loss of generality consider a 3-bit counter and let L_{abc} denote a parallel load of the contents a b c into the register. The following rules can be used to generate all multi-time frame input sequences which results in contents a b c.

(1) L_{abc} ;

(2) Replace L_{abl} by the sequence L_{ab0} U;

(3) Replace L_{ab0} by L_{abl} D;

(4) Replace L_{a10} by L_{a01} U;

(5) Replace L_{a01} by L_{a1C} D;

(6) Replace L_{100} by L_{011} U;

(7) Replace L_{011} by L_{100} D; and

(8) Replace L_{abc} by H L_{abc} or L_{abc} H.

EXAMPLE:

Consider a 4-bit counter for which we wish to justify the contents $\underset{\sim}{Y} = (0 \times 0\,1)$. Figure 21 shows a partial tree of solutions and a directed state graph for this problem.

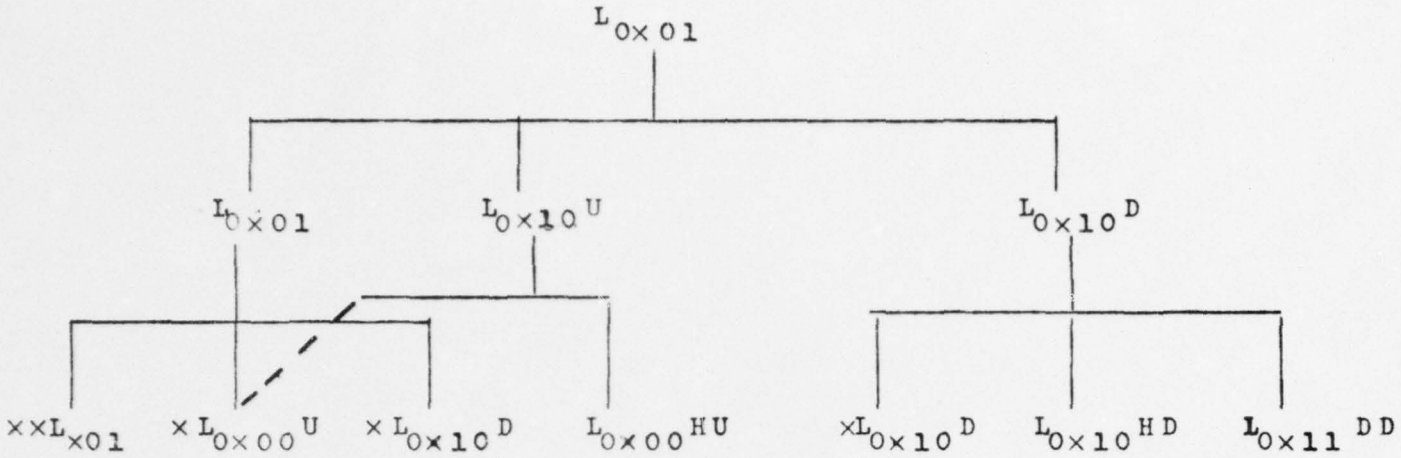


FIGURE 21(a):

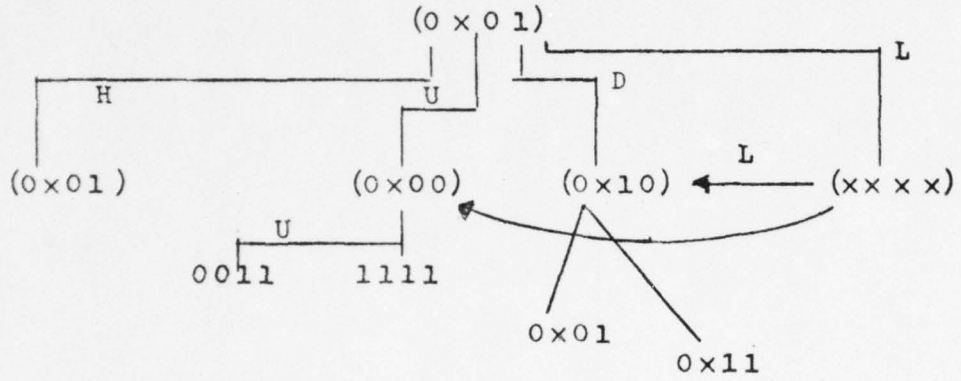


FIGURE 21(b)

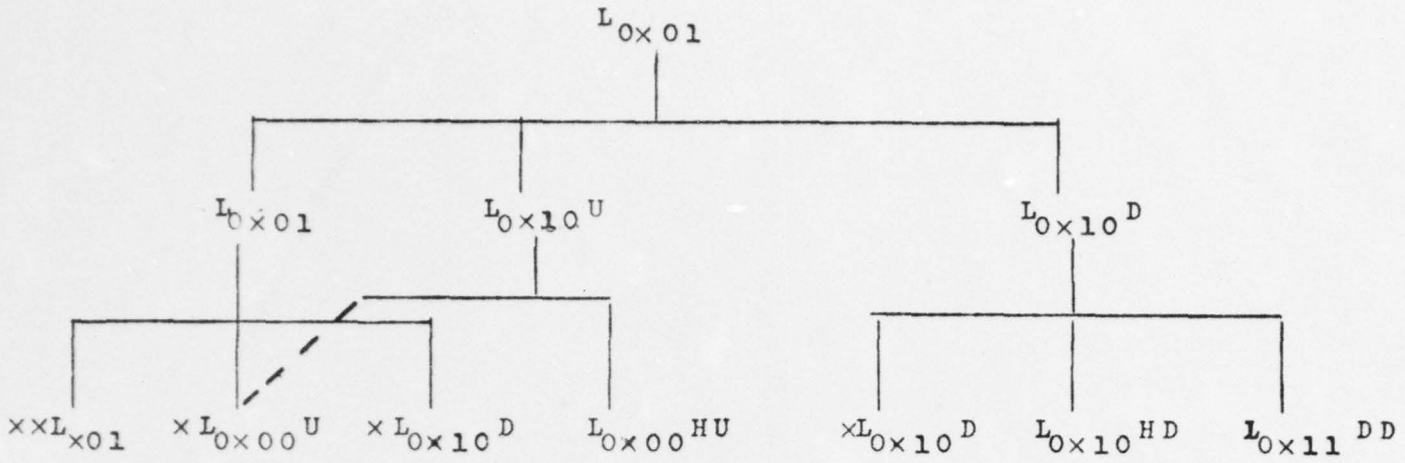


FIGURE 21(a):

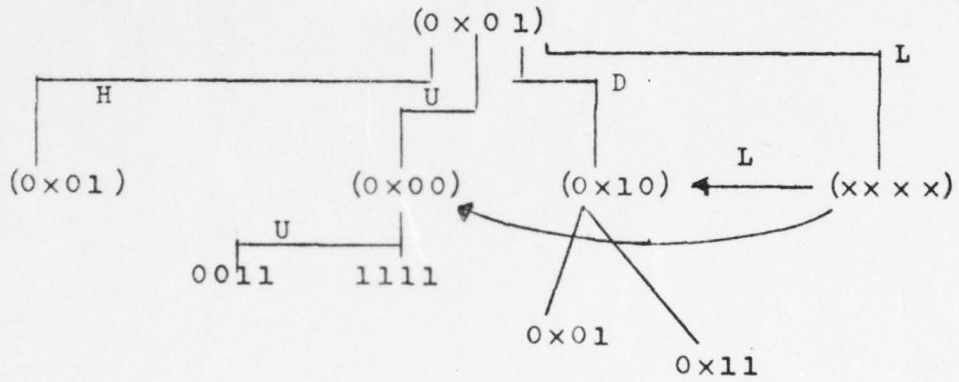


FIGURE 21(b)

5. Summary

In this report we have shown the feasibility of developing functional models for use in an ATG system. The models meet the requirements imposed by an ATG system based upon the concept of path sensitization. We have demonstrated this feasibility by looking at three specific element types, namely a flip-flop, a counter, and a shift register. Specifically, we have considered the following subproblems:

(1) implication (including backwards implication, a special case of line justification);

(2) D-drive
(3) Line justification

} Single (step) time frame and Multiple time frame.

Two distinct approaches have been presented:

(A) an algorithmic approach - treats functional box as an algorithm generator,

and

(B) an equation approach.