Chapter 10
Introduction to the Parallel Boolean Processor

10.1 INTRODUCTION

The triadic notation introduced in Chapter 9 is used heavily in the two previously-published papers on the parallel Boolean processor. The triadic map (see Figure 9.3) is used here to pictorially demonstrate the basic theorems. Detailed proofs of the theorems are not presented (they are available in the referenced papers). The table of intersections(Table 9.2) is based on the theorems.

10.2 THE THEOREMS

10.2.1 Theorem 3.1

Theorem 3.1 states that if the set of triadic points represented by the set of point identifiers {a,b c} form a triad, then if the triadic terms t_b and t_c both imply a function y, the triadic term t_a implies y. Also, if t_a implies y then both t_b and t_c imply y.

$$(t_a \rightarrow y) \equiv [(t_b \rightarrow y) \cap (t_c \rightarrow y)]$$

$$t_b \rightarrow t_a \rightarrow y \qquad t_c \rightarrow t_a \rightarrow y$$

The theorem also holds for the complement function $Y = \bar{y}$:

$$(t_a \rightarrow Y) \equiv [(t_b \rightarrow Y) \cap (t_c \rightarrow Y)]$$

Figure 10.1 demonstrates Theorem 3.1. The term t_{24} is the t_a of a triad formed by points {24,25,26}. Since t_{24} is a given term of the function y, it implies y and therefore both t_{25} and t_{26} imply y.

Points {39,40,41} form a triad. t_{40} and t_{41} are given terms of the function y, and therefore t_{39} must imply y.

The notation for the complement space in terms of the function Y is given in Figure 10.2.

10.2.2 Theorem 3.2

Theorem 3.2 states that if the set of triadic points represented by the set of point identifiers {a,b,c} form a triad, then if the triadic term t_b is a nonimplicant of (does not imply) function y, then the triadic term t_a is a nonimplicant of y. Also, if t_c is a nonimplicant of y, then t_a is a nonimplicant of y.

$$\sim(t_b \rightarrow y) \rightarrow \sim(t_a \rightarrow y)$$

$$\sim(t_c \rightarrow y) \rightarrow \sim(t_a \rightarrow y)$$

It is not necessary for both t_b and t_c to be nonimplicants of y for t_a to be a nonimplicant of y.

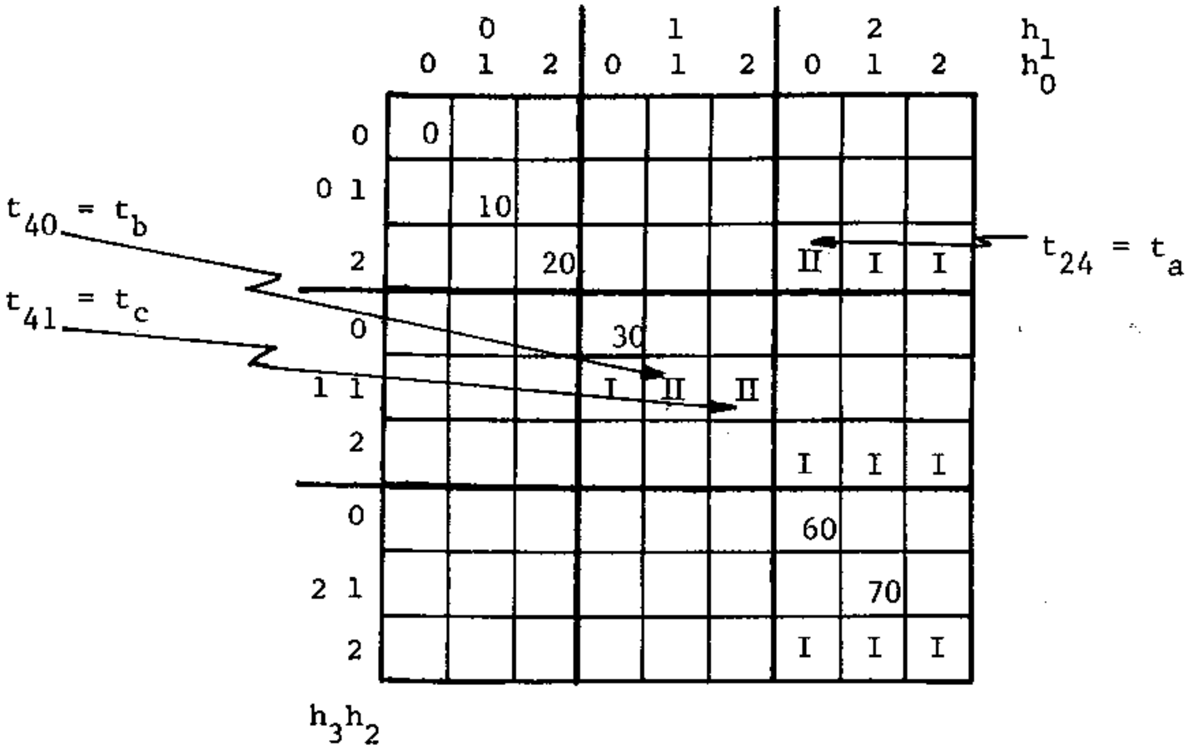
The theorem also applies to the complement space:

$$\sim(t_b \rightarrow Y) \rightarrow \sim(t_a \rightarrow Y)$$

$$\sim(t_c \rightarrow Y) \rightarrow \sim(t_a \rightarrow Y)$$

Using the notations of Figures 10.1 and 10.2, and adding to it to obtain

II	a form term for the ΣΠ-form of y
N	a form term for the ΣΠ-form of Y
I	a term added using Theorem 3.1 from the given terms shown as II
N	a term added using Theorem 3.1 from the given terms shown as *N*



$$y = \bar{x}_2\bar{x}_1 + x_3x_2x_1x_0 + x_3x_2x_1\bar{x}_0 \;=\; t_{24} + t_{40} + t_{41}$$

t_{24}: $(t_a \Rightarrow y)$ therefore $(t_b \Rightarrow y) \cap (t_c \Rightarrow y)$

t_{40}; t_{41}: $(t_b \Rightarrow y) \cap (t_c \Rightarrow y)$ therefore $(t_a \Rightarrow y)$

Figure 10.1. Theorem 3.1.

	h_1	0			1			2		
h_3	h_2 \ h_0	0	1	2	0	1	2	0	1	2
0	0	0								
	1		10							N̲
	2			20	N̲	N	N			
1	0									
	1									N
	2				N	N	N			
2	0									
	1								70	N
	2				N	N	N			80

$$Y = x_2\bar{x}_1\bar{x}_0 + \bar{x}_2x_1$$

$$= t_{17} + t_{21} \qquad N$$

Additional terms: N̲

Figure 10.2. Theorem 3.1 in complement space.

0 a term added using Theorem 3.2 where the term is a nonimplicant of y

(blank) a blank is a "don't care" term

/ a term added using Theorem 3.2 where the term is a nonimplicant of Y

∅ a term which is a nonimplicant of y and a nonimplicant of Y

Figure 10.3 presents a triadic map which combines Figures 10.1 and 10.2 and adds the notations derived by applying Theorem 3.2 to the expressions for y and for Y. A Marquand map is also given for reference. This is an incompletely specified function in that there are points which are considered "don't care".

10.2.3 Theorem 3.3

Theorem 3.3 states that given the triadic term t_h, which is an implicant of the function Y, and given any other triadic term, if the sum of the coefficients of the identifiers forms non-3 in all positions, then the second term is a nonimplicant of y. If the sum for any coefficient position is 3, no conclusion is made about the second term.

if $t_h \in \{t_h\}_Y$

$$h = (h_{n-1}\, h_{n-2} \cdots h_1 \;\; h_0)_3$$

where h_i is a coefficient of h in the i^{th} position, then

$$(h_j + h^*_j \neq 3 \text{ for every } j) \rightarrow [(t_h \rightarrow Y) \rightarrow \sim(t_{h^*} \rightarrow y)]$$

This theorem forms the basis for the hardware design of the parallel Boolean processor. It is demonstrated with a triadic map in Figure 10.4.

h_3h_2		0			1			2		h_1	
		0	1	2	0	1	2	0	1	2	h_0
0	0	∅	∅	∅	∅	∅	∅	∅	/	∅	
	1	∅	/	∅	/	/	/	0		N	
	2	∅	∅	∅	N	N	N	II	I	I	
1	0	∅	∅	∅	∅	∅	∅	∅	/	∅	
	1	∅	/	∅	I	II	II	0		N	
	2	∅	∅	∅	N	N	N	I	I	I	
2	0	∅	∅	∅	0	0	0	∅	/	∅	← t_{62}
	1	0		0				0		N	← t_{71}
	2	∅	∅	∅	N	N	N	I	I	I	← t_{80}

$$y = \bar{x}_2\bar{x}_1 + x_3x_2x_1x_0 + x_3x_2x_1\bar{x}_0$$

$$Y = x_2\bar{x}_1\bar{x}_0 + \bar{x}_2\bar{x}_1$$

$t_{62} \not\Rightarrow y \quad \cap \quad t_{62} \not\Rightarrow Y$

$t_{71} \not\Rightarrow y \quad \cap \quad t_{71} \Rightarrow Y$

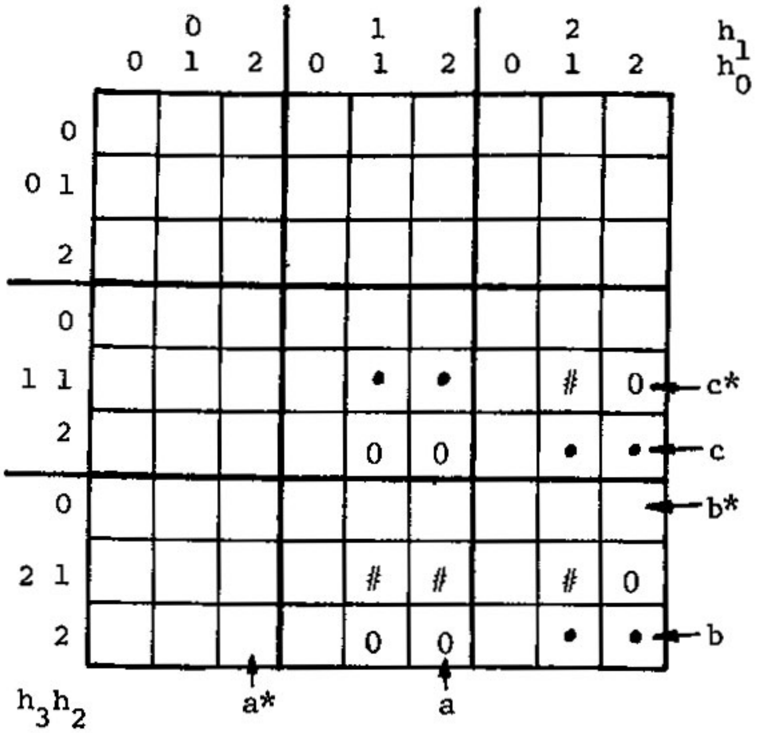
$t_{80} \Rightarrow y \quad \therefore \quad t_{80} \not\Rightarrow Y$

X_1 X_0

●	●	0	0
0	#	#	#
●	●	0	0
0	#	●	●

X_3 X_2

Figure 10.3. Theorem 3.2.



	h_3	h_2	h_1	h_0	
a*	2	2	0	2	$t_a \Longrightarrow Y \therefore t_a^* \not\Longrightarrow y$
a	2	2	1	2	
	4	4	1	4	h_j = NON3
b*	2	0	2	2	$t_b \Longrightarrow y \therefore t_b^* \not\Longrightarrow Y$
b	2	2	2	2	
	4	2	4	4	h_j = NON3
c*	1	1	2	2	$\therefore$ no conclusion about t_c^*
c	1	2	2	2	
	1	3	4	4	$\leftarrow h_j = 3 = h_2$
a*	2	2	0	2	$t_b \Longrightarrow y \therefore t_a^* \Longrightarrow Y$
b	2	2	2	2	
	4	4	2	4	h_j = NON3

Figure 10.4. Theorem 3.3.

10.2.4 Theorem 3.4

The "ordering of implicants" referred to earlier is stated in Theorem 3.4:

For any two triadic terms t_h and t_h^*, if t_h implies t_h^* then the point identifier (triadic index) h is not smaller than the point identifier h*.

$$(t_h \rightarrow t_h^*) \rightarrow (h \geq h^*)$$

This is obvious from an examination of Figures 9.4 and 9.7.

10.2.5 Theorem 3.5

Theorem 3.5 clarifies prime implicants and is the basis of the prime implicant generation technique demonstrated in Chapter 9. The theorem states that any term in the maximal ΣΠ-form of y which does not imply any other term in the maximal ΣΠ-form whose triadic index is less than its own is a prime implicant of the function (see Figure 9.14).

given $\{t_h\}_{max}$ is the set of terms of the maximal
ΣΠ-form of y,
$t_h \in \{t_h\}_{max}$ and $t_p \in \{t_h\}_{max}$
$[\sim(t_p \rightarrow t_h)$ for every t_h where $h < p]$
$\rightarrow$ (t_p is a prime implicant of y)

10.2.6 Theorem 3.6

Theorem 3.6 further defines prime implicants: Given that the set $\{t_h\}_{h*}$ represents terms from the complete sum of y (the set of all prime implicants of y) such that

$$h < h^*,$$

$$h = (h_{n-1}\, h_{n-2}\, h_{n-3} \cdots h_1\, h_0)_3,$$

$$h^* = (h^*_{n-1}\, h^*_{n-2}\, h^*_{n-3} \cdots h^*_1\, h^*_0)_3,$$

then if at least one t_h from this set exists for which:

$$(h_j + h^*_j \neq 3)$$

and $(h^*_j \neq 0) \cup (h = 0)$ for every j,

then the term t^*_h is not a prime implicant of y.

10.2.7 Theorem 3.7

The last two theorems, 3.7 and 3.7A, are used in the hardware algorithms. For two triadic terms t_h and $t_h{}^*$, if the sum of the coefficients of their identifiers is not equal to 3 in all positions while the corresponding coefficient position of h* is not equal to 0, then t_h implies t_h^*. (Figure 10.5)

For two terms t_h, t_h^*,

$$\{[(h_j + h_j^* \neq 3) \cap (h_j^* > 0)] \text{ for every } j\}$$

$$\rightarrow (t_{h^*} \rightarrow t_h)$$

10.2.8 Theorem 3.7A

This theorem modifies Theorem 3.7. Given $\{t_h\}_{\max Y}$, the set of prime implicants for the function $Y = \bar{y}$, then for any term t_h from this set and any term t_{h^*}, if the sum of the coefficients of h and h* is not equal to 3 in all positions while the corresponding coefficient of h^* is not equal to 0, then t_{h^*} is a nonimplicant of y.

$$\{[(h_j + h_j^* \neq 3) \cap (h_j^* > 0)] \text{ for every } j\}$$

$$\rightarrow \sim(t_{h^*} \rightarrow y)$$

		0			1			2			h_1
		0	1	2	0	1	2	0	1	2	h_0
	0	0									
0	1		10					I	I	I	t_{17}
	2			20							
	0				30						
1	1					0	0		•	#	
	2					0	#		•	0	
	0							60			
2	1					0	0		•	•	t_{71}
	2					#	•		#	•	
h_3h_2											

$$\begin{array}{lcccc} t_{71} & 2 & 1 & 2 & 2 \\ t_{17} & 0 & 1 & 2 & 2 \\ \hline & 2 & 2 & 4 & 4 \end{array} \qquad t_{71} \Longrightarrow t_{17}$$

$$\begin{array}{lcccc} t_{17} & 0 & 1 & 2 & 2 \\ t_{15} & 0 & 1 & 2 & 0 \\ \hline & 0 & 2 & 4 & 2 \end{array} \qquad t_{17} \Longrightarrow t_{15}$$

Figure 10.5. Theorem 3.7.

10.3 THE ORIGINAL DESIGN OF THE PARALLEL BOOLEAN PROCESSOR

10.3.1 Introduction

The preceeding theorems and the previously-presented material on triadic maps form the basis for the parallel Boolean processor designed by Svoboda. The processor has been described in the literature under the name of the Boolean analyzer.

The parallel Boolean processor is a special processor designed to solve certain fundamental problems. Some of these are:

1. The listing of prime implicants
2. The solution of general systems of Boolean equations
3. The solution of special design automation problems
4. The generation of a test sequence for combinational circuits
5. The generation of the Boolean difference for x_i for a function F
6. The coverage problem

It was originally designed to be interfaced to the "variable" system at UCLA. Updated designs have been issued by students for MSI-level implementation. It now appears that a reasonably-sized processor could be built from LSI chips and an interface to a microprocessor system with disk memory and some form of input/output (possibly a printing terminal). With the changes in technology, the cost has become more than reasonable for such a device.

The original design was for 2 MHz operation. Present technology would allow a faster clock and therefore faster operation.

10.3.2 The Original Design

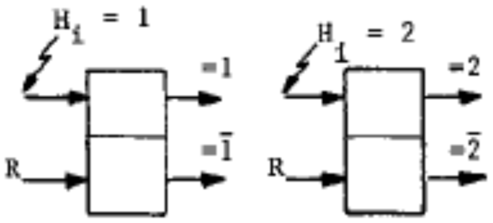
The first design called for 100 processing registers that were to be operated in parallel. These were to be loaded (via software interface to a main system) with the triadic equivalent of terms from the complement space. The processor works on the complement to the function, with the result left in main memory as:

1 = cancelled

0 = implicant or prime implicant

Each register containts the storage for the coefficients of a point in a 22-variable space stored in triadic notation. Figure 10.6 presents the original configuration of the processor and its registers.

Each coefficient is represented by its triadic digit, and this requires two flip-flop (F/F) elements per digit:



The encoding on the bus is:

$H_i = 2$

$H_i = 1$ for the i^{th} digit

$H_i = \bar{0}$

This requires 44 F/F elements per register or a total of 4400 F/Fs for the register network. The details of the processing registers are diagrammed in Figures 10.7 and 10.8.

The triadic or binary space (depending upon the algorithm being used) is represented by storing one bit per point in the space and by addressing this bit via a triadic or binary "clock" which is encoded in triadic notation. The bus from the clock contains three signal lines per digit, with the maximum number of digits fixed at 22.

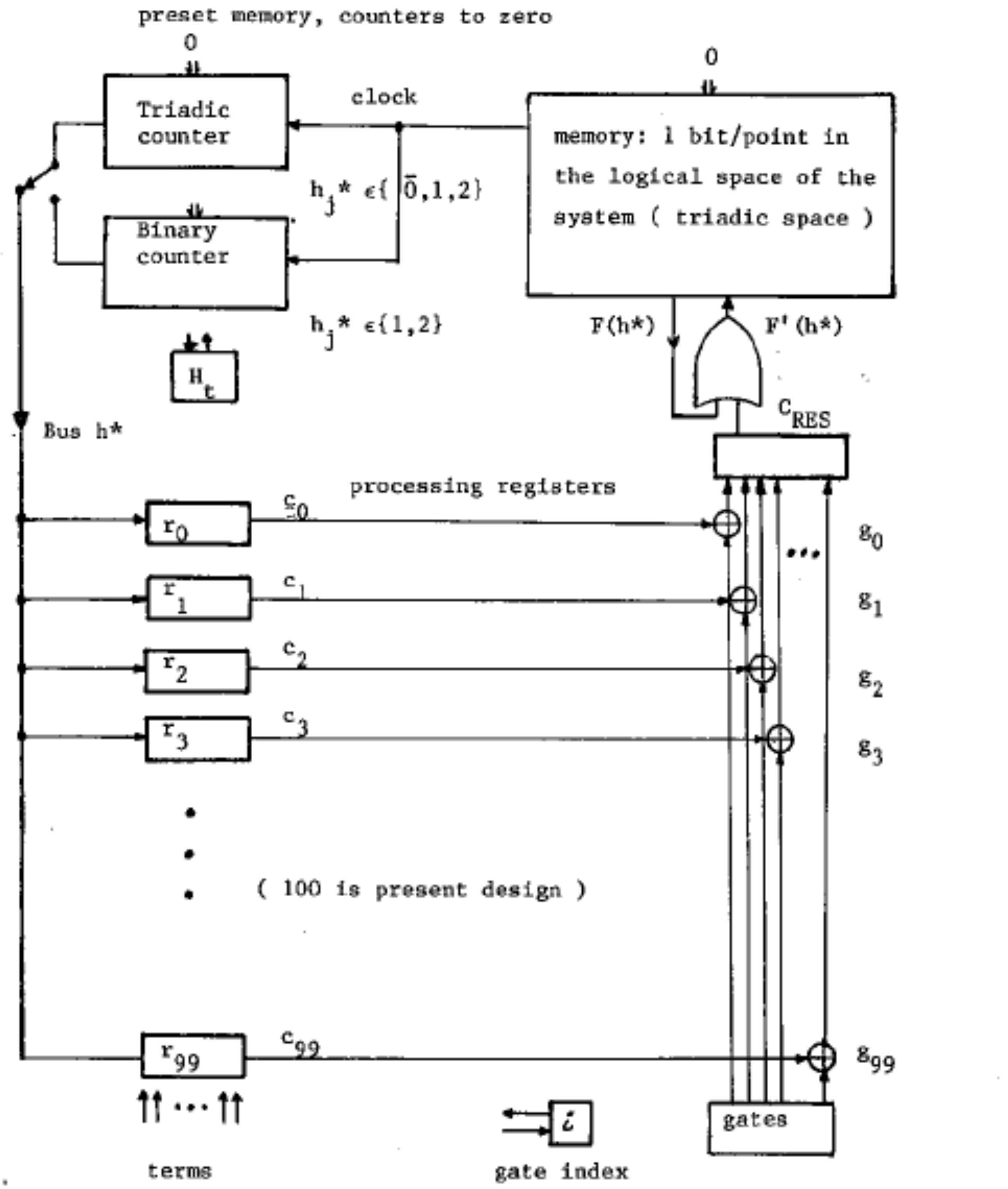


Figure 10.6. Block diagram - Parallel Boolean Processor.

Using the notation of the original paper:

$$Q = q_{\substack{\text{prev}\\\text{digit}}} \left(\left(h_0{}^* = \bar{0} \right) \cup \left(h_0 = \bar{1} \right)\left(h_0 = \bar{2} \right) \right)$$

Implementation of theorem 3.6:

$$\left(\left(h_j{}^* = \text{NØNO} \right) \cup \left(h_j = 0 \right) \text{ for every } j \right)$$

$$\bar{P} = p_{\substack{\text{prev}\\\text{digit}}} + \left(h_0 = 1 \right)\left(h_0{}^* = 2 \right) + \left(h_0 = 2 \right)\left(h_0{}^* = 1 \right)$$

Implementation of theorem 3.3:

$$\left(h_j + h_j{}^* = \text{NØN3 for every } j \right)$$

$C^k = P^k$ implicant listing

$C^k = P^k \cap Q^k$ prime implicant

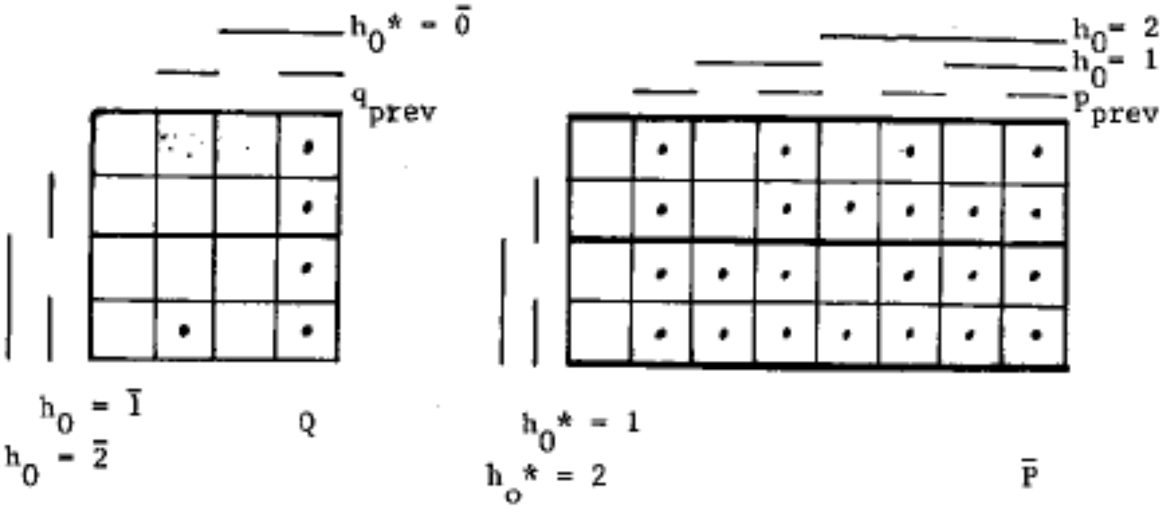
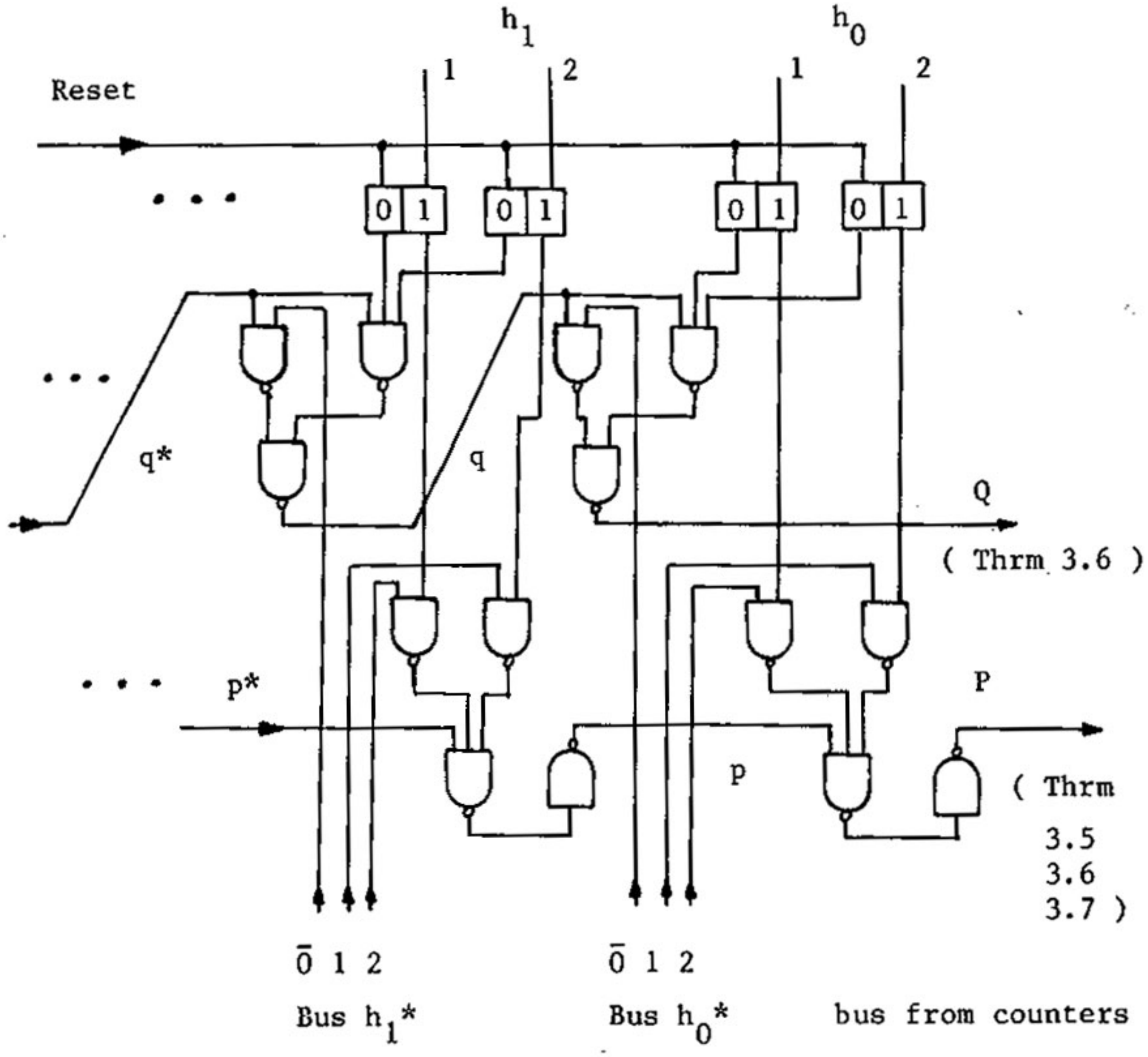


Figure 10.7. Processing register detail.



q* , p* at unused high order positions are set ON

Figure 10.8. Original design of the two lower digits for the processing registers.

The original design provides for the comparison of the digits of the bus with each corresponding digit of each of the 100 processing registers at the same time. This parallelism of 22x100 effective comparisons produces one bit of information for each clock step. The bit is "OR"d into the appropriate position in the memory.

10.3.3 Improved Parallelism in the Processor Implementati

The original design provided for up to 22x100 simultaneous effective comparisons of triadic digits per clock step. This may be improved upon with a slight modificatio of the processor hardware. The modification is the additi of logic to the processing registers such that for each comparison step of the clock, 3^j bits rather than one bit of the result is completed. The clock is then stepped 3^j rather than 3^0 (triadic or binary clock).

The functions $\emptyset_{q*}(q)$ are defined as functions of the values of the lower-order digits of the terms and not of the clock digits. The clock digits are represented by the added hardware.

$$q = h_{i-1}3^{i-1} + h_{i-2}3^{i-2} + \dots + h_0 3^0$$

$$\emptyset_{q*}(q) = \emptyset_{q*} =$$

$$1 \text{ for } h_j + h_{j*} \neq 3$$

$$0 \text{ for } h_j + h_{j*} = 3$$

$$\text{for every } j = 0, 1, \dots, i-1$$

For the case 3^2, Figure 10.9 provides the map and the functions $\emptyset_{q*}$. Nine bits are produced for the result for each step of the clock. Figure 10.10 gives the actual implementation of these functions for one processing register. Each of the 100 registers would be the same.

Figures 10.11 and 10.12 present the map and functions for a 27-bit result per clock. The hardware tradeoffs make this version impractical at this time. If implemented it would mean 27×100×22 effective comparisons per clock step.

h_3h_2		0			1			2			h_1	
		0	1	2	0	1	2	0	1	2	h_0	for P^k:
0	0	1	1	1	1	1	1	1	1	1		$\Phi = 1$
	1	1	1	0	1	1	0	1	1	0		$\Phi = \text{NON}(h_0 = 2)$
	2	1	0	1	1	0	1	1	0	1		$\Phi = \text{NON}(h_0 = 1)$
1	0	1	1	1	1	1	1	0	0	0		$\Phi = \text{NON}(h_1 = 2)$
	1	1	1	0	1	1	0	0	0	0		$\Phi = \text{NOR}(h_1 = 2, h_0 = 2)$
	2	1	0	1	1	0	1	0	0	0		$\Phi = \text{NOR}(h_1 = 2, h_0 = 1)$
2	0	1	1	1	0	0	0	1	1	1		$\Phi = \text{NON}(h_1 = 1)$
	1	1	1	0	0	0	0	1	1	0		$\Phi = \text{NOR}(h_1 = 1, h_0 = 2)$
	2	1	0	1	0	0	0	1	0	1		$\Phi = \text{NOR}(h_1 = 1, h_0 = 1)$

Figure 10.9. The functions Φ for a clock step of 3**2.

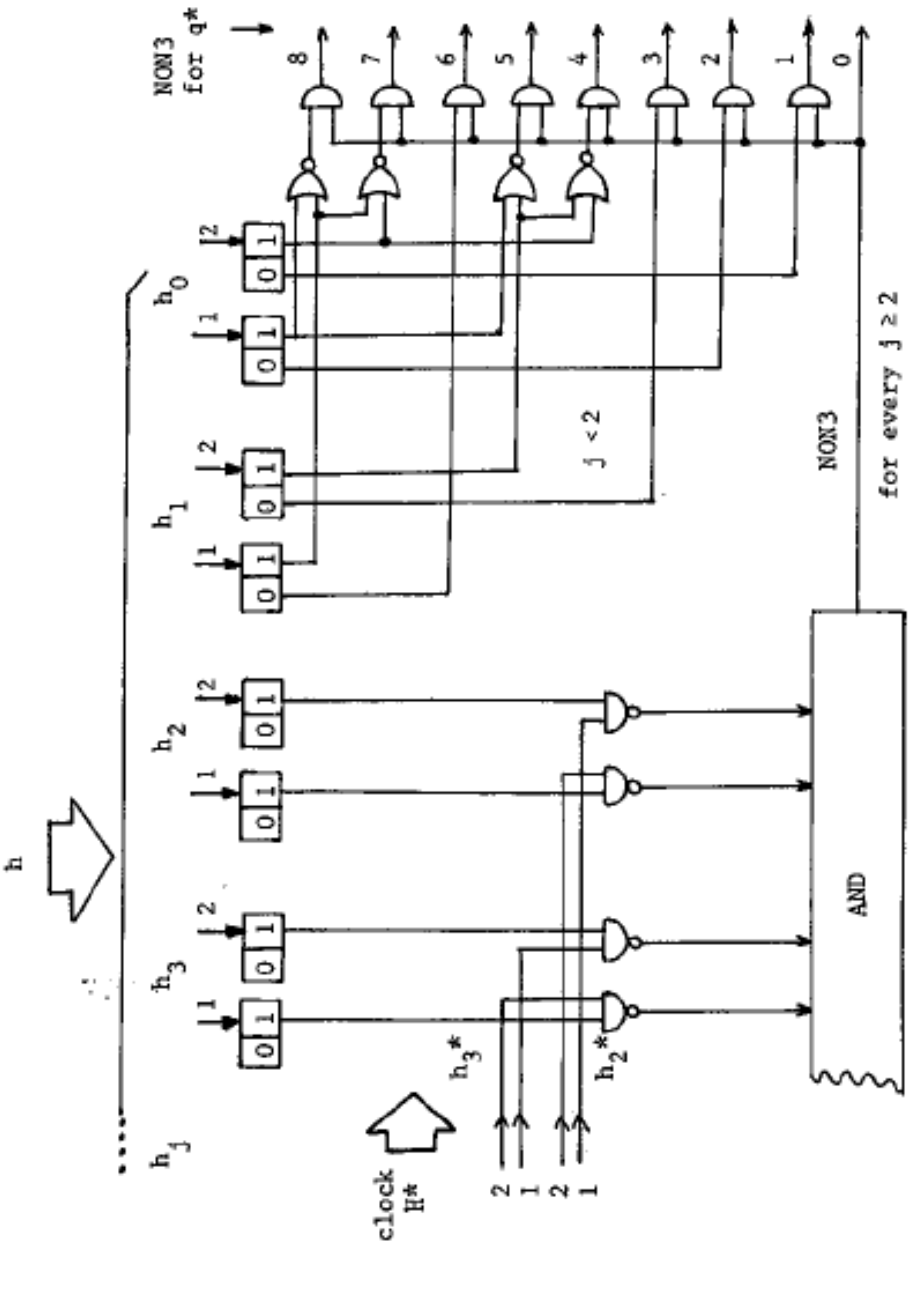


Figure 10.10. Processing register with 9 parallel outputs.

			0									1									2									h_2
			0			1			2			0			1			2			0			1			2			h_1
			0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	h_0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
		1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	
		2	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	
	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	
		1	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	
		2	1	0	1	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	
	2	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	
		1	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	
		2	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	1	
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	
		1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	
		2	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	
	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
		1	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
		2	1	0	1	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
	2	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	
		1	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
		2	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
2	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	
		1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	
		2	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	1	
	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	
		1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	
		2	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	
	2	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	
		1	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	
		2	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	
h_2^*	h_1^*	h_0^*																												

Triadic map of 6 variables

$\Phi_{q*} = 1$ iff $h_j + h_j* = \text{NON3}$ for any $j < 3$

$\Phi_{q*}(q) = \Phi_{q*}$ $\qquad$ $q = h_2 3^2 + h_1 3^1 + h_0 3^0$

Figure 10.11. Map of Φ_{q*} where i = 3.

The functions: each clock step produces 27 bits of output

$$\begin{aligned}
\Phi_0 &= 1 \\
\Phi_1 &= \mathrm{NON}(\ h_0 = 2\ ) \\
\Phi_2 &= \mathrm{NON}(\ h_0 = 1\ ) \\
\Phi_3 &= \mathrm{NON}(\ h_1 = 2\ ) \\
\Phi_4 &= \mathrm{NOR}(\ h_1 = 2,\ h_0 = 2\ ) \\
\Phi_5 &= \mathrm{NOR}(\ h_1 = 2,\ h_0 = 1\ ) \\
\Phi_6 &= \mathrm{NON}(\ h_1 = 1\ ) \\
\Phi_7 &= \mathrm{NOR}(\ h_1 = 1,\ h_0 = 2\ ) \\
\Phi_8 &= \mathrm{NOR}(\ h_1 = 1,\ h_0 = 1\ ) \\
\Phi_9 &= \mathrm{NON}(\ h_2 = 2\ ) \\
\Phi_{10} &= \mathrm{NOR}(\ h_2 = 2,\ h_0 = 2\ ) \\
\Phi_{11} &= \mathrm{NOR}(\ h_2 = 2,\ h_0 = 1\ ) \\
\Phi_{12} &= \mathrm{NOR}(\ h_2 = 2,\ h_1 = 2\ ) \\
\Phi_{13} &= \mathrm{NOR}(\ h_2 = 2,\ h_1 = 2,\ h_0 = 2\ ) \\
\Phi_{14} &= \mathrm{NOR}(\ h_2 = 2,\ h_1 = 2,\ h_0 = 1\ ) \\
\Phi_{15} &= \mathrm{NOR}(\ h_2 = 2,\ h_1 = 1\ ) \\
\Phi_{16} &= \mathrm{NOR}(\ h_2 = 2,\ h_1 = 1,\ h_0 = 2\ ) \\
\Phi_{17} &= \mathrm{NOR}(\ h_2 = 2,\ h_1 = 1,\ h_0 = 1\ ) \\
\Phi_{18} &= \mathrm{NON}(\ h_2 = 1\ ) \\
\Phi_{19} &= \mathrm{NOR}(\ h_2 = 1,\ h_0 = 2\ ) \\
\Phi_{20} &= \mathrm{NOR}(\ h_2 = 1,\ h_0 = 1\ ) \\
\Phi_{21} &= \mathrm{NOR}(\ h_2 = 1,\ h_1 = 2\ ) \\
\Phi_{22} &= \mathrm{NOR}(\ h_2 = 1,\ h_1 = 2,\ h_0 = 2\ ) \\
\Phi_{23} &= \mathrm{NOR}(\ h_2 = 1,\ h_1 = 2,\ h_0 = 1\ ) \\
\Phi_{24} &= \mathrm{NOR}(\ h_2 = 1,\ h_1 = 1\ ) \\
\Phi_{25} &= \mathrm{NOR}(\ h_2 = 1,\ h_1 = 1,\ h_0 = 2\ ) \\
\Phi_{26} &= \mathrm{NOR}(\ h_2 = 1,\ h_1 = 1,\ h_0 = 1\ )
\end{aligned}$$

Figure 10.12. The Φ_{q*} functions for a 3^3 clock step.

10.4 APPLICATIONS

Applications of the processor are detailed in the literature. The following sections will give a rough outline of the various procedures.

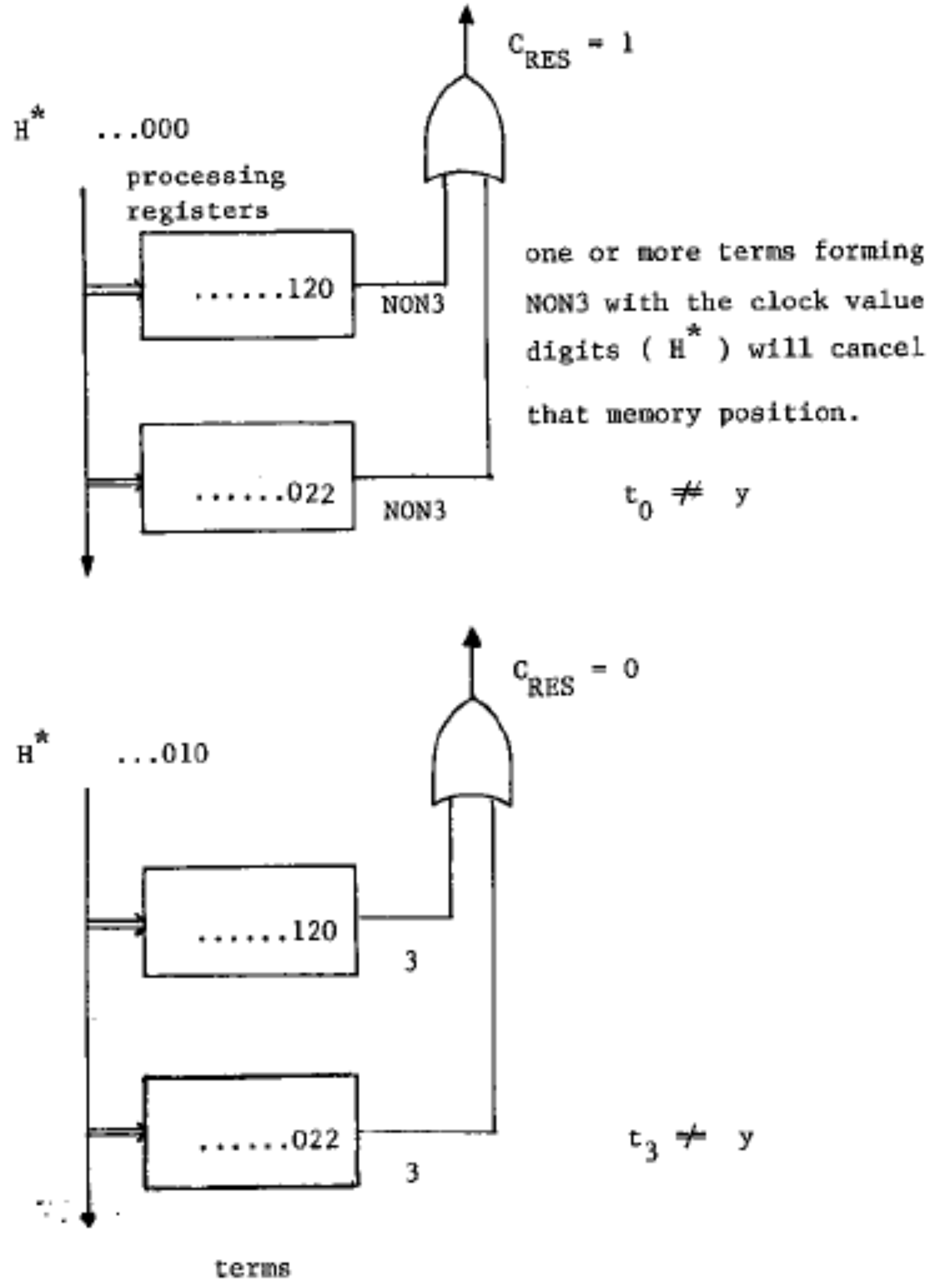
10.4.1 Implicant Listing

To find the implicants of a function, the processing registers are loaded with the terms of the complement function, $\{t_h\}_Y$ (see Figure 10.13). As the h* counter is incremented from 0 to 3^n in steps of 3^0, representing points in the triadic space, the registers are summed with a 1 used for any NON3 sum and a 0 used for any sum containing a 3 (refer to Figure 10.14). The sum results are OR'd to form one bit of the resulting memory pattern at address h*. The resulting bit pattern in memory is an inverse of the desired map, i.e., it contains a 0 everywhere there is a triadic term implying y. Since the results from all of the 100 registers at any clock step are OR'd, it is sufficient for 1 term to form NON3 for the term addressed in memory to be marked "cancelled".

The implicant listing application uses the triadic clock, and the triadic map.

10.4.2 Implicant Listing Under Improved Parallelism

Figure 10.15 presents the implicant listing of a function and details the register contents over the clock cycle for the case when 9 bits of results are computed per clock step. (The distinguishment of the "•" and "I" symbols is not represented in the memory bit patterns as presented. The "•" comes from knowing where the minterm space appears on the map.) Each clock step is 3^2 and each step produces an output equal to one row of the Marquand map in this example.



one or more terms forming NON3 with the clock value digits (H^*) will cancel that memory position.

$t_0 \neq y$

$t_3 \neq y$

$$y = x_2 + \bar{x}_3 x_1 \qquad Y = \bar{y} = \bar{x}_2 \ (\ x_3 + \bar{x}_1 \ ) = \overset{t_{15}}{x_3\bar{x}_2} + \overset{t_8}{\bar{x}_2\bar{x}_1}$$

Figure 10.13. Implicant listing - example clock step.

h* Counter	Terms	Term 1 h* + h = NON3	Term 2 h* + h = NON3	Result F'(h*)
0 0 0	1 2 0	1 (1 2 0)	1 (0 2 2)	1
0 0 1	0 2 2	1	0	1
0 0 2		1	1	1
0 1 0		0 (1 3 0)	0 (0 3 2)	0
0 1 1		0	0	0
0 1 2		0	0	0
0 2 0		1	1	1
0 2 1		1	0	1
0 2 2		1	1	1
1 0 0		1	1	1
1 0 1		1	0	1
1 0 2		1 (2 2 0)	1 (1 2 4)	1
1 1 0		0	0	0
1 1 1		0	0	0
1 1 2		0	0	0
1 2 0		1	1	1
1 2 1		1	0	1
1 2 2		1	1	1
2 0 0		0	1	1
2 0 1		0	0	0
2 0 2		0	1	1
2 1 0		0	0	0
2 1 1		0	0	0
2 1 2		0	0	0
2 2 0		0	0	0
2 2 1		0	0	0
2 2 2		0 (3 4 0)	1 (2 4 4)	1

	0			1			2			x_2
	0	1	2	0	1	2	0	1	2	x_1
0	c	c	c				c	c	c	
1	c	c	c				c	c	c	result
2	c		c				c		c	
x_3										

c cancel point

Figure 10.14. Detail of implicant listing.

clock	Processing registers (parallel)			
h*	(2100)	(2011)	(1021)	(0120)
00--	111111111	110110000	110000110	111000111
01--	111111111	110110000	110000110	111000111
02--	000000000	110110000	110000110	000000000
10--	000000000	000000000	110000110	111000111
11--	000000000	000000000	110000110	111000111
12--	000000000	000000000	110000110	000000000
20--	111111111	110110000	000000000	111000111
21--	111111111	110110000	000000000	111000111
22--	000000000	110110000	000000000	000000000

$$\bar{y} = \bar{x}_3x_2 + \bar{x}_3x_1x_0 + x_3\bar{x}_1x_0 + x_2\bar{x}_1$$

Terms for the registers: (2100), (2011), (1021), (0120)

			0			1			2		h_1
		0	1	2	0	1	2	0	1	2	h_0
	0										
0	1										
	2			I			I			I	
	0				I	I	I				
1	1				I						
	2			I	I						
	0										
2	1										
	2			I				I			
h_3h_2											

The final map

			0			1			2		h_1
		0	1	2	0	1	2	0	1	2	h_0
	0	1	1	1	1	1	1	1	1	1	
0	1	1	1	1	1	1	1	1	1	1	
	2	1	1	0	1	1	0	1	1	0	
	0	1	1	1	0	0	0	1	1	1	
1	1	1	1	1	0	0	0	1	1	1	
	2	1	1	0	0	0	0	1	1	0	
	0	1	1	1	1	1	1	1	1	1	
2	1	1	1	1	1	1	1	1	1	1	
	2	1	1	0	1	1	0	0	0	0	
h_3h_2											

Result in memory

Figure 10.15. Implicant listing.

10.4.3 Existence Function

The Boolean processor is operated in binary mode for this application (no triadic term contains a 0). The processing is done in triadic notation.

The processing registers are loaded with the terms of Y from each of the equations of the system (100 of them) as shown in Figure 10.16. The binary counter to correspond to the memory location being addressed, the counter in triadic form, is input to bus h^*. The processing is the same as for implicant listing. A "1"is produced for any term producing a NON3 coefficient sum with the h^* bus and the results of all 100 registers are OR'd into memory.

Upon completion the complement map is stored in memory. That is, each "0" in memory is a "1" in the existence function map of the system.

The terms of Y are derived from the equations of y by the relationships:

$$(a = b) \equiv (y = 1)$$

$$(\bar{a}b + a\bar{b} = 0) \equiv (Y = 0)$$

For example, the first equation of Figure 10.16:

$$x_4 = x_0x_1$$

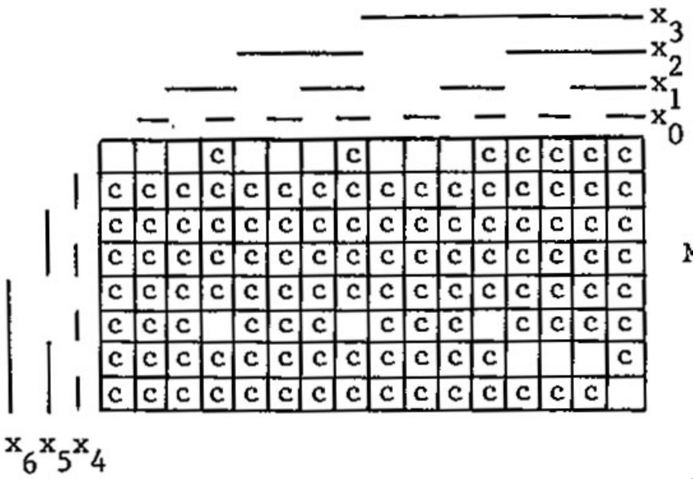
produces the complement equation:

$$\overline{x_4}x_1x_0 + x_4\overline{x_0} + x_4\overline{x_1} = 0$$

10.4.4 Larger Systems

10.4.4.1 More terms

Where there are more than 100 terms to be used for implicant listing, existence function generation, etc., then several passes may be made. Each pass through memory restarts the addressing bus clock h^* at the origin and is performed with a new set of 100 terms in the registers.



			c				c				c	c	c	c	c
c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
c	c	c		c	c	c		c	c	c		c	c	c	c
c	c	c	c	c	c	c	c	c	c	c	c				c
c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	

Memory in binary mode

Equations	$\{t_h\}_Y$	Terms in Triadic	memory (Binary)	h* Triadic	Sum	
$x_4 = x_0 x_1$	$\bar{x}_4 x_0 x_1$	0020011	0000000	2222222	=3 all t_h	
	$x_4 \bar{x}_0$	0010002	0000001	2222221		
	$x_4 \bar{x}_1$	0010020	0000010	2222212		
$x_5 = x_2 x_3$	$\bar{x}_5 x_2 x_3$	0201100	0000011	2222211	= NON3	t_1
	$x_5 \bar{x}_2$	0100200	0000100	2222122		
	$x_5 \bar{x}_3$	0102000				
$x_6 = x_4 + x_5$	$\bar{x}_6 x_4$	2010000	0010011	2212211	= NON3	t_7
	$\bar{x}_6 x_5$	2100000				
	$x_6 \bar{x}_5 \bar{x}_4$	1220000				
			1101100	1121122	=3 all t_h	
			1111111	1111111	=3 all t_h	

There are nine terms loaded into the registers for this example

cancellation done in triadic notation

Figure 10.16. Generating the existence matrix.

As many passes as are needed may be performed until all of the terms have been fed through. There is no need to combine or reduce the Y terms of the various equations prior to processing.

The ability to reprocess with new sets of terms comes from the OR'ing function; the result of each pass is OR'd with that of the preceeding passes.

$$F'(h^*) = F(h^*) + C_{res}$$

The memory is correctly filled at the end of the final pass.

10.4.4.2 More variables

Where there are more than 22 variables, the hardware design is limited. The limitation is due to the memory requirements. The design has not been extended at this time.

The memory requirements may be gauged by noting that:

for binary space:

2^{22} = 4,194,304 bits of storage

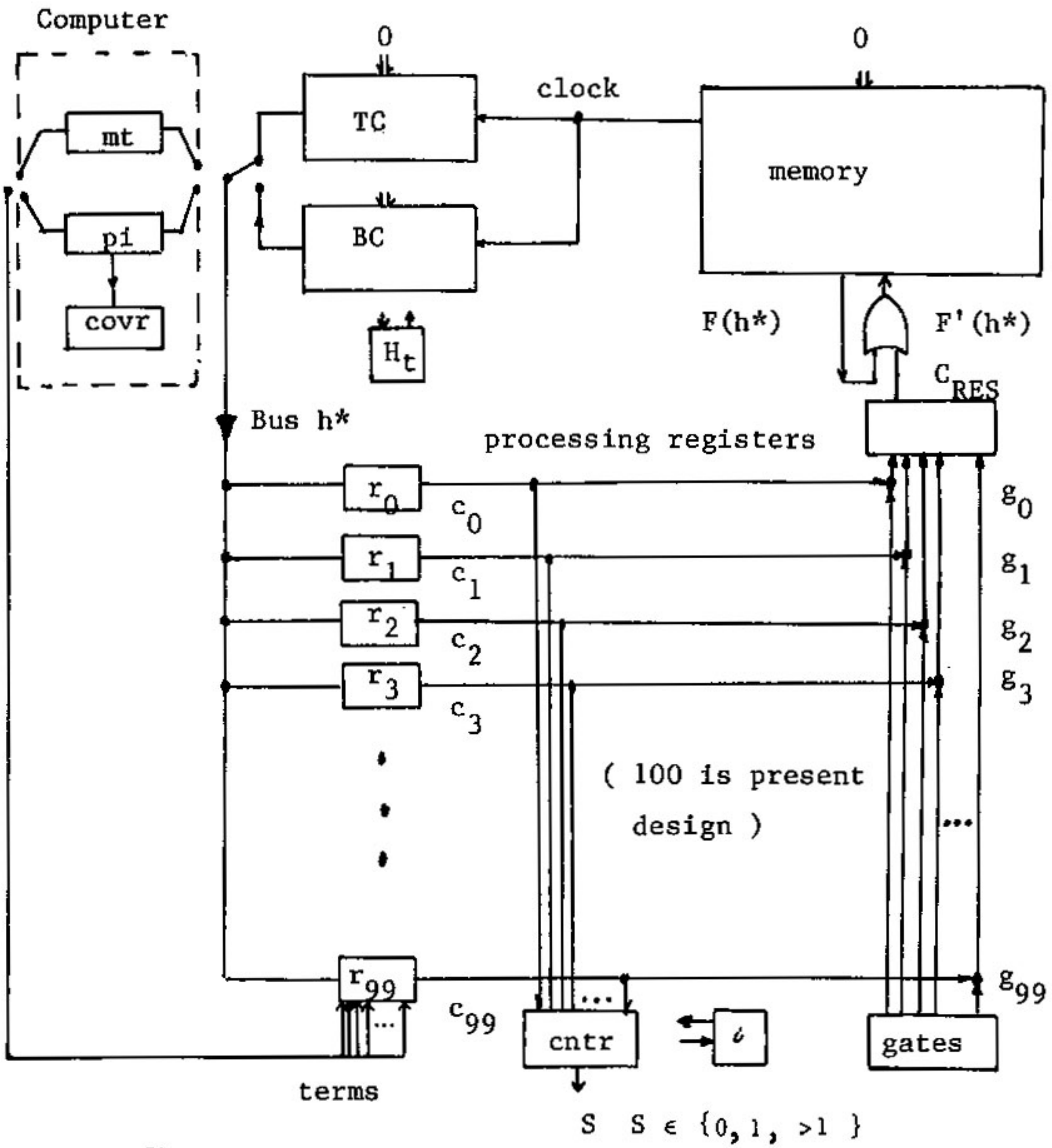
for trinary space:

3^{14} = 4,782,969 bits of storage

10.5 THE COVERAGE ALGORITHM

The coverage algorithm is a recently developed application for the parallel Boolean processor. It is represented in the last step of the logical instruments solution to minimization. (The paper defining the application in detail has not yet been published.)

The solution makes use of two concepts: (1) multilicity and (2) coverage. A special counter is also required for the processor that is capable of counting a 0, 1, or more-than-1 condition. Figure 10.17 presents the



mt = minterms

pi = prime implicants

covr = coverage

TC = triadic counter $h_i^* \in \{ \bar{0}, 1, 2 \}$

BC = binary counter $h_j^* \in \{ 1, 2 \}$

Figure 10.17. Block diagram - Parallel Boolean Processor.

modified block diagram of the processor.

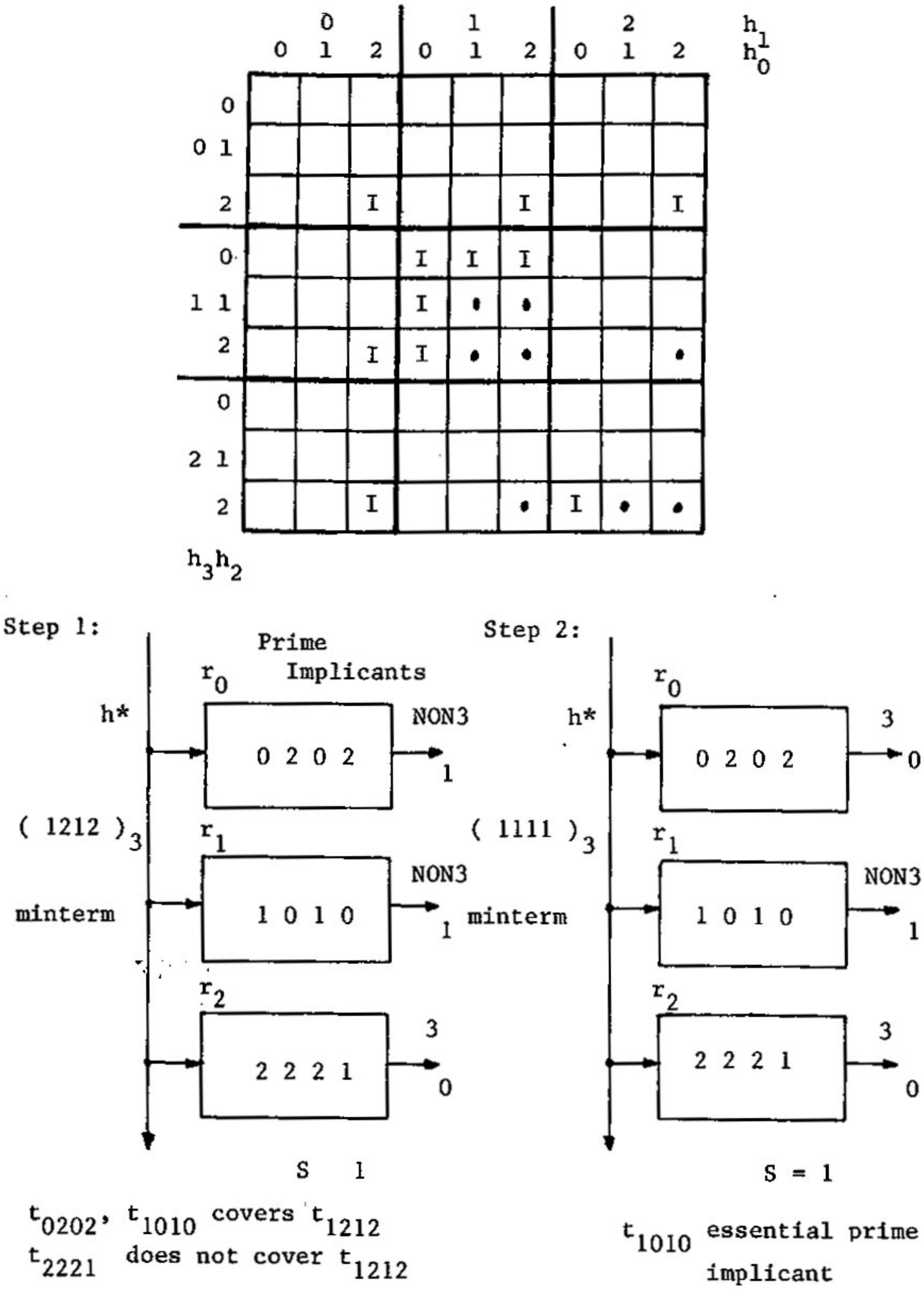
Multiplicity is diagrammed in Figure 10.18. It is the production of a count of the number of prime implicants which cover a given minterm. The processing registers are loaded with the prime implicants found for the function. The clock bus represents the minterms. The result bits are formed by checking for NON3 as in the implicant, prime implicant, and existence function applications. A NON3 indicates coverage of the minterm on the bus by the prime implicant in the register. An essential prime implicant is found when the count of prime implicants forming NON3 with any minterm is 1. The sum condition must be recorded for each minterm.

Coverage is demonstrated in Figure 10.19. It is an algorithm to find the optimum ΣΠ-form of the solution for a function. First, the processing registers are loaded with the minterms of the function and the prime implicants are sent down the bus. The prime implicants are sorted based on their status as essential or dominant prime implicants. A NON3 indicates that the prime implicant on the bus covers the minterm in the register. The prime implicant is recorded as belonging to the solution and all minterms forming non-3 with it are removed from the register.

The next step is to send the next prime implicant down the bus, repeating the above. This process is repeated until there are no more terms in the register.

This application is based upon the modified form of Theorem 3.3:

$$(h_j + h_j^* \neq 3 \text{ for every } j)$$
$$\rightarrow [(t_{h^*} = m_a) \rightarrow (m_a = t_h)]$$

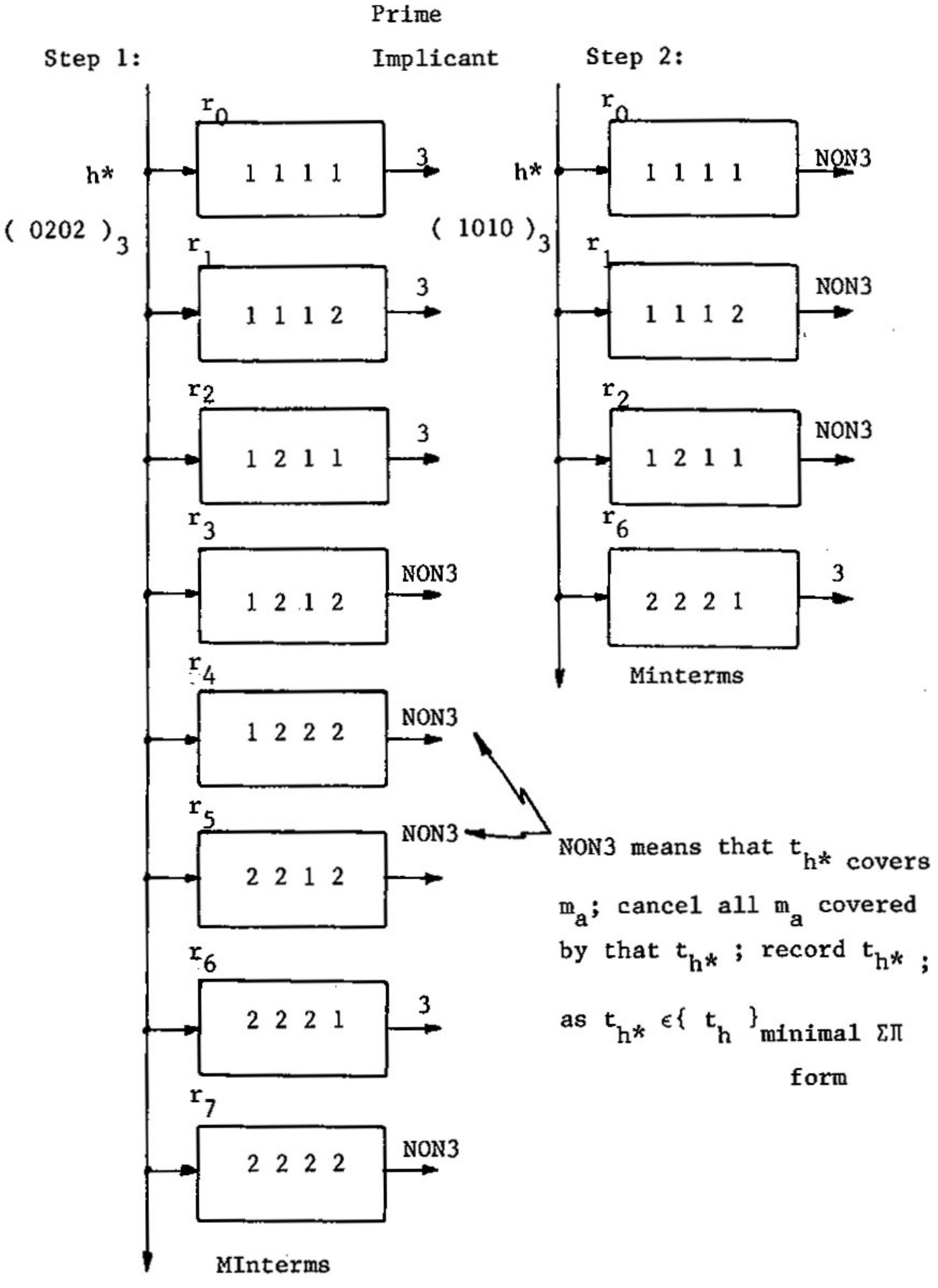


		0			1			2			h_1
		0	1	2	0	1	2	0	1	2	h_0
0	0										
	1										
	2			I			I			I	
1	0				I	I	I				
	1				I	•	•				
	2			I	I	•	•			•	
2	0										
	1										
	2			I			•	I	•	•	
h_3h_2											

t_{0202}, t_{1010} covers t_{1212}
t_{2221} does not cover t_{1212}

t_{1010} essential prime implicant

Figure 10.18. Map and clock step detail of MULTIPLICITY.



NON3 means that t_{h*} covers m_a; cancel all m_a covered by that t_{h*} ; record t_{h*} ; as $t_{h*} \epsilon \{ t_h \}_{\text{minimal } \Sigma\Pi}$ form

Figure 10.19. Clock step detail of COVERAGE.

10.6 THE BOOLEAN DIFFERENCE

The Boolean difference is defined as:

$$D_iF(x) = F(x_0, x_1, \ldots , x_i, \ldots , x_{n-1})$$
$$\veebar F(x_0, x_1, \ldots , \overline{x_i}, \ldots , x_{n-1})$$

or,

$$D_iF(x) = F(x_0, x_1, \ldots , 1, \ldots , x_{n-1})$$
$$\veebar F(x_0, x_1, \ldots , 0, \ldots , x_{n-1})$$

When $D_iF(x) = 0$, $y = F(x)$ is unconditionally independent of x_i.

When $D_iF(x) = 1$, $y = F(x)$ is unconditionally dependent on x_i.

When $D_iF(x) = G(x)$, $y = F(x)$ is conditionally dependent on x_i.

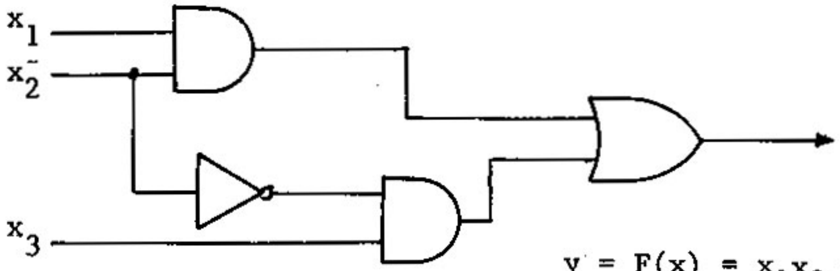
The literature previously published on the Boolean difference also defines operational properties for the difference.

The function is well documented in the literature and is used in some methods of minimal or optimum test set generation.

The Boolean difference has been derived for an example function in Figure 10.20 both by computation and by using a graphical map techique. The map technique shown uses a Marquand map; previous papers have used Karnaugh maps.

An alternative map approach, again using Marquand maps and the example is performed as follows:

1. Map the function $y = F(x)$ (Figure 10.21a).



$$y = F(x) = x_1x_2 + \bar{x}_2x_3$$

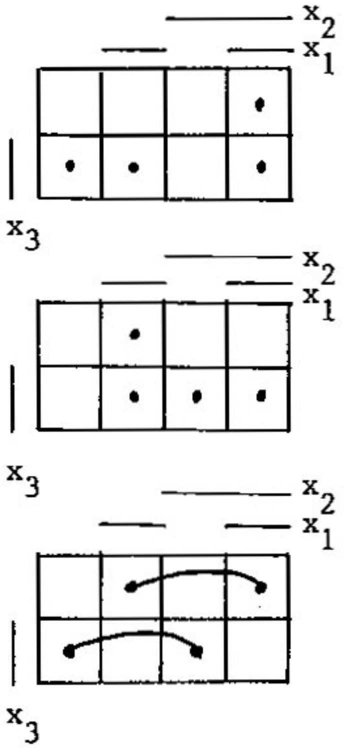
$$D_2F(x) = (x_1x_2 + \bar{x}_2x_3) \veebar (x_1\bar{x}_2 + x_2x_3)$$

$$= (x_1x_2 + \bar{x}_2x_3)(\bar{x}_1 + x_2)(\bar{x}_2 + \bar{x}_3) + (x_1\bar{x}_2 + x_2x_3)(\bar{x}_1 + \bar{x}_2)(x_2 + \bar{x}_3)$$

$$= \bar{x}_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2\bar{x}_3$$

$$= x_1\bar{x}_3 + \bar{x}_1x_3$$

a. Computation solution.



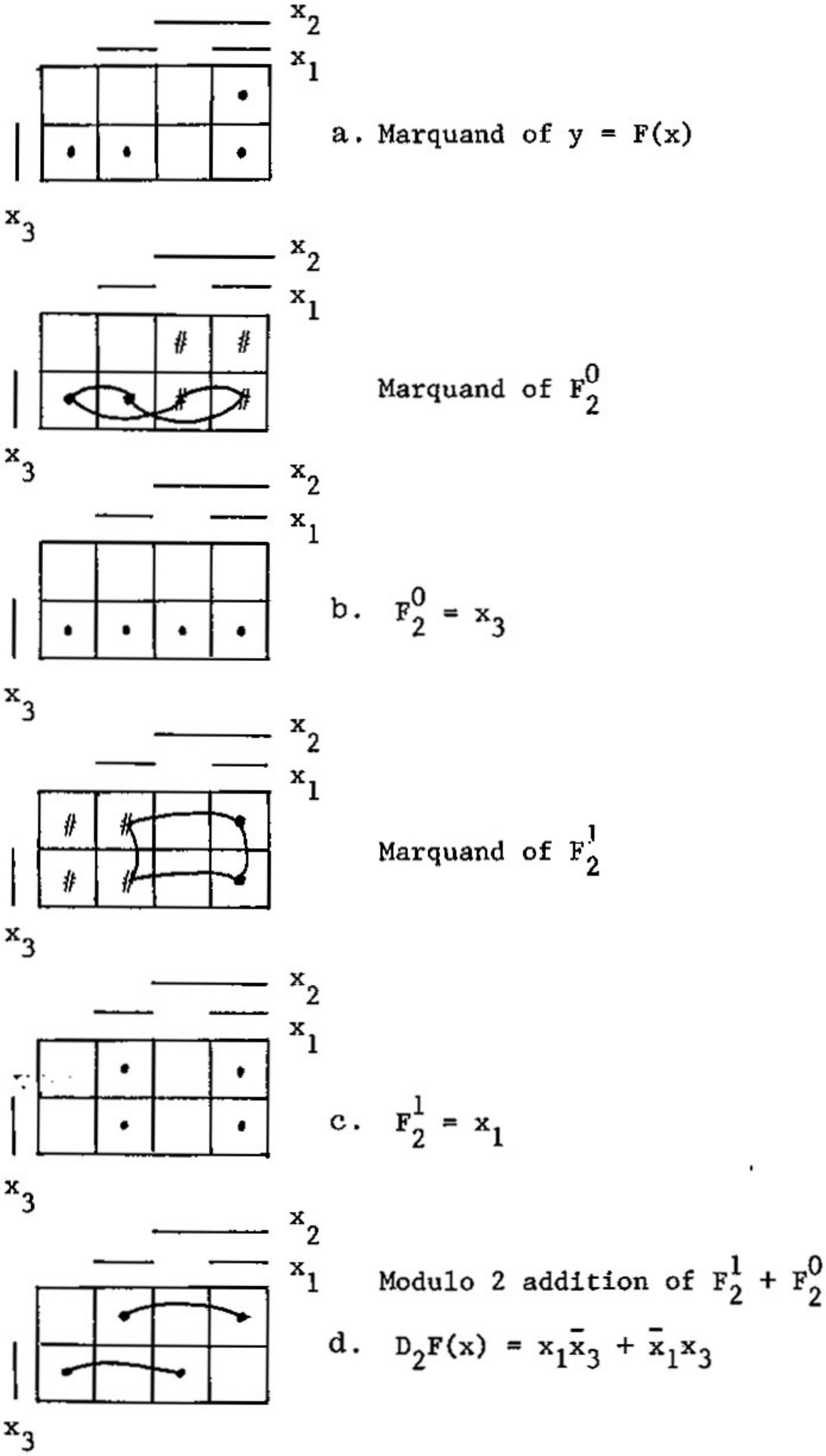
Marquand map of $F(x)_{x_2 = 1}$

Marquand map of $F(x)_{x_2 = 0}$

$$(F(x)_{x_2 = 1} + F(x)_{x_2 = 0})_{\text{Mod } 2}$$

b. Map solution.

Figure 10.20. Boolean Difference by computation and mapping.



a. Marquand of $y = F(x)$

Marquand of F_2^0

b. $F_2^0 = x_3$

Marquand of F_2^1

c. $F_2^1 = x_1$

Modulo 2 addition of $F_2^1 + F_2^0$

d. $D_2F(x) = x_1\bar{x}_3 + \bar{x}_1x_3$

Figure 10.21. An alternative map approach.

2. To find $D_2F(x)$ computationally, we may define

   $$D_2F(x) = F_2^1 \veebar F_2^0$$

   To find F_2^0 consider all points where $x_2 = 1$ to be "don't care"s and minimize the resulting function for F_2^0 (Figure 10.21b).

3. To find F_2^1 consider all points where $\overline{x_2} = 1$ to be "don't care"s and minimize for F_2^1 (Figure 10.21c).

4. The two functions, F_2^0 and F_2^1, must be exclusive OR'd together to find $D_2F(x)$, which is accomplished by the modulo 2 addition of their maps (Figure 10.21d).

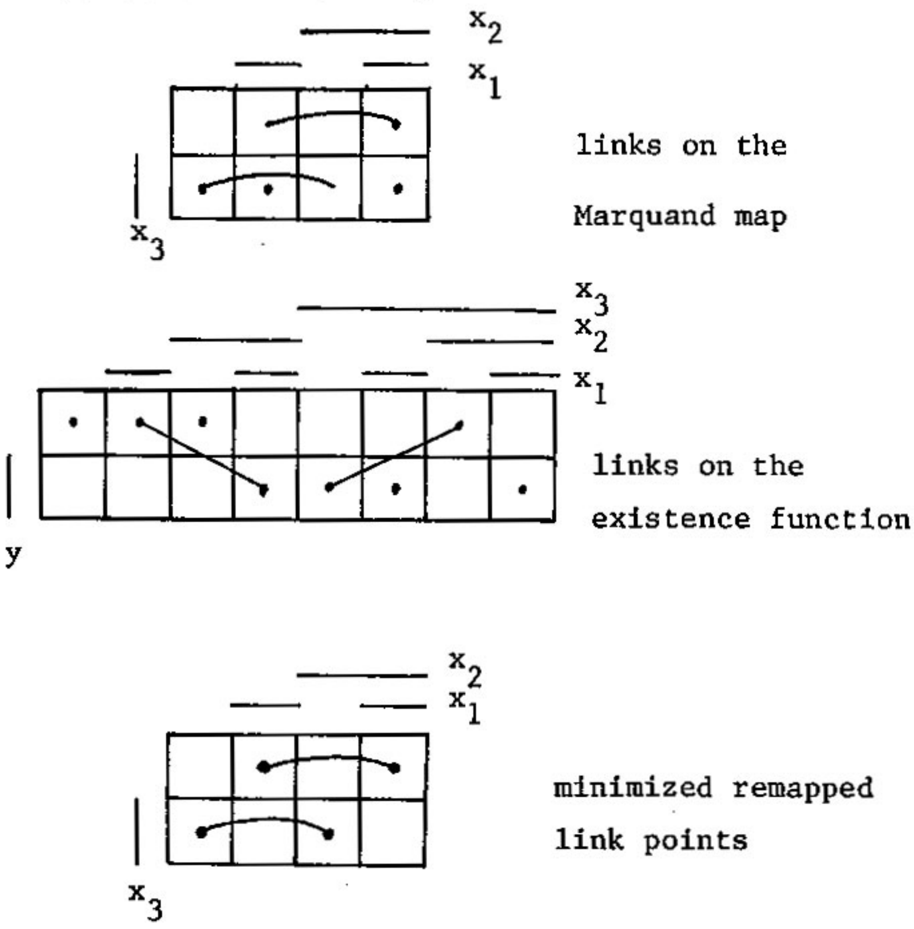
This approach was suggested by Svoboda. The choice of one of these methods over the other is arbitrary, dependent upon the ease of switching x_i with $\overline{x_i}$ and remapping, versus minimization with "don't care" masks. The choice of which will be automated has not been made. The map approach was originally intended for manual use for cases where n was reasonably small.

A third approach involves the use of "links" (defined in "Fault Detection through Parallel Processing in Boolean Algebra", Ph.D. Thesis, UCLA, 1976, by D. E. White). A "link" exists if, for all x_j except the x_i in question held constant, a change in x_i produces a change in y. (Figure 10.22a.) This is best seen using the Existence Function map (Figure 10.22b).

The procedure:

1. Using the Existence Function map, identify all links for the variable x_i.

Find all links for the variable x_i for which the difference is being sought. This is clearer on the Existence function map.



links on the Marquand map

links on the existence function

minimized remapped link points

The minimized function found from the points on the Marquand map is the Boolean Difference being sought.

Figure 10.22. Solution with links and the existence function map.

2. On a new Marquand map, map only those points bound by links for the variable x_i (Figure 10.22c).

3. Minimize the function represented by these points to find $D_iF(x)$.

Given these methods, it is desired to define algorithms for the parallel Boolean processor. The present design limitation will limit these algorithms to cases where $n \le 22$.

Algorithms/routines which already exist for the parallel Boolean processor are referenced but are not detailed here.

The algorithm for the Marquand map approach first described:

1. Load a 2^n Boolean space with $F(x)\big|_{x_i=1}$.

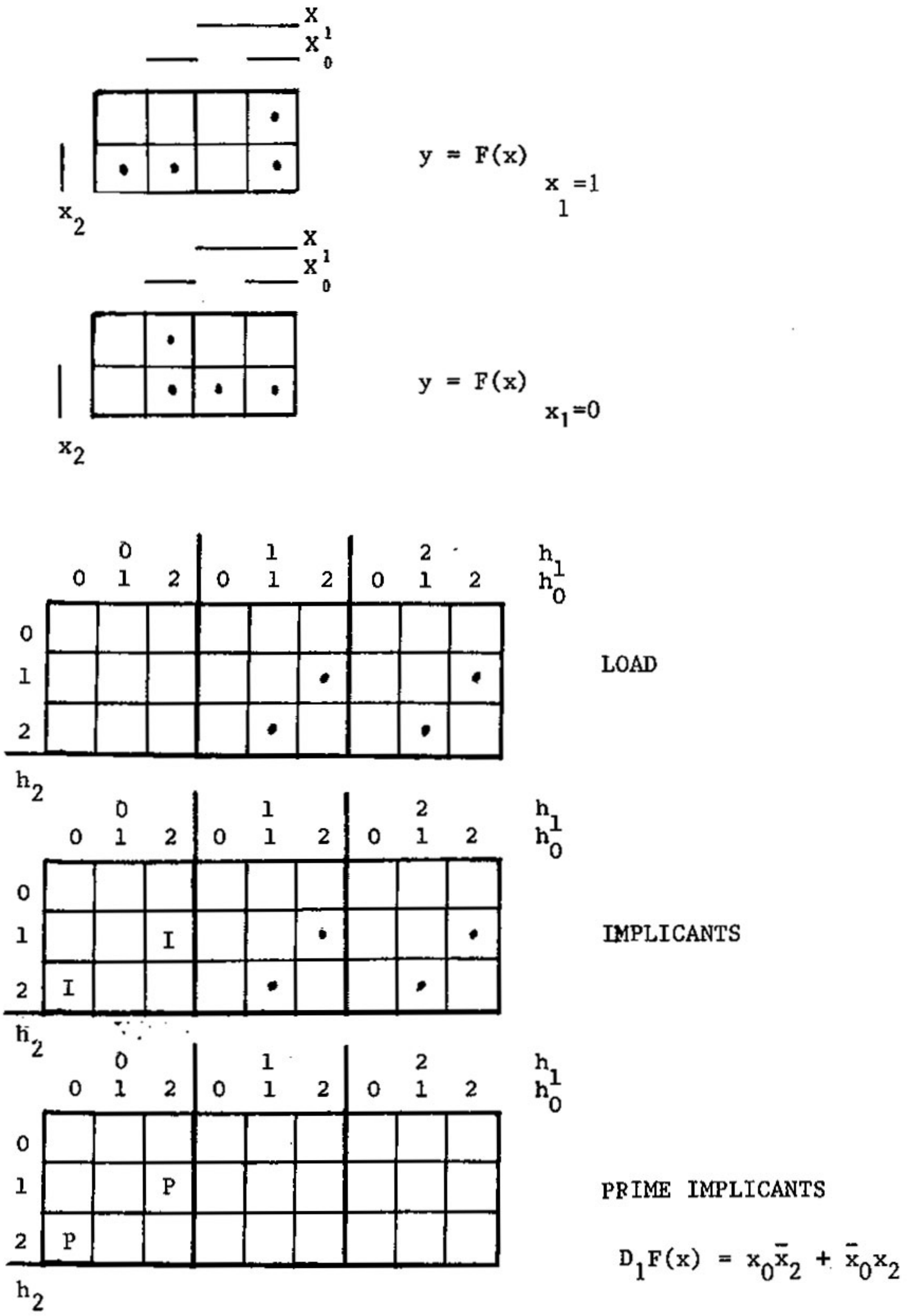
2. Load a second 2^n Boolean space with $F(x)\big|_{x_i=0}$ (software instruction).

3. Using software to add these maps modulo 2, load a 3^n triadic space with the result map.

4. Run the implicant algorithm.

5. Run the prime implicant algorithm. The result is $D_iF(x)$.

The storage required for the maps (spaces) is $2^{n+1} + 3^n$ bits. The algorithm is illustrated in Figure 10.23.



$$D_1F(x) = x_0\bar{x}_2 + \bar{x}_0x_2$$

Figure 10.23. Parallel Boolean Processor algorithm for implementation of map approach.

The algorithm for the link approach is as follows:

1. Load the existence function E(y) into a 2^{n+1} Boolean space.
2. Find the links for the variable x_i (this algorithm is needed for the test sequence generation and is therefore considered to exist).
3. Load a 3^n triadic space with the linked points of E(y).
4. Run the implicant algorithm.
5. Run the prime implicant algorithm. The result is $D_iF(x)$.

The storage required for the maps (spaces) is $2^{n+1} + 3^n$ bits.

A simple example has been used throughout this description. The algorithms have, however, been succesfully applied to a number of examples of varying complexity, including several published in the literature. A more complex example is illustrated in Figures 10.24 (Marquand map method) and 10.25 (link method).

These algorithms have been extended for the generation of the Boolean difference of y with respect to two variables:

(both)

$$D_{ij}F(x) = F(x_1, \ldots , x_i, \ldots , x_j, \ldots , x_n)$$
$$F(x_1, \ldots , \overline{x_i}, \ldots , \overline{x_j}, \ldots , x_n)$$

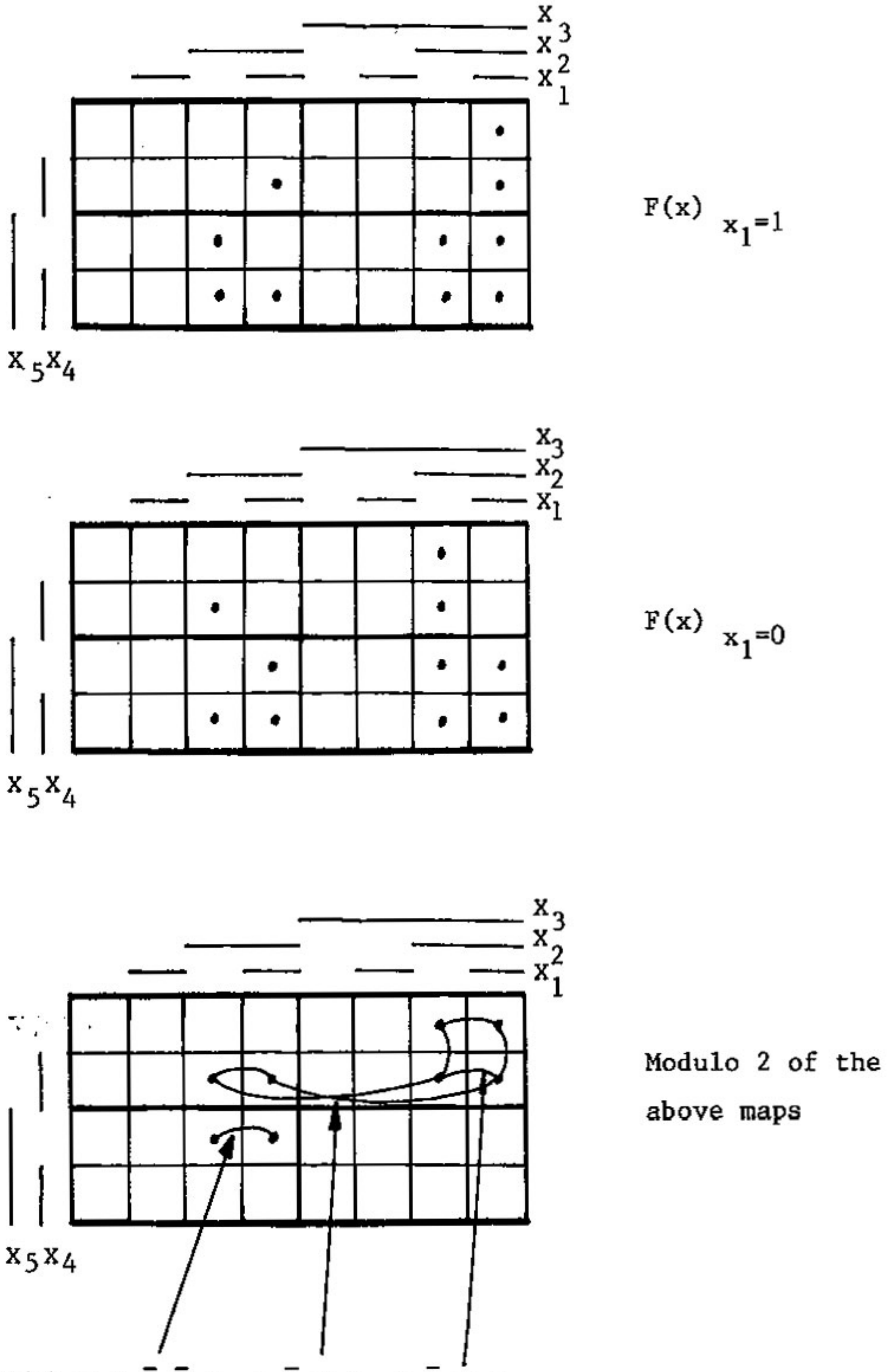
(either)

$$D_{i \veebar j}F(x) = D_iF(x) + D_jF(x)$$

(either or both)

$$D_{i+j}F(x) = D_{ij}F(x) + D_{i \veebar j}F(x)$$

$$y = F(x) = x_1x_2x_3 + x_1x_2x_4 + \bar{x}_1x_2x_5$$



$$D_1F(x) = x_5\bar{x}_4\bar{x}_3x_2 + \bar{x}_5x_4x_2 + \bar{x}_5x_3x_2$$

Figure 10.24. Parallel Boolean Processor algorithms example.

By computation:

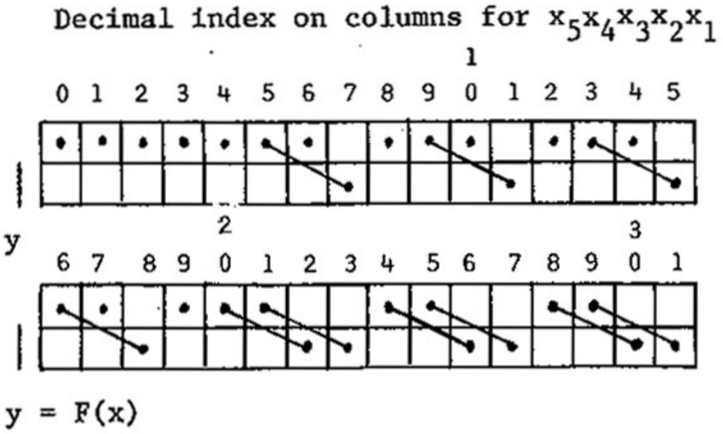
$$\begin{aligned}
D_1F(x) &= D_1 [\ x_1x_2x_3 + x_1x_2x_4 + \bar{x}_1x_2x_5\ ] \\
&= (\ x_1x_2x_3 + x_1x_2x_4 + \bar{x}_1x_2x_5\ ) \forall (\ \bar{x}_1x_2x_3 + \bar{x}_1x_2x_4 \\
&\quad + x_1x_2x_5\ ) \\
&= (\ x_1x_2x_3 + x_1x_2x_4 + \bar{x}_1x_2x_5\ )(\ x_1 + \bar{x}_2 + \bar{x}_3\ )(\ x_1 \\
&\quad + \bar{x}_2 + \bar{x}_4\ )(\ x_1 + \bar{x}_2 + \bar{x}_5\ ) + (\ \bar{x}_1x_2x_3 + \bar{x}_1x_2x_4 \\
&\quad + x_1x_2x_5\ )(\ \bar{x}_1 + \bar{x}_2 + \bar{x}_3\ )(\ \bar{x}_1 + \bar{x}_2 + \bar{x}_4\ )(\ x_1 \\
&\quad + \bar{x}_2 + \bar{x}_5\ )
\end{aligned}$$

$$\begin{aligned}
&= x_1x_2x_3\bar{x}_5 + x_1x_2x_4\bar{x}_5 + x_1x_2\bar{x}_3x_4x_5 + x_1x_2x_3\bar{x}_4\bar{x}_5 \\
&\quad + \bar{x}_1x_2\bar{x}_3\bar{x}_4x_5 + x_1x_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1x_2x_3\bar{x}_5 + \bar{x}_1x_2x_4\bar{x}_5 \\
&\quad + \bar{x}_1x_2\bar{x}_3x_4\bar{x}_5 + \bar{x}_1x_2x_3\bar{x}_4\bar{x}_5
\end{aligned}$$

$$= x_2\bar{x}_5\ (\ x_3 + x_4\ ) + x_2x_5\bar{x}_3\bar{x}_4$$

Ref: Sellers, et. al. "Analyzing Errors with the Boolean Difference." *IEEE Trans.*, C-17(4):676-683, July 1968; correction C-20(11);1245-1251, November 1971.

Figure 10.24. (con't)



	x_3	0									1									2								
	x_2	0			1			2			0			1			2			0			1			2		
x_5x_4	x_1	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
0	0																											
	1																											
	2																											
1	0																											
	1																											
	2																					I	•	•				
2	0													I	I	I												
	1				I	I	I							I	•	•						I	•	•				
	2													I	•	•												

points of links I implicants I prime implicants

$$D_1F(x) = x_5\bar{x}_4\bar{x}_3x_2 + \bar{x}_5x_4x_2 + \bar{x}_5x_3x_2$$

Figure 10.25. The solution with links.

and may be extended to all n variables. No work has yet been done to extend them to the partial Boolean difference.

A final example is given in Figure 10.26.

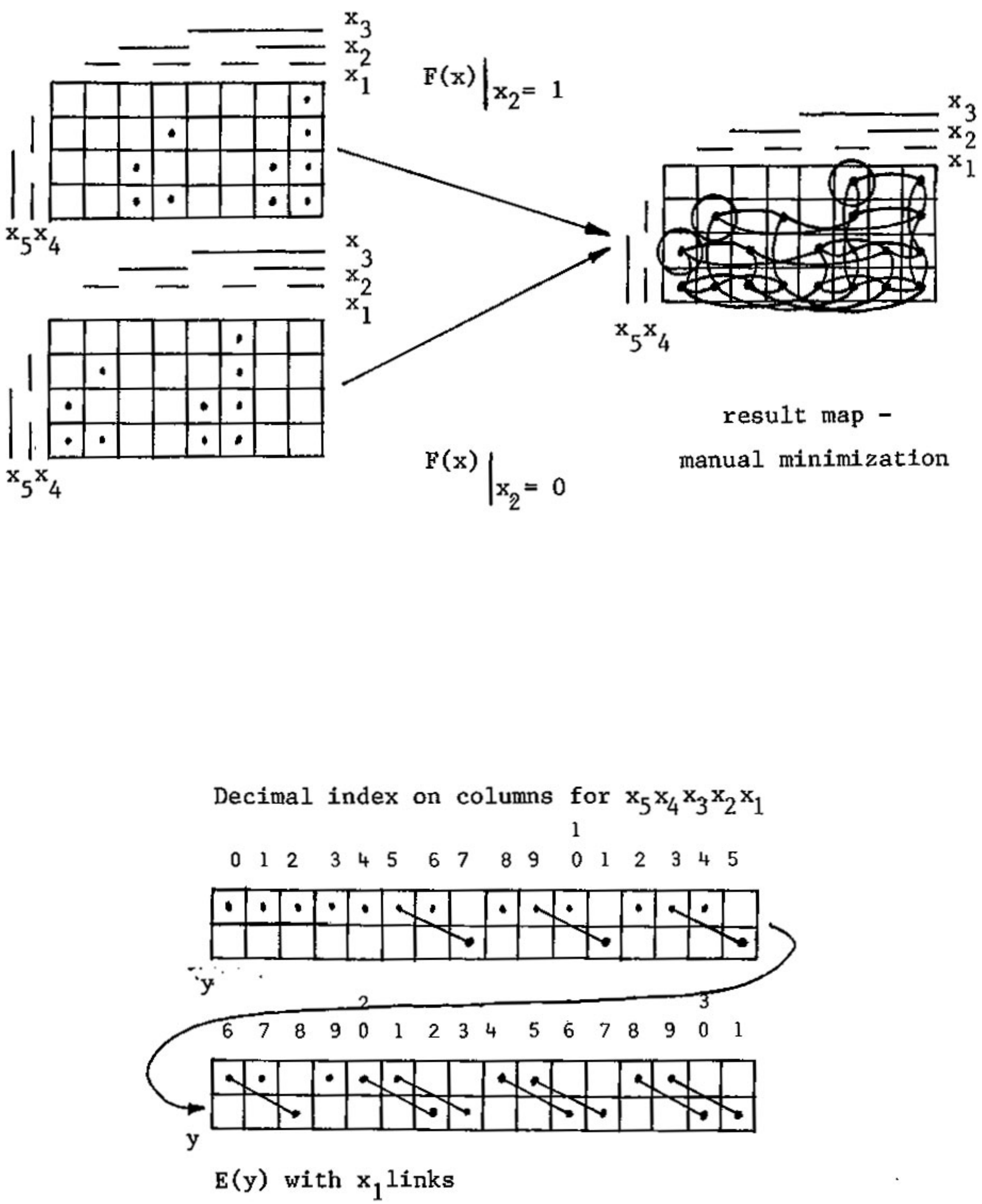


Figure 10.26. Another example difference from Sellers.

	x_3	0									1									2								
	x_2	0			1			2			0			1			2			0			1			2		
x_5x_4	x_1	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
0	0											I			I			I										
	1		I			I			I			I			I			I			I			I			I	
	2											I			I			I										
1	0			I			I			I	I	I	I	I	I	I	I	I	I			I			I			I
	1	I	I	I	I	I	I	I	I	I	I	I	I	I	•	•	I	•	•	I	I	I	I	•	•	I	•	•
	2			I			I			I	I	I	I	I	•	•	I	•	•			I			•			•
2	0											I			I			I										
	1		I									I			•			•				I		•			•	
	2											I			•			•										

points for links showing implicants (I) and prime implicants

$$D_2F(x) = x_1x_3 + x_1x_4 + \bar{x}_1x_5$$

Figure 10.26. (con't)