Winter 1985

The Direct Summation of Totally Self-Checking Checkers

Stefean Andrew Fedyschyn
Old Dominion University

THE DIRECT SUMMATION OF TOTALLY SELF-CHECKING CHECKERS

by

Stefan Andrew Fedyschyn
B.S.E.E., June 1978, United States Naval Academy

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

OLD DOMINION UNIVERSITY
December 1985

Approved by:

Sharad V. Kanetkar (Director)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

THE DIRECT SUMMATION OF
TOTALLY SELF-CHECKING CHECKERS

Stefan Andrew Fedyschyn
Old Dominion University, 1985
Thesis Director: Dr. Sharad V. Kanetkar

A new technique for reducing the complexity of designing Totally Self-Checking (TSC) checkers for m-out-of-n codes is presented. The method is based on the partitioning of the input variables into r classes, then partitioning the code groups generated into Z output partitions. Comparison with earlier results reveals improvements in design simplicity and logic and testing complexity.

This thesis also presents a new method of TSC checker design where a j-level m_1/n_1 code and a k-level m_2/n_2 code TSC checker are directly summed to form a max[j,k]-level $(m_1 + m_2)/(n_1 + n_2)$ TSC checker. A library of m/n code TSC checkers can then be used as building blocks for other m/n code TSC checkers.

Chapter One

INTRODUCTION

The need to communicate reliable information is an important requirement in a wide range of systems. Many methods have been developed to increase reliability. One of these methods is the practice of encoding the information. Another is the use of circuitry that checks the coded input and determines if it belongs to the set of code words that are being used. Some circuits of this type are called Totally Self-Checking (TSC) checkers.

Totally self-checking checkers are the subject of many papers presenting many design strategies. The importance of these circuits in fault-tolerant computers has received considerable attention. Fault tolerant devices use these checkers for fault detection. These circuits are used to detect the presence of an error. The error these checkers detect is an error in the structure of the code used. Their ability to detect an error extends to detecting errors within themselves. If a TSC checker cannot properly differentiate between code words and non code words, because of an error in the checker itself, the checker will indicate this as an error also. TSC checkers accomplish this through the use of coded inputs.

These circuits are usually designed for a specific class of code. This class of code is called the m-out-of-n (m/n) code. These codes are used because of their ability to detect errors which have the same direction, i.e., errors that make ones into zeros or zeros into ones but not both. The code is defined as each code word having a length of n made up of m ones and n-m zeros. For example, the set of 3-out-of-5 (3/5) code words is (11100, 11010, 11001, 10110, 10101, 10011, 01110, 01101, 01011, 00111).

Used in communications and computer systems, these circuits are placed in systems on data lines where reliability is important. A totally self-checking checker is designed to detect errors in code words used for transmitting information between a transmitter and a receiver. Therefore, they can be used in any device that needs to send reliable information from one point to another.

Many design strategies have been presented. One design, forwarded by Gaitanis and Halatsis [1], shows results that are a significant improvement in logic and testing complexity over earlier designs. Their method involves partitioning the input variables, the formation of a product (AND gate) array and partitioning the array's output. Systematic separation of the array's output variables into output separation partitions by specific rules generates the TSC checker. This generation of the output separation partitions is the key factor in the design. A similar design, forwarded by Halatsis, Gaitanis and Sigala [2], translates m-out-of-(2m+-1) codes to a 1-out-of-2 code in three gate levels.

It is because of the results obtained in those two papers that this research took its direction. The work accomplished here presents a method of obtaining comparable results, with improvements in some cases, and an overall improvement in the simplicity of design. This thesis also extends past work done with a new method of checker design for m/n codes not previously considered.

The first method described here partitions the input variables and forms the product array in the same manner as in [1]. This thesis presents a new method of partitioning the product array's output variables, i.e., a new method of determining the output separation partition.

The second method presented combines TSC checkers made with a product array by a recursive technique never before considered. By a direct summation method a m_1/n_1 code TSC checker and a m_2/n_2 code TSC checker generate a $(m_1+m_2)/(n_1+n_2)$ code TSC checker.

This method allows TSC checkers to be designed, for a large range of m/n codes, in only three gate levels. The only requirement is for both checkers to have exactly two output separation partitions each. The method is then generalized to construct new TSC checkers from checkers having any number of output separation partitions.

This new method simplifies the design and, as an added benefit, the calculation of the design cost is greatly simplified. The cost of the design is determined by the number of gates and gate input lines necessary in the circuit. These two methods show cost improvements over earlier totally self-checking checker designs. The design presented here lends itself to directly calculating the size of the product array, the number of input lines in the array, and the number of code words required to test the checker from a simple polynomial expansion.

Included as an appendix are results of research completed that supplement this thesis. It forwards original work that can serve as a starting point for future research.

The notation used throughout is consistent with previous work.

Chapter Two

DEFINITIONS

2.1 Fault Model

The design of checking circuits necessitates the identification of the types of errors that are most likely to be encountered. In previous works, the logical fault model used is line stuck-at faults. Stuck-at-one (s-a-1) and stuck-at-zero (s-a-0) faults have been used with great success in single fault and multiple fault designs. This fault modelling is successful because most failures in a circuit have the same effect as a circuit line being stuck at a logical value.

The term fault will refer to one or more lines of a circuit s-a-1 or s-a-0. Faults only go forward in a circuit and have the effect, in the model, of causing gate outputs to be independent of network inputs. A single fault circuit is defined as a circuit having only one stuck-at line and a multiple fault circuit is defined as a circuit having more than one stuck-at line. A unidirectional fault is defined as a multiple fault where all the stuck-at lines are stuck-at the same logical value. For example, all faults are either s-a-1 or s-a-0. A fault set is defined as the set of faults currently under examination. An error is defined as the incorrect logic signal caused by a fault. A fault does not necessarily produce an error for a particular input. Logical operations of the gates are unaffected.

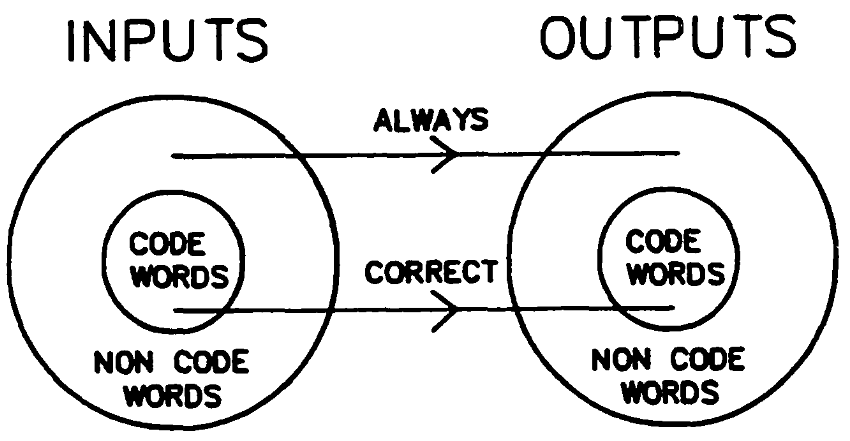
Because of the number of possible faults a circuit can experience, previous work in this area has confined the fault set, which we require the circuit to detect, to single or unidirectional faults. This thesis confines itself to consider this same prescribed fault set.

2.2 Totally Self-Checking Model

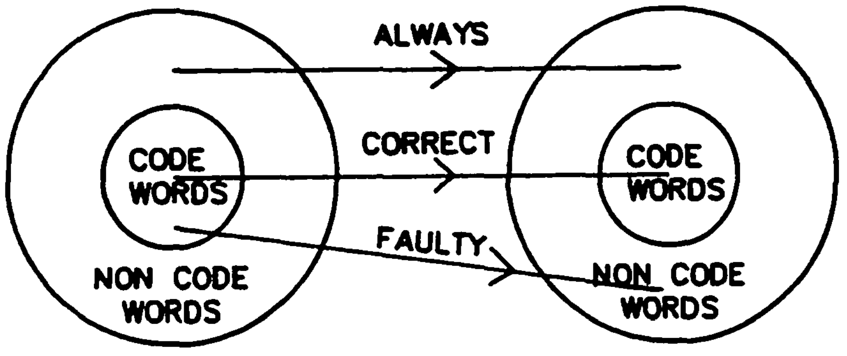
Totally self-checking checkers have previously been defined by Anderson and Metze [3]. A totally self-checking circuit must have at least two outputs, and no output may take a constant logical value for code word inputs. The requirement for no constant valued output permits the checking for a s-a-1 fault at that output. It follows that the circuit must have at least two output values for code word inputs. At least one code word input should be mapped to each code word output. It has been formalized by Anderson [3] and used in all work since that, for a two output checker, a code word input will map to either output (1,0) or (0,1) and a non code word input will map to either output (0,0) or (1,1).

Definition 2-1: A code disjoint circuit maps code word inputs to code word outputs and non code word inputs to non code word outputs. See Figure 1(a).

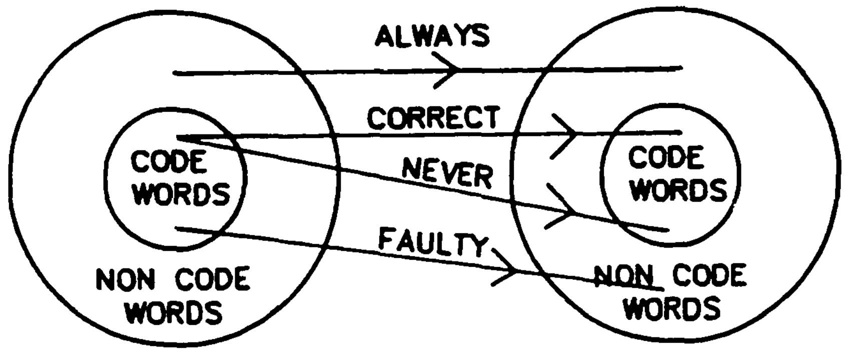
Using this definition, a checker can be seen as a m-out-of-n to a 1-out-of-2 code disjoint translator. Most TSC checker designs realize this translation in one of two



a) Code Disjoint Property



b) Self–Testing Property



c) Fault Secure Property

PROPERTIES OF A TSC CHECKER
FIGURE 1

designs. Both divide the checker into three subcheckers.

The first,

a) m/n to 1/(n/m) code translator,

b) 1/(n/m) to k/2k code translator,

and c) k/2k to 1/2 code translator.

The second,

a) m/n to 1/Z (Z = 4, 5, or 6),

b) 1/Z to 2/4 code translator,

and c) 2/4 to 1/2 code translator.

A totally self-checking checker has been defined by Anderson and Metze in [3] as follows:

Definition 2-2: A circuit is self-testing if for every fault from a prescribed set, the circuit produces a non code word output for at least one code word input. See Figure 1(b).

Definition 2-3: A circuit is fault-secure if for every fault from a prescribed set, the circuit never produces an incorrect code word output for a code word input. See Figure 1(c).

Definition 2-4: A circuit is totally self-checking (TSC) if it is both self-testing and fault secure.

With the above definitions it can be seen that a circuit can be diagnosed using only code word inputs. For example, the set of all code word inputs can detect every fault from the prescribed set of faults. When a circuit receives its entire input space with just the application of code word inputs, the circuit is defined to be fully exercised.

2.3 Test Set

As seen from the definitions above, an important criteria and capability is the circuit's ability to test itself. It has been noted that a circuit can be diagnosed using only code word inputs and that the entire set of code words can detect the entire set of prescribed errors. However, in some TSC checkers it is not necessary to use all the code words as inputs to detect all the errors. The minimum size set of input code words that are required to insure the correct operation of the checker is called the test set. It is obvious that the smaller the test set, the faster and easier it becomes to "test" a checker's operation. For m/n codes with many code words, a reduction in the number of tests necessary to insure correct operation may be significant.

The objective of using a test set is to insure that there are no s-a-0 or s-a-1 faults in the circuit. This is done by insuring that every gate input and every gate output is enabled at least once while conducting the test.

2.4 Majority Detection Logic

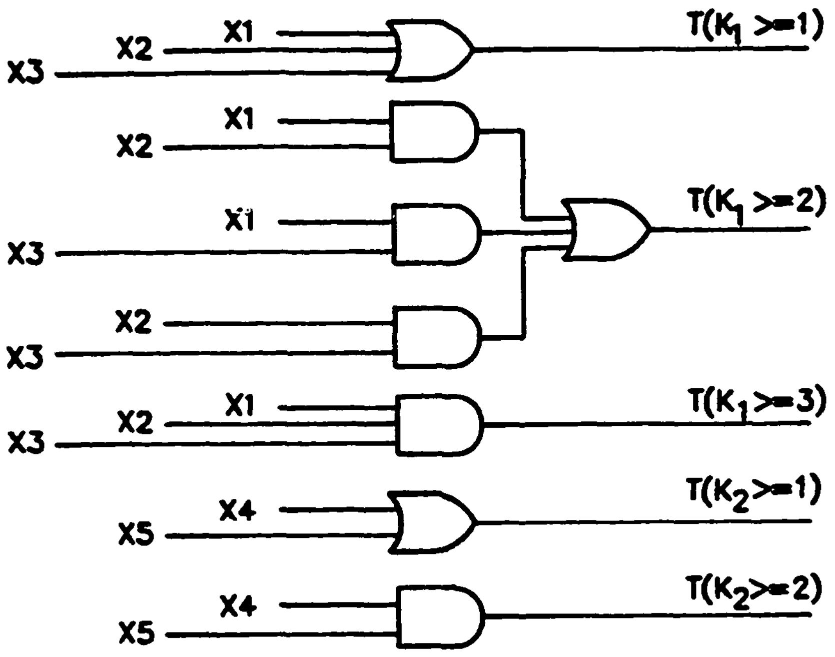
A large number of TSC checker designs use majority logic detection circuits. These circuits and some methods of their implementation are presented.

Consider a set of A input bits, where $A = (x_1, x_2, \ldots, x_n)$. The number of bits in the set is n. The number of ones occurring at any one time is k. The majority function defined as T(k>=i), where 1 =< i =< n, is to be equal to 1 if and only if the condition in the parenthesis is true. There are two general ways to implement this function. The first is minimum gate level design and the second is multilevel cellular (minimum gate) design. See Figure 2. These two methods vary considerably in size and the number of code word inputs needed to test their correct operation. See Tables 1 and 2.

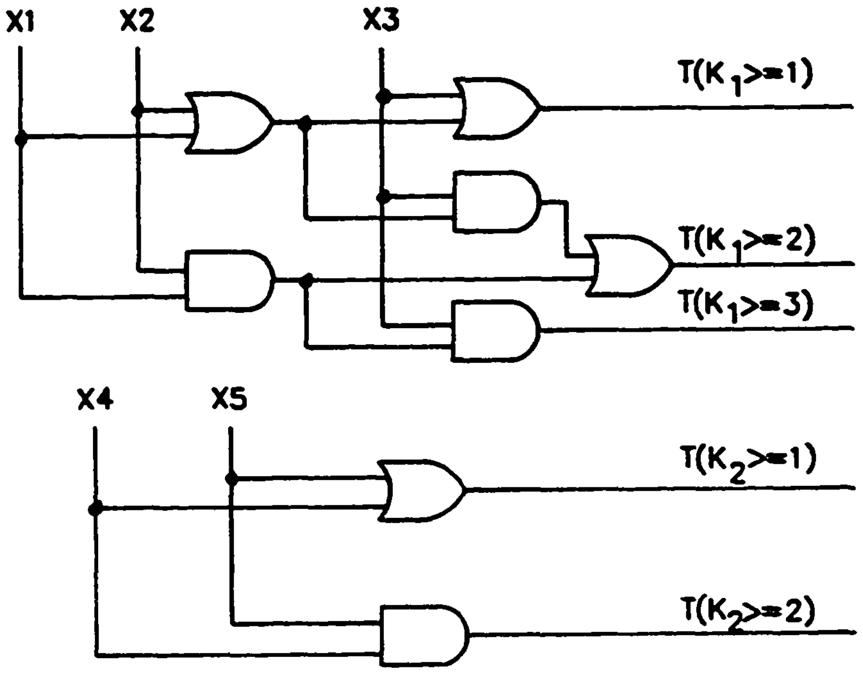
Using a m-out-of-n code as an example, a minimum gate level realization will be described. It is possible to construct a 2-level AND-OR implementation of the function $T(k>i)$, $1 =< i =< n$. There are ${}_nC_i = n!/i!(n-i)!$ AND gates with i inputs for $1 =< i =< n$. The outputs of these AND gates are then the inputs of a single OR gate for each value of i for a total of n OR gates. For example, the implementation of $T(k>=3)$ for $A = (a_1, a_2, a_3, a_4)$ shows

$$T(k>=3) = a_1a_2a_3 + a_1a_2a_4 + a_1a_3a_4 + a_2a_3a_4$$

For this implementation of $T(k>=3)$ there are four AND gates and the one OR gate. When $i = 1$, all n lines are inputs to a single OR gate. When $i = n$, all lines are inputs to one AND gate and there is no second level OR gate. It is obvious that each AND gate realizing $T(k>=i)$ has i inputs.



a) Minimum Gate Level Design



b) Multilevel Cellular Design

MAJORITY DETECTION LOGIC
FIGURE 2

Chapter Three

DESIGN PROCEDURE

3.1 Philosophy

A TSC checker is fundamentally a data reduction circuit that translates ${}_nC_m$ m/n code words into two 1/2 code words. This reduction must preserve the integrity of the information that describes the correctness of the inputs and the TSC checker itself. This reduction should also be realized in a manner to minimize gate delay and hardware costs.

A valuable characteristic of m/n codes is that they have a constant number of ones in each code word. It follows that the first step in reducing the code space and the easiest procedure to determine the presence of an error is to count the number of ones. This is accomplished by using majority detection logic.

As stated earlier, there are two realizations of majority logic, minimum level and multilevel cellular. The minimum level design has the smaller gate delay and the multilevel cellular design has the smaller hardware cost. We can design for both of these characteristics by partitioning the input variables.

The input variables are partitioned, according to a few criteria, to make tradeoffs between gate delays and hardware costs. Depending on the sizes of the partitions, the code space can be greatly reduced in this one step. Since we are counting the number of ones, this information is not lost by such a procedure. By summing outputs from the majority logic we can determine the number of ones in the input.

The comparison is done by an array of AND gates. These gates form a product array. Each AND gate has an input from the majority logic outputs of every different partition. These inputs are the distributions that add to m for a m/n TSC checker. When a valid code word is the input, one and only one AND gate will have an output.

We can see that if there are less than m ones there will be no output from any AND gate and if there are more than m ones there will be an output from more than one AND gate. The presence of a s-a-0 or s-a-1 fault can be seen to have the same effect, respectively. When the inputs and the circuit are correct one and only one partition will have an output.

The outputs from this array are again partitioned for a further reduction step. This partitioning must be done in such a manner that any error up to this point will produce either no outputs from any partition or cause more than one partition to have an output. And when there is no error present one and only one partition will have an output.

This manner of partitioning and reducing is continued until there are only two outputs. At this point any code word is mapped to (0, 1) or (1, 0) and any non code word or error is mapped to (0, 0) or (1, 1).

3.2 Design

The TSC checker presented here is of the same general form as the one forwarded by Gaitanis and Halatsis [1]. It consists of two major circuits, c_1 and c_2. Circuit c_1 is a m/n to 1/Z code translator and circuit c_2 is a 1/Z to 1/2 code translator. See Figure 3a.

Circuit c_1 consists of three subcircuits, c_{11}, c_{12}, and c_{13}. Circuit c_{11} is the majority detection logic. Circuit c_{12} implements product functions and is an array of AND gates. Circuit c_{13} is an output reducing circuit and is an array of OR gates. See Figure 3b.

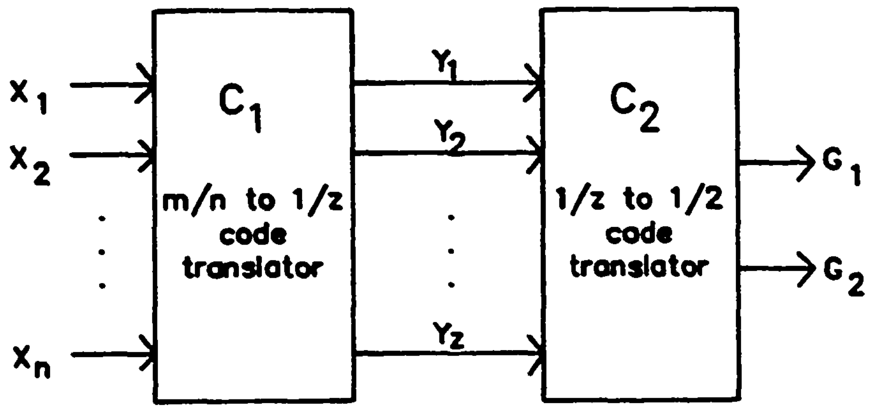
Anderson and Metze [3] forwarded a theorem for designing TSC checkers by the interconnection of subcircuits.

Theorem 3-1: A circuit c consisting of an interconnection of subcircuits is TSC for unidirectional faults if it contains no inverters and each subcircuit is

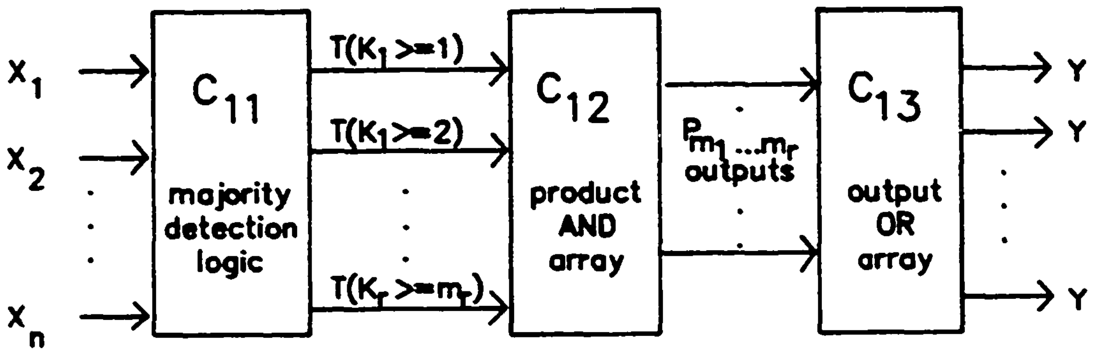
1) TSC for unidirectional faults,
2) code disjoint,
3) fully exercised,
4) input encoded with a completely unidirectional error detection code (e.g., m/n code).

If the circuit c contains inverting gates, then it is TSC only for single faults.

Two theorems that establish conditions for the proper design of the TSC checker follow from this and were first forwarded as propositions in [1]. Theorem 3-2 insures s-a-0



a) Major Subcircuits



b) Subcircuits of C_1

PROPOSED TSC CHECKER

FIGURE 3

faults are detected and Theorem 3-3 insures s-a-1 faults are detected.

Theorem 3-2: For any m/n code ($m >= 2$, $n >= 4$), c_{11} concatenated by c_{12} (c_{11} @ c_{12}) is a code disjoint circuit if and only if $n_i =< m$ for every $i = 1$ to r.

Theorem 3-3: For any m/n code ($m >= 2$, $n >= 4$) the circuit c_{11} @ c_{12} is a self-testing checker if and only if

$$n_i =< m =< n - n_i \text{ over all } i = 1 \text{ to } r.$$

Consider an input partition p(A) on the set A of input variables, $A = (x_1, x_2, \ldots, x_n)$, into r subsets, i.e.,

$$p(A) = (A_1, A_2, \ldots, A_r).$$

Let $n_i =< m$, $i = 1$ to r, be the number of variables in subset A_i. Let k_i be the number of variables in A_i that are equal to 1.

Circuit c_{11} uses majority logic detection circuitry to implement the majority functions

$$T(k_i>=m_j),\ i = 1 \text{ to } r \text{ and } m_j = 1 \text{ to } n_i.$$

Circuit c_{12} uses AND gates to form the product array implementing the product functions. The product functions $P_{m_1 m_2 \ldots m_r}$ are defined as the logical product of majority functions such that

$$P_{m_1 m_2 \ldots m_r} = T(k_1>=m_1) * T(k_2>=m_2) * \ldots * T(k_r>=m_r)$$

where

$$m_1 + \ldots + m_r = m$$

and

$$\max(m-n+n_i, 0) =< m_i =< n_i,\ i = 1 \text{ to } r.$$

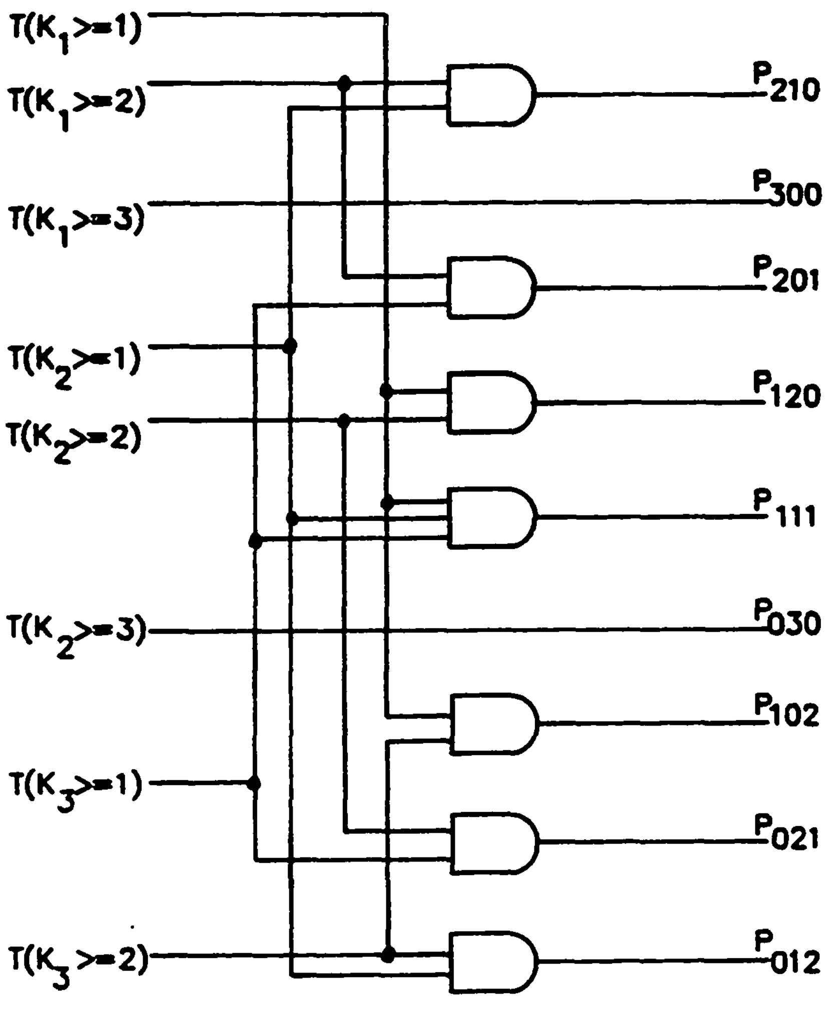
The product functions are combinatorial distributions of the outputs from the majority logic for the m/n code words over p(A). For example, a 3/8 code with r = 3 partitions of size n_i(3,3,2), has the product functions:

P_{300}, P_{210}, P_{201}, P_{120}, P_{111}, P_{102}, P_{030}, P_{021} and P_{012}. See Figure 4.

For a TSC checker with this type of realization, the size of the test set is the number of distributions of the code words over the input partitions. The test set is the set of code words which are inputs to the majority logic that generate an output from each AND gate and those functions not requiring an AND gate, i.e., those functions in a m/n code that correspond to $P_{00\ldots m_i\ldots 0}$, $m_i = m$. These inputs test the entire circuit and are generated at the same time as the TSC checker itself with no additional calculations. The test is the required input for the circuit to demonstrate the self-checking property.

To be self-checking requires the circuit to produce an output of 1, from every gate in the circuit, at least once for some code word input. It also requires the circuit not to produce an output, i.e., an output of 0, from every gate in the circuit, at least once for some word input. These two requirements test for s-a-0 and s-a-1, respectively.

The size of the input partitions must allow the two requirements, testing for s-a-0 and s-a-1, to be realized. For an m/n code, if the partition's size, n_i, is larger than m, the section of the majority logic that



PRODUCT FUNCTIONS' REALIZATION

FIGURE 4

implements $T(k_i>=m+1)$ can never be tested for s-a-0. In this event, a non code word input of $(m+1)/n$ may not be mapped to a non code word output.

Conversely, if the number of partitions is too small, it will not allow each partition's majority logic to have an output of 0, at least once, for some code word input. A s-a-1 fault may not be detected. For example, a 5/6 code TSC checker with partitions $n_i(2,2,2)$ has product functions corresponding to [(2,2,1), (2,1,2), (1,2,2)]. If the output line of any of the $T(k_i>=1)$ majority logic OR gates is s-a-1, none of the code words will detect this fault. Therefore, at least one 4/6 non code word will produce a valid code word output.

From the above it [illegible] $r >= \max(\lceil n/m \rceil,$ that the number of input partitions r should satisfy

$$r >= \max(\lceil n/m \rceil, \lceil n/(n-m) \rceil).$$

(The notation $\lceil\ \rceil$ means to take the next highest whole number, and $\lfloor\ \rfloor$ means to truncate.)

The output reducing circuit, c_{13}, is an array of OR gates. The partitioning of the set of product functions, i.e., the outputs from c_{12}, results in forming the output separation partition B and determines the number, z, of OR gates.

$$B = (B_0, B_1, \ldots, B_{z-1}).$$

The method of partitioning the product functions will be presented in Chapter Four. A maximum lower bound for z as a function of r was established in [1]. This bound was given for odd and even values of r as

$$z =< 3 * 2^{u-1} \text{ for } r = 2u + 1$$

$$z =< 2^{u} \text{ for } r = 2u.$$

Circuit c_2 is a 1/Z to 1/2 code translator. It is now obvious that Z = z. Circuit c_1 converts the m/n code to a 1/Z code where Z satisfies the following condition [1]

$$\min(n-m, m) >= \lceil n/Z \rceil .$$

For some m/n codes, Z may take the value of 3. To date, there have been no TSC checkers designed using standard design procedures. However, two 1/3 TSC checkers designs were presented, David [4] and Golan [5], that do not fulfill our requirement for minimization. The first uses sequential circuits and the second uses a combined fixed weight code. Neither of these designs serve the purpose of cost efficiency, compared to designs using Z = 4. Therefore, where Z = 3, we will use Z = 4 in the design.

For a m/n code, there are three types of inputs that need to be considered. The first, where

$$k_1 + k_2 + \ldots + k_r = m, \text{ a code word.}$$

The second, where

$$k_1 + k_2 + \ldots + k_r < m, \text{ a non code word.}$$

And the third, where

$$k_1 + k_2 + \ldots + k_r > m, \text{ a non code word.}$$

The first type is the required m/n code word input. The checker maps the code word input to a code word output, either (0,1) or (1,0).

The second type is a non code word input that maps into the non code word type output (0,0). Since

$$\sum_{i=1}^{r} k_i < m$$

no product function is realized and all outputs are 0.

The third type is a non code word input that is designed to map into the non code word output (1,1). To realize this mapping at least one product function that maps to (0,1) and at least one product function that maps to (1,0) must produce an output. Specifically, for every non code word that corresponds to $k_1k_2...k_r$ with

$$k_1 + k_2 + ... + k_r = m+1,$$

there are at least two product functions

$$P_{k_1k_2...k_i-1...k_r}$$

and

$$P_{k_1k_2...k_j-1...k_r}$$

where

$$k_1 + k_2 + ... + k_i-1 ... + k_r = m$$

and

$$k_1 + k_2 + ... + k_j-1 ... + k_r = m$$

that produce an output of 1. This leads to the following.

Definition 3-1: The error cover set $E_{k_1k_2...k_r}$ is the set of product functions that become equal to 1 for every non code word that corresponds to $k_1k_2...k_r$ with $k_1 + k_2 + ... + k_r > m$ and $k_i =< m$ for every $i = 1$ to r.

Example (continued): The error cover sets for the circuit c_{11} @ c_{12} of the 3/8 code, with $r = 3$ partitions of size $n_i(3,3,2)$, are:

$E_{310} = (P_{300}, P_{210})$

$E_{301} = (P_{300}, P_{201})$

$E_{220} = (P_{210}, P_{120})$

$E_{211} = (P_{210}, P_{201}, P_{111})$

$E_{202} = (P_{201}, P_{102})$

$E_{130} = (P_{120}, P_{030})$

$E_{121} = (P_{120}, P_{111}, P_{021})$

$E_{112} = (P_{111}, P_{102}, P_{012})$

$E_{031} = (P_{030}, P_{021})$

$E_{022} = (P_{021}, P_{012})$

Definition 3-2: The fault cover set $F_{m_1 m_2 .. m_i * .. m_r}$ is the set of product functions that become one in the event of a s-a-1 fault in the i-th path of $P_{m_1 m_2 .. m_i * .. m_r}$ for any of the code words that correspond to the following r-digit numbers:

$(m_1+1)m_2 \ldots m_{i-1}(m_i-1)m_{i+1} \ldots m_r$

$m_1(m_2+1)m_3 \ldots m_{i-1}(m_i-1)m_{i+1} \ldots m_r$

:

$m_1 \ldots m_{i-2}(m_{i-1}+1)(m_i-1)m_{i+1} \ldots m_r$

$m_1 m_2 \ldots m_i \ldots m_r$

$m_1 \ldots m_{i-1}(m_i-1)(m_{i+1}+1)m_{i+2} \ldots m_r$

:

$m_1 \ldots m_{i-1}(m_i-1)m_{i+1} \ldots m_{r-1}(m_r+1)$.

Definition 3-3: $F_{m_1 m_2 \ldots (m_j+1) h \ldots (m_i-1) \ldots m_r} =$ $F_{m_1 m_2 \ldots m_j \ldots m_i * \ldots m_r}$ for every product function $P_{m_1 m_2 \ldots (m_j+1) \ldots (m_i-1) \ldots m_r}$ contained in the fault set $F_{m_1 m_2 \ldots m_j \ldots m_i \ldots m_r}$.

Example (continued): The fault cover sets for the circuit c_{11} @ c_{12} of the 3/8 code are:

$F_{3*00} = F_{21*0} = F_{201*} = (P_{300}, P_{210}, P_{201})$.

$F_{2*10} = F_{12*0} = F_{111*} = (P_{210}, P_{120}, P_{111})$.

$F_{2*01} = F_{11*1} = F_{102*} = (P_{201}, P_{111}, P_{102})$.

$F_{1*11} = F_{02*1} = F_{012*} = (P_{111}, P_{021}, P_{012})$.

$F_{1*20} = F_{03*0} = F_{021*} = (P_{120}, P_{030}, P_{021})$.

$F_{1*02} = F_{01*2} = \qquad (P_{102}, P_{012})$.

Theorem 3-4: For any m/n code (m >= 2, n >= 4) the circuit c_{11} @ c_{12} is a TSC checker if and only if

n_i =< m =< n - n_i over all i = 1 to r

for any input partition p(A) of r subsets.

Definition 3-4: B = (B_0, B_1, ..., B_{z-1}) is called an output separation partition on the set of the product functions if and only if neither any error cover set nor any fault cover set is contained entirely in one subset of B.

Theorem 3-5: For any m/n code (m >= 2, n >= 4) the circuit c_{11} @ c_{12} @ c_{13} is a TSC checker if and only if

a) n_i =< m < n-n_i for i = 1 to r for any p(A) and

b) partition B is an output separation partition.

Chapter Four

OUTPUT SEPARATION PARTITION

This section presents a new method of determining the output separation partition B. Different than the method forwarded by Gaitanis and Halatsis [1], this technique simplifies the design of TSC checkers substantially.

Assign to each A_i an arbitrary weight w_i, e.g.,

$$w_i = i \bmod Z \text{ (or } w_i = (i-1) \bmod Z).$$

Here we define the product weight as the algebraic sum

$$W = \left(\sum_{i=1}^{r} w_i * m_i \right) \bmod Z.$$

Let $P_{m_1 m_2 \ldots m_r}$ and $P_{m'_1 m'_2 \ldots m'_r}$ be two different product functions. The weight for $P_{m_1 m_2 \ldots m_r}$ is

$$W_p = \left(\sum_{i=1}^{r} w_i * m_i \right) \bmod Z,$$

and the weight for $P_{m'_1 m'_2 \ldots m'_r}$ is

$$W_{p'} = \left(\sum_{i=1}^{r} w_i * m'_i \right) \bmod Z.$$

Since the product terms are different, there are at least two terms in $P_{m_1 m_2 \ldots m_r}$ which are not equal in $P_{m'_1 m'_2 \ldots m'_r}$ at positions m_j (m'_j) and m_k (m'_k). At these positions $m_j = m'_j + 1$ and $m_k = m'_k - 1$, for $j,k = 1$ to r and no $m_i < 0$.

Here we define the product distance as

$$\begin{aligned} D &= |W_p - W_{p'}| \bmod Z \\ &= \left(\sum_{i=1}^{r} | w_i * m_i - w_i * m'_i | \right) \bmod Z \end{aligned}$$

$$= |w_j m_j + w_k m_k - w_j(m_j-1) - w_k(m_k+1)| \bmod Z$$
$$= |w_j - w_k| \bmod Z$$

An examination of the error cover sets for the product distances of the product terms in the sets reveal the following. By definition 3-1, those product terms with $k_1 + k_2 + \ldots + k_r > m$ and $k_i =< m$ for every $i = 1$ to r belong to the same error cover set. The error

$$E_{k_1 k_2 \ldots k_r} = (P_{(k_1-1)k_2 \ldots k_r},$$
$$P_{k_1(k_2-1)\ldots k_r}, \ldots, P_{k_1 k_2 \ldots (k_r-1)}).$$

The product distance between any two product terms in an error cover set is

$$D_E = |(k_i-1)w_i + k_j w_j - k_i w_i - (k_j-1)w_j| \bmod Z$$
$$= |w_i - w_j| \bmod Z,\ i \neq j \text{ and } i,j = 1 \text{ to } r.$$

Since every error cover set has at least two function elements and $Z >= 2$, there is at least a product distance of $D = 1$ between at least two of its elements. In fact, for $Z = 4$ and $r =< 4$, there is a product distance of at least $D = 1$ between any two elements. The same can be shown for the fault cover set, $D_F >= 1$.

Since in the error cover sets and in the fault cover sets, the minimum distance is $D = 1$ between product functions, we cannot use the same output separation partition B as in [1]. Here we make the following proposition.

Proposition 1: Any partition B on the set of product functions such that $D = 0$ for every pair of functions that belong to the same subset of B, i.e., the product functions have the same weight, is an output separation partition.

From the definition of an output separation partition (Definition 3-4), no subset of B can completely contain an error cover set or a fault cover set. Since all product functions in an error cover set or a fault cover set have a minimum distance of $D = 1$, and elements in each subset of B have $D = 0$, at least one function from at least two output separation partitions will produce an output indicating an error. Set B is an output separation partition.

Example (continued): The output separation partitions for the 3/8 code, with $r = 3$, $Z = 4$ and partitions of size $n_i(3,3,2)$, are designed as follows. Use the weighting scheme $w_i(3,2,1)$. Determine the product weight for each product term.

$P_{300} \Rightarrow W = 3*3 \bmod 4 = 1$

$P_{210} \Rightarrow W = 2*3 + 1*2 \bmod 4 = 0$

$P_{201} \Rightarrow W = 2*3 + 1*1 \bmod 4 = 3$

$P_{120} \Rightarrow W = 1*3 + 2*2 \bmod 4 = 3$

$P_{111} \Rightarrow W = 1*3 + 1*2 + 1*1 \bmod 4 = 2$

$P_{102} \Rightarrow W = 1*3 + 2*1 \bmod 4 = 1$

$P_{030} \Rightarrow W = 3*2 \bmod 4 = 2$

$P_{021} \Rightarrow W = 2*2 + 1*1 \bmod 4 = 1$

$P_{012} \Rightarrow W = 1*2 + 2*1 \bmod 4 = 0$

The output partitions become:

$B_0 = (P_{210}, P_{012})$

$B_1 = (P_{300}, P_{102}, P_{021})$

$B_2 = (P_{111}, P_{030})$

$B_3 = (P_{201}, P_{120})$

No error cover set or fault cover set is entirely contained in any one output separation partition.

For any m/n codes, m >= 2 and n >= 4, the equation

$$\min(n-m,\ m) >= \lceil n/Z \rceil$$

is used to determine the value for Z. As stated, when Z = 3 the value used for the design is Z = 4. For m/2m codes, the value of Z = 2. Using two partitions of size m and realizing m+1 product functions into m-1 AND gates, the output separation partition is of size 2, $B = (B_0, B_1)$. Circuit c_2 is nonexistent.

For m/2m codes, using m partitions of size 2 realizes a three level TSC checker.

At this point it is important to stress the fact that this method is significantly less complex to implement than any previous method. The design and analysis of this design is within the capabilities of researchers without the aid of computers. Many of the designs previously forwarded have required computer programs to determine the output separation partitions and the hardware cost of the designs.

Chapter Five

THE DIRECT SUMMATION METHOD

This chapter presents a new method for TSC checker design. By a recursive procedure, two TSC checkers are directly summed to form a new TSC checker for a larger code without increasing the gate level size over the previous two checkers.

Two TSC checkers, for the codes m_1/n_1 and m_2/n_2, are directly summed to form a third checker for the code $(m_1+m_2)/(n_1+n_2)$. For example, two 3/7 three level TSC checkers are directly summed to form a 6/14 three level checker, then this new checker could be summed with a m_3/n_3 three level checker to form a $(6+m_3)/(14+n_3)$ three level checker.

This design procedure is a direct sum of two TSC checkers. These two TSC checkers must be made using a product array designed in Chapter Three or designed by the methods described by Gaitanis and Halatsis [1], Halatsis, Gaitanis and Sigala [2], Bose and Lin [6] or Reddy [7]. While this method designs the output separation partition for the new TSC checker differently than described earlier, the output separation partitions for the two smaller TSC checkers play an important role.

5.1 Partitioning for Three Level Checkers

Because of the importance of speed in TSC checker designs, we will first design three level checkers. This necessitates using input partitions of size 2 and two output separation partitions in each of the TSC subcheckers. The recursive nature of this procedure does not make this a large restriction. Many three level checkers have been designed using product arrays [2, 6, 7, 9, appendix].

The input partitioning of the first checker is

$$p(A_1) = (A_{11}, A_{12}, \ldots, A_{1r})$$

and the second is

$$p(A_2) = (A_{21}, A_{22}, \ldots, A_{2s})$$

In the new checker the input partitioning is

$$\begin{aligned} p(A) &= (p(A_1), p(A_2)) \\ &= (A_{11}, \ldots, A_{1r}, A_{21}, \ldots, A_{2s}) \\ &= (A_1, A_2, \ldots, A_{r+s}). \end{aligned}$$

The formation of the new product array follows the method outlined in Chapter Three. The product functions are the combinatorial distributions of the $(m_1+m_2)/(n_1+n_2)$ code words over p(A). The product function $P_{m_{11}\ldots m_{1r}m_{21}\ldots m_{2s}}$ is again defined as

$$m_{11} + \ldots + m_{1r} + m_{21} + \ldots + m_{2s} = m_1 + m_2.$$

Let k_1 be the number of variables in $p(A_1)$ that are equal to 1 and let k_2 be the number of variables in $p(A_2)$ that are equal to 1. In this method the values of k_1 and k_2 are important in determining the output separation partitions. The $P_{m_{11}\ldots m_{1r}m_{21}\ldots m_{2s}}$ product function's output separation partition's assignment is determined by the distribution

$$m_{11} + \dots + m_{1r} = k_1$$

and

$$m_{21} + \dots + m_{2s} = k_2.$$

The new checker is formed by partitioning the product functions by one of two methods determined by two cases. These two cases are differentiated by the value of k, i.e., those variables equal to 1, for the inputs to each distinct checker's input partition.

The cases considered are differentiated as

CASE 1 -- code words belonging to subset 2

CASE 2 -- code words belonging to subsets 1 and 3.

Where

Subset 1 -- code words where

$$k_1 < m_1 \text{ and } k_2 > m_2,$$

Subset 2 -- code words where

$$k_1 = m_1 \text{ and } k_2 = m_2,$$

Subset 3 -- code words where

$$k_1 > m_1 \text{ and } k_2 < m_2.$$

An important property of these two cases is that they are mutually exclusive. No code word considered in CASE 1 is considered in CASE 2 and no code word considered in CASE 2 is considered in CASE 1. The sets are easily seen to be mutually exclusive by the strictly greater than or less than relations. Since the subchecker (i.e., subscript) designation as one or two is arbitrary, subsets 1 and 3 are symmetric. In all subsets $k_1 + k_2 = m_1 + m_2$.

CASE 1. $k_1 = m_1$ and $k_2 = m_2$.

In this case each individual checker behaves as the distinct m/n checker it was originally designed as. The output partitions are determined as follows.

The respective output separation partitions associated with the distributions are determined, i.e., the partitions that include $P_{m_{11} \cdots m_{1r}}$ and $P_{m_{21} \cdots m_{2s}}$, and the two compared. The product function's output separation partition is then determined for the corresponding $P_{m_{11} \cdots m_{1r} m_{21} \cdots m_{2s}}$, where

$$m_{11} + \ldots + m_{1r} = k_1$$

and

$$m_{21} + \ldots + m_{2s} = k_2.$$

If the subchecker output separation partitions for $(P_{m_{11} \cdots m_{1r}}, P_{m_{21} \cdots m_{2s}})$ are the same, the product function belongs to B_0, arbitrarily. If the subchecker output separation partitions are different, the product function belong to B_1. For example, if the subchecker output separation partitions are either (B_0, B_0) or (B_1, B_1), then the function belongs to B_0. If the subchecker output separation partitions are either (B_0, B_1) or (B_1, B_0), then the function belongs to B_1.

CASE 2. $k_1 \neq m_1$, $k_2 \neq m_2$ and $k_1 + k_2 = m_1 + m_2$.

This condition would create an error to be detected in both of the subcheckers; therefore, a different method is needed to determine these valid product functions' output separation partitions.

For this case we use the same method presented in Chapter Four. The partition that the product function belongs to is determined by its weight

$$W = iA_i \bmod 2.$$

All product functions whose weight is zero are in B_0 and all product functions whose weight is one are in B_1.

5.2 Error and Fault Detection

Identical to the reasoning presented in Chapter Three, any non code word input where $k_1 + k_2 < m_1 + m_2$, i.e., an error of less than m ones for an m/n code, produces no output from any product function. The TSC checker's output will be the non code word (0,0).

The output separation partitions need to be examined to insure that no error cover set or fault cover set is completely contained in any one partition. This is done by examining both of the cases individually and noting that each case is mutually exclusive over the set of code words.

CASE 1. $k_1 + k_2 > m_1 + m_2$

with either $k_1 = m_1$ and $k_2 > m_2$

or $k_1 > m_1$ and $k_2 = m_2$.

Symmetry enables the examination of only one of the two faults in this case. Suppose the fault where $k_1 = m_1$ and $k_2 > m_2$ exists. Then according to design procedure, the first subchecker will have a function in either B_0 or B_1, say B_0. The second subchecker with $k_2 > m_2$ will produce a term in both B_0 and B_1 because it acts as a distinct m_2/n_2 checker and would indicate an error by producing a non code word output of (1,1). The total checker will then partition the input non code word into (B_0, B_0) and (B_0, B_1), which maps into both B_0 and B_1.

CASE 2. $k_1 + k_2 > m_1 + m_2$

with either $k_1 < m_1$ and $k_2 > m_2$

or $k_1 > m_1$ and $k_2 < m_2$.

Again symmetry enables the examination of only one of these faults. We have defined the product distance as

$$D = |W_p - W_{p'}| \bmod Z,$$

here $Z = 2$. The error cover set and the fault cover set were seen to have at least two functions, with a minimum distance of $D = 1$ between at least two functions. The output separation partition is again designed with $D = 0$ between functions. As described in Chapter Four, at least one function from both output separation partitions will produce an output indicating an error.

5.3 Partitioning for Multilevel Checkers

There are some codes that have no three level TSC checker implementation, i.e., cannot be designed using only two output separation partitions. It is possible to directly sum two checkers having more than two output separation partitions. Also, it is possible to directly sum two checkers with an unequal number of output separation partitions.

The summation method involving checkers with more than two output separation partitions is similar to that involving two output partitions. A generalized method is presented for designing a TSC checker by summing two TSC checkers with output separation partition size Z, Z >= 2. While the design of a checker which reduces to an 1/3 code is not commonly in use, the design is included for completeness.

Consider two TSC checkers. The first for a m_1/n_1 code with input partitions

$$p(A_1) = (A_{11}, \ldots, A_{1r})$$

and output separation partitions

$$B_1 = (a_1, a_2, \ldots, a_{z_1}), z_1 >= 2;$$

and the second for a m_2/n_2 code with input partitions

$$p(A_2) = (A_{21}, \ldots, A_{2s})$$

and output separation partitions

$$B_2 = (b_1, b_2, \ldots, b_{z_2}), z_2 >= 2.$$

The resultant checker designed by the summation method will be for a $(m_1+m_2)/(n_1+n_2)$ code with input partition

$$p(A) = (p(A_1), p(A_2))$$

and with output separation partitions

$$B = (c_1, c_2, \ldots, c_z),$$

where $z = \max(z_1, z_2)$.

The new $(m_1+m_2)/(n_1+n_2)$ code words are distributed over the input partitions of the two checkers. The output separation partitions for the product functions

$P_{m_{11}\ldots m_{1r}m_{21}\ldots m_{2s}}$ are determined by the following two cases corresponding to the distributions

$$m_{11} + \ldots + m_{1r} = k_1$$

and

$$m_{21} + \ldots + m_{2s} = k_2.$$

CASE 1. $k_1 = m_1$ and $k_2 = m_2$.

The individual checker maintains its distinct m/n code TSC checker behavior. Each checker's c_1 subcircuit initially had as an individual output a $1/Z_a$ code, where Z_a is the checker's respective number of output separation partitions. As in the three level design above, the output separation partitions for the product functions $P_{m_{11}\ldots m_{1r}}$ and $P_{m_{21}\ldots m_{2s}}$ are compared. The result of this comparison determines the output separation partition of the corresponding $P_{m_{11}\ldots m_{1r}m_{21}\ldots m_{2s}}$ product function.

The assignment is determined by the following rule. For a_ib_j, the terms are assigned to B such that no ordered pair is duplicated and each a_i or b_j term in any one partition is distinct.

This method is general and many solutions exist. As a starting point for the examples, the following criteria is used for all partitioning except for the three level checker i.e., where $Z_1 = Z_2 = Z = 2$, . For $P_{m_{11}\ldots m_{1r}} \varepsilon a_i$ and $P_{m_{21}\ldots m_{2s}} \varepsilon b_i$, denoted by a_ib_i, then the corresponding term $P_{m_{11}\ldots m_{1r}m_{21}\ldots m_{2s}} \varepsilon c_i$, $a_ib_i \varepsilon c_i$.

For example, the following partition assignments are forwarded for various combinations of TSC checkers.

$Z_1 = Z_2 = Z = 4$

$c_1 = (a_1b_1, a_2b_3, a_3b_4, a_4b_2)$

$c_2 = (a_1b_4, a_2b_2, a_3b_1, a_4b_3)$

$c_3 = (a_1b_2, a_2b_4, a_3b_3, a_4b_1)$

$c_4 = (a_1b_3, a_2b_1, a_3b_2, a_4b_4)$

$Z_1 = Z = 4, Z_2 = 3$

$c_1 = (a_1b_1, a_2b_3, a_3b_2)$

$c_2 = (a_1b_2, a_3b_3, a_4b_1)$

$c_3 = (a_1b_3, a_2b_1, a_4b_2)$

$c_4 = (a_2b_2, a_3b_1, a_4b_3)$

$Z_1 = Z = 4, Z_2 = 2$

$c_1 = (a_1b_1, a_3b_2)$

$c_2 = (a_1b_2, a_4b_1)$

$c_3 = (a_2b_2, a_3b_1)$

$c_4 = (a_2b_1, a_4b_2)$

$Z_1 = Z_2 = Z = 3$

$c_1 = (a_1b_1, a_2b_3, a_3b_2)$

$c_2 = (a_1b_3, a_2b_2, a_3b_1)$

$c_3 = (a_1b_2, a_2b_1, a_3b_3)$

$Z_1 = Z = 3, Z_2 = 2$

$c_1 = (a_1b_1, a_3b_2)$

$c_2 = (a_1b_2, a_2b_1)$

$c_3 = (a_2b_2, a_3b_1)$

$Z_1 = Z_2 = Z = 2$

$c_1 = (a_1b_1, a_2b_2)$

$c_2 = (a_1b_2, a_2b_1)$

CASE 2. $k_1 \neq m_1$, $k_2 \neq m_2$ and $k_1 + k_2 = m_1 + m_2$.

Under these conditions, an error would be detected by both subcheckers and none of the product functions assigned in CASE 1 would produce an output indicating a valid code word.

For this case we again use the method presented in Chapter Four and used above in CASE 2 of the three level checker design. The partitioning is determined by the code word's weight

$$W = iA_i \bmod Z.$$

The value of Z is determined by

$$Z = \max(Z_1, Z_2).$$

5.4 Error and Fault Detection

Showing that CASE 2 fulfills the requirements for a TSC checker has already been addressed. It is necessary to examine CASE 1. The criteria for each i, j, $1 =< i =< Z_1$ and $1 =< j =< Z_2$, to be distinct in each partition is an important one.

Consider the partitioning. If a valid code word was the input to subchecker one, it would produce an output for the associated product function in a_i. Now, suppose a non code word was the input to subchecker two, where $k_2 > m_2$. (In the case of $k_2 < m_2$, no product function would produce an output and the new $(m_1+m_2)/(n_1+n_2)$

checker would have (0,0) as its output.) This causes two product functions to produce an output, say for partitions b_j and b_k. If in one partition both a_ib_j and a_ib_k were included, no error would be detected. Therefore, a_ib_j and a_ib_k must be in different partitions. This implies each subchecker partition a_i and b_j must be distinct in each partition in B.

Chapter Six

DESIGN COST

6.1 Majority Detection Logic

With minimum level realization of the majority detection logic, the number and sizes of the input partitions are critical factors in determining the cost of the TSC checker. The input variables are partitioned such that $m >= n_i$, $i = 1$ to r. It was shown [1] that the gate cost is minimized when

$$n = s * r + t, \quad t < r$$

where $s = \lfloor n/r \rfloor$. For partition p(A) with t subsets where $n_i = s + 1$ and $r - t$ subsets where $n_i = s$, the gate cost $G_{11}(r)$ is

$$G_{11}(r) = (r + t)2^s - 2r.$$

The number of gate input lines to the majority logic has been shown to be

$$I_{11}(r) = t(2^{s-1}+1)(s+3) + (r-t)(2^{s-1}-1)(s+2).$$

Table 1 shows gate counts and input line counts for individual two-level realizations of majority logic and Table 2 shows multilevel cellular design costs.

6.2 Product Array

The gate cost of the product term array is equal to the number of product functions, since each product function forms one AND gate. In [1], formulae were given to

TABLE 1

Majority detection logic minimum level design

n	gates	inputs
2	2	4
3	6	15
4	14	42
5	30	105
6	62	248
7	126	567
8	254	1270
9	510	2805
10	1022	6132
11	2046	13299
12	4094	28658

TABLE 2

Majority detection logic multilevel cellular design

n	gates	inputs
2	2	4
3	6	12
4	11	23
5	18	38
6	26	56
7	35	77
8	45	101
9	57	129
10	70	160
11	84	195
12	99	231

calculate the number of gates and inputs necessary to construct the product array. This paper forwards a different presentation of the calculations needed.

As stated, the product functions are the combinatorial distributions of the code words over the majority logic. To determine the number of product functions, it is necessary to calculate the number of ways m things can be partitioned into r groups of size n_i. The method presented here uses a polynomial expansion.

The generating function takes the form

$$G(y) = 1 + yx + yx^2 + \ldots + yx^{ni}.$$

For an m/n code with r partitions the gate count function is

$$G_{12}(y) = (1+yx+\ldots+yx^{n1})(1+yx+\ldots+yx^{n2})***(1+yx+\ldots+yx^{nr})$$

or, in closed form

$$\Pi\sum \frac{n!}{n_0!n_1!\ldots n_p!} \; 1^{n_0}(yx)^{n_1}(yx^2)^{n_2}\ldots(yx^p)^{n_p}$$

The sum is over all non-negative integers n_0, $n_1,\ldots,n_p$ for which $\sum_{i=0}^{p} n_i = n$ and the product is over all partitions.

The gate count for a specific m/n code is found by evaluating the coefficient using

$$G_{12}(1) = G_{12}(y)\big|_{x^m} - e, \text{ (evaluated at } x^m).$$

The input lines needed in the product array are

$$I_{12}(1) = d/dy \;\; G_{12}(y)\big|_{x^m} - e.$$

Where e is the correction needed if for some i = 1 to r, $n_i = m$. When this condition is met, the product term takes the form $P_{00..m_i..00}$ where $m_i = m$ and no AND gate is necessary. Therefore, e equals the number of partitions of size m.

Using the same polynomial, the number of error cover sets can be found to be

$$E(y) = g(y)\big|_{x^{(m+1)}}.$$

The number of fault cover sets is

$$F(y) = d/dy \;\; G_{12}(y)\big|_{x^m}$$
$$= I_{12}(1)+e$$

The test set size is

$$T(y) = G_{12}(y)\big|_{x^m}$$

Example (continued): For the 3/8 code, there are r = 3 partitions of size $n_i(3,3,2)$. The costs are found by evaluating the polynomial

$$G(y) = (1+yx+yx^2+yx^3)^2 \; (1+yx+yx^2).$$

Evaluated for x^3 gives the cost function as

$$G_{12}(y)\big|_{x^3} = 2y + 6y^2 + y^3 - e$$

where $e = 2$ (P_{300} and P_{030}). The gate count is

$$G_{12}(1) = 9 - 2 = 7.$$

And the input line cost is

$$\begin{aligned} I_{12}(1)\big|_{x^3} &= 2 + 12y + 3y^2 - e \\ &= 2 + 12 + 3 - 2 = 15. \end{aligned}$$

The number of the error cover sets is

$$\begin{aligned} E(1)\big|_{x^4} &= 7y^2 + 3y^3 \\ &= 7 + 3 = 10. \end{aligned}$$

The number of the fault cover sets is

$$F(1) = 17.$$

6.3 Output Reducing Array

The output reducing circuit c_{13} is an array of OR gates, the number of which depends on the output separation partition. The number of gates will be determined in Chapter Four. The number of input lines to c_{13} is equal to the number of gates in c_{12} without the correction factor.

$$\begin{aligned} I_{13}(1)\big|_{x^m} &= G_{12}(y)\big|_{x^m} \\ &= G_{12}(1)+e. \end{aligned}$$

Finally, circuit c_2 for $z = 4$ has a gate cost of 10 and an input line cost of 20.

6.4 Cost Minimization Example

To determine the best partition size, begin with $r = \lceil n/m \rceil$ and calculate $G(r)$ and $I(r)$, until for some r

$$G(r) =< G(r + 1).$$

It has been shown that the most economical TSC checker is designed with the input partitions of approximately the same size.

Example (continued): For a 3/8 code, determining the partition sizes for the most economical design is found by the following method. First, determine the various sizes of partitioning for consideration. For a 3/8 code, we can use $n_i(3,3,2)$, $n_i(3,2,2,1)$ and $n_i(2,2,2,2)$, this is not a complete list. Calculate the cost of circuit c_{11}, the majority logic, using the equations

$G_{11}(r) = (r + t)2^s - 2r$ and

$I_{11}(r) = t(2^{s-1}+1)(s+3) + (r-t)(2^{s-1}-1)(s+2)$.

$n_i(3,3,2)$ -- $s = 2$, $r = 3$ and $t = 2$

$G(r) = (3+2)2^2 - 2(3) = 14$ gates.

$I(r) = 2(2^{2-1}+1)(2+3) + (3-2)(2^1-1)(2+2)$

$= 34$ inputs.

$n_i(3,2,2,1)$ -- $s = 2$, $r = 3$ and $t = 1$

$G(r) = (3+1)2^2 - 2(3) = 10$ gates.

$I(r) = 1(2^1+1)(2+3) + (3-1)(2^1-1)(2+2)$

$= 23$ inputs.

$n_i(2,2,2,2)$ -- $s = 2$, $r = 4$ and $t = 0$

$G(r) = (4)2^2 - 2(4) = 8$ gates.

$I(r) = 0 + 4(2^1-1)(2+2)$

$= 16$ inputs.

Next, calculate the cost of circuit c_{12}, the product array, using and evaluating the following:

$n_i(3,3,2)$ -- $G(y) = (1+yx+yx^2+yx^3)^2(1+yx+yx^2)$

$G(y)|_{x^3} = 2y + 6y^2 + y^3 - e \; (e = 2)$

$G(1) = 2 + 6 + 1 - 2 = 7$ gates,

and

$$I(y) = 2 + 12y + 3y^2 - e$$

$$I(1) = 2 + 12 + 3 - 2 = 15 \text{ inputs.}$$

$$n_i(3,2,2,1) \text{ -- } G(y) = (1+yx+yx^2+yx^3)(1+yx+yx^2)^2(1+yx)$$

$$G(y)\,\Big|_{x^3} = y + 9y^2 + 4y^3 - e \ (e = 1)$$

$$G(1) = 1 + 9 + 4 - 1 = 13 \text{ gates,}$$

and

$$I(y) = 1 + 18y + 12y^2 - e$$

$$I(1) = 1 + 18 + 12 - 1 = 30 \text{ inputs.}$$

$$n_i(2,2,2,2) \text{ -- } G(y) = (1+yx+yx^2)^4$$

$$G(y)\,\Big|_{x^3} = 12y^2 + 4y^3 - e \ (e = 0)$$

$$G(1) = 12 + 4 = 16 \text{ gates,}$$

and

$$I(y) = 24y + 12y^2 - e$$

$$I(1) = 24 + 12 = 36 \text{ inputs.}$$

Since circuits c_{13} (4 gates and G(y) + e inputs) and c_2 (10 gates and 20 inputs) are used in all three test partitions, we can select the partitioning of $n_i(3,3,2)$ as our most economical design, with a total of 35 gates and 78 inputs.

Chapter Seven

DIRECT SUMMATION EXAMPLE

Using the direct summation method for construction of TSC checkers is a straightforward procedure that is demonstrated here. First, a three level 3/7 code TSC checker and a three level 2/4 code TSC checker will be directly summed to form a three level 5/11 code TSC checker. Then this new checker can be used to make other checkers.

Consider a TSC checker for the code 3/7 with input partitions of size $n_i(2,2,2,1)$ and output separation partitions of

$$A_1 = (2100,\ 1020,\ 0210,\ 2001,\ 0201,\ 1110,\ 0021)$$
$$B_1 = (1200,\ 2010,\ 0120,\ 1101,\ 0111,\ 1011).$$

Consider a 2/4 code TSC checker with input partition size $n_i(2,2)$ and output separation partitions of

$$A_2 = (20,\ 02)$$
$$B_2 = (11).$$

These two checkers will be summed to form the three level 5/11 code TSC checker. This checker will have input partitions of size $n_i(2,2,2,1,2,2)$, where the first four partitions correspond to the 3/7 checker and the last two partitions correspond to the 2/4 checker.

The generating polynomial for the 5/11 code using these input partitions is

$$G(y) = (1 + yx + yx^2)^5(1 + yx).$$

Expanding and evaluating at x^5, for a m = 5 code, the size of the product array becomes

$$G(y)|_{x^5} = 40y^3 + 50y^4 + 6y^5$$

so

$$G(1)|_{x^5} = 40 + 50 + 6 = 96 \text{ gates.}$$

The number of input lines in the array is

$$I(y)|_{x^5} = 120y^2 + 200y^3 + 30y^4$$

and

$$I(1)|_{x^5} = 120 + 200 + 30 = 350 \text{ input lines.}$$

Therefore, the total cost of the checker will include 10 gates with 20 inputs in the majority logic and 2 OR gates with 96 inputs to form the output separation partitions. The cost for this design is then

108 gates, 466 input lines and a test set of size 96.

The procedure divides the code words into two cases:

CASE 1, where $k_1 = m_1$ and $k_2 = m_2$, and

CASE 2, where $k_1 = m_1$ and $k_2 = m_2$.

CASE 1 code words.

These are the code words that are formed by the concatenation of an element from set A_1 or B_1 with an element from set A_2 or B_2. For example, from A_1 take (2100) and from A_2 take (20) to form (210020). The number of code words included in this set are (the total number of elements in sets A_1 plus B_1) times (the total number of elements in sets A_2 plus B_2). Here, 13 * 3 = 39 code words making 39 product functions or 39 AND gates in the product array from CASE 1 code words.

The mapping for this case will be

$$A_3 = A_1 @ A_2 \text{ and } B_1 @ B_2$$

$$B_3 = A_1 @ B_2 \text{ and } A_2 @ B_1.$$

That is, elements from partitions concatenated with elements from partitions with the same letter designation go into A_3, if not, they go into B_3. The 39 code words are then partitioned as shown in Table 3.

CASE 2 code words.

The remaining code words are mapped in accordance with the rule

$$A_3 = 0 = (\sum_{i=1}^{6} in_i) \bmod 2$$

$$B_3 = 1 = (\sum_{i=1}^{6} in_i) \bmod 2.$$

For example, the code word 221000 is mapped

[6*2 + 5*2 + 4*1 + 3*0 + 2*0 + 1*0 = 26]mod 2 = 0 => A_3.

The code word 220100 is mapped

[6*2 + 5*2 + 4*0 + 3*1 + 2*0 + 1*0 = 25]mod 2 = 1 => B_3.

For the 5/7 code there are 96 - 39 = 57 CASE 2 code words. Twenty eight map into A_3 and twenty nine map into B_3. See Table 3.

The code words from the two cases listed in Table 3 become the product functions. These 96 product functions make the AND gates in the product array. The outputs of these AND gates are then the inputs of two OR gates partitioned as listed. This partitioning makes the completed three level 5/11 code TSC checker.

A comparison of this three level 5/11 code checker with the three level 5/11 code checker designed in [2] shows an equal number of gates used and the same test set size. However, where their design uses 517 inputs, this uses 466--a 9.9% improvement over theirs. The most dramatic difference is the simplicity of design in this method over that in [2].

This same procedure can now be used again with the 5/11 code checker partitioned in this manner with another three level checker, say the 3/7 code checker to form a three level 8/18 code TSC checker or the 2/4 code checker to form a three level 7/15 code checker.

An interesting result of this method uses a 1/2 code TSC checker with input partitions of size $n_i(1,1)$ and output separation partitions of $A = (1,0)$ and $B = (0,1)$. This checker, having no gates with the input lines becoming the output lines, can be directly summed with other TSC checkers using the above procedure.

Table 3

PARTITIONING FOR 5/11 CODE

CASE 1 CODE WORDS

A_3 = (210020, 210002, 201011, 200120, 200102, 120011, 111020, 111002, 110111, 102020, 102002, 101111, 021020, 021002, 020120, 020102, 012011, 011111, 002120, 002102)

B_3 = (210011, 201020, 201002, 200111, 120020, 120002, 111011, 110120, 110102, 102011, 101120, 101102, 021011, 020111, 012020, 012002, 011120, 011102, 002111).

CASE 2 CODE WORDS

A_3 = (221000, 220010, 211100, 211001, 210110, 202010, 201101, 200012, 122000, 121010, 120101, 112100, 112001, 111110, 110021, 102101, 101012, 100121, 100022, 022010, 021101, 020012, 012110, 011021, 010112, 002012, 001121, 001022)

B_3 = (220100, 220001, 212000, 211010, 210101, 202100, 202001, 201110, 200021, 121100, 121001, 120110, 112010, 111101, 110012, 102110, 101021, 100112, 022100, 022001, 021110, 020021, 012101, 011012, 010121, 010022, 002021, 001112, 000122)

Chapter Eight
SUMMARY AND RECOMMENDATIONS FOR FUTURE RESEARCH

This thesis has forwarded two new methods of designing TSC checkers. The first method designs basic checkers with a simplification in both analysis and construction. This new technique enhances capabilities of insuring that the most cost effective checkers are designed.

The second method, the direct summation method, designs TSC checkers from a library of other TSC checkers. This most important capability, never before presented, gives designers prior knowledge of the existence of checkers meeting design criteria and a method for constructing those checkers.

The direct summation method gives designers the capabilities of constructing TSC checkers for a wide range of codes, i.e., m/(2m+2), etc., that previously had no realization in three level checkers. This method has great potential in PLA architectures.

Future research should be aimed at the construction of basic three level checkers. This translates to identifying the output separation partitions for m/n codes.

Further work can be done in the investigation of the structure of the code partitions as suggested in the appendix.

REFERENCES

[1] Nicolas Gaitanis and Constantine Halatsis, "A New Design Method for m-out-of-n TSC Checkers," IEEE Trans. Comput., Vol. C-32, No. 3, March 1983, pp. 273-283.

[2] C. Halatsis, N. Gaitanis, and M. Sigala, "Fast and Efficient Totally Self Checking Checkers for m-out-of-(2m+-) Codes," IEEE Trans. Comput., Vol. C-32, No. 5, May 1983, pp. 507-511.

[3] D. A. Anderson and G. Metze, "Design of Totally Self Checking Check Circuits for m-out-of-n Codes," IEEE Trans. Comput., Vol. C-22, No. 3, March 1973, pp. 263-269.

[4] Rene David, "A Totally Self Checking 1-out-of-3 Checker," IEEE Trans. Comput., Vol. C-27, No. 6, June 1978, pp. 570-572.

[5] Peter Golan, "Design of Totally Self Checking Checker for 1-out-of-3 Code," IEEE Trans. Comput., Vol. C-33, No. 3, March 1984, p. 285.

[6] Bella Bose and Der Jei Lin, "PLA Implementation of k-out-of-n Code TSC Checker," IEEE Trans. Comput., Vol. C-33, No. 6, June 1984, pp. 538-588.

[7] S. M. Reddy, "A Note on Self Checking Checkers," IEEE Trans. Comput., Vol. C-23, October 1974, pp. 1100-1102.

[8] M. A. Marouf and A. D. Friedman, "Efficient Design of Self Checking Checker for any m-out-of-n Code," IEEE Trans. Comput., Vol. C-27, No. 6, June 1978, pp. 482-490.

[9] T. Nanya and Y. Tohma, "A 3-level Realization of Totally Self Checking Checkers for M-out-of-N Codes," Proc. 13th Conf. on Fault-Tolerant Computing, Milan, Italy, June 1983, pp. 173-176.

APPENDIX

A GRAPHICAL METHOD FOR DETERMINING M/N CODE OUTPUT SEPARATION PARTITIONS

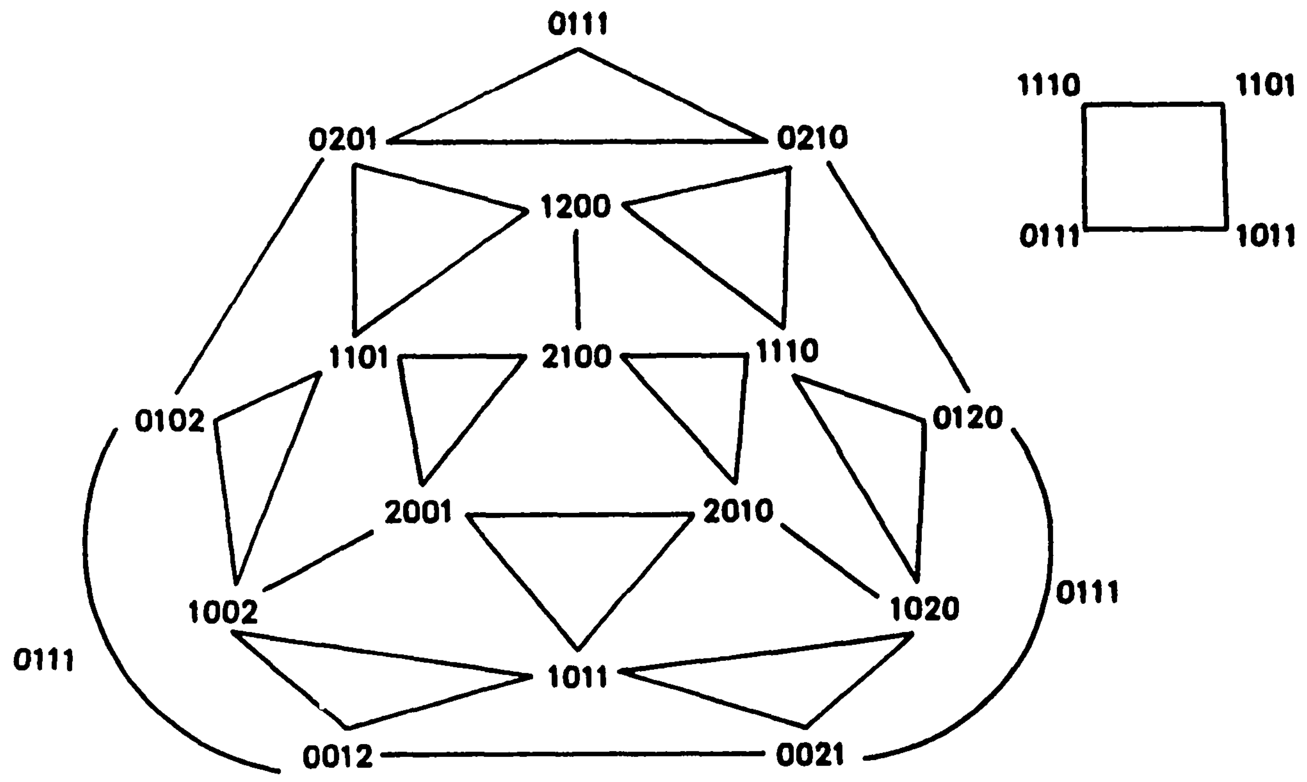
In researching and developing the methods presented here, much work was done outside the main thrust of this thesis. Some of this work was trying to derive output partitions for m/n codes, specifically trying to derive output separation partitions of size Z = 2. Methods for obtaining these output separation partitions by formulae have been presented in many papers [1],[2],[6],[7] and [9].

During the research for this thesis, a graphical method was developed. This method has not been previously forwarded and provides some interesting results. It is presented in an appendix because it does not fall in the focus of the main thesis. The method is presented here for output separation partitions of size Z = 2. The method can be extended for larger values of Z.

STEP 1. Determine the product functions for a m/n code TSC checker.

STEP 2. Determine the error cover sets and the fault cover sets.

STEP 3. Using the product functions as nodes, connect all nodes corresponding to the error cover sets. See Figure A-1.



GRAPHICAL REPRESENTATION OF 3/7
ERROR COVER SETS
FIGURE A-1

STEP 4. Members of an error cover set, which have only two elements, are connected by a line. All such pairs must have one member in each output separation partition.

STEP 5. Each member of an error cover set with more than two elements are connected by a polygon. At least one product function, from the inclusive nodes of each polygon, must be in each output separation partition.

(Note: It may not always be possible to connect nodes neatly. As in Figure A-1, the code word 0111 belongs to four polygons. Any method of connection is allowable as long as proper partitioning can be completed. The use of multiple graphs (see Figure A-1) and colors may be helpful.)

STEP 6. The final output separation partitions are then compared with the fault cover sets to determine if no fault cover set is contained completely in any one partition.

While this method does nothing more than present the error cover sets graphically, it presents the relationships between these sets clearly.

From Figure A-1, for a 3/7 code or the dual 4/7 code, two different partitionings can be found. See Tables A-1 and A-2. Examining these partitions reveals that each partition can be divided into two segments -- a core and a supplement. Both partitionings consist of one core in each partition with the supplements interchanged.

Examination of the results obtained in the direct summation method also shows this core/supplement structure. The CASE 1 code words can be related to the core, while

CASE 2 code words can be related to the supplement. With the arbitrary assignment of CASE 2 code words, two different output separation partitions can be derived.

No other method of deriving these partitions forward more than one solution. The characteristic of a core/supplement structure also has not been presented.

Also, derived from this method was a size Z = 4 partitioning for the 2/9 code. See Table A-3. This has never been done previously and opens new possibilities in designing more cost effective checkers.

Table A-1

PARTITIONING FOR THE 3/7 CODE

	A	B
Core	2100	1200
	1020	2010
	0210	0120
Supplement	2001	1101
	0201	0111
	1110	1011
	0021	

or

	A	B
Core	2100	1200
	1020	2010
	0210	0120
Supplement	1101	2001
	0111	0201
	1011	1110
		0021

Table A-2

PARTITIONING FOR THE 4/7 CODE

	A	B
Core	1021	2011
	2101	0121
	0211	1201
Supplement	2200	2110
	2020	1210
	0220	1120
	1111	

Table A-3

PARTITIONING FOR THE 2/9 CODE

A	B
20000	10010
02000	01100
00200	00011
00020	01010

C	D
10100	11000
00101	10001
01001	00110