On concurrent error detection schemes for a parameter filter IC

Nam Hyung Kim

March 1991

VLSI Design Laboratory
Department of Electrical Engineering
McGill University
3480 University street
Montreal, P.Q.
Canada H3A 2A7

Table of Contents

List of Figures

List of Tables

Acknowledgements

I am very grateful to Dr Vinod Kumar Agarwal not only for his academic guidance but also for his encouragement and inspiration throughout the course of my studies

I want to give many thanks to each member of the VLSI lab faculty and staff and to graduate students who have been always friendly and willing to help and share valuable ideas

I wish to express my appreciation to Ajai Jain for proof reading of my thesis Eric Masson for doing the French translation of the abstract I am also very thankful to Centre de Recherche Informatique de Montreal (CRIM) for their financial support

Last, but not least, I deeply thank my parents who have sacrificed much to give me this wonderful opportunity of education

Abstract

Electronic Support Measures (ESM)-Parameter Filter IC was specifically designed to process radar signal data This is one of the essential ESM functions which must be done in real time Considering the complexity of the Filter IC and its crucial service in Electronic Warfare (EW), fault-tolerance should be an important element of the circuit However, in the initial design, fault-tolerance was not included Therefore, fault-tolerance schemes especially suitable for ESM Filter IC are presented in this thesis

A concurrent error detection technique is presented The method utilizes a code, referred to as MATCH code that is inherent in the function of ESM Filter IC Effectiveness of the approach is measured by its error coverage In the analysis of the coverage, error detection capability of MATCH code and probability of fault detection are incorporated The hardware overhead required to implement a lookup-table which stores MATCH code is also given for various CAM (content-addressable memory) cells

As an alternative approach, recomputation with rotated comparands (RERC) is suggested for detection of all single errors The possibility of multiple error detection and single error correction by the same method is also discussed

Résumé

Le circuit intégré dénommé Electronic Support Measures Parameter Filter IC à été conçu dans le cadre du projet Electronic Support Measures (ESM) spécifiquement pour le traitement des signaux de radars Le traitement est une fonction qui doit être faite concurremment et qui est essentielle au EMS À cause de la complexite du circuit et de son importance dans l'Electronic Warfare (EW guerre électronique) le circuit doit être capable de tolérer certains défaults (de fabrication, d'opération, etc) Cependant dans la conception initiale du ESM parameter Filter IC, la protection contre les défaults etait absente Cette thèse présente des méthodes de protection qui sont particulières au ESM parameter Filter IC

Une technique de détection qui s'applique concurremment est présentée Cette méthode utilise le code MATCH qui est inhérent à la fonction du parameter Filter IC L'efficacité de cette approche est mesurée par le nombre d'erreurs qu'elle détecte Dans l'analyse de détection des erreurs, les capacités du code MATCH et les probabilités de détection sont prises en considération Le coût additionnel en matériaux qui est requis pour réaliser un tableau de cellules de type CAM (Content Addressable Memory mémoire associative) est examiné

Alternativement, une autre approche est suggérée celle du RERC (Recomputation with Rotated Comparands - Recalcul avec rotation de l'argument) qui détecte toutes les erreurs singulières La possibilité de détection d'erreurs multiples et la correction d'erreurs singulières par le RERC est aussi discutée

Chapter 1 Introduction

1.1 Fault-tolerant designs

The main purpose of fault-tolerant designs is to ensure the dependability of systems even under the presence of faults. *Dependability* of a system is measured by reliability and availability [1]-[3]. *Reliability* of a system, $R(t)$ as a function of time, is the conditional probability that the system will operate without any failures during the time interval $[0, t]$ where $t = 0$ is a reference initial instant with $R(0) = 1$. Reliability is used to describe systems in which they are serving critical functions and can not be interrupted even for the duration of a repair without catastrophic consequences. For example, real-time systems such as those used in military, aerospace and nuclear power plant control systems require very high reliability since occurrence of even one error during their operation time can be fatal. *Availability* of a system, $A(t)$ is the probability that the system will be operational at time t. Availability is typically used to describe systems in which their service can be delayed or absent for short periods without serious loss.

The reliability of general purpose systems has also become of more significance because system complexity, repair cost as well as our dependence on such systems are increasing. High system reliability has been achieved through two basic strategies: fault avoidance (fault intolerance) and fault tolerance. The principle of fault avoidance is based

on conservative design practices such as the use of reliable components for the system and to use extensive testing to ensure that the components are fault-free to lessen the possibility of system failures

With the advent of VLSI technology, it has become possible to integrate many number of processing elements onto a single chip, but then the risk of having any faults on the chip has also increased In fact even a single failure may make the entire computing effort null unless some degree of fault tolerance is incorporated in the process of computation Hence, fault avoidance alone may not be sufficient to meet system reliability goal because faults (i.e , transient faults) will eventually occur and it will result in system failure even with the most careful fault avoidance

In fault-tolerant designs, we assume that faults are expected to occur in system operation, but the effect of failures is to be automatically negated by the use of redundancy Accordingly, a fault-tolerant system is the one which is able to operate continuously without failure in spite of faults having occurred or occurring.

The fault tolerance of a system is achieved by using redundant resources The redundancy in the fault-tolerant system may be one of the two: extra time or extra components. One form of time redundancy is having extra executions of the same computation to produce multiple results Then some operations, like comparisons on those results can be used to decide subsequent actions depending on the presence (or absence) of errors Component redundancy involves the use of extra gates, memory cells, bus lines and functional modules, etc to provide extra informations to nullify the effect of failures

1.2 Fault-tolerant techniques

Different degrees of fault tolerance can be achieved either by detection or masking of errors, recovery from errors, isolation of faulty units, reconfiguration of the system One

typical way of error masking is done by replicating the modules to provide enough redundant information Then errors are masked by means of a majority vote on the outputs of the modules This type of error masking techniques includes triple modular redundancy (TMR) [5] which was basically developed from the work done by von Neumann [6], and N-tuple modular redundancy (NMR) [7] This approach is referred to as a system-level redundancy For error masking techniques in gate-level redundancy, there are quadded logic [8] and radial logic [9] These approaches do not use explicit voting, and the capability of error masking is inherent in its logic structure itself. Pierce [10] extended these schemes into a general theory called interwoven logic.

Concurrent error detection techniques (CED), which are designed to detect errors but not to mask them with normal operation, include totally self-checking circuits (special case of self-checking circuits) (TSC) [13], alternating logic [14], recomputation with shifted operands (RESO) [16] Algorithm-based fault tolerance schemes have also been proposed for parallel matrix operations [17]-[20]

1.3 Thesis outline

Our main goal in this work is to devise an effective concurrent error detection scheme which is particularly suitable for Electronic Support Measures (ESM)-Parameter Filter IC, and to prove the effectiveness of such scheme. The technique must also meet the time constraint of the processor with reasonably low hardware overhead.

In chapter 2 we review three relevant concurrent error detection techniques Self-checking circuit scheme [11]-[13] is reviewed to give basic concept of error detection code, which will help understand the proposed error detection scheme based on a code A rather recently developed technique, algorithm-based fault tolerance [17]-[20] describes a technique specially oriented for matrix operations by array of processors. Although the

method is not applicable to our problem, it shows a very efficient way of taking advantages of multiple processors to obtain fault tolerance with checksum code. The fault model used in this scheme also fits our case. At the end time redundancy technique recomputation with shifted operand (RESO) [16] is discussed. This method is applicable to fault tolerance of ESM Filter.

In chapter 3, ESM System function in a global view of Electronic Warfare (EW) is introduced to give some background knowledge on ESM-Parameter Filter IC. Then the general and functional description of ESM-Parameter Filter IC are given in details.

In chapter 4 we consider various fault and error models for VLSI digital circuits. Inadequacy of the classical stuck-at fault model is discussed. Then the fault and error model for our concurrent error detection techniques are defined.

In chapter 5 we propose a concurrent error detection scheme based on MATCH code. The output of ESM Filter is treated as a form of the code, called as MATCH code. It is shown that the duplication of processors is necessary for the detection of all single errors under a reasonable assumption. Then, a lookup table based on content addressable memory (CAM) architecture is suggested to implement the proposed scheme which can detect significant portion of all single errors. The hardware overhead ratios are given for various CAM cells. The last section of this chapter is devoted to measure the error coverage of our error detection technique. Probabilistic analyses are carried out to obtain the probability of fault detection and the error detection capability of MATCH code.

Chapter 6 describes another error detection method, mainly derived from recomputation with shifted operands (RESO) using time redundancy. The possibility of multiple error detection and single error correction by the same method is discussed in this chapter.

The conclusion summarizes the results obtained in this work with comments regarding the limitation on our coverage measure and future prospects of fault tolerant designs.

Chapter 2 Concurrent Error Detection Techniques

In this chapter, we review some of the relevant concurrent error detection techniques such as self-checking, algorithm-based fault tolerance, and recomputation with shifted operands (RESO) Their concepts, capabilities and applications are discussed with some examples

2.1 Self-checking circuits

Self-checking circuits are used to detect errors concurrently with normal operation [11]-[13] The gate level single-stuck at fault model is assumed These circuits operate on encoded inputs to produce encoded outputs Self-checking checkers are used to monitor the outputs to indicate an error when a non-code word is detected as shown in figure 2 1

For circuits to be *totally self-checking circuits*, the circuits should be both **self-testing** and **fault-secure** as defined by Anderson [13]

A circuit is **self-testing** if, for every fault from a prescribed set of faults F, the circuit produces a non-code word for at least one valid input from a code space input C. In other words, all faults in F can be detected by some input in C.

A circuit whose outputs define a code C' is **fault secure** for a prescribed set of faults F with an input code C if, for any fault in F and any input in C, the fault is either

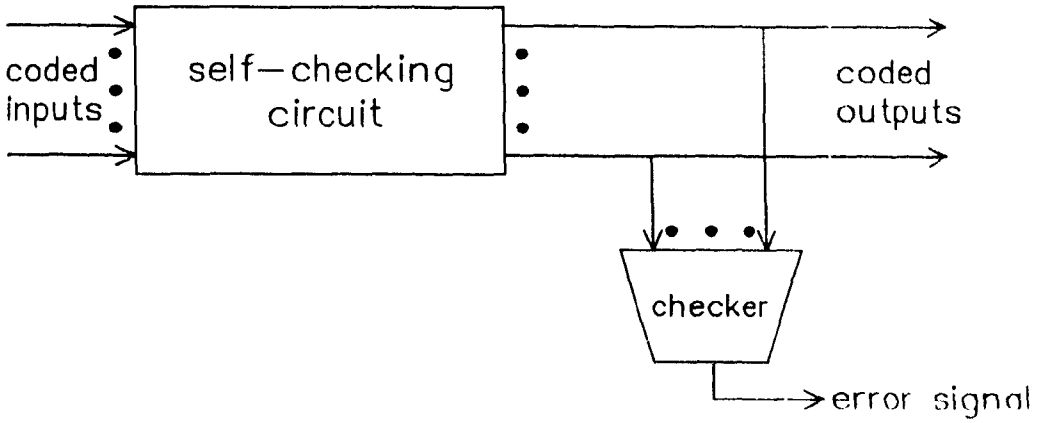


Figure 2.1 Self-checking circuit and checker

not detected (i e , the output is a correct word) or the output is not in C' (i e , the output is a non-code word). For a given input X, a fault may or may not result in error If the fault causes an error, the output may change into non-code word (a detectable error) or an incorrect code word (an undetectable error) if the circuit is not fault secure The fault-secure property of self-checking circuits removes the possibility in which undetectable errors can occur

A simple example of a totally self-checking circuit is a buffer for n bit parity checked operands shown in figure 2 2

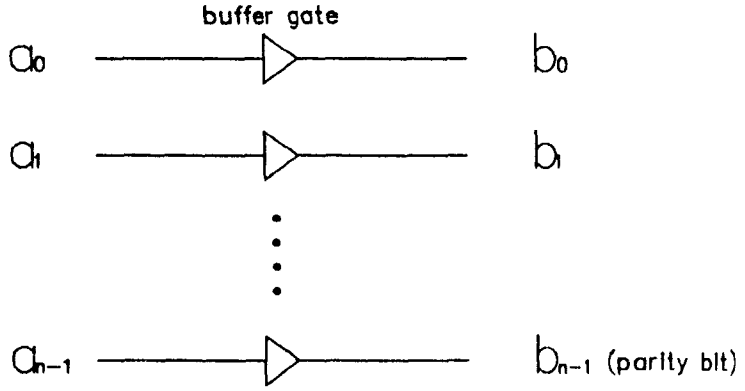


Figure 2.2 Totally self-checking buffer

There are n identical buffer gates, one for each output bit The output vectors and

input vectors both are equal to the set of all even-parity n bit vectors Then the circuit is fault-secure for all single faults, because a single fault causes either no error, or one bit change in the output vector producing an odd-parity vector - a detectable error

If a circuit is fault-secure for only a subset of the valid inputs ($c \subset C$) and self-testing for a valid input set C and a fault set F, then it is called *partially self-checking*. If a circuit is self-testing for a valid input set C and a fault set F, but not fault-secure for any faults in F then it is called *self-testing-only* The term, "self-checking" refers to all of these desirable properties (totally self-checking, partially self-checking, self-testing-only) in general.

Both the self-testing and fault-secure properties only specify the behavior of the circuit for correct code word inputs But in the design of the self-checking checker, non-code words are the inputs to the checker when any error occur in the self-checking circuit, and it should be mapped to non-code word outputs of the checker, which signals the presence of errors to the user. For the absence of errors the self-checking checker should map code word inputs into code word outputs. The circuit with the described behavior is called code disjoint, which is required characteristic for the self-checking checker in addition to the self-checking properties

In summary, a totally self-checking circuit produces always the correct output code word as long as no errors occur If errors occur, no erroneous results go undetected A partially self-checking circuit has the same property except for certain set of faults that can lead to undetectable errors All self-checking circuits at least have the self-testing property, ensuring that all faults in the set of fault F will be detected if every input vector is applied at least once in a reasonable period of time.

2.2 Algorithm-based fault tolerance

Algorithm-based fault tolerance techniques are not as generally applicable as some of the classical techniques such as TMR [5] However, fault tolerance can be achieved with very low overhead by this method for basic matrix operations such as matrix multiplication, the QR decomposition, and the LU decomposition on multiple processor systems

It uses module level (or functional level) fault model which assumes that a small area of the chip is affected by physical failure and the nature of the failure is not precisely known At some higher functional level, (a processor, a bit-slice of an ALU, or any computational unit) the effect of failures will appear as changes in logical values

In algorithm-based fault tolerance schemes, the input data used by the operation algorithm is encoded in some form of error detecting code (error correcting as well in some cases), and the original algorithm is redesigned to operate on the encoded input data Then the encoded output data can be checked at the end of the computation for errors

It is important to distribute appropriately the computation steps on multiple processing units, so that an error in the processing element will affect as small portion of the data as possible The main goal of algorithm-based fault tolerance is to design efficient coding schemes to provide error detection, and possibly error correction as well, with small overhead.

Checksum schemes were proposed to provide single-error detection for basic matrix operations [17], [18]. For example the checksum matrix scheme is used to detect and correct errors for the multiplication of two 3-by-3 matrices A and B as illustrated in figure 2.3

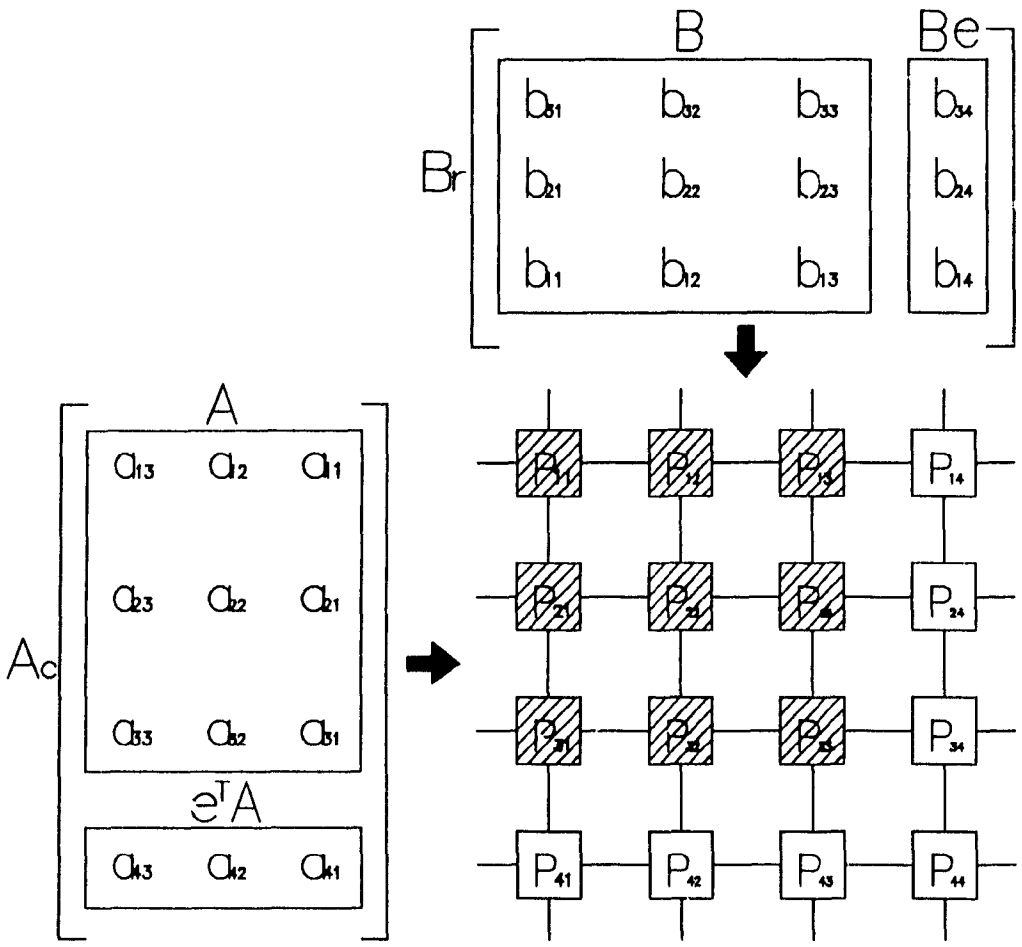


Figure 2.3 Multiplication of two checksum matrices in mesh-connected processor array

It shows a mesh-connected processor array with row/column broadcast capability,* and the streams of the input data for the matrix multiplication $e^T A$, an extra set of elements added right below the matrix A is called *column summation vector*, where e^T is a 1-by-3 vector [1 1 1] The elements of the column summation vector are generated as

$$a_{4,j} = \sum_{i=1}^{3} a_{i,j} \; for \; 1 \leq j \leq 3$$

The entire matrix including the matrix A and the column summation vector $e^T A$ is called *column checksum matrix* A_c of the matrix A.

* The array consists of a common data bus for each processors, and data can be broadcast to all processors at the same time

Be, an extra elements added right next to the matrix B is called *row summation vector*, where e is a 3-by-1 unity vector. The elements of the row summation vector are generated as

$$b_{i,4} = \sum_{j=1}^{3} b_{i,j} \quad for\ 1 \le i \le 3$$

The entire matrix including the matrix B and the row summation vector Be is called *row checksum matrix* B_r of the matrix B

The elements $a_{i,j}$ are broadcast from the left boundary of the array to processors in the ith row at time j. The elements $b_{j,k}$ are also broadcast to processors in the kth column at time j. At time j, the processor $P_{i,k}$, located on the intersection of the ith row and kth column of the array, performs the product of $a_{i,j}$ and $b_{j,k}$ and accumulates the result in a register. Thus, after 3 time steps, each processor computes an element of the resultant 4-by-4 matrix C_f

From the theorem 4.1 in [17] the final solution matrix $A \times B = C$ is in the first 3 rows and 3 columns of the matrix C_f, stored in the hatched portion of the processors. And 4th row/column of the matrix C_f should be the column/row summation vectors of the matrix C_f. Therefore it should also satisfy the following relations

$$c_{4,j} = \sum_{i=1}^{3} c_{i,j} \quad for\ 1 \le j \le 4$$

$$c_{i,4} = \sum_{j=1}^{3} c_{i,j} \quad for\ 1 \le i \le 4$$

The sum of the solution matrix elements in each row and column is computed. Then each sum is compared with the corresponding checksum in the summation vector. Assuming that only one faulty processor exits, no more than one row and column will have an inconsistency which is the indication of an error. The intersection of such inconsistent row and column locates an error. The erroneous element can be corrected by adding the difference of the

computed sum of the row or column elements and the checksum to the erroneous elements of the solution matrix If the checksum is the incorrect one, then we can replace the checksum by the computed sum of the solution matrix elements in the summation vector

The checksum scheme can only correct errors in matrix multiplication, it can detect, but not correct errors in LU decomposition, matrix inversion, etc In order to solve such problems with the checksum scheme, Jou and Abraham [18] developed a scheme, called the weighted checksum code (WCC) The checksum code is a subset of WCC Later Anfinson and Luk unified ideas from linear algebra and coding theory to provide a theoretical basis for weighted checksums [19].

2.3 Recomputation with shifted operands

RESO is a concurrent error detection scheme based on time redundancy, and can be used for arithmetic and logic units [16] It uses functional level fault model

Initially the computation for some function of the bit slice of an ALU, $f(x)$ is done with unshifted operands, and the result is stored in a register In the recomputation step, the operands are shifted left by k bits and then input to the same unit which performs the function $f(x)$ The result is right shifted and finally compared with the previous result stored in the register Such operations as shift left and right can be denoted respectively as $c(x)$ and $c^{-1}(x)$ Thus the equivalent notation to the recomputation step is $c^{-1}(f(c(x)))$, which equals to $f(x)$ Hence the mismatch of the comparison indicates an error in computation as shown in figure 2 4

This method using $k = 1$ can detect all errors in an ALU for all bitwise operations AND OR, NOT, XOR, NOR when the failure is confined to a single bit-slice

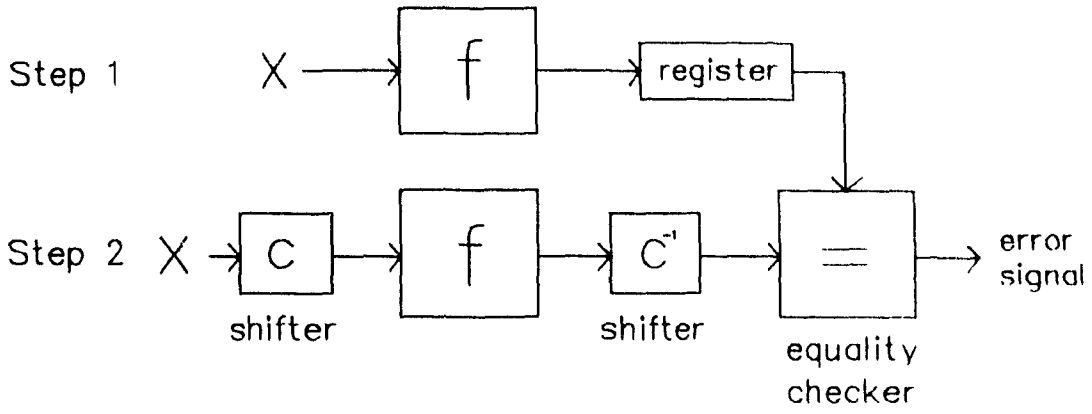


Figure 2.4 Error detection with time redundancy

The hardware for the RESO-k* to detect errors in an ALU for n bit operations includes two shifters, a register, an $(n + k)$-bit ALU, and a totally self checking equality checker. The totally self-checking equality checker can be implemented based on 1 out of 2 code checkers [13] We may also use a proper parity code to detect errors in the shifter

The execution time is doubled in RESO-1 but it is only a small portion of the entire instruction cycle if the ALU is viewed in the global context involving the entire computer

The method of error detection using RESO may can be modified in several different ways One of such cases is recomputing with rotated operands The rotation of operands is the same as a circular shift, and so some of the results derived for RESO are also valid with rotation. It will be shown that rotation can be substituted for logical shifts as we apply principles of RESO to ESM-Parameter Filter IC in chapter 6 One advantage of rotation over shifts is that no additional bit-slices are required

To conclude, the fault model is the key deciding factor in selection of an effective error detection scheme in various levels At the processing element level, one can incorporate error detection capability into the internal architecture of the processing elements by

* RESO-k is the name of the error detection scheme achieved by recomputing with k bit shifted operands

employing techniques such as self-checking based on gate level stuck at fault model On the other hand, algorithm-based error detection technique or RESO may be more efficient approach to achieve concurrent error detection at the network level

Chapter 3 Electronic Support Measures System

3.1 Introduction

Electronic Support Measures (ESM) refers to the branch of electronic warfare (EW) which also includes electronic countermeasures (ECM) and electronic counter countermeasures (ECCM) Electronic support measures (ESM) is mainly concerned with the interception, location, and identification of various radar-emitters for the purpose of immediate threat recognition and the tactical employment of forces such as ECM equipment The information collected by an ESM system are typically fed into a command central where the various interceptions are analyzed and decisions are made to deploy ECM techniques ECM is used to prevent or reduce the enemy's effective use of electronic systems by jamming or disrupting In order to encounter such hostile environments as jamming, electronics equipments (i e , surveillance radar) can incorporate some of the ECCM techniques in their design, such as spread spectrum, error control coding [21], [22]

The ESM function is reserved for real-time or near real time reaction In contrast Electronic Intelligence (ELINT), which has the similar functions as ESM, generally involves subsequent or non-real time analysis of the intercepted data for intelligence collection purposes.

Some examples of ESM system are a radar warning receiver (RWR) and reconnaissance/surveillance receiver (RSR) systems RWR intercepts radar signals and analizes their relative threat in real-time, used by aircraft or ships for self protection RSR intercepts, collects, analizes, and locates radar signals in near-real time to update the local electronic order of battle (EOB)* for targeting or warning [22]

An ESM receiver intercepts the radiation of emitters. The situation may be that interfering signals are present, or several signals of interest are present at the same time For that reason the intercepted signals are the train of interleaved radar pulses with small number of randomly occurring overlaps [23]

The ESM receiver then quantizes each received radar pulse into digital state vector on the basis of a few selected state variables The state variables are the parameters of radar pulse, such as radio frequency (RF) band, angle-of-arrival (AOA), time-of-arrival (TOA), pulse width (PW) band, amplitude (PA) The digital state vector is called radar *pulse descriptor word* (PDW) These PDWs are processed in a pulse-sort processor for deinterleaving the pulse trains It sorts each individual PDWs into groups of pulse trains where each has unique pulse repetition interval (PRI) associated with a particular radar emitter [22], [24]

After deinterleaving, the next step is to form the PDW of each group using PRI, RF and PW bands The PDWs are then compared against a stored threat library It contains limits on the ranges of pulse parameters associated with the known threat data types If any matches are found in the comparisons, it identifies the type of emitter which generated the measured pulse train

The key function used in the ESM system as overviewed is the matching of radar

* The enemy s electronic order of battle is a emitter data file which contains the ranges of pulse parameters associated with known threat data types, classified by threat significance

PDWs To perform such a function of the ESM system, we will consider an arbitrary ESM-Parameter Filter IC, which is described in the following section

3.2 ESM parameter filter IC

To devise an effective fault tolerance scheme it is important to have a good understanding of the architecture, functionality and operation of ESM Parameter Filter IC We describe the blocks which make up the system, their functionality and operation

3.2.1 General description

ESM (Electronic Support Measures) Parameter Filter IC implements 32 **match processors** using CMOS technology. Each of 32 match processors tests a 16 bit parameter data input for 32 independent ranges of limit values simultaneously, and outputs a discrete status signal M_i for each test within $100ns$

The parameter data inputs and set of limit values are actually radar pulse descriptor words generated by the ESM System's receivers, on the basis of a few selected parameters The set of limit values is collected according to the number of parameter values which are the likely ranges of the parameters for a given type of radar emitters

In a typical user configuration, several ESM Filter ICs operate in parallel upon different parameter fields of a given radar PDW, and the respective MATCH outputs are joined in a wire-AND configuration to form the ESM System MATCH output for each match processor Such parallel operation of ESM Filter ICs may become necessary to process PDWs which exceed 16 bits

There are various programmable options which configures ESM Filter IC to operate in different function modes. The "normal" function mode of a match processor, which we will focus on, is to output a match signal M_i equal to logic value 1 only if the 16 bit parameter data input is both less than or equal to the upper limit value (Y_i) and greater than or equal to the lower limit value (X_i). It is expressed as the following:

$$IF\ (X_i \leq InputData)\ AND\ (InputData \leq Y_i)\ THEN\ M_i = 1\ ELSE\ M_i = 0$$

We call such a condition as a **match condition**. There are 32 binary outputs, $(M_1, M_1, ... M_{32})$ corresponding to 32 match conditions. Each of such output vectors is referred to as a **match word**.

3.2.2 Functional description

Each match processor (window comparator cell) consists of a pair of 16 bit registers to store the upper limit (Y_i) and lower limit (X_i) values, a pair of 16 bit comparators which tests for ($InputData \leq Y_i$) and ($X_i \leq InputData$) respectively, a comparator function module to generate AND/OR/NOT combinations of the comparator outputs, a 7 bit window function (or match function) control register for selecting various operational function mode; and I/O control logic as shown in figure 3.1. A common Reset signal initializes all of the window control registers to the default mode ("no-programmed") so that all of the match outputs are forced to the "no-match" state (M_i=0).

An implementation of the proposed ESM-Parameter Filter IC, a pair of comparators (lower/upper) and a compare function module are the combinational part of the processor, which consists of about 700 transistors. The total number of transistors used in a match processor is about 1100.

All of the limit registers of ESM Filter IC are individually addressable for writing and reading by the Host controller. The Host controller also can program the window function

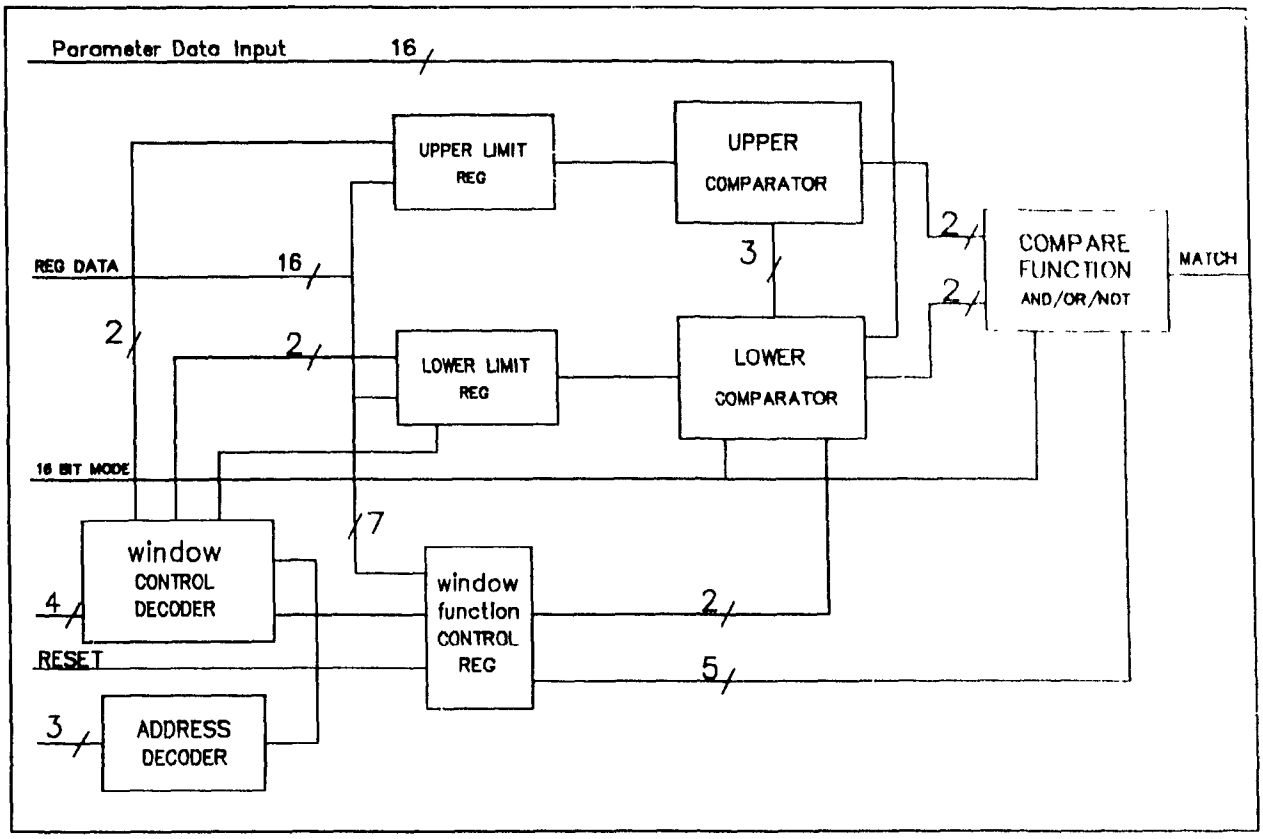


Figure 3.1 Block diagram of a match processor

control registers to set each match processors for desired function modes by four control signals: Zero cross-over(ZEROX), Accept(ACC), NOT, and REJECT. The "ZEROX" and "ACC" signals can be used in combination with "NOT" signal to invert the selected function mode. The "REJECT" control signal overrides all the other signals and sets the match output, M_i to zero ("no-match" state)

The "ZEROX" control signal, if activated, alters the "normal" function mode of a match processor to operate as the following

$$IF\ (X_i \leq InputData)\ OR\ (InputData < Y_i)\ THEN\ M_i = 1\ ELSE\ M_i = 0$$

This control signal is necessary specially when the upper limit value is near zero and the lower limit value is close to full-scale, and the range is defined across the zero point (initial reference point). It allows the match processor to wrap-around from full scale to zero. This

case may occur when the limit values are generated based on some radar parameters, such as angle of arrival

When several ESM Filter ICs are operating in parallel upon different parameter fields of a given radar PDW, it may become necessary to selectively disable testing of one parameter field without disrupting the compare tests of other related parameter fields This can be accomplished by activating the "ACC" control signal, which enforce the MATCH status for a particular match processor to the TRUE state Thus it enables the ESM System MATCH output to reflect the combined status of the related parameter field of the match processors which are still active

The "NOT" control signal, if activated, inverts the previous function mode of the match processor For example, if used in combination with the "ACC" control signal, the effect is to set the particular match processor output to the "no-match" state (M_i) which is in fact equivalent to the active "REJECT" signal. Similarly the "NOT" control signal can be used in conjunction with the "normal" function mode, or with the "ZEROX" control signal to perform the following functions respectively as expressed·

$$IF\ (X_i\ >\ InputData)\ OR\ (InputData\ >\ Y_i)\ THEN\ M_i := 1\ ELSE\ M_i := 0$$

$$IF\ (X_i\ \cdot\ InputData)\ AND\ (InputData\ >\ Y_i)\ THEN\ M_i := 1\ ELSE\ M_i := 0$$

These types of operation would be necessary if it is to program the Filter IC to ignore input data occurring within a known match range and search for other input data occurring outside of that range.

Once we introduce our graphical model which depicts the functionality of ESM-Parameter Filter IC, the control options "ZEROX" and "NOT" are illustrated with such graphs in chapter 5.†

† Refer to figure 5 3

Chapter 4 The Fault and Error Model

First step toward a fault-tolerant system design is to understand the likely physical failure modes in a given system so that redundant sources can be efficiently used to protect against such failures. This chapter is devoted to the study of fault and error models so as to define adequate models of our error detection techniques for ESM Filter. Concepts of technical terms used in fault-tolerant computing, and models of faults and errors in VLSI are reviewed. At the end, the fault and error model for ESM Filter are defined.

4.1 Introduction

The term *failure*, *fault* and *error* have different meanings in the context of fault tolerant computing. A system **failure** occurs when the delivered service deviates from the expected service because of an error. An **error** is a subset of the system state which is liable to lead to failure. The cause of an error is a **fault**. Thus, a system failure is the effect of an error on the service, and an error is the manifestation of a fault in the system [2], [3]. A fault in a system does not necessarily result in an error. An error occurs only when a fault is "sensitized" for a particular system state and input excitation. A fault that has not been sensitized is referred to as *latent* fault. Similarly an error may be *latent* or detected before it leads to a system failure. Some redundancy can be introduced to prevent an error from resulting in a system failure as what fault tolerant

system designers intend to do Causes of faults, referred to as **physical defects** or **failures** include mistakes in system design or implementation, and external disturbances, such as ionizing radiation, electromagnetic interference, damage, or other deterioration Faults caused by design mistakes and external disturbances are especially hard to model and protect against, since their effects and occurrences are very difficult to predict

In order to design a fault-tolerant system, it is important to consider the physical failure that are likely to occur with the specific technology being used. Once the likely physical failure modes are understood, it is desirable to determine the effects of those defects on the operation of a circuit at some higher level (i e , logic gate level, functional block level) A description of these effects is called a *fault model*. Fault models are also very important for test generation and evaluation of a test quality defined by the coverage of modeled faults (i.e , fault coverage of the fault simulation). However, fault models do not give direct insight to the behavior of errors due to the modeled faults on the output of a circuit during normal operation The behavior of errors, which is categorized by its nature such as unidirectional or asymmetric, nonrecurrent or recurrent, can be investigated by modeling the type of errors caused by physical failures

In the next section, we will briefly review some fault and error models considered in VLSI technology With this in mind, we will then define appropriate fault and error models for our error detection schemes

4.2 Fault and error models for VLSI

Fault models -

We consider fault models for general NMOS and CMOS VLSI digital circuits

One of the classical and still widely used fault model is the gate-level stuck-at fault

model [26]. In this model, it is assumed that physical defects or failures will result in the lines at the logic gate level of the circuit permanently being stuck at logic value 0 or 1. Because of the cost constraints, single line stuck at fault is typically assumed. This fault model is quite accurate for small and medium scale ICs where bonding failures can be represented by permanently stuck lines at the logic level. However, as many works have already shown [27]-[31] this model is not sufficient for modeling physical failures in MOS VLSI circuits. In the following such cases where the simple stuck-at fault model can not represent the given physical failure modes are discussed.

In a CMOS inverter, suppose that the n-channel transistor fails permanently in the on state. If 1 is applied to the input, no error will appear since the correct logic value (0) is equal to the incorrect value. On the other hand, if 0 is applied to the input then both the n- and p-channel transistors will be conducting since n channel can not be turned off. This will result in an output voltage that lies between the voltages assigned to logic 0 and logic 1 (i.e., indeterminate mode). Same problem can occur with incorrect dosage of ion implants, which may cause a threshold shift in a load transistor in nMOS.

Other failures may cause a combinational circuit to become sequential circuit, or sequential circuit to become a combinational circuit as shown in [29]. In CMOS, a break in a line or a transistor permanently in the off state can make the output of a combinational circuit dependent on the previous output rather than the current input. A fault model for this type of failure, referred to as *stuck-open fault*, was described in [31]. Such a fault may not be detected even if all possible input vectors are used to test unless a specific sequence of test pattern is applied.

Some physical failures are especially difficult to model because they may disappear after producing errors in a random way. These failures are referred to as *nonpermanent faults*. The nonpermanent faults are again divided by *intermittent* and *transient faults*

The distinction between the two is made by their recurrent nature and the applicability of repair

Transient faults, usually the result of temporary external disturbances, exist for a finite time only and are not repeated Since the fault occurs only once and leaves no physical damage* to a circuit thereafter, it is not possible to repair Detection of transient fault usually require on-line detection method since it is not possible to test for such faults

On the other hand, intermittent faults are recurrent faults that result from marginal or unstable device operation such as a bad wire bonding. They are potentially detectable and repairable by replacement or redesign. While some permanent fault model can be applied to intermittent faults, transient faults do not have a well-defined basic fault model

Transient and intermittent faults typically occur with greater frequency than permanent faults and hence, are becoming a major cause of errors in systems [1]

- **Error models** -

Error models are ways of classifying the effect of physical failures on the system during periods of normal operation

Unidirectional error model describes the type of errors when a physical failure creates an electrical short or open and all the resulting bit errors at the output of a circuit are all of the same type, either 0s becoming 1s, or 1s becoming 0s. An example of this behavior can be found in some of the LSI single-transistor-cell memories, where failures are most likely caused by the leakage of charge [4] If the presence of charge in a cell is represented by logic value 1 and the absence of charge as 0, then the errors in these types of memories

* However it is possible for a transient fault to cause a permanent change in the state of a circuit with memory elements

can be modeled as $(1 \rightarrow 0)$-type unidirectional errors since the charge can not be created except by a rewrite process. Hence, $(0 \rightarrow 1)$-type errors are most unlikely in this case

Independent error model represents the type of errors where errors may occur in either directions and the two events ($0 \rightarrow 1$ and $1 \rightarrow 0$ errors) have an equal chance of occurring The errors in some of the random access memories can be considered by *independent error model* The *independent error model*, sometimes called as *symmetric error model* has been assumed for a long time in developing most of the codes [4]

However, it has been experimentally validated that in general, such assumption is not valid, and so more general bidirectional error model, called an *asymmetric error model* was introduced [32] The *asymmetric error model* does not assume that the probability of the event 1(correct value) $\rightarrow$ 0(erroneous value) occurring and the event 0(correct value) $\rightarrow$ 1(erroneous value) occurring as being equal Thus, *independent error model* can be viewed as a special case of the *asymmetric error model*.

Soft errors are temporary, nonrecurrent errors caused by transient faults such as ionizing radiation (i e , alpha particles, cosmic rays) and electromagnetic interference [33] Once corrected, soft errors usually leave no physical damage in the system For detection of soft errors, concurrent error detection technique can be used based on a reasonable assumption regarding its frequency of occurring Single bit error detection/correction has been shown to be sufficient for most soft errors in MOS memories [34], [35] It has been also reported [33] that the soft error rate of a large VLSI chip ($1\ cm^2$) could be on the order of 10^{-4} per hour based on a number of reasonable assumptions However, *soft errors* may become the limiting factor for reliability as circuit feature sizes are scaling down for higher densities as discussed in the same report [33]

Another aspect of errors is its bit dependency, independent bit error or physically clustered bit errors For example, if the bits of a coded word are stored sparsely, their

errors are more likely to be independent, whereas if the bits of word are densely stored, it is more likely that errors are clustered.

4.3 The fault and error model for ESM filter

The fault model -

The traditional fault model of single line permanently stuck-at 1 or 0 has been shown to be inadequate for VLSI technology The trend has been toward broader fault models including transient/intermittent faults which are confined to a certain area rather than line

In VLSI technology, a more appropriate fault assumption is that failures in a circuit affects a small area of the chip and that the nature of the failure is not precisely known [16], [36], [37] The assumption that physical failures in components simply change the logical values of the outputs of these components may not always be valid since indeterminate logic levels may occur as we described earlier However, at some higher functional level, the effect of failures will be recognized as changes in logical values. Thus, in lack of the knowledge of physical failure modes, it is more reasonable to deal with the failures at the functional level

Throughout the rest of this thesis, we assume that physical failures confined to a certain area of the chip will make their effect at the level of a functional block in terms of altered logic values of the output. The functional level we choose for our fault model is at the level of a match processor of ESM Filter IC. Furthermore, we assume that at most one processor may be faulty within a given period of time, which will be relatively short compared to the mean time between failures; this assumption is practical if the chip is periodically checked to flush out the latent failures (i.e , permanent/intermittent errors) Hence, we are focusing on soft errors, which is becoming a major cause of errors in many systems However, physical defects on the area of outer perimeter of the processor, which

are devoted for data paths, interconnection lines, busses are not covered by this fault model. The fault-free assumption on these components has been remarked as unreasonable [36] [38], [39] since large area of a VLSI chip is used up by such elements. Specially for FSM Filter where the input data to each processors are fed by common data paths, any physical defects such as a single broken line that fans out to many input lines may result in multiple faults. We assume that failures on these parts can be protected by error correction schemes such as alternate data retry (ADR) [15],[25] and Hamming codes [40], which are effective for data paths, communication lines. Therefore, we will concentrate on the physical failures of the processor array.

We also note that the failures confined to a certain area of the chip is not necessarily same as the failures confined to a certain functional block of the chip. That is true only if the area of the chip does not contain unrelated lines and components of other functional blocks. Thus, any failures occurring in the area would never affect other functional blocks, but only the block exclusively laid on its confined area. It is possible to layout a circuit in such a way that the area of a functional block does not contain components or lines of other blocks. We will assume this to be the case for our error detection techniques.

- The error model -

We assume that at the most one error occurs per operation cycle. A faulty processor may produce erroneous output bit in either way, 1(correct value) → 0(erroneous value) or 0(correct value) → 1(erroneous value). Since we do not know the probability of (1 → 0) error occurring and the probability of (0 → 1) error occurring, it is reasonable to assume that two probabilities are not equal. Hence, *asymmetric error model* will be assumed for an error occurring in the processor array of ESM Filter.

Chapter 5 Concurrent Error Detection for ESM filter

In this chapter we will consider a concurrent error detection technique, which is particularly appropriate for ESM filter. The method will utilize a code that is inherent in the function of ESM Filter

From chapter 3, we have noted that each match condition is defined by two limit values, Lower Limit values (X_i) and Upper Limit values (Y_i) For n pairs of limit values, it would appear that there exist 2^n number of match words. In the following section, however, it is shown that at most $2n$ number of distinct match words represents valid information instead of 2^n. Thus, for n=32, maximum 64 distinct match words instead of a total possible 2^{32} match words represent valid information. The valid subset of match words will be called **MATCH code** Then the problem becomes the error detection with the code, which consists of determining whether a match word is a correct match code word as described in figure 5 1

First we will define a graph model which describes the functionality of ESM filter Then it will be shown that the output of ESM filter forms a kind of code, referred to as MATCH code from the model The algorithm for extracting the code and its property will be discussed A concurrent error detection scheme based on MATCH code will be described The cost-effectiveness of the scheme is measured by hardware overhead ratios

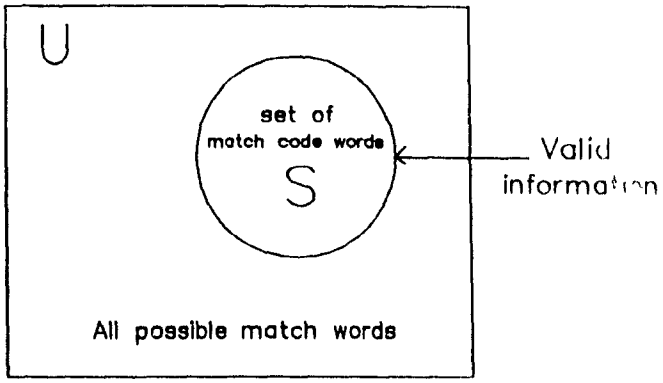


Figure 5.1 Code space

depending on various options for implementing the technique It will be followed by the performance measure on the proposed error detection scheme in terms of its error coverage

5.1 The error detection code

5.1.1 Graph model of ESM filter function

Since the input space of ESM filter is finite ($0 \sim 2^{16}$), the entire input space can be regarded modulo 2^{16} Thus a circle is suitable to describe the finite input space of ESM filter. Then each match condition is defined by two nodes (lowe/upper limit values) placed on the circumference of a circle Hence the circle becomes a polygon with $2n$ number of nodes and edges In this manner we describe the functionality of ESM filter as shown in figure 5.2 according to the following definitions

- **Definitions** -

(1) A **Zero point** is the initial reference point indicating zero scale

(2) A **match region** m_i is a path of consecutive nodes and edges bounded by a lower

limit(X_i) and upper limit(Y_i) counterclockwise where intermediate nodes represent limit values

(3) A **code region** C_j is an edge determined by a pair of *consecutive nodes.*

(4) A **zero code region** C'_j is the code region where no match condition is specified. The labels of zero code regions are marked by superscript prime(′)

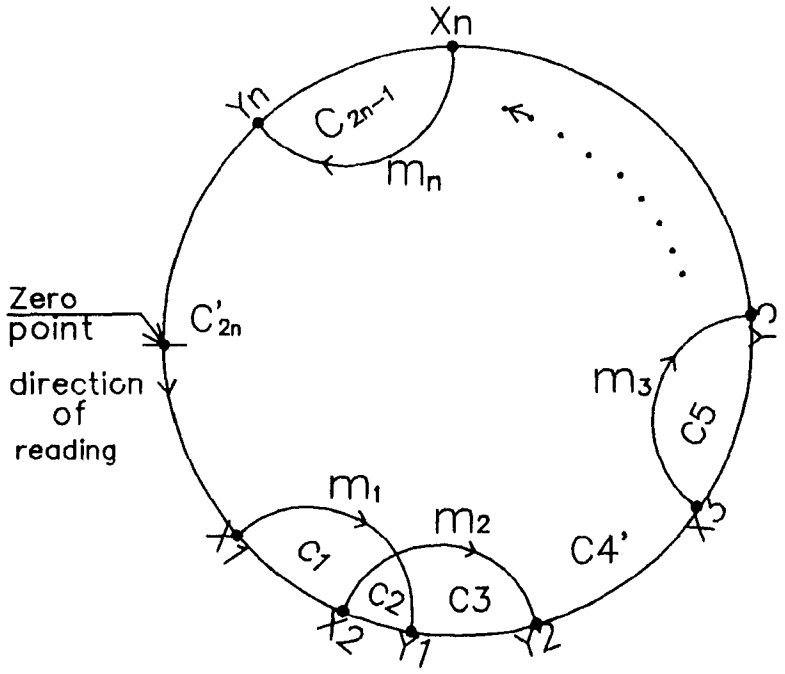


Figure 5.2 A circle representation of n match conditions

To illustrate the above definitions consider the following example:

Suppose we have $2n$ limit points $\{(X_1, Y_1), .., (X_n, Y_n)\}$ embedded on the circumference of a circle as shown in figure 5 2 for n match conditions $(M_1, ., M_n)$ Then set of edges or an edge enclosed by a directed arc line that are incident to two points (X_i, Y_i) represents a **match region** as labeled m_i. The number of match regions is naturally the same as the number of match conditions.

A **code region**, labeled as C_i, is an edge where an arbitrary input data may fall in, and the region covers a set of match conditions that can be satisfied if the input data lies within the range Such a set of match conditions for each code region is defined as a **match code word** For example, C_2 is an edge determined by the pair of consecutive points (X_2, Y_1). If a 16-bit input data (D) satisfies the condition $(X_2 \quad D \quad Y_1)$, two match conditions M_1 and M_2 are simultaneously matched Therefore, the match code word $(M_1, M_2, .., M_n)$=(1,1,0,...,0) would be the correct match output response for code region C_2. C'_4, C'_{2n} are **zero code regions** since no match conditions are specified in those regions. Hence, the match code words for zero code regions are all 0's, $(M_1, M_2, .., M_n)$=(0,0,...,0). We use notation $\{C_i\}$ to indicate the set of input data that belongs to C_i, and $|C_i|$ for the cardinality of a set $\{C_i\}$ In the same example, any input data D satisfying $X_1 < D < X_2$ is denoted as a set $\{C_1\}$. Similarly $\{C_2\}$, $\{C_3\}$ denote sets of input data D satisfying respectively $X_2 < D < Y_1$ and $Y_1 \quad D \quad Y_2$

Using the circle diagram the control options of ESM filter, ZEROX and NOT can be described as shown in figure 5 3.

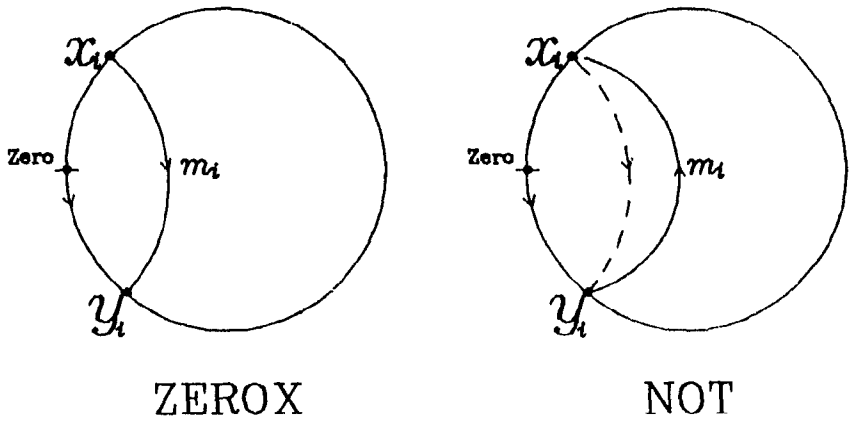
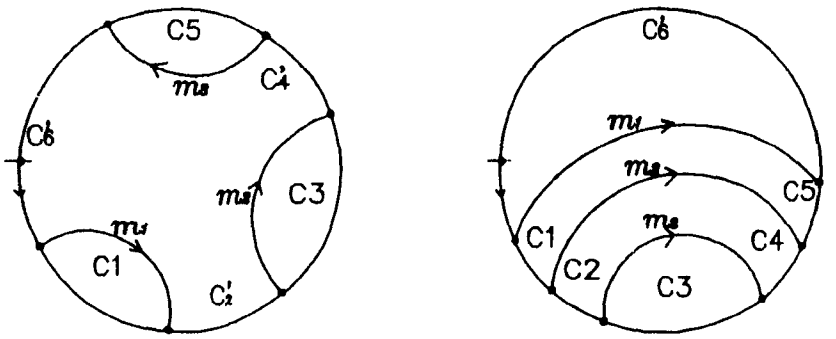


Figure 5.3 The control options ZEROX and NOT

From the diagram it can be shown that there exist at most $2n$ distinct valid match

code words for any given set of n match conditions ($n \geq 1$). Having some n number of match conditions or $2n$ number of limit values is the same as to place $2n$ nodes on a circle. The polygon formed by $2n$ nodes naturally has $2n$ edges. Each edge then corresponds to each code region Each match code word represented by a distinct code region is not necessarily a distinct word Such is the case when more than one zero code regions exist (case a) Another possibility is that certain number of match regions are enveloped together (case b) Both cases are illustrated in figure 5.4 For case (b) two distinct code regions $\{C_1, C_5\}$ have the same match code word $(M_1, M_2, M_3) = (1, 0, 0)$. Similarly a match code word $(M_1, M_2, M_3) = (1, 1, 0)$ represents two code regions $\{C_2, C_4\}$ Therefore the number of distinct match code words for n match conditions is less than or equal to the number of edges of the graph - $2n$.



(a)
Same match code word for {C'2, C'4, C'6}

(b)
Same match code words for {C1, C5} and {C2, C4}

Figure 5.4 Identical match code words determined by some code regions

In another example we refer to the figure 5.5 There exist 8 code regions for match conditions (M_1, M_2, M_3, M_4) Since we have same match code words for C'_6 and C'_8, the

total number of distinct match code words is 7 as listed in Table 5 1

$$M_1 = (\ X_1 \leq \mathit{Input\ Data} \leq Y_1) \qquad M_3 = (\ X_3 \leq \mathit{Input\ Data} \leq Y_3)$$

$$M_2 = (\ X_2 \leq \mathit{Input\ Data} \leq Y_2) \qquad M_4 = (\ X_4 \leq \mathit{Input\ Data} \leq Y_4)$$

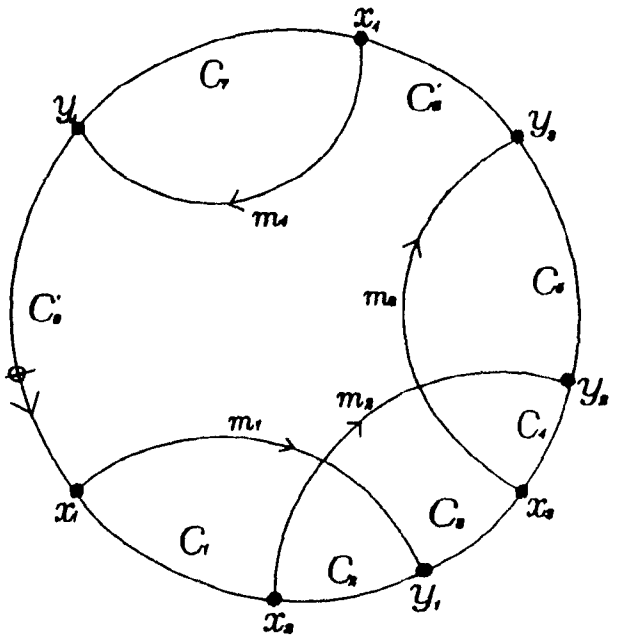


Figure 5.5 Example with 4 match conditions

Code Regions	match conditions satisfied	Match Code Words M_1	M_2	M_3	M_4
C_8', C_6'	$\{\ \}$	0	0	0	0
C_1	$\{M_1\}$	1	0	0	0
C_2	$\{M_1, M_2\}$	1	1	0	0
C_3	$\{M_2\}$	0	1	0	0
C_4	$\{M_2, M_3\}$	0	1	1	0
C_5	$\{M_3\}$	0	0	1	0
C_7	$\{M_4\}$	0	0	0	1

Table 5.1 Example with 4 match conditions

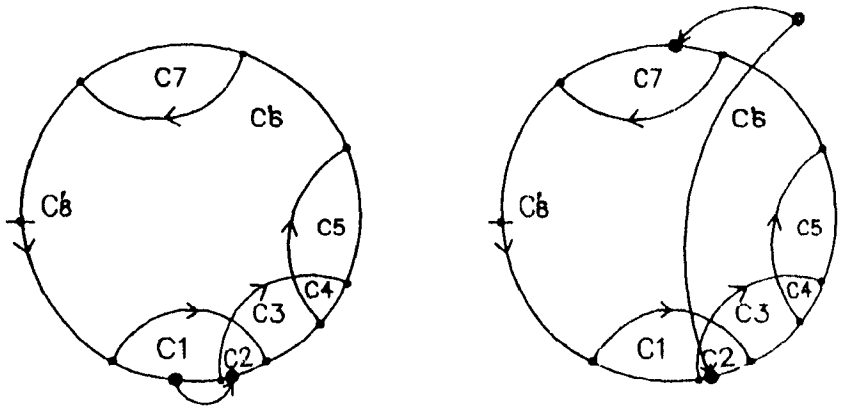
The size of match code words relative to that of all match words is considerably small when match conditions are many: $\lim_{n \to +\infty}(2n/2^n) = 0$ In what follows we describe different types of errors that can occur in the previous example

- **Types of errors** -

The circular diagrams are used to explain two types of single errors in figure 5 6 The first type of error occurs when the faulty processor generates a match condition which still conforms to the match code As a result, it still gives us a match word that is not a correct match code word In the example of figure 5.6 (a), suppose the input data that actually belongs to code region C_1 is appeared to be in region C_2. Then it changes the match word $(M_1, M_2, M_3, M_4) = (1\ 0\ 0\ 0)$ to another match word (1 1 0 0), which conforms to the code Thus the error can not be detected by the code.

In the second case, more than one code regions contain the input data at the same time as illustrated in figure 5 6 (b) The input data is simultaneously present in both code regions C_2 and C_7 In consequence, it outputs a match word $(M_1, M_2, M_3, M_4) = (1\ 1\ 0\ 1)$ that no longer conforms to the code words in table 5 1. Thus the presence of the noncode word indicates an error

While both errors are due to the erroneous output bits being 1, the first type of single errors are undetectable by the match code unlike the second one There exist other types of single errors that are due to erroneous output bits being 0. The input data unseen in a code region would result in such errors Hence, errors can be classified by their direction of occurring as well as their detectability These two elements will be well reflected on the analysis of error coverage



(a) Error type 1 (b) Error type 2

Figure 5.6 Two types of single errors in a circle model

5.1.2 Extraction and property of MATCH code

The algorithm, MATCH extracts the set of match code words for a given set of limit values It mainly consists of two procedures, sorting and scanning In the process of sorting a list of limit values is ordered according to linear order such as · Simplest sorting algorithms, such as selection sort and bubble sort, take $O(n^2)$ time to sort n number and are good enough for sorting short list.

Once the sorting is completed, we have well defined code regions In the process of scanning, all the code regions are scanned, and the corresponding match code words are generated for every code region This procedure takes $O(n)$ steps

```
for i:=1 to 2n do
  read(NextLimitValue).
  {scan code regions to verify which match conditions can be satisfied}
  If NextLimitValue =X_i then M_i =1 else { match condition M_i is satisfied }
  If NextLimitValue =Y_i then M_i =0.  { match region m_i is scanned completely }
    {generate a new code word for each code region scanned}
end.      {scan}
```

The overall complexity of MATCH algorithm is $O(n^2)$ for n number of match con

ditions

We can imagine this scanning procedure as if an input data is moving along counterclockwise around the circumference of a circle. As it passes through each match region once, each match condition is satisfied only for once during the entire circular scanning. For this reason the column of the code generated in this manner has only one segment of consecutive 1's in a cycle The distance* between adjacent code words varies depending on the given set of limit values. If the code is extracted from a set of distinct limit values then every match code word has at least one match code word that is distance one away from it.

For three match conditions ($n = 3$) we list different types of **match patterns** and match codes respectively in figure 5 7 and table 5 2.

a			b			c			d			e		
$M1$	$M2$	$M3$	$M1$	$M2$	$M3$	$M1$	$M2$	$M3$	$M1$	$M2$	$M3$	$M1$	$M2$	$M3$
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0	1	1	0
0	1	0	0	1	0	1	1	0	1	1	0	0	1	0
0	0	1	0	1	1	0	0	1	0	1	0	0	1	1
			0	0	1				0	1	1	0	0	1
									0	0	1	1	0	1

Table 5.2 Match codes for five different match patterns

* The distance between two word is the number of bit positions in which they differ

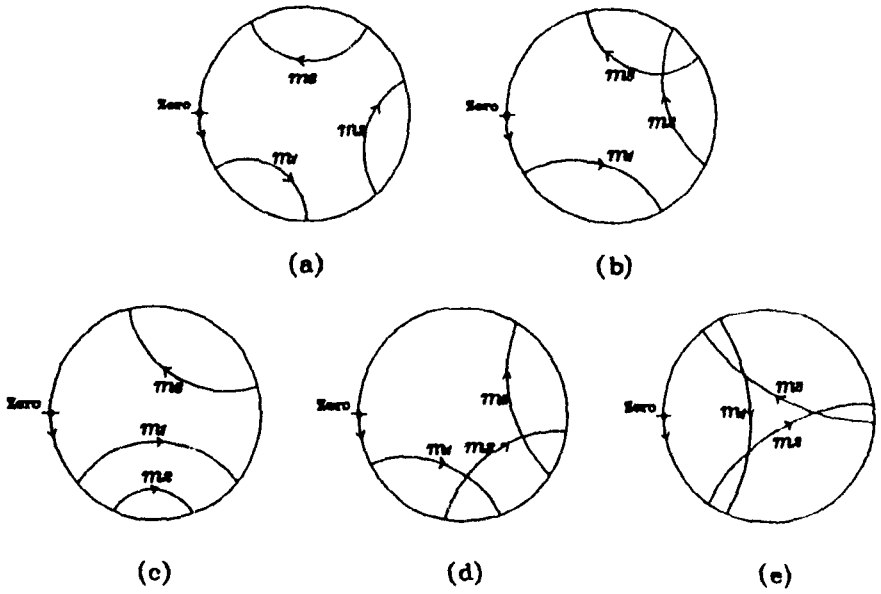


Figure 5.7 Various match patterns for n=3

5.2 The error detection scheme

Knowing that the output of ESM filter forms a code, called MATCH code, we investigate how the code can actually be used to detect single errors in the following.

5.2.1 Assignments of limit values

The Hamming distance between two words is the number of bit positions in which the words differ The minimum distance of a code is the minimum Hamming distance found between any two code words [41] If a set of limit values is given so that the set of match code words is extracted from match words, then the minimum distance of match code words can be determined In general, a set of limit values may have to be updated according to the changes in EOB by the Host Controller. Hence the minimum distance of match code words is not always the same unless we make the following assumptions

We assume that all set of limit values are assigned properly to each match processor so that each set of match processors checks for a unique match pattern that can not be examined otherwise by using fewer number of match processors. For examples in figure 5 8 $(a).(b).(c)$, a single match processor could have been sufficient to check the same match patterns that are checked by using two processors. Thus they are not reasonably assigned limit values

Those undesirable cases are consequently excluded if we assume that *no identical limit values exist for any given set of match conditions* The code extracted from such set of limit values will always have the minimum distance equal to one Since the greater the minimum distance of a code, the more advantage we have in detection of all single errors, this restriction sets us for the worst case As we also recall the physical meaning of the limit values from chapter 2, it is very unlikely to have identical limit values in a given set

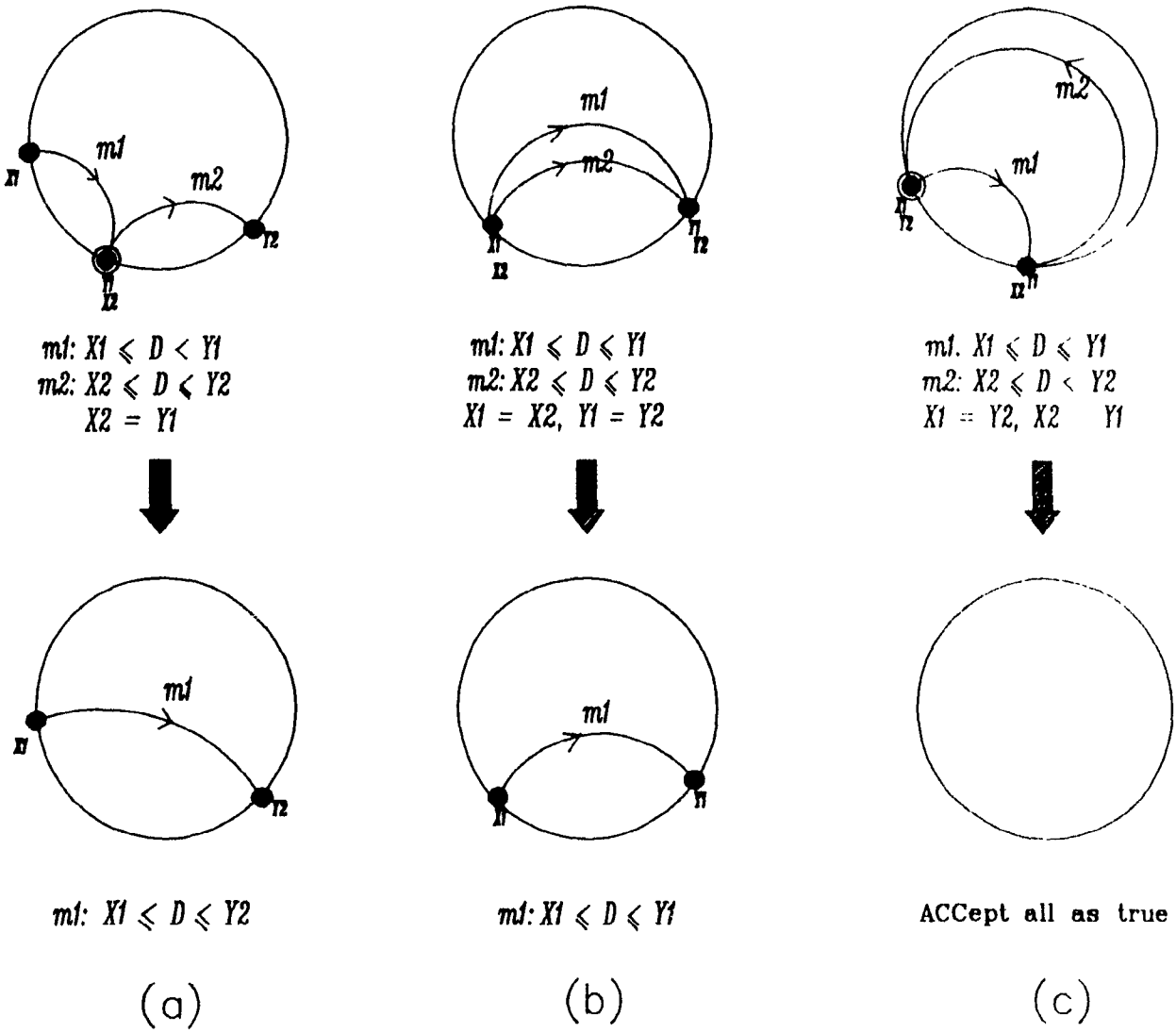


Figure 5.8 Assignments of limit values

Therefore every match code word has at least another match code word that is distance one away from it in the same code

5.2.2 Single error detection

During fault-free operation the output is always a match code word If the minimum distance between match code words is greater than or equal to two, then all single errors result in a noncode word which is easily detectable Unfortunately in our case, single errors

may change the correct output into another match code word - incorrect code word as described in figure 5.9. These are the types of error we have seen in figure 5.6 Thus, all single errors are not detectable by the code.

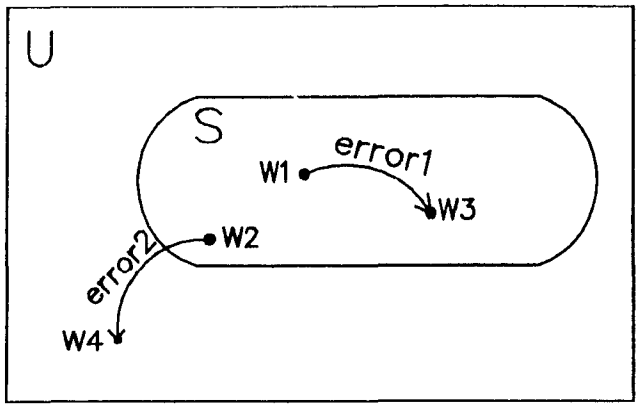


Figure 5.9 Errors in an error-detecting code

One of the ways to deal with these undetectable errors is to add extra match processors. For this we would like to know the minimum number of extra processors required to ensure the detection of all single errors. In what follows we show that it takes twice the number of original processors to detect all single errors.

Proposition 5.2 At least n extra match processors are needed so that the minimum distance of a code can be equal to two for a given set of n match conditions

Proof. Since no limit values are identical, there always exist $2n$ nodes and edges in a polygon which represents a given match pattern. Any adjacent edges* correspond to match code words that are distance one away from each other Each edge has two

* Edges that are connected by a common node are adjacent edges

adjacent edges in a polygon Hence each match code word has two code words† that are distance one away from it Introducing an extra match processor is equivalent to adding two extra nodes to the polygon. As illustrated in figure 5.10 these two extra nodes can take the identical values from the original set of limit values. Now the whole graph can be considered as the union of two components, A and B - one piece of match region covered by an extra match processor, the other piece that is left uncovered

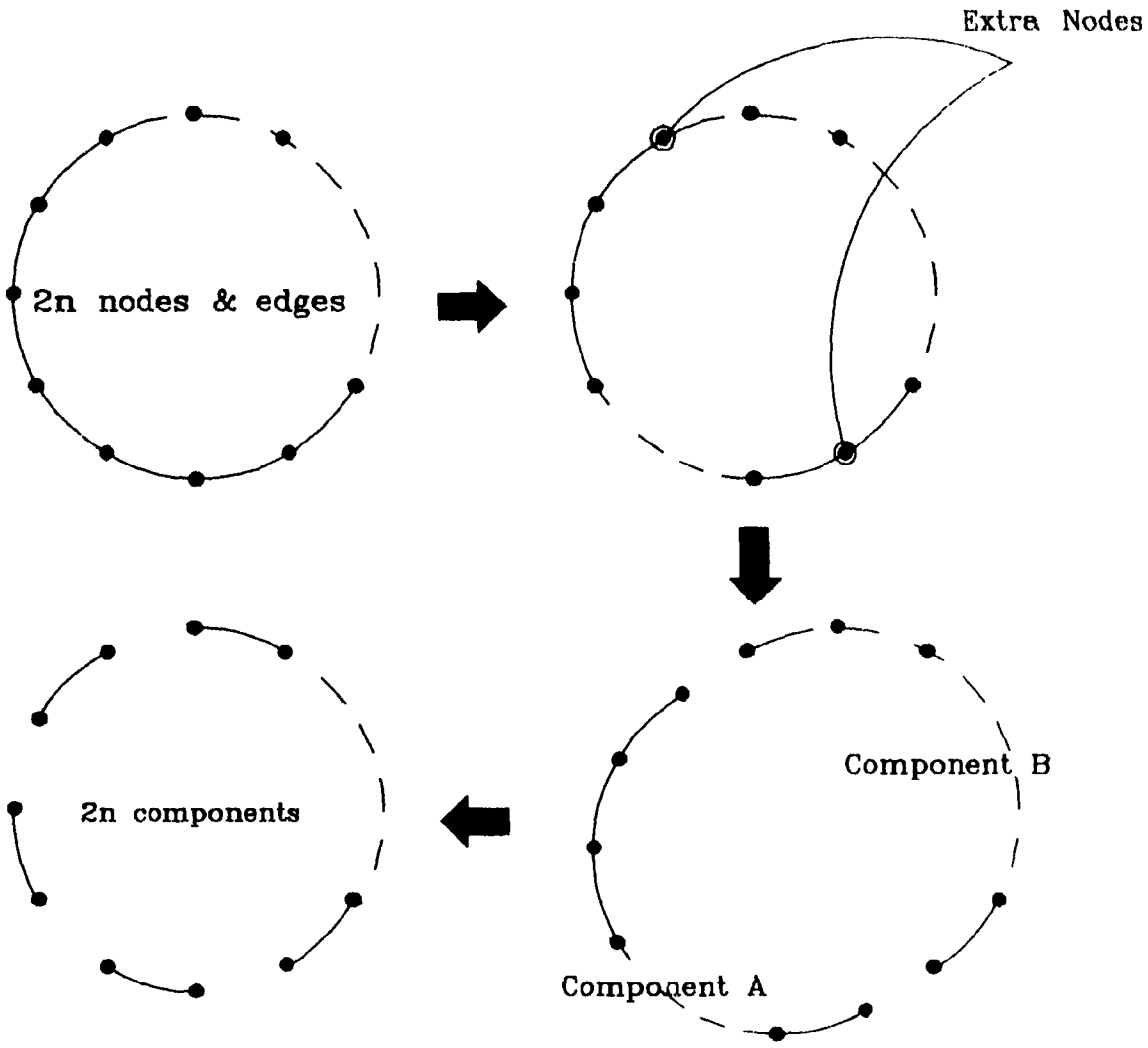


Figure 5.10 Dividing a polygon into n components

† These are not necessarily distinct words as pointed out in figure 5 4

Then the distance between the match code words of edge set A[‡] and B is increased by one Therefore the minimum distance between the match code words from the two different groups is guaranteed to be two But those code words in the same set of edges have the minimum distance of one To ensure the distance of two at least between every code words, each edge should be covered by different component of the graph. For $2n$ edges we need to divide the graph into $2n$ components as shown in figure 5 10. It takes at least n pair of nodes, which is equivalent to n extra processors Q E D

The duplication of 32 match processors is an expensive way to achieve the detection of all single errors An alternative is to use a time redundancy technique for the detection of all single errors as described in chapter 6 However, the speed of ESM filter is a very crucial aspect since it is designed for real time application. Therefore, we do not prefer to trade time for area

5.2.3 Lookup table based on CAM

We have shown that the duplication of 32 processors are inevitable to detect all single errors However, based on the match code we can design more cost efficient method than the duplication, which is able to detect, if not all, most of single errors

The outputs of each match processor, **a 32-bit match word** can be compared against all match code words stored in a lookup-table (32-bit × 64-words) to check its membership If the match word does not conform to the code stored in the lookup table, it indicates the presence of an error By this practice, we can not actually detect all single errors due to the limit on the minimum Hamming distance of the code as discussed earlier However, the effectiveness of this approach can be shown by estimating the percentage of detectable single errors

‡ Set of edges in component A

The lookup-table can be provided using a content addressable memory (CAM) [†] The comparison between a match word and set of match code words could be done also using extra registers and comparators. CAM based lookup-table is preferred for our case because it is more efficient in area and time

First we introduce the basic concept and general architecture of a content addressable memory to consider its effectiveness and efficiency as a lookup-table

5.2.3.1 Introduction to CAM

Conventional random access memories (RAMs) access memory serially by location which consumes great deal of time and leaves a large proportion of the memory hardware idle To improve the speed of a memory as well as its density, content addressable memories have been considered for some time The regular and iterative structure of the CAM is very suitable for VLSI technology, which has improved the feasibility of content addressable memory systems and has overcome many implementation obstacles

A content addressable memory (CAM) is a memory system with the property that stored data items can be retrieved by their content or part of their content rather than by their address In order to retrieve stored data items by their content, a CAM accesses the memory words by comparing their content with the given search-key word [42] [46]

The basic element of the CAM is called the *bit cell or CAM cell* One bit informations can be written in, read out, and compared to the interrogating informations by the array of CAM cells A bit cell is able to signal exact hits (matches) or no hits (mismatches) [‡], as well as have the capability to be externally masked A hit is detected if the stored

[†] CAM has been also called associative memory distributed logic memory parallel search memory data-addressed memory

[‡] To avoid dual usage of the word "match", we prefer to use the word hit in the context of the

and presented datum have the same logical value or if either contains a "don't care". The external masking is used to exclude the same bit positions in all words from comparison. Therefore three possible states must be presented to the cell· ONE (hit), ZERO (no-hit), or "don't care" (MASK)

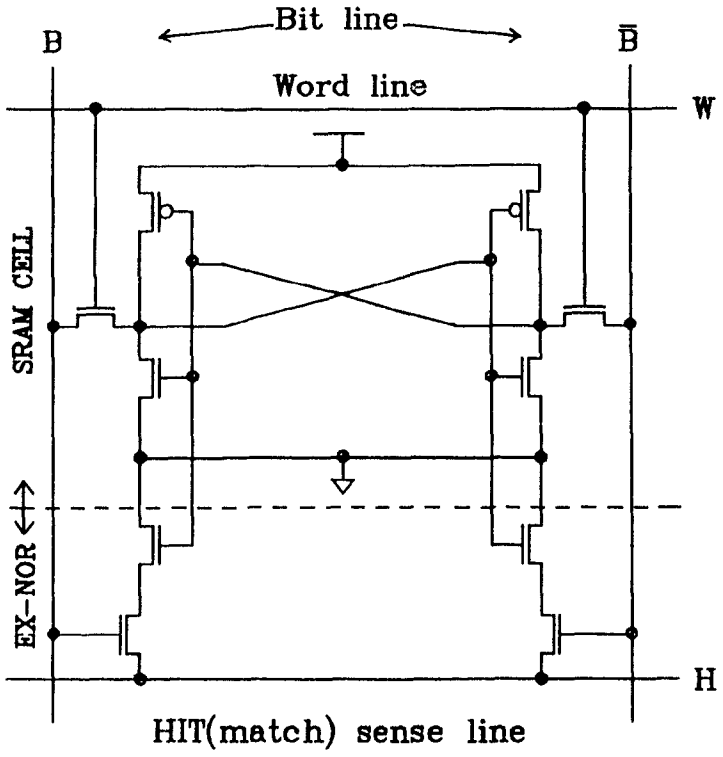


Figure 5.11 A static CMOS CAM cell

Figure 5 11 shows an example of a static CMOS CAM cell. The CAM cell is composed of a static random access memory (SRAM) cell plus an exclusive-NOR with a total of ten transistors Figure 5 12 shows the content addressable memory organized as an array of such CAM cells

-Equality comparison -

CAM

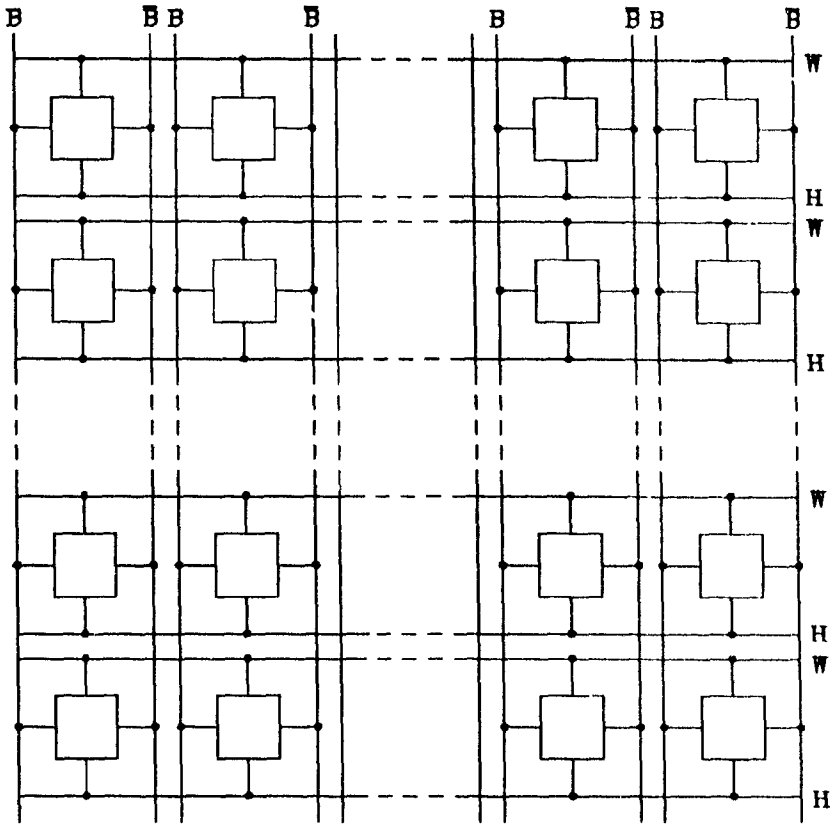


Figure 5.12 Memory organization based on the array of CAM cells

In equality search operation a search-key word is compared with all words stored in a CAM for the equality check. A search-key word is presented to the array of CAM cells simultaneously The bits of search-key word are compared against the respective bits of the word rows of the CAM cells These equality comparisons are executed simultaneously in an all-parallel CAM Here each CAM cell has its own output lines (HIT sense line) as shown in figure 5 11. Each of cell's comparison results in a word lines has to be collected by an AND or NOR logic to form a final hit or no-hit signal associated with each word Implementation of these functions by a logic gate would have very high fan-in, which is not a practical solution One efficient solution practiced in many cases [43], [47] is the *wired-AND* logic The output lines from each CAM cells of word rows can be wire ANDed together Initially, each of the wire ANDed lines is precharged high, and remains high if

each cell's logic value in the CAM word is identical with the presented data on the bit lines Conversely, if any cell in the word mismatches the presented data, the line will be discharged to a low potential

Figure 5 13 shows an all-parallel content addressable memory. The all-parallel CAMs are extremely fast with an access time of only a few nanoseconds, and therefore typically used in fast but small buffer memories because of the high price per bit cell compared to that of usual addressed memories The system inspects the words in the memory in a parallel fashion using the search-key word and the mask register The data can be input to the memory for storage either serially or simultaneously (i e when initializing data elements) The output hit response indicates which words hit the key word and mask combination, and the output register reads the hitting words

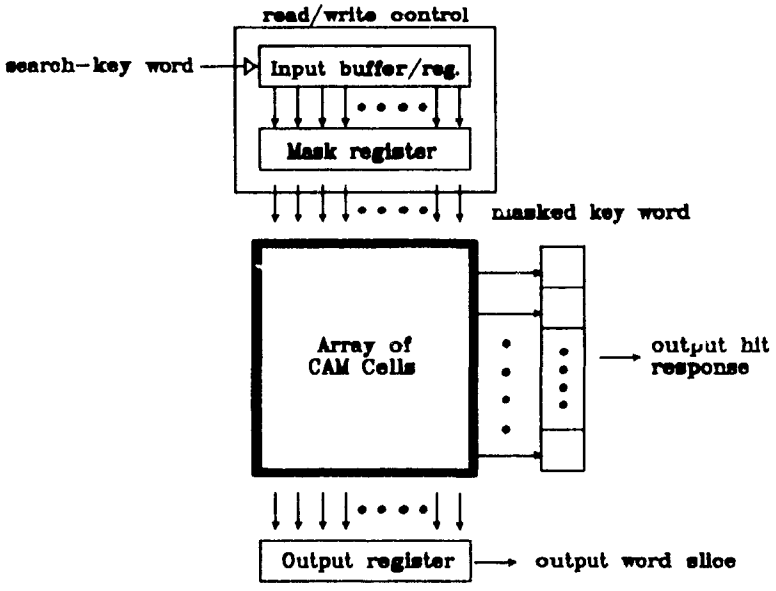


Figure 5.13 All-parallel CAM block diagram

There are other CAM architectures such as the Word-Parallel/Bit(or Byte)-Serial CAM and the Word Serial/Bit(or Byte) parallel CAM [43] The bit- and byte slice architectures perform better on arithmetic computation, while the all-parallel architecture is best

for equality comparisons [42], which satisfies our requirement

5.2.3.2 The Match code lookup table

For our application, we need a high-speed lookup table of size (32 bit × 64 words) which can perform parallel examination of the content (set of match code words) and output the equality comparison results indicating whether a perfect hit exists for the given 32 bit match word. The lookup table can be implemented based on an all parallel CAM architecture as shown in figure 5.14. In this case the masking function is omitted since all 32 bits are always to be compared. The output register is not mandatory since our main concern is to only know if single hit response exists (no error) or not (error). No error is detected if there is only one hit in the output response. If there are no hits, or more than one hit in the output response, then an error is detected. If we presume that newly added components are fault free, the case of multiple hits can be discarded. Then either single hit or no hit would be the only possible output response. Under this assumption the output response lines from each CAM cells of word rows can be wire-ORed together to form a final error signal.

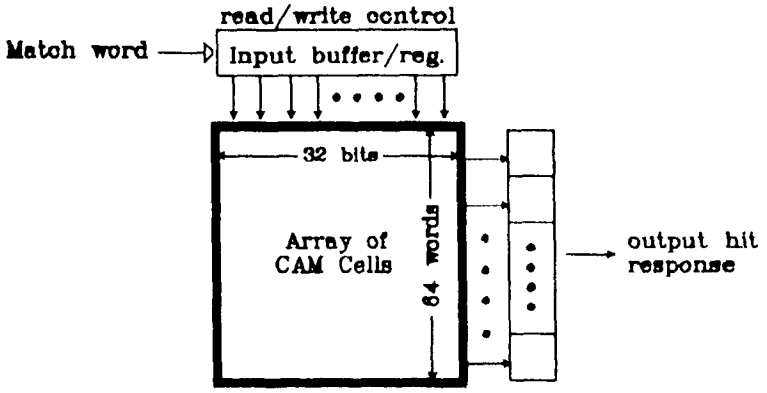


Figure 5.14 The all-parallel lookup table

The complexity of the lookup table depends on the various choice of CAM cells

There are different types of CAM cells such as static, pseudo-static [47] and dynamic CAM cells [48], [49] The investigation into the design constraints, trade-offs and implementation issues for various types of CAM cells using VLSI CMOS technology has been reported in [47] † These CAM cells are designed to meet or surpass the following performance requirements.

equality search: $< 5\ ns$; write: $< 5\ ns$; read: $< 5\ ns$

Hence the time delay to detect an error is reasonably less than the time constraint of the match processor - $100ns$ In addition to the static CMOS CAM cell as shown in figure 5 11, two different types of CAM cells as shown in figures 5.15, 5.16 called as pseudo-static nMOS CAM cell and self-isolating CAM cell respectively, are also considered.

The pseudo-static nMOS CAM cell in figure 5 15 with a total of eight transistors is the simplest design considered The data storage part of the cell is pseudo-static, and needs to be refreshed frequently Hence the power consumption is relatively higher than the other designs, but with the modest area cost

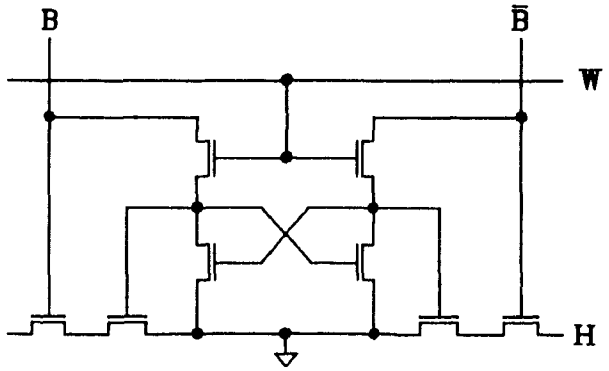


Figure 5.15 A pseudo-static nMOS CAM cell

The static CMOS CAM cell in figure 5 11 stores a datum in two cross- coupled

† Different types of CAM cells are designed to support cost-effective image associative processors, which requires real-time processing

CMOS inverters, and therefore does not need refreshing. The power consumption of the CAM cell is lower than that of the pseudo-static cell, but it would require two additional transistors

The self-isolating CAM cell in figure 5.16 isolates the data storage part of the cell from the two bit lines B, $\overline{B}$ during write operations This saves the power consumption that is required for write operations The power consumption of the self isolating CAM cell is the lowest of all considered, but it takes a total of twelve transistors

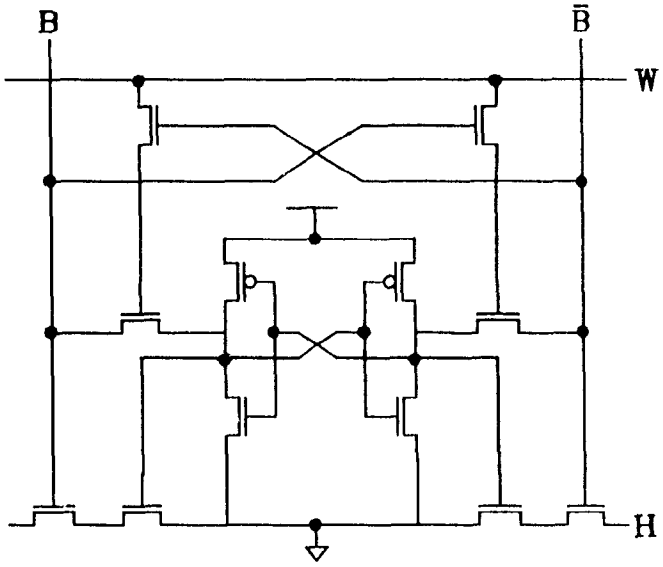


Figure 5.16 A self-isolating CAM cell

We summarize the variations in these types of CAM cells, and their respective *hardware overhead ratios* * involved in the implementation of the lookup table for ESM filter chip as below

power consumption	pseudo-static nMOS	static CMOS	self-isolating CAM
transistors # /cell	8	10	12

* The ratio of the extra hardware required by the fault tolerance technique to the hardware of the original system without fault tolerance Here the ratio is given in terms of transistors not actual area

overhead ratio(%)	46	57	69

The hardware overhead ratios are approximated as the transistor number ratio of the array of CAM cells in the lookup table to the n number of match processors in ESM filter by the following equation

$$Overhead\ Ratio = \frac{2n^2t}{nT} = \frac{2nt}{1126} \quad (8 \leq t \leq 12)$$

where t is the number of transistors in a CAM cell, T is that of a match processor, and n is the number of match processors in ESM filter. The above ratios are figured for n=32

As we introduce the extra components to the original filter chip for fault tolerance, the newly added components become a new source of faults. The problem of fault tolerance involved in CAMs is a nontrivial research problem by itself. For conventional RAMs, the space for a full address must be ensured in order to operate without faults. CAMs are fault tolerant in that respect since it will operate correctly by a given logic as long as the logic is correct On the other hand, the more complicated memory structure of CAMs has greater problems for testability and fault tolerance. Fault tolerance techniques applicable to RAMs may not be readily transferred to CAMs because of its different memory structure compared with RAMs The problems of testability and fault tolerance in different CAMs has been studied [50], [51], and a fault tolerant CAM memory has been suggested in [52] Here we simply assume that the lookup table is fault free to stay within the scope of our work

5.2.4 Parallel operation of ESM filter ICs

The proposed single error detection scheme for ESM filter is also applicable when several ESM filter ICs operate in parallel to form ESM System MATCH outputs The radar

pulse descriptor word (PDW) typically occupy anywhere between 96 256 bits representing various input parameters (i.e., PA has 16 bits and PW also has 16 bits, etc) Since the length of *parameter data input* of ESM filter is only 16 bits, several ESM filters are necessary to operate in parallel for processing different parameters of a given radar PDW And then the respective match outputs can be joined in a wire AND configuration to form a ESM System MATCH output for each match processor as mentioned in chapter 3

For example, a radar PDW is generated based on three different parameters (PA, PW, AOA), each of 16 bits, thus 48 bits total To process the PDW, three ESM filter ICs would be required to operate in parallel Using the same example, we will explain the possibility that ESM System level MATCH code can be extracted, and used for the detection of single errors in ESM System MATCH outputs.

Under the situation described in the above example, suppose we have an ESM filter whose parameter data input is 48 bits, thus all the limit value registers and comparators have capability of 48 bits as well Then we can have one set of 32 pairs of 48 bit limit values stored in such an imaginary filter instead of having three sets of 32 pairs of 16 bit limit values, each set stored in the actual ESM filter IC The MATCH output results of the imaginary ESM filter would be the same as the wire-ANDed match outputs from three of the actual ESM filter ICs As we have extracted the match code for the actual ESM filter IC, the match code for the imaginary ESM filter can be also extracted from the 64 of 48 bit limit values Then this match code can be used to verify ESM System MATCH outputs that are obtained by operating the three ESM Filter ICs in parallel Therefore the match code can be regarded as ESM System MATCH code The size of the code is determined only by the number of match conditions, not by the length of the limit values Hence its size still remains the same (32-bit 64 words), which can be stored in the same size of the lookup table as the one suggested for ESM filter IC

For the detection of single errors in ESM System MATCH outputs, one additional lookup table can be provided for ESM Filter ICs that are operating in parallel, preferably each having its own built in lookup table

5.3 The error coverage

The effectiveness of MATCH code scheme is represented by the conditional probability that an error will occur but will not result in the system failure referred to as the error coverage Thus in considering the error coverage, two measures are actually involved the probability that an error will occur, and the probability that such an error will be detected While the latter can be predicted accurately for a given match code the former is difficult to obtain because our fault model is rather abstract to facilitate simulation at such behavioral level Nevertheless, fault simulation based on the conventional gate level stuck at fault model instead the actual model was used to provide an approximation

The error detection capability of MATCH code may differ from one match pattern to another pattern, which is determined by the assignment of limit values Since there exist many different match patterns for 32 match conditions, one particular match pattern was not sufficient to support the validity of our measure. To begin with, we approximated number of distinct match patterns by combinatorial methods Then a subset of the match patterns was randomly sampled, and its respective match codes were extracted The normality of these codes was examined to predict the probability that a random match code will detect an error provided that an error will be present At last these two measures, each obtained from the fault simulation and the analysis on random match codes, were incorporated together to compute the error coverage

In the following we will formulate the equation which links the two statistical data to obtain the error coverage Then the combinatorial study on the match pattern, and the fault simulation results will be discussed

5.3.1 Detection probability

The probability that an error will occur is defined as *the detection probability* as the

following

P_k The detection probability of a fault k is the probability that a single random test vector will detect the fault This detection probability equals the normalized cardinality of the test set for that fault with respect to the total number of possible input vectors

T Total number of test vectors, which is equal to 2^J if the number of input lines of the circuit is J

T_k Subset of test vectors of T which detect fault k. Its cardinality is denoted as $| T_k |$.

$$P_k = \frac{| T_k |}{T} \tag{5.1}$$

The computation of detection probability, P_k can be converted into a signal probability computation problem Let's suppose that we apply a random set of input vectors $(I_1, .., I_m)$ to a fault-free MP(match processor) and a faulty MP_f as shown in figure 5 17 The single bit serial output of fault-free MP and that of faulty MP_f are labeled as *fault-free sequence* and *faulty sequence* respectively Let F be the exclusive-ORed output of MP and MP_f The single bit serial output on F is called as *error sequence*. The detection probability of the fault k is the signal probability at the output F, in other words, the probability that the *error sequence* will have the logic value 1 as a result of a randomly applied input vector.

With the table 5 3, we will explain how the detection probability P_k can be computed from these output sequences In consequence, the terms P_D and $P_{\overline{D}}$ will be defined In Table 5 3, l is the logic value of the fault-free sequence Event e denotes the occurrence of an error and event ef is the occurrence of a correct (*error-free*) output response in the faulty sequence The other notations used in the table are defined as the following

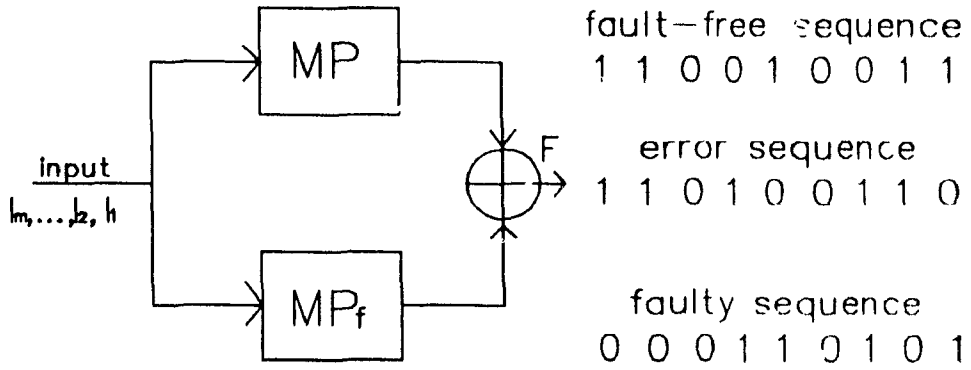


Figure 5.17 Computing the detection probability

α: The total number of *error-free* 1's in the faulty sequence

β: The total number of *erroneous* 0's in the faulty sequence †

γ: The total number of *error-free* 0's in the faulty sequence

ω: The total number of *erroneous* 1's in the faulty sequence ‡

m: The total number of bits in the faulty sequence

l	ef	e	$total$
1	α	β	$\alpha+\beta$
0	γ	ω	$\gamma+\omega$
$total$	$\alpha+\gamma$	$\beta+\omega$	m

Table 5.3 Computing the detection probability P_k

From Table 5.3, the probability of e occurring, $P(e)$ can be derived as

$$P(e) = P(l=1 \cap e) + P(l=0 \cap e) \qquad (5.2)$$

where $P(l=1 \cap e) = \frac{\beta}{m} = (\frac{\alpha+\beta}{m})(\frac{\beta}{\alpha+\beta}) = P(l=1)(\frac{\beta}{\alpha+\beta}) = P(l=1)P(e \mid l=1)$* and

† Event D 1 (*error free*) → 0 (*erroneous*)

‡ Event $\overline{D}$ 0 (*error free*) → 1 (*erroneous*)

* The term $(\frac{\beta}{\alpha+\beta})$ is the conditional probability of event e given that l is equal to 1

similarily for $P(l = 0 \cap e)$

Then the equation (5.2) can be written as the following:

$$\begin{aligned} P(e) &= P(l=1)P(e \mid l=1) + P(l=0)P(e \mid l=0) \\ &= (\frac{\alpha+\beta}{m})(\frac{\beta}{\alpha+\beta}) + (\frac{\gamma+\omega}{m})(\frac{\omega}{\gamma+\omega}) \\ &= \frac{\beta+\omega}{m} = P_k \end{aligned} \tag{5.3}$$

We may now define:

$$\begin{aligned} P_D &= P(e \mid l=1) = \frac{\beta}{\alpha+\beta} \\ P_{\overline{D}} &= P(e \mid l=0) = \frac{\omega}{\gamma+\omega} \end{aligned} \tag{5.4}$$

The presence of a 1 in the error sequence can be either due to the *error-free* bit being 1 and the *erroneous* bit being 0, or the *error-free* bit being 0 and the *erroneous* bit being 1. These two events are denoted as D & $\overline{D}$ respectively in [53], and the probability of event D, $\overline{D}$ occurring are respectively P_D and $P_{\overline{D}}$. These two terms P_D and $P_{\overline{D}}$ are essential to the error coverage measure

Finally the detection probability of fault k can be written as:

$$P_k = P_D \cdot P(l=1) + P_{\overline{D}} \cdot P(l=0) \tag{5.5}$$

where P(l=1) can be considered as the fault-free signal probability of the output.

For example we have (α, β, γ, ω, m)= (2, 3, 2, 2, 9) from the fault free sequence and the faulty sequence as given in figure 5.17. Using the above equations the values of P_D, $P_{\overline{D}}$ and P_k are obtained respectively as 0.60, 0.50 and 0.56.

5.3.2 Probability of masking

In the next step of our discussion we assume that one such MP_f (with known P_D, $P_{\overline{D}}$) is present in the filter chip which consists of n MPs. For example let n = 4 and the match code for a given match pattern * be the code as listed in figure 5.18. Let one of those four MPs be the faulty one with $P_D = 0.6$ and $P_{\overline{D}} = 0.5$. The set of binary numbers in each column of the code would be regarded as correct match code words.

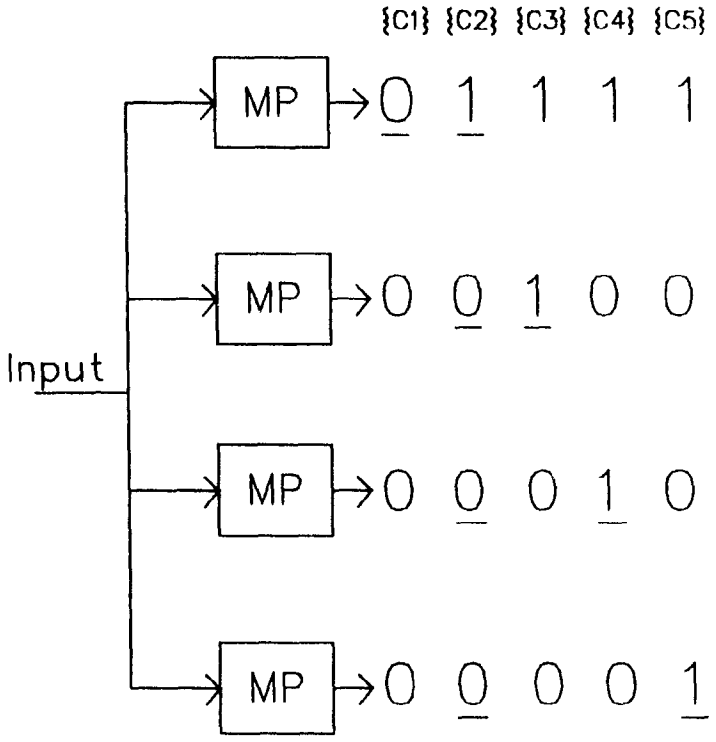


Figure 5.18 A filter chip with n=4 of match processors

For the time being, we assume that each of five code words has equal chance to become the correct output of the filter chip with a random input vector. Therefore the failure of the MP_f would randomly complement one of those 20 bit values in the code

* The match pattern can be represented with pairs of identical numbers (1 2 2 3 3 4 4 1). This notation will be explained in the following section.

The consequence of such a failure in any one of those *underlined bits* can lead to the system failure since the code would fail to detect such an error, referred to as a *critical error*. We call underlined bits *critical bits*

Referring to the table 5 4, we will explain how the *probability of masking* is formulated

P_m The *probability of masking* is the probability that an output match word will become another match code word with a random input data. Hence the error is masked up by the code word

In table 5 4, L is the correct logic value in the code, and event nC denotes the presence of a *noncritical bit* in the code and event C denotes the presence of a *critical bit* in the code The other notations used in the table are defined as the following:

a The total number of noncritical 1's in the code.

b: The total number of critical 1's in the code

c: The total number of noncritical 0's in the code

d: The total number of critical 0's in the code. *

N The total number of bits in the code.

L	nC	C	$total$
1	a	b	$a+b$
0	c	d	$c+d$
$total$	$a+c$	$b+d$	N

Table 5.4 Computing the probability of masking, P_m

From the table 5 4, we can express the probability of event C occurring, $P(C)$ as

$$P(C) = P(L=1 \cap C) + P(L=0 \cap C) \tag{5.6}$$

* Note that b is always equal to d by the property of MATCH code

where $P(L=1 \cap C)=\frac{b}{N}=(\frac{a+b}{N})(\frac{b}{a+b})=P(L=1)(\frac{b}{a+b})=P(L=1)P(C \mid L=1)$ and similarily for $P(L=0 \cap C)$.

Then the equation (5.6) can be written as the following

$$\begin{aligned} P(C) &= P(L=1)P(C \mid L=1) + P(L=0)P(C \mid L=0) \\ &= (\frac{a+b}{N})(\frac{b}{a+b}) + (\frac{c+d}{N})(\frac{d}{c+d}) \\ &= \frac{b+d}{N} \end{aligned} \tag{5.7}$$

We may now define:

t Total number of distinct words in the code

W_i: ith word of the code. $(i=1,..,t)$

$cb_i^{1(0)}$: Total number of critical 1(0) bits in W_i of the code.

$b_i^{1(0)}$: Total number of 1(0) bits in W_i of the code.

$$\begin{aligned} P_{10} &= P(C \mid L=1) = (\frac{b}{a+b}) = \frac{\sum_{i=1}^{t} cb_i^1}{\sum_{i=1}^{t} b_i^1} \\ P_{01} &= P(C \mid L=0) = (\frac{d}{c+d}) = \frac{\sum_{i=1}^{t} cb_i^0}{\sum_{i=1}^{t} b_i^0} \\ P(L=1) &= (\frac{a+b}{N}) = \frac{\sum_{i=1}^{t} b_i^1}{\sum_{i=1}^{t}(b_i^0+b_i^1)}, \quad (t \quad \frac{N}{n} \quad 2n) \end{aligned} \tag{5.8}$$

For a critical error to occur or equivalently for an error to be undetected, two independent events must happen simultaneously: an error occurs, and it occurs in one of the critical bits of the code. Finally the *probability of masking* is expressed as the following, where P_D & $P_{\overline{D}}$ incorporate:

$$
\begin{aligned}
P_m &= P(L=1 \text{ and } e \cap C') + P(L=0 \text{ and } e \cap C) \\
&= P(L=1)P(e \mid L=1)P(C \mid L=1) + P(L=0)P(e \mid L=0)P(C \mid L=0) \\
P_m &= P(L=1) \cdot P_D \cdot P_{10} + P(L=0) \cdot P_{\overline{D}} \cdot P_{01} \\
&\textit{further simplified as} \\
&= P_D(\frac{b}{N}) + P_{\overline{D}}(\frac{d}{N}) \\
&= \frac{b}{N}(P_D + P_{\overline{D}}) \quad (\textit{since } b = d)
\end{aligned}
\tag{5 9}
$$

For example we have (a, b, c, d, N)=(3, 4, 9, 4, 20) from the code as shown in figure 5 18, and the *probability of masking* for the code can be calculated as $P_m = \frac{1}{5}(0.6 + 0.5) = 22\%$.

The equation (5 9) is not strictly valid unless we keep the previous assumption each code word has equal probability to become the correct output of the filter with a random input vector This assumption is same as to require that *(a) all code regions of the code have the same sizes*, and that *(b) all inputs are uniformly distributed* Since the sizes of code regions may vary with arbitrary limit values, the equations (5 8) need to be modified to remove the restriction (a), but the restriction (b) is still required at this point

P_i The probability of the input set i is the probability that a single random input vector will be in the set $\{C_i\}$ when random selection of the input is evenly distributed

By the definition P_i is the measure of how often a match code word W_i will become the correct output response of the Filter with a random input vector Assuming that all inputs are uniformly distributed, P_i is equal to the normalized cardinality of the set $\{C_i\}$ with respect to the entire input space of the Filter $|C_i|$ is determined based on the given set of 32 pairs of 16 bit limit values for $(i = 1, \ldots, t)$ Then P_i can be expressed as the

following

$$P_i = \frac{|C_i|}{2^{16}}$$

Accordingly any critical bits of the code words with higher P_i would have the greater affect on increasing the probability of masking. Therefore P_{10}, P_{01}, $P(L)$ need to include P_i in their expressions as the following

$$P_{10}^{weighted} = \frac{\sum_{i=1}^{t} P_i \cdot cb_i^1}{\sum_{i=1}^{t} P_i \cdot b_i^1} \quad ; \quad P_{01}^{weighted} = \frac{\sum_{i=1}^{t} P_i \cdot cb_i^0}{\sum_{i=1}^{t} P_i \cdot b^0}$$

$$P(L=1)^{weighted} = \frac{\sum_{i=1}^{t} P_i \cdot b_i^1}{\sum_{i=1}^{t} P_i \cdot (b_i^0 + b_i^1)} \quad , (t \quad n \quad 2n) \tag{5.10}$$

Then the equation (5.9) can be modified as

$$P_m = P_D \cdot P_{10}^w \cdot P^w(L=1) + P_{\overline{D}} \cdot P_{01}^w \cdot P^w(L=0)$$

$$= P_D \cdot \frac{\sum_{i=1}^{t} P_i \cdot cb_i^1}{\sum_{i=1}^{t} P_i \cdot (b_i^0 + b_i^1)} + P_{\overline{D}} \cdot \frac{\sum_{i=1}^{t} P_i \cdot cb_i^0}{\sum_{i=1}^{t} P_i \cdot (b_i^0 + b_i^1)} \tag{5.11}$$

$$= \frac{\sum_{i=1}^{t} P_i \cdot cb_i^1}{\sum_{i=1}^{t} P_i \cdot (b_i^0 + b_i^1)} \cdot (P_D + P_{\overline{D}}) \quad (since \sum_{i=1}^{t} cb_i^1 = \sum_{i=1}^{t} cb_i^0)$$

We can further generalize the above equations by removing the second restriction (b) as well. If the distribution of the parameter data input over long period of time can be found, then P_i can be modified as P_i' to readjust the weight of each code word W_i. P_i' is expressed as

$$P_i' = P_i \cdot \left(\frac{Number\ of\ applied\ input\ data\ whose\ correct\ output\ word\ is\ W_i}{Total\ number\ of\ applied\ input\ data} \right)$$

However this type of information may not be readily available because of its classified nature. For this reason we maintain the same simplifying assumptions (a) and (b) to measure the mean value of the probability of masking $\overline{P}_m$ in subsequent sections.

5.3 3 Combinatorial analysis on match patterns

In this section we consider how many different match patterns actually exist for n match conditions As we have shown in section 5 2 1 figure 5 7, various match patterns can be formed for the given n match conditions Each unique match pattern corresponds to a distinct match code and each match code has its own value of P_m First we observe how the number of distinct match patterns grows as n increases Next we generate random samples of distinct match graphs so as to compute the average $\overline{P}_m$ of the corresponding set of match codes for $n = 32$ case

We can describe the enumeration of match graph problem as the following Suppose that $2n$ points are arranged on the circumference of a circle We can pair up these points and join corresponding points by chords of the circle The graph formed in such manner is called as a *match graph* To represent match graphs, we can use pairs of duplicated numbers to label each ends of the chords, thus each pair representing a certain chord of the circle For example we use three pairs of duplicated numbers $\{1, 1\}, \{2, 2\}, \{3, 3\}$ to label match graphs as illustrated in figure 5 19 for $n = 3$ case

These five graphs are distinct (or *non isomorphic*) match patterns since one can not be obtained from any other graphs by a plane rotation or reflection of the graph Here the directions of the chords are not considered, unlike the way it was treated in figure 5 7

By ignoring directions of the chords, it actually reduces the number of distinct match graphs that we have to count For example, figure 5 20 shows two undirected match graphs, where graph (a) can represent both directed graphs (a) and (c) of figure 5 21 Hence we have only two distinct match graphs instead of three if the directions are not concerned

Since directions of the chords can be easily altered by the "NOT" control option of ESM filter, it is omitted in the consideration of distinct match graphs Therefore the

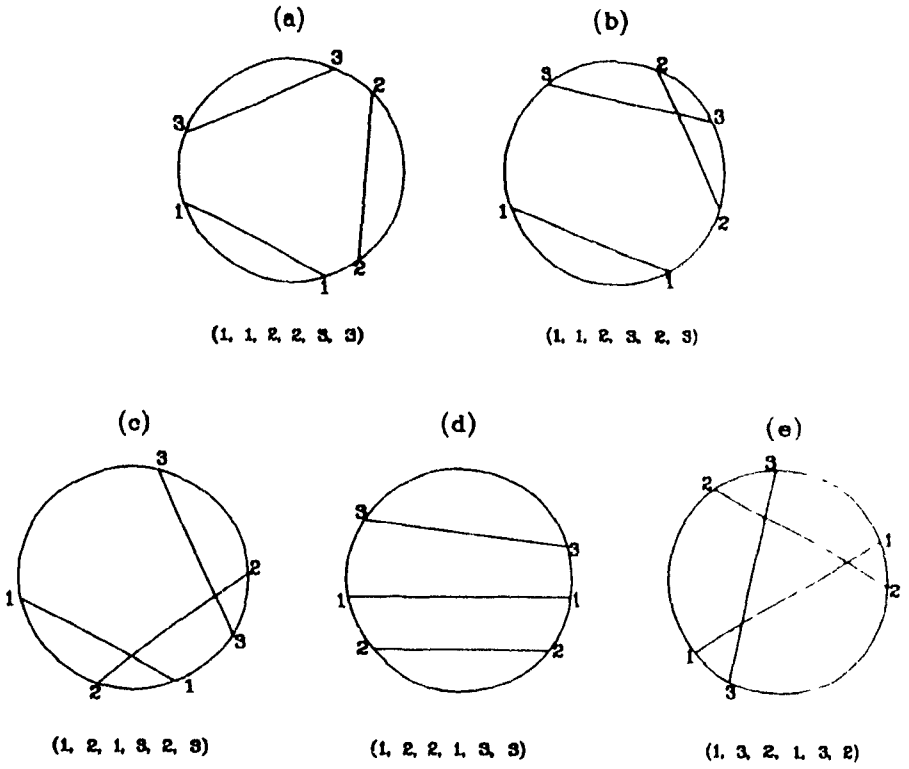


Figure 5.19 Labeled match graphs for $n = 3$

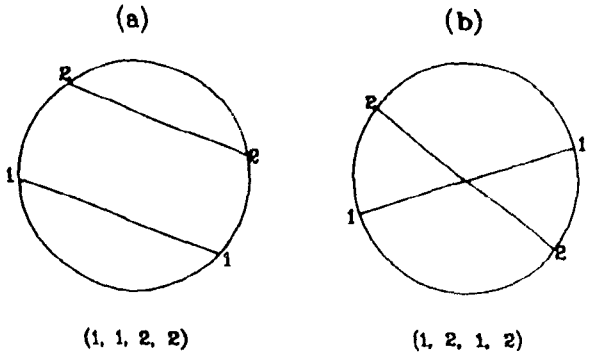


Figure 5.20 Labeled match graphs for $n = 2$

number of unique match graphs for n is the number of ways of doing this pairing so that none of the graphs are isomorphic * regardless of the chords' directions

---

* Two graphs G_1 and G_2 are isomorphic if there is a one one correspondence between the vertices of G_1 and those of G_2 with the property that the number of edges joining any two vertices of G_1 is equal to the number of edges joining the corresponding vertices of G_2 [54]

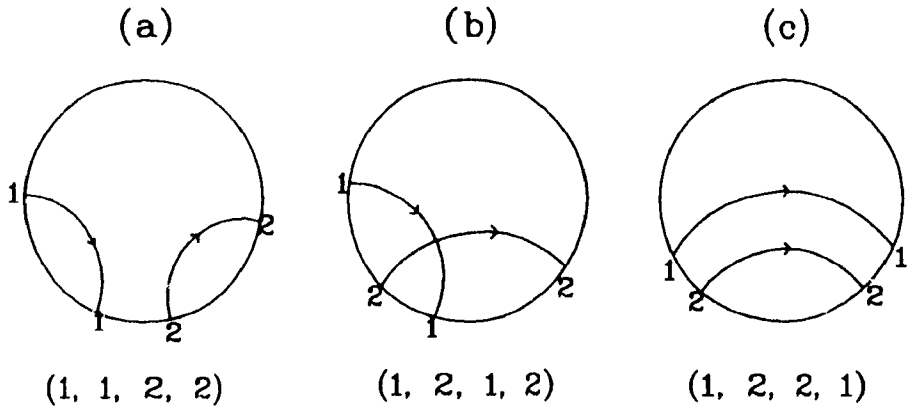


Figure 5.21 Match graphs with directed chords for $n = 2$

Similar problem was considered in [55], [56], where none of the chords were allowed to cross, and the number of ways of pairing the $2n$ points was given by a Catalan number as shown below

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Since the crossing of the chords are allowed in our problem, the number of ways will be certainly larger than a Catalan number

These types of counting problems are often very complex because apparently different objects often turn out to be equal, or we say, isomorphic All objects which are considered equal are placed in a single class called an *equivalence class* and it is the number of such classes which is of our interest

In figure 5 19, we have shown only the distinct graphs out of 16 labeled match graphs for n 3 case The total number of labeled match graphs equals the number of ways of labeling around a circle using n pairs, each pair having duplicated label numbers The total number of cyclic permutations for n pairs of duplicated elements can be expressed [58] as

$$L = \left[\frac{(2n)!}{2^n} \quad n! \right] \frac{1}{n} + (n \quad 1)!$$

The next example displays how the equation works out The number of cyclic permutations of two pairs of duplicated elements is counted by the following steps

- The number of permutations with four elements in a row $\frac{4!}{2!2!} = 6$ For n pairs of duplicated elements we have $\frac{(2n)!}{2^n}$ of permutations in a row

- The number of symmetrical permutations in a row: 2! It forms symmetries by 180° when total number of elements is even Figure 5.22 shows the formation of symmetries in cyclic permutations

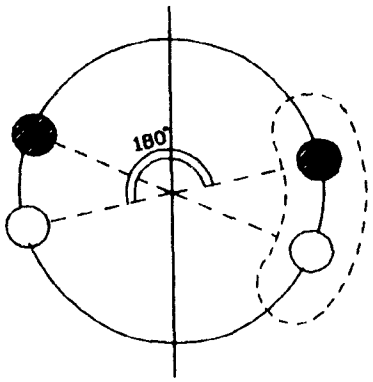


Figure 5.22 Symmetries in cyclic permutations

For n pairs of duplicated elements we have $n!$ symmetrical permutations Since such symmetrical permutations belong to a equivalence class we have to exclude accordingly from the total number of permutations in a row $(6 \quad 2!) = 4$ In general we have $\frac{(2n)!}{2^n} \quad n!$ for n pairs of duplicated elements

- Finally the number of cyclic permutations for two pairs is given as

$$\frac{(6 \quad 2!)}{4\ (total\ number\ of\ elements)} + \frac{2!}{2\ (cyclic\ period\ of\ symmetry)} \quad ?$$

where the cyclic period of symmetry equals the number of elements inside dotted line of the figure 5.22

There are many situations in counting of graphs when graphs are to be counted only as a single distinct graph if they are equivalent under some specified operations (i.e. rotation, reflection) The relation between the total number of labeled graphs and the

number of distinct graphs depends closely upon the structure of the equivalence group, and may be obtained by applying *Polya*'s theorem, often called fundamental theorem in enumerative combinatorial analysis [55], [57], [58]. The complete solution to the number of distinct match graph problem remains unsolved here, but may be solved by applying *Polya*'s theorem

We have generated exhaustively all the vectors representing labeled match graphs from n = 2 to 6, and counted only the distinct match graphs for each set of labeled match graphs The results are as shown in table 5 5.

n	# of labeled graphs	# of distinct graphs
2	2	2
3	16	5
4	318	16
5	11,352	58
6	623,760	206

Table 5.5 Number of distinct match graphs for n match conditions

The table shows almost exponential increases in the total number of distinct match graphs as n increases Therefore it is practically impossible to survey all of these cases for obtaining the mean value of P_m with $n = 32$

Nevertheless we can show how values of P_m are distributed for complete set of distinct match graphs when n is small (i.e $n \leq 6$) From that we may be able to predict the distribution of P_m values for larger size like $n = 32$ Then a random set of match codes can be generated for random sampling The average value $\overline{P}_m$ of the random sample, denoted as $\overline{X}$, can be used to predict the true $\overline{P}_m$ of the complete set of distinct match graphs for $n = 32$

For n = 2 to 6, we have examined all match codes for every distinct match graph

to compute its respective P_m values. The upper bound of P_m values, denoted as $\lceil P_m$, are computed by the equation (5.9) where P_D and $P_{\overline{D}}$ both are set equal to one * in absence of fault simulation result.

For example we have $\lceil P_m$ = 0.50, 0.67, 0.50, 0.78, 0.67 respectively for match codes (a), ..(e) of the table 5.2 in section 5.1.2. We note that both pairs (a), (c) and (b), (e) have same $\lceil P_m$ values even though their match patterns are all distinct.

The plots of figure 5.23 shows the relationship between the value of $\lceil P_m$ and the number of corresponding match codes for each case of n. In table 5.6 the average value of $\lceil P_m$ and standard deviation are shown for a given n.

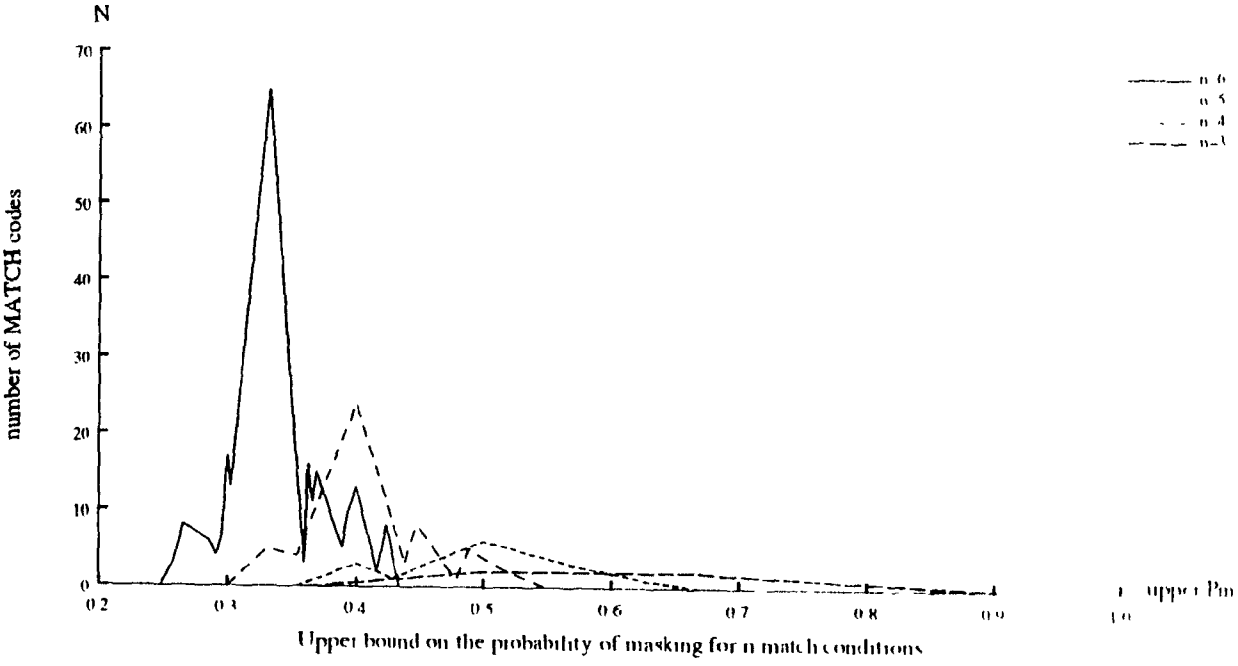


Figure 5.23 Distribution of $\lceil P_m$ values for n match conditions.

As we can see from these curves, the mean value of $\lceil P_m$ becomes smaller and the curve becomes sharper for larger n. The distribution of $\lceil P_m$ values do not radically depart

* It is same as to say that output of the faulty processor always have the complementary logic value to the intended logic value. This is only a hypothetical situation to give the upper bound on P_m.

n	Data size N	$\overline{P}$m (%)	Standard deviation σ
2	2	83.34	0 1667
3	5	62.22	0.1077
4	16	48.76	0 0696
5	58	41.79	0.0460
6	206	34.16	0.0405

Table 5.6 $|\overline{P}_m$ of match codes for n

from the normal distribution. There is a bell-shaped density that is nearly symmetric. The curves of figures 5 24 and 5.25 show the distribution of $|P_m$ approximated by the normal distribution curves, $f(x)$ * for $n = 5$ and 6 respectively This pattern of well-behaved normal distribution is also expected for larger n

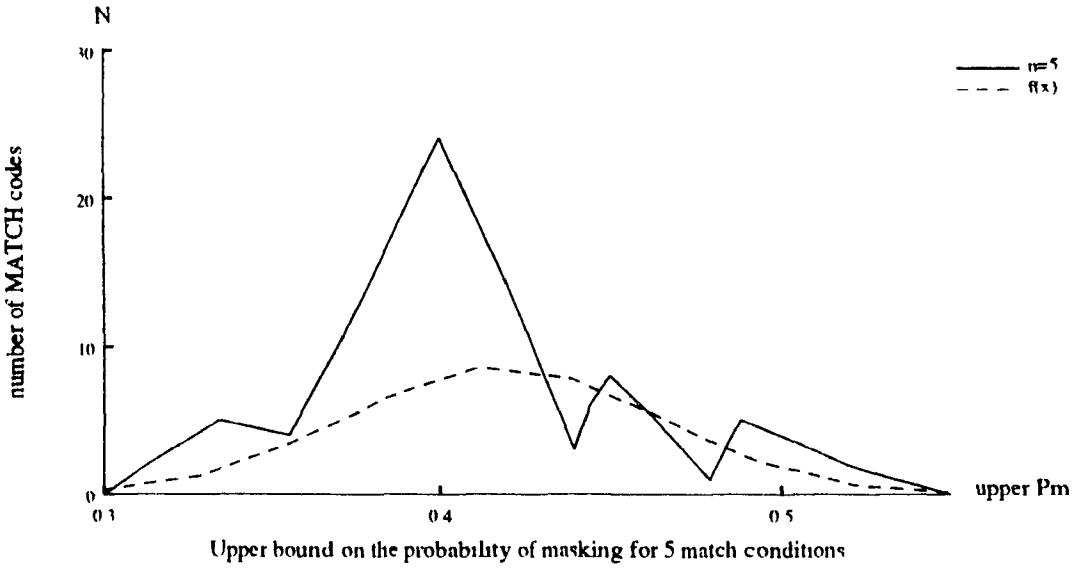


Figure 5.24 Normal distribution curve for $n = 5$

Assuming that the distribution of $|P_m$ follows the normal distribution for $n = 32$, we have generated random samples of distinct match graphs There are two methods of

* The density function of the normal distribution $f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-(x \quad m)^2/2\sigma^2)$, where m is $|\overline{P}_m$ and r is $|P_m$ in our case

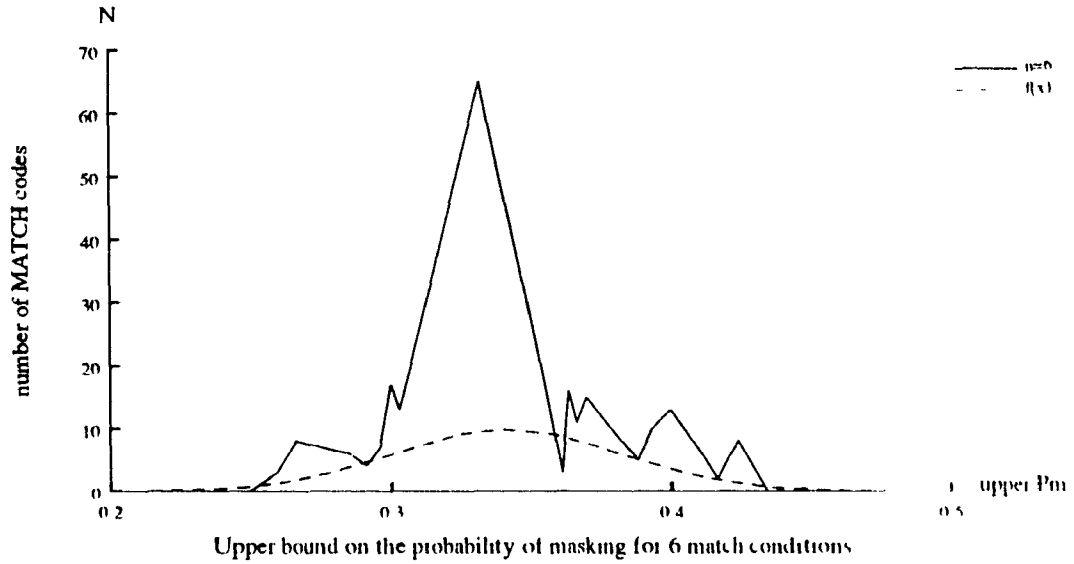


Figure 5.25 Normal distribution curve for $n = 6$

choosing random samples from the population (total number of distinct match graphs of $n = 32$)· (1) choose a random sample and replace it back to the population before choosing the next random sample. (2) choose a random sample one after another without replacement Since the size of the population is very large or practically infinite the statistics has the same distribution whether we sample with or without replacement In this study, we used the second method of sampling We have generated number of match graphs larger than the target sample number so that our target number was satisfied when the set was reduced to a set of distinct match graphs

According to the *Central Limit theorem* [59], even if the distribution of the population is unknown, either finite or infinite, the sampling distribution of $\overline{X}$ will still be approximately normal with mean value $\overline{P}_m$ and variance σ^2/N provided that the sample size N is large. The question immediately encountered in practice is how large N must be to get the normal approximation for $\overline{X}$ This is not an easy question to answer since it depends on the characteristics of the distribution of the population, which is not clearly known From a practical standpoint, some very crude rules of thumb suggest that size

N greater or equal to 100 should be satisfactory even if the population has no prominent mode of distribution [60] On the other hand, if the population is normally distributed for $n = 32$, as we have assumed, the distribution of the sample should also follow the normal distribution, no matter how small the size of the sample. Considering all these facts, we have chosen arbitrarily large number of sample 500 to ensure the credibility of our measure by double standard

From 500 distinct match graphs, match codes were extracted to compute $\lceil \overline{P}_m$ of the sample, which is denoted as $\lceil \overline{X}$, and the standard deviation of the sample s. Table 5 7 shows the final result of the computation.

n	Sample size N	Average of the sample $\lceil \overline{X}$ (%)	Max($\lceil X$) (%)	Standard deviation s
32	500	5.85	6.77	0.0032

Table 5.7 $\lceil \overline{X}$ of random match codes for $n = 32$

A 95% two-sided confidence interval on the mean value of $\lceil P_m$ is given as.

$$\lceil \overline{X} - 1\,96\frac{\sigma}{\sqrt{N}} \leq \lceil \overline{P}_m \leq \lceil \overline{X} - 1.96\frac{\sigma}{\sqrt{N}}$$

where s value substitutes for σ Therefore we are 95% confident that $\lceil \overline{P}_m$ is in between 5 82% and 5 88% for $n = 32$ In other words if we choose a random match pattern for $n = 32$, we can guess that $\lceil P_m$ for its match code lies between 5 82% and 5 88% with 5% chance of being wrong. For the worst case in a set of random sample, we have $\lceil P_m = 6\,77\%$

5.3.4 Fault simulation

The fault simulation is conducted using Tulip [61] so as to obtain two measures of the equation (5 9)· $(P_D + P_{\overline{D}})$, which was previously set equal to two Tulip is a multi-

option gate level fault simulator based on the single stuck at fault model for combinational circuits

The quality of our measure is limited by the fact that fault simulation is done only for the combinational part of the circuit based on the single stuck at fault model The fault model actually used in the fault simulation is not consistent with the fault model we have assumed from the beginning The combinational part of the processor comprises a pair of comparators (upper/lower) and a compare function module as shown in figure 3 1 These components take about 63% of the total number of transistors used in the processor The rest of sequential part of the processor remains uncounted in the simulation

With 5568 random test vectors, 505 faults in *collapsed fault set* † are fully simulated (no fault dropping) In other words, when a fault is first detected, it is not removed from the fault list, but accumulates the number of times that the random test vectors detected the particular fault 432 faults were detected out of 505 faults for a fault coverage of 85 5% ‡ From this simulation result P_D and $P_{\overline{D}}$ are computed by the equation (5 4) of section 5 3.1.

$$P_D = 3.09\%, \quad P_{\overline{D}} = 0.88\%, \quad \overline{P}_m = 0.0582 \sim 0\ 0588$$

Finally the probability of masking incorporated with these values is given as

$$\overline{P}_m = (\frac{0.0582}{2} \sim \frac{0.0588}{2})(P_D + P_{\overline{D}}) \quad 0\ 12\%$$

The limiting factors on the accuracy and the key assumptions of our measure are summarized as below·

---

† For any two faults whose effect is identical at the output of the circuit called *equivalent* faults only one representative fault from each set of equivalent faults is selected to form the collapsed fault set

‡ Any longer random test length than the given was not feasible with the available memory space to achieve the higher fault coverage

- Only the combinational part of the circuit is simulated based on the single stuck at fault model

- The fault coverage of the simulation is 85.5%.

- $\lceil \overline{P}_m$ is approximated by $\lceil \overline{X}$, which is $\lceil \overline{P}_m$ of the sample, instead of the complete set We have assumed that the distribution of $\lceil P_m$ follows the normal distribution. A 95% two-sided confidence interval on the value of $\lceil \overline{P}_m$ is used.

We also have assumed the followings

- All code regions of the code have equal sizes.

- All valid inputs to ESM filter are uniformly distributed.

- No identical limit values exist for a given set of match conditions.

In summary, we have shown that the proposed concurrent error detection scheme can detect 99 88% of all single errors on the average according to our fault simulation result. The lower bound on the error coverage is 94.12% as we set $(P_D + P_{\overline{D}})$ equal to two The inaccuracy involved in fault simulation has no significance for the lower bound measure of the error coverage Therefore, even if an error occurs in every operation cycle, 94.12% of single errors can be detected

Chapter 6 Recomputation with Rotated Comparands

In this chapter, we suggest another approach using time redundancy based on the same fault and error model as used before The basic idea is to recompute the match outputs with rotated limit values by applying the same principle of RESO [16] described in chapter 2 The same set of lower/upper limit values are assigned to different match processors so that no match processors hold the same pair of limit values in the recomputation step It allows each match processor to test different ranges of limit values, yet keeping the same configuration of match pattern as was in the initial place Then the second match word $MW1'$ is modified by shifting a bit, and compared with the first match word $MW1$ stored in a register Any inconsistency in the comparison indicates the presence of an error or errors The possibility of multiple error detection and single error correction are also discussed.

6.1 Single error detection

In figure 6 1, we describe the recomputation with rotated limit values using the array of n match processors One faulty match processor MP_i $(1 \quad i \quad n)$ is present among n number of MPs For the first step the set of MP_i $(i = 1, \ldots, n)$ compute the match outputs $\langle b_{1,1}, b_{1,2}, \ldots, b_{1,n} \rangle$ according to the initial assignment of lower/upper limit

values. Due to one faulty MP_i, one of those match outputs may or may not be correct. In this case, the output of MP_i, $b_{1,i}$ is potentially incorrect.

In the recomputation step, the pair of limit values are circularly shifted as the following

$$[X_i, Y_i] \rightarrow [X_{(i+1) \bmod n}, Y_{(i+1) \bmod n}] \quad (1 \leq i \leq n)$$

then each MP produces the match outputs $b_{2,1}, b_{2,2}, \ldots, b_{2,n}$. This time each MP tests for the ranges of limit values that were examined previously by the other adjacent MP positioned one below.

For example, initially the MP_3 tests for the range $[X_3, Y_3]$ to output $b_{1,3}$. In the second step the range $[X_3, Y_3]$ is assigned to the MP_2, which computes for the match output $b_{2,2}$. Therefore the match output for the range $[X_3, Y_3]$ are double checked by both MPs. If both MPs are fault free then their respective match outputs $b_{1,3}$ and $b_{2,2}$ for MP_3 and MP_2 should have the same logic values[†]

Likewise all the other match outputs from each match word $MW1$, $MW1'$ are compared as expressed

$$b_{1,i} = b_{2,i-1} \quad for\ 1 < i \leq n$$
$$b_{1,1} = b_{2,n} \quad for\ i = 1$$

where $b_{i,j}$ is the output of MP_j for ith computation.

Such comparison results can be read as the exclusive-ORed form of the two match words ($MW1'$ is right shifted by one bit).

$$\langle b_{1,1}, b_{1,2}, \ldots, b_{1,t}, b_{1,t+1}, \ldots, b_{1,n} \rangle \oplus \langle b_{2,n}, b_{2,1}, \ldots, b_{2,t-1}, b_{2,t}, \ldots, b_{2,n-1} \rangle =$$
$$\langle e_{1,1}, e_{1,2}, \ldots, e_{1,t}, e_{1,t+1}, \ldots, e_{1,n} \rangle$$

[†] The reversed argument is not true, but only ensures that both bits are error free, not fault free.

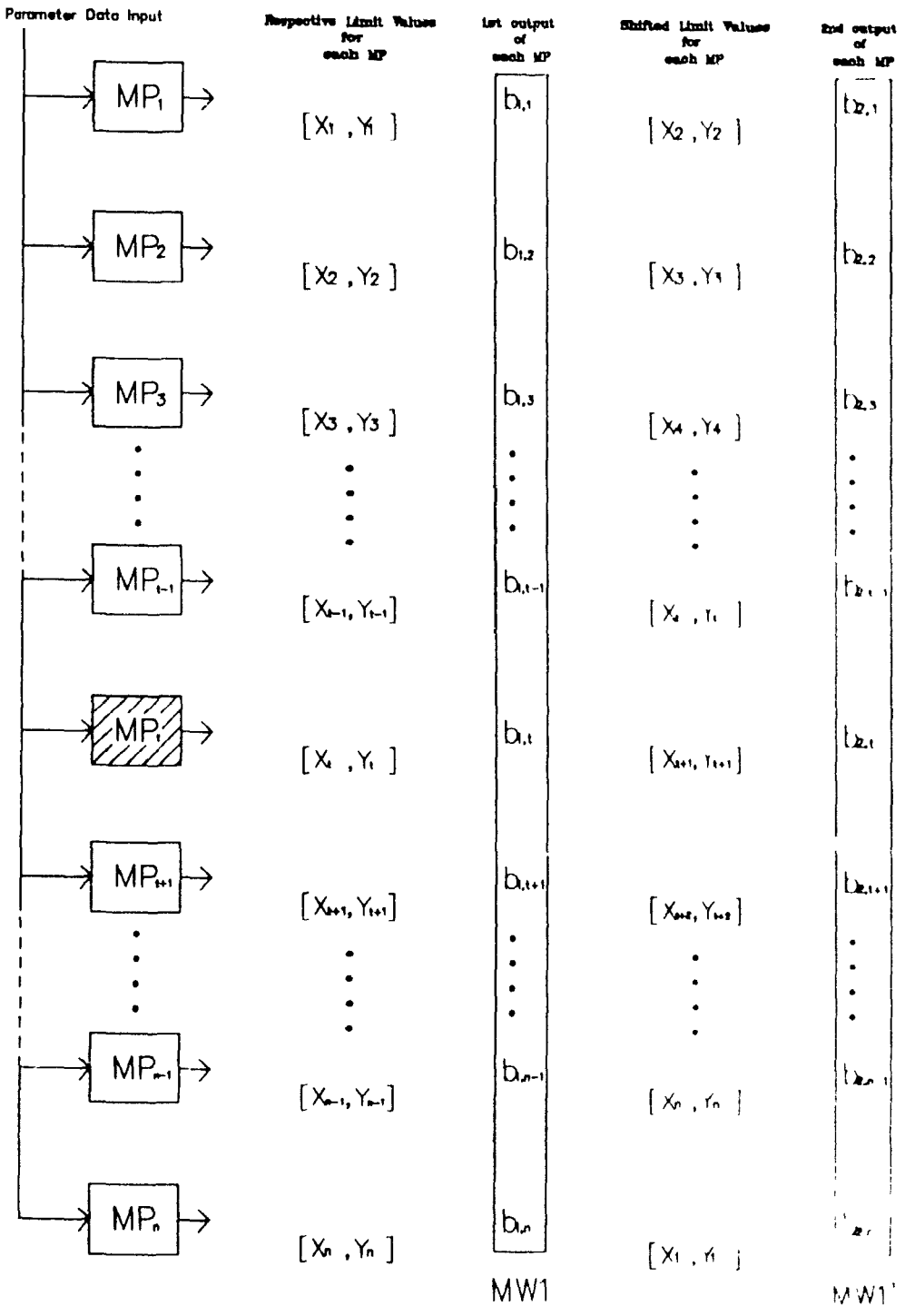


Figure 6.1 Recomputation of a match word with rotated limit value

where $e_{i,j}$ is the comparison result between $b_{i,j}$ and $b_{i+1,j-1}$ $(1 \leq j \leq n$ $b_{i,1}$ $b_{i+1,n}$ *for* $i = 1$) for ith comaprision This comparison result vector is called as an *error*

vector $\bar{e}_1$

If no inconsistent bits are found in the comparison then there is no error If the weight of the error vector is not equal to zero then it indicates the presence of at least one error Under the assumption of single error, the weight of an error vector $W(\bar{e}_1)$ must be less than or equal to 2

If the fault in the faulty processor MP_f is a permanent fault and is insensitive to the alterations of the lower/upper limit values, then the MP_f produces either correct outputs for both $b_{1,t}$ and $b_{2,t}$ or incorrect outputs for both computation results Thus either two pairs of inconsistent bits or no inconsistency would be found in the comparison of the two match words For the former case it can be shown that

$$b_{1,t} = b_{2,t-1} \quad and \quad b_{1,t+1} \neq b_{2,t}$$

If the fault is a non permanent fault (i e , intermittent or transient fault) [†] then the faulty MP_f behaves unstably producing an error during the first computation step (a) or second (b) or both steps (c) It may also produce no errors at all These three cases occur when

$$b_{1,t} = b_{2,t-1} \quad (a)$$

$$b_{1,t+1} \neq b_{2,t} \quad (b)$$

$$b_{1,t} = b_{2,t-1} \quad and \quad b_{1,t+1} \neq b_{2,t} \quad (c)$$

One question remained is in whether transient fault would always affect the same MP_f throughout the both computation steps Since the cause of a transient fault lies on external disturbances, the affected areas of the circuit may change time to time Thus the location of faulty MP_f may vary with time Therefore, for this method of error detection as well as for RESO we need another assumption to ensure the detection of transient faults.

[†] In the real field these type of faults are actually dominant cause of errors in a system operation as pointed out in [1]

The assumption is that the time interval between the first and the second computations is so small that there is no variation of external physical conditions. Hence the location of the faulty MP_i due to transient faults does not change during the period of recomputation time.

In summary there exist three patterns of error vectors for indication of an error:

$$e_{1,1}, \ldots e_{1,t}, e_{1,t+1}, \ldots e_{1,n} = 0\ 10\ 0 \qquad (a)$$

$$e_{1,1}, \ldots e_{1,t}, e_{1,t+1}, \ldots e_{1,n} = 0\ 01\ 0 \qquad (b)$$

$$e_{1,1}, \ldots e_{1,t}, e_{1,t+1}, \ldots e_{1,n} = 0\ 11\ 0 \qquad (c)$$

For the two pairs of inconsistent bits, the two 1s should appear next to each other as shown in (c). Appearance of any pattern other than these would violate our single error assumption. Nevertheless if any such case exists, then it will mean that more than one error are present.

6.2 Multiple error detection

We have demonstrated that recomputation with rotated comparands (RERC) technique can produce an error vector $\bar{e}_1$ leading to detection of all single errors. The natural question that follows is whether the same method is capable of detecting t errors ($t \geq 2$).

We say that t errors are detectable if all errors in the circuit consisting of n MP can be detected provided that the number of faulty MPs does not exceed t.

In the next proposition we provide sufficient and necessary condition to ensure t error detectability.

Definition. The minimum distance $min \mid MP_i, MP_j \mid$ is the minimum number of

movements of MP_i to meet MP_j on the circle by any directions

$$min\,|\,MP_i,\ MP_j\,|\ =\ min(\,|\,i-j\,|,\ n-|\,i-j\,|\,)\ \ for\ \ (1\quad i,j\ \cdot\ n)$$

Proposition 6.2 If and only if any pair of MP_i, MP_j $(1 \cdot i,j \cdot n)$ from t-faulty set of MP's always has $min\,|\,MP_i,\ MP_j\,| > 2$ then t errors can be always detected.

Proof Since no two faulty MP's are next to each other, none of limit values in a given set is repeatedly assigned twice to faulty MP's Thus each match output is correctly computed at least once If there are any incorrect outputs then it always has its complementary output Therefore logic value of 1 bits would appear in an error vector to signal the presence of any error

Let's suppose that t errors can be always detected even if two faulty MP's are adjacent Then both logic values computed from the first and the second step may have the same incorrect logic values since faulty MP's could be used repeatedly to compute both results Therefore, errors can not be detected in such cases unless the minimum distance between two faulty MP's is greater than one Q E D

The most tight arrangement satisfying the condition is to put each of t faulty MP's one next to each of t-good MP's alternately Thus the minimum number of MP's is $2t$ $(n \quad 2t)$

The capability of multiple t-error detection is uncertain as any two faulty MP's from subset t of n can get clustered together Such measure of uncertainty we call P_{tn} and it can be computed by solving cyclic permutation problems We assume that faulty MP's are randomly distributed among n distinct MP's arranged on a circle.

P_{tn} is the probability that at least one adjacent pair will be formed from the subset t out of n when n distinct objects are randomly arranged on a circle

$$P_{tn} = \frac{\textit{Total number of ways to put } n \textit{ objects such that at least one adjacent pair is from subset } t}{\textit{Total numer of ways to put } n \textit{ objects}}$$

In cyclic permutation, two arrangements are considered equal if one can be obtained from the other by a rotation. Hence the cyclic permutation of n distinct objects can be considered as permutation of $(n-1)$ objects after placing one of n objects at a fixed point of a circle. Thus the cyclic permutation of n objects is $(n-1)!$

The next question is in how many ways n distince objects can be arranged on a circle such that no two objects from the subset t are not adjacent. First we can think of different ways of arranging $(n-t)$ objects on a circle (a), then the ways of arranging t objects between those $(n-t)$ objects (b). The first term (a) is $[(n-t)-1]!$ and the second term (b) is ${}_{(n-t)}P_t$. Finally combining the both terms $(n-t-1)!{}_{(n-t)}P_t$ leads to the final solution

$$P_{tn} = \frac{(n-1)! - (n-t-1)!{}_{(n-t)}P_t}{(n-1)!}$$

We consider an example of double error detection with $n = 32$

$$P_{tn} = \left.\frac{(n-1)! - (n-t-1)!{}_{(n-t)}P_t}{(n-1)!}\right|_{t=2,\, n=32}$$

$$= \frac{31! - 29!(30 \cdot 29)}{31!} \approx 6.5\%$$

As we have done on the analysis of the error coverage in chapter 5, P_{tn} can also be incorporated with the detection probability P_k to predict the probability of not detecting t-errors on the output of the circuit.

P_{tm} The probability of t-error masking is the conditional probability that t errors will occur and will result in an incorrect code word with a random input vector

Thus for the above example with $P_k = 0.013$ gives the following result:

$$P_{2m} = P_{tm}|_{t=2,n=32} \cdot (P_k)^2 = 0.0011\%$$

The method of error detection presented in this chapter can be extended for error correction For single error correction a third computation with another rotation can be done Now each bit of the result is computed by at least two fault-free MP's Therefore 2-out of-3 majority votes on each bit will decide the correct value The execution time will be tripled in this case

The technique of error detection reduces the execution rate in half, which may not be acceptable for real time process However, the matching function of the ESM Filter is also performed for non real time, near-real time applications such as ELINT, RSR systems mentioned in chapter 3

The extra hardware cost includes a 32-bit register for storing the first match word, a comparator for comparing the two match words and control hardware needed for rotating the limit values (i e , extra bus lines, buffers) For the equality comparison of the two match words we can also use a parallel 32-bit word CAM cell instead of using a 32 bit register with a comparator

Chapter 7 Conclusions

The main objective of the work in this thesis was to design a concurrent error detection scheme, especially applicable to ESM Filter IC. Before designing such a scheme appropriate fault and error model had to be defined. Then the problem was formulated concisely based on graphical description of the functionality of the Filter. MATCH code technique was suggested for concurrent single error detection using redundant hardwares which satisfied the time constraint for real time application. Hardware implementation of the scheme was considered in detail for various options and their trade offs. CAM (content addressable memory) architecture was suggested because of its speed and high density. We also showed the possibility of extending MACH code technique in the parallel operation of ESM Filter ICs.

To evaluate the effectiveness of the error detection method, probabilistic analysis was done to obtain the error coverage. In consi ring the probability of masking, we recognized that size of code regions may vary from one another and input data are unevenly distributed in reality. The expressions for the probability of masking were driven based on such assumptions as well as based on simplified assumptions. The simplification was done by assuming that size of code regions are all equal and input data are evenly distributed.

We came up with a combinatorial problem: enumeration of distinct match graphs. Although its complete solution was not sought in this paper, the problem was precisely

formulated and clues were suggested for further mathematical elaboration.

The error coverage was 99 88% based on the analysis of random match codes and the fault simulation Considering inaccuracies involved in the fault simulation, the lower bound of the error coverage was 94 12% The estimated hardware overhead was approximately 46% - 69% for MATCH code scheme

The advantage of the higher-level fault model, such as the one we have used, is its simplicity However, it is difficult to predict accurately how such faults will manifest as errors by simulating the faults In many recent works, fault models at this level of abstraction have been used in designing fault-tolerant systems [16]-[19], [37] However, their approaches do not require fault simulation for failure-rate predictions because the detection of all possible single errors are ensured [16]-[18]. For our case the error coverage is not solely dependent upon the error detection capability of MATCH code, but it also depends on the probability that the fault will become an error on the output of a match processor Thus the probability of fault detection was desirable to measure accurately the error coverage of our error detection scheme, which in turn required fault simulation However, we did not actually have means to simulate such abstract level of faults, but only simulated based on the gate-level stuck at fault model Nevertheless, the uncertainty in our fault simulation result can not afflict the accuracy of the error coverage measure by more than 5 76%

It is possible to treat the more complex fault model by dividing the processor into smaller sub functional blocks (i e., register, comparator) by including the list of their fault model descriptions as the way shown in [27], [36] But it would also require to devise a tool that can simulate sequential parts of the processor as well as combinational parts based on the formulated fault models The result obtained from such effort would become obsolete if the circuit is to be implemented in a different way

In this view, our approach for MATCH code scheme was reasonable since our main target errors were soft errors its cause - transient faults do not have well defined fault model for fault simulation We note that the error detection capability of MACH code (not the error coverage) does not depend on the details of implementation

As an alternative the time redundant scheme RERC, a version of RESO was proposed for the detection of all single errors The method was also shown to be expendable for single error correction and multiple error detection Although RERC may not be practical to ESM Filter because of its time overhead, it is still useful to the function of ESM Filter which may be performed in several other branches of EW (Electronic Warfare) for non real time applications

To conclude, it is very important to define accurate models (fault/error), which can truly represent the most likely physical failure modes for fault-tolerant system designs Otherwise, limited redundant resources would be wasted for some unrealistic models, whose high coverage measure can be misleading Although there already exist well established fault-tolerant techniques for various types of circuits, more effective design may be found when each individual circuit is carefully considered for its own fault tolerance scheme In considering a given circuit or a system, all informations regarding the circuit can be very useful to determine efficient approach Such information may include not only about the internal (i e , implementation details) but also external conditions such as environment of operation, likely range or distribution of input/output of the circuit, application specific requirements, etc From such informations, we can take several advantages such as making some valid assumptions, which may significantly ease the task of design

Along with advancing VLSI technology more research work for fault tolerant system designs will be necessary For example, modern VLSI analog circuits such as those used for neural systems [51], [62] have different failure mechanisms from conventional VLSI digital

circuits, and thus more study in the area is necessary to consider fault tolerance in such circuits.

References

[1] D.P. Siewioreck and R S. Swarz, *The theory and Practice of Reliable System Design* Billenca, MA.Digital Press, 1982

[2] J C Laprie, "Dependable computing and fault tolerance concepts and terminology," *in Proc 15th Int Symp on Fault-Tolerant Computing, Ann Arbor*, pp 2 11, June 1985

[3] A Avizienis and J C Laprie, "Dependable Computing From Concepts to Design Diversity," *in Proc IEEE* vol 74, No 5, pp 629-638, May 1986

[4] D.K. Pradhan, Ed , *Fault-Tolerant Computing, Theory and Techniques*, vol 1 Englewood Cliffs, NJ Prentice-Hall, 1986

[5] R.E. Lyions and W Vanderkulk, "The use of triple modular redundancy to improve computer reliability," *IBM J Res Develop* , vol 7, pp 200 209, 1962

[6] J. von Neumann, "Probabilistic logic and the synthesis of reliable organisms from unreliable components," *Automata Studies, Annals of Mathematical Studies*, no 34 Princetion, NJ Princeton University Press, 1956, pp 43 98

[7] R Toeste, "Digital circuit redundancy" *IEEE Trans Reliability* vol R 13, pp 42 61, 1964

[8] J C Tryon, "Quadded logic," *in Redundancy Techniques for Computing Systems*, W C Mann and R H Wilcox, Ed Washington, DC Spartan Books, 1962, pp 205 228

[9] T.F Klaschka, "Reliability Improvement by Redundancy in Electronic Systems, II An Efficient New Redundancy Scheme Radial Logic," *Royal Aircraft Establishment Ministry of Technology* 69045 Farnborough U K , 1969

[10] W H Pierce, *Failure Tolerant Design* New York Academic Press, 1965

[11] J F Wakerly, *Error Detecting Codes Self-Checking circuits, and application* Elsevier North-Holland c1978

[12] J F Wakerly, "Partially self-checking circuits and their use in performing logical operations," *IEEE Trans Comput* , vol C 23, pp 658 666, Dec 1974

[13] 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, pp 263 269, March 1973

[14] D A Raynolds and G Metze, "Fault Detection Capabilities of Alternating Logic *IEEE Trans. Comput.*, vol. C 27, pp 1093-1098, Dec 1978

[15] J J Shedletsky, "Error correction by alternate-data retry," *IEEE Trans Comput*, vol C 27, pp 106 112, Feb 1978

[16] J H Patel and L Y Fung, "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands," *IEEE Trans Comput*, vol C-31, pp 589 595, December 1988

[17] K -H Huang and J A Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans Comput*, vol C-33, pp 518-528, June 1984

[18] J Y Jou and J A Abraham, "Fault-Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent Computing Structures," *in Proc IEEE* vol 74, No 5, pp 732 741, May 1986

[19] C J Anfinson and F T Luk, "A Linear Algebraic Model of Algorithm-Based Fault Tolerance," *IEEE Trans Comput*, vol C-37, pp 1599-1604, Dec 1988

[20] C M Liu and C W Jen, "Design of algorithm-based fault-tolerant VLSI array processor," *IEE Proc Computers and Digital Techniques (Part E)*, vol 136, No 6, pp 539-547, Nov 1989

[21] R H Pettit, *ECM and ECCM techniques for digital communication systems* Belmont, Calif Lifetime Learning Publications, 1982

[22] D C Schleher, *Introduction to Electronic Warfare* Dedham, MA Artech House, 1986

[23] R G Wiley, *Electronic Intelligence. the Interception of Radar Signals* Dedham, MA Artech House, 1985

[24] R G Wiley, *Electronic Intelligence. the Analysis of Radar Signals* Dedham, MA Artech House, 1982

[25] Agarwal, V K and Agarwal, D P, "On-Line fault Diagnosis in Microprocessor Systems," *Journal of Digital Systems*, vol 4, pp.337-392, Winter 1980

[26] J A Abraham and W K Fuchs, "Fault and Error Models for VLSI," *in Proc IEEE* vol 74, No 5, pp 639-654, May 1986

[27] W K Fuchs and J A Abraham, "A unified approach to concurrent error detection in highly structured logic arrays," *in Proc 14th Int Symp on Fault-Tolerant Computing*, pp 4-9, June 1984

[28] C C Beh, K H Arya, C E Radke, and K E Torku, "Do stuck fault models reflect manufacturing defects?" *in Proc Int Test Conf*, pp 35-41, Nov 1982

[29] P Banerjee and J A. Abraham, "Fault characterization of VLSI MOS circuits," *in Proc Int Conf on Circuits and Computers*, pp 564-568, Sept 1982

[30] J Galiay, Y Crouzet and M Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits Impact on Their Testability," *IEEE Trans Comput* vol C 29 pp 527-531, June 1980

[31] R L Wadsack, "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *Bell Syst Tech J*, vol.57, no 5, pp 1449 1474, May June 1978

[32] D Xavier, R C Aitken, A Ivanov and V K Agarwal, "Experiments on Aliasing in Signature Analysis Registers," *International Test Conference* pp 344 353, 1989

[33] Y Savaria, N C Rumin, J F Hayes, and V K Agarwal "Soft error filtering A solution to the reliability problem of future VLSI digital circuits, *Proc IEEE* vol 74, pp 669-683, May 1986

[34] T C. May, "Soft errors in VLSI Present and future," *IEEE Trans Comp Hybrids, Manuf. Technol*, vol CHMT-2, pp 377-387, Dec 1979

[35] C H Sie, R A Youngblood, J H Liao, and A Turk, "Soft failure modes in MOS RAMs," *in Proc Reliability Phys Symp*, pp 27-32, 1977

[36] J H Kim and S M Reddy, "On the design of Fault-Tolerant Two Dimensional Systolic Arrays for Yield Enhancement," *IEEE Trans Comput*, vol 38, pp 515 525, April 1989

[37] J. H. Patel and L Y Fung, "Concurrent Error Detection in Multiply and Divide Arrays," *IEEE Trans Comput*, vol C-32, pp 417-422, April 1983

[38] M C Howells, R C Aitken and V K Agarwal, "Defect Tolerant Interconnects for VLSI," *Defect and Fault Tolerance in VLSI Systems*, vol 1, I Koren, Ed, Plenum New York, 1989, pp 65-76

[39] I. Koren and D K Pradhan, "Yield and Performance Enhancement Through Redundancy in VLSI and WSI Multiprocessor Systems," *in Proc IEEE* vol 74, No 5, pp 699-711, May 1986

[40] R. W. Hamming, "Error detecting and error correcting codes," *Bell syst Tech J* vol 29, pp 147-160, Jan 1950

[41] R.W *Hamming, Coding and Information Theory* NJ Prentice Hall, 1986

[42] L. Chisvin and R J Duckworth, "Content-Addressable and Associative Memory Alternatives to the Ubiquitous RAM," *Computer*, vol 22 No 6, pp 51 64, July 1989

[43] T. Kohonen, *Content Addressable Memories*, Berlin, New York Springer Verlag, 1980.

[44] S. S. Yau and H S Fung, "Associative Processor Architecture A Survey, *Computing Surveys*, vol.9, no.1, pp 3-27, March 1977

[45] B. Parhami, "Associative Memories and Processors. An Overview and Selected Bibliography," *Proc. IEEE*, vol.61, No 6, pp.722 730, June 1973

[46] J Koo, "Integrated-Circuit Content-Addressable Memories," *IEEE J Solid State Circuits*, vol SC-5, pp 208-215, Feb 1970

[47] S R Jones, "Design, selection and implementation of a content addressable memory for a VLSI CMOS chip architecture," *IEE Proc Computers and Digital Techniques (Part E)*, vol 135, No 3, pp 165-172, May 1988

[48] J Wade and C Sodini, "Dynamic cross-coupled bit line content addressable memory cell for high-density arrays," *IEEE J Solid-State Circuits*, vol SC 22, pp 119 121, Feb 1987

[49] J L. Mundy, J F Burgess, R E Joynson, and C Neugebauer "Low Cost Associative Memory," *IEEE J Solid-State Circuits*, vol SC-7, pp 364 369 Oct 1972

[50] K E Grosspietsch, "Architectures for testability and fault tolerance in content addressable systems," *IEE Proc Computers and Digital Techniques (Part E)*, vol 136, No 5, pp.366-373, Sept 1989

[51] J A G Nijhuis and L Spaanenburg, "Fault tolerance of neural associative memories," *IEE Proc Computers and Digital Techniques (Part E)*, vol 136, No 5, pp 389 394, Sept. 1989

[52] G. M. Blair, "A Content Addressable Memory with a Fault Tolerance Mechanism," *IEEE J. Solid-State Circuits*, vol SC-22, No 4, pp 614-616, August 1987

[53] J. P Roth, "Diagnosis of Automata Failures A Calculus and a Method," *IBM J Res. Devel*, vol 10, July 1966

[54] R. J. Wilson, *Introduction to graph theory*, New York Longman, 1985

[55] F. S. Roberts, *Applied Combinatorics*, Englewood Cliffs, N J Prentice-Hall pp 243, 1984

[56] J. Riordan, "The distribution of Crossings of Chords Joining Pairs of 2n Points on a Circle," *Math.Comp*, vol 29, pp215-222, 1975

[57] J. K Percus, *Combinatorial Methods*, New York Springer Verlag, Applied mathematical sciences, vol 4, 1971

[58] M Eisen, *Elementary combinatorial analysis*, New York Gordon and Breach, 1969

[59] R. E. Walpole and R H. Myers, *Probability and Statistics for Engineers and Scientists*, New York Macmillan, 1978

[60] W W Hines and D C Montgomery, *Probability and Statistics in Engineering and Management Science*, New York: Wiley, 1980

[61] F Maamari and J. Rajski, "A Fault Simulation Method Based on Stem Regions," *Proc Int. Conf. on Computer-Aided Design*, Nov. 1988

[62] C Mead, *Analog VLSI and Neural Systems*, New York: Addison Wesley, 1989.