AN ABSTRACT OF THE DISSERTATION OF

Steven Scott Gorshe for the Doctor of Philosophy in Electrical and Computer Engineering presented on April 19, 2002.

Title: Concurrent Error Detection

Redacted for Privacy

Abstract Approved: _____

Bella Bose

Concurrent error detection (CED) is the detection of errors or faults in a circuit or data path concurrent with normal operation of that circuit. The general approach for CED is to calculate a check symbol for the inputs to the circuit under operation, predict the check symbol that will result for the output of the circuit for those inputs, and compare the predicted check symbol to the one that is actually calculated for the output. If the predicted and actual check symbols are different, an error or fault has been detected. The alternative to this check symbol prediction is to use a second copy of the circuit under operation and compare the results of the two circuits. For some classes of circuits the prediction of the output check symbol can require less circuitry than a second copy of the circuit being tested. Four examples of these types of circuits are examined in this dissertation: Arithmetic Logic Units (ALUs), array multipliers, self-synchronous scrambler-descrambler pairs with their intervening data path, and switch fabrics.

Faults in integrated circuits tend to produce unidirectional errors. Unidirectional errors are those in which all of the errors are in the same direction (e.g., 0 to 1 errors) within the block of data covered by a given check symbol. For this reason, codes that are optimized for unidirectional errors are the focus of investigation for most of the applications. In particular, the Bose-Lin codes are examined for those applications where unidirectional errors are expected to be typical. In order to examine the performance of the Bose-Lin codes in one of these applications, it was necessary to determine the theoretical performance for Bose-Lin codes for error rates beyond what had been previously studied. This analysis of Bose-Lin codes with large numbers of "burst" errors also included a further generalization of the codes.

Concurrent Error Detection

by

Steven Scott Gorshe

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of
Doctor of Philosophy

Presented April 19, 2002
Commencement June 2002

Doctor of Philosophy dissertation of Steven S. Gorshe presented on April 19, 2002

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Head of the Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Redacted for Privacy

Steven Scott Gorshe, Author

ACKNOWLEDGEMENTS

The author thanks Bella Bose for his patient and thoughtful guidance throughout my program, as well as his continuing encouragement and instruction.

The author wishes to thank Adam Lender and M. Robert Aaron for their invaluable reviews, comments, and encouragement on many occasions. I also thank my employers NEC America and PMC-Sierra for their support of my research, and especially those in my management, Cliff Davidow, Rikio Maruta, Toshikazu Matsumoto, Kunitetsu Makino, Luke Falconer, Steve Lang, and Vern Little for their support and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

DEDICATION

Dedicated to my wife Bonnie with deepest appreciation for all her patient support and sacrifices throughout this long process, and also to my boys Alex and Ian for their patience and support. It is also dedicated to my father Louis for instilling an unquenchable curiosity and love of learning, and my mother Ella May for instilling an appreciation of education and perseverance, as well as for their ongoing support.

S.D.G. - J.J.

1. INTRODUCTION TO CONCURRENT ERROR DETECTION

Concurrent error detection (CED) refers to the detection of faults and errors concurrent with the normal operation of a circuit. A typical implementation for concurrent error testing is to first calculate error check codes for the inputs to the circuit. Then, based on the knowledge of how the circuit processes these inputs, a second circuit calculates a predicted value for the output of the original circuit for these inputs. The circuit performing this prediction calculation is referred to as a check prediction circuit. Finally, the error check code is calculated for the actual output of the original circuit and compared to the predicted value. An alternative implementation is for the check prediction circuit to be an identical, second copy of the original circuit and for the outputs of the two circuits to be compared directly. For some classes of circuits, however, the check prediction circuit can be implemented with a much smaller circuit than a second copy of the circuit being tested. This dissertation focuses on such circuits where the check prediction circuit will be smaller than the circuit being tested under operation.

Typically, the faults and errors encountered in integrated circuits are unidirectional. Unidirectional errors are defined as having all of the errors occur in the same direction. In other words, either the error causes 0 to 1 data errors or 1 to 0 data errors, but not both in the same region of interest (i.e., within the data covered by the same error check code).

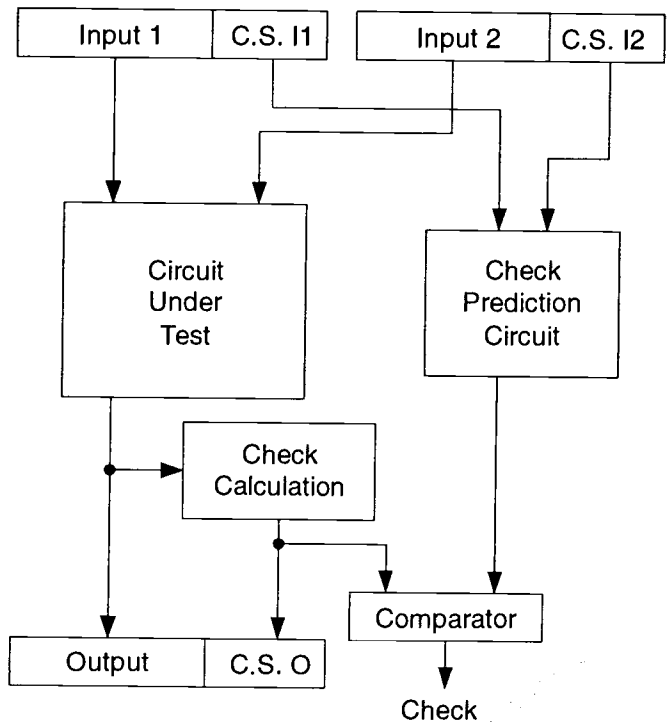


Figure 1.1 – Illustration of a circuit under operation with CED using a check prediction approach

Berger codes, in which the check symbol is binary count of the number of zeros in the data block, have been popular for CED applications because they have the ability to detect all unidirectional errors in a block of data [1]. Bose-Lin codes [2] are also based on the count of the number of zeros in the information data block. The Bose-Lin check symbol is the binary value of this count shortened to a fixed number of bits by taking the modulo remainder of this count. The most significant bits of the check symbol are an $m/2$ out of m code. Bose-Lin codes with r check bits have the advantage of being able to detect up to an appropriate t unidirectional errors in a data block, irrespective of the length of that data block. Due to this property, and the relative simplicity of the Bose-Lin codes, they appear to be

potentially well-suited for concurrent error detection applications. Other popular error detecting codes are Bit-Interleaved Parity (BIP) and Cyclic Redundancy Check (CRC) codes. The Berger, Bose-Lin, BIP, and CRC codes are described in greater detail in the following portion of this introduction.

The types of circuits that are best suited to concurrent error detection are those that have a regular structure. For example, [3], [4], and [5] show the application of Berger codes to the concurrent error detection of arithmetic logic units (ALUs) and unsigned array multipliers. Since the Bose-Lin check symbol is an arithmetic value based on the number of zeros in the data, similar to the Berger codes, circuits that perform arithmetic functions should be particularly amenable to using the Bose-Lin codes for concurrent error detection.

Chapter 2 of this dissertation examines the application of Bose-Lin codes to concurrent error detection with ALUs and presents the results of this analysis. Chapter 3 examines the applicability of Bose-Lin codes to concurrent error detection in unsigned and two's complement array multipliers. Chapter 5 examines concurrent error detection in switch fabrics and compares the performance of Bose-Lin, BIP, and CRC codes that are popular in many telecommunications applications.

One interesting additional question is how the Bose-Lin codes perform when there are errors beyond t unidirectional errors. Chapter 4 gives a framework for analyzing the performance of Bose-Lin codes with $>t$ unidirectional errors. The results of chapter 4 are used in the analysis of chapter 5.

Chapter 6 examines the interaction between CRC concurrent error detection codes and self-synchronous payload scramblers. The problem here is that the feedback

taps inherent in a self-synchronous descrambler give multiple errors in the data due to each transmission channel error. The analysis of Chapter 6 examines the criteria required to maintain error detecting capability in this situation, as well as the criteria required for error correction.

1.1 BERGER AND BOSE-LIN CODE DESCRIPTION

1.1.1 Berger codes

The Berger error detecting codes are implemented by counting the number of zeros in the information word and appending this (binary) number to the information word. Thus, a Berger code requires a minimum of r check bits, where r is the smallest integer such that $r \geq \log_2(k+1)$ and k is the number of bits in the original data word. For example, for an information word of 10010100, which has five zeros, the Berger coded word is 100101000101.

1.1.2 Bose-Lin codes for detecting double and triple errors

Both the double and triple error-detecting codes are constructed by counting the number of zeros in the information word, in the same manner as the Berger codes. The counts for the double and triple error-detecting codes are performed modulo 4 and 8, respectively. In other words, the double and triple error-detecting codes have check length $r=2$ and $r=3$ bits, respectively, and the check symbol (CS) is calculated as:

$$CS = k0 \bmod 2^r.$$

(*k0* is the number of zeros in the information word.)

1.1.3 Bose-Lin codes with more than three check bits (Method 1)

These codes are constructed by taking the modulo 2^{r-1} count of the number of zeros in the information word and then creating the most significant bit (MSB) of the CS by adding 2^{r-2} to this count value. i.e.,

$$CS = (k0 \bmod 2^{r-1}) + 2^{r-2}$$

where adding 2^{r-2} is the same as setting the $(r-1)^{st}$ bit equal to the $(r-2)^{nd}$ bit, and then complementing the $(r-2)^{nd}$ bit. The resulting codes are capable of detecting $2^{r-2}+r-2$ unidirectional errors [1].

1.1.4 Bose-Lin codes with more than four check bits (Method 2)

These codes are formed by following steps. First, take the modulo$(6 \times 2^{r-4})$ of the number of zeros in the information word. The r-4 least significant bits (LSBs) of the remainder are used as the r-4 LSBs of the CS. The three MSBs of this remainder can take the values {000, 001, 010, 011, 100, 101}. These six values are then mapped to one of the possible 2-out-of-4 codes {0011, 0101, 0110, 1001, 1010, 1100}. In summary, then:

$$CS_{LSB} = r\text{-}4 \text{ LSBs of } k0 \bmod (6 \text{ X } 2^{r-4})$$

$$CS_{MSB} = f[3 \text{ MSBs of } k0 \bmod (6 \text{ X } 2^{r-4})]$$

where f[•] is the function mapping from the modulo remainder to the 2-out-of-4 codes. The resulting codes are capable of detecting $5 \text{ X } 2^{r-4} + r - 4$ unidirectional errors. These codes are more efficient than the above Method 1 codes for $r \geq 6$ [1].

Table 1.1 – Example check code values for the Berger and Bose-Lin codes for a 64-bit data word with 37 zeros.

Code Type	Code Value
Berger	0100101
Bose-Lin Double Error	01
Bose-Lin Triple Error	101
Bose-Lin with > 3 check bits (Method 1 with 4 bits)	1001
Bose-Lin with > 4 check bits (Method 2 with 5 bits)	11001

1.2 BIT-INTERLEAVED PARITY (BIP) CODE DESCRIPTION

A BIP-*r* code is constructed by partitioning the data block into *r* interleaved blocks and applying a parity check bit to each partition. For example, a BIP-2 uses one

parity check bit over all the even numbered bits in the data block and a second parity check bit over all the odd numbered bits in the data block. Another popular example is the BIP-8 code that is used on data blocks that contain some number of eight-bit bytes. The first bit of the error check symbol is a parity check over first bit in each data byte, the second check symbol bit is a parity check over the second bit in each data byte, etc. Both the BIP-2 and BIP-8 are described in [6].

1.3 CYCLIC REDUNDANCY CHECK CODE DESCRIPTION

A CRC is a linear, cyclic code. A CRC-r is an r-bit code that is constructed with the following steps. The data block to be protected is regarded as a GF[2] polynomial. Specifically, a k-bit long data block, is regarded as a GF[2] polynomial of degree k-1 with the data block's most significant bit (MSB) being the coefficient of the x^{k-1} term and its least significant bit (LSB) being the coefficient of the x^0 term. Letting $m(x)$ represent this data block, the first step is to multiply $m(x)$ by 2^r which results in a degree $k+r$-1 polynomial with zeros in the r LSBs. The second step is to divide this resulting degree $k+r$-1 polynomial by the code's degree r generator polynomial $g(x)$. (GF[2] division is used, which is also called modulo 2 division.) The r-bit remainder resulting from this division, $r(x)$, is the CRC-r and is then appended to the LSB end of $m(x)$ to form the r LSBs of the degree $k+r$-1 code word $c(x)$. At the receiving or decoding end, $c(x)$ is divided by $g(x)$. A consequence of the code construction is that all code words are divisible by $g(x)$. A non-zero remainder at the decoder therefore indicates the presence or an error. There are variations in the specific details of the code construction (e.g., the remainder is complemented before appending it as the check symbol for some implementations), however the analysis of this dissertation assumes the approach outlined here without loss of generality. CRC codes are discussed in detail in [7].

1.4 REFERENCES

[1] J. M. Berger, "A Note on Error Detecting Codes for Asymmetric Channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.

[2] B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

[3] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. Computer-Aided Design* vol. 11, pp. 525–540, April 1992.

[4] J.-C. Lo, S. Thanawastein, and T.R.N. Rao, "Concurrent Error Detection in Arithmetic and Logical Operations Using Berger Codes," in *Proc. 9th Symp. Computer Arithmetic*, Sept. 1989, pp. 233-240

[5] J.-C. Lo, S. Thanawastein, and T.R.N. Rao, "Berger Check Prediction for Array Multipliers and Array Dividers," *IEEE Trans. Comput.*, vol. 42, pp. 892-896, July 1993

[6] ANSI/ATIS T1.105-2002, *Telecommunications – Synchronous Optical Network (SONET) – Basic Description Including Multiplex Structure, Rates and Formats-2001*

[7] S. Wicker, *Error Control Systems for Digital Communications and Storage*, Prentice-Hall, Upper Saddle River, NJ, 1995

2. A SELF-CHECKING ALU DESIGN WITH EFFICIENT CODES

Steven S. Gorshe

Bella Bose

Proceedings of the IEEE VSLI Test Symposium

445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331

1996

Abstract: uses Berger codes and compares the check value of the ALU output to a predicted check value that is calculated based on the input operand check values. Berger codes have the property of being able to detect all unidirectional errors. More efficient codes exist for detecting up to t unidirectional errors. This paper examines applying these codes to self-testing ALU designs and shows that the potential savings in check circuitry over Berger codes is up to 61 %, depending on the code and the information word length.

2.1 INTRODUCTION

Recently, a self-checking ALU circuit was proposed based on Berger error detecting codes [1]. Berger codes have the property of being optimal for detecting all unidirectional errors in a data word [2]. In many applications, however, it is not necessary to be able to detect all unidirectional errors, but is sufficient to detect up to *t* unidirectional errors. A family of codes has been developed by Bose and Lin [3] that are optimal for detecting up to *t* unidirectional errors and require fewer bits than Berger codes. This paper begins with a review of the Berger and Bose-Lin check codes and then reviews the Berger check prediction ALU of [1]. The application of Bose-Lin codes to this type of ALU is then discussed, and the potential circuitry savings is shown to be significant. (Depending the check code and the length of the information word, the check circuitry savings over Berger codes is in the range of 0–61%.) Finally, it is shown that when Bose-Lin codes are used, the single fault secure property of the ALU is preserved.

2.2 ERROR DETECTING CODES

Both the Berger and Bose-Lin codes are systematic. Bose-Lin codes have the additional property that they need only a fixed number of check bits, independent of the number of information bits. The codes are implemented as follows with examples shown in Table 2.1:

2.2.1 Berger codes

The Berger error detecting codes are implemented by counting the number of zeros in the information word and appending this (binary) number to the information word. Thus, a Berger code requires a minimum of r check bits, where r is the smallest integer such that $r \geq \log_2(k+1)$ and k is the number of bits in the original data word. For example, for an information word of 10010100, the Berger coded word is 100101000101.

2.2.2 Bose-Lin codes for detecting double and triple errors

Both the double and triple error-detecting codes are constructed by counting the number of zeros in the information word, similar to the Berger codes. The counts for the double and triple error-detecting codes are performed modulo 4 and 8, respectively. In other words, the double and triple error-detecting codes have check length $r = 2$ and $r = 3$ bits, respectively, and the check symbol (CS) is calculated as:

$$CS = k0 \bmod 2^r.$$

(*k0* is the number of zeros in the information word.)

2.2.3 Bose-Lin codes with more than three check bits (Method 1)

These codes are constructed by taking the modulo 2^{r-1} count of the number of zeros in the information word and then creating the most significant bit (MSB) of the CS by adding 2^{r-2} to this count value. i.e.,

$$CS = (k0 \bmod 2^{r-1}) + 2^{r-2}$$

where adding 2^{r-2} is the same as setting the $(r-1)^{st}$ bit equal to the $(r-2)^{nd}$ bit, and then complementing the $(r-2)^{nd}$ bit. The resulting codes are capable of detecting $2^{r-2} + r - 2$ unidirectional errors [3].

2.2.4 Bose-Lin codes with more than four check bits (Method 2)

These codes are formed by following steps. First, take the modulo $(6 \times 2^{r-4})$ of the number of zeros in the information word. The r-4 least significant bits (LSBs) of the remainder are used as the r-4 LSBs of the CS. The three MSBs of this remainder can take the values {000, 001, 010, 011, 100, 101}. These six values are then mapped to one of the possible 2-out-of-4 codes {0011, 0101, 0110, 1001, 1010, 1100}. In summary, then:

$$CS_{LSB} = r\text{-}4 \text{ LSBs of } k0 \bmod (6 \text{ X } 2^{r-4})$$

$$CS_{MSB} = f[3 \text{ MSBs of } k0 \bmod (6 \text{ X } 2^{r-4})]$$

where f[•] is the function mapping from the modulo remainder to the 2-out-of-4 codes. The resulting codes are capable of detecting $5 \text{ X } 2^{r-4}$ r - 4 unidirectional errors. These codes are more efficient than the above Method 1 codes for $r \geq 6$ [3].

Table 2.1 – Example check code values for the Berger and Bose-Lin codes for a 64-bit data word with 37 zeros.

Code Type	Code Value
Berger	0100101
Bose-Lin Double Error	01
Bose-Lin Triple Error	101
Bose-Lin with > 3 check bits (Method 1 with 4 bits)	1001
Bose-Lin with > 4 check bits (Method 2 with 5 bits)	11001

2.3 CHECK PREDICTION ALU STRUCTURE

The basic idea of the check prediction ALU is to predict the check value of the ALU output based on the values of the operands and their check values. In [1], it

has been shown that for Berger codes, the check symbols for the following arithmetic and logic operations are:

$S = X + Y \Rightarrow \quad S_C = X_C + Y_C - C_C - c_{in} + c_{out}$

$S = X - Y \Rightarrow \quad S_C = X_C - Y_C - C_C - \tilde{c}_{in} + c_{out} + n$ (2's complement)

$S = X \wedge Y \Rightarrow \quad S_C = X_C + Y_C - (X \vee Y)_C$

$S = X \vee Y \Rightarrow \quad S_C = X_C + Y_C - (X \wedge Y)_C$

$S = X \oplus Y \Rightarrow \quad S_C = X_C + Y_C - 2(X \wedge Y)_C + n$

$S = X \Rightarrow \quad S_C = X_C$

$S = \tilde{X} \Rightarrow \quad S_C = n - X_C$

$S = 0 \Rightarrow \quad S_C = n$

$S = 1 \Rightarrow \quad S_C = 0$

For addition and subtraction, the formulas are derived from the observation that for the addition of the i^{th} bit of the two operands, the operation can be described as:

$$x_i + y_i + c_{i-1} = 2c_i + s_i$$

where c is the carry. C_C is the number of zeros in the internal carries from the ALU. Note that [1] includes the prediction formulas for the complete set of logic functions, but the above set is sufficient for the immediate application.

The ALU proposed in [1] is shown in Figure 2.1 and Table 2.2. The a_0–a_2 signals are the control signals which select the ALU operation. The t_i signals are test control signals that are generated as part of the test logic. Examination of the

circuit in Figure 2.1 confirms that it is capable of performing all of the required check prediction calculations for Berger codes for both arithmetic and logic operations. The MCSA (Multi-operand Carry Save Adder) block is effectively a two-stage, 2's complement adder circuit that performs the check prediction arithmetic specified above.

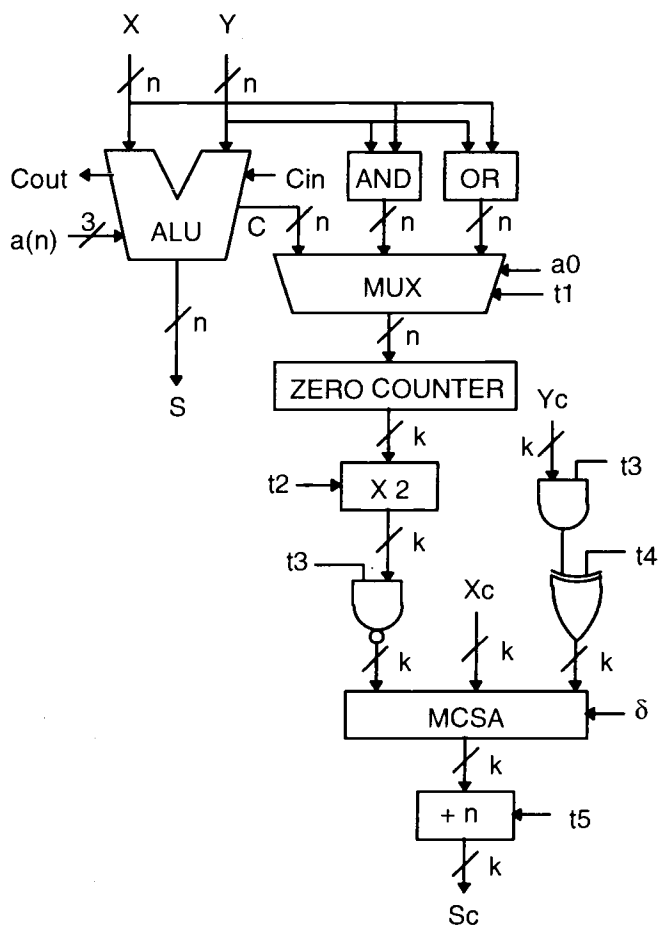


Figure 2.1 – Berger check prediction arithmetic and logic unit

Table 2.2 – Function table for the BCP ALU of Figure 2.1

Control a0 a1 a2	Functions	ti i = 1	2	3	4	5	δ
0 0 X	$S = X + Y + c_{in}$	0	0	1	0	0	$c_{out} - c_{in} + 1$
0 1 X	$S = X - Y - c_{in}$	0	0	1	1	1	$c_{out} - c_{in} + 2$
1 0 0	$S = X \wedge Y$	1	0	1	0	0	1
1 0 1	$S = X \oplus Y$	0	1	1	0	1	1
1 1 0	$S = X \vee Y$	0	0	1	0	0	1
1 1 1	$S = X$	X	X	0	0	0	0

2.4 APPLICATION OF BOSE-LIN CODES TO CHECK PREDICTION ALUS

Throughout this paper, notations such as X_C are used to denote Berger check values, while the notation X_C' is used for an equivalent Bose-Lin check value.

2.4.1 Double and triple error-detecting Bose-Lin code

For these codes, the adaptation of the ALU in Figure 2.1 is straightforward. The modulo 4 and modulo 8 arithmetic here is simply implemented by discarding the appropriate MSBs. Thus, the *k*-bit wide portions of the check prediction circuit are reduced to being two and three bits wide, respectively (i.e., use $k = 3$ and $k = 4$). Also, the Bose-Lin check codes for the operands can be used directly (i.e.,

use X_C' and Y_C' instead of X_C and Y_C in the prediction circuit). Consider for example the prediction formula for addition:

$$S_C' = S_C \bmod 2^r = (X_C + Y_C - C_C - c_{in} + c_{out}) \bmod 2^r$$

$$S_C' = (X_C' + Y_C' - C_C' - c_{in} + c_{out}) \bmod 2^r$$

2.4.2 Bose-Lin codes with more than three check bits

For these codes, the arithmetic circuits are not as straightforward. Care must be taken in determining the MSB, because including the MSB in the arithmetic operations of the prediction circuits will not always produce the correct results. Remember that the MSB is formed by adding 2^{r-2} to the r-1 LSBs. This operation guarantees that the resulting two MSBs will have different values. If all r bits are used in the check prediction calculations, overflow can occur in the MSB location that will produce a different predicted MSB than the CS MSB of the result. (The addition of $FF0F_{HEX}$ and $E09F_{HEX}$ provides an example illustrating this problem.)

To solve the MSB overflow problems, we can first convert the r-1 LSBs back into a modulo 2^{r-1} count by subtracting 2^{r-2}, and using these LSBs in the check prediction operation. (Note that subtracting 2^{r-2} here is the same as complementing the $(r-2)^{nd}$ bit.) After the final r-1 LSB values have been calculated, the MSB is determined in the final stage by again adding 2^{r-2}. For example, consider again the addition operation (recalling that for modulo 2^{r-1} arithmetic, adding 2^{r-2} is the same as subtracting 2^{r-2}):

$$S_C' = (S_C \bmod 2^{r-1}) + 2^{r-2}$$

$$= (X_C + Y_C - C_C - c_{in} + c_{out}) \bmod 2^{r-1} + 2^{r-2}$$

$$= [(X_C' + 2^{r-2}) + (Y_C' + 2^{r-2}) - C_C - c_{in} + + c_{out}] \bmod 2^{r-1} + 2^{r-2}$$

$$S_C' = [X_C' + Y_C' - C_C - c_{in} + c_{out}] \bmod 2^{r-1} + 2^{r-2}$$

Similarly, the prediction formulas for the other operations become:

$$S = X - Y \Rightarrow S_C' = [X_C' - Y_C' - C_C - \tilde{}c_{in} + c_{out} + + n\ ] \bmod 2^{r-1} + 2^{r-2}$$

$$S = X \wedge Y \Rightarrow S_C' = [X_C' + Y_C' - (X \vee Y)_C] \bmod 2^{r-1} + 2^{r-2}$$

$$S = X \vee Y \Rightarrow S_C' = [X_C' + Y_C' - (X \wedge Y)_C] \bmod 2^{r-1} + 2^{r-2}$$

$$S = X \oplus Y \Rightarrow S_C' = [X_C' + Y_C' - 2(X \wedge Y)_C + n] \bmod 2^{r-1} + 2^{r-2}$$

$$S = X \Rightarrow S_C' = X_C'$$

$$S = \tilde{}X \Rightarrow S_C' = [n - X_C' + 4] \bmod 2^{r-1} + 2^{r-2}$$

$$S = 0 \Rightarrow S_C' = (n) \bmod 2^{r-1} + 2^{r-2}$$

$$S = 1 \Rightarrow S_C' = 0100$$

Thus, the check prediction can be performed by making the k-bit wide portions of Figure 2.1 r-1 bits wide, and making the appropriate arithmetic circuit changes in the MCSA block (including adding 2^{r-2} to the result in most cases). Again, X_C' and Y_C' are used instead of X_C and Y_C in the prediction circuit.

2.4.3 Bose_Lin codes with more than four check bits

Remember that the four MSBs of the CS are a 2-out-of-4 code derived from the three MSBs of the modulo $(6 \text{ X } 2^{r-4})$ remainder. Rather than using the 2-out-of-4 codes for the prediction calculations, the four MSBs of X_C' and Y_C' are mapped back to the three-bit values (i.e., take f^{-1}[four MSBs of X_C'] and f^{-1}[four MSBs of Y_C']). The modulo $(6 \text{ X } 2^{r-4})$ remainder of the zeros counter output is taken, and all operations are performed with modulo $(6 \text{ X } 2^{r-4})$ arithmetic. The NAND, AND, and XOR gates of Figure 2.1 remain the same with $k = r\text{-}1$. Since all arithmetic for the MSBs is performed modulo $(6 \text{ X } 2^{r-4})$, the "X 2" block is more than just a simple right-shift. The "X 2" block becomes a $(z + z)$ mod $(6 \text{ X } 2^{r-4})$ circuit, where z is the input. At the output of the check circuit, either the three MSBs must be mapped into a 2-out-of-4 code, or the MSBs of S_C' must be mapped (left) as three-bit values in order to compare the predicted CS to actual CS.

To perform modulo $(6 \text{ X } 2^{r-4})$ arithmetic, some modification is required to conventional adder circuits. For example, if A and B are already modulo $(6 \text{ X } 2^{r-4})$ numbers, then:

if $A + B < 6 \text{ X } 2^{r-4}$:

$$(A + B) \bmod (6 \text{ X } 2^{r-4}) = A + B$$

if $A + B \geq 6 \text{ X } 2^{r-4}$:

$$(A + B) \bmod (6 \text{ X } 2^{r-4}) = A + B - 6 \text{ X } 2^{r-4}$$

The modulo ($6 \times 2^{r-4}$) arithmetic can be implemented as modulo 2^{r-1} with an additional subtraction being performed when $A + B \geq 6 \times 2^{r-4}$. This subtraction function can be included into the adder logic. (Note that for modulo 2^{r-1} arithmetic, subtracting $6 \times 2^{r-4}$ is the same as adding 2^{r-3}.)

2.5 COMPARISON OF HARDWARE FOR THE DIFFERENT CODES

In order to compare the hardware impacts of various code alternatives, the following assumptions are made. (1) A "gate" here is equivalent to the number of transistors in a two-input NAND gate. (2) The circuitry considered includes the registers for the CS in both operands and the CS of the ALU output, as well as all circuits shown outside the ALU in Figure 2.1 and the logic to generate the t_j and d control signals. (3) The circuitry that generates the CS for the operands and ALU output is not included. (4) The size of the check prediction circuits for the Berger codes is 270, 417, 635, and 1035 gates for 8, 16, 32, and 64 bit words, respectively. and (5) For check codes with more than four bits, the comparison of the CS MSBs is performed with the 3-bit values rather than with the 2-out-of-4 codes.

The hardware impacts are compared in Table 2.3 for all cases in which the Bose-Lin codes require fewer bits and less circuitry than the Berger code. The word sizes considered are 8, 16, 32, and 64 bits. It can be seen from Table 2.3 that the additional complexity of calculating the predicted value of the MSBs for Bose-Lin codes with more than four bits (Method 2) offsets the savings from reducing the CS length.

As discussed extensively in [1], the Berger check prediction circuits of Figure 2.1 add some delay to the ALU operation. The use of Bose-Lin codes reduces this delay in most cases by reducing the length of the adder circuits in the MCSA block. For the Bose-Lin codes with more than four check bits, however, the additional delay of performing the arithmetic (including the X2 shift) with modulo $6 \times 2^{r-4}$ offsets the savings from the shorter MCSA adder circuits, so that these codes will have somewhat more delay than with the Berger codes.

Table 2.3 – Comparison of the hardware impacts of the alternative codes

Word Length	# of Bose-Lin code bits	Estimated gate savings over the Berger check ALU	% Savings
8	2	120 gates	44 %
	3	56 gates	21 %
16	2	200 gates	48 %
	3	120 gates	29 %
	4	56 gates	13 %
32	2	336 gates	53 %
	3	200 gates	31 %
	4	117 gates	18 %
64	2	634 gates	61 %
	3	336 gates	32 %
	4	197 gates	19 %
	5 (Method 1)	117 gates	11 %
	5 (Method 2)	0 gates	0 %

2.6 SINGLE FAULT SECURENESS

It has been shown in [1] that if the ALU uses a ripple carry adder, then single faults in the adder circuits will affect the check prediction such that:

$$S_c^* - S_c'' = 2(c_m - c_m') + (s_m - s_m')$$

where: S_c^* = CS of the corrupted ALU output,

S_c'' = corrupted prediction circuit CS,

c_m = uncorrupted carry-out from adder cell *m*,

s_m = uncorrupted sum output from adder cell *m*,

c_m' = corrupted carry-out from adder cell *m*,

s_m' = corrupted sum output from adder cell *m*.

When no faults are present, $S_c^* - S_c'' = 0$. When a fault is present, $|S_c^* - S_c''| \leq 2$. All of the codes discussed in this paper are capable of detecting this fault, since the smallest modulo used is four.

It was also shown in [1] that if the ALU uses group lookahead adders, a single fault will affect the check prediction such that:

$$S_c^* - S_c'' = 2(c_p - c_p') + 2(c_q' - c_q)$$

where: c_p = uncorrupted carry input of a group of slices *m*,

c_q = uncorrupted carry output of group of slices *m*,

c_p' = corrupted carry input of a group of slices *m*,

c_q' = corrupted carry output of group of slices *m*.

This difference holds true as the worst case for the group lookahead adders of interest. Here, $|S_c^* - S_c''| \leq 4$, so all codes except the double error-detecting codes are capable of detecting the fault. (Since the double error-detecting code uses modulo 4 arithmetic, it cannot distinguish between $S_c^* - S_c'' = 0$ and $S_c^* - S_c'' = 4$.)

2.7 CONCLUSIONS

For applications where it is sufficient to detect *t* unidirectional errors, it is more efficient to use Bose-Lin codes than Berger codes. It has been shown in this paper that Bose-Lin codes may also be applied to check prediction ALUs. The use of Bose-Lin codes allows a significant reduction in the amount of check circuitry, but preserves the single fault secure property of the Berger check prediction ALUs.

2.8 REFERENCES

[1] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. Computer-Aided Design* vol. 11, pp. 525–540, April 1992.

[2] J. M. Berger, "A Note on Error Detecting Codes for Asymmetric Channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.

[3] B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

3. SINGLE-FAULT-SECURE CONCURRENT ERROR DETECTION OF TWO'S COMPLEMENT MULTIPLIERS USING BOSE-LIN CODES

Steven S. Gorshe

Paper to be submitted to the *IEEE Transactions on Computers*

445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331

Abstract: Concurrent error detection allows the real-time detection of faults in a circuit during its normal operation. It has been shown that Berger codes can be used to provide a single-fault-secure concurrent error detection for unsigned array multipliers. As demonstrated in this paper, the overhead of the check circuit can be greatly reduced if a Bose-Lin code can be used instead of a Berger code. This paper extends the analysis from an unsigned array multiplier to a more general two's-complement multiplier and demonstrates that a Bose-Lin code is adequate for providing single-fault-secure coverage of both types of multipliers.

3.1 INTRODUCTION

In concurrent error testing, error check codes associated with the input to a circuit are used to predict the error check code that will be calculated for the circuit's output. If the predicted and actual values of the output's check code differ, then a fault has been detected. It is important to keep the size of the prediction circuit small relative to the size of the circuit under operation being tested, in order to keep the probably of a fault in the prediction circuit much, much lower than for one occurring in the circuit being tested. It has been shown that Berger codes may be used for concurrent error detection with unsigned array multipliers to give single-fault-security [1]. Unfortunately, the check prediction circuits can become rather large with the Berger code, especially relative to residue check code approaches [9]. The advantage of a Berger code, however, is that it has also been shown to work well in the more general application of concurrent testing for arithmetic logic units (ALUs), and it is convenient to use a single type of check code for all mathematic circuits [2], [8]. It has also been shown that the Bose-Lin codes, which are related to the Berger codes, can provide a more efficient single-fault-secure ALU test [3]. In this paper, the Bose-Lin codes are analyzed for their performance

with array multipliers. In addition to the unsigned array multiplier, the analysis is extended to a more general two's complement array multiplier.

Section 2 of this paper reviews the Berger and Bose-Lin error detecting codes. Section 3 analyzes their single fault performance for unsigned and two's complement array multipliers. Section 4 shows the potential circuit efficiency gained by using the Bose-Lin codes instead of the Berger codes.

3.2 REVIEW OF BERGER AND BOSE-LIN CODES

Both the Berger and Bose-Lin codes are systematic. Bose-Lin codes have the additional property that they need only a fixed number of check bits, independent of the number of information bits. The codes are implemented as follows with examples shown in Table 3.1:

3.2.1 Berger codes

The Berger error detecting codes are implemented by counting the number of zeros in the information word and appending this (binary) number to the information word [7]. Thus, a Berger code requires a minimum of r check bits, where r is the smallest integer such that $r \geq \log_2(k+1)$ and k is the number of bits in the original data word. For example, for an information word of 10010100, the Berger coded word is 100101000101.

3.2.2 Bose-Lin codes for detecting double and triple errors

Both the double and triple error-detecting codes are constructed by counting the number of zeros in the information word, similar to the Berger codes. The counts for the double and triple error-detecting codes are performed modulo 4 and 8, respectively. In other words, the double and triple error-detecting codes have check length $r = 2$ and $r = 3$ bits, respectively, and the check symbol (CS) is calculated as:

$$CS = k0 \bmod 2^r.$$

($k0$ is the number of zeros in the information word.)

3.2.3 Bose-Lin codes with more than three check bits (Method 1)

These codes are constructed by taking the modulo 2^{r-1} count of the number of zeros in the information word and then creating the most significant bit (MSB) of the CS by adding 2^{r-2} to this count value. i.e.,

$$CS = (k0 \bmod 2^{r-1}) + 2^{r-2}$$

where adding 2^{r-2} is the same as setting the $(r\text{-}1)^{st}$ bit equal to the $(r\text{-}2)^{nd}$ bit, and then complementing the $(r\text{-}2)^{nd}$ bit. The resulting codes are capable of detecting $2^{r-2} + r - 2$ unidirectional errors [4].

3.2.4 Bose-Lin codes with more than four check bits (Method 2)

These codes are formed by following steps. First, take the modulo ($6 \times 2^{r-4}$) of the number of zeros in the information word. The r-4 least significant bits (LSBs) of the remainder are used as the r-4 LSBs of the CS. The three MSBs of this remainder can take the values {000, 001, 010, 011, 100, 101}. These six values are then mapped to one of the possible 2-out-of-4 codes {0011, 0101, 0110, 1001, 1010, 1100}. In summary, then:

$$CS_{LSB} = r\text{-}4 \text{ LSBs of } k0 \bmod (6 \times 2^{r-4})$$

$$CS_{MSB} = f[3 \text{ MSBs of } k0 \bmod (6 \times 2^{r-4})]$$

where f[•] is the function mapping from the modulo remainder to the 2-out-of-4 codes. The resulting codes are capable of detecting $5 \times 2^{r-4} + r - 4$ unidirectional errors. These codes are more efficient than the above Method 1 codes for $r \geq 6$ [3].

Table 3.1 – Example check code values for the Berger and Bose-Lin codes for a 64-bit data word with 37 zeros.

Code Type	Code Value
Berger	0100101
Bose-Lin Double Error	01
Bose-Lin Triple Error	101
Bose-Lin with > 3 check bits (Method 1 with 4 bits)	1001
Bose-Lin with > 4 check bits (Method 2 with 5 bits)	11001

3.3 ANALYSIS OF BOSE-LIN CODES FOR SINGLE-FAULT-SECURITY IN ARRAY MULTIPLIERS

3.3.1 Unsigned array multipliers

In [1], it was shown that if the mathematical effects of all of the array multiplier cells compromising the unsigned multiplier of Figure 3.1 are summed together, the result is:

$$S_c = nX_c + nY_c - X_cY_c - C_c$$

where:

$$X_c = \sum_{i=0}^{n-1} x_i$$

$$Y_c = \sum_{i=0}^{n-1} y_i$$

$$C_c = n^2 - n - \sum_{i=1}^{n-1}\sum_{j=1}^{n} c_{ij}$$

$$S = \sum_{k=0}^{2n-1} s_k$$

The derivation of this result can be understood from the analysis of the two's complement multiplier in the next section.

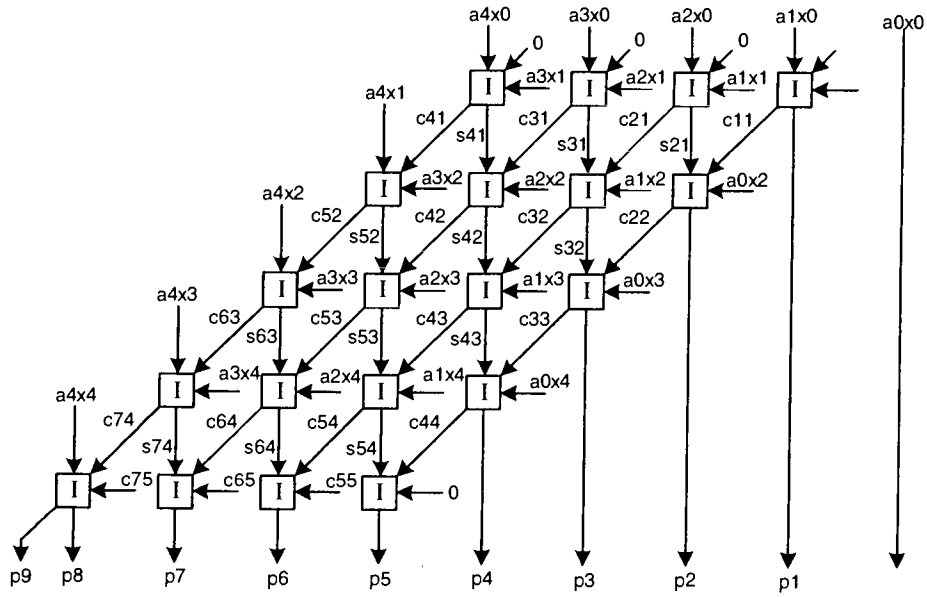


Figure 3.1 – Example of an unsigned array multiplier

3.3.2 Two's complement array multipliers

The two's complement array multiplier is similar to the unsigned multiplier except that two new different types of multiplier cells are introduced [5]. The three types of cells and their mathematical representations are given in Table 3.2. An illustration of a two's complement array multiplier constructed from these cells is shown in Figure 3.2.

3.3.2.1 Check prediction calculation derivation

The individual rows of the two's complement array multiplier can be mathematically represented as follows:

Row 1:

$a0x0 = p0$

$a1x0 + a0x1 = p1 + 2c11$

$a2x0 + a1x1 = s21 + 2c21$

:

$a_{n-2}x0 + a_{n-3}x1 = s_{n-2,1} + 2c_{n-2,1}$

$-a_{n-1}x0 + a_{n-2}x1 = -s_{n-1,1} + 2c_{n-1,1}$

Row 2:

$a0x2 + c11 + s21 = p2 + 2c22$

$a1x2 + c21 + s31 = s32 + 2c32$

:

$a_{n-4}x2 + c_{n-3,1} + s_{n-2,1} = s_{n-2,2} + 2c_{n-2,2}$

$a_{n-3}x2 + c_{n-2,1} - s_{n-1,1} = -s_{n-1,2} + 2c_{n-2,2}$

$-a_{n-1},x1 + a_{n-2}x2 + c_{n-1,1} = -s_{n,2} + 2c_{n,2}$

Row r: $(1 < r \leq n-2)$

$a0x_r + c_{r-1,r-1} + s_{r,r-1} + p_r + 2c_{r,r}$

:

$a_{n-r-2}xr + c_{n-3,r-1} + s_{n-2,r-1} = s_{n-2,r} + 2c_{n-2,r}$

$a_{n-r-1}x_r + c_{n-2,r-1} - s_{n-1,r-1} = -s_{n-1,r} + 2c_{n-1,r}$

:

$a_{n-3}x_r + c_{n-4+r,r-1} - s_{n-3+r,r-1} = -s_{n-3+r,r} + 2c_{n-3+r,r}$

$-a_{n-1}x_{r-1} + a_{n-2}x_r + c_{n-3+r,r-1} = -s_{n-2+r,r} + 2c_{n-2+r,r}$

Row n-1:

$-a0x_{n-1} + c_{n-2,n-2} - s_{n-1,n-2} = p_{n-1} - 2c_{n-1,n-1}$

$-a1x_{n-1} + c_{n-1,n-2} - s_{n,n-2} = s_{n,n-1} - 2c_{n,n-1}$

:

$-a_{n-3}x_{n-1} + c_{2n-5,n-2} - s_{2n-4,n-2} = s_{2n-4,n-1} - 2c_{2n-4,n-1}$

$-a_{n-2}x_{n-1} + c_{2n-4,n-2} - a_{n-1}x_{n-2} = s_{2n-3,n-1} - 2c_{2n-3,n-1}$

This row information can be generalized for each individual cell as follows:

k^{th} cell in Row 1:

k=0: $a0x0 = p0$

$1 \leq k \leq n-2$: $a_kx0 + a_{k-1}x1 = s_{k,1} + 2c_{k,1}$

k=n-1: $-a_{n-1}x0 + a_{n-2}x1 = -s_{n-1,1} + 2c_{n-1,1}$

k^{th} cell in Row r $(2 \leq r \leq n-2)$:

$0 \leq k \leq n-r-2$: $a_kx_r + c_{r+k-1,r-1} + s_{r+k,r-1} = s_{r+k,r} + 2c_{r+k,r}$

$n\text{-}r\text{-}1 \le k \le n\text{-}3$: $a_k x_r + c_{r+k-1,r-1} - s_{r+k,r-1} = -s_{r+k,r} + 2c_{r+k,r}$

$k = n\text{-}2$: $-a_{n-1}x_r + a_{n-2}x_r + c_{n-3+r,r-1} = -s_{n-2+r,r} + 2c_{n-2+r,r}$

<u>k^{th} cell in Row n-1</u>:

$0 \le k \le n\text{-}3$ $-a_k x_{n-1} + c_{n+k-2,n-2} - s_{n+k-1,n-2} = s_{n+k-1,n-1} - 2c_{n+k-1,n-1}$

$k=n\text{-}2$: $-a_{n-2}x_{n-1} + c_{2n-4,n-2} - a_{n-1}x_{n-2} = s_{2n-3,n-1} - 2c_{2n-3,n-1}$

<u>k^{th} cell in Row n:</u>

$k=0$: $-c_{n-1,n-1} + s_{n,n-1} = p_n - 2c_{n,n}$

$1 \le k \le n\text{-}3$: $-c_{n+k-1,n-1} - c_{n+k-1,n} + s_{n+k,n-1} = p_{n+k} - 2c_{n+k,n}$

$k=n\text{-}2$: $-c_{2n-3,n-1} - c_{2n-3,n} + a_{n-1}x_{n-1} = p_{2n-2} - 2p_{2n-1}$

Next let:

N_{pm} = number of 1s in the magnitude portion of product P

$$N_{pm} = \sum_{j=0}^{2n-2} p_j$$

N_{am} = number of 1s in the magnitude portion of input A

$$N_{am} = \sum_{j=0}^{n-2} a_j$$

N_{xm} = number of 1s in the magnitude portion of input X

$$N_{xm} = \sum_{j=0}^{n-2} x_j$$

N_{spos} = the number of 1s in the positive cell sum outputs

$$N_{spos} = \sum_{i=1}^{n-2} \sum_{j=i}^{n-2} s_{ji} + \sum_{j=n-1}^{2n-3} s_{j,n-1}$$

N_{sneg} = the number of 1s in the negative cell sum outputs

$$N_{sneg} = \sum_{i=1}^{n-2} \sum_{j=n-1}^{n+1-2} s_{ji}$$

N_{cpos} = the number of 1s in the positive cell carry outputs

$$N_{cpos} = \sum_{i=1}^{n-2} \sum_{j=i}^{n+i-2} c_{ji}$$

N_{cneg} = the number of 1s in the negative cell carry outputs

$$N_{cneg} = \sum_{i=n-1}^{n} \sum_{j=i}^{2n-3} c_{ji}$$

Summing the equations for all of the cells in all of the rows and substituting gives:

$$(N_{am})(N_{xm}) - (a_{n-1})(N_{xm}) - (x_{n-1})(N_{am}) + a_{n-1}x_{n-1} = N_{pm} - 2p_{2n-1} + N_{cpos} - N_{cneg}$$

If we let N_a, N_x, and N_p represent the number of ones in the *a* and *x* inputs out product output, respectively, then the check prediction equations become:

$$Np = (Na)(Nx) - (2)(x_{n-1})(Na) - (2)(a_{n-1})(Nx) + (4)(a_{n-1})(x_{n-1}) + (3)(p_{2n-1}) - N_{cpos} + N_{cneg}$$ Berger

$$Np = [(Na)(Nx) - (2)(x_{n-1})(Na) - (2)(a_{n-1})(Nx) + (4)(a_{n-1})(x_{n-1}) + (3)(p_{2n-1}) - N_{cpos} + N_{cneg}] \mathrm{mod} m$$ Bose-Lin

The check symbols for the product, a, and x inputs are CSp, CSa, and CSx, respectively.

$CSp = 2n - (N_{pm} + p_{2n})$.

$$CSa = n - (N_{am} + a_{n-1})$$
$$CSx = n - (N_{xm} + x_{n-1})$$

Substituting and solving for CSp gives:

$$CSp = (n)(CSx + CSa + 2x_{n-1} + 2a_{n-1} + 2) - n^2 - (CSa)(CSx) - (2)(x_{n-1})(CSa) - (2)(a_{n-1})(CSx) + (4)(a_{n-1})(x_{n-1}) - (3)(p_{2n-1}) + N_{cpos} - N_{cneg}$$ Berger

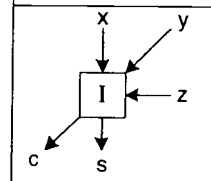
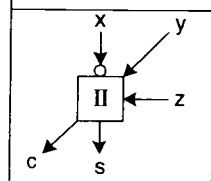
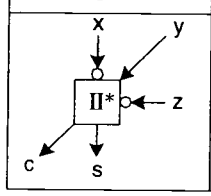
$$CSp = [(n)(CSa + CSx + 2x_{n-1} + 2a_{n-1} + 2) - n^2 - (CSa)(CSx) - (2)(x_{n-1})(CSa) - (2)(a_{n-1})(CSx) + (4)(a_{n-1})(x_{n-1}) + (3)(p_{2n-1}) - N_{cpos} + N_{cneg}]\mathrm{mod}A$$ Bose-Lin

where A is the modulo for the Bose-Lin code (A=4 for the double error detecting 2-bit code and A=8 for the triple error detecting 3-bit code). In the case where $(n)\mathrm{mod}A=0$, which will be relatively common, the Bose-Lin check prediction equation further reduces to:

$$CSp = [- (CSa)(CSx) - (2)(x_{n-1})(CSa) - (2)(a_{n-1})(CSx) + (4)(a_{n-1})(x_{n-1}) + (3)(p_{2n-1}) - N_{cpos} + N_{cneg}]\mathrm{mod}A$$

Bose-Lin with $(n)\mathrm{mod}A=0$

Table 3.2 – Multiplier array cell types

CELL TYPE	EQUATION
	2c + s = x + y + z
	2c - s = y + z - x
	-2c + s = y - x - z

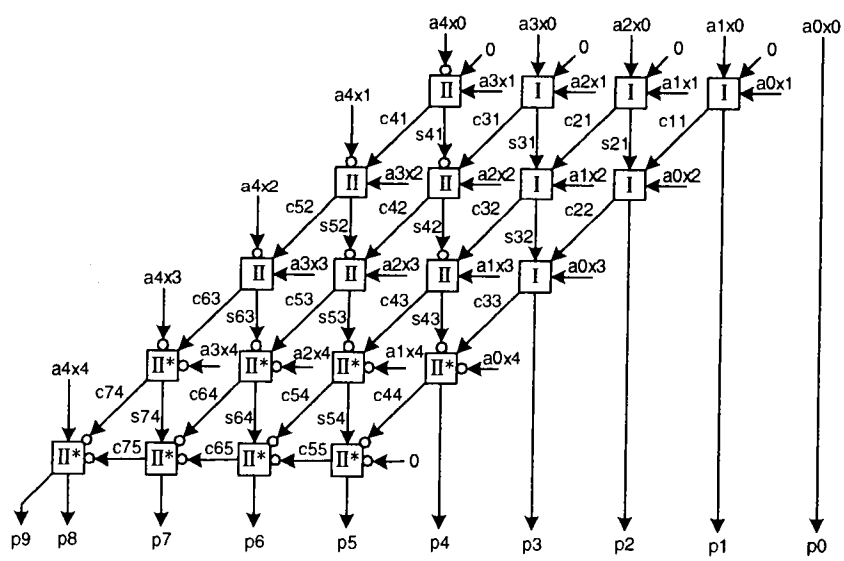


Figure 3.2 – Example two's complement array multiplier

3.3.2.2 Single fault security

From the check prediction equation, it can be seen that a fault it undetectable if

$$\Delta N_p = \Delta N_{cneg} - \Delta N_{cpos}$$

For the purpose of determining single fault security, it is useful to partition the multiplier array into three sections as shown in Figure 3.3 based on the type of cells.

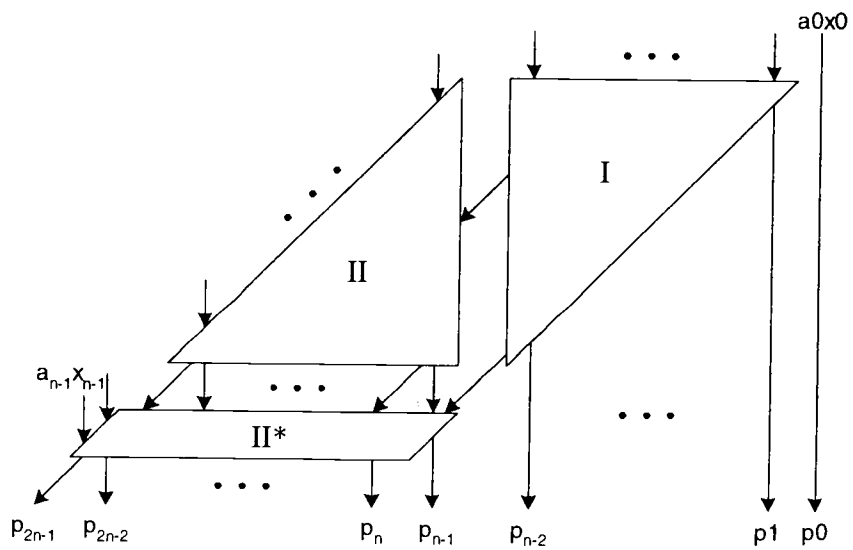


Figure 3.3 – Partitioned array multiplier

In the I section, there are four possible cases resulting from a single fault in a column. Each of these faults results in a change to the product terms coming from the I section, and in some cases also affects the carry total N_{cpos}. For the cases of sum input faults, the change to N_{cpos} is due to the carry output, and for the carry input faults, the change in N_{cpos} is due to the combination of the original fault and its effect on the carry output. A $\Delta N=+1$ corresponds to a 0 to 1 change and a $\Delta N=-1$ corresponds to a 1 to 0 change. These cases are summarized in Table 3.3

Table 3.3 – Affects of single input faults to cells in the I region

Case	Affect
1	$\Delta Np + \Delta N_{cpos} = 1$
2	$\Delta Np + \Delta N_{cpos} = -1$
3	$\Delta Np + \Delta N_{cpos} = +2$
4	$\Delta Np + \Delta N_{cpos} = -2$
5	$\Delta Np + \Delta N_{cpos} = 0$ with a +1 carry out of I
6	$\Delta Np + \Delta N_{cpos} = 0$ with a –1 carry out of I
NOTE: The ΔN_{cpos} values here include the initial fault if that fault was on a carry input.	

Cases 1-4 are contained within the I region are detectable. Cases 5-6 are not detectable within in the I region, but affect either the adjacent II or II* regions.

In the II region, there is no direct affect on a product output since all of the outputs of the II region feed into the II* region. The affects of faults in the II region are summarized in Table 3.4. As noted in the table, cases 7-10 can also be caused by a carry-in from a fault in the I section.

Table 3.4 – Affects of single input faults to cells in the II region or a carry-in from an I region fault

Case	Affect
7	$\Delta N_{sneg} = \Delta N_{cpos}$ (Note 2)
8	$\Delta N_{sneg} - 1 = \Delta N_{cpos}$ (Note 2)
9	$\Delta N_{sneg} + 1 = \Delta N_{cpos}$ (Note 2)
10	$\Delta N_{sneg} - 2 = \Delta N_{cpos}$
11	$\Delta N_{sneg} + 2 = \Delta N_{cpos}$

NOTE 1: The ΔN_{cpos} values here include the initial fault if that fault was on a carry input for a cell in the II region.

NOTE 2: The outcomes in the II region are also possible due to carry in from a fault in the I region, in which case the effect of the fault in the I region are not taken into account in this table.

Next examine the II* region. The possible outcomes due to a fault in the II* region are shown in Table 3.5.

Table 3.5 – Affects of single input faults to cells in the II* region

Case	Affect
12	$\Delta Np - \Delta N_{cneg} = +1$
13	$\Delta Np - \Delta N_{cneg} = -1$
14	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +1$
15	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +2$
16	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -1$
17	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -2$
18	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = 0$ with an effect on p_{2n-1}
NOTE: The ΔN_{cpos} values here include the initial fault if that fault was on a carry input.	

Of course, any fault affecting the II region directly effects the II* region since all of the II outputs are inputs to the II* region. It is also possible that the carry out from an I region fault directly affects the II* region without affecting the II region. Finally, of course, an I fault can affect the II region which in turn affects the II* region. The possible outcomes these combinations of affected regions are shown in Table 3.6.

Table 3.6 – Affects of faults in the I or II region leading to an impact on the II* region

Case	Affect	
19	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +1$	I into II*
20	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -1$	I into II*
21	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -1$	II into II*
22	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +1$	II into II*
23	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +2$	II into II*
24	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -2$	II into II*
25	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +1$	I into II into II*
26	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -1$	I into II into II*
27	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = +2$	I into II into II*
28	$\Delta Np + \Delta N_{cpos} - \Delta N_{cneg} = -2$	I into II into II*

From Tables 3.3-3.6, it can be seen that any single input fault will be detectable. Further, since the mathematical changes are –2, -1, +1, or +2, a 2-bit Bose-Lin

code, which uses modulo 4 arithmetic, is capable of detecting the fault. Hence, single fault security can be accomplished with the 2-bit Bose-Lin code.

3.4 CHECK PREDICTION CIRCUITS

The check prediction circuit for the two's complement array multiplier is shown in Figure 3.4. The MCSA is a multi-function carry-save adder. The amount of circuitry required for the check prediction can be calculated on the basis if how many full adder (FA) and half adder (HA) cells are required. It is understood, however, that analog techniques such as those described by Lo and Metra would further reduce the amount of overhead circuitry [6].

In order to calculate the amount of circuitry, first consider the following. The number of cells in the LSB column for both the carry counter and the MCSA is 1 + (#inputs-3)/2, since the first three inputs occupy one FA cell and the output of that cell combines with two other inputs for the next FA cell, and etc. The number of inputs to the second column of the carry counter or MCSA is the number of FA and HA cells from the LSB column (i.e., the number of possible carries out of the LSB column). Again, there are 1 + (#inputs-3)/2 FA cells in the second column, and etc. for $\log_2$(#original input) columns. Fractional remainders in a column are a HA cell. Note that for Bose-Lin codes, the cells in the second column don't need their carry output circuits, and as a result are counted as fraction of a FA cell.

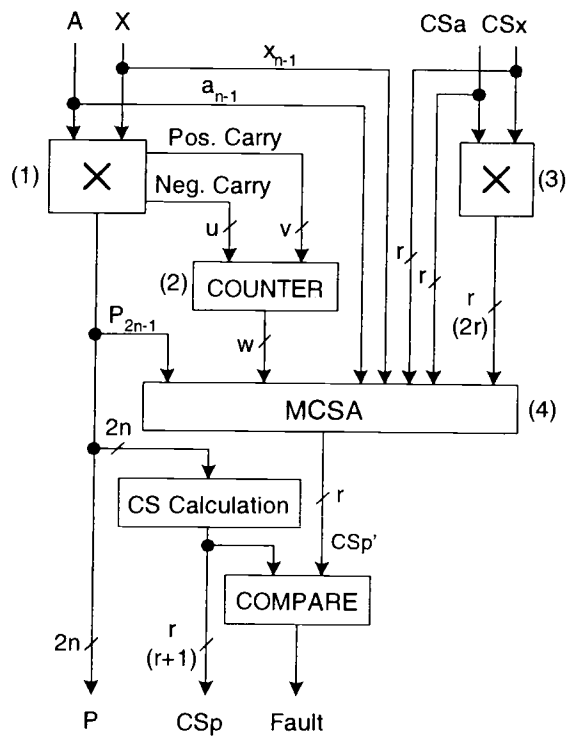


Figure 3.4 – Check prediction circuit

The number of inputs to the carry counter (2) is u + v = $n^2 - n - 1$ (i.e., one less than the number of cells in the main multiplier (1)). The number of inputs to the MCSA for the Berger code case is: $3 + w + r + r + 2r = 3 + 4r + \log_2(n^2 - n - 1)$. For the case of the Bose-Lin codes, the number of MCSA inputs is $3 + r + r + r + r = 3 + 4r = 11$ for 2-bit Bose-Lin codes, regardless of the size of n. The values for amount of circuitry in the check prediction circuits is shown in Table 3.7 for the cases if n = 8, 16, 32, and 64. The amount of circuitry for a second multiplier is included for the sake of comparison.

Table 3.7 – Amount of check prediction circuitry required for various values of n

n	Berger	Bose-Lin 2	2^{nd} Multiplier
8	80 FA + 8 HA (4-bit comparator)	41 FA + 3 HA (2-bit comparator)	48 FA + 8 HA (8-bit comparator)
16	277 FA + 9 HA (5-bit comparator)	156 FA + 2 HA (2-bit comparator)	224 FA + 16 HA (16-bit comparator)
32	1044 FA + 9 HA (6-bit comparator)	626 FA + 2 HA (2-bit comparator)	960 FA + 32 HA (32-bit comparator)
64	4101 FA + 10 HA (7-bit comparator)	2526 FA + 2 HA (2-bit comparator)	3968 FA + 64 HA (64-bit comparator)

As can be seen from Table 3.7, the Berger code check prediction circuits are always larger that those of a second multiplier, so there is little advantage to a Berger code approach. The Bose-Lin codes, however, consistently have less circuitry than using a second multiplier. For n=8, the Bose-Lin code check prediction circuits are only 84% of the circuitry required a second multiplier. For n=64, the Bose-Lin code check circuits are only 63% of that for a second multiplier. (The check symbol calculation circuits are left out of the size comparison under the assumption that they are already in use to protect the integrity of the other circuits and data paths.)

3.5 DELAY CONSIDERATIONS

The delay carry outputs from the main multiplier can be input to the carry counter such that the delay between when the product is available and when the predicted CS is available is the propagation delay through the columns of the MCSA. If the delay through an adder cell is t_{cell} and the delay through the comparator is t_{comp}, then the delay differential is:$(t_{cell})\log_2(3 + 4r + \log_2(n^2 - n - 1)) + t_{comp}$ for Berger codes and $(t_{cell})(3) + t_{comp}$ for 2-bit Bose-Lin codes.

3.6 CONCLUSIONS

The 2-bit Bose-Lin codes have been shown to provide single fault security for two's complement array multiplier. They have also been shown to require much less circuitry than either a second multiplier or a Berger code approach. In addition, the amount of delay for performing the check prediction is comparable to that required to compute the CS value over the final product. Thus, Bose-Lin codes provide an attractive alternative to Berger codes for check prediction in array multipliers, especially if check prediction is also being applied to ALUs with the same operands.

3.7 REFERENCES

[1] J.-C. Lo, S. Thanawastein, and T.R.N. Rao, "Berger Check Prediction for Array Multipliers and Array Dividers," *IEEE Trans. Comput.*, vol. 42, pp. 892-896, July 1993

[2] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. Computer-Aided Design* vol. 11, pp. 525–540, April 1992.

[3] S. Gorshe and B. Bose, "A Self-Checking ALU Design with Efficient Codes," in *Proc. 14th IEEE VLSI Test Symp.*, pp. 157-161, 1996

[4] B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

[5] I. Koren, *Computer Arithmetic Algorithms*, Englewood Cliffs, NJ: Printice-Hall, 1993

[6] C. Metra and J.-C. Lo, "Compact and High Speed Berger Code Checker." in *Proc. of 2nd IEEE On-Line Test Workshop*, pp. 144-149, 1996

[7] J. M. Berger, "A Note on Error Detecting Codes for Asymmetric Channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.

[8] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. Computer-Aided Design* vol. 11, pp. 525–540, April 1992.

[9] [Spar1] U. Sparmann and S. M. Reddy, "On the Effectiveness of Residue Code Checking for Parallel Two's Complement Mutlipliers," *IEEE Trans. VLSI Systems*, vol. 4, pp. 227-239, June 1996

4. GENERALIZED BOSE-LIN CODES AND AN ANALYSIS OF THEIR PERFORMANCE FOR CASES BEYOND *t* UNIDIRECTIONAL ERRORS

Steven S. Gorshe

Paper to be submitted to the *IEEE Transactions on Computers*

445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331

Abstract: Bose-Lin codes are systematic codes where the check symbol is based on the count of the number of zeros in the information word. The paper generalizes the codes such that the MSBs are considered to be an $m/2$-out-of-m code, where m can range between 0 and $r/2$, with r being the check symbol length. A Bose-Lin code is guaranteed to detect up to t unidirectional errors, regardless of the length of the information word. This paper examines the Bose-Lin codes for cases where $>t$ unidirectional errors occurs. The analysis shows that the codes can reliably detect many error cases of $>t$ unidirectional errors, and that the performance improves with larger m.

4.1 INTRODUCTION

Bose-Lin error detecting codes have the property of being optimum for detecting up to t unidirectional errors in an information word, regardless of the length of the information word. [1] Unidirectional errors are typical of those created by faults within integrated circuits. A fault may occur that affects $>t$ bits (e.g., a fault on one or more bits of a data path). These cases raise the questions of how well Bose-Lin codes perform in the presence of errors other than $<t$ unidirectional errors, which are the subject of analysis of this paper. The results show that Bose-Lin codes can perform very well for $>t$ unidirectional, and also shows the choice of m that gives t_{max} (i.e., $m = 4$) is not the optimum choice for $>t$ errors.

The paper begins with a review and generalization of Bose-Lin codes, and then proceeds to the analysis for $>t$ unidirectional errors.

4.2 GENERALIZATION OF BOSE-LIN CODES

In the original paper by Bose and Lin, the codes were described as a family with three different methods of construction, depending on the number of bits used for the check symbol (CS). Here they are presented in a generalized form with the three familiar methods being seen as special cases.

In general, Bose-Lin codes are systematic codes that are based on the count of the number of data zeros in the information word. With Berger codes [2] the CS is directly the binary number representing the number of zeros in the information word. If k is the number of bits in the information word and $k0$ is the number data zeros, then the required number of CS bits, r, is $r=\lceil \log_2(k+1) \rceil$, where $\lceil x \rceil$ is the smallest integer $\geq x$. Hence, CS length r is dependent on k. With Bose-Lin codes, count of $k0$ is taken modulo A so that the r remains the same regardless of k.

Specifically, for Bose-Lin codes count $k0$ is taken modulo A:

$$A = \binom{m}{m/2} \times 2^{r-m}$$

where the m MSBs of the CS are an $m/2$-out-of-m code and the r-m CS LSBs are the modulo 2^{r-m} remainder of $k0$. Examples are given in Table 4.1 Clearly, $m=0$ is the easiest to implement since the $>r$ MSBs of count $k0$ are discarded. For $m\neq 0$, the one-to-one mapping of the m-1 modulo remainder bits into $m/2$-out-of-m code words is arbitrary.

Table 4.1 – Code construction example for r=10 and k0 = $13017_{10}=11001011011001_2$

m	*k0* mod *A*	*A*	Code Word
0	1011011001	2^{10}	1011011001
2	011011001	2×2^8	01 11011001
4	011001001	6×2^6	0110 001001
6	11011001	20×2^4	111000 1001
8	10001001	70×2^2	11000011 01
10	10100101	252	0000011111

4.3 CODE PERFORMANCE WITH UNIDIRECTIONAL ERRORS

Note that the proof of the unidirectional error detecting capability of the Bose-Lin codes here is different than the proof offered in [1] since the approach to the proof taken here is more instructive for the remainder of this section. To begin with, first the conditions under which undetectable errors occur must be established. Then we will determine the fraction of the total possible error cases in which undetectable errors occur.

Theorem 4.1: Errors are undetectable whenever $(u+v) \bmod A = 0$, where:

u = the number of errors in the information word

v = the arithmetic change in the CS due to errors in the CS LSBs

Proof: Clearly, if the number of unidirectional errors in the information word is equal to the code modulo *A* and there are no errors in the CS, then the resulting CS will be the same as the original CS, and hence no errors can be detected. Also, clearly any errors in the *m* MSBs are detectable since they will destroy the *m*/2-out-of-*m* code. For combinations of errors in the information word and the CS LSBs, we have the following, situation, which is illustrated in Figure 4.1. Take first the case of 0 to 1 errors. Each information word error here takes away one zero, decreasing the received *k0* (*k0'*) by 1. Each CS error here increases the arithmetic value of the received CS LSBs by 2^j, where *j* is power of 2 represented by that bit. By definition, errors are undetectable if the CS calculated over the received information word is equal to the received CS (i.e., the received CS is correct for that information word). If the raw count *k0* was used as the CS instead of a modulo remainder of *k0*, then all error cases would be detectable. (This type of code is known as a Berger code [2].) However, due to the modulo remainder being used for the CS, errors are undetectable if the decrease in the number of zeros in the information word (u) plus the arithmetic change in the CS LSBs (v) is equal to the code modulo. QED

	↓	111
Effects of information	↓	101
word errors (u)	↓	100
		...
Effects of	↑	010
check symbol errors (v)	↑	001
	↑	000

Figure 4.1 – Error effect example for 0 to 1 errors

Theorem 4.2: A generalized Bose-Lin code is capable of detecting up to t unidirectional errors, where:

$$t_{max} = \left(\frac{m!}{(m/2)!^2} - 1 \right) \left(2^{r-m} \right) + r - m$$

Proof: The conditions for undetectable errors were established in Theorem 4.1. Errors in the CS LSBs contribute a 2^j term to $(u+v) \bmod A = 0$, whereas information word errors only contribute a 1 term. Hence, the smallest number of total errors that satisfy the undetectability condition occurs when then CS LSBs are originally all zeros and all are received in error (assuming 0 to 1 errors). Arithmetically, this gives $v = 2^{r-m}-1$. So, the smallest number of total errors occurs with $(u+2^{r-m}-1) \bmod A = 0$. The smallest value of u for which this is true is $A-2^{r-m}-1$. The maximum number of detectable errors, then is one less than this u_{min} plus errors in all the LSBs (r-m):

$$t_{max}=\left(\frac{m!}{(m/2)!^2}-1\right)\left(2^{r-m}\right)+r-m \quad \text{QED}$$

It has been noted that for $<t$ unidirectional errors, m=4 gives the best code performance (i.e., the largest value of t for a given r as long as $r\geq 5$).

A point that becomes apparent from the proof of theorem 4.2 is that not all values of $t>t_{max}$ will result in undetectable errors. In fact, due to the modulo arithmetic used in creating the CS, there is a repeating pattern of errors interacting with the CS. The number of errors, E, that can be undetectable is given in theorem 4.3. The number of undetectable error cases is given in theorem 4.4 and the fraction of all the potential error cases that are undetectable is given in the theorem 4.5.

Theorem 4.3: The number of different error counts, E (i.e., the different numbers of errors), that can produce undetectable errors is:

$$NEu=2^{r-m-1}$$

Proof: Per theorem 4.2, errors are undetectable whenever $v+(u)\text{mod}A=A$. Now E is the number of errors in both the information word and CS, so $E=u+\text{weight}(v)$, where weight(v) is the number of errors producing v (assuming 0 to 1 errors without loss of generality). Substituting, we have $(E)\text{mod}A=A-v+\text{weight}(v)$ for undetectable errors. $A-v+\text{weight}(v)$ will take on unique values for each value of v. However, values of v that differ in only the LSB produce the same $A-v+\text{weight}(v)$ value, since an error in the CS LSB affects both v and weight(v) in the same direction. (e.g., a 0 to 1 error in the LSB adds one to both v and weight(v). Hence, only half the CS values can produce a unique, undetectable value of E.

Since there are 2^{r-m} CS values, there are $2^{r-m}/2=2^{r-m-1}$ unique undetectable values of $(E)\mathrm{mod}A$. QED

Theorem 4.4: The number unique cases, *EC*, with undetectable unidirectional errors for a Bose-Lin code is:

$$EC=\left(\frac{(m)!}{(m/2)!^2}\right)\left(3^{r-m}\right)$$

Note: Here the term "unique" means the following. Since *E* errors in the information word is indistinguishable from *E+A* errors, only the case of *E* errors will be considered to be unique for our purposes. Also, assuming 0 to 1 errors, the number of cases here ignores the specific number of ways in which each given *k0* zeros can be distributed among the *I* information word bits and ignores the ways in which the errors can be distributed among the *k0* zeros. Similarly for 1 to 0 errors. As *I* increases, these combinations of zero distributions and error distributions averages out such that each CS can be regarded as having roughly an equal number of these combinations.

Proof: Examine the case where only 0 to 1 errors occur. (The proof is the same for 1 to 0 errors.) There are 2^{r-m} ways for errors to be distributed in the CS LSBs, including zero errors, however the errors can only occur in the vulnerable bits (i.e., the 0 bits here). There are $\binom{r-m}{0}$ LSB values in which there are no zeros in the *r-m* LSBs, and here only one LSB error patterns has undetectable errors (i.e., no errors with $(u)\mathrm{mod}A=0$). There are $\binom{r-m}{1}$ LSB values in which there are a single zero in the LSBs. Here, there are two values of *u* that can have undetectable errors

with LSB error patterns (i.e., no errors and a single error in the one 0 bit). Similarly, there are $\binom{r-m}{r-m}$ LSB values with all 0s, and 2^{r-m} error patterns are possible in the LSBs, since all of the bits are vulnerable. Summing all of these gives:

$$N = \binom{r-m}{0}(2^0) + \binom{r-m}{1}(2^1) + \cdots + \binom{r-m}{r-m}(2^{r-m})$$

$$N = \sum_{i=0}^{r-m} \binom{r-m}{i}(2^i) = 3^{r-m}$$

Since there are $\left(\frac{m!}{(m/2)!^2}\right)$ different MSB patterns per LSB combination, we have the stated results. QED

Theorem 4.5: The fraction of error cases that result in undetectable errors for a Bose-Lin code is:

$$Fund = \frac{1}{(2^{m/2})\left[\left(\frac{m!}{(m/2)!^2}\right)(2^{r-m}) - \frac{m}{4} - \frac{r-m}{3}\right]}$$

Proof: The number of unique cases with undetectable errors was established in Theorem 4.4. Letting A be the code modulo, the total number of ways in which 1-A errors can distribute themselves to give a unique case for a given MSB code is:

$$Cases/MSB = \sum_{i=0}^{m/2} \binom{m/2}{i} \left[\sum_{k=0}^{r-m} \binom{r-m}{k} \sum_{j=0}^{k} \binom{k}{j} (A-i-j) \right]$$

where i = the number of errors in the MSBs

k = the number of LSBs vulnerable to errors

j = the number or errors in the LSBs

A-i-j = the number of errors in the information word

$$A = \left(\frac{m!}{(m/2)!^2} \right) (2^{r-m})$$

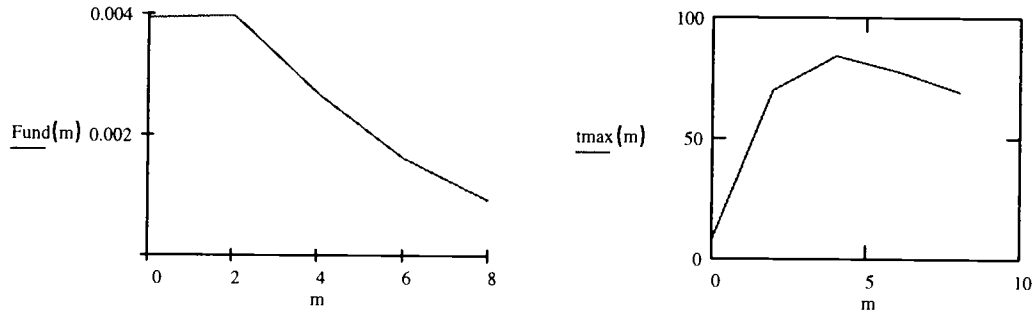
Reducing and substituting gives:

$$Cases/MSB = (3^{r-m})(2^{m/2}) \left(\left(\frac{m!}{(m/2)!^2} \right) - \frac{m}{4} - \frac{r-m}{3} \right)$$

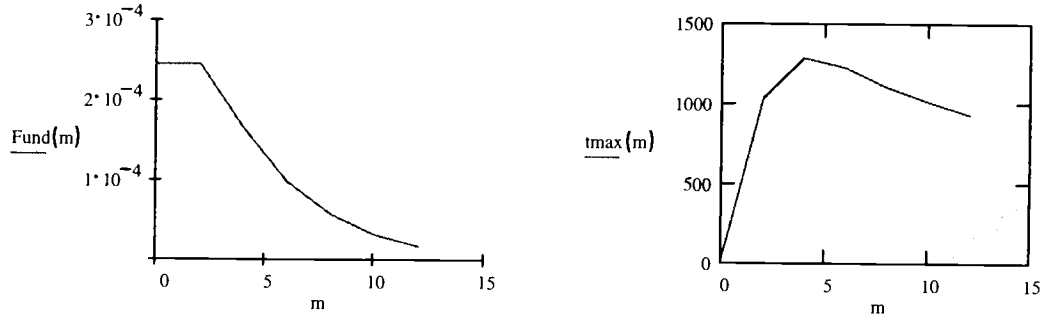
(Note that again, the ways in which a given number of errors distribute themselves in the information word are not "unique" since they are all identical in how each would affect the code performance.) Since there are $\left(\frac{m!}{(m/2)!^2} \right)$ different MSB values, we have:

$$Cases = \left(\frac{m!}{(m/2)!^2} \right) (3^{r-m})(2^{m/2}) \left(\left(\frac{m!}{(m/2)!^2} \right) (2^{r-m}) - \frac{m}{4} - \frac{r-m}{3} \right)$$

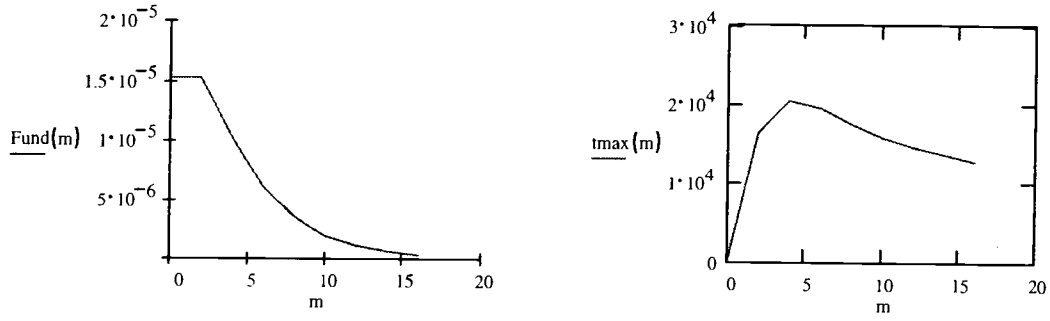
Dividing the number of undetectable cases in a modulo window by the number of unique error cases in a modulo window and simplifying gives the stated result. QED



a) r=8



b) r=12



c) r=16

Figure 4.2 – Fraction of undetectable errors with Bose-Lin codes as a function of the number of code MSBs, *m*, used in the *m*/2-out-of-*m* code

As m increases beyond 4, *Fund* decreases with increasing m for a given r. This means that while m=4 is the optimum value of m for detecting all errors up to t errors, m>4 can be better if >t unidirectional errors are possible. *Fund* as a function of m is shown in Figure 4.2 for three values of r. The choice of m for a given r, therefore, depends on whether the number of errors is expected to occasionally exceed t_{max}. The value of t_{max} decreases much more slowly with increasing m than *Fund* increases with increasing m. Therefore, in general it would be best to choose m=r when it is possible to exceed t_{max}.

4.4 CONCLUSIONS

The Bose-Lin codes retain a high degree of error detecting capability for >t unidirectional errors. The undetectable errors patterns repeat themselves using the same modulo as the code. For >t unidirectional errors, the optimum value of m is not m=4, which maximizes t. Here the optimum value of m is m=r. In a system in which there is a combination of random errors <t and burst error >t, a compromise value can be chosen.

4. 5 REFERENCES

[1] B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

[2] J. M. Berger, "A Note on Error Detecting Codes for Asymmetric Channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.

5. CONCURRENT ERROR DETECTION IN TELECOMMUNCATIONS AND DATA COMMUNICATIONS SWITCH FABRICS USING EFFICIENT CODES

Steven S. Gorshe

Paper to be submitted to the *IEEE Transactions on Communications*

445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331

Abstract: A large portion of modern telecommunications and data communications equipment contains switch fabrics for the cross-connecting or routing of data. As the amount of data transiting these fabrics increases, it becomes increasingly important to detect faults in the fabric, including its associated data paths. Popular error detecting codes include the bit-interleaved parity (BIP) and cyclic redundancy check (CRC) codes. This paper analyses the performance of these popular codes and also analyses the performance of Bose-Lin codes for this application. The Bose-Lin codes are shown to give superior performance to the both the BIP and CRC codes. An extension to the well-known Bose-Lin codes is also discussed which increases its data path fault detecting capability.

5.1 INTRODUCTION

Switch fabrics are an integral part of much data and telecommunications equipment, and increasing data rates make it increasingly desirable to detect faults in these fabrics during normal operation. These switch fabrics can either operate on time division multiplexed (TDM) data streams or packets/cells such as ATM cells. Previous work concentrated on off-line tests for pattern-dependent memory faults [6]. Currently, r-bit bit-interleaved parity (BIP-r) codes are popular for error detection in TDM switches and r-bit cyclic redundancy check (CRC-r) codes are popular for error detection in packet/cell switches. BIP codes are simple to implement in high-speed circuits, however, they are not optimal for this application from a performance standpoint. CRC codes offer good performance, however, as discussed in this paper, the complexity of the code construction limits how they can be used in this application. A different type of error detecting code known as a Bose-Lin code [1] is proposed in this paper that overcomes the

limitations of the BIP and CRC codes, offering superior performance for most cases with reasonable implementation complexity.

The paper begins with a brief review of the BIP-*r*, CRC-*r*, and Bose-Lin codes. Section 3 of the paper discusses the problem of concurrent error detection in switch fabrics, including the implications of the different switch types on this problem. Section 4 discusses the application of the three error detecting codes to switch fabrics and compares the performance and some implementation complexity issues of the codes. The Bose-Lin codes are shown to have superior performance to both the BIP and CRC codes for this application.

5.2 REVIEW OF THE CANDIDATE ERROR DETECTING CODES

Error detection that occurs during the normal operation of a circuit is referred to as concurrent error detection (CED). The error detection codes examined in this paper are systematic, which means that the error check symbol (CS) is appended to the information blocks that it covers, leaving the original information bits unchanged.

5.2.1 BIP-*r* codes

A BIP-*r* code has *r* parity bits. Conceptually, the data in the information word is divided into an integer number of blocks of *r* bits. The first bit of the CS is a parity check over the first bit in each of these blocks. The second CS bit is the parity check over the second bits in each of these blocks, and etc. For example, a typical BIP-2 implementation has one CS bit provide parity over all the even-

numbered bits and the other CS bit providing parity over the odd-numbered bits. An example of a BIP-8 code is shown in [9] where the data is partitioned into octets, and the first CS bit is the parity over the first bit in all data octets, the second CS bit is parity over the second bit in all the octets, etc. BIP-*r* codes have the advantage of being able to detect up to *r* errors in a data word as long as each error occurs in a different partition. BIP codes have the drawback, however, of being unable to detect an even number of errors in the data covered by an individual CS bit.

5.2.2 CRC-*r* codes

A CRC (Cyclic Redundancy Check) code is formed by treating the data block to be covered as a polynomial of the form $m(x)=a_{k-1}x^{k-1}+a_{k-2}x^{k-2}+\ldots+a_1x+a_0$, where m(x) is a *k* bit long message block and a_i is the data value at the ith data position (a_{k-1}=MSB). For a CRC-*r* code, m(x) is divided by the CRC generator polynomial g(x) and the remainder of that division is appended to the end of m(x) as the CRC value such that dividing any such resulting $n=k+r$ bit block by g(x) will result in a remainder that has the same constant value regardless of the original value of m(x). [Note that there are some variations among different CRC techniques regarding exactly how the remainder is formatted to create the CRC CS, however these do not affect the analysis for the purposes of this paper.] At the receiver, the original data is regarded as error-free if the division of the received data block (m(x) and the CRC) yields this constant remainder, since transmission bit errors effectively change the a_i values of the transmitted polynomial. Errors are undetectable whenever the received data block has been changed into another valid code word (i.e., into a polynomial that will give the desired constant remainder of zero when divided by g(x)).

5.2.3 Bose-Lin codes

Bose-Lin codes are optimized for applications with unidirectional errors. Unidirectional error are defined as having all of the errors in the data block covered by a given CS being in the same direction (e.g., 0 to 1). Errors due to faults in integrated circuits are typically unidirectional, which has make unidirectional error detecting codes popular for CED in ICs. Bose-Lin codes have the property that they need only a fixed number of check bits, independent of the number of information bits to be able to detect up to t unidirectional errors.

Bose-Lin codes are formed by taking the count of the number of zeros in the information data bits modulo $(m!/(m/2)!^2) \times 2^{r-m}$. The r-m LSBs of this binary count form the r-m LSBs of the CS. The m MSBs of the modulo count are mapped into an m/2-out-of-m code that forms the m MSBs of the CS. For example, if r=8 with m=4, the zero count is taken modulo 6×2^4. Of the seven bits in the resulting count, the four LSBs are taken directly as the four CS LSBs. The three MSBs of this remainder can take the values {000, 001, 010, 011, 100, 101}. These six values are then mapped to one of the possible 2-out-of-4 codes {0011, 0101, 0110, 1001, 1010, 1100}. The MSB mapping is arbitrary. The versions of the Bose-Lin codes discussed in the literature use m = 0, 2, and 4, since m=4 is optimum for guaranteeing the detection of the maximum number of unidirectional errors. Examples of these codes are shown in Table 5.1. As discussed below, extending the Bose-Lin codes to m>4 can provide better burst error performance.

Table 5.1 – Example check code values for Bose-Lin codes for a data word with 37 zeros.

Code Type	Code Value
Bose-Lin with $m = 0$, $r = 3$	101
Bose-Lin with $m = 2$, $r = 4$	10 01
Bose-Lin with m = 4, r = 5	1100 1

5.3 SWITCH FABRICS

The basic function of a switch fabric is to route data between two different ports on the network. An example switch is shown in Figure 5.1. The two most general ways to characterize a switch fabric are by the type data it switches and the structure of the switch fabric. Each of these variations has a somewhat different implication for error detection.

The two basic traffic types are packets/cells (e.g., ATM cells or Ethernet frames) and TDM streams (e.g., SONET/SDH signals). In the case of TDM stream switches, the switch fabric is sometimes called a time-slot interchange (TSI) matrix since it separates out TDM time-slots (channels) from the incoming data streams and regroups them into different output TDM streams. The example of Figure 5.1 is this type of switch, and is very typical in telecommunications applications. In the case of a packet/cell switch, the packets/cells are taken from

the incoming data streams and are switched into the appropriate outgoing data streams on a packet/cell-by-packet/cell basis.

The two general categories of fabric structures are a crossbar switch or a multi-stage switch network. For TDM signals, crossbar switches are typically implemented with a shared memory such that the incoming data is written into this memory. The switch output control then determines which data is read out of the memory for each outgoing data stream. For example, incoming TDM data is written into memory locations that are typically determined by their time-slots within the incoming streams. A control memory then establishes the read addresses and read sequences that are used to place this data into the outgoing data streams. The size of the memory is usually established such that a convenient amount of data is stored. For TDM switches, this is usually an integer number of TDM frames worth of the incoming data. For packet/cell switches, the amount of memory must take into account overflow situations where packets/cells from multiple input streams are simultaneously destined for the same output data stream. Examples of multi-stage switch networks include Clos, Benes, and Banyan networks. The key feature of a multi-stage switch network is that the fabric consists of data paths with no memory elements.

There are two general approaches to applying error-detecting codes to switch fabrics. The first approach is to apply error-detecting codes to each input grain to the switch and check the CS at the output port. In the case of TDM streams, the input grain would be the channel size to be switched (e.g., STS-1 or VT1.5 from SONET). For packet/cell streams, the grain would be the packet or cell. With this approach, the CS travels over the same data paths as the information block it covers. Depending on the grain size or typical packet/cell size, this first approach could require a significant number of CS bits relative to the information bits.

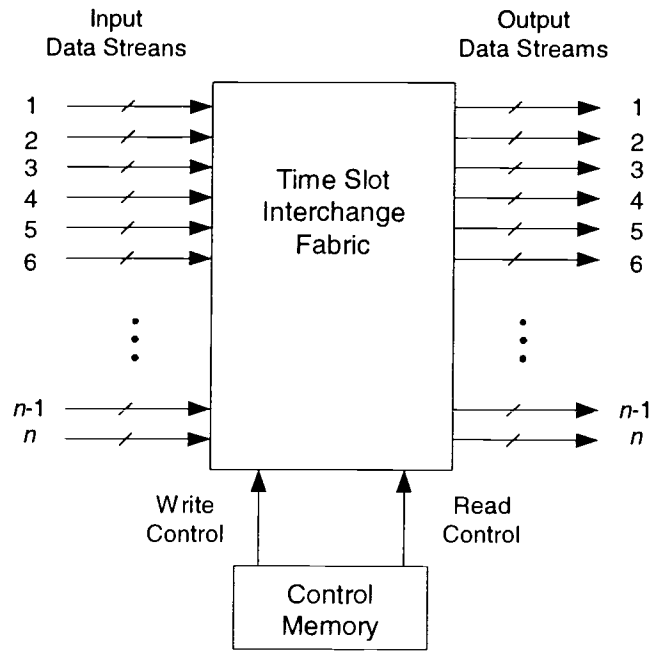


Figure 5.1 – Example of memory-based space-time-space switch fabric

A second approach is to apply the CS across a group of input grains and check the CS across the rearranged outputs. This approach is more efficient in terms of check bits, but is more complicated. For example, it could be applied to a TDM TSI matrix by calculating the CS over all the data in one TDM stream across all input streams. When the output is checked, however, the check must take into account the effects of idle data that is not destined for an output as well as data that is connected to multiple outputs (multi-cast). Another complication for packet/cell switches is that it can be difficult to establish the appropriate boundaries for this check with the corresponding time correlation between the incoming and outgoing data. Since the point of the fabric is to rearrange the data,

a polynomial code like a cyclic redundancy check (CRC) is too difficult to use in this application. In this approach, the CS would typically not travel over the same data paths as the data.

It is assumed in this paper that each memory location is used at most once per data block. Using the same memory cells multiple times per block leads to multiple potential errors resulting from the same fault in the same block. As is apparent from the analysis below, using the memory cells multiple times will degrade the performance of BIP codes more than Bose-Lin or CRC codes.

5.4 CODE PERFORMANCE

Faults within ICs are typically unidirectional, and that is the assumption made in the following analysis. There are two general types of faults that must be considered. The first is a data path fault. The second fault type is a memory element fault. Each of these fault types are treated separately rather than in combination.

5.4.1 Data path fault performance

Data path faults will corrupt many bits of data. For example, a 'stuck-at-0' fault on a data path will cause any data 1 that transits that path to be set to 0. The number of errors is the number of input bits that have the opposite value as the stuck fault. Parallel data paths are typically used in high bit-rate systems in order to keep the data path clock rate at a reasonable value. In a typical data or telecommunications system, the data is oriented around 8-bit bytes or multiples of

bytes. All switch fabrics here are assumed to be implemented using M x 8 bit paths. The r-bit CS also typically uses $r = N$ x 8 bits where N and M are not necessarily the same.

5.4.1.1 BIP-r performance for data path faults

The BIP-r code can be thought of as partitioning the information word into r partitions. A data path fault is undetectable whenever the number of affected information bits in that partition for that check bit is even. For 1 to 0 errors, this means that all odd weight codes in the faulted partition will produce detectable errors. If we assume that even and odd weight codes are equally likely in each partition, then the probability that a fault in a partition is detectable is 1/2. The performance of BIP-r codes with multiple data path faults depends on the ratio of the check code length r to the data path width w. If all of the faults are in data paths covered by the same check bit (i.e., are in the same code partition), then the probability that the fault is detectable is still 1/2. If the faults are spread across multiple partitions, then the probability of detecting the presence of at least one of these faults is higher.

Given the presence of n faults, the probability of at least one being detected is:

$$Pund \mid ndpfault = \sum_{\text{all combinations of } n \text{ in } w} (1/2)^k$$

where k is the number of affected partitions for a given combination of n data path faults.

In the case of up to two data path faults, the total probability of the faults being undetectable can be readily seen to be:

$$Pund = 1 - [\ (Pdpfault)(1 - Pdpfault)^{w-1}(1/2) + (Pdpfault)^2 (1 - Pdpfault)^{w-2}\left[\binom{r}{1}\binom{w/r}{2}(1/2) + \binom{r}{2}\binom{w/r}{1}^2 (1/2)^2\right]\ ] \quad (5.1)$$

where the first term in the inner brackets represents the cases where all faults occur in the same partition, the last term in the brackets represents the case where each of the faults is in a different partition, and *Pdpfault* is the probability of a data path having a fault. In the special case where $r=w$, only a single fault can occur in each partition and

$$Pund|ndpfault = 2^{-n}. \quad (r=w) \quad (5.2)$$

The results of (5.2) are plotted in Figure 5.3 along side the performance of the other codes. The resulting total probability of the faults being undetectable is:

$$Pund\,total = 1 - \left[\sum_{i=1}^{r}\binom{r}{i}(1/2)^i (Pdpfault)^i (1 - Pdpfault)^{r-i}\right] \quad \text{for } r=w$$

$$Pund\,total = 1 - \left[\sum_{i=0}^{r}\binom{r}{i}\left(Pdpfault/2\right)^i (1 - Pdpfault)^{r-i} - (1 - Pdpfault)^r\right]$$

$$Pund\,total = 1 - \left[\left(1 - \frac{Pdpfault}{2}\right)^r - (1 - Pdpfault)^r\right] \quad (5.3)$$

Since *Pdfault* << 1, we can used the approximation $(1-x)^n \approx (1-nx)$ and simplify equation (5.3) to:

$$Pund\,total \approx 1-\left(\frac{(r)(Pdpfault)}{2}\right) \tag{5.4}$$

For the more typical case of $r<w$, there is no convenient closed form solution. Since the probability of data path faults is very small, the two-fault equation above is a reasonable approximation and lower bound for *Pund*. As written, (5.1) assumes that $w \geq 2r$. The *w/r* ratio makes relatively little difference for any *w/r* ratio in the range of interest, with (5.2) and (5.4) forming an upper bound on *Pund|ndpfault* and *Pund total*, respectively, as the *w/r* ratio increases beyond 1.

A drawback of the BIP code relative to the CRC or Bose-Lin codes is that it will typically not be capable of detecting misconnected data paths. For $r=w$, each CS bit travels exactly the same data path as the data it checks so that the CS at the output of the fabric will appear to be correct if no errors or faults have occurred regardless of data path misconnections. For $w>r$, the misconnection detectability is improved, but is still less than would be provided by a CRC or Bose-Lin code.

5.4.1.2 CRC performance for data path faults

For a CRC, the effect of a data path fault is similar to that of a burst error. Since the burst length will be longer than *r* for the cases of interest, the probability of an undetected error will be [2]:

$$Pund|dpfault \approx (2^{-r}) \tag{5.5}$$

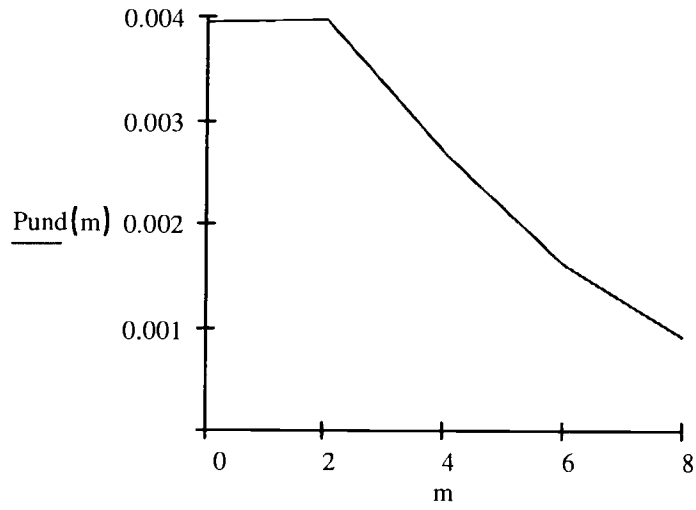
which is essentially constant for any number of data path faults. The results of (5.5) are compared to the other codes in Figure 5.3.

5.4.1.3 Bose-Lin performance for data path faults

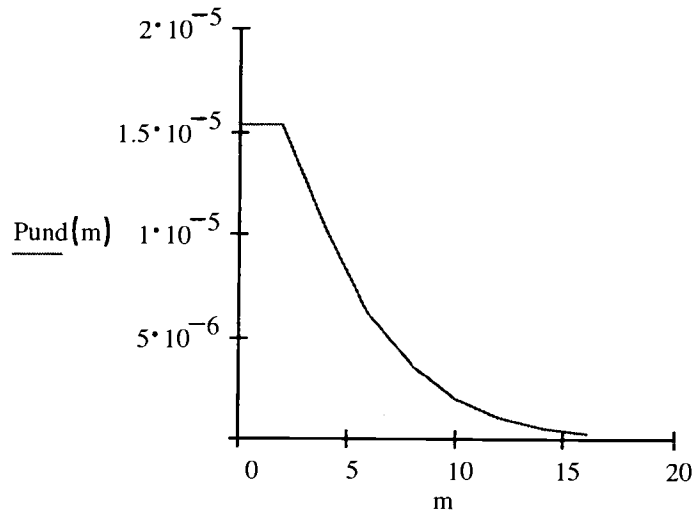
If the number of effected bits is $\leq t$, then the Bose-Lin code will detect the data path fault. For example, with $r = 16$, $m = 4$, the Bose-Lin code will detect up to 20492 unidirectional errors. If the data block is $\leq 20492\text{-}16 = 20476$, then the Bose-Lin code will detect all data path faults regardless of how many paths are affected. On the other hand, assume that the bits covered by an 8-bit Bose-Line code are the same as what is covered by the SONET/SDH B3 BIP-8 (6264 bits). If an 8-bit data path is used, a single data path fault would cause up to 783 errors, which is beyond the codes $t_{max} = 84$. In this case, similar to CRC codes, the effect of a data path fault will look like a burst error. It has been shown [3] that the probability of an undetected unidirectional burst error for Bose-Lin codes is:

$$Pund \leq \frac{1}{(2^{m/2})\left\lfloor\left(\frac{m!}{(m/2)!^2}\right)(2^{r-m})-\frac{m}{4}-\frac{r-m}{3}\right\rceil} \tag{5.6}$$

As seen in Figure 5.2, while $m = 4$ is optimum from the standpoint of t_{max} [1], *Pund* for burst errors decreases with increasing m. The results of (5.6) are plotted in Figure 5.3 alongside the plots for the BIP and CRC codes.

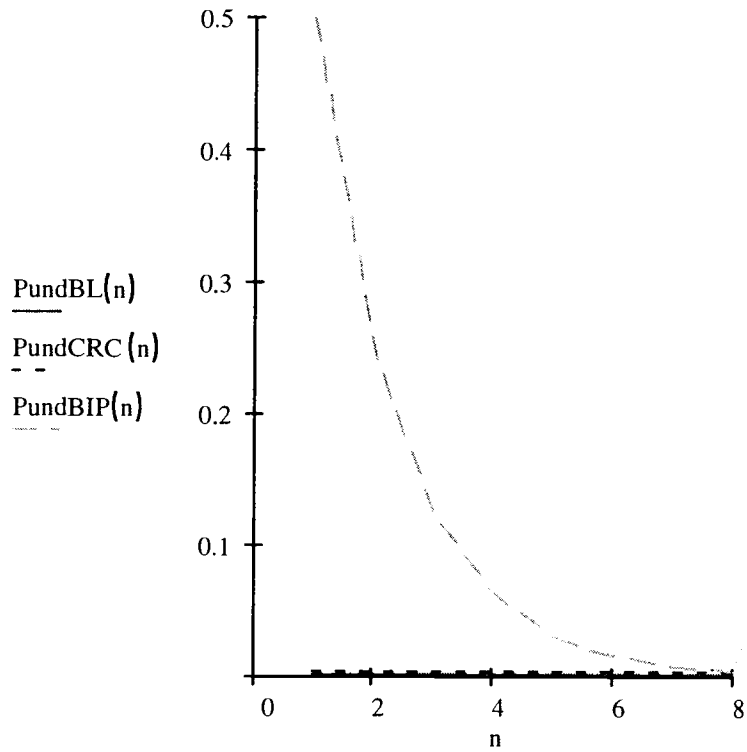


a) $r = 8$



b) $r = 16$

Figure 5.2 – Burst error performance of Bose-Lin codes as a function of the number of MSBs m used in the code construction given that a burst error (e.g., data path fault) has occurred.



$PundBIP(n)min = 3.9 \times 10^{-3}$

$PundBL(n) = 1.6 \times 10^{-3}$

$PundCRC(n) = 3.9 \times 10^{-3}$

(Note: For 16 bit codes (m=10 for Bose-Lin):

$PundBIP(n)min = 1.5x10^{-5}$, $PundBL(n) = 1.9x10^{-6}$,

and .$PundCRC(n) = 1.5x10^{-5}$.)

Figure 5.3 – Comparison of BIP-8, CRC-8, and 8-bit Bose-Lin (m=6) codes for data path faults where Pundx(n) is the probability of the fault presence being undetectable given that there are n faults for code x

5.4.2 Memory fault performance

The memory faults will, of course, only occur in those switch fabrics that use memory for either the crossbar fabric or for buffering. A switch network can be implemented without memory elements (i.e., as a pure data path). Even if memory is used for input or output buffers, this memory will look like part of the data path if it is implemented as shift registers. This analysis assumes a memory-based crossbar switch. The memory cell faults are assumed to be independent events rather than a block occurrence.

5.4.2.1 BIP-*r* performance for memory faults

Let N be the total number of bits in the block, including the check symbol (i.e., $N = I+r$), and let n be the number of errors due to a fault. Within each code partition, the number of vulnerable bits (assuming 0 to 1 faults and $p0$ and $p1$ being the respective probabilities of a 0 or 1 occurring in the information word from the information source) is:

$$Pnvulnerable = \sum_{k=n}^{N/r} \binom{N/r}{k} p0^k \, p1^{\frac{N}{r}-k} \binom{k}{n} = \sum_{k=n}^{N/r} \binom{N/r}{n} \binom{N/r - n}{k-n} p0^k \, p1^{\frac{N}{r}-k}$$

where k is the number of zeros in the information word. (For example, a bit is vulnerable to a 0 to 1 error iff that bit is originally a 0.) This equation reduces to:

$$Pnvulnerable = (p0)^n \binom{N/r}{n} \tag{5.7}$$

Similarly, the vulnerable bits across all partitions is:

$$Pnvulnerable = \sum_{k=n}^{N} \binom{N}{k} p0^{k}\, p1^{N-k} \binom{k}{n} = (p0)^{n} \binom{N}{n} \tag{5.8}$$

Given n errors due to faults:

$$Pund \mid nerrors = \frac{\left\lceil \begin{array}{c} \text{All combinations with an even number} \\ \text{of errors in each affected partition (totalling } n\text{)} \end{array} \right\rceil}{\left\lceil \begin{array}{c} \text{All combinations of } n \text{ errors} \\ \text{across all partitions} \end{array} \right\rceil}$$

For example,

$$Pund \mid 2errors = \binom{r}{1}\binom{N/r}{2} p0^{2} \Big/ \binom{N}{2} p0^{2} = (r)\binom{N/r}{2} \Big/ \binom{N}{2} \tag{5.9}$$

$$\begin{aligned} Pund \mid 4errors &= \left\lceil \binom{r}{1}\binom{N/r}{4} + \binom{r}{2}\binom{N/r}{2}^{2} \right\rceil p0^{4} \Big/ \binom{N}{4} p0^{4} \\ &= \left\lceil (r)\binom{N/r}{4} + \binom{r}{2}\binom{N/r}{2}^{2} \right\rceil \Big/ \binom{N}{2} \end{aligned} \tag{5.10}$$

If the fault probability is small, then *Pund* can be approximated as the sum of (5.9) and (5.10). *Pund* is plotted in Figure 5.5 as a function of the given number of errors with all terms taken into account in the example.

The total probability for all numbers of errors can be determined as follows. The probability of having n 0 to 1 errors resulting from faults in a single partition is:

$$Pne/part = \sum_{k=n}^{N/r} \binom{N/r}{k} p0^k \, p1^{N/r-k} \left[\binom{k}{n} pe^n (1-pe)^{k-n} \right] \tag{5.11}$$

where pe is the probability of a 0 to 1 fault, and k is the number of zeros in the partition. Equation (5.11) can be simplified as follows:

$$Pne/part = \left(\frac{pe}{1-pe} \right)^n \sum_{k=n}^{N/r} \binom{N/r}{k} \binom{k}{n} [p0(1-pe)]^k \, p1^{N/r-k}$$

$$Pne/part = \left(\frac{pe}{1-pe} \right)^n \binom{N/r}{n} \sum_{k=n}^{N/r} \binom{N/r-n}{k-n} [p0(1-pe)]^k \, p1^{\frac{N}{r}-k}$$

$$Pne/part = \left(\frac{pe}{1-pe} \right)^n \binom{N/r}{n} \sum_{k=0}^{N/r-n} \binom{N/r-n}{k} [p0(1-pe)]^{k+n} \, p1^{N/r-n-k}$$

$$Pne/part = (p0\,pe)^n \binom{N/r}{n} (1-p0\,pe)^{N/r-n} \tag{5.12}$$

The probability of an undetectable error in a given partition is:

$$Pund/part = \sum_{i=even}^{N/r} \binom{N/r}{n} (p0pe)^i (1-p0\,pe)^{N/r-i} \tag{5.13}$$

The total undetectable error probability across all partitions becomes:

$$Pund\,tot = \left\lceil \sum_{i=even}^{N/r} \binom{N/r}{n} (p0pe)^{i} (1-p0\,pe)^{N/r-i} \right\rceil^{r} - (1-p0pe)^{N} \qquad (5.14)$$

5.4.2.2 CRC performance for memory faults

The limit for a CRC-8 is that it is only guaranteed to detect a single error for blocks of over 255 bits [4], and all of the blocks of interest will be larger than that in the TDM cases. A CRC-16 can be designed to detect up to 3 errors for blocks in the range of interest [4], [5]. Beyond 3 errors, we must again assume that *Pund* $\approx 2^{-r}$. Overall, we would have the approximation:

$$Pund|n\ errors \approx (2^{-r}) \qquad (5.15)$$

The CRC generator polynomial can be chosen so that the probability of undetectable four-error patterns is better than 2^{-r}, so this approximation is somewhat rough but will be in the right order of magnitude. A plot of a CRC-8 performance as a function of the given number of errors is shown in Figure 5.5.

5.4.2.3 Bose-Lin performance for memory faults

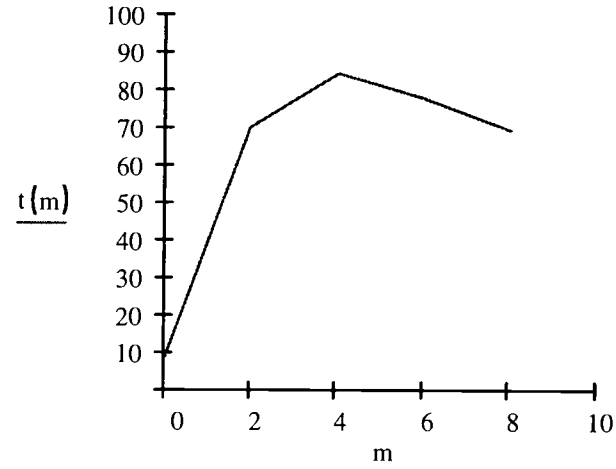
Independent of the information word block length, the Bose-Line code can detect up to *t* errors [1] where:

$$t = \left\lfloor \left(\frac{m!}{\left(m/2\right)^2} - 1 \right) \left(2^{r-m}\right) \right\rfloor + r - m \qquad (5.16)$$

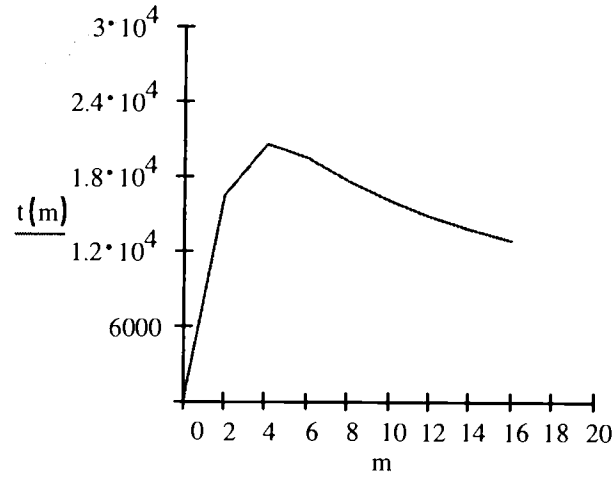
As illustrated in Figure 5.4, an 8-bit Bose-Lin code can detect up to 84 memory faults if it is optimized for t_{max} (i.e., with $m = 4$). If $m = 8$ is chosen to optimize the burst error performance, the resulting is still a $t_{max} = 69$. For a 16-bit code, $m = 4$ and $m = 16$ will give a t_{max} of 20492 and 12869, respectively. Therefore, for memory faults:

$$Pund|n\ errors \approx 0$$

for n in the range of interest.



a) $r = 8$



b) $r = 16$

Figure 5.4 – The maximum number of unidirectional errors t that are guaranteed detectable by Bose-Lin codes as a function of m

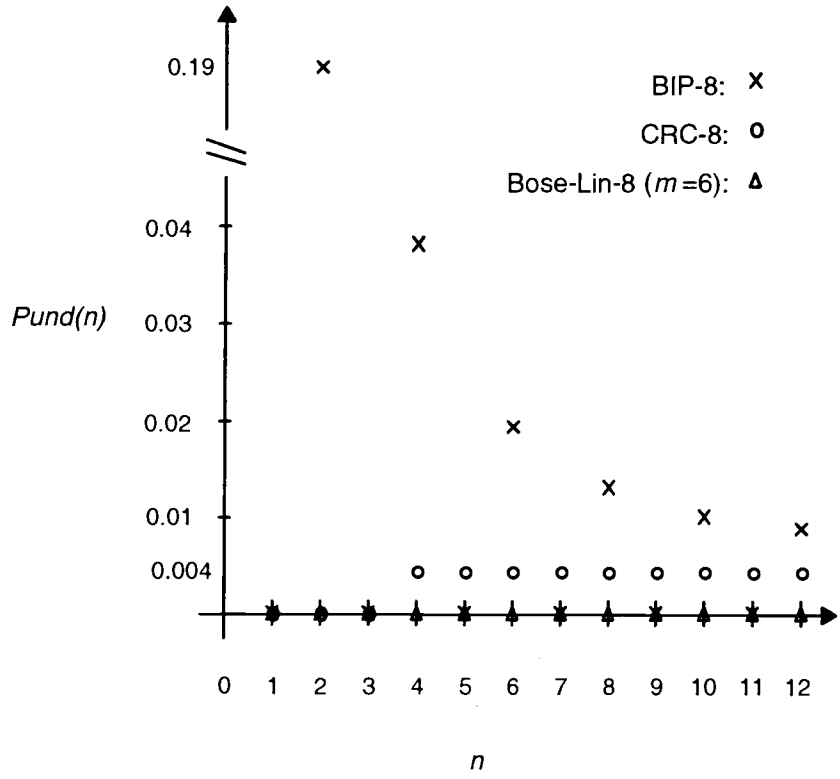


Figure 5.5 – Comparison of memory fault performance for BIP-8, CRC-8, and 8-bit Bose-Lin (m=6) codes with N=96 as a function of the given number of errors, n, resulting from memory cell faults

5.4.3 Comparison summary

For data path faults, the BIP-r code offers the worst performance. The total probability of undetectable errors decreases linearly with increasing r. As a function of the given number of errors, the BIP code *Pund* equals that for the CRC only there is a fault in every data path, and is never better than the predicted Bose-Lin performance. The CRC code performs much better with the probability of any data path fault event being undetectable being 2^{-r}. Bose-Lin code provides the

best performance for data path faults, outperforming the BIP and CRC codes for all data path fault combinations.

For memory faults, the BIP codes again offer the worst performance for low probability of errors. As is illustrated in Figure 5.5, if the given number of errors is odd, then the BIP is guaranteed to detect the presence of a fault. If the number of errors is even, then the BIP appears to converge on $Pund = 1/2^{r-1}$. The CRC code has a low limit on the number of detectable errors. The Bose-Lin codes offer full fault error detectability for the numbers of faults that would be expected in practical situations. (For higher numbers of faults, the memory faults look like burst errors where the relative performance of the codes would be similar to the data path fault cases.) The Bose-Lin code performance can be optimized for this combined data path and memory fault application by choosing a compromise value for m. For example, m=6 would be a good choice for r=8, and m=10 would be a good choice for r=16.

5.5 CONCLUSIONS

For switch fabrics, the affects of both data path and memory cell faults must be taken into account in determining the integrity of the fabric. Within an integrated circuit, faults tend to cause unidirectional errors. The Bose-Lin codes, which are optimized for unidirectional error performance offer superior performance to the popular BIP and CRC codes for concurrent error/fault detection in switch fabrics for both data path and memory faults. In order to optimize the Bose-Lin codes for the type of burst errors that would result from a data path fault, a generalized version of the codes with $m > 4$ should be used.

5.6 REFERENCES

[1] B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

[2] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, NJ: Prentice-Hall, 1995

[3] S. Gorshe, Ph.D. Dissertation – Oregon State University, Corvallis, Oregon, 2002

[4] D. Bertsekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ, Prentice-Hall, 1987

[5] G. Castagnoli, J. Ganz, and P. Graber, "Optimum Cyclic Redundancy-Check Codes with 16-Bit Reduncancy," *IEEE Trans. Commun.*, vol. 38, pp. 111-114, Jan. 1990.

[6] J. Tyszer, "Test Generation for Pattern-Sensitive Faults in Integrated Switches," *IEEE Trans. Commun.*, vol. 39, pp. 1546-1548, Nov. 1991.

6. ANALYSIS OF THE INTERACTION BETWEEN CRC ERROR DETECTING POLYNOMIALS AND SELF-SYNCHRONOUS PAYLOAD SCRAMBLERS

Steven S. Gorshe

Paper submitted to the *IEEE Transactions on Communications*

(Preliminary version published in the *Proceedings of the IEEE ICC2002* conference)

445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331

Abstract: In order to protect public network data transmission from potential Layer 1 attacks by malicious users, self-synchronous scramblers have come into widespread use. Such networks include those using ATM, Packet over SONET (POS), and the new Generic Framing Procedure (GFP). Unfortunately, feedback taps inherent in self-synchronous descramblers cause multiplication of transmission errors, which in turn degrades the performance of most popular CRC error check codes. This paper analyzes this scrambler/CRC interaction with respect to the resulting probability of undetectable errors and single transmission error correction capability and establishes the theoretical criteria required for a CRC to maintain its performance in the presence of the scrambler. These theoretical results are extended for the general case of a t-error correcting linear, cyclic code (e.g., BCH or Reed Solomon) in the presence of a self-synchronous scrambler. Some CRC-16 codes are also presented that are optimized for these applications.

6.1 INTRODUCTION

As LAN data traffic is increasingly being carried over public WANs, issues arise concerning the potential harm data from one subscriber could cause to data from other subscribers. As will be discussed in more detail in section 2 of this paper, this issue has lead to the use of self-synchronous payload scramblers for protocols such as ATM, Packet over SONET/SDH (POS), and Generic Framing Procedure (GFP). The drawback to self-synchronous scramblers is that the descrambling process will multiply errors that have occurred during transmission, which in turn can decrease the effectiveness of a Cyclic Redundancy Check (CRC) error code over the payload data. After some further background on self-synchronous scramblers, this paper provides a general analysis of the interaction between self-

synchronous scramblers and CRC polynomials and *t*-error correcting linear cyclic codes (e.g., BCH or Reed-Solomon) in general. This analysis establishes the theoretical criteria required for the code to maintain its overall probability of undetectable errors and its error correction capability in the presence of a self-synchronous scrambler. In order to address these interactions issues, a new class of CRC-16 code has been identified that meets these criteria. It will also be shown that for several important applications where packet lengths are known to be relatively short (e.g., <1000bits), these new CRC-16 polynomials provide substantially better performance than existing standard CRC-16 polynomials. The preliminary results of this research were shown in [11].

6.2. SCRAMBLERS AND THEIR INTERACTION WITH CRCS

6.2.1 Background on self-synchronous scramblers

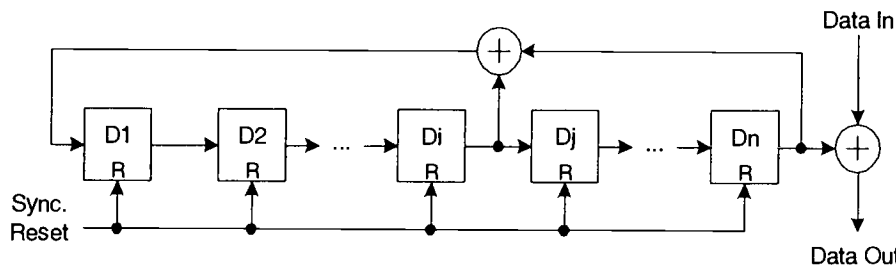
The transmission equipment that forms the backbone of the public telephone network (i.e., SONET/SDH) uses a NRZ line code. The critical advantage of the NRZ line code is that it makes the most efficient use of the channel bandwidth of any baseband line code, and is very simple to implement. The main drawback to NRZ is that if there is no transition between the values of the bits in the transmitted data, there is no change in the level of the transmitted signal. The receiver relies on these transitions to synchronize its clock/data recovery circuit for determining the boundaries of the individual bits. During a long period with no line code transitions, the relative clock differences between the transmitter and receiver can cause the receiver to mis-sample the incoming data stream. The solution used in SONET/SDH is to scramble the data with a frame-synchronized scrambler [1]. A

frame-synchronized scrambler, as illustrated in Figure 6.1.a, is one in which the transmitted data is exclusive-ORed bit-by-bit with the output of a pseudo-random sequence generator, with the sequence generator being reset to a know state at the beginning of every frame. The frame-synchronized scramblers are very effective in increasing the transition density to an acceptable level for typical traffic. One drawback of a frame-synchronized scrambler is that it is a known, relatively short (2^7-1) pseudo-random sequence and it is possible for a malicious subscriber to attempt to mimic this pattern within the data he sends. The result is that if the subscriber data lines up with the SONET/SDH scrambler correctly, a long string can occur with no transitions, which in turn can cause the receiver to fail. This phenomenon was observed with early ATM and POS systems and was addressed from the outset with GFP. The solution used for each of these three protocols is a self-synchronous scrambler over the payload region of the cell/frame.

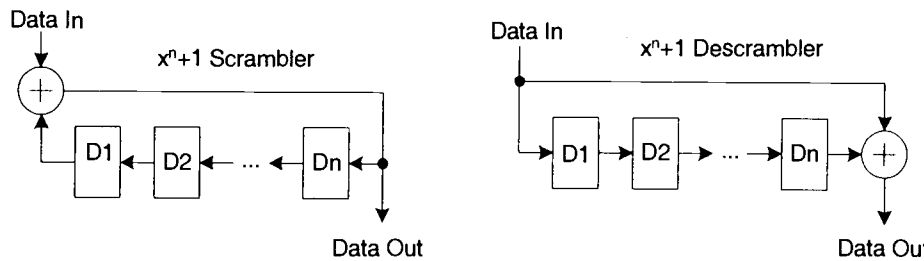
A self-synchronous scrambler, as illustrated in Figure 6.1.b, is one in which the data is exclusive-ORed with a delayed version of itself on a bit-by-bit basis, which is effectively a GF[2] division process. The specific scrambler used for ATM, POS, and GFP exclusive-ORs the input data with scrambler output data after a 43-bit delay [2], [10]. In other words, they use an $x^{43}+1$ scrambler polynomial. The descrambler reverses the process by multiplying the received signal by the same scrambler polynomial. The advantage to such a scrambler in this application is that it is very hard for a malicious user to duplicate due to its never having a known reset point. The value of the scrambler state is function of the previous data rather than the position of the data within the SONET/SDH frame. The drawback to a self-synchronous scrambler is that any errors occurring on the transmission channel will be duplicated 43 bits later by the descrambler. As a result, an error check code over the data will have to deal with twice the bit error rate as that experienced by the transmission channel.

6.2.2 Interaction between self-synchronous scramblers and CRCs

A CRC code is formed by treating the data block to be covered as a polynomial of the form $m(x)=a_{k-1}x^{k-1}+a_{k-2}x^{k-2}+\ldots+a_1x+a_0$, where m(x) is a *k* bit long message block and a_i is the data value at the ith data position (a_{k-1}=MSB). For a CRC-*r* code, m(x) is divided by the CRC generator polynomial g(x) and the remainder of that division is appended to the end of m(x) as the CRC value such that dividing any such resulting $n=k+r$ bit block by g(x) will result in a remainder that has the same constant value regardless of the original value of m(x). [Note that there are some variations among different CRC techniques regarding exactly how the remainder is formatted to create the CRC. For a typical CRC-16 application, the division by g(x) is performed on $x^{16}m(x)$ (i.e., m(x) shifted left by 16 places with zero fill). The remainder is then appended to the LSB end of the m(x) as the CRC code. As a result, the resulting $k+r$ data pattern is always divisible by g(x) and gives a constant remainder of 0.] At the receiver, the original data is regarded as error-free if the division of the received data block (m(x) and the CRC) yields this constant remainder, since transmission bit errors effectively change the a_i values of the transmitted polynomial. Errors are undetectable whenever the received data block has been changed into another valid code word (i.e., into a polynomial that will give the desired constant remainder when divided by g(x)).



a) Frame synchronized scrambler example



b) Self-synchronous scrambler example

Figure 6.1 – Scrambler examples

The transmission errors occurring during the transmission of a data block can be represented by the polynomial e(x). If the transmitted data block is represented by n(x) (m(x) and the CRC), then the received data block r(x)=n(x)+e(x). As noted above, error pattern e(x) is only undetectable when r(x)/g(x) leads to the desired

constant value. Without loss of generality, the subsequent analysis of this paper assumes that the CRC is implemented as typical for CRC-16s in which the remainder of n(x)/g(x)=0. Then, the remainder of r(x)/g(x) is the remainder of [n(x)/g(x)+e(x)/g(x)], which implies that an error pattern is only undetectable if the remainder of e(x)/g(x)=0. In mathematical representation, the remainder of a(x)/b(x) can be written as a(x)modb(x). The following error detection analysis is a generalization of the work of [3], which addressed the specific case of the interaction of the Ethernet CRC-32 polynomial with the SONET/SDH $x^{43}+1$ self-synchronous scrambler, and a further generalization of the work in [11].

Since the descrambling process for a self-synchronous scrambler effectively multiplies the received data polynomial r(x) by the scrambler polynomial s(x), if we let r'(x) be the descrambled data block, then:

$$r'(x) = r(x)s(x)).$$

Since r(x) = n(x)+e(x), we have

$$r'(x) = s(x)[n(x)+e(x)] = s(x)n(x) + s(x)e(x)$$

$$\begin{aligned} r'(x)\text{mod}g(x) &= [s(x)n(x) + s(x)e(x)]\text{mod}g(x) \\ &= 0 + [s(x)e(x)]\text{mod}g(x). \end{aligned}$$

Consider a case where s(x) and g(x) have a common factor, f(x) of degree *z*. Then, letting a(x) = s(x)/f(x) and b(x) = g(x)/f(x), we have [s(x)e(x)]modg(x) = [a(x)e(x)]modb(x). This common factor effectively reduces the degree of the CRC by z, giving a performance equivalent to a CRC-(*r*-*z*) polynomial. On the average, the probability of undetectable errors for a CRC-r code is $1/2^r$ for error magnitudes

beyond what is guaranteed detectable. The common s(x), g(x) factor therefore increases the undetectable error probability by a factor of 2^z, if the transmission errors and the multiplied errors are all contained within r'(x). [It should be noted here that the CRC-32 used for Ethernet is a primitive polynomial, and therefore has no common factors with any typical self-synchronous scramblers.]

Theorem 6.1 – The overall probability of undetectable burst errors is unchanged by the descrambler as long as for CRC generator polynomial g(x) and scrambler polynomial s(x), gcd(g(x),s(x))=1.

Proof: As shown in Figure 6.2, there are six cases resulting from the error multiplication due to the descrambling process. These cases can be analyzed as follows. It is assumed in this analysis that $s(x)$ and $g(x)$ share no common factors, or equivalently, that their greatest common divisor is 1, written as gcd($s(x)$, $g(x)$)=1.

Case a): All the transmission errors are contained within the one block and the multiplied errors are in another block.

In this case, there is no error multiplication in this block and there is no change in the CRC's error detection capability.

Case b): Both the original transmission errors and the multiplied errors are contained within the same block.

The errors in this case are the original error polynomial multiplied by the scrambler polynomial, i.e., e(x)s(x). Since s(x) is chosen by design such

that gcd(s(x),g(x))=1, the errors are only undetectable when g(x) divides e(x) (i.e., e(x)modg(x)=0), and hence there is no change to the error detecting capability.

Case c): The transmission errors occurred in the previous block and all of the multiplied errors are in the current block.

Here the errors that are present in the current block are the error polynomial multiplied by the scrambler polynomial minus the original errors. (The original errors can be thought of as the error polynomial times the scrambler polynomial's "1" term.) These errors are detectable as long as the product (e(x)[s(x)-1])modg(x)≠0. If s(x)-1 is chosen by design to have no common factors with g(x), these errors will only be undetectable if e(x) is divisible by g(x), and hence there is no change in the error detecting capability.

Case d): The transmission errors are split between the current and previous blocks with all the multiplied errors in the current block.

The error pattern in this case is $h(x)s(x) + j(x)[s(x)-1] = s(x)[h(x)+j(x)] - j(x) = s(x)e(x) - j(x)$, since $e(x) = j(x) + h(x)$. Hence, errors are only undetectable when [s(x)e(x)-j(x)]modg(x)=0. Since we have chosen s(x) such that s(x)modg(x)≠0, there are the following four resulting subcases.

1) Both e(x)modg(x)=0 and j(x)modg(x)=0 – The probability of undetectable errors here is $\leq (2^{-r})(2^{-r})$ (i.e., the probability of occurrence for this sub-case). The inequality here is due to the fact that the degree of j(x) is less than the degree of e(x), and is thus less likely to be divided by g(x).

2) e(x)modg(x)=0 and j(x)modg(x)≠0 – The probability of undetectable errors here is 0.

3) j(x)modg(x)=0 and e(x)modg(x)≠0 – The probability of undetectable errors here is 0.

4) e(x)modg(x)≠0 and j(x)modg(x)≠0 – Errors are undetectable here if [e(x)+j(x)]modg(x)=0. The resulting probability of undetectable errors is

$(1-2^{-r})(2^{-r})$ (i.e., the probability of e(x)modg(x)≠0 and j(x) having a value that meets the [e(x)+j(x)]modg(x)=0 criteria).

Summing for all four sub-cases gives a total of 2^{-r}, and hence there is no overall change in the probability of undetectable errors.

Case e): The transmission errors are all contained within the current block and the multiplied errors are split between the current block and the subsequent block.

The error pattern in this case is $j(x)s(x) + h(x) = e(x) + j(x)[s(x)-1] = e(x) + j'(x)$, and errors are undetectable when $[e(x)+j'(x)] \bmod g(x)=0$. As with Case d, there are four resulting sub-cases.

1) Both $e(x) \bmod g(x)=0$ and $j'(x) \bmod g(x)=0$ – The probability of undetectable errors here is $\leq (2^{-r})(2^{-r})$. The inequality here is due to the fact that the degree of $j(x)$ is less than the degree of $e(x)$, and is thus less likely to be divided by $g(x)$.
2) $e(x) \bmod g(x)=0$ and $j'(x) \bmod g(x)\neq 0$ – The probability of undetectable errors here is 0.
3) $j'(x) \bmod g(x)=0$ and $e(x) \bmod g(x)\neq 0$ – The probability of undetectable errors here is 0.
4) $e(x) \bmod g(x)\neq 0$ and $j'(x) \bmod g(x)\neq 0$ – Errors are undetectable here if $[e(x)+j'(x)] \bmod g(x)=0$. The resulting probability of undetectable errors is

   $(1-2^{-r})(2^{-r})$.

Summing for all four sub-cases gives a total of 2^{-r}, and hence there is no overall change in the probability of undetectable errors.

Case f): The transmission errors and the multiplied error both straddle the boundaries of the block and an adjacent block.

The error pattern in this case is $h(x) + h'(x) = e(x)s(x) - k(x)$, where $k(x) = j(x) + j'(x)$. The analysis is not as straightforward here since $h(x) + h'(x)$ can either lead to a higher weight error polynomial than $e(x)[s(x)-1]$ or a lower weight error polynomial due to terms in $h(x)$ canceling terms in $h'(x)$. The resulting error polynomial has lost much of its correlation with $e(x)$. In any case, we know that the length of the error burst within the block is shorter than $e(x)[s(x)-1]$. Here there are two sub-cases:

1) $e(x) \bmod g(x) = 0$ – Here the probability of $[h(x)+h'(x)] \bmod g(x) = 0$ is (2^{-16}), so there is no change to the undetected error probability.
2) $e(x) \bmod g(x) \neq 0$ – Here the probability of $[h(x)+h'(x)] \bmod g(x) = 0$ remains (2^{-16}).

The total error probability is $(2^{-r})(2^{-r}) + (1-2^{-r})(2^{-r}) = (2^{-r})$ as before, so even though the specific undetectable error patterns change, there is no overall change in the probability of undetectable burst errors.

QED

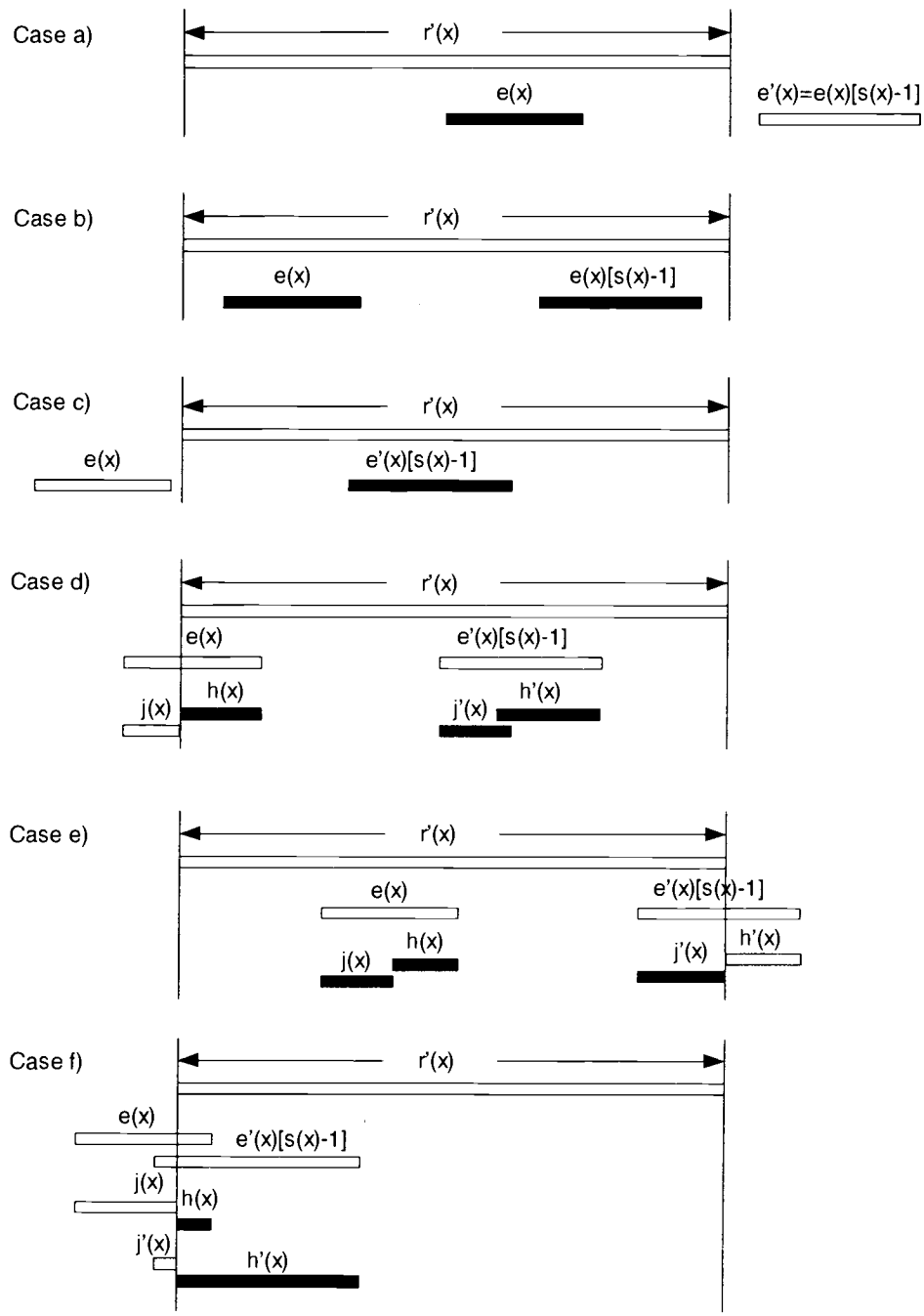


Figure 6.2 – Error multiplication cases resulting from descrambling

It can be seen, however, that the larger the weight and degree of s(x), the more likely it is that an error pattern small enough to be guaranteed as detectable with no scrambler will lead to an undetectable error. For example, consider a case with g(x) = 100101011 (a degree 8 primitive polynomial that is assumed in this example to provide d_{min}=4 over the block size if interest), s(x) = 1010101, and e(x) = 1011 (which would be detectable due to the d_{min}=4 assumption). e(x)s(x) = 1001010111, so if things line up such that the LSB of e(x)s(x) is in the next block, then g(x) divides the resulting 100101011 error pattern and the three transmission error pattern is undetectable.

Note that the scrambler polynomial is assumed to be sufficiently shorter than the block length to preclude the case where both the transmission errors and the multiplied errors straddle block boundaries.

Hence, the overall undetectable error probability is not changed as long as the scrambler and the CRC generator polynomial have no common factors, so the overall code performance is unchanged. It can be seen from case (f), however, that the best choice for s(x) will be one with low weight were the spacing between the terms/taps is longer than the expected burst error length so that there will be no overlap between the original e(x) error burst pattern and the multiplied e(x)[(s(x)-1] pattern. The common x^m+1 scramblers with reasonably large *m* are ideal from this standpoint.

6.2.3 Criteria for error detection and correction

As has been shown, the CRC will only maintain its error detecting performance in the presence of the self-synchronous scrambler if g(x) and s(x) have no common factors. In order to minimize the implementation complexity as well as error multiplication, self-synchronous scramblers are typically implemented with a single feedback tap (i.e., use an $s(x) = x^m+1$ polynomial). As noted earlier, the SONET/SDH payload scrambler uses an $x^{43}+1$ scrambler polynomial. Any polynomial of the form x^m+1 has x+1 as a factor. Unfortunately, since this x+1 factor guarantees that a CRC will be capable of detecting all odd numbers of errors [4], most of the popular standard CRC polynomials, including all the popular CRC-16s, also have x+1 as a factor. The challenge for data that may be transported over SONET/SDH, then, is finding a g(x) that is single, double, and triple error detecting (i.e., has a minimum Hamming distance of $d_{min}=4$) without any common factors with the $x^{43}+1$ scrambler polynomial.

It is worthwhile reviewing the error detecting criteria for CRC polynomials at this point. A g(x) is single error detecting if it has more than one term. As noted in [4], g(x) can be guaranteed to be double error detecting for a block of length n as long as it contains a factor that is primitive of degree of at least $\log_2 n$. This consequence follows from the definition of a primitive polynomial, which is an irreducible polynomial for which, if the polynomial has degree m, the smallest x^n+1 polynomial that it divides is for $n=2^m-1$. Since a double error pattern in which the errors are k bits apart is represented by an $e(x)=x^k+1$, we know that the remainder of e(x)/g(x) can not be zero as long as g(x) contains a primitive factor of degree at least $\log_2 k$. As long as $k>n$, we can not have two errors in the same block and still have g(x) divide e(x). The challenge comes for triple error detecting. The x+1 factor is popular for the obvious reason that it provides an

easy method to guarantee triple error detection. Since the x+1 factor cannot be used for an application with a x^m+1 scrambler, different polynomials had to be tested to determine the smallest degree e(x) for which g(x) divides e(x).

A further desired criterion for g(x) is that it allows single transmission error correction. A CRC-*r* is capable of single error correction for blocks of up to 2^r-1 bits as long as it maintains a $d_{min} \geq 3$ over this range. In general, in order to perform error correction a code must produce unique syndromes for each error pattern. (A syndrome for a CRC is the remainder produced by the division of the received block, which is [r'(x)]modg(x) here.)

Theorem 6.2 – Single error correction is possible with CRC generator polynomial g(x) and a scramble polynomial of the form $s(x)=x^m+1$ as long as gcd(g(x), s(x))=1 and $d_{min} \geq 4$ over the block size *n*.

Proof: Single error detection is possible if each of the following criteria are met, which are the mathematical statement for the required uniqueness for each syndrome:

1. $x^i \text{mod} g(x) \neq x^j \text{mod} g(x)$ for all $i \neq j \leq n$
2. $[(x^i)(x^m+1)] \text{mod} g(x) \neq [(x^j)(x^m+1)] \text{mod} g(x)$ for all $i \neq j \leq n$
3. $x^i \text{mod} g(x) \neq [(x^j)(x^k+1)] \text{mod} g(x)$ for all i and $j \leq n$.

Assume an equality for each criterion, and then determine the criteria under which a contradiction occurs.

Criterion 1: $(x^i + x^j) \bmod g(x) = 0$

$[(x^i)(1 + x^k)] \bmod g(x) = 0$ for $j = i + k$.

But, since $\gcd(x^i, g(x))=1$ and $\gcd(1+x^i, g(x))=1$ due to g(x) providing $d_{min} \geq 4$, criterion 1 is met.

Criterion 2: $[(x^m+1)(x^i+x^j)] \bmod g(x) = 0$

$[(x^m+1)(x^i)(1+x^k)] \bmod g(x) = 0$ for $j = i + k$

But, since $\gcd(x^i, g(x))=1$ and $\gcd(1+x^i, g(x))=1$ due to g(x) providing $d_{min} \geq 4$, criterion 2 is also met.

Criterion 3: $[x^i+(x^j)(1+x^m)] \bmod g(x) = 0$

which gives:

$[(x^j)(1+x^k+x^m)] \bmod g(x) = 0$ for $i = j + k$ and

$[(x^i)(1+x^k+x^{k+m})] \bmod g(x) = 0$ for $j = i + k$

Again, $\gcd(x^i, g(x))=1$. Since $1+x^k+x^m$ and $1+x^k+x^{k+m}$ have a weight of three, we are guaranteed that $\gcd(1+x^k+x^m, g(x))=1$ and $\gcd(1+x^k+x^{k+m}, g(x))=1$ for our assumed $d_{min} \geq 4$.

QED

The proof of theorem 2 is for the strong case in which all possible single errors and all possible descrambled errors within the block have unique polynomials. A weaker case is also possible as long the data is always guaranteed to pass through a scrambler/descrambler, since in this case the only single errors of concern will occur within k bits of the boundaries of the n bit block. Theorem 2 and its proof can be further generalized as follows:

Theorem 6.3 – Single error correction is possible with CRC generator polynomial g(x) and a scramble s(x) as long as gcd(g(x), s(x))=1 and $d_{min} \geq (2 + wt(s(x)))$ over the block size n, where wt(s(x)) is the weight of the polynomial s(x).

Proof: Single error detection is possible if each of the the following criteria are met, which are the mathematical statement for the required uniqueness for each syndrome:

1. $x^i \text{mod} g(x) \neq x^j \text{mod} g(x)$ for all $i \neq j \leq n$
2. $[(x^i)(s(x))]\text{mod} g(x)$ $[(x^j)(s(x))]\text{mod} g(x)$ for all $i \neq j \leq n$
3. $x^i \text{mod} g(x) \neq [(x^j)(s(x))]\text{mod} g(x)$ for all i and $j \leq n$.

Assume an equality for each criterion, and then determine the criteria under which a contradiction occurs.

Criterion 1: $(x^i + x^j) \bmod g(x) = 0$

$[(x^i)(1 + x^k)] \bmod g(x) = 0$ for $j = i + k$.

But, since $\gcd(x^i, g(x))=1$ and $\gcd(1+x^i, g(x))=1$ due to g(x) providing $d_{min} \geq 2$, criterion 1 is met.

Criterion 2: $[(s(x))(x^i+x^j)] \bmod g(x) = 0$

$[(s(x))(x^i)(1+x^k)] \bmod g(x) = 0$ for $j = i + k$

But, since $\gcd(x^i, g(x))=1$, $\gcd(1+x^i, g(x))=1$, and $\gcd(s(x), g(x))=1$ due to g(x) providing $\gcd(g(x), s(x))=1$ and $d_{min} \geq (2 + wt(s(x)))$, criterion 2 is also met.

Criterion 3: $[x^i+(x^j)(s(x))]\text{mod}g(x) = 0$

which gives:

$[(x^i)(1+x^k s(x))]\text{mod}g(x) = 0$ for $j = i + k$ and

$[(x^j)(x^k+s(x))]\text{mod}g(x) = 0$ for $i = j + k$

Again, $\gcd(x^i, g(x))=1$. Since the weight of the polynomials $1+x^k s(x)$ and $x^k+s(x)$ are at most one greater than $s(x)$, we are guaranteed that $\gcd(1+x^k s(x), g(x))=1$ and $\gcd(x^k+s(x), g(x))=1$ as long as $d_{min} \geq (2 + \text{wt}(s(x)))$.

QED

Again, the strong case proof has been presented. This theorem can be further generalized for any linear, cyclic t error correcting code as follows:

Theorem 6.4 – It is possible to correct t errors with a generator polynomial $g(x)$ and a scramble $s(x)$ as long as $\gcd(g(x), s(x))=1$ and $d_{min} \geq 1 + (t)(1+\text{wt}(s(x)))$ over the block size n, where $\text{wt}(s(x))$ is the weight of the polynomial $s(x)$.

Proof: In general, a $d_{min} \geq 2t+1$ is required for the correction of t errors. For any arbitrary error patterns $e(x)$ or $e'(x)$ with weight $\leq t$, which are therefore detectable if no scrambler is present, the error detection criteria for this case become:

1. $[(x^i)(e(x))]\text{mod}g(x) \neq [(x^j)(e(x))]\text{mod}g(x)$ for all $i \neq j$,
2. $[(x^i)(e(x))(s(x))]\text{mod}g(x) \neq [(x^j)(e(x))(s(x))]\text{mod}g(x)$ for all $i \neq j$,

3. $[(x^i)(e(x))]\text{mod}g(x) \neq [(x^j)(e'(x))(s(x))]\text{mod}g(x)$ for all i, j, e(x), and e'(x)

Criterion 1: $[(x^i + x^j)(e(x))]\text{mod}g(x) = 0$

$[(x^i)(1 + x^k)(e(x))]\text{mod}g(x) = 0$ for $j = i + k$.

But, since $\gcd(x^i, g(x))=1$, $\gcd(1+x^i, g(x))=1$, and $\gcd(e(x), g(x))=1$ due to g(x) providing $d_{min} \geq 2t+1$, criterion 1 is met.

Criterion 2: $[(s(x))(e(x))(x^i+x^j)]\text{mod}g(x) = 0$

$[(s(x))(e(x))(x^i)(1+x^k)]\text{mod}g(x) = 0$ for $j = i + k$

But, $\gcd(x^i, g(x))=1$ and $\gcd(1+x^i, g(x))=1$ as with criterion 1. Also $\gcd(s(x), g(x))=1$ due to choice of g(x). It's not necessarily true that $\gcd(e(x), g(x))=1$, however as long as the gcd of the other factors with g(x) is 1, it is sufficient that $e(x)\text{mod}g(x) \neq 0$, and so criterion 2 is also met.

Criterion 3: $[(x^i)(e(x)) + (x^j)(e'(x))(s(x))]\text{mod}g(x) = 0$ for all i, j, e(x), and e'(x)

The worst case weight of $(x^i)(e(x)) + (x^j)(e'(x))(s(x))$ is $t+(t)(\text{wt}(s(x)))$, so the criterion is met as long as $d_{min} \geq 1 + (t)(1+\text{wt}(s(x)))$.

QED

The worst case number of errors seen in the block is $(wt_{max}(e(x)))(wt(s(x))) = (t)(wt(s(x)))$. In general, to correct this many errors would need $d_{min} = (2)(t)(wt(s(x)))+1$. But, $(2)(t)(wt(s(x)))+1 > 1 + (t)(1+wt(s(x)))$, since $wt(s(x)) \geq 2$. Hence, as either t or wt(s(x)) grow, the d_{min} savings become increasingly substantial for a code to correct *t'* errors produced by descrambling relative to a code that must correct *t'* errors in general.

For theorems 2-4, the d_{min} requirement for criterion 3 is an upper bound. Since a CRC is typically a shortened block code, it is possible that there may exist unique syndromes that allow error correction with smaller d_{min} values. For the GFP example shown below, only a total of 536+450=980 syndromes are required to be unique out of a total of 65365 possible syndromes. The only way to determine whether the error correction is still possible with the smaller d_{min} is to perform an exhaustive evaluation. One method is to calculate the syndromes for each possible error case that we desire to correct, and compare them for uniqueness. Alternatively, using criterion 3 directly, it is sufficient to determine whether $[x^i+(x^j)(s(x))]\text{mod}g(x) = 0$ for all *i* and *j* in the range of interest. If we assume a serial shift division, both alternatives are $O(n^2)$.

6.2.4 Example – Transparent GFP superblock

The target application that originally motivated the search was the 536-bit superblock structure of Transparent GFP [12]. The results from this example are instructive since x^m+1 is the typical form for a self-synchronous payload scrambler. Since a factor with at least degree 10 was required to guarantee double error detecting capability ($2^{10}-1=1023$), all g(x) candidates containing a factor of degree of least 10, but with no common factors with $x^{43}+1$, were evaluated. Note

that [5] demonstrated that the largest block for which a CRC-16 polynomial can detect 4 errors is 257, and hence four-error detection is not possible with the 536-bit block size. The four error detection capability was used, however, as the deciding factor in choosing g(x) for the Transparent GFP application.

For single error correction with SONET/SDH transport, the CRC must be capable of correcting not only single errors in the block, but also double errors spaced 43 bits apart from the output of the payload descrambler. As shown in theorem 2, the choice of g(x) with d_{min}=4 guaranteed the capability.

One significant result of this study is that the largest block over which d_{min}=4 is possible without an x+1 factor is at least 32768 bits, with several degree 16 primitive polynomials giving this coverage. Another interesting result in this regard is that all g(x) that are the products of any degree 10 primitive polynomial times the product of the two unique degree 3 primitive polynomials maintain d_{min}=4 for blocks of up to 7160 bits. Since the highest degree factor is 10, these particular g(x) polynomials are only theoretically guaranteed to be double error detecting for up to 1023 bits.

In general, the probability of undetectable errors is:

$$Pund(e,n) = \sum_{i=d_{\min}(n)}^{n} A_i^{(n)} e^i (1-e)^{n-i}$$

where e is the bit error rate and $A_i^{(n)}$ is the number of length n codewords with Hamming weight i.

An exhaustive analysis would take a prohibitive amount of computation. Alternatively, the MacWilliams identity could be used to reduce the computations as in [5], [6], [7], [8]. For the specific case of the 536-bit Transparent GFP superblock, and a worst case transmission system bit error rate (BER) of 10^{-3}, which is a typical worst case assumption for SONET, it was adequate to test up to four transmission errors. (i.e., $Pund \approx A_4^{(536)} e^4$.) A total of 1002 g(x) polynomials provided the desired d_{min}=4 and single error correcting capability. The polynomial that had the best four error performance is $x^{16} + x^{15} + x^{12} + x^{10} + x^4 + x^3 + x^2 + x + 1$. This g(x) fails to detect only a total of 44,909 out a total 3,400,578,530 possible four-error cases, which gives an undetectable four error probability of $1.32X10^{-5}$. As a comparison, the undetectable four error probabilities of the standard CRC-CCITT and CRC-ANSI CRC-16s are $2.95X10^{-5}$ and $5.00X10^{-5}$, respectively. Remember, too that the average undetectable error probability for >3 errors is 2^{-16} = $1.53X10^{-5}$ with a CRC-16. Hence, this polynomial was adopted for the Transparent GFP superblock application [9], [10]. This g(x) is also an ideal choice for any relatively short frame that may be carried over GFP, ATM, or POS. (This polynomial retains its d_{min} =4 for blocks of up to 755 bits or 94 bytes.)

6.2.5 CRCs over larger block sizes

An exhaustive analysis was performed to determine the largest block size over which the d_{min}=4 could be maintained without the x+1 factor. It was determined that the largest block size that can be covered with d_{min}=4 by a degree 16 primitive polynomial is 2251 bits (281 bytes). One such g(x) in this category is $x^{16} + x^{13} + x^{11} + x^8 + x^7 + x + 1$. However, primitive degree 16 polynomials did not provide the largest block size. The largest block size that can be covered with d_{min}=4 and no x+1 factor is 19685 bits (2460 bytes). Interestingly, each g(x) that is a product

of $x^4 + x^3 + x^2 + x + 1$ times any degree 5 primitive polynomial times any degree 7 primitive polynomial will cover this 19685 block length with $d_{min}=4$, and no other polynomials reaching this block length. One such polynomial is $x^{16} + x^4 + x^3 + x^2 + 1$.

If a larger block must be covered, then a higher degree CRC polynomial must be used. For example, the Ethernet CRC-32 is a primitive polynomial with a $d_{min}=4$ for up to 1518 bytes [3], and therefore will also not suffer a performance degradation due to the payload descrambler. A topic for further study is to determine whether the result for the longest CRC-16 block size can be generalized to generate a CRC-*r* g(x) that covers the longest block size with a $d_{min}=4$ and no x+1 factor for a given *r*.

6.3 CONCLUSIONS

Self-synchronous scramblers have become an important part of protecting public WANs from attacks by malicious users. While the descrambling process causes multiplication of transmission errors, it has been demonstrated that it is possible to maintain a CRC's undetectable burst error performance with the appropriate choice of g(x). Specifically, the performance of CRC is maintained as long as its generator polynomial has no common factors with the scrambler polynomial. It has also proven possible to maintain single error correction capability in the presence of a self-synchronous scrambler with a CRC. The generalization of these results has shown that with the appropriate choice of g(x), a linear cyclic code such as a Reed-Solomon or BCH code, *t* transmission error correction is possible after descrambling.

Two resulting CRC-16 polynomials have also been presented that are capable of maintaining their capabilities in the presence of typical self-synchronous payload scramblers. The first has been determined to be optimum for use with the Transparent GFP 536-bit superblock, and may also be used for any block up to 754 bits. The other CRC-16 maintains its d_{min}=4 capability for up through 2251 bits. An area for further research is to determine the optimum generator polynomial for other block sizes.

6.4 REFERENCES

[1] ANSI/ATIS T1.105-2001, *Telecommunications – Synchronous Optical Network (SONET) – Basic Description Including Multiplex Structure, Rates and Formats-2001*

[2] ANSI/ATIS T1.105.02-2001, *Telecommunications – Synchronous Optical Network (SONET) – Payload Mappings.*

[3] T1X1.5/2001-094, "Impact of x43 + 1 Scrambler on the Error Detection Capabilities of Ethernet CRC," standards contribution from Norival Figueira, Nortel Networks, March 2001. (www.t1.org/t1x1/_x1-grid.htm)

[4] D. Bersekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ, Prentice-Hall, 1987.

[5] G. Castagnoli, Juerg Ganz, and Patrick Graber, "Optimum Cyclic Redundancy-Check Codes with 16-bit Redundancy," *IEEE Trans. Commun.*, vol. 38, pp111-114, Jan. 1990.

[6] D. Chun and J.K. Wolf, "Special Hardware for Computing the Probability of Undetected Error for Certain Binary CRC Codes and Test Results," *IEEE Trans. Commun.*, vol.42, pp. 2769-2772, Oct. 1994.

[7] T. Fujiwara, T. Kasami, and S. Lin, "Error Detecting Capabilities of the Shortened Hamming Codes Adopted for Error Detection in IEEE Standard 802.3," *IEEE Trans. Commun.*, vol. 37, pp 986-989, Sept. 1989

[8] G. Castagnoli, S. Braeuer, and M. Herrmann, "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits," *IEEE Trans. Commun.*, vol. 41, pp. 883-892, June 1993.

[9] T1X1.5/2001-174, "Optimum CRC-16 Polynomial for the Transparent GFP Superblock," standard contribution from S. Gorshe, PMC-Sierra, Sept. 2001. (www.t1.org/t1x1/ x1-grid.htm)

[10] ITU-T Recommendation Generic Framing Procedure - G.7041/Y.1303 - 2001, (S. Gorshe, technical editor)

[11] S. Gorshe, "CRC-16 Polynomials Optimized for Applications Using Self-Synchronous Scramblers," paper accepted for publication at ICC'2002.

[12] S. Gorshe and T. Wilson, "Transparent Generic Framing Procedure (GFP) – a Protocol for Efficient Transport of Block Coded Data through SONET/SDH Networks," paper accepted for publication in the May 2002 *IEEE Communications Magazine*

7. CONCLUSIONS

The intent of concurrent error detection is to be able to detect errors or faults in a circuit during its normal operation. The technique of check prediction, which uses a check prediction circuit to calculate the error check code for the circuit's outputs based on the error detection codes for the circuit's input, is effective if the check prediction circuit is smaller than the circuit being tested. Check prediction circuits are most likely to have this property when they are used to test circuits that have regular structures. Berger codes have previously been shown to be effective in check prediction for circuits that perform arithmetic or logical operations, including array multipliers. Bose-Lin codes have been shown in this dissertation to provide similar single-fault-secure CED performance to Berger codes for these circuits, while having significantly smaller check prediction circuits.

Another class of circuit that are analyzed in this dissertation was telecommunications and data communications switch fabrics. BIP and CRC codes have previously been used to detect faults in such circuits. The Bose-Lin codes have been shown to give a lower probability of undetected errors in this application than either the BIP or CRC codes with the same number of check bits.

In order to effectively detect errors, the error detecting code must be properly chosen for the type of errors or faults that are expected for that circuit. The above examples are all typically confined to be internal to an integrated circuit where faults are typically unidirectional. Bose-Lin, like Berger codes, are optimized for unidirectional error detection. In the case of a scrambler / descrambler pair with an intervening transmission channel, bi-directional errors can be expected in the

transmission channel. Here, the CRC is known to be a more appropriate error detecting code. The descrambling process can degrade the CRC's effectiveness, however, due to error multiplication. This dissertation has derived the conditions under which this degradation can be avoided or minimized. The analysis was further generalized to include the case of error correcting capability for any linear, cyclic code.

One topic for further research is to apply Bose-Lin codes to other circuit types. Potential candidates here include array dividers and sub-circuits with digital signal processing circuits. The application to switch fabrics can be extended to cover error detection in the control memory that determines the input to output mapping. Another topic for further study is to determine the characteristics of the CRC generator polynomials that are three-error detecting over the largest block size without using the x+1 factor.

BIBLIOGRAPHY

[Agr1] V. D. Agrawal, C. R. Kime, and K. K. Saluia, "A Tutorial on Built-In Self-Test – Part 1," *IEEE Design and Test of Computers*, pp. 73-82, March 1993

[Agr2] V. Agarwal and A. Ivanov, "Computing the Probability of Undetected Error for Shortened Cyclic Codes," *IEEE Trans. on Commun.*, vol. 40, No. 3, pp. 494-499, March 1992

[Alb] S. Al-Bassam, "Another Method for Constructing *t*-EC/UED Codes," *IEEE Trans. Comput.*, vol. 49, pp. 964-970, 2000

[And] D. A. Anderson and G. Metze, "Design of Totally Self-Checking Check Circuits for m-Out-of-n Codes," *IEEE Trans. on Comput.*, vol. c-22, pp. 263-269, March 1973

[Ann] M. Annaratone and R. Steffanelli, "A multiplier with multiple error correction capability," in *Proc. of the 6th Symp. on Comput. Arithmetic*, 1983, pp. 44-51

[ANS] ANSI/ATIS T1.105-2001, *Telecommunications – Synchronous Optical Network (SONET) – Basic Description Including Multiplex Structure, Rates and Formats-2001*

[Ash] M. J. Ashjaee and S. M. Reddy, "On totally self-checking checkers for separable codes," *IEEE Trans. on Comput.*, vol. c. 26, pp. 737-744, Aug. 1977

[Avi] A. Avizienis, "Arithmetic Algorithms for Error-Coded Operands," *IEEE Trans. on Comput.*, vol. c-22, pp. 567-572, June 1973

[Bai] T. Baicheva, S. Dodunekov, and P. Kazakov, "Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy," *IEE Proc. on Commun.* vol. 147, pp. 253-256, Oct. 2000

[Bec1] B. Becker, "An easily testable optimal-time VLSI multiplier," *Acta Informatica*, vol. pp. 363-380, 1987

[Bec2] B. Becker, "Efficient Testing of Optimal Time Adders," *IEEE Trans. on Comput.*, vol. 37, pp. 1113-1120, Sept. 1988

[Berg] J. M. Berger, "A Note on Error Detecting Codes for Asymmetric Channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.

[Berl] E.Berlekamp, *Algebraic Coding Theory*, New York: MacGraw-Hill, 1968

[Bers] D. Bersekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ, Prentice-Hall, 1987.

[Bla1] R. E. Blahut, *Digital Transmission of Information*, Addison-Wesley, Reading, Mass., 1990

[Bla2] M. Blaum, "On systematic burst unidirectional error detecting codes," *IEEE Trans. Comput.*, vol. 37, pp. 453-457, Apr. 1988

[Bor] J. M. Borden, "Optimal asymmetric error detecting codes," *Inform. Contr.*, Apr. 1984

[Bos1] B. Bose and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

[Bos2] B. Bose, "Burst Unidirectional Error-Detecting Codes,: *IEEE Trans. Comput.*, c-35, pp. 350-353, Apr. 1986

[Bos3] B. Bose and T. R. N. Rao, "Theory of unidirectional error correcting/detecting codes," *IEEE Trans. Comput.*, vol. c-31, pp. 521-530, June 1982

[Bos4] B. Bose and D. K. Pradhan, "Optimal unidirectional error detecting codes," *IEEE Trans. Comput.*, vol. c-31, pp. 564-568, June 1982

[Bro] T. J. Brosnan and N. R. Strader II, "Modular error detection for bit-serial multiplication," *IEEE Trans. on Comput.*, vol. c-37, pp. 1043-1052, Sept. 1988

[Car] W. C. Carter and P. R. Schneider, "Design of dynamically checked computers," in *Proc. IFIP'68*, vol. 2, Edinburgh, Scotland, pp. 878-883, Aug. 1968

[Cas1] G. Castagnoli, Juerg Ganz, and Patrick Graber, "Optimum Cyclic Redundancy-Check Codes with 16-bit Redundancy," *IEEE Trans. Commun.*, vol. 38, pp111-114, Jan. 1990

[Cas2] G. Castagnoli, S. Braeuer, and M. Herrmann, "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits," *IEEE Trans. Commun.*, vol. 41, pp. 883-892, June 1993

[Che] S. C. Cheng and J. K. Wolf, "A simple derivation of the MacWilliams identity for linear codes," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 476-477, July 1980

[Chu] D. Chun and J.K. Wolf, "Special Hardware for Computing the Probability of Undetected Error for Certain Binary CRC Codes and Test Results," *IEEE Trans. Commun.*, vol.42, pp. 2769-2772, Oct. 1994

[Das] D. Das and N. Touba, "Synthesis of Circuits with Low-Cost Concurrent Error Detection Based on Bose-Lin Codes," in *Proc. of 16th IEEE VLSI Test Symp.*, pp. 309-315, 1998

[De] K. De, S. Natarajan, D. Nair, and P. Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Trans. on VLSI Systems*, vol. 2, pp. 186-195, 1994

[Deb] W. H. Debany, A. R. Macera, D. E. Daskiewich, M. J. Gorniak, K. A. Kwiat, and H. B. Dussault, "Effective concurrent test for a parallel-input multiplier using modulo 3," *IEEE VLSI Test Symposium*, pp. 280-284, 1992

[Don] H. Dong, "Modified Berger codes for detection of unidirectional errors," *IEEE Trans. Comput.*, vol. c-33, pp. 572-575, June 1984

[Fer] J. Ferguson and J. P. Shen, "The design of two easily-testable VLSI array multipliers," in *Proc. 6th Symp. Comput. Arithmetic*, pp. 20-22, 1983

[Fre] C. V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," *Inform. Contr.*, vol. 5, pp. 64-71, Mar. 1962

[Fuh] E. Fuhwara and K. Matsuoka, "A Self-Checking Generalized Prediction Checker and Its Use for Built-In Testing," *IEEE Trans. Comput.*, vol. c-36, pp. 86-93

[Fuj1] T. Fujiwara, T. Kasami, and S. Lin, "Error Detecting Capabilities of the Shortened Hamming Codes Adopted for Error Detection in IEEE Standard 802.3," *IEEE Trans. Commun.*, vol. 37, pp 986-989, Sept. 1989

[Fuj2] E. Fujiwara and K. Haruta, "Fault-Tolerant Arithmetic Logic Unit Using Parity-Based Codes," *Trans. Inst. Electron. Commun. Eng. Japan*, pp. 653-660, Oct. 1981

[Fuj3] T. Fujiwara, T. Kasami, A. Kitai, and S. Lin, "On the undetected error probability of shortened Hamming codes," *IEEE Trans. Commun.*, vol. COM-33, pp. 570-574, June 1985

[Gor1] S. Gorshe and B. Bose, "A Self-Checking ALU Design with Efficient Codes," in *Proc. 14th IEEE VLSI Test Symp.*, pp. 157-161, 1996

[Gor2] S. Gorshe, "CRC-16 Polynomials Optimized for Applications Using Self-Synchronous Scramblers," paper accepted for publication at ICC'2002.

[Gor3] S. Gorshe and T. Wilson, "Transparent Generic Framing Procedure (GFP) – a Protocol for Efficient Transport of Block Coded Data through SONET/SDH Networks," paper accepted for publication in the May 2002 *IEEE Communications Magazine*

[Hay] J. P. Hayes, "Fault Modeling," *IEEE Design & Test*, pp. 88-95, April 1985

[Hon] S. Je Hong, "An easily testable parallel multiplier," in *Proc. FTCS 18*, Tokyo, pp. 214-219, 1988

[Hsu] Y. M. Hsu and E. E. Swatzlander, Jr., "Time redundant error correcting adders and multipliers," *IEEE Int. Work. on Defect and Fault Tolerance in VLSI Sys.*, pp. 247-256, 1992

[Jha1] N. Jha, "Totally Self-Checking Checker Design for Bose-Lin, Bose, and Blaum Codes," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 136-143, Jan. 1991

[Jha2] N. Jha and S.-J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits," *IEEE Trans. on Computer-Aided Design*, vol. 12, pp. 878-887, June 1993

[Jha3] N. K. Jha and M. B. Vora, "A systematic code for detecting t-unidirectional errors," in *Proc. Int. Symp. Fault-Tolerant Comput.*, Pittsburgh, PA, pp. 96-101, June 1987

[Kas1] T. Kasami, T. Klove, and S. Lin, "Linear block codes for error detection," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 131-136, Jan. 1983

[Kas2] T. Kasami, "Optimum shortened cyclic codes for burst-error correction," *IEEE Trans. Inform. Theory*, vol. IT-9, No. 2, pp. 105-109, 1963

[Kas3] T. Kasami, T. Fujiwara, and S. Lin, "An approximation of the weight distribution of binary linear codes," in *Proc. 6th Conf. Inform. Theory and Its Appl.*, Matsuyama, Japan, Nov. 1983

[Kav] X. Kavousianos and D. Nikolos, "Novel TSC Checkers for Bose-Lin and Bose Codes," *3rd IEEE Int. On-Line Testing Workshop*, pp. 172-176, July 1998

[Kho] B. Khodadad-Mostashiry, "Parity Prediction in combinational circuits," in *Proc. FTCS-9*, pp. 185-188, 1979

[Kor] I. Koren, *Computer Arithmetic Algorithms*, Englewood Cliffs, NJ: Printice-Hall, 1993

[Leu1] S. K. Leung-Yan-Cheong, E. R. Barnes, and D. U. Freidman, "On some properties of the undetected error probability of linear codes," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 110-112, Jan. 1979

[Leu2] C. Leung and M. E. Hellman, "Concerning a bound on undetected error probability," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 235-237, Mar. 1976

[Lo1] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. Computer-Aided Design* vol. 11, pp. 525–540, April 1992.

[Lo2] J.-C. Lo, S. Thanawastein, and T.R.N. Rao, "Concurrent Error Detection in Arithmetic and Logical Operations Using Berger Codes," in *Proc. 9th Symp. Computer Arithmetic*, Sept. 1989, pp. 233-240

[Lo3] J.-C. Lo, S. Thanawastein, and T.R.N. Rao, "Berger Check Prediction for Array Multipliers and Array Dividers," *IEEE Trans. Comput.*, vol. 42, pp. 892-896, July 1993

[Lo4] J.-C. Lo and S. Thanawastien, "The Design of Fast Totally Self-Checking Berger Code Checkers Based on Berger Code Partitioning," in *Proc. FTCS-19*, pp. 226-231, June 1988

[Lo5] J.-C. Lo and E. Fujiwara, "Probability to Achieve TSC Goal," *IEEE Trans. on Comput.*, vol. 45, pp. 450-460, 1996

[Lo6] J.-C. Lo, "Self-Checking VLSI Reduced Instruction Set Computers," Ph.D. Dissertation, Univ. of Southwestern Lousiana, 1989

[Lo7] J.-C. Lo, "A Novel Area-Time Efficient Static CMOS Totally Self-Checking Comparator," *IEEE J. of Solid State Circuits*, vol. 28, pp. 165-168, Feb. 1993

[Mac] F. J. MacWilliams, "A theorem on the distribution of weights in a systematic code," *Bell Sys. Tech. J.*, vol. 42, pp. 79-98, 1963

[Mar] M. A. Marouf and A. D. Friedman, "Design of Self-Checking Checkers for Berger Codes," *Proc. FTCS-8*, pp. 179-184, June 1978

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

[Mer] P. Merkey and E. C. Posner, "Optimum cyclic redundancy codes for noisy channels," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 865-867, Nov. 1984

[Met1] C. Metra and J.-C. Lo, "Compact and High Speed Berger Code Checker." in *Proc. of 2nd IEEE On-Line Test Workshop*, pp. 144-149, 1996

[Met2] C. Metra, M. Favalli, and B. Ricco, "Novel Berger Code Checker," in *Proc. of IEEE Int. Work. on Defect and Fault Tolerance in VLSI Sys.*, pp. 287-295, 1995

[Met3] C. Metra, M. Favalli, P. Olivo, and B. Ricco, "Design of Bridging and Transistor Stuck-on Faults," *J. of Electronic Testing: Theory and Application*, vol. 6, pp. 7-22, Feb. 1995

[Mon] R. K. Montoye and J. A. Abraham, "Built-in tests for arbitrarily structured VLSI carry look-ahead adders," in Proc. IFIP, pp. 361-371, 1983

[Nic1] M. Nicolaidis and B. Courtois, "Strongly Code Disjoint Checkers," *IEEE Trans. on Comput.*, vol. 37, pp. 751-756, June 1988

[Nic2] M. Nicolaidis, "Efficient implementation of self-checking adders and ALUs," *23rd Int. Symp. on Fault Tolerant Comp. Sys.*, pp. 586-595, 1993

[Pat] J. H. Patel and L. Y. Fong, "Concurrent error detection in multiply and divide arrays," *IEEE Trans. on Comput.* vol., c-37, pp. 417-422, April 1983

[Pet1] W. W. Peterson and E. J. Weldon, *Error Correcting Codes*, Cambridge, MA: M.I.T. Press, 1972

[Pet2] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, pp. 228-235, Jan. 1961

[Pie1] D. Pierce and P. K. Lala, "Efficient Self-Checking Checkers for Berger Codes," in *Proc. of 1st IEEE Int. On-Line Testing Work.*, pp. 238-242, 1995

[Pie2] S. J. Piestrak, "Design of encoders and self-testing checkers for some systematic unidirectional error detecting codes," Proc. of 1997 IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Sys., pp. 119-127, Oct. 1997

[Pra1] D. K. Pradhan and J. J. Stiffler, "Error correcting codes and self-checking circuits in fault tolerant computers," *Computer*, pp. 27-37, Mar. 1980

[Pra2] D. K. Pradhan and S. M. Reddy, "Error Control Techniques for Logic Processors," *IEEE Trans. on Comput.*, vol. c-21, No. 12, pp. 1331-1336, Dec. 1972

[Psa] M. Psarakis, D. Gizopoulos, A. Paschalis, Y. Zorian, "Robustly Testable Array Mulitpliers under Realistic Sequential Cell Fault Model," in *Proc. of 16th IEEE VLSI Test Symp.*, pp. 152-157, 1998

[Rao1] T. R. N Rao and E. Fujiwara, *Error-Control coding for computer systems*, Prentice-Hall, Englewood Cliffs, NJ, 1989

[Rao2] T. R. N. Rao and P. Monteiro, "A Residue Checker for Arithmetic and Logic Operations," Proc. *FTSC-2*, pp. 8-13, June 1972.

[Rao3] T. R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press, New York and London, 1974

[Rao4] T. R. N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall, New Jersey, 1989

[Rao5] T. R. N. Rao, G. Feng, M. S. Kolluru, and J.-C. Lo, "Novel Totally-Self-Checking Berger Code Checker Designs Based on Generalized Berger Code Partitioning," *IEEE Trans. on Comput.*, vol. 42, pp. 1020-1024, Aug. 1993

[Sel] F. F. Sellers, M. Y. Hsaio, and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill, 1968

[She] J. P. Shen and F. J. Ferguson, "The Design of Easily Testable VLSI Array Multipliers," *IEEE Trans. on Comput.*, vol. c-33, No. 6, pp. 554-560, June 1984

[Spar1] U. Sparmann and S. M. Reddy, "On the Effectiveness of Residue Code Checking for Parallel Two's Complement Mutlipliers," *IEEE Trans. VLSI Systems*, vol. 4, pp. 227-239, June 1996

[Spar2] U. Sparmann and S. M. Reddy, "On the effectiveness of residue code checking for parallel two's complement multipliers," TR-8-21-93, Elec. and Comp. Eng. Dept., Univ. of Iowa, Iowa City, Iowa 52242, 1993

[Sto] J. Stone, M. Greenwald, C. Partridge, and J. Hughes, "Performance of Checksums and CRC's over Real Data," *IEEE Trans. on Networking*, vol. 6, No. 5, pp. 529-543, Oct. 1998

[Tak] N. Takagi and S. Yajima, "On-line error-detectable high-speed multiplier using redundant binary representation and three-rail logic," *IEEE Trans. on Comput.*, vol. c-36, pp. 1310-1317, Nov. 1987

[Tys] J. Tyszer, "Test Generation for Pattern-Sensitive Faults in Integrated Switches," *IEEE Trans. Commun.*, vol. 39, pp. 1546-1548, Nov. 1991

[Tze] K. K. Tzeng and C. R. P. Hartmann, "On the minimum distance of certain reversible cyclic codes," *IEEE Trans. Inform. Theory,* vol. IT-16, pp. 644-646, Sept. 1970

[Wak] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, North-Holland, 1978

[Wic] S. Wicker, *Error Control Systems for Digital Communications and Storage*, Prentice-Hall, Upper Saddle River, NJ, 1995

[Wit] K. A. Witzke and C. Leung, "A comparison of some error detecting CRC code standards," *IEEE Trans. Commun.*, vol. COM-33, pp. 996-998, Sept. 1985

[Wol1] J. K. Wolf and D. Chen, "The Single Burst Error Detection Performance of Binary Cyclic Codes," *IEEE Trans. on Commun.*, vol. 42, No. 1, pp. 11-13, Jan. 1994

[Wol2] J. K. Wolf, A. M. Michelson, and A. H. Levesque, "On the probability of undetected error for linear block codes," *IEEE Trans. Commun.*, vol. COM-30, pp. 317-324, Feb. 1982

[Wol3] J. K. Wolf and R. D. Blakenay II, "An exact evaluation of the probability of undetected error for certain shortened binary CRC codes," in *Proc. MILCOM '88*, pp. 15.2.1-15.2.6

[Won] W. S. Wong, J. D. Moores, J. Korn, and H. A. Haus, "Photon statistics of NRZ signals in a high-bit-rate optically pre-amplified direct detection receiver," in Proc. of Optical Fiber Conference, pp. 265-267, 1999